aws

데이터베이스 개발자 안내서

Amazon Redshift

Amazon Redshift: 데이터베이스 개발자 안내서

Table of Contents

Amazon Redshift 소개

Amazon Redshift 데이터베이스 개발자 안내서입니다. 이 안내서에서는 Amazon Redshift를 사용하여 데이터 웨어하우스를 생성하고 관리하는 방법을 이해하는 데 중점을 둡니다. 데이터베이스를 사용하는 설계자, 소프트웨어 개발자 또는 관리자라면 데이터 웨어하우스의 설계부터 빌드, 쿼리 및 유지 관리에 이르기까지 이 안내서를 통해 필요한 정보를 얻을 수 있습니다.

Amazon Redshift는 클라우드에서 완전히 관리되는 페타바이트급 데이터 웨어하우스 서비스입니다. Amazon Redshift Serverless를 사용하면 평소와 같이 프로비저닝된 데이터 웨어하우스를 구성하지 않아도 데이터를 액세스하고 분석할 수 있습니다. 리소스가 자동으로 프로비저닝하고 데이터 웨어하우스 용량이 지능적으로 크기 조정되어 가장 까다롭고 예측할 수 없는 워크로드에도 빠른 성능을 제공합니다. 데이터 웨어하우스가 유휴 상태일 때는 요금이 발생하지 않으므로 사용량에 대한 요금만 지불합니다. 데이터 세트의 크기에 관계없이 Amazon Redshift 쿼리 에디터 v2 또는 자주 사용하는 비즈니스 인텔리전스(BI) 도구에서 바로 데이터를 로드하고 쿼리를 시작할 수 있습니다. 사용하기 쉽고 관리가 필요 없는 환경에서 최고의 가격 대비 성능과 친숙한 SQL 기능을 활용하세요.

주제

Amazon Redshift를 사용하기 위한 사전 조건

이 주제에서는 Amazon Redshift를 사용하는 데 필요한 사전 조건을 설명합니다.

이 안내서를 사용하기 전에 다음 작업을 완료하는 방법을 설명하는 Amazon Redshift Serverless를 읽어야 합니다.

- Amazon Redshift Serverless를 사용하여 데이터 웨어하우스 만들기
- Amazon Redshift 쿼리 에디터 v2를 사용하여 샘플 데이터 로드
- Amazon S3에서 데이터 로드

그 밖에 SQL 클라이언트의 사용 방법에 대해서 알아야 하고, SQL 언어에 대한 기본적인 이해도 필요합니다.

Amazon Redshift 아키텍처

이 주제에서는 Amazon Redshift 구성 요소에 대해 알아볼 수 있습니다.

Amazon Redshift 데이터웨어 하우스는 엔터프라이즈 급 관계형 데이터베이스 쿼리 및 관리 시스템입니다.

Amazon Redshift는 비즈니스 인텔리전스(BI), 보고, 데이터 및 분석 도구를 비롯한 다양한 유형의 애플리케이션과의 클라이언트 연결을 지원합니다.

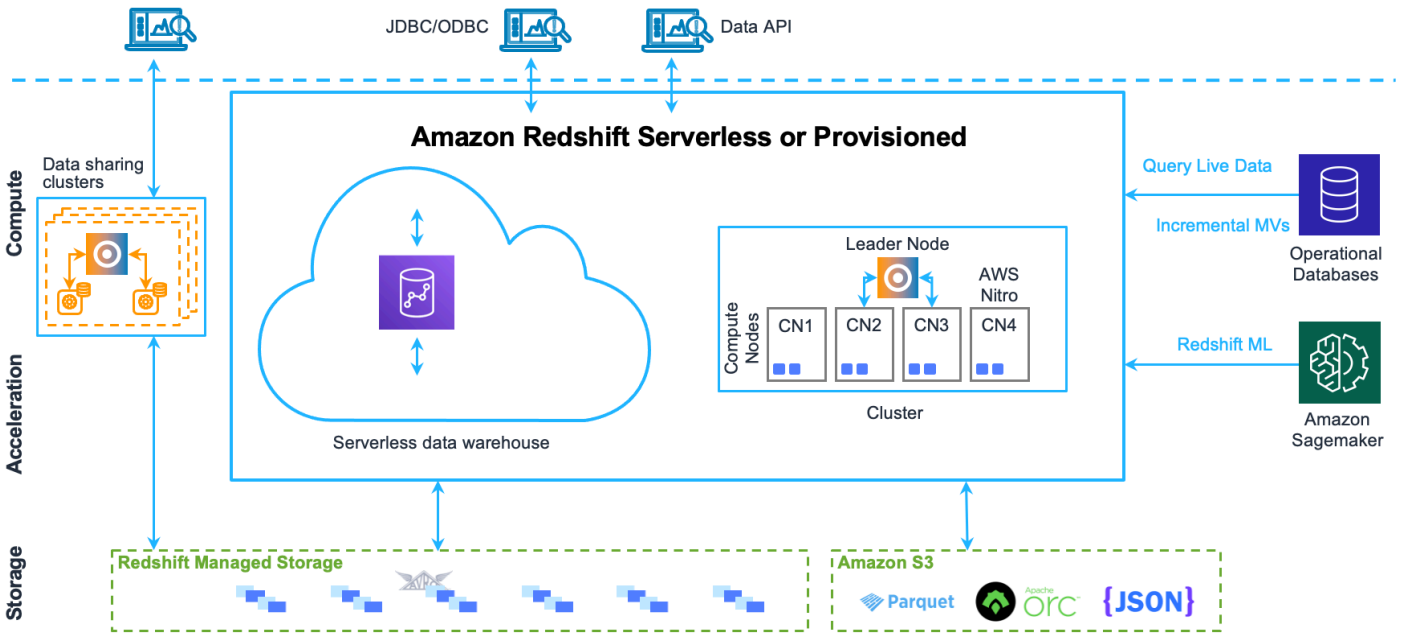
분석 쿼리를 실행할 때는 다단계 작업을 통해 대용량의 데이터를 가져와서 비교하고 평가하면서 최종 결과를 산출합니다.

Amazon Redshift는 대규모 병렬 처리, 컬럼 데이터 저장 및 매우 효율적인 대상 데이터 압축 인코딩 체계의 조합을 통해 효율적인 저장 및 최적의 쿼리 성능을 달성합니다. 이 섹션에서는 Amazon Redshift 시스템 아키텍처에 대해 소개합니다.

주제

데이터 웨어하우스 시스템 아키텍처

이 섹션에서는 다음 그림과 같이 Amazon Redshift 데이터 웨어하우스 아키텍처를 구성하는 요소를 설명합니다.



클라이언트 애플리케이션

Amazon Redshift는 다양한 데이터 로드 및 추출, 변환, 로드(ETL) 도구, 비즈니스 인텔리전스(BI) 보고, 데이터 마이닝 및 분석 도구와 통합됩니다. Amazon Redshift는 개방형 표준 PostgreSQL을 기반으로 하므로 대부분의 기존 SQL 클라이언트 애플리케이션은 최소한의 변경만으로 작동합니다. Amazon Redshift SQL과 PostgreSQL의 중요한 차이점에 대한 정보는 Amazon Redshift 및 PostgreSQL 섹션을 참조하세요.

클러스터

Amazon Redshift 데이터 웨어하우스의 핵심 인프라 구성 요소는 클러스터입니다.

클러스터는 하나 이상의 컴퓨팅 노드로 구성됩니다. 클러스터에 두 개 이상의 컴퓨팅 노드가 제공된 경우 추가 리더 노드가 컴퓨팅 노드를 조정하고 외부 통신을 처리합니다. 클라이언트 애플리케이션은 리더 노드와만 직접 상호작용합니다. 컴퓨팅 노드는 외부 애플리케이션에서 인식됩니다.

리더 노드

리더 노드는 클라이언트 프로그램과 일어나는 통신을 비롯해 컴퓨팅 노드와 일어나는 모든 통신을 관리합니다. 또한 구문을 분석하여 데이터베이스 작업, 특히 복합 쿼리의 결과를 얻는 데 필요한 단계를 연이어 실행하기 위한 실행 계획을 작성합니다. 리더 노드는 이렇게 작성된 실행 계획에 따라 코드를 컴파일하여 컴퓨팅 노드로 배포한 후 데이터 구간을 각 컴퓨팅 노드로 할당합니다.

리더 노드는 쿼리가 컴퓨팅 노드에 저장된 테이블을 참조할 때만 SQL 문을 컴퓨팅 노드에 배포합니다. 다른 모든 쿼리는 리더 노드에서만 실행됩니다. Amazon Redshift는 리더 노드에서만 특정 SQL 기능을

구현하도록 설계되었습니다. 이러한 함수를 사용하는 쿼리는 컴퓨팅 노드에 저장되어 있는 테이블을 참조할 경우 오류를 반환합니다. 자세한 내용은 리더 노드에서 지원되는 SQL 함수 단원을 참조하십시오.

노드 계산

리더 노드는 실행 계획을 구성하는 개별 요소마다 코드를 컴파일하여 각 컴퓨팅 노드에 할당합니다. 그러면 컴퓨팅 노드가 컴파일 코드를 실행한 후 최종 집계를 위해 중간 결과를 리더 노드에 다시 보냅니다.

각 컴퓨팅 노드마다 전용 CPU와 메모리가 따로 있으며, 이는 노드 유형에 따라 결정됩니다. 그래도 워크로드가 증가하더라도 노드 수를 늘리거나, 노드 유형을 업그레이드하거나, 혹은 두 가지 방법 모두 사용하여 클러스터의 컴퓨팅 용량을 늘릴 수 있습니다.

Amazon Redshift에서는 컴퓨팅 요구에 맞는 여러 노드 유형을 제공합니다. 각 노드 유형의 자세한 내용은 Amazon Redshift 관리 가이드의 Amazon Redshift 클러스 섹션을 참조하세요.

Redshift 관리형 스토리지

데이터 웨어하우스 데이터는 별도의 스토리지 계층 Redshift 관리형 스토리지(RMS)에 저장됩니다. RMS는 Amazon S3 스토리지를 사용하여 스토리지를 페타바이트로 확장할 수 있는 기능을 제공합니다. RMS를 사용하면 컴퓨팅 및 스토리지에 대해 독립적으로 확장하고 비용을 지불할 수 있으므로 컴퓨팅 요구 사항에 따라서만 클러스터 크기를 조정할 수 있습니다. 자동으로 고성능 SSD 기반 로컬 스토리지를 Tier-1 캐시로 사용합니다. 또한 데이터 블록 온도, 데이터 블록 기간 및 워크로드 패턴과 같은 최적화를 활용하여 고성능을 제공하는 동시에 필요한 경우 조치 없이 스토리지를 Amazon S3로 자동 확장합니다.

노드 조각

컴퓨팅 노드는 다수의 조각으로 분할됩니다. 분할된 조각은 다시 각각 노드의 메모리 및 디스크 공간으로 할당되고, 여기에서 노드에 할당되는 워크로드를 처리합니다. 리더 노드는 조각에 대한 데이터 분산을 관리하면서 쿼리 워크로드 또는 기타 데이터베이스 작업 워크로드를 각 조각으로 할당합니다. 그러면 각 조각이 병렬 방식으로 실행되어 작업을 완료합니다.

노드당 조각 수는 클러스터의 노드 크기에 따라 달라집니다. 각 노드 크기의 슬라이스 수에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 클러스터 및 노드 정보 섹션을 참조하세요.

테이블을 생성할 때는 열 1개를 분산 키로 지정할 수 있습니다(선택 사항). 이후 테이블에 데이터를 로드하면 테이블에 정의되어 있는 분산 키에 따라 행이 노드 조각으로 분산됩니다. 좋은 배포 키를 선택

하면 Amazon Redshift가 병렬 처리를 사용하여 데이터를 로드하고 쿼리를 효율적으로 실행할 수 있습니다. 분산 키 선택에 대한 자세한 내용은 최상의 배포 스타일 선택 단원을 참조하세요.

내부 네트워크

Amazon Redshift는 고 대역폭 연결, 근접성 및 맞춤형 통신 프로토콜을 활용하여 리더 노드와 컴퓨팅 노드 사이에 개인 고속 네트워크 통신을 제공합니다. 컴퓨팅 노드는 클라이언트 애플리케이션이 절대로 직접 액세스하지 못하도록 격리된 네트워크에서 실행됩니다.

데이터베이스

클러스터에는 하나 이상의 데이터베이스가 포함되어 있습니다. 사용자 데이터는 컴퓨팅 노드에 저장됩니다. SQL 클라이언트는 리더 노드와 통신하며, 리더 노드는 컴퓨팅 노드로 쿼리 실행을 조정합니다.

Amazon Redshift는 관계형 데이터베이스 관리 시스템(RDBMS)이므로 다른 RDBMS 애플리케이션과 호환됩니다. 데이터 삽입 및 삭제와 같은 온라인 트랜잭션 처리(OLTP) 기능을 포함하여 일반적인 RDBMS와 동일한 기능을 제공하지만, Amazon Redshift는 매우 큰 데이터 집합의 고성능 분석 및 보고를 위해 최적화되어 있습니다.

Amazon Redshift는 PostgreSQL을 기반으로 합니다. Amazon Redshift와 PostgreSQL은 데이터웨어 하우스 애플리케이션을 설계하고 개발할 때 고려해야 할 몇 가지 매우 중요한 차이점이 있습니다. Amazon Redshift SQL이 PostgreSQL과 어떻게 다른지 자세히 알아보려면 Amazon Redshift 및 PostgreSQL 섹션을 참조하세요.

Amazon Redshift 성능

이 주제에서는 성능을 높이는 Amazon Redshift 구성 요소에 대해 설명합니다. 이러한 구성 요소를 이해하면 Amazon Redshift의 성능을 조정하고 성능 저하 문제를 해결하는 데 도움이 됩니다.

Amazon Redshift는 이러한 성능 기능을 사용하여 매우 빠른 쿼리 실행을 달성합니다.

주제

- 대용량 병렬 처리
- 열 기반 데이터 스토리지
- 데이터 압축
- 쿼리 최적화 프로그램
- 결과 캐싱

- 컴파일 코드

대용량 병렬 처리

대용량 병렬 처리(MPP)는 아무리 복잡한 쿼리라고 해도 빠른 속도로 실행하여 대용량 데이터를 처리할 수 있습니다. 다수의 컴퓨팅 노드가 각 노드의 코어마다 전체 데이터를 분할하여 동일하게 컴파일된 쿼리 세그먼트를 실행하면서 최종 결과 집계에 이를 때까지 모든 쿼리를 처리합니다.

Amazon Redshift는 데이터를 병렬로 처리 할 수 있도록 테이블의 행을 계산 노드에 배포합니다. 각 테이블마다 적절한 분산 키를 선택하면 데이터 분산을 최적화함으로써 워크로드의 밸런스를 유지할 뿐만 아니라 노드 간 데이터 이동을 최소화할 수도 있습니다. 자세한 내용은 최상의 배포 스타일 선택 단원을 참조하십시오.

플랫 파일에서 데이터를 로드할 때는 다수의 파일에서 데이터를 읽어오는 동시에 워크로드를 다수의 노드로 분산시켜서 병렬 방식으로 데이터를 처리합니다. 데이터를 테이블에 로드하는 방법에 대한 자세한 내용은 데이터 로드에 대한 Amazon Redshift 모범 사례 단원을 참조하세요.

열 기반 데이터 스토리지

데이터베이스 테이블을 위한 열 기반 스토리지는 전체 디스크 I/O 요건을 크게 줄여주며 분석 쿼리 성능을 최적화하는 데 중요한 요인입니다. 자세한 내용은 열 기반 스토리지 단원을 참조하십시오.

열이 적절하게 정렬되면 쿼리 프로세서가 대용량의 데이터 블록 하위 집합을 빠르게 필터링할 수 있습니다. 자세한 내용은 최상의 정렬 키 선택 단원을 참조하십시오.

데이터 압축

데이터 압축은 스토리지 요건을 줄여 디스크 I/O를 감소시킴으로써 쿼리 성능이 향상되는 효과가 있습니다. 쿼리를 실행하면 압축된 데이터를 메모리로 읽어온 후 쿼리 실행 도중 압축이 해제됩니다. 적은 데이터를 메모리에 로드하면 Amazon Redshift는 더 많은 메모리를 할당하여 데이터를 분석할 수 있습니다. 원주형 스토리지는 유사한 데이터를 순차적으로 저장하기 때문에 Amazon Redshift는 원주형 데이터 형식에 특별히 연결된 적응형 압축 인코딩을 적용할 수 있습니다. 테이블 열에 데이터 압축을 가능하게 하는 가장 좋은 방법은 데이터로 테이블을 로드할 때 Amazon Redshift가 최적의 압축 인코딩을 적용하는 것입니다. 자동 데이터 압축의 사용에 대한 자세한 내용은 자동 압축을 사용하여 테이블 로드 단원을 참조하세요.

쿼리 최적화 프로그램

Amazon Redshift 쿼리 실행 엔진은 MPP를 인식하고 열 기반 데이터 리포지토리를 활용하는 쿼리 옵티마이저를 통합합니다. Amazon Redshift 쿼리 옵티마이저는 종종 다중 테이블 조인, 하위 쿼리 및 집

계를 포함하는 복잡한 분석 쿼리를 처리하기위한 중요한 향상 및 확장을 구현합니다. 쿼리 최적화에 대한 자세한 내용은 쿼리 성능 튜닝 단원을 참조하세요.

결과 캐싱

쿼리 런타임을 줄이고 시스템 성능을 향상시키기 위해 Amazon Redshift는 특정 형식의 쿼리 결과를 리더 노드의 메모리에 캐시합니다. 사용자가 쿼리를 제출하면 Amazon Redshift는 결과 캐시에서 쿼리 결과의 유효한 캐시 된 복사본을 확인합니다. 결과 캐시에서 일치 항목이 발견되면 Amazon Redshift는 캐시된 결과를 사용하고 쿼리를 실행하지 않습니다. 결과 캐싱은 사용자가 볼 수 있습니다.

결과 캐싱은 기본적으로 설정되어 있습니다. 현재 세션에 대해 결과 캐싱을 해제하려면 enable_result_cache_for_session 파라미터를 `off`로 설정합니다.

Amazon Redshift는 다음 조건이 모두 충족될 때 캐시된 결과를 새 쿼리에 사용합니다.

- 쿼리를 제출하는 사용자가 쿼리에 사용되는 객체에 대한 액세스 권한을 보유합니다.
- 쿼리 내 테이블 또는 보기가 수정되지 않았습니다.
- 쿼리는 실행 시마다 평가되어야 하는 함수(예: GETDATE)를 사용하지 않습니다.
- 쿼리는 Amazon Redshift Spectrum 외부 테이블을 참조하지 않습니다.
- 쿼리 결과에 영향을 미칠 수 있는 구성 파라미터가 변경되지 않았습니다.
- 구문상 쿼리가 캐시된 쿼리와 일치합니다.

캐시 효율성과 리소스의 효율적인 사용을 극대화하기 위해 Amazon Redshift는 일부 대규모 쿼리 결과 집합을 캐시하지 않습니다. Amazon Redshift는 여러 요인에 따라 쿼리 결과를 캐시할지 여부를 결정합니다. 이러한 요소에는 캐시의 항목 수와 Amazon Redshift 클러스터의 인스턴스 유형이 포함됩니다.

특정 쿼리에 결과 캐시가 사용되었는지 여부를 확인하려면 SVL_QLOG 시스템 보기를 쿼리합니다. 쿼리에 결과 캐시가 사용된 경우 소스 쿼리 열이 소스 쿼리의 쿼리 ID를 반환합니다. 결과 캐싱이 사용되지 않은 경우 source_query 열 값은 NULL입니다.

다음 예에서는 userid 104 및 userid 102가 제출한 쿼리에 userid 100이 실행한 쿼리의 결과 캐시가 사용됩니다.

```
select userid, query, elapsed, source_query from svl_qlog
where userid > 1
order by query desc;
```

```
userid | query  | elapsed  | source_query
-------+--------+----------+-------------
   104 | 629035 |       27 |       628919
   104 | 629034 |       60 |       628900
   104 | 629033 |       23 |       628891
   102 | 629017 |  1229393 |
   102 | 628942 |       28 |       628919
   102 | 628941 |       57 |       628900
   102 | 628940 |       26 |       628891
   100 | 628919 | 84295686 |
   100 | 628900 | 87015637 |
   100 | 628891 | 58808694 |
```

컴파일 코드

리더 노드는 완전히 최적화된 컴파일 코드를 모든 클러스터 노드로 분산시킵니다. 쿼리를 컴파일하면 인터프리터와 관련된 오버헤드가 감소함으로써 특히 복잡한 쿼리의 경우 런타임 속도가 빨라집니다. 컴파일된 코드는 동일한 클러스터의 세션 간에 캐시되고 공유됩니다. 그러면 동일한 쿼리의 향후 실행은 종종 파라미터가 다른 경우에도 더 빨라질 것입니다.

쿼리 실행 엔진은 JDBC 연결 프로토콜일 때와 ODBC 연결 프로토콜일 때 컴파일하는 코드가 다르기 때문에, 클라이언트 2개가 서로 다른 프로토콜을 사용하는 경우에는 각각 코드를 컴파일하는 최초 비용이 발생합니다. 하지만 동일한 프로토콜을 사용하는 클라이언트는 캐시 코드 공유라는 이점을 누릴 수 있습니다.

열 기반 스토리지

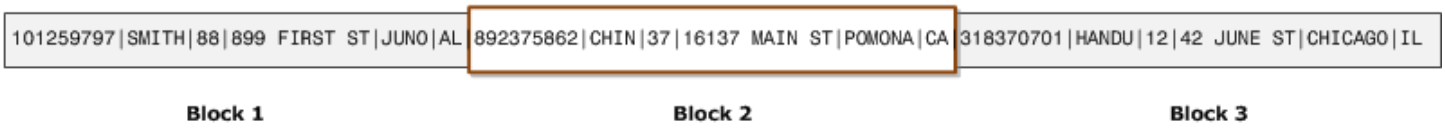
이 섹션에서는 Amazon Redshift가 테이블 형식 데이터를 효율적으로 저장하는 데 사용하는 방법인 열 기반 스토리지에 대해 설명합니다.

데이터베이스 테이블에 대한 열 기반 스토리지는 전체 디스크 I/O 요구 사항을 크게 줄이기 때문에 분석 쿼리 성능을 최적화하는 데 중요한 요소입니다. 그러면 디스크에서 로드해야 하는 데이터의 양이 줄어듭니다.

아래 이어지는 그림들은 열 기반 데이터 스토리지가 어떻게 효율성을 구현하는지, 그리고 이를 통해 데이터를 메모리에 가져올 때 어떻게 효율성이 실현되는지 보여줍니다.

첫 번째 그림은 데이터베이스 테이블의 레코드가 디스크 블록에 행으로 저장되는 일반적인 방식을 나타낸 것입니다.

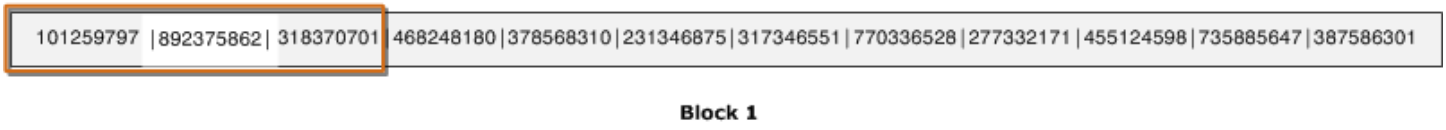
SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL



일반적인 관계형 데이터베이스 테이블에서는 각 행마다 단일 레코드의 필드 값이 포함됩니다. 행 방향 데이터베이스 스토리지에서는 데이터 블록이 연속되는 열마다 순차적으로 값을 저장하여 전체 행을 구성합니다. 이때 블록 크기가 레코드 크기보다 작으면 전체 레코드를 저장하는 스토리지에 블록이 1개 이상 필요할 수도 있습니다. 블록 크기가 레코드 크기보다 크면 전체 레코드를 저장하는 스토리지에 필요한 블록이 1개 미만일 수 있으므로, 디스크 공간을 비효율적으로 사용하는 결과가 발생합니다. 온라인 트랜잭션 처리(OLTP) 애플리케이션일 때는 대부분 트랜잭션에서 전체 레코드의 모드 값을 읽고 쓰는 작업이 자주 일어납니다(일반적으로 한 번에 1개 또는 소수의 레코드). 결과적으로 OLTP 데이터베이스에서는 행 방향 스토리지가 가장 효과적입니다.

다음 그림은 열 기반 스토리지일 때 각 열 값이 디스크 블록에 순차적으로 저장되는 방식을 나타낸 것입니다.

SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL



열 기반 스토리지를 사용할 때는 각 데이터 블록이 여러 행에 대한 단일 열 값을 저장합니다. 레코드가 시스템에 들어가면 Amazon Redshift는 데이터를 각 열의 컬럼 스토리지로 투명하게 변환합니다.

위의 간단한 예에서는 열 기반 스토리지를 사용하기 때문에 데이터 블록에 저장되는 열 필드 값이 행 방향 스토리지에 저장되는 레코드 수보다 3배 더 많습니다. 다시 말해 레코드 수가 동일하다고 가정할 때 동일한 수의 열 필드 값을 읽어오는 데 필요한 I/O 작업이 행 방향 스토리지에 비해 1/3로 줄어듭니다. 실제로 열과 행의 수가 매우 많은 테이블을 사용하면 스토리지 효율성이 더욱 커집니다.

각 블록은 동일한 형식의 데이터를 보유하므로 블록 데이터는 열 데이터 형식에 맞게 선택된 압축 스키마를 사용하여 디스크 공간 및 I/O를 추가로 줄일 수 있다는 이점이 있습니다. 데이터 형식에 기반한 압축 인코딩에 대한 자세한 내용은 압축 인코딩 섹션을 참조하세요.

데이터를 디스크에 저장하기 위한 공간을 절약하면 데이터를 가져와서 메모리에 저장하는 작업에도 영향을 미칩니다. 대다수 데이터베이스 작업이 한 번에 1개 또는 소수의 열에서만 액세스하거나 실행되기 때문에 실제로 쿼리에 필요한 열 블록을 가져오는 것만으로도 메모리 공간을 절약할 수 있습니다. OLTP 트랜잭션에서 레코드 수가 적을 때 일반적으로 행 하나에 대부분 또는 모든 열이 포함될 경우, 행의 수가 매우 많을 때도 데이터 웨어하우스 쿼리는 일반적으로 몇 개의 열만 읽습니다. 이는 동일한 수의 행에 대해 동일한 수의 열 필드 값을 읽으려면 I/O 작업의 일부가 필요함을 의미합니다. 행 방향 블록을 처리하는 데 필요한 메모리의 일부만 사용합니다. 실제로 열과 행의 수가 매우 많은 테이블을 사용하면 효율성의 이점도 비례하여 커집니다. 예를 들어 테이블에 100개의 열이 있다고 가정하겠습니다. 이때 열을 5개 사용하는 쿼리는 테이블에 포함된 데이터 중에서 약 5%만 읽어오면 됩니다. 이러한 절약 효과는 대용량 데이터베이스의 레코드 수가 10억 개든, 혹은 심지어 1억 개든 반복됩니다. 이와 대조적으로 행 방향 데이터베이스는 불필요한 열 95개가 포함된 블록까지 읽어옵니다.

일반적인 데이터베이스 블록 크기 범위는 2~32KB입니다. Amazon Redshift는 1MB의 블록 크기를 사용하므로 데이터베이스 로드 또는 쿼리 실행의 일부인 다른 작업을 수행하는 데 필요한 I/O 요청의 수를 줄이고 더욱 효율적입니다.

워크로드 관리

이 섹션에서는 Amazon Redshift가 쿼리를 준비하고 실행하는 방법을 이해하는 데 도움이 되는 워크로드 관리(WLM)에 대해 설명합니다.

Amazon Redshift 워크로드 관리(WLM)는 워크로드 내에서 유연한 관리 우선 순위를 지원하므로, 짧고 빠르게 실행되는 쿼리가 오래 실행되는 쿼리 뒤에 대기열에 갇히지 않습니다. Amazon Redshift는 내부 시스템 대기열 및 사용자 액세스 가능 대기열을 포함하여 다양한 유형의 대기열에 대한 구성 파라미터를 정의하는 서비스 클래스에 따라 런타임에 쿼리 대기열을 작성합니다. 사용자 관점에서 보면 사용자 액세스 서비스 클래스와 대기열은 기능적으로 동일합니다. 일관성을 유지하기 위해 이 설명서에서는 queue라는 용어를 사용하여 사용자가 액세스할 수 있는 서비스 클래스와 런타임 큐를 정의합니다.

Redshift는 다양한 워크로드를 처리하도록 튜닝된 자동 워크로드 관리(자동 WLM)를 제공하며, 기본값으로 권장됩니다. 자동 WLM을 통해 Redshift는 쿼리가 도착하면 리소스 사용률을 결정하고 쿼리를 메인 클러스터에서 실행할지, 동시성 규모 조정 클러스터에서 실행할지, 아니면 각각 대기열로 보낼지를 동적으로 결정합니다. (쿼리가 대기열에 있는 경우 자동 WLM은 더 짧은 기간의 쿼리에 우선 순위를 둡니다.) 자동 WLM은 총 처리량을 극대화하고 효율적인 데이터 웨어하우스 리소스를 유지할 수 있도록 지원합니다. 워크로드의 크기나 예약 방식에 신경 쓸 필요 없이 워크로드를 실행할 수 있습니다. 프로비저닝된 클러스터의 기본값은 자동 WLM입니다. 자세한 내용은 자동 WLM 구현을 참조하세요.

> **Note**
>
> Amazon Redshift Serverless 작업 그룹은 항상 자동 WLM을 사용합니다.

많은 쿼리 또는 리소스 집약적인 쿼리가 실행되는 경우 워크로드가 로컬 리소스에 대기열에 대기할 때 워크로드 관리를 통해 추가 컴퓨팅 리소스로 확장할 수 있습니다. 자동 WLM을 통한 동시성 규모 조정은 사실상 무제한의 동시 사용자 및 쿼리에 대해 일관된 성능을 지원합니다.

세분화된 수동 최적화가 필요한 경우 Redshift 프로비저닝된 클러스터는 수동 WLM을 제공합니다. 여기서 고객은 리소스 할당, 쿼리 동시성 및 대기열을 관리합니다. 쿼리가 실행되면 WLM은 사용자의 사용자 그룹에 따라 또는 대기열 구성에 나열된 쿼리 그룹과 일치하여 쿼리를 대기열에 할당합니다. 이는 사용자가 설정하는 쿼리 그룹 레이블로 구성됩니다. 자세한 내용은 수동 WLM 구현을 참조하세요.

수동 WLM은 시간이 지남에 따라 워크로드 패턴에 맞게 미세 조정할 수 있지만, 대부분의 경우 정적 특성으로 인해 하루 또는 장기간에 걸쳐 변화하는 워크로드에 적응하기가 더 어려워질 수 있으므로 사용을 권장하지 않습니다. 더 많은 모니터링과 지속적인 튜닝이 필요합니다. 또한 수동 WLM은 할당된 메모리를 제한하기 위해 대기열을 수동으로 설정하는 등 많은 경우 자동 WLM만큼 컴퓨팅 리소스를 효율적으로 사용하지 못합니다.

워크로드 관리 구성의 성공을 측정하는 중요한 지표는 시스템 처리량, 즉 얼마나 많은 쿼리가 성공적으로 완료되었는지입니다. 시스템 처리량은 초당 쿼리 수로 측정됩니다. 시스템 지표에 대한 자세한 내용은 Amazon Redshift 클러스터 성능 모니터링 섹션을 참조하세요.

WLM 구성을 관리하는 가장 쉬운 방법은 Amazon Redshift 관리 콘솔을 사용하는 것입니다. Amazon Redshift 명령줄 인터페이스(CLI) 또는 Amazon Redshift API를 사용할 수도 있습니다. 워크로드 관리 구현 및 사용에 대한 자세한 내용은 워크로드 관리 구현 섹션을 참조하세요.

다른 서비스와 함께 Amazon Redshift 사용

이 섹션에서는 Amazon Redshift 데이터의 소스 및 대상으로 다른 서비스를 사용할 수 있는 방법을 설명합니다.

Amazon Redshift는 다른 AWS 서비스와 통합되어 데이터 보안 기능을 사용하여 데이터를 빠르고 안정적으로 이동, 변환 및 로드할 수 있습니다.

S3

Amazon Simple Storage Service(Amazon S3)는 클라우드에 데이터를 저장하는 웹 서비스입니다. Amazon Redshift는 병렬 처리를 활용하여 Amazon S3 버킷에 저장된 여러 데이터 파일에서 데이터를 읽고 로드합니다. 자세한 내용은 Amazon S3에서 데이터 로드 단원을 참조하십시오.

병렬 처리를 사용하여 Amazon Redshift 데이터웨어 하우스의 데이터를 Amazon S3의 여러 데이터 파일로 내보낼 수도 있습니다. 자세한 내용은 Amazon Redshift에서 데이터 언로드 단원을 참조하십시오.

DynamoDB

Amazon DynamoDB는 완전관리형 NoSQL 데이터베이스 서비스입니다. COPY 명령을 사용하여 단일 Amazon DynamoDB 테이블에서 데이터와 함께 Amazon Redshift 테이블을 로드할 수 있습니다. 자세한 내용은 Amazon DynamoDB 테이블에서 데이터 로드 단원을 참조하십시오.

SSH

Amazon Redshift의 COPY 명령을 사용하여 Amazon EMR 클러스터, Amazon EC2 인스턴스 또는 다른 컴퓨터와 같은 하나 이상의 원격 호스트에서 데이터를 로드할 수 있습니다. COPY는 SSH를 사용하여 원격 호스트에 연결하고 원격 호스트에서 명령을 실행해 데이터를 생성합니다. Amazon Redshift는 여러 동시 연결을 지원합니다. COPY 명령이 다수의 호스트 원본에서 병렬 방식으로 데이터를 읽어와 로드합니다. 자세한 내용은 원격 호스트에서 데이터 로드 단원을 참조하십시오.

AWS Data Pipeline

AWS Data Pipeline을 사용하여 Amazon Redshift에서 데이터 이동 및 변환을 자동화할 수 있습니다. AWS Data Pipeline의 기본 제공 예약 기능을 사용하면 직접 복잡한 데이터 전송 또는 변환 로직을 작성할 필요 없이 반복 작업을 예약 및 실행할 수 있습니다. 예를 들어 Amazon DynamoDB에서 Amazon Redshift로 자동으로 데이터를 복사하도록 되풀이 작업을 설정할 수 있습니다. Amazon S3에서 Amazon Redshift로 데이터를 주기적으로 이동하는 파이프라인을 생성하는 과정을 안내하는 튜토리얼은 AWS Data Pipeline Developer Guide의 Copy data to Amazon Redshift using AWS Data Pipeline 섹션을 참조하세요.

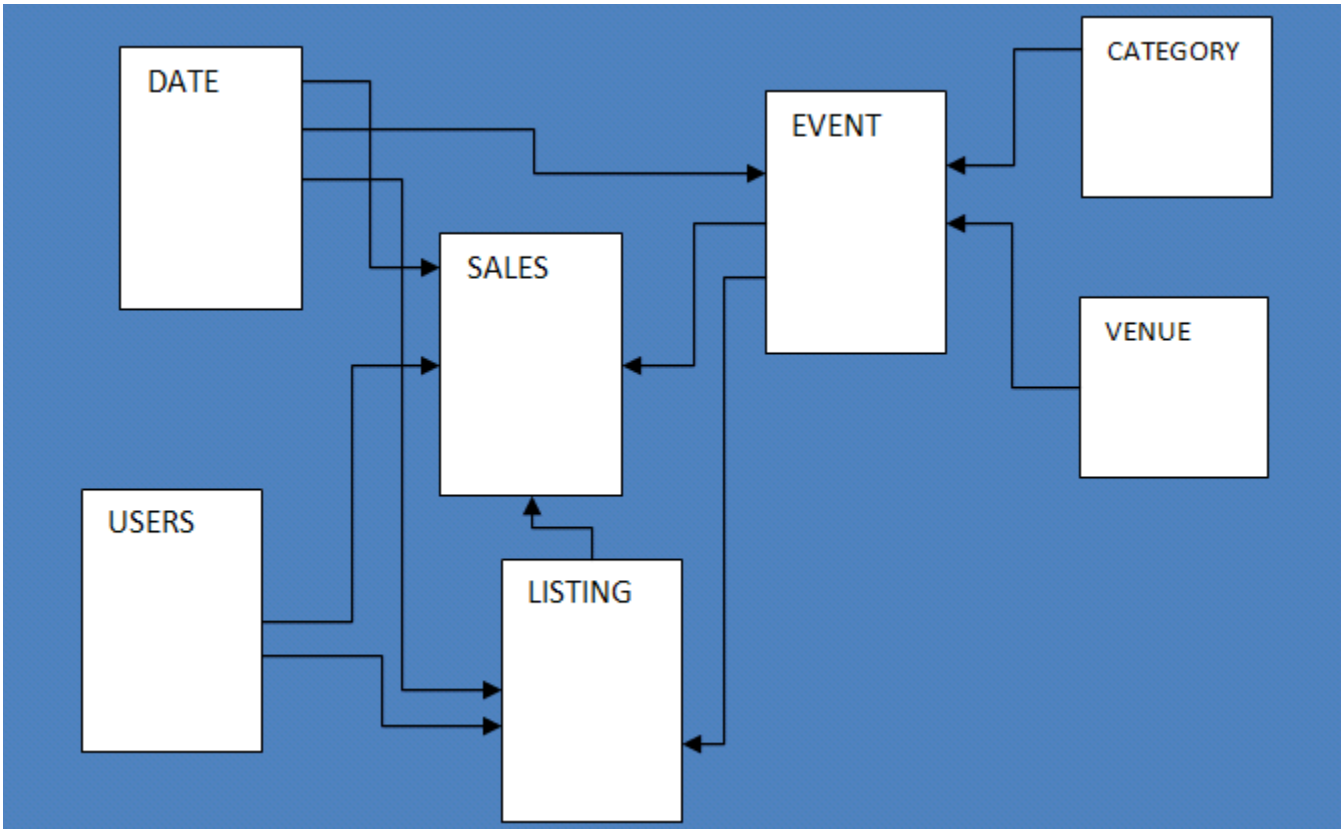
AWS DMS

AWS Database Migration Service를 사용하여 데이터를 Amazon Redshift로 마이그레이션할 수 있습니다. AWS DMS은 Oracle, PostgreSQL, Microsoft SQL Server, Amazon Redshift, Aurora DB 클러스터, DynamoDB, Amazon S3, MariaDB 및 MySQL과 같이 널리 사용되는 상용 및 오픈 소스 데이터베이스간에 데이터를 마이그레이션할 수 있습니다. 자세한 내용은 AWS Database Migration Service 대상으로 Amazon Redshift 데이터베이스 사용 섹션을 참조하세요.

샘플 데이터베이스

이 섹션에서는 Amazon Redshift 설명서 예제에서 사용하는 샘플 데이터베이스인 TICKIT에 대해 설명합니다.

이 작은 크기의 데이터베이스는 팩트 테이블 2개와 차원 테이블 5개, 총 7개의 테이블로 구성되어 있습니다. Amazon Redshift 시작 안내서의 4단계: Amazon S3에서 Amazon Redshift로 데이터 로드 단계에 따라 TICKIT 데이터 세트를 로드할 수 있습니다.



이 샘플 데이터베이스를 적용하면 사용자가 스포츠 이벤트, 공연 및 콘서트 등을 위한 온라인 티켓을 사고 팔 수 있는 가상의 TICKIT 웹사이트에서 분석가가 판매 작업을 추적하는 데 효과적입니다. 특히 시간 경과에 따른 티켓 이동과 판매자의 성공률, 그리고 티켓이 가장 많이 팔리는 이벤트, 공연장 및 계절 등을 구분할 수 있습니다. 분석가는 이러한 정보를 사용하여 사이트를 자주 방문하는 구매자와 판매자 모두에게 신규 사용자를 유치하거나 광고 및 홍보 효과를 높이는 동기를 제공할 수 있습니다.

예를 들어 다음은 2008년에 판매된 티켓 수를 기준으로 San Diego의 상위 판매자 5명을 찾는 쿼리입니다.

```
select sellerid, username, (firstname ||' '|| lastname) as name,
city, sum(qtysold)
```

```
from sales, date, users
where sales.sellerid = users.userid
and sales.dateid = date.dateid
and year = 2008
and city = 'San Diego'
group by sellerid, username, name, city
order by 5 desc
limit 5;

sellerid | username |       name        |   city    | sum
----------+----------+-------------------+-----------+-----
49977 | JJK84WTE | Julie Hanson      | San Diego |  22
19750 | AAS23BDR | Charity Zimmerman | San Diego |  21
29069 | SVL81MEQ | Axel Grant        | San Diego |  17
43632 | VAG08HKW | Griffin Dodson    | San Diego |  16
36712 | RXT40MKU | Hiram Turner      | San Diego |  14
(5 rows)
```

본 설명서의 예에서 사용되는 데이터베이스에는 작은 크기의 데이터 세트가 저장됩니다. 팩트 테이블 2개에는 각각 200,000개 미만의 행이 있고, 차원 테이블은 CATEGORY 테이블의 11개 행부터 USERS 테이블의 50,000개 행에 이르기까지 다양합니다.

특히 본 설명서의 데이터베이스 예는 다음과 같이 Amazon Redshift 테이블 설계의 주요 특성을 잘 드러내고 있습니다.

- 데이터 분산
- 데이터 정렬
- 열 기반 압축

TICKIT 데이터베이스의 테이블 스키마에 대한 자세한 내용을 보려면 다음 탭을 선택하세요.

CATEGORY table

열 명칭	데이터 유형	설명
CATID	SMALLINT	각 행의 고유 ID 값을 나타내는 기본 키입니다. 각 행마다 티켓이 매매되는 특정 유형의 이벤트를 나타냅니다.
CATGROUP	VARCHAR(10)	이벤트 그룹을 나타내는 서술형 이름(**Shows**, **Sports** 등)입니다.

열 명칭	데이터 유형	설명
CATNAME	VARCHAR(10)	그룹 내 이벤트 유형을 나타내는 짧은 서술형 이름(**Opera**, **Musicals** 등)입니다.
CATDESC	VARCHAR(50)	이벤트 유형을 나타내는 긴 서술형 이름(**Musical theatre** 등)입니다.

DATE table

열 명칭	데이터 유형	설명
DATEID	SMALLINT	각 행의 고유 ID 값을 나타내는 기본 키입니다. 각 행마다 연중 날짜를 나타냅니다.
CALDATE	날짜	날짜(**2008-06-24** 등)입니다.
DAY	CHAR)	짧은 형식의 주중 요일(**SA** 등)입니다.
주	SMALLINT	주의 수(**26** 등)입니다.
MONTH	CHAR)	짧은 형식의 월 이름(**JUN** 등)입니다.
QTR	CHAR)	분기 수(**1** ~ **4**)입니다.
YEAR	SMALLINT	4자리 연도(**2008** 등)입니다.
holiday	BOOLEAN	공휴일(미국 기준) 여부를 나타내는 플래그입니다.

EVENT table

열 명칭	데이터 유형	설명
EVENTID	INTEGER	각 행의 고유 ID 값을 나타내는 기본 키입니다. 각 행마다 특정 장소와 시간에 열리는 이벤트를 나타냅니다.
VENUEID	SMALLINT	VENUE 테이블에 대한 외래 키 참조입니다.

열 명칭	데이터 유형	설명
CATID	SMALLINT	CATEGORY 테이블에 대한 외래 키 참조입니다.
DATEID	SMALLINT	DATE 테이블에 대한 외래 키 참조입니다.
EVENTNAME	VARCHAR(200)	**Hamlet** 또는 **La Traviata**와 같은 이벤트 이름입니다.
StartTime	TIMESTAMP	이벤트가 개최되는 연도 및 날짜와 시작 시간(**2008-10-10 19:30:00** 등)입니다.

VENUE table

열 명칭	데이터 유형	설명
VENUEID	SMALLINT	각 행의 고유 ID 값을 나타내는 기본 키입니다. 각 행마다 이벤트가 열리는 특정 장소를 나타냅니다.
VENUENAME	VARCHAR(100)	정확한 장소 이름(**Cleveland Browns Stadium** 등)입니다.
VENUECITY	VARCHAR(30)	도시 이름(**Cleveland** 등)입니다.
VENUESTATE	CHAR)	미국 및 캐나다를 기준으로 2자로 축약된 주 또는 지방(**OH** 등)입니다.
VENUESEATS	INTEGER	알려진 경우에 한해 해당 장소에서 사용할 수 있는 최대 좌석 수(**73200** 등)입니다. 이 열에는 표현상 null 값이나 0이 포함되기도 합니다.

USERS table

열 명칭	데이터 유형	설명
USERID	INTEGER	각 행의 고유 ID 값을 나타내는 기본 키입니다. 각 행마다 1개 이상의 이벤트에서 티켓을 판매하였거나

열 명칭	데이터 유형	설명
		구매한 등록 사용자(구매자, 판매자 또는 둘 다)를 나타냅니다.
USERNAME	CHAR(8)	8자로 구성된 영숫자 사용자 이름(**PGL08LJI** 등)입니다.
FirstName	VARCHAR(30)	사용자의 이름(**Victor** 등)입니다.
LastName	VARCHAR(30)	사용자의 성(**Hernandez** 등)입니다.
CITY	VARCHAR(30)	사용자가 태어난 도시(**Naperville** 등)입니다.
STATE	CHAR)	사용자가 태어난 주(**GA** 등)입니다.
EMAIL	VARCHAR(100)	사용자의 이메일 주소입니다. 이 열에는 임의의 Latin 값(**turpis@accumsanlaoreet.org** 등)이 포함됩니다.
PHONE	CHAR)	14자리의 사용자 전화 번호(**(818) 765-4255** 등)입니다.
LIKESPORTS, ...	BOOLEAN	**true** 및 **false** 값으로 사용자의 호불호를 식별할 수 있도록 연속된 10개의 열입니다.

LISTING table

열 명칭	데이터 유형	설명
LISTID	INTEGER	각 행의 고유 ID 값을 나타내는 기본 키입니다. 각 행마다 특정 이벤트의 티켓 목록을 나타냅니다.
SELLERID	INTEGER	USERS 테이블에 대한 외래 키 참조로서 티켓을 판매하는 사용자를 식별합니다.
EVENTID	INTEGER	EVENT 테이블에 대한 외래 키 참조입니다.
DATEID	SMALLINT	DATE 테이블에 대한 외래 키 참조입니다.

열 명칭	데이터 유형	설명
NUMTICKETS	SMALLINT	판매할 수 있는 티켓 수(**2**, **20** 등)입니다.
PRICEPERTICKET	DECIMAL(8,2)	티켓 1장당 정가(**27.00**, **206.00** 등)입니다.
TOTALPRICE	DECIMAL(8,2)	이 목록의 가격 총액(NUMTICKETS*PRICEPERTICKET)입니다.
LISTTIME	TIMESTAMP	이 목록이 게시된 연도 및 날짜와 시간(**2008-03-18 07:19:35** 등)입니다.

SALES table

열 명칭	데이터 유형	설명
SALESID	INTEGER	각 행의 고유 ID 값을 나타내는 기본 키입니다. 각 행마다 특정 목록에서 나타낸 것처럼 특정 이벤트에 대한 1장 이상의 티켓 판매를 나타냅니다.
LISTID	INTEGER	LISTING 테이블에 대한 외래 키 참조입니다.
SELLERID	INTEGER	USERS 테이블(티켓을 판매한 사용자)에 대한 외래 키 참조입니다.
BUYERID	INTEGER	USERS 테이블(티켓을 구매한 사용자)에 대한 외래 키 참조입니다.
EVENTID	INTEGER	EVENT 테이블에 대한 외래 키 참조입니다.
DATEID	SMALLINT	DATE 테이블에 대한 외래 키 참조입니다.
QTYSOLD	SMALLINT	판매된 티켓 수(**1** ~ **8**)입니다. 거래 1회에 판매 가능한 최대 티켓 수는 8장입니다.
PRICEPAID	DECIMAL(8,2)	티켓을 판매한 가격 총액(**75.00**, **488.00** 등)입니다. 티켓 1장당 가격은 PRICEPAID/QTYSOLD입니다.

열 명칭	데이터 유형	설명
COMMISSION	DECIMAL(8,2)	사업체에서 티켓 판매 시 징수하는 15% 수수료(**11.25**, **73.20** 등)입니다. 판매자는 PRICEPAID 값의 85%를 받습니다.
SALETIME	TIMESTAMP	판매가 완료된 연도 및 날짜와 시간(**2008-05-24 06:21:47** 등)입니다.

Amazon Redshift 모범 사례

다음에는 개념 증명 계획, 테이블 디자인, 테이블에 데이터로드 및 Amazon Redshift에 대한 쿼리 작성, 그리고 Amazon Redshift Advisor 작업에 대한 토론에 대한 모범 사례를 찾을 수 있습니다.

Amazon Redshift는 다른 SQL 데이터베이스 시스템과 다릅니다. Amazon Redshift 아키텍처의 장점을 완전히 구현하려면 테이블을 전문적으로 설계, 빌드 및 로드하여 대용량 병렬 처리, 열 기반 데이터 스토리지 및 열 기반 데이터 압축을 사용해야 합니다. 데이터 로딩 및 쿼리 실행 시간이 예상보다, 혹은 원하는 것보다 길어지면 키 정보를 놓칠 수도 있기 때문입니다.

숙련된 SQL 데이터베이스 개발자인 경우 Amazon Redshift 데이터 웨어하우스 개발을 시작하기 전에 이 주제를 검토하는 것이 좋습니다.

SQL 데이터베이스 개발을 처음 하는 경우 이 주제는 시작하기 위한 위치로 적합하지 않습니다. 먼저 Amazon Redshift 시작 안내서에서 데이터 웨어하우스의 데이터베이스를 정의하고 사용하기 위한 실행 명령을 읽고 직접 예제를 사용해 보는 것이 좋습니다.

이 주제에서는 가장 중요한 개발 원칙에 대한 개요를 살펴보고 이러한 원칙을 구현하기 위한 구체적인 팁, 예 및 모범 사례도 알아볼 수 있습니다. 모범 사례라고 해서 일일이 모든 환경에 적용할 수 있는 것은 아닙니다. 데이터베이스 설계를 마무리하기 전에 모든 옵션을 평가하십시오. 자세한 내용은 자동 테이블 최적화Amazon Redshift에서 데이터 로드쿼리 성능 튜닝 및 & Reference를 참조하십시오.

주제

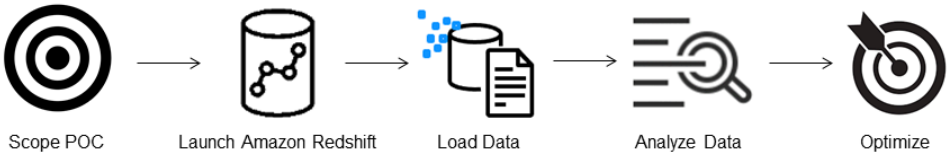
- Amazon Redshift에 대한 개념 증명(POC) 수행
- Amazon Redshift 테이블 설계 모범 사례
- 데이터 로드에 대한 Amazon Redshift 모범 사례
- Amazon Redshift 쿼리 설계 모범 사례
- Amazon Redshift Advisor의 권장 사항 준수

Amazon Redshift에 대한 개념 증명(POC) 수행

Amazon Redshift는 널리 사용되는 클라우드 데이터 웨어하우스로, 조직의 Amazon Simple Storage Service 데이터 레이크, 실시간 스트림, 기계 학습(ML) 워크플로, 트랜잭션 워크플로 등과 통합되는 완전관리형 클라우드 기반 서비스를 제공합니다. 이어지는 섹션에서는 Amazon Redshift에 대한 개념 증명(POC)을 수행하는 과정을 안내합니다. 이 정보는 POC의 목표를 설정하고 POC에 대한 서비스 프로비저닝 및 구성을 자동화할 수 있는 도구를 활용하는 데 도움이 됩니다.

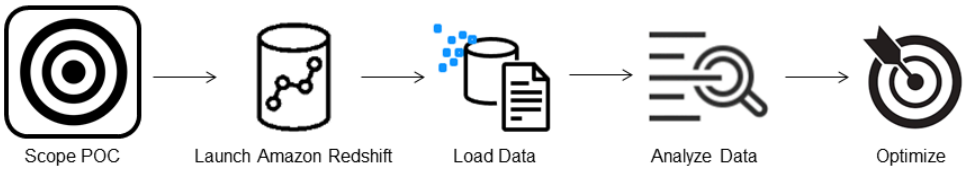
> **Note**
>
> 이 정보를 PDF로 복사하려면 Amazon Redshift 리소스 페이지에서 자체 Redshift POC 실행 링크를 선택하세요.

Amazon Redshift의 POC를 수행할 때는 동급 최고의 보안 기능, 탄력적 규모 조정, 간편한 통합 및 수집, 유연한 분산형 데이터 아키텍처 옵션에 이르는 다양한 기능을 테스트하고 검증하고 채택합니다.



POC를 성공적으로 수행하려면 다음 단계를 따르세요.

1단계: POC의 범위



POC를 수행할 때 자체 데이터를 사용하거나 벤치마킹 데이터세트를 사용할 수 있습니다. 자체 데이터를 선택하면 해당 데이터에 대해 자체 쿼리를 실행합니다. 벤치마킹 데이터를 선택하면 벤치마크와 함께 샘플 쿼리가 제공됩니다. 아직 자체 데이터로 POC를 수행할 준비가 되지 않은 경우 자세한 내용은 샘플 데이터세트 사용을 참조하세요.

일반적으로 Amazon Redshift POC를 위해 2주 분량의 데이터를 사용하는 것이 좋습니다.

다음을 수행하여 시작하세요.

1. 비즈니스 및 기능 요구 사항을 파악한 다음 거슬러 올라갑니다. 일반적인 예로는 성능 속도 향상, 비용 절감, 새로운 워크로드 또는 기능 테스트, Amazon Redshift와 다른 데이터 웨어하우스 간의 비교 등이 있습니다.

2. POC의 성공 기준이 되는 구체적인 목표를 설정합니다. 예를 들어, 성능 속도 향상을 먼저 결정한 후 가속화하려는 상위 5개 프로세스의 목록을 작성하고 현재 실행 시간과 필요한 실행 시간을 포함하세요. 보고서, 쿼리, ETL 프로세스, 데이터 모으기 등 현재 겪고 있는 문제점이라면 무엇이든 가능합니다.
3. 테스트를 실행하는 데 필요한 구체적인 범위와 아티팩트를 식별합니다. Amazon Redshift로 마이그레이션하거나 지속적으로 수집해야 하는 데이터세트는 무엇이며 성공 기준을 바탕으로 측정하기 위한 테스트를 실행하려면 어떤 쿼리와 프로세스가 필요한가요? 이렇게 하는 방법은 두 가지입니다.

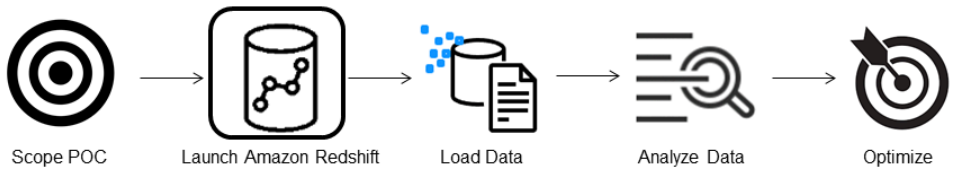
   자체 데이터 사용

   - 자체 데이터를 테스트하려면 성공 기준을 바탕으로 테스트하는 데 필요한 최소 실행 가능한 데이터 아티팩트 목록을 작성하세요. 예를 들어 현재 데이터 웨어하우스에 200개의 테이블이 있지만 테스트하려는 보고서에는 20개만 필요한 경우 더 적은 수인 20개만 사용하여 POC를 더 빠르게 실행할 수 있습니다.

   샘플 데이터세트 사용

   - 자체 데이터세트가 준비되지 않은 경우에도 TPC-DS 또는 TPC-H와 같은 업계 표준 벤치마크 데이터세트를 사용하고 샘플 벤치마킹 쿼리를 실행하면 Amazon Redshift에 대한 POC를 시작하여 Amazon Redshift의 성능을 활용할 수 있습니다. 이러한 데이터세트는 Amazon Redshift 데이터 웨어하우스가 생성된 후 해당 데이터 웨어하우스 내에서 액세스할 수 있습니다. 이러한 데이터세트와 샘플 쿼리에 액세스하는 방법에 대한 지침은 2단계: Amazon Redshift 시작 섹션을 참조하세요.

2단계: Amazon Redshift 시작



Amazon Redshift는 규모에 따른 빠르고 쉽고 안전한 클라우드 데이터 웨어하우징을 통해 인사이트를 얻는 시간을 단축합니다. Redshift Serverless 콘솔에서 웨어하우스를 실행하여 빠르게 시작하고 몇 초 만에 데이터에서 인사이트를 얻을 수 있습니다. Redshift Serverless를 사용하면 데이터 웨어하우스 관리에 대한 걱정 없이 비즈니스 성과를 제공하는 데 집중할 수 있습니다.

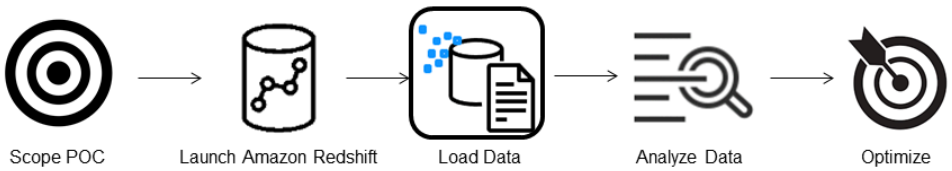
Amazon Redshift Serverless 설정

Redshift Serverless를 처음 사용할 때 콘솔은 웨어하우스를 시작하는 데 필요한 단계를 안내합니다. 또한 계정의 Redshift 서버리스 사용량에 대한 크레딧을 받을 자격이 있을 수도 있습니다. 무료 평가판 선택에 대한 자세한 내용은 Amazon Redshift 무료 평가판을 참조하세요. Amazon Redshift 시작 안내서의 Redshift 서버리스를 사용하여 데이터 웨어하우스 생성에 나와 있는 단계에 따라 Redshift Serverless로 데이터 웨어하우스를 생성하세요. 로드하고 싶은 데이터세트가 없는 경우 샘플 데이터세트를 로드하는 방법에 대한 단계도 이 안내서에서 확인할 수 있습니다.

이전에 계정에서 Redshift Serverless를 시작한 경우 Amazon Redshift 관리 안내서의 네임스페이스가 있는 작업 그룹 생성의 단계를 따르세요. 웨어하우스를 사용할 수 있게 되면 Amazon Redshift에서 사용할 수 있는 샘플 데이터를 로드하도록 선택할 수 있습니다. Amazon Redshift 쿼리 에디터 v2를 사용한 데이터 로드에 대한 자세한 내용은 Amazon Redshift 관리 안내서의 샘플 데이터 로드를 참조하세요.

샘플 데이터세트를 로드하는 대신 자체 데이터를 사용하는 경우 3단계: 데이터 로드 섹션을 참조하세요.

3단계: 데이터 로드



Redshift Serverless를 시작한 후 다음 단계는 POC에 사용할 데이터를 로드하는 것입니다. 간단한 CSV 파일을 업로드하든, S3에서 반정형 데이터를 수집하든, 데이터를 직접 스트리밍하든, Amazon Redshift는 데이터를 소스에서 Amazon Redshift 테이블로 빠르고 쉽게 이동할 수 있는 유연성을 제공합니다.

다음 방법 중 하나를 선택하여 데이터를 로드합니다.

로컬 파일 업로드

빠른 수집 및 분석을 위해 Amazon Redshift 쿼리 에디터 v2를 사용하여 로컬 데스크톱에서 데이터 파일을 쉽게 로드할 수 있습니다. CSV, JSON, AVRO, PARQUET, ORC 등과 같은 다양한 형식의 파일을 처리할 수 있는 기능이 있습니다. 관리자가 사용자에게 쿼리 에디터 v2를 사용하여 로컬 데스크톱에

서 데이터를 로드하는 권한을 부여하려면 공통 Amazon S3 버킷을 지정하고 사용자 계정을 적절한 권한으로 구성해야 합니다. 단계별 안내는 Data load made easy and secure in Amazon Redshift using Query Editor V2를 확인하세요.

Amazon S3 파일 로드

Amazon S3 버킷에서 Amazon Redshift로 데이터를 로드하려면 먼저 소스 Amazon S3 위치와 대상 Amazon Redshift 테이블을 지정하여 COPY 명령을 사용하세요. Amazon Redshift가 지정된 Amazon S3 버킷에 액세스할 수 있도록 IAM 역할 및 권한이 적절하게 구성되어 있는지 확인하세요. 단계별 지침은 튜토리얼: Amazon S3에서 데이터 로드를 참조하세요. 쿼리 에디터 v2에서 데이터 로드 옵션을 선택하여 S3 버킷에서 직접 데이터를 로드할 수도 있습니다.

지속적인 데이터 모으기

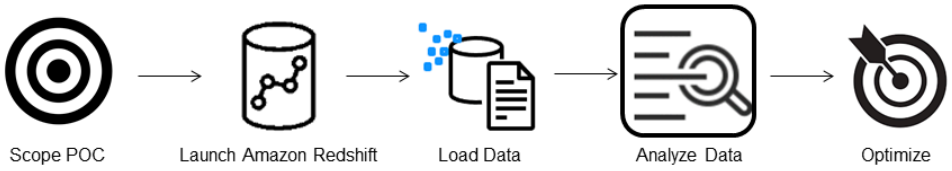
자동 복사(미리 보기)는 COPY 명령의 익스텐션으로, Amazon S3 버킷에서의 지속적인 데이터 로드를 자동화합니다. 복사 작업을 생성하면 Amazon Redshift는 지정된 경로에 새 Amazon S3 파일이 생성될 때 감지하여 사용자 개입 없이 자동으로 로드합니다. Amazon Redshift는 로드된 파일을 추적하여 한 번만 로드되었는지 확인합니다. 복사 작업을 생성하는 방법에 대한 설명은 COPY JOB 섹션을 참조하세요.

> **Note**
>
> 자동 복사는 현재 미리 보기 단계이며 특정 AWS 리전에서 프로비저닝된 클러스터에만 지원됩니다. 자동 복사를 위한 미리 보기 클러스터를 생성하려면 Amazon S3 버킷에서 자동으로 파일을 복사하기 위해 S3 이벤트 통합 만들기 섹션을 참조하세요.

스트리밍 데이터 로드

스트리밍 수집을 사용하면 지연 시간이 짧고 빠른 속도로 Amazon Kinesis Data Streams 및 Amazon Managed Streaming for Apache Kafka에서 Amazon Redshift로 스트림 데이터를 수집할 수 있습니다. Amazon Redshift 스트리밍 수집은 구체화된 뷰를 사용합니다. 이 뷰는 자동 새로 고침 사용 중에 스트림에서 직접 업데이트됩니다. 구체화된 보기는 스트림 데이터 원본에 매핑됩니다. 구체화된 뷰 정의의 일부로 스트림 데이터에 대한 필터링 및 집계를 수행할 수 있습니다. 스트림에서 데이터를 로드하기 위한 단계별 지침은 Amazon Kinesis Data Streams 시작하기 또는 Getting started with Amazon Managed Streaming for Apache Kafka를 참조하세요.

4단계: 데이터 분석



Redshift Serverless 작업 그룹과 네임스페이스를 만들고 데이터를 로드한 후에는 Redshift Serverless 콘솔의 탐색 패널에서 쿼리 에디터 v2를 열어 즉시 쿼리를 실행할 수 있습니다. 쿼리 에디터 v2를 사용하여 쿼리 기능을 테스트하거나 자체 데이터세트를 바탕으로 성능을 쿼리할 수 있습니다.

Amazon Redshift 쿼리 에디터 v2를 사용하여 쿼리

Amazon Redshift 콘솔에서 쿼리 에디터 v2에 액세스할 수 있습니다. 쿼리 에디터 v2를 사용하여 쿼리를 구성, 연결 및 실행하는 방법에 대한 전체 가이드는 Simplify your data analysis with Amazon Redshift query editor v2를 참조하세요.

또는 POC의 일환으로 로드 테스트를 실행하려는 경우 다음 단계에 따라 Apache JMeter를 설치하고 실행하면 됩니다.

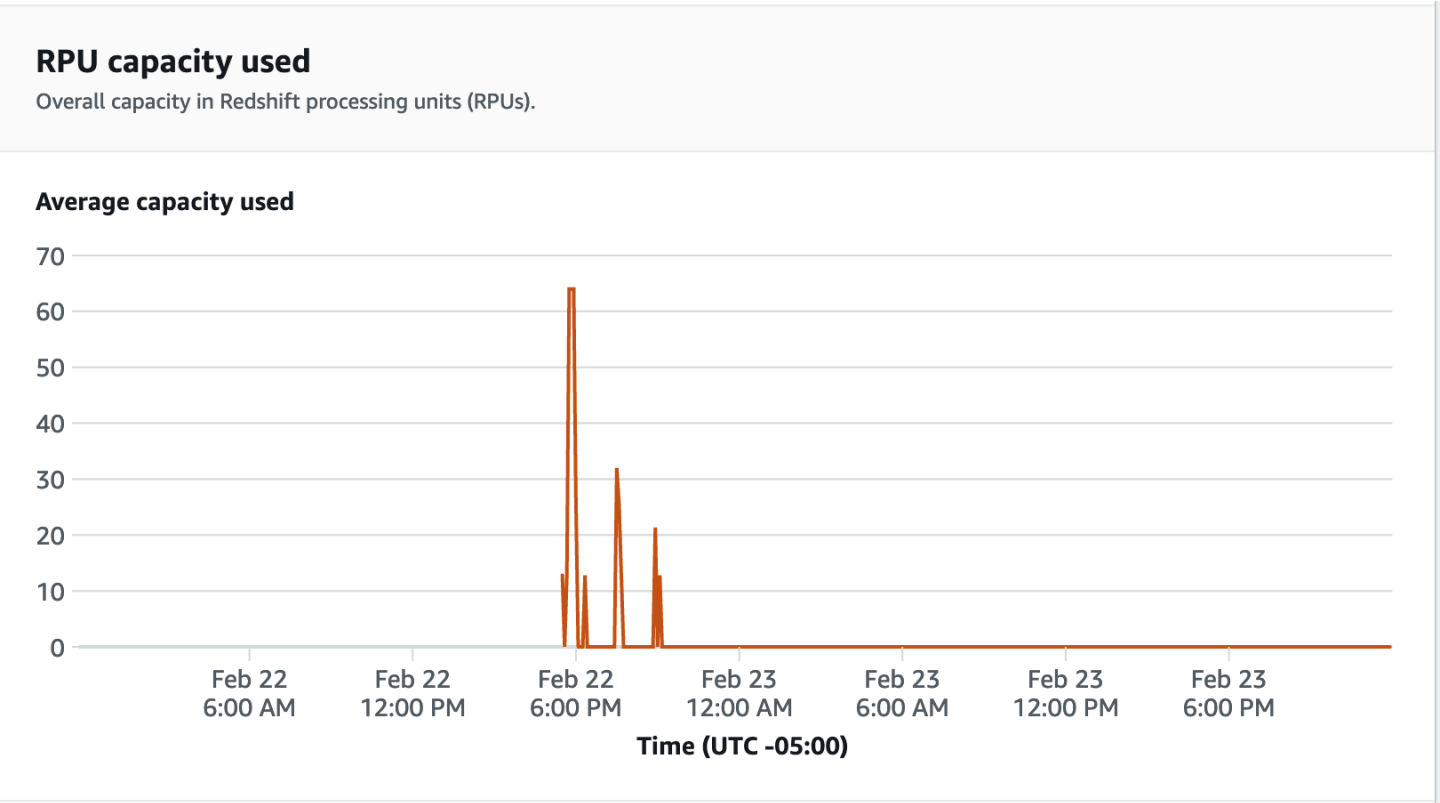
Apache JMeter를 사용하여 로드 테스트 실행

Amazon Redshift에 쿼리를 동시에 제출하는 'N'명의 사용자를 시뮬레이션하는 로드 테스트를 수행하려면 오픈 소스 Java 기반 도구인 Apache JMeter를 사용하면 됩니다.

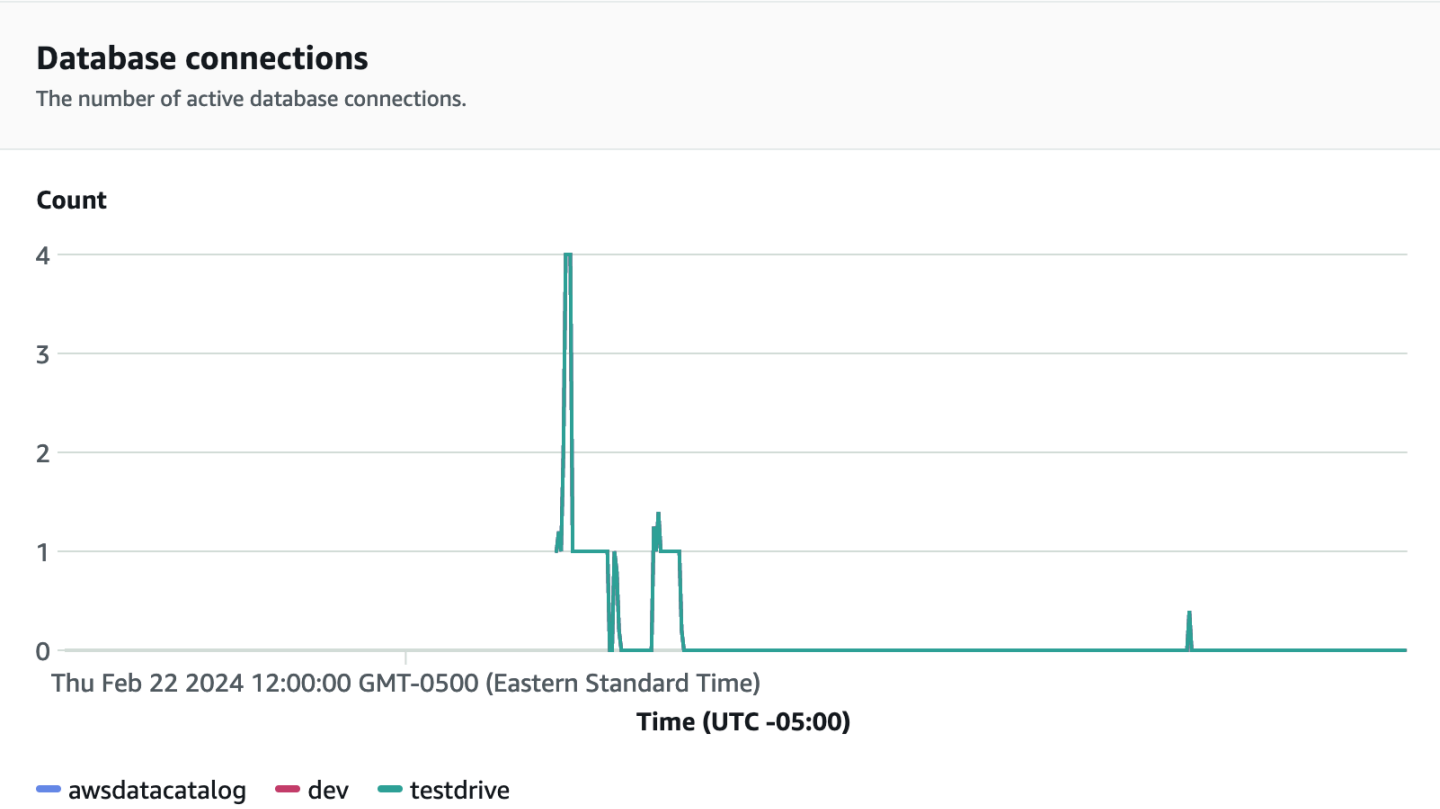
Redshift Serverless 작업 그룹에 대해 실행되도록 Apache JMeter를 설치하고 구성하려면 Automate Amazon Redshift load testing with the AWS Analytics Automation Toolkit의 지침을 따르세요. 이 도구는 Redshift 솔루션을 동적으로 배포하기 위한 오픈 소스 유틸리티인 AWS Analytics Automation toolkit(AAA)을 사용하여 이러한 리소스를 자동으로 시작합니다. Amazon Redshift에 자체 데이터를 로드한 경우 #5단계 - SQL 사용자 지정 옵션을 수행하여 테이블에 대해 테스트하려는 적절한 SQL 문을 제공해야 합니다. 쿼리 에디터 v2를 사용하여 각 SQL 문을 한 번 테스트하여 오류 없이 실행되는지 확인하세요.

SQL 문을 사용자 지정하고 테스트 계획을 완료한 후에는 테스트 계획을 저장하고 Redshift Serverless 작업 그룹에 대해 실행하세요. 테스트 진행 상황을 모니터링하려면 Redshift Serverless 콘솔을 열고 쿼리 및 데이터베이스 모니터링으로 이동한 다음 쿼리 기록 탭을 선택하고 쿼리에 대한 정보를 확인합니다.

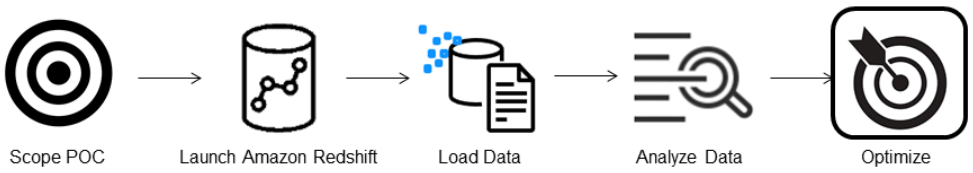
성능 지표의 경우 Redshift Serverless 콘솔에서 데이터베이스 성능 탭을 선택하여 데이터베이스 연결 및 CPU 사용률과 같은 지표를 모니터링할 수 있습니다. 여기에서 그래프를 통해 작업 그룹에서 로드 테스트가 실행되는 동안 사용된 RPU 용량을 모니터링하고 Redshift Serverless가 동시 워크로드 수요를 충족하기 위해 어떻게 자동으로 규모를 조정하는지 볼 수 있습니다.



데이터베이스 연결은 작업 그룹이 증가하는 워크로드 수요를 충족하기 위해 주어진 시간에 수많은 동시 연결을 처리하는 방식을 확인하기 위해 로드 테스트를 실행하는 동안 모니터링할 수 있는 또 다른 유용한 지표입니다.



5단계: 최적화



Amazon Redshift는 개별 사용 사례를 지원하는 다양한 구성과 기능을 제공하여 수만 명의 사용자가 매일 엑사바이트 규모의 데이터를 처리하고 분석 워크로드를 강화할 수 있도록 지원합니다. 이러한 옵션 중에서 선택할 때 고객은 Amazon Redshift 워크로드 지원에 가장 적합한 데이터 웨어하우스 구성을 결정하는 데 도움이 되는 도구를 찾고 있습니다.

테스트 드라이브

테스트 드라이브를 사용하면 잠재적 구성에서 기존 워크로드를 자동으로 재생하고 해당 출력을 분석하여 워크로드를 마이그레이션할 최적의 대상을 평가할 수 있습니다. 테스트 드라이브를 사용하여 각기 다른 Amazon Redshift 구성을 평가하는 방법에 대한 자세한 내용은 Find the best Amazon Redshift configuration for your workload using Redshift Test Drive를 참조하세요.

Amazon Redshift 테이블 설계 모범 사례

데이터베이스를 계획할 때는 전반적인 쿼리 성능에 크게 영향을 미칠 수 있는 몇 가지 핵심적인 테이블 설계 조건이 있습니다. 이러한 설계 조건은 스토리지 요건에도 상당한 영향을 미치기 때문에 I/O 작업 수를 줄이거나 쿼리를 처리하는 데 필요한 메모리를 최소화하여 결국 쿼리 성능까지 떨어지게 됩니다.

이 섹션에서는 가장 중요한 설계 결정에 대한 요약과 쿼리 성능을 최적화하기 위한 최상의 방법을 볼 수 있습니다. 자동 테이블 최적화에서는 테이블 설계 옵션에 대한 보다 자세한 설명과 예를 제공합니다.

주제

- 최상의 정렬 키 선택
- 최상의 배포 스타일 선택
- COPY를 통한 압축 인코딩 선택
- 기본 키와 외래 키의 제약 조건 정의
- 가능한 최소 열 크기 사용
- 날짜 열의 날짜/시간 데이터 형식 사용

최상의 정렬 키 선택

Amazon Redshift는 정렬 키에 따라 정렬된 순서로 디스크에 데이터를 저장합니다. Amazon Redshift 쿼리 옵티마이저는 최적의 쿼리 계획을 결정할 때 정렬 순서를 사용합니다.

> **Note**
>
> 자동 테이블 최적화를 사용하는 경우 테이블의 정렬 키를 선택할 필요가 없습니다. 자세한 내용은 자동 테이블 최적화 단원을 참조하십시오.

가장 좋은 방법에 대한 몇 가지 제안은 다음과 같습니다.

- Amazon Redshift에서 적절한 정렬 순서를 선택하도록 하려면 정렬 키에 대해 AUTO를 지정합니다.
- 최신 데이터를 가장 자주 쿼리하는 경우, 타임스탬프 열을 정렬 키의 선행 열로 지정합니다.

  그러면 시간 범위에서 벗어나는 블록 전체를 건너뛸 수 있기 때문에 쿼리 효율성이 더욱 향상됩니다.
- 하나의 열에 범위 필터링이나 동등 필터링을 자주 수행하는 경우, 해당 열을 정렬 키로 지정합니다.

Amazon Redshift는 해당 열의 전체 데이터 블록 읽기를 건너뛸 수 있습니다. 그렇게 할 수 있는 이유는 각 블록에 저장된 최소 및 최대 열 값을 추적하여 조건자 범위에 적용되지 않는 블록을 건너뛸 수 있기 때문입니다.

- 자주 테이블을 조인할 경우 조인 열을 정렬 키와 분배 키로 지정합니다.

  그러면 쿼리 최적화 프로그램이 느린 해시 조인 대신 정렬 병합 조인을 선택할 수 있습니다. 또한 데이터가 이미 조인 키를 기준으로 정렬되어 있기 때문에 쿼리 옵티마이저가 정렬 병합 조인의 정렬 단계를 우회할 수 있습니다.

최상의 배포 스타일 선택

쿼리를 실행하면 쿼리 옵티마이저가 필요에 따라 쿼리 행을 컴퓨팅 노드로 다시 분산시켜 조인 및 집계를 실행합니다. 테이블 배포 스타일을 선택하는 목적은 쿼리 실행 이전에 데이터의 배포 위치를 지정하여 재배포 단계의 영향을 최소화하는 데 있습니다.

> **Note**
>
> 자동 테이블 최적화를 사용하는 경우 테이블의 배포 스타일을 선택할 필요가 없습니다. 자세한 내용은 자동 테이블 최적화 단원을 참조하십시오.

가장 좋은 방법에 대한 몇 가지 제안은 다음과 같습니다.

1. 공통 열을 기준으로 팩트 테이블과 차원 테이블 1개를 분산시킵니다.

   팩트 테이블은 분산 키가 1개로 제한됩니다. 따라서 다른 키로 조인하는 테이블은 팩트 테이블과 함께 배치되지 않습니다. 조인 횟수와 조인 행의 크기를 기준으로 함께 배치할 차원 테이블을 1개 선택합니다. 그런 다음 차원 테이블의 기본 키와 팩트 테이블의 해당 외래 키를 모두 DISTKEY로 지정합니다.
2. 필터링된 데이터 집합의 크기를 기준으로 가장 큰 차원을 선택합니다.

   조인에 사용되는 행만 분산시켜야 하기 때문에 테이블 크기가 아닌 필터링 이후 데이터 세트의 크기를 고려합니다.
3. 필터링 된 결과 집합에서 카디널리티가 높은 열을 선택합니다.

   예를 들어 판매 테이블을 날짜 열에 배포하는 경우 대부분의 판매가 계절에 따라 편중되지 않는 한 데이터가 상당히 고르게 배포될 수 있습니다. 하지만 범위 제한 조건자를 공통적으로 사용하여 좁은

날짜 기간 동안 필터링하는 경우에는 필터링되는 행의 대부분이 제한된 조각(slice) 세트에서 발생하므로 쿼리 워크로드가 편향됩니다.

4. ALL 분산을 사용하도록 일부 차원 테이블을 변경합니다.

   차원 테이블을 팩트 테이블이나 기타 중요한 조인 테이블과 함께 배치할 수 없는 경우에는 전체 테이블을 모든 노드로 분산시켜 쿼리 성능을 크게 높일 수 있습니다. ALL 분산을 사용하면 스토리지 공간 요건이 크게 늘어날 뿐만 아니라 로그 시간 및 유지 관리 작업도 증가합니다. 따라서 ALL 분산을 선택하려면 먼저 모든 인자에 가중치를 반영해야 합니다.

Amazon Redshift에서 적절한 배포 스타일을 선택하도록 하려면 배포 스타일에 AUTO를 지정합니다.

배포 스타일의 선택에 대한 자세한 내용은 쿼리 최적화를 위한 데이터 배포 섹션을 참조하세요.

COPY를 통한 압축 인코딩 선택

압축 인코딩은 테이블을 생성할 때 지정할 수 있지만 대부분의 경우 자동 압축을 사용하면 최상의 결과를 얻을 수 있습니다.

ENCODE AUTO는 테이블의 기본값입니다. 테이블이 ENCODE AUTO로 설정되어 있는 경우 Amazon Redshift는 테이블의 모든 열에 대한 압축 인코딩을 자동으로 관리합니다. 자세한 내용은 CREATE TABLE 및 ALTER TABLE 단원을 참조하세요.

COPY 명령은 데이터를 분석한 후 로드 작업의 일환으로서 압축 인코딩을 빈 테이블에 자동으로 적용합니다.

자동 압축은 압축 인코딩을 선택할 때 전반적인 성능의 밸런스를 유지합니다. 범위가 제한된 스캔은 정렬 키 열이 동일한 쿼리의 다른 열보다 훨씬 높게 압축된 경우 성능이 떨어질 수 있습니다. 결과적으로 자동 압축은 비교적 덜 효율적인 압축 인코딩을 선택하여 정렬 키 열과 다른 열의 밸런스를 유지합니다.

테이블의 정렬 키가 날짜 또는 타임스탬프이고 테이블에서 큰 varchar 열을 많이 사용한다고 가정합니다. 이 경우 정렬 키 열을 전혀 압축하지 않아서 더 좋은 성능을 얻을 수 있습니다. 테이블에서 ANALYZE COMPRESSION 명령을 실행한 후 인코딩을 사용하여 새로운 테이블을 생성하고, 정렬 키의 압축 인코딩은 배제하면 됩니다.

자동 압축 인코딩은 성능 비용이 있지만, 이는 테이블이 비어 있고 압축 인코딩이 없는 경우에 한합니다. 스테이징 테이블과 같이 자주 생성하는 수명이 짧은 테이블 및 테이블의 경우 자동 압축으로 테이블을 한 번 로드하거나 ANALYZE COMPRESSION 명령을 실행합니다. 그런 다음이 인코딩을 사용하여 새 테이블을 만듭니다. 또한 인코딩을 CREATE TABLE 문에 추가하거나, 혹은 CREATE TABLE LIKE를 사용하여 동일한 인코딩으로 새로운 테이블을 생성할 수 있습니다.

자세한 내용은 자동 압축을 사용하여 테이블 로드 단원을 참조하십시오.

기본 키와 외래 키의 제약 조건 정의

해당되는 경우가 있으면 테이블 사이에 기본 키와 외래 키 제약 조건을 정의합니다. 제약 조건이 참고용이긴 하지만 쿼리 옵티마이저는 이 제약 조건을 사용하여 더욱 효율적인 쿼리 계획을 작성합니다.

애플리케이션에서 제약 조건을 적용하지 않는 한 기본 키 제약 조건과 외래 키 제약 조건을 정의하지 않습니다. Amazon Redshift는 고유, 기본 키 및 외래 키 제약 조건을 적용하지 않습니다.

Amazon Redshift에서 제약 조건을 사용하는 방법에 대한 자세한 내용은 테이블 제한 사항 섹션을 참조하세요.

가능한 최소 열 크기 사용

편리하다는 이유로 최대 열 크기를 자주 사용하는 것은 좋지 않습니다.

대신 열에 저장할 가능성이 가장 큰 값을 고려하고 그에 따라 열의 크기를 조정합니다. 예를 들어, 우체국에서 사용하는 미국 주 및 영토 약어를 저장하기 위한 CHAR 열은 CHAR(2)이면 됩니다.

날짜 열의 날짜/시간 데이터 형식 사용

Amazon Redshift는 CHAR 또는 VARCHAR보다 DATE 및 TIMESTAMP 데이터를 더 효율적으로 저장하므로 쿼리 성능이 향상됩니다. 필요한 해상도에 따라 다르겠지만 날짜/시간 정보를 저장할 때는 문자 형식보다는 DATE 또는 TIMESTAMP 데이터 형식을 사용하십시오. 자세한 내용은 날짜/시간 형식 단원을 참조하십시오.

데이터 로드에 대한 Amazon Redshift 모범 사례

대용량 데이터 세트를 로드하려면 오랜 시간이 걸릴 뿐만 아니라 컴퓨팅 리소스를 많이 소비할 수 있습니다. 또한 데이터의 로딩 방식은 쿼리 성능에도 영향을 미치기도 합니다. 이번 단원에서는 COPY 명령, 대량 삽입, 스테이징 테이블을 사용하여 데이터를 효율적으로 로드하기 위한 모범 사례에 대해서 살펴보겠습니다.

주제

- 자습서를 통해 데이터를 로드하는 방법 알아보기
- COPY 명령을 사용하여 데이터 로드
- 단일 COPY 명령을 사용하여 여러 파일에서 로드
- 파일에서 데이터 로드

- 데이터 파일 압축
- 로드 전후의 데이터 파일 확인
- 다중 행 삽입 사용
- 대량 삽입 사용
- 정렬 키 순서로 데이터 로드
- 순차적 블록으로 데이터 로드
- 시계열 테이블 사용
- 유지 관리 기간 예약

자습서를 통해 데이터를 로드하는 방법 알아보기

튜토리얼: Amazon S3에서 데이터 로드은 데이터를 Amazon S3 버킷에 업로드하고 COPY 명령을 사용하여 데이터를 테이블에로드하는 단계를 시작합니다. 또한 로드 문제 해결에 필요한 도움말도 포함되어 있으며, 그 밖에 단일 파일에서 로드할 때와 다중 파일에서 로드할 때의 성능 차이를 비교합니다.

COPY 명령을 사용하여 데이터 로드

COPY 명령은 원격 호스트의 Amazon S3, Amazon EMR, Amazon DynamoDB 또는 여러 데이터 원본에서 병렬로 데이터를 로드합니다. 또한 INSERT 문을 사용할 때보다 대용량 데이터를 로드하는 효율이 훨씬 클 뿐만 아니라 더욱 효과적으로 데이터를 저장할 수도 있습니다.

COPY 명령 사용에 대한 자세한 내용은 Amazon S3에서 데이터 로드 및 Amazon DynamoDB 테이블에서 데이터 로드 섹션을 참조하세요.

단일 COPY 명령을 사용하여 여러 파일에서 로드

Amazon Redshift는 여러 압축 데이터 파일에서 병렬로 자동 로드할 수 있습니다. Amazon S3 객체 접두사를 사용하거나 매니페스트 파일을 사용하여 로드할 파일을 지정할 수 있습니다.

그러나 동시에 다수의 COPY 명령을 사용하여 여러 파일에서 테이블 하나를 로드하면 Amazon Redshift가 강제로 직렬화 로딩을 실행합니다. 이러한 유형의 로드는 속도가 훨씬 더 느리며 테이블에 정렬 열이 정의되어 있는 경우 끝에 VACUUM 프로세스가 필요합니다. COPY를 사용하여 데이터를 병렬 방식으로 로드하는 방법에 대한 자세한 내용은 Amazon S3에서 데이터 로드 섹션을 참조하세요.

파일에서 데이터 로드

소스 데이터 파일은 다양한 형식으로 제공되며 다양한 압축 알고리즘을 사용합니다. COPY 명령을 사용하여 데이터를 로드하는 경우 Amazon Redshift는 Amazon S3 버킷 접두사가 참조하는 모든 파일을

로드합니다. (접두사는 객체 키 이름의 시작 부분에 있는 문자열입니다.) 접두사가 여러 파일이나 분할할 수 있는 파일을 가리키는 경우 Amazon Redshift는 Amazon Redshift의 MPP 아키텍처를 활용하여 데이터를 병렬로 로드합니다. 이렇게 하면 워크로드가 클러스터 내 노드로 분할됩니다. 반면 분할할 수 없는 파일에서 데이터를 로드하면 Amazon Redshift는 훨씬 느린 직렬화된 로드를 수행해야 합니다. 다음 섹션에서는 다양한 파일 유형을 형식과 압축에 맞게 Amazon Redshift에 로드하는 데 권장되는 방법을 설명합니다.

분할할 수 있는 파일에서 데이터 로드

다음 파일은 데이터가 로드될 때 자동으로 분할될 수 있습니다.

- 압축되지 않은 CSV 파일
- 컬럼 파일(Parquet/ORC)

Amazon Redshift는 128MB 이상의 파일을 자동으로 청크로 분할합니다. 컬럼 형식 파일, 특히 Parquet와 ORC는 128MB 미만이면 분할되지 않습니다. Redshift는 병렬로 작동하는 슬라이스를 사용하여 데이터를 로드합니다. 이는 빠른 로드 성능을 제공합니다.

분할할 수 있는 파일에서 데이터 로드

JSON이나 CSV 같은 파일 유형은 GZIP과 같은 다른 압축 알고리즘으로 압축된 경우 자동으로 분할되지 않습니다. 이러한 경우에는 압축하면 크기가 거의 비슷해지는(1MB~1GB) 작은 파일로 데이터를 수동으로 분할하는 것이 좋습니다. 또한 파일 수가 클러스터 조각 수의 승수인지 확인해야 합니다. 데이터를 여러 파일로 분할하는 자세한 방법과 COPY를 사용하여 데이터를 로드하는 예제는 Amazon S3에서 데이털 로드 섹션을 참조하세요.

데이터 파일 압축

큰 로드 파일을 압축하려면 gzip, lzop, bzip2 또는 Zstandard를 사용하여 압축하고 데이터를 여러 개의 작은 파일로 분할하는 것이 좋습니다.

GZIP, LZOP, BZIP2 또는 ZSTD 옵션은 COPY 명령으로 지정합니다. 아래 예에서는 파이프("|")로 구분된 lzop 파일에서 TIME 테이블을 로드합니다.

```
copy time
from 's3://amzn-s3-demo-bucket/data/timerows.lzo'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
lzop
delimiter '|';
```

압축되지 않은 데이터 파일을 분할할 필요가 없는 경우가 있습니다. 데이터를 분할하는 방법과 COPY를 사용하여 데이터를 로드하는 예에 대한 자세한 내용은 Amazon S3에서 데이터 로드 섹션을 참조하세요.

로드 전후의 데이터 파일 확인

Amazon S3에서 데이터를 로드하기 전에 먼저 Amazon S3 버킷에 올바른 파일이 모두 포함되어 있는지 및 해당 파일만 포함되어 있는지 확인합니다. 자세한 내용은 버킷에 올바른 파일이 존재하는지 확인 단원을 참조하십시오.

로드 작업이 완료된 후 STL_LOAD_COMMITS 시스템 테이블을 쿼리하여 예상되는 파일이 로드되었는지 확인합니다. 자세한 내용은 데이터가 올바로 로드되었는지 확인 단원을 참조하십시오.

다중 행 삽입 사용

COPY 명령이 옵션이 아니고, SQL 삽입이 필요한 경우에는 가능하다면 다중 행 삽입을 사용하십시오. 한 번에 1개 또는 소수의 행에 데이터를 추가할 때는 데이터 압축이 비효율적입니다.

이때는 연속된 삽입을 일괄 처리하는 다중 행 삽입으로 성능을 높일 수 있습니다. 다음은 INSERT 문만 사용하여 3개의 행을 4열 테이블로 삽입하는 예로서 다중 행 삽입 구문을 간략히 보여주기 위한 간단한 삽입 구문입니다.

```
insert into category_stage values
(default, default, default, default),
(20, default, 'Country', default),
(21, 'Concerts', 'Rock', default);
```

세부 정보와 예는 INSERT 섹션을 참조하세요.

대량 삽입 사용

대량 삽입 작업을 SELECT 절에 사용하여 데이터 삽입 성능을 높일 수 있습니다.

한 테이블에서 다른 테이블로 데이터 또는 하위 데이터 세트를 이동시켜야 하는 경우에는 INSERT 및 CREATE TABLE AS 명령을 사용하십시오.

예를 들어 다음 INSERT 문은 CATEGORY 테이블에서 모든 행을 선택하여 CATEGORY_STAGE 테이블로 삽입합니다.

```
insert into category_stage
(select * from category);
```

다음은 CATEGORY_STAGE를 CATEGORY 복사본으로 생성한 후 CATEGORY의 모든 행을 CATEGORY_STAGE로 삽입하는 예입니다.

```
create table category_stage as
select * from category;
```

정렬 키 순서로 데이터 로드

정리(vacuum)할 필요가 없도록 데이터를 정렬 키 순서대로 로드합니다.

새로운 데이터의 각 배치(batch)가 테이블의 기존 행을 따르는 경우 데이터가 정렬 순서대로 저장되기 때문에 따로 VACUUM 명령을 실행할 필요가 없습니다. COPY 명령이 로드되는 데이터 배치를 각각 정렬하기 때문에 로드할 때마다 행을 미리 정렬하지 않아도 됩니다.

예를 들어 당일 작업에 따라 데이터를 매일 로드한다고 가정하겠습니다. 정렬 키가 타임스탬프 열인 경우 데이터는 정렬 순서로 저장됩니다. 이 순서가 발생하는 이유는 당일 데이터가 항상 전일 데이터의 끝에 추가되기 때문입니다. 자세한 내용은 정렬 키 순서로 데이터 로드 단원을 참조하십시오. Vacuum 작업에 대한 자세한 내용은 테이블 Vacuum 섹션을 참조하세요.

순차적 블록으로 데이터 로드

대용량의 데이터를 추가해야 하는 경우에는 정렬 순서에 따라 순차적 블록으로 데이터를 로드하여 정리 요건을 제거합니다.

예를 들어 2017년 1월부터 2017년 12월까지 이벤트가 포함된 테이블을 로드한다고 가정하겠습니다. 각 월이 단일 파일에 있다고 가정하면 1월, 2월 등의 순서로 행을 로드합니다. 그러면 행의 로드가 완료된 후에도 테이블이 완벽하게 정렬되어 따로 정리할 필요가 없습니다. 자세한 내용은 시계열 테이블 사용 단원을 참조하십시오.

대용량의 데이터 세트를 로드할 때는 정렬에 필요한 공간이 사용 가능한 전체 공간보다 클 수 있습니다. 이 경우 더 작은 블록으로 데이터를 로드하면 개별 로드에 대한 중간 정렬 공간이 크게 줄어듭니다. 또한 COPY가 실패하여 롤백되는 경우에도 더 작은 블록을 로드하면 더 쉽게 다시 시작할 수 있습니다.

시계열 테이블 사용

데이터의 보존 기간이 고정되어 있는 경우에는 데이터를 시계열 테이블의 시퀀스로 구성할 수 있습니다. 이러한 시퀀스에서 각 테이블은 동일하지만 다른 시간 범위의 데이터를 포함합니다.

해당 테이블에서 간단히 DROP TABLE 명령을 실행하여 기존 데이터를 쉽게 제거할 수 있습니다. 이 접근 방식은 대규모로 DELETE 프로세스를 실행하는 것보다 속도가 빠르며 공간 회수를 위해 이후에 VACUUM 프로세스를 실행할 필요기 없습니다. 데이터가 다른 테이블에 저장된다는 사실을 감추기 위

해 UNION ALL 뷰를 생성할 수 있습니다. 이전 데이터를 삭제할 때 UNION ALL 뷰를 조정하면 삭제된 테이블이 제거됩니다. 마찬가지로 새로운 기간을 새로운 테이블에 로드할 때도 새로운 테이블을 뷰에 추가합니다. 최적화 프로그램에 쿼리 필터와 일치하지 않는 테이블의 스캔을 건너뛰도록 신호를 보내기 위해 뷰 정의에서 각 테이블에 해당되는 날짜 범위에 대해 필터링합니다.

UNION ALL 뷰에 테이블 수가 너무 많지 않도록 하십시오. 각 추가 테이블은 쿼리에 작은 처리 시간을 추가합니다. 테이블에서 동일한 기간을 사용할 필요가 없습니다. 예를 들어 매일, 매달 및 매년 등 기간이 서로 다른 테이블이 있을 수 있습니다.

타임스탬프 열이 있는 시계열 테이블을 정렬 키에 사용할 경우 데이터를 정렬 키 순서로 효과적으로 로드할 수 있습니다. 그러면 VACUUM을 실행하여 데이터를 다시 정렬할 필요가 없습니다. 자세한 내용은 정렬 키 순서로 데이터 로드 단원을 참조하십시오.

유지 관리 기간 예약

쿼리 실행 도중 예정된 유지 관리가 실행되면 쿼리는 종료 후 롤백되고 나중에 다시 시작해야 합니다. 따라서 대용량 데이터 로드나 VACUUM 같이 시간이 오래 걸리는 작업은 유지 관리 기간을 피해서 예약하는 것이 좋습니다. 또한 증분 값을 더욱 작게 하여 데이터를 로드하고 VACUUM 작업의 크기를 관리하면 위험도 최소화할 뿐만 아니라 필요한 경우 시스템 재시작도 더욱 쉬워질 수 있습니다. 자세한 내용은 순차적 블록으로 데이터 로드 및 테이블 Vacuum 섹션을 참조하세요.

Amazon Redshift 쿼리 설계 모범 사례

쿼리 성능을 극대화하려면 쿼리 생성 시 다음 권장 사항을 따르십시오.

- 쿼리 성능을 높일 수 있는 탄탄한 기반을 완성하려면 모범 사례에 따라 테이블을 설계하십시오. 자세한 내용은 Amazon Redshift 테이블 설계 모범 사례 단원을 참조하십시오.
- `select *` 사용은 피하십시오. 그리고 반드시 필요한 열만 추가하십시오.
- 동일한 테이블에서 반복적으로 선택하지 말고 CASE 조건식을 사용하여 복합적인 집계를 실행하십시오.
- 절대적으로 필요한 경우가 아니라면 크로스 조인을 사용하지 마십시오. 이 조인 방법은 조인 조건이 없기 때문에 두 테이블의 데카르트 곱이 발생하는 원인이 됩니다. 크로스 조인은 일반적으로 중첩 루프 조인으로 실행되기 때문에 가능한 조인 유형 중에서 속도가 가장 느립니다.
- 쿼리에서 한 테이블만 조건자 조건에 사용되는 경우에는 하위 쿼리를 사용하십시오. 그러면 하위 쿼리가 소수의 행(약 200개 미만)을 반환합니다. 다음은 하위 쿼리를 사용하여 LISTING 테이블을 조인하지 않는 예입니다.

```
select sum(sales.qtysold)
```

```
from sales
where salesid in (select listid from listing where listtime > '2008-12-26');
```

- 조건자를 사용하여 최대한 많은 데이터 세트를 제한하십시오.
- 술어에서 할 수 있는 가장 비용이 적게 드는 연산자를 사용하십시오. 비교 조건 연산자는 LIKE 연산자보다 바람직합니다. 그리고 LIKE 연산자는 SIMILAR TO 또는 POSIX 연산자보다 효과적입니다.
- 쿼리 조건자에서 함수 사용은 자제하십시오. 함수를 사용하면 쿼리의 중간 단계 해결을 위해 엄청난 수의 행이 필요하기 때문에 쿼리 비용이 더욱 커질 수 있습니다.
- 가능하다면 WHERE 절을 사용하여 데이터 세트를 제한하십시오. 그러면 쿼리 플래너가 기준과 일치하는 레코드를 결정하는 데 행 순서를 사용할 수 있기 때문에 엄청난 수의 디스크 블록을 스캔하는 단계를 건너뛸 수 있습니다. 이렇게 하지 않으면 쿼리 실행 엔진이 포함되어 있는 열을 모두 스캔해야 합니다.
- 동일한 필터가 적용되는 조건자라고 해도 조건자를 추가하여 조인에 참여하는 테이블을 필터링하십시오. 쿼리는 동일한 결과 집합을 반환하지만 Amazon Redshift는 스캔 단계 전에 조인 테이블을 필터링 할 수 있으므로 이러한 테이블에서 검색 블록을 효율적으로 건너 뛸 수 있습니다. 조인 조건에 사용되는 열을 기준으로 필터링하는 경우에는 중복 필터가 필요하지 않습니다.

  예를 들어 SALES 테이블과 LISTING 테이블을 조인하여 12월 이후 판매자별로 목록에 나열된 티켓의 판매량을 찾으려고 한다고 가정하겠습니다. 그리고 두 테이블은 날짜를 기준으로 정렬되어 있습니다. 다음 쿼리는 공통 키를 통해 두 테이블을 조인한 후 12월 1일보다 더 큰 `listing.listtime` 값을 필터링합니다.

```
select listing.sellerid, sum(sales.qtysold)
from sales, listing
where sales.salesid = listing.listid
and listing.listtime > '2008-12-01'
group by 1 order by 1;
```

  WHERE 절에는 `sales.saletime` 조건자가 없기 때문에 실행 엔진이 전체 SALES 테이블을 강제로 스캔해야 합니다. 이때 필터링을 통해 조인에 참여하는 행의 수가 줄어든다는 사실을 알고 있다면 필터 역시 추가할 수 있습니다. 다음은 실행 시간을 크게 줄일 수 있는 예입니다.

```
select listing.sellerid, sum(sales.qtysold)
from sales, listing
where sales.salesid = listing.listid
and listing.listtime > '2008-12-01'
and sales.saletime > '2008-12-01'
group by 1 order by 1;
```

- 쿼리 플래너의 집계 효율을 높일 수 있도록 GROUP BY 절에 정렬 키를 사용하십시오. GROUP BY 목록에 정렬 키 열만 포함되어 있으면(이중 하나는 분산 키임) 쿼리 실행 시 1단계 집계가 가능합니다. GROUP BY 목록의 정렬 키 열에는 1차 정렬 키를, 그리고 정렬 키 순서에 사용할 다른 정렬 키를 추가해야 합니다. 예를 들어 1차 정렬 키, 1차/2차 정렬 키, 1차/2차/3차 정렬 키 등을 사용하는 것이 유효합니다. 1차/3차 정렬 키를 사용하는 것은 유효하지 않습니다.

  1단계 집계의 사용 여부는 EXPLAIN 명령을 실행한 후 쿼리의 집계 단계에서 `XN GroupAggregate`를 찾아보면 확인할 수 있습니다.
- GROUP BY 절과 ORDER BY 절을 동시에 사용하는 경우에는 두 절 모두 동일한 순서대로 열을 삽입해야 합니다. 즉, 다음과 같은 접근법을 사용하십시오.

```
group by a, b, c
order by a, b, c
```

  다음과 같은 접근법을 사용하지 마십시오.

```
group by b, c, a
order by a, b, c
```

Amazon Redshift Advisor의 권장 사항 준수

Amazon Redshift 클러스터의 성능을 향상시키고 운영 비용을 줄이기 위해 Amazon Redshift Advisor는 변경 사항에 대한 구체적인 권장 사항을 제공합니다. Advisor는 클러스터의 성능 및 사용 지표를 분석하여 사용자 지정 권장 사항을 개발합니다. 이러한 맞춤형 권장 사항은 작업 및 클러스터 설정에 관련됩니다. 최적화를 우선적으로 처리할 수 있도록 Advisor는 영향력 순서로 권장 사항의 순위를 설정합니다.

Advisor는 성능 통계 및 작업 데이터에 관한 관찰을 기반으로 권장 사항을 작성합니다. Advisor는 클러스터에서 테스트를 실행하고 테스트 값이 지정된 범위 내에 속하는지 여부를 결정하여 관찰을 개발합니다. 테스트 결과가 해당 범위를 벗어나면 Advisor는 클러스터에 대한 관찰을 생성합니다. 동시에 Advisor는 관찰된 값을 모범 사례 범위로 되돌리는 방법에 대한 권장사항을 생성합니다. Advisor는 성능과 작업에 상당한 영향을 미치는 권장 사항만 표시합니다. Advisor가 권장 사항이 처리되었다고 결정하면 권장 사항 목록에서 해당 권장 사항을 제거합니다.

예를 들어, 데이터 웨어하우스에 압축되지 않은 테이블 열이 많이 포함되어 있다고 가정해 보겠습니다. 이러한 경우 열 압축을 지정하는`ENCODE` 파라미터를 사용하여 테이블을 다시 빌드하면 클러스터 스토리지 비용을 절약할 수 있습니다. 다른 예로, Advisor가 클러스터의 압축되지 않은 테이블 데이터에 상

당한 양의 데이터가 포함되어 있는 것을 확인했다고 가정해 보겠습니다. 이러한 경우 Advisor는 압축의 후보가 되는 테이블 열을 찾기 위한 SQL 코드 블록과 이러한 열을 압축하는 방법을 설명하는 리소스를 제공합니다.

Advisor가 지원되는 Amazon Redshift 리전

Amazon Redshift Advisor 기능을 사용할 수 있는 AWS 리전은 다음으로 제한됩니다.

- 미국 동부(버지니아 북부) 리전(us-east-1)
- 미국 동부(오하이오) 리전(us-east-2)
- 미국 서부(캘리포니아 북부) 리전(us-west-1)
- 미국 서부(오레곤) 리전(us-west-2)
- 아프리카(케이프타운) 리전(af-south-1)
- 아시아 태평양(홍콩) 리전(ap-east-1)
- 아시아 태평양(하이데라바드) 리전(ap-south-2)
- 아시아 태평양(자카르타) 리전(ap-southeast-3)
- 아시아 태평양(멜버른) 리전(ap-southeast-4)
- 아시아 태평양(뭄바이) 리전(ap-south-1)
- 아시아 태평양(오사카) 리전(ap-northeast-3)
- 아시아 태평양(서울) 리전(ap-northeast-2)
- 아시아 태평양(싱가포르) 리전(ap-southeast-1)
- 아시아 태평양(시드니) 리전(ap-southeast-2)
- 아시아 태평양(도쿄) 리전(ap-northeast-1)
- 캐나다(중부) 리전(ca-central-1)
- 캐나다 서부(캘거리) 리전(ca-west-1)
- 중국(베이징) 리전(cn-north-1)
- 중국(닝샤) 리전(cn-northwest-1)
- 유럽(프랑크푸르트) 리전(eu-central-1)
- 유럽(아일랜드) 리전(eu-west-1)
- 유럽(런던) 리전(eu-west-2)
- 유럽(밀라노) 리전(eu-south-1)
- 유럽(파리) 리전(eu-west-3)

- 유럽(스페인) 리전(eu-south-2)
- 유럽(스톡홀름) 리전(eu-north-1)
- 유럽(취리히) 리전(eu-central-2)
- 이스라엘(텔아비브) 리전(il-central-1)
- 중동(바레인) 리전(me-south-1)
- 중동(UAE) 리전(me-central-1)
- 남아메리카(상파울루) 리전(sa-east-1)

주제

Amazon Redshift Advisor 권장 사항 보기

Amazon Redshift 콘솔, Amazon Redshift API 또는 AWS CLI를 사용하여 Amazon Redshift Advisor 권장 사항에 액세스할 수 있습니다. 권장 사항에 액세스하려면 IAM 역할 또는 자격 증명에 `redshift:ListRecommendations` 권한이 연결되어 있어야 합니다.

Amazon Redshift 프로비저닝 콘솔에서 Amazon Redshift Advisor 권장 사항 보기

AWS Management Console에서 Amazon Redshift Advisor 권장 사항을 볼 수 있습니다.

콘솔에서 Amazon Redshift 클러스터에 대한 Amazon Redshift Advisor 권장 사항을 보려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 Advisor를 선택합니다.
3. 각 권장 사항을 확장하여 추가 세부 정보를 확인합니다. 이 페이지에서 권장 사항을 정렬하고 그룹화할 수 있습니다.

Amazon Redshift API 작업을 사용하여 Amazon Redshift Advisor 권장 사항 보기

Amazon Redshift API를 사용하여 Amazon Redshift 클러스터에 대한 Amazon Redshift Advisor 권장 사항을 나열할 수 있습니다. 일반적으로 원하는 프로그래밍 언어로 애플리케이션을 개발하여 AWS

SDK를 사용해 `redshift:ListRecommendations` API를 직접적으로 호출합니다. 자세한 내용은 Amazon Redshift API 참조에서 ListRecommendations를 참조하세요.

AWS Command Line Interface 작업을 사용하여 Amazon Redshift Advisor 권장 사항 보기

AWS Command Line Interface를 사용하여 Amazon Redshift 클러스터에 대한 Amazon Redshift Advisor 권장 사항을 나열할 수 있습니다. 자세한 내용은 AWS CLI 명령 참조에서 list-recommendations를 참조하세요.

Amazon Redshift Advisor 권장 사항

Amazon Redshift Advisor는 성능을 향상시키고 운영 비용을 절감하기 위해 Amazon Redshift 클러스터를 최적화하는 방법에 대한 권장 사항을 제공합니다. 위의 설명과 같이 콘솔에서 각 권장 사항에 대한 설명을 찾아볼 수 있습니다. 다음 단원에서 이러한 권장 사항에 대한 추가 세부 정보를 찾아볼 수 있습니다.

주제

- COPY를 통해 로드되는 Amazon S3 파일 객체 압축
- 여러 활성 데이터베이스 격리
- 워크로드 관리(WLM) 메모리 다시 할당
- COPY 중 압축 분석 건너뛰기
- COPY로 로드되는 Amazon S3 객체 분할
- 테이블 통계 업데이트
- 단기 쿼리 가속화 활성화
- 테이블의 분산 키 변경
- 테이블의 정렬 키 변경
- 열의 압축 인코딩 변경
- 데이터 형식 권장 사항

COPY를 통해 로드되는 Amazon S3 파일 객체 압축

COPY 명령은 Amazon Redshift의 대량 병렬 처리(MPP) 아키텍처를 이용하여 병렬로 데이터를 읽고 로드합니다. Amazon S3, DynamoDB 테이블 및 하나 이상의 원격 호스트의 텍스트 출력에서 파일을 읽을 수 있습니다.

많은 양의 데이터를 로드할 때는 COPY 명령을 사용하여 S3에서 압축된 데이터 파일을 로드하는 것이 좋습니다. 큰 데이터 세트를 압축하면 Amazon S3에 파일을 업로드하는 시간이 단축됩니다. 또한 COPY가 파일을 읽으면서 압축을 해제해 로드 프로세스를 단축할 수 있습니다.

분석

압축되지 않은 큰 데이터 세트를 로드하는 COPY 명령을 장기 실행하면 성능을 대폭 향상할 기회가 있습니다. Advisor 분석은 압축되지 않은 큰 데이터 세트를 로드하는 COPY 명령을 식별합니다. 이러한 경우 Advisor는 Amazon S3에서 소스 파일에 압축을 구현하라는 권장 사항을 생성합니다.

권장 사항

상당한 양의 데이터를 로드하거나 상당한 기간 동안 실행하는 각 COPY가 Amazon S3에서 압축되지 않은 데이터 객체를 수집하도록 합니다. 수퍼유저 권한으로 다음 SQL 명령을 실행하면 Amazon S3에서 압축되지 않은 큰 데이터 세트를 로드하는 COPY 명령을 식별할 수 있습니다.

```
SELECT
    wq.userid, query, exec_start_time AS starttime, COUNT(*) num_files,
    ROUND(MAX(wq.total_exec_time/1000000.0),2) execution_secs,
    ROUND(SUM(transfer_size)/(1024.0*1024.0),2) total_mb,
    SUBSTRING(querytxt,1,60) copy_sql
FROM stl_s3client s
JOIN stl_query q USING (query)
JOIN stl_wlm_query wq USING (query)
WHERE s.userid>1 AND http_method = 'GET'
    AND POSITION('COPY ANALYZE' IN querytxt) = 0
    AND aborted = 0 AND final_state='Completed'
GROUP BY 1, 2, 3, 7
HAVING SUM(transfer_size) = SUM(data_size)
AND SUM(transfer_size)/(1024*1024) >= 5
ORDER BY 6 DESC, 5 DESC;
```

스테이징된 데이터가 로드한 후에도 Amazon S3에 그대로 남아 있는 경우(데이터 레이크 아키텍처에서 일반적으로 발생함) 이 데이터를 압축된 형식으로 저장하면 스토리지 비용을 줄일 수 있습니다.

구현 팁

- 이상적인 객체 크기는 압축 후 1–128MB입니다.
- gzip, lzop 또는 bzip2 형식을 사용하여 파일을 압축할 수 있습니다.

여러 활성 데이터베이스 격리

모범 사례로, Amazon Redshift의 데이터베이스를 서로 격리하는 것이 좋습니다. 쿼리는 특정 데이터베이스에서 실행되며 클러스터에 있는 다른 데이터베이스의 데이터에 액세스할 수 없습니다. 하지만 클러스터의 모든 데이터베이스에서 실행하는 쿼리는 동일한 기본 클러스터 스토리지 공간을 공유하고 리소스를 컴퓨팅합니다. 단일 클러스터에 여러 활성 데이터베이스가 포함되는 경우 각 워크로드는 일반적으로 서로 관련되지 않습니다.

분석

Advisor 분석은 클러스터에 있는 모든 데이터베이스를 검토하여 동시에 실행되는 활성 워크로드를 확인합니다. 동시에 실행되는 활성 워크로드가 있는 경우 Advisor는 데이터베이스를 별도의 Amazon Redshift 클러스터로 마이그레이션하는 것이 좋다는 권장 사항을 생성합니다.

권장 사항

활성으로 쿼리되는 각 데이터베이스를 별도의 전용 클러스터로 이동하는 것이 좋습니다. 별도의 클러스터를 사용하면 리소스 경합을 줄이고 쿼리 성능을 향상할 수 있습니다. 그렇게 하면 각 워크로드의 스토리지, 비용 및 성능 필요에 맞게 각 클러스터의 크기를 설정할 수 있기 때문입니다. 또한 관련되지 않은 워크로드는 흔히 다른 워크로드 관리 구성에서 이점을 얻을 수 있습니다.

데이터베이스가 활성으로 사용되는지 확인하려면 수퍼유저 권한으로 다음 SQL 명령을 실행할 수 있습니다.

```
SELECT database,
  COUNT(*) as num_queries,
  AVG(DATEDIFF(sec,starttime,endtime)) avg_duration,
  MIN(starttime) as oldest_ts,
  MAX(endtime) as latest_ts
FROM stl_query
WHERE userid > 1
GROUP BY database;
```

구현 팁

- 사용자는 각 데이터베이스에 특정하게 연결해야 하고 쿼리는 단일 데이터베이스에만 액세스할 수 있기 때문에 데이터베이스를 별도의 클러스터로 이동해도 사용자에게는 미치는 영향은 최소한입니다.

- 데이터베이스를 이동하는 한 가지 옵션은 다음 단계를 수행하는 것입니다.
  1. 현재 클러스터의 스냅샷을 동일한 크기의 클러스터로 임시로 복원합니다.
  2. 이동할 대상 데이터베이스를 제외한 모든 데이터베이스를 새 클러스터에서 삭제합니다.
  3. 데이터베이스의 워크로드에 적합한 노드 유형 및 개수로 클러스터를 크기 조정합니다.

워크로드 관리(WLM) 메모리 다시 할당

Amazon Redshift는 사용자 쿼리를 수동 WLM 구현로 라우팅하여 처리합니다. 이러한 쿼리의 대기열 라우팅 방식을 정의하는 것이 바로 워크로드 관리(WLM)입니다. Amazon Redshift는 클러스터의 사용 가능한 메모리 중 일부를 각 대기열에 할당합니다. 이렇게 할당된 대기열의 메모리는 다시 쿼리 슬롯으로 분할됩니다.

워크로드에 필요한 것보다 더 많은 슬롯으로 대기열이 구성되면 이러한 사용되지 않은 슬롯에 할당된 메모리는 과소 사용 상태가 됩니다. 구성된 슬롯을 피크 워크로드 요구 사항에 맞게 줄이면 과소 사용 상태의 메모리가 활성 슬롯에 다시 배포되므로 쿼리 성능을 향상할 수 있습니다.

분석

Advisor 분석은 워크로드 동시성 요구 사항을 검토하여 사용되지 않는 슬롯이 있는 쿼리 대기열을 식별합니다. 다음을 발견하면 Advisor는 대기열의 슬롯 수를 줄이기 위한 권장 사항을 생성합니다.

- 분석하는 동안 완전히 비활성인 슬롯이 있는 대기열
- 분석하는 동안 최소 2개의 비활성 슬롯이 포함된 4개 이상의 슬롯이 있는 대기열

권장 사항

구성된 슬롯을 피크 워크로드 요구 사항에 맞게 줄이면 과소 사용 상태의 메모리가 활성 슬롯에 다시 배포됩니다. 슬롯이 완전히 사용된 적이 없는 대기열의 경우 구성된 슬롯 개수를 줄이는 것이 좋습니다. 이러한 대기열을 식별하려면 수퍼유저 권한으로 다음 SQL 명령을 실행하여 각 대기열의 피크 시간별 슬롯 요구 사항을 비교할 수 있습니다.

```
WITH
 generate_dt_series AS (select sysdate - (n * interval '5 second') as dt from (select
 row_number() over () as n from stl_scan limit 17280)),
 apex AS (
     SELECT iq.dt, iq.service_class, iq.num_query_tasks, count(iq.slot_count) as
 service_class_queries, sum(iq.slot_count) as service_class_slots
```

```
    FROM
        (select gds.dt, wq.service_class, wscc.num_query_tasks, wq.slot_count
        FROM stl_wlm_query wq
        JOIN stv_wlm_service_class_config wscc ON (wscc.service_class =
 wq.service_class AND wscc.service_class > 5)
        JOIN generate_dt_series gds ON (wq.service_class_start_time <= gds.dt AND
 wq.service_class_end_time > gds.dt)
        WHERE wq.userid > 1 AND wq.service_class > 5) iq
    GROUP BY iq.dt, iq.service_class, iq.num_query_tasks),
    maxes as (SELECT apex.service_class, trunc(apex.dt) as d, date_part(h,apex.dt) as
 dt_h, max(service_class_slots) max_service_class_slots
                    from apex group by apex.service_class, apex.dt,
 date_part(h,apex.dt))
SELECT apex.service_class - 5 AS queue, apex.service_class, apex.num_query_tasks AS
 max_wlm_concurrency, maxes.d AS day, maxes.dt_h || ':00 - ' || maxes.dt_h || ':59' as
 hour, MAX(apex.service_class_slots) as max_service_class_slots
FROM apex
JOIN maxes ON (apex.service_class = maxes.service_class AND apex.service_class_slots =
 maxes.max_service_class_slots)
GROUP BY  apex.service_class, apex.num_query_tasks, maxes.d, maxes.dt_h
ORDER BY apex.service_class, maxes.d, maxes.dt_h;
```

`max_service_class_slots` 열은 해당 시간 동안 쿼리 대기열에 있는 최대 WLM 쿼리 슬롯 수를 나타냅니다. 사용률이 낮은 대기열이 있는 경우 Amazon Redshift 관리 가이드에 설명된 대로 파라미터 그룹을 수정하여 슬롯 감소 최적화를 구현합니다.

구현 팁

- 워크로드의 볼륨이 매우 가변적인 경우 분석에서 피크 사용률 기간이 포착되었는지 확인합니다. 그렇지 않은 경우 이전의 SQL을 반복적으로 실행하여 피크 동시성 요구 사항을 모니터링합니다.
- 위의 SQL 코드에서 쿼리 결과를 해석하는 방법에 대한 자세한 내용은 GitHub의 wlm_apex_hourly.sql 스크립트 를 참조하십시오.

COPY 중 압축 분석 건너뛰기

COPY 명령으로 선언된 압축 인코딩을 사용하여 데이터를 빈 테이블로 로드하면 Amazon Redshift는 스토리지 압축을 적용합니다. 이 최적화를 사용하면 최종 사용자가 로드하더라도 클러스터의 데이터가 효율적으로 저장됩니다. 압축을 적용하기 위해 필요한 분석에는 상당한 시간이 소요될 수 있습니다.

분석

Advisor 분석은 자동 압축 분석에서 지연된 COPY 작업을 확인합니다. 이 분석은 로드되는 동안 데이터를 샘플링하여 압축 인코딩을 결정합니다. 이 샘플링은 ANALYZE COMPRESSION 명령이 수행하는 것과 비슷합니다.

야간 추출, 변환, 로드(ETL) 배치와 같이 데이터를 구조적 프로세스의 일부로 로드하는 경우 압축을 미리 정의할 수 있습니다. 또한 테이블 정의를 최적화하여 부정적인 영향 없이 이 단계를 영구적으로 건너뛸 수도 있습니다.

권장 사항

압축 분석 단계를 건너뛰어서 COPY 응답성을 향상하려면 다음 두 가지 옵션 중 하나를 구현합니다.

- COPY 명령을 사용하여 로드하는 테이블을 생성할 때 열 ENCODE 파라미터를 사용합니다.
- COPY 명령에서 COMPUPDATE OFF 파라미터를 제공하여 압축을 완전히 해제합니다.

가장 좋은 솔루션은 일반적으로 테이블 생성 중에 열 인코딩을 사용하는 것입니다. 이 접근 방식을 사용하면 압축된 데이터를 디스크에 저장하는 장점을 그대로 유지할 수 있기 때문입니다. ANALYZE COMPRESSION 명령을 사용하여 압축 인코딩을 제안할 수 있지만, 이러한 인코딩을 적용하려면 테이블을 다시 생성해야 합니다. 이 프로세스를 자동화하려면 GitHub에 있는 AWSColumnEncodingUtility를 사용합니다.

자동 압축 분석을 트리거한 최신 COPY 작업을 식별하려면 다음 SQL 명령을 실행합니다.

```
  WITH xids AS (
    SELECT xid FROM stl_query WHERE userid>1 AND aborted=0
    AND querytxt = 'analyze compression phase 1' GROUP BY xid
    INTERSECT SELECT xid FROM stl_commit_stats WHERE node=-1)
SELECT a.userid, a.query, a.xid, a.starttime, b.complyze_sec,
    a.copy_sec, a.copy_sql
FROM (SELECT q.userid, q.query, q.xid, date_trunc('s',q.starttime)
    starttime, substring(querytxt,1,100) as copy_sql,
    ROUND(datediff(ms,starttime,endtime)::numeric / 1000.0, 2) copy_sec
    FROM stl_query q JOIN xids USING (xid)
    WHERE (querytxt ilike 'copy %from%' OR querytxt ilike '% copy %from%')
    AND querytxt not like 'COPY ANALYZE %') a
LEFT JOIN (SELECT xid,
    ROUND(sum(datediff(ms,starttime,endtime))::numeric / 1000.0,2) complyze_sec
    FROM stl_query q JOIN xids USING (xid)
    WHERE (querytxt like 'COPY ANALYZE %'
    OR querytxt like 'analyze compression phase %')
```

```
    GROUP BY xid ) b ON a.xid = b.xid
WHERE b.complyze_sec IS NOT NULL ORDER BY a.copy_sql, a.starttime;
```

구현 팁

- ETL 프로세스 중에 생성된 상당한 크기의 모든 테이블(예: 스테이징 테이블 및 임시 테이블)이 첫 번째 정렬 키를 제외한 모든 열에 대해 압축 인코딩을 선언하는지 확인합니다.
- 이전의 SQL 명령으로 식별되는 각 COPY 명령에 대해 로드되는 테이블의 예상 수명 크기를 추정합니다. 테이블이 매우 작은 크기로 유지될 것을 확신하는 경우 `COMPUPDATE OFF` 파라미터를 사용하여 압축을 완전히 해제합니다. 그렇지 않으면 COPY 명령을 사용하여 로드하기 전에 명시적 압축을 사용하여 테이블을 생성합니다.

COPY로 로드되는 Amazon S3 객체 분할

COPY 명령은 Amazon Redshift의 대량 병렬 처리(MPP) 아키텍처를 활용하여 Amazon S3 파일의 데이터를 읽고로드합니다. COPY 명령은 여러 파일에서 데이터를 병렬 방식으로 로드하여 클러스터의 노드에게 워크로드를 분할합니다. 최적의 처리량을 달성하려면 데이터를 여러 파일로 분리하여 병렬 처리를 이용하는 것이 좋습니다.

분석

Advisor 분석은 Amazon S3에서 스테이징된 작은 수의 파일에 포함된 큰 데이터 세트를 로드하는 COPY 명령을 식별합니다. 적은 수의 파일에서 큰 데이터 세트를 로드하는 COPY 명령을 장기 실행하면 성능을 대폭 향상할 기회가 있습니다. Advisor가 이러한 COPY 명령에 상당한 시간이 걸리는 것으로 확인하면 Amazon S3에서 데이터를 추가 파일로 분할하여 병렬 처리를 증가시키기 위한 권장 사항을 생성합니다.

권장 사항

이 경우 나열된 우선 순위에 따라 다음 작업을 수행하는 것이 좋습니다.

1. 클러스터 노드 수보다 적은 파일을 로드하는 COPY 명령을 최적화합니다.
2. 클러스터 조각 수보다 적은 파일을 로드하는 COPY 명령을 최적화합니다.
3. 파일 수가 클러스터 조각 수의 배수가 아닌 COPY 명령을 최적화합니다.

특정 COPY 명령은 상당한 양의 데이터를 로드하거나 상당 기간 동안 실행됩니다. 이러한 명령의 경우 클러스터 내 조각 개수의 배수와 같은 수의 여러 데이터 객체를 Amazon S3에서 로드하는 것이 좋습니

다. 각 COPY 명령이 로드한 S3 객체 수를 확인하려면 수퍼유저 권한으로 다음 SQL 코드를 실행합니다.

```
SELECT
    query, COUNT(*) num_files,
    ROUND(MAX(wq.total_exec_time/1000000.0),2) execution_secs,
    ROUND(SUM(transfer_size)/(1024.0*1024.0),2) total_mb,
    SUBSTRING(querytxt,1,60) copy_sql
FROM stl_s3client s
JOIN stl_query q USING (query)
JOIN stl_wlm_query wq USING (query)
WHERE s.userid>1 AND http_method = 'GET'
    AND POSITION('COPY ANALYZE' IN querytxt) = 0
    AND aborted = 0 AND final_state='Completed'
GROUP BY query, querytxt
HAVING (SUM(transfer_size)/(1024*1024))/COUNT(*) >= 2
ORDER BY CASE
WHEN COUNT(*) < (SELECT max(node)+1 FROM stv_slices) THEN 1
WHEN COUNT(*) < (SELECT COUNT(*) FROM stv_slices WHERE node=0) THEN 2
ELSE 2+((COUNT(*) % (SELECT COUNT(*) FROM stv_slices))/(SELECT COUNT(*)::DECIMAL FROM
 stv_slices))
END, (SUM(transfer_size)/(1024.0*1024.0))/COUNT(*) DESC;
```

구현 팁

- 노드에 있는 조각 수는 클러스터의 노드 크기에 따라 다릅니다. 다양한 노드 유형의 슬라이스 수에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 Amazon Redshift의 노드 및 클러스터 섹션을 참조하세요.
- 집합에 공통 접두사 또는 접두사 키를 지정하거나 매니페스트 파일에서 파일을 명시적으로 나열하여 여러 파일을 로드할 수 있습니다. 파일 로드에 대한 자세한 내용은 압축 및 비압축 파일에서 데이터 로드 섹션을 참조하세요.
- Amazon Redshift는 워크로드를 분리할 때 파일 크기를 고려하지 않습니다. 이때는 압축 후 파일 크기가 1MB~1GB로 거의 같아질 수 있도록 로딩 데이터 파일을 분할합니다.

테이블 통계 업데이트

Amazon Redshift는 비용 기반 쿼리 옵티마이저를 사용하여 쿼리에 최적의 실행 계획을 선택합니다. 예상 비용은 ANALYZE 명령을 사용하여 수집된 테이블 통계를 기반으로 합니다. 통계가 오래되었거나 누락된 경우 데이터베이스는 매우 복잡한 쿼리의 쿼리 실행에 덜 효율적인 계획을 선택할 수 있습니다. 통계를 최신 상태로 유지하면 최대한 빠른 시간 내에 복잡한 쿼리를 실행하는 데 도움이 됩니다.

분석

Advisor 분석은 통계가 오래되거나 누락된 테이블을 추적합니다. 이 기능은 복잡한 쿼리와 연결된 테이블 액세스 메타데이터를 검토합니다. 복잡한 패턴으로 자주 액세스되는 테이블에 통계가 누락된 경우 Advisor는 ANALYZE를 실행하기 위한 중요 권장 사항을 생성합니다. 복잡한 패턴으로 자주 액세스되는 테이블에 오래된 통계가 있는 경우 Advisor는 ANALYZE를 실행하기 위한 제안 권장 사항을 생성합니다.

권장 사항

테이블 콘텐츠가 대폭 변경될 때마다 ANALYZE를 사용하여 통계를 업데이트합니다. COPY 또는 INSERT 명령을 통해 상당한 수의 새로운 데이터 행이 기존 테이블에 로드될 때마다 ANALYZE를 실행하는 것이 좋습니다. 또한 UPDATE 또는 DELETE 명령을 통해 상당한 수의 행이 수정될 때마다 ANALYZE를 실행하는 것이 좋습니다. 어떤 테이블에 누락되거나 오래된 통계가 있는지 확인하려면 수퍼유저 권한으로 다음 SQL 명령을 실행합니다. 결과는 가장 큰 테이블부터 가장 작은 테이블까지의 순서로 정렬됩니다.

어떤 테이블에 누락되거나 오래된 통계가 있는지 확인하려면 수퍼유저 권한으로 다음 SQL 명령을 실행합니다. 결과는 가장 큰 테이블부터 가장 작은 테이블까지의 순서로 정렬됩니다.

```
SELECT
   ti.schema||'.'||ti."table" tablename,
   ti.size table_size_mb,
   ti.stats_off statistics_accuracy
 FROM svv_table_info ti
 WHERE ti.stats_off > 5.00
 ORDER BY ti.size DESC;
```

구현 팁

기본적으로 ANALYZE 임계값은 10퍼센트로 설정됩니다. 이와 같은 기본값은 마지막 ANALYZE 실행 후 테이블에서 변경된 행이 전체 행의 10% 미만인 경우 ANALYZE 명령이 주어진 테이블을 건너뜀을

의미합니다. 결과적으로 ETL 프로세스의 끝에 ANALYZE 명령을 실행하도록 선택할 수 있습니다. 이 방법을 사용한다는 것은 ANALYZE가 종종 생략되고 ANALYZE가 필요할 때 실행된다는 것을 의미합니다.

ANALYZE 통계는 조인 시 사용되는 열(예: `JOIN tbl_a ON col_b`) 또는 조건자로 사용되는 열(예: `WHERE col_b = 'xyz'`)에 가장 많은 영향을 미칩니다. 기본적으로 ANALYZE는 지정된 테이블에 있는 모든 열에 대한 통계를 수집합니다. 필요한 경우 가장 많은 영향을 미치는 열에 대해서만 ANALYZE를 실행하여 ANALYZE를 실행하는 데 필요한 시간을 줄일 수 있습니다. 다음 SQL 명령을 실행하여 조건자로 사용되는 열을 식별할 수 있습니다. 또한 Amazon Redshift에서 `ANALYZE PREDICATE COLUMNS`를 지정하여 분석할 열을 선택할 수 있습니다.

```
WITH predicate_column_info as (
SELECT ns.nspname AS schema_name, c.relname AS table_name, a.attnum as col_num,
 a.attname as col_name,
        CASE
            WHEN 10002 = s.stakind1 THEN array_to_string(stavalues1, '||')
            WHEN 10002 = s.stakind2 THEN array_to_string(stavalues2, '||')
            WHEN 10002 = s.stakind3 THEN array_to_string(stavalues3, '||')
            WHEN 10002 = s.stakind4 THEN array_to_string(stavalues4, '||')
            ELSE NULL::varchar
        END AS pred_ts
   FROM pg_statistic s
   JOIN pg_class c ON c.oid = s.starelid
   JOIN pg_namespace ns ON c.relnamespace = ns.oid
   JOIN pg_attribute a ON c.oid = a.attrelid AND a.attnum = s.staattnum)
SELECT schema_name, table_name, col_num, col_name,
       pred_ts NOT LIKE '2000-01-01%' AS is_predicate,
       CASE WHEN pred_ts NOT LIKE '2000-01-01%' THEN (split_part(pred_ts,
 '||',1))::timestamp ELSE NULL::timestamp END as first_predicate_use,
       CASE WHEN pred_ts NOT LIKE '%||2000-01-01%' THEN (split_part(pred_ts,
 '||',2))::timestamp ELSE NULL::timestamp END as last_analyze
FROM predicate_column_info;
```

자세한 내용은 테이블 분석 단원을 참조하십시오.

단기 쿼리 가속화 활성화

단기 쿼리 가속화(SQA)는 선택한 단기 실행 쿼리를 장기 실행 쿼리보다 우선적으로 적용합니다. SQA 쿼리가 대기열에서 장기 쿼리 뒤에서 대기해야 하지 않도록 SQA는 전용 공간에서 단기 실행 쿼리를 실행합니다. SQA는 단기 실행되고 사용자 정의 대기열에 있는 쿼리에만 우선순위를 부여합니다. SQA가 있으면 단기 실행 쿼리가 더 빠르게 실행하기 시작하며 사용자가 더 빨리 결과를 확인합니다.

SQA를 켜면 단기 쿼리 실행에 전용되는 WLM(Workload Management) 대기열을 축소하거나 제거할 수 있습니다. 또한 장기 실행 쿼리가 단기 실행 쿼리와 대기열의 슬롯에 대해 경합할 필요가 없으므로, 더 적은 쿼리 슬롯을 사용하도록 WLM 대기열을 구성할 수 있습니다. 더 적은 동시성을 사용하면 대부분의 워크로드에 대해 쿼리 처리량이 증가되고 전체 시스템 성능이 향상됩니다. 자세한 내용은 단기 쿼리 가속화 단원을 참조하십시오.

분석

Advisor가 워크로드 패턴을 확인하고 최근 쿼리 개수를 보고합니다. 이때, SQA 적격 쿼리에 대한 지연 시간 및 일일 대기열 시간이 줄어듭니다.

권장 사항

SQA를 켜 WLM 구성을 수정합니다. Amazon Redshift는 기계 학습 알고리즘을 사용하여 각 적격 쿼리를 분석합니다. SQA가 쿼리 패턴에서 학습하면 예측이 향상됩니다. 자세한 내용은 워크로드 관리 구성을 참조하십시오.

SQA를 켜면 WLM이 단기 쿼리에 대한 최대 실행 시간을 '동적(dynamic)'으로 기본 설정합니다. SQA 최대 실행 시간에 대해서는 동적 설정을 유지하는 것이 좋습니다.

구현 팁

SQA가 켜져 있는지 확인하려면 다음 쿼리를 실행합니다. 쿼리가 행을 반환한다면 SQA가 켜진 상태입니다.

```
select * from stv_wlm_service_class_config
where service_class = 14;
```

자세한 내용은 SQA 모니터링 단원을 참조하십시오.

테이블의 분산 키 변경

Amazon Redshift는 테이블 배포 스타일에 따라 클라스터에 테이블 행을 분산합니다. KEY 분산이 있는 테이블에는 분산 키(DISTKEY)로서 열이 필요합니다. 테이블 행은 DISTKEY 열 값에 따라 클러스터의 노드 조각에 할당됩니다.

적절한 DISTKEY는 각 노드 조각에 비슷한 행 수를 배치하고 조인 조건에서 자주 참조됩니다. 테이블이 DISTKEY 열에 조인될 때 최적화된 조인이 발생하여 쿼리 성능이 향상됩니다.

분석

Advisor는 클라우저의 워크로드를 분석하여 키 분산 스타일에서 큰 이점을 얻을 수 있는 테이블에 가장 적절한 분산 키를 식별합니다.

권장 사항

Advisor는 분석을 기반으로 테이블의 DISTSTYLE 및 DISTKEY를 변경하는 ALTER TABLE 문을 제공합니다. 상당한 성능 이점을 실현하려면 권장 그룹 내의 모든 SQL 문을 구현해야 합니다.

ALTER TABLE로 큰 테이블을 다시 분산하면 클러스터 리소스가 사용되고 다양한 시간에 임시 테이블 잠금이 필요합니다. 다른 클러스터 워크로드가 적은 경우 각 권장 그룹을 구현합니다. 테이블 배포 속성의 최적화에 대한 자세한 내용은 Amazon Redshift Engineering의 고급 테이블 설계 플레이북: 배포 스타일 및 배포 키에서 확인할 수 있습니다.

ALTER DISTSYLE 및 DISTKEY에 대한 자세한 내용은 ALTER TABLE 섹션을 참조하세요.

> **Note**
>
> 권장 사항이 표시되지 않는다고 해서 현재 배포 스타일이 가장 적합하다는 의미는 아닙니다. 데이터가 충분하지 않거나 재배포할 경우의 예상되는 이점이 적은 경우에는 Advisor에서 권장 사항을 제공하지 않습니다.
>
> Advisor 권장 사항은 특정 테이블에 적용되며 반드시 동일한 이름의 열이 포함된 테이블에만 적용되지는 않습니다. 열 이름을 공유하는 테이블은 테이블 내부의 데이터가 동일하지 않으면 해당 열에 대해 다른 특성을 가질 수 있습니다.
>
> ETL 작업으로 생성되거나 삭제된 스테이징 테이블에 대한 권장 사항이 표시되면, Advisor 권장 분산 키를 사용하도록 ETL 프로세스를 수정하십시오.

테이블의 정렬 키 변경

Amazon Redshift는 테이블 정렬 키에 따라 테이블 행을 정렬합니다. 테이블 행의 정렬은 정렬 키 열 값을 기준으로 합니다.

적절한 정렬 키로 테이블을 정렬하면 디스크에서 요청하는 테이블 블록 읽기 수가 감소되므로 쿼리, 특히 범위 제한 술어를 사용하는 쿼리의 성능을 가속화할 수 있습니다.

분석

Advisor는 며칠 동안 클러스터의 워크로드를 분석하여 테이블에 적합한 정렬 키를 식별합니다.

권장 사항

Advisor는 분석을 기반으로 테이블의 정렬 키를 변경하는 2개의 ALTER TABLE 문 그룹을 제공합니다.

- 현재 COMPOUND 정렬 키를 추가하기 위한 정렬 키가 없는 테이블을 변경하는 문입니다.
- 정렬 키를 INTERLEAVED에서 COMPOUND 또는 정렬 키 없음으로 변경하는 문입니다.

  복합 정렬 키를 사용하면 유지 관리 오버헤드가 크게 줄어듭니다. 복합 정렬 키가 있는 테이블에는 인터리브 정렬에 필요하고 비용이 많이 드는 VACUUM REINDEX 작업이 필요하지 않습니다. 실제로, 대다수의 Amazon Redshift 워크로드에는 인터리브 정렬 키보다 복합 정렬 키가 훨씬 효율적입니다. 그러나 테이블이 작은 경우 정렬 키 스토리지 오버헤드를 피하기 위해 정렬 키가 없는 것이 더 효율적입니다.

ALTER TABLE을 사용하여 큰 테이블을 정렬할 경우 클러스터 리소스가 사용되고 테이블 잠금이 여러 번 필요합니다. 클러스터의 워크로드가 보통인 경우 각 권장 사항을 구현합니다. 테이블 정렬 키 구성 최적화에 대한 자세한 내용은 Amazon Redshift Engineering's Advanced Table Design Playbook: Compound and Interleaved Sort Keys를 참조하세요.

ALTER SORTKEY에 대한 자세한 내용은 ALTER TABLE 섹션을 참조하세요.

> **Note**
>
> 테이블에 대한 권장 사항이 표시되지 않는다고 해서 현재 구성이 가장 적합하다는 의미는 아닙니다. 데이터가 충분하지 않거나 정렬할 경우 예상되는 이점이 적은 경우에는 Advisor에서 권장 사항을 제공하지 않습니다.
>
> Advisor 권장 사항은 특정 테이블에 적용되며 동일한 이름과 데이터 형식의 열이 포함된 테이블에 반드시 적용할 필요는 없습니다. 열 이름을 공유하는 테이블은 테이블의 데이터와 워크로드에 따라 다른 권장 사항을 가질 수 있습니다.

열의 압축 인코딩 변경

압축은 열 수준에서 저장되는 데이터의 크기를 줄일 수 있는 작업입니다. 압축은 Amazon Redshift에서 디스크 I/O 양을 줄여 스토리지 공간을 절약하고 쿼리 성능을 개선하는 데 사용됩니다. 데이터 형식과 쿼리 패턴을 기반으로 각 열에 대해 최적의 압축 인코딩을 권장합니다. 최적의 압축을 통해 쿼리를 보다 효율적으로 실행할 수 있으며 데이터베이스는 최소한의 스토리지 공간을 차지할 수 있습니다.

분석

Advisor는 클러스터의 워크로드 및 데이터베이스 스키마를 지속적으로 분석하여 각 테이블 열에 대한 최적의 압축 인코딩을 식별합니다.

권장 사항

Advisor는 분석을 기반으로 특정 열의 압축 인코딩을 변경하는 ALTER TABLE 문을 제공합니다.

ALTER TABLE로 열 압축 인코딩을 변경하면 클러스터 리소스가 소모되고 여러 번 테이블 잠금이 필요합니다. 클러스터 워크로드가 적은 경우 권장 사항을 구현하는 것이 가장 좋습니다.

참고로 ALTER TABLE 예은 열의 인코딩을 변경하는 여러 문을 보여줍니다.

> **Note**
>
> 데이터가 충분하지 않거나 인코딩을 변경할 경우 예상되는 이점이 적은 경우에는 Advisor에서 권장 사항을 제공하지 않습니다.

데이터 형식 권장 사항

Amazon Redshift에는 다양한 사용 사례를 위한 SQL 데이터 형식 라이브러리가 있습니다. INT와 같은 정수 형식, VARCHAR과 같이 문자를 저장하는 형식 등을 예로 들 수 있습니다. Redshift는 빠른 액세스와 뛰어난 쿼리 성능을 제공하기 위해 최적화된 방식으로 형식을 저장합니다. 또한 Redshift는 형식을 지정하거나 쿼리 결과에 대한 계산을 수행하는 데 사용할 수 있는 특정 형식을 위한 함수를 제공합니다.

분석

Advisor는 클러스터의 워크로드 및 데이터베이스 스키마를 지속적으로 분석하여 데이터 형식 변경으로 이점을 얻을 수 있는 열을 식별합니다.

권장 사항

Advisor는 제안된 데이터 형식의 새 열을 추가하는 ALTER TABLE 문을 제공합니다. 함께 제공되는 UPDATE 문은 기존 열에서 새 열로 데이터를 복사합니다. 새 열을 생성하고 데이터를 로드한 후 쿼리 및 수집 스크립트를 변경하여 새 열에 액세스합니다. 그런 다음 SQL 함수 참조에서 찾을 수 있는 새로운 데이터 형식에 특화된 기능 및 함수를 활용합니다.

기존 데이터를 새 열에 복사하는 데 시간이 걸릴 수 있습니다. 클러스터의 워크로드가 적은 경우 각 Advisor 권장 사항을 구현하는 것이 좋습니다. 데이터 타입에서 사용 가능한 데이터 형식 목록을 참조합니다.

데이터가 충분하지 않거나 데이터 형식 변경으로 예상되는 이점이 적은 경우에는 Advisor에서 권장 사항을 제공하지 않습니다.

Amazon Redshift 튜토리얼

이 튜토리얼의 단계에 따라 Amazon Redshift 기능에 대해 알아봅니다.

튜토리얼: Amazon S3에서 데이터 로드	이 자습서에서는 S3 버킷에 있는 데이터 파일에서 Amazon Redshift 데이터베이스 테이블로 데이터를 로드하는 프로세스를 처음부터 끝까지 살펴봅니다.
튜토리얼: Amazon Redshift Spectrum을 사용한 중첩 데이터 쿼리	이 자습서에서는 중첩된 데이터를 쿼리할 때 Redshift Spectrum을 사용합니다. Redshift Spectrum을 사용하면 Parquet, ORC, JSON 또는 Ion 파일 형식으로 데이터를 쿼리할 수 있습니다.
자습서: 수동 워크로드 관리(WLM) 대기열 구성	이 자습서에서는 수동 워크로드 관리(WLM) 대기열을 사용하도록 Amazon Redshift를 구성합니다. Amazon Redshift는 WLM 대기열을 통해 리소스를 분할하여 동시 쿼리를 실행하는 방법을 관리합니다. 여러 WLM WLM 대기열을 사용해야 하는 경우 수동 WLM을 사용하도록 Amazon Redshift를 구성해야 합니다.
튜토리얼: Amazon Redshift에서 공간 SQL 함수 사용	이 자습서에서는 공간 함수를 사용하여 데이터를 쿼리합니다. 공간 함수를 사용하여 지오메트리 및 지리 데이터를 쿼리합니다.
Amazon Redshift ML 튜토리얼	이 자습서에서는 기계 학습 모델을 만들고 사용합니다.
자습서: RBAC를 사용한 역할 생성 및 쿼리	이 자습서에서는 생성하는 데이터베이스에서 역할 기반 액세스 제어(RBAC)를 사용하여 권한을 만들고 사용합니다. RBAC를 사용하면 읽기 전용 또는 읽기-쓰기 권한과 같은 특정 권한을 갖는 역할을 만들 수 있습니다. 그런 다음 사용자에게 이러한 역할을 할당하여 지정된 권한을 부여할 수 있습니다.

자동 테이블 최적화

자동 테이블 최적화는 관리자 개입 없이 정렬 및 배포 키를 적용하여 테이블 디자인을 자동으로 최적화하는 자체 조정 기능입니다. 자동화를 사용하여 테이블 설계를 조정하면 테이블 최적화를 수동으로 조정하고 구현하는 데 시간을 투자할 필요 없이 시작하고 가장 빠른 성능을 얻을 수 있습니다.

자동 테이블 최적화는 쿼리와 테이블의 상호 작용 방식을 지속적으로 관찰합니다. 고급 인공 지능 방법으로 정렬 및 배포 키를 선택하여 클러스터의 워크로드에 대한 성능을 최적화합니다. Amazon Redshift에서 키를 적용하면 클러스터 성능이 향상되었다고 판단하면 쿼리에 대한 영향을 최소화하면서 클러스터가 생성된 시점으로부터 몇 시간 이내에 테이블이 자동으로 변경됩니다.

이 자동화를 활용하기 위해 Amazon Redshift 관리자가 새 테이블을 생성하거나 자동 최적화를 사용할 수 있도록 기존 테이블을 변경합니다. 배포 스타일 또는 정렬 키가 AUTO인 기존 테이블이 이미 자동화에 사용됩니다. 이러한 테이블에 대해 쿼리를 실행하면 Amazon Redshift가 정렬 키 또는 배포 키가 성능을 향상시킬지 결정합니다. 이 경우 Amazon Redshift는 관리자 개입 없이 자동으로 테이블을 수정합니다. 최소 수의 쿼리를 실행하면 클러스터가 시작된 후 몇 시간 이내에 최적화가 적용됩니다.

Amazon Redshift에서 배포 키가 쿼리 성능을 향상시킨다고 판단하면 배포 스타일이 AUTO인 테이블은 배포 스타일을 KEY로 변경할 수 있습니다.

주제

- 자동 테이블 최적화 사용, 사용 해제, 모니터링
- 열 압축으로 저장된 데이터 크기 축소
- 쿼리 최적화를 위한 데이터 배포
- 정렬 키
- 테이블 제한 사항

자동 테이블 최적화 사용, 사용 해제, 모니터링

기본적으로 정렬 키나 배포 키를 명시적으로 정의하지 않고 생성된 테이블은 AUTO로 설정됩니다. 테이블 생성 시 수동으로 정렬 또는 배포 키를 명시적으로 설정할 수도 있습니다. 정렬 또는 배포 키를 설정하면 테이블이 자동으로 관리되지 않습니다.

자동 테이블 최적화 사용

기존 테이블이 자동으로 최적화되도록 하려면 ALTER 문 옵션을 사용하여 테이블을 AUTO로 변경합니다. 정렬 키에 대해 자동화를 정의하지만 배포 키에 대해서는 자동화를 정의하지 않도록 선택할 수 있

습니다(반대의 경우도 마찬가지). ALTER 문을 실행하여 테이블을 자동화된 테이블로 변환하면 기존 정렬 키와 배포 스타일이 유지됩니다.

```
ALTER TABLE table_name ALTER SORTKEY AUTO;
```

```
ALTER TABLE table_name ALTER DISTSTYLE AUTO;
```

자세한 내용은 ALTER TABLE 단원을 참조하십시오.

처음에는 테이블에 배포 키나 정렬 키가 없습니다. 배포 스타일은 테이블 크기에 따라 EVEN 또는 ALL로 설정됩니다. 테이블 크기가 커지면 Amazon Redshift가 최적의 배포 키와 정렬 키를 적용합니다. 최적화는 최소 수의 쿼리 실행 후 몇 시간 이내에 적용됩니다. 정렬 키 최적화를 결정할 때 Amazon Redshift는 테이블 스캔 중 디스크에서 읽은 데이터 블록을 최적화하려고 합니다. 배포 스타일 최적화를 결정할 때 Amazon Redshift는 클러스터 노드 간에 전송되는 바이트 수를 최적화하려고 합니다.

테이블에서 자동 테이블 최적화 제거

자동 최적화에서 테이블을 제거할 수 있습니다. 자동화에서 테이블을 제거하려면 정렬 키나 배포 스타일을 선택해야 합니다. 배포 스타일을 변경하려면 특정 배포 스타일을 지정합니다.

```
ALTER TABLE table_name ALTER DISTSTYLE EVEN;
```

```
ALTER TABLE table_name ALTER DISTSTYLE ALL;
```

```
ALTER TABLE table_name ALTER DISTSTYLE KEY DISTKEY c1;
```

정렬 키를 변경하려면 정렬 키를 정의하거나 [없음(none)]을 선택합니다.

```
ALTER TABLE table_name ALTER SORTKEY(c1, c2);
```

```
ALTER TABLE table_name ALTER SORTKEY NONE;
```

자동 테이블 최적화 모니터링

시스템 뷰 SVV_ALTER_TABLE_RECOMMENDATIONS는 테이블에 대한 현재 Amazon Redshift Advisor 권장 사항을 기록합니다. 이 뷰는 모든 테이블(자동 최적화를 위해 정의된 테이블과 그렇지 않은 테이블)에 대한 권장 사항을 보여줍니다.

자동 최적화를 위해 테이블이 정의되어 있는지 보려면 시스템 뷰 SVV_TABLE_INFO를 쿼리합니다. 현재 세션의 데이터베이스에 표시되는 테이블에 대한 항목만 나타납니다. 권장 사항은 클러스터가 생성된 후 몇 시간 이내에 시작하여 하루에 두 번 뷰에 삽입됩니다. 권장 사항이 사용 가능한 상태가 되면 1시간 이내에 시작됩니다. Amazon Redshift 또는 사용자에 의해 적용된 권장 사항은 더 이상 뷰에 표시되지 않습니다.

시스템 보기 SVL_AUTO_WORKER_ACTION에는 Amazon Redshift에서 수행한 모든 작업에 대한 감사 로그와 테이블의 이전 상태가 표시됩니다.

시스템 뷰 SVV_TABLE_INFO에는 테이블의 정렬 키 및 배포 스타일이 AUTO로 설정되었는지 여부를 나타내는 열과 함께 시스템의 모든 테이블이 나열됩니다.

이러한 시스템 뷰에 대한 자세한 내용은 시스템 모니터링(프로비저닝만 해당) 섹션을 참조하세요.

열 압축으로 저장된 데이터 크기 축소

압축은 열 수준에서 저장되는 데이터의 크기를 줄일 수 있는 작업입니다. 이 작업은 스토리지 공간을 절약하여 스토리지에서 읽어오는 데이터의 크기를 줄임으로써 디스크 I/O의 크기가 감소하고 쿼리 성능이 개선되는 효과가 있습니다.

ENCODE AUTO는 테이블의 기본값입니다. 테이블이 ENCODE AUTO로 설정되어 있는 경우 Amazon Redshift는 테이블의 모든 열에 대한 압축 인코딩을 자동으로 관리합니다. 자세한 내용은 CREATE TABLE 및 ALTER TABLE 단원을 참조하세요.

그러나 테이블의 열에 압축 인코딩을 지정하면 테이블이 더 이상 ENCODE AUTO로 설정되지 않습니다. Amazon Redshift는 더 이상 테이블의 모든 열에 대한 압축 인코딩을 자동으로 관리하지 않습니다.

테이블을 생성할 때 테이블의 열에 압축 형식 또는 인코딩을 수동으로 적용할 수 있습니다. 또는 COPY 명령을 사용하여 압축을 자동으로 분석하고 적용할 수 있습니다. 자세한 내용은 COPY를 통한 압축 인코딩 선택 단원을 참조하십시오. 자동 압축 적용에 대한 자세한 내용은 자동 압축을 사용하여 테이블 로드 섹션을 참조하세요.

> **Note**
>
> COPY 명령을 사용하여 자동 압축을 적용하는 것이 매우 바람직합니다.

새 테이블이 다른 테이블과 동일한 데이터 특성을 공유하는 경우 압축 인코딩을 수동으로 적용하도록 선택할 수 있습니다. 또는 테스트에서 자동 압축 중 적용된 압축 인코딩이 데이터에 적합하지 않다는

것을 발견한 경우 그렇게 할 수 있습니다. 압축 인코딩을 수동으로 적용하려면 먼저 이미 데이터가 로드된 테이블을 대상으로 ANALYZE COMPRESSION 명령을 실행한 후 그 결과를 가지고 압축 인코딩을 선택할 수 있습니다.

또한 압축을 수동으로 적용할 때는 각 열에 대한 압축 인코딩을 CREATE TABLE 문에 포함시켜 지정합니다. 구문은 다음과 같습니다.

```
CREATE TABLE table_name (column_name
data_type ENCODE encoding-type)[, ...]
```

여기서 encoding-type은 다음 섹션에서 설명하는 키워드 표에서 가져옵니다.

예를 들어 다음은 2열 테이블인 PRODUCT를 생성하는 문입니다. 데이터를 테이블에 로드하면 PRODUCT_ID 열은 압축되지 않지만 PRODUCT_NAME 열이 BYTEDICT(Byte Dictionary Encoding)를 사용하여 압축됩니다.

```
create table product(
product_id int encode raw,
product_name char(20) encode bytedict);
```

열의 인코딩은 ALTER TABLE 명령을 사용해 테이블에 추가할 때 지정할 수 있습니다.

```
ALTER TABLE table-name ADD [ COLUMN ] column_name column_type ENCODE encoding-type
```

주제

- 압축 인코딩
- 압축 인코딩 테스트

압축 인코딩

행을 테이블에 추가할 때 데이터 값의 열에 적용되는 압축 형식은 압축 인코딩에 따라 결정됩니다.

ENCODE AUTO는 테이블의 기본값입니다. 테이블이 ENCODE AUTO로 설정되어 있는 경우 Amazon Redshift는 테이블의 모든 열에 대한 압축 인코딩을 자동으로 관리합니다. 자세한 내용은 CREATE TABLE 및 ALTER TABLE 단원을 참조하세요.

그러나 테이블의 열에 압축 인코딩을 지정하면 테이블이 더 이상 ENCODE AUTO로 설정되지 않습니다. Amazon Redshift는 더 이상 테이블의 모든 열에 대한 압축 인코딩을 자동으로 관리하지 않습니다.

CREATE TABLE을 사용하는 경우 테이블의 열에 대한 압축 인코딩을 지정하면 ENCODE AUTO가 비활성화됩니다. ENCODE AUTO가 비활성화된 경우 Amazon Redshift는 다음과 같이 ENCODE 유형을 지정하지 않은 열에 압축 인코딩을 자동으로 할당합니다.

- 정렬 키로 정의된 열은 RAW 압축이 할당됩니다.
- BOOLEAN, REAL 또는 DOUBLE PRECISION 데이터 형식으로 정의된 열은 RAW 압축이 할당됩니다.
- SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIMESTAMP 또는 TIMESTAMPTZ 데이터 형식으로 정의된 열에는 AZ64 압축이 할당됩니다.
- CHAR 또는 VARCHAR 데이터 형식으로 정의된 열에는 LZO 압축이 할당됩니다.

테이블을 만든 후 ALTER TABLE을 사용하여 테이블의 인코딩을 변경할 수 있습니다. ALTER TABLE을 사용하여 ENCODE AUTO를 비활성화하면 Amazon Redshift는 더 이상 열의 압축 인코딩을 자동으로 관리하지 않습니다. 모든 열은 사용자가 변경하거나 ENCODE AUTO를 다시 활성화하기 전까지는 ENCODE AUTO를 비활성화한 시점의 압축 인코딩 유형을 그대로 유지합니다.

Amazon Redshift는 다음과 같은 압축 인코딩을 지원합니다.

Raw

Raw 인코딩은 정렬 키로 지정되는 열과 BOOLEAN, REAL 또는 DOUBLE PRECISION 데이터 형식으로 정의되는 열에서 기본 인코딩입니다. Raw 인코딩에서는 데이터가 압축되지 않은 원시 형태로 저장됩니다.

AZ64

AZ64는 Amazon에서 높은 압축비와 개선된 쿼리 처리 기능을 달성하도록 설계된 독점 압축 인코딩 알고리즘입니다. 핵심적으로, AZ64 알고리즘은 더 작은 그룹의 데이터 값을 압축하고 병렬 처리를 위해 단일 명령, 다중 데이터(SIMD) 명령을 사용합니다. AZ64를 사용하면 스토리지를 크게 절약하고 숫자, 날짜, 시간 데이터 유형에서 뛰어난 성능을 달성할 수 있습니다.

다음 데이터 유형의 CREATE TABLE 및 ALTER TABLE 문을 사용하여 열을 정의할 때 AZ64를 압축 인코딩으로 사용할 수 있습니다.

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL

- 날짜
- TIMESTAMP
- TIMESTAMPTZ

Byte-dictionary

Byte Dictionary 인코딩에서는 디스크에 저장되어 있는 열 값 블록마다 고유 값의 딕셔너리가 별도로 생성됩니다. Amazon Redshift 디스크 블록은 1MB를 차지합니다. 딕셔너리에는 1바이트 값이 최대 256개까지 원래 데이터 값에 대한 인덱스로 저장됩니다. 단일 블록에 저장되는 값이 256개를 넘어가면 추가 값이 압축되지 않은 원시 형태로 블록에 작성됩니다. 이 프로세스가 각 디스크 블록마다 반복됩니다.

이 인코딩은 카디널리티가 낮은 문자열 열에서 매우 효과적입니다. 그래서 열의 데이터 도메인이 256개의 고유 값보다 적을 때 가장 적합합니다.

BYTEDICT로 인코딩된 문자열 데이터 유형(CHAR 및 VARCHAR)이 있는 열의 경우 Amazon Redshift는 압축된 데이터를 대상으로 직접 작동하는 벡터화된 스캔 및 조건자 평가를 수행합니다. 이러한 스캔은 하드웨어별 단일 명령과 다중 데이터(SIMD) 명령을 사용하여 병렬 처리를 수행합니다. 이렇게 하면 문자열 열 스캔 속도가 크게 향상됩니다. Byte-Dictionary 인코딩은 CHAR/VARCHAR 열에 긴 문자열이 저장되어 있는 경우에 특히 공간 효율적입니다.

예를 들어 테이블에 CHAR(30) 데이터 형식의 COUNTRY 열이 있다고 가정하겠습니다. 데이터가 로드되면서 Amazon Redshift가 딕셔너리를 생성한 후 인덱스 값으로 COUNTRY 열을 채웁니다. 딕셔너리에 고유의 인덱스 값이 저장되고, 테이블 자체에는 해당하는 값의 1바이트 서브스크립트만 저장됩니다.

> **Note**
>
> 고정 길이 문자 열(column)에서는 후행 공백이 저장됩니다. 따라서 CHAR(30) 열에서 Byte-Dictionary 인코딩을 사용할 때는 모든 값이 압축되어 29바이트의 스토리지를 절약합니다.

다음 표는 COUNTRY 열의 딕셔너리를 나타낸 것입니다.

고유 데이터 값	딕셔너리 인덱스	크기(고정 길이, 값 1개당 30바이트)
England	0	30

고유 데이터 값	딕셔너리 인덱스	크기(고정 길이, 값 1개당 30바이트)
United States of America	1	30
Venezuela	2	30
Sri Lanka	3	30
Argentina	4	30
Japan	5	30
합계		180

다음 표는 COUNTRY 열의 값을 나타낸 것입니다.

원래 데이터 값	원래 크기(고정 길이, 값 1개당 30바이트)	압축된 값(인덱스)	새로운 크기(바이트)
England	30	0	1
England	30	0	1
United States of America	30	1	1
United States of America	30	1	1
Venezuela	30	2	1
Sri Lanka	30	3	1
Argentina	30	4	1
Japan	30	5	1
Sri Lanka	30	3	1

원래 데이터 값	원래 크기(고정 길이, 값 1개당 30바이트)	압축된 값(인덱스)	새로운 크기(바이트)
Argentina	30	4	1
합계	300		10

위 예에서 압축된 전체 크기는 다음과 같은 방법으로 계산합니다. 딕셔너리에 저장되는 입력 항목이 6개이고(6 * 30 = 180), 테이블에는 1바이트로 압축된 값 10개가 저장되어 전체 크기는 190바이트가 됩니다.

Delta

Delta 인코딩은 날짜/시간 열에 매우 유용합니다.

Delta 인코딩은 열에서 서로 인접한 값 사이의 차이를 기록하여 데이터를 압축합니다. 이 차이는 디스크에 저장되어 있는 열 값 블록마다 별도의 딕셔너리에 기록됩니다. Amazon Redshift 디스크 블록은 1MB를 차지합니다. 예를 들어 열에 1에서 10까지의 순서로 10개의 정수가 포함되어 있다고 가정합니다. 첫 번째는 4바이트 정수(1바이트 플래그 포함)로 저장됩니다. 다음 9개는 각각 값이 1인 바이트로 저장되어 이전 값보다 1이 큼을 나타냅니다.

Delta 인코딩은 다음 2가지 유형이 있습니다.

- DELTA는 차이를 1바이트 값으로 기록합니다(8비트 정수).
- DELTA32K는 차이를 2바이트 값으로 기록합니다(16비트 정수).

열에 있는 대부분의 값을 단일 바이트로 압축할 수 있다면 1바이트 변형이 매우 효과적입니다. 그러나 델타가 더 크면 최악의 경우 이 인코딩이 압축되지 않은 데이터를 저장하는 것보다 효과가 떨어질 수 있습니다. 16비트 버전에도 비슷한 로직이 적용됩니다.

두 값의 차이가 1바이트 범위(DELTA) 또는 2바이트 범위(DELTA32K)를 벗어나면 1바이트 플래그가 선행하면서 원래 값 전체가 저장됩니다. 여기에서 1바이트 범위란 -127~127을, 2바이트 범위란 -32~32K를 의미합니다.

다음 표는 숫자 열에서 Delta 인코딩의 압축 방식을 나타낸 것입니다.

원래 데이터 값	원래 크기(바이트)	차이(델타)	압축된 값	압축된 크기(바이트)
1	4		1	1+4(플래그 + 실제 값)
5	4	4	4	1
50	4	45	45	1
200	4	150	150	1+4(플래그 + 실제 값)
185	4	-15	-15	1
220	4	35	35	1
221	4	1	1	1
합계	28			15

LZO

LZO 인코딩은 높은 압축비로 우수한 성능을 제공하기 때문에 긴 문자열이 저장되는 CHAR 및 VARCHAR 열에서 매우 효과적입니다. 특히 제품 설명, 사용자 의견 또는 JSON 문자열과 같은 자유 형식 텍스트에 적합합니다.

Mostly

Mostly 인코딩은 열의 데이터 형식이 대부분 저장되는 값에 필요한 데이터 형식보다 큰 경우에 유용합니다. 이러한 열 형식에 Mostly 인코딩을 지정하면 열에 저장되어 있는 대부분 값을 더 작은 표준 스토리지 크기로 압축할 수 있습니다. 압축되지 않는 나머지 값들은 원시 형태로 저장됩니다. 예를 들어 INT2 열 같은 16비트 열을 8비트 스토리지로 압축할 수 있습니다.

일반적으로 Mostly 인코딩은 다음과 같은 데이터 형식에서 유효합니다.

- SMALLINT/INT2(16비트)
- INTEGER/INT(32비트)
- BIGINT/INT8(64비트)

- DECIMAL/NUMERIC(64비트)

열의 데이터 형식 크기에 따라 적합한 유형의 Mostly 인코딩을 선택합니다. 예를 들어 16비트 정수 열로 정의되어 있는 열에는 MOSTLY8을 적용합니다. 16비트 데이터 형식의 열에 MOSTLY16을 적용하거나, 혹은 32비트 데이터 형식의 열에 MOSTLY32를 적용하는 것은 허용되지 않습니다.

열에서 압축할 수 없는 값이 비교적 많은 경우에는 인코딩이 압축하지 않는 것보다 대부분 효과적이지 못할 수도 있습니다. 이러한 인코딩 중 하나를 열에 적용하기 전에 확인합니다. 로드하려고 하는(혹은 앞으로 로드할 가능성이 있는) 대부분 값이 다음 표의 범위에 해당해야 합니다.

인코딩	압축된 스토리지 크기	압축할 수 있는 값의 범위(범위를 벗어나는 값은 원시 형태로 저장됨)
MOSTLY8	1바이트(8비트)	-128~127
LZO	2바이트(16비트)	-32768~32767
MOSTLY32	4바이트(32비트)	4 bytes

> **Note**
>
> 소수 값인 경우에는 값이 범위에 해당하는지 확인할 때 소수점을 무시하십시오. 예를 들어 1,234.56은 123,456으로 처리되어 MOSTLY32 열에서 압축될 수 있습니다.

예를 들어 VENUE 테이블의 VENUEID 열은 원시 형태의 정수 열로 정의됩니다. 이 말은 열의 값이 4바이트의 스토리지를 소비한다는 것을 의미합니다. 하지만 현재 열 값의 범위는 **0~309**입니다. 따라서 이 테이블을 재생성한 후 VENUEID 열에 MOSTLY16 인코딩을 적용하여 다시 로드하면 해당 열 값의 스토리지가 2바이트로 줄어듭니다.

다른 테이블에서 참조한 VENUEID 값의 범위가 대부분 0~127이었다면 외래 키 열을 MOSTLY8로 인코딩하는 것이 좋을 수도 있습니다. 따라서 선택을 하기 전에 참조 테이블 데이터를 대상으로 여러 쿼리를 실행하여 값이 해당하는 범위가 8비트인지, 16비트인지 또는 32비트인지 확인해야 합니다.

다음 표는 MOSTLY8, MOSTLY16 및 MOSTLY32 인코딩을 사용할 때 특정 숫자 값의 압축 크기를 나타낸 것입니다.

원래 값	원래 INT 또는 BIGINT 크기(바이트)	MOSTLY8 압축 크기(바이트)	MOSTLY16 압축 크기(바이트)	MOSTLY32 압축 크기(바이트)
1	4	1	2	4
10	4	1	2	4
100	4	1	2	4
1000	4	원시 데이터 크기와 동일함	2	4
10000	4		2	4
20000	4		2	4
40000	8		원시 데이터 크기와 동일함	4
100000	8			4
2000000000	8			4

Run length

Run length 인코딩은 연속해서 반복되는 값을 해당 값과 연속해서 발생하는 횟수(실행 길이)로 구성된 토큰으로 대체합니다. 디스크에 저장되어 있는 열 값 블록마다 고유 값의 딕셔너리가 별도로 생성됩니다. Amazon Redshift 디스크 블록은 1MB를 차지합니다. 이 인코딩은 예를 들어 테이블이 데이터 값을 기준으로 정렬되는 경우처럼 데이터 값이 종종 연속해서 반복되는 테이블에 가장 적합합니다.

예를 들어 큰 차원 테이블의 열에 가능한 값이 10개 미만인 COLOR 열과 같이 예상대로 작은 도메인이 있다고 가정합니다. 이러한 값은 데이터가 정렬되지 않은 경우에도 테이블 전체에서 긴 순서로 나타날 수 있습니다.

정렬 키로 지정하는 열에 run length 인코딩을 적용하는 것은 바람직하지 않습니다. 블록에 비슷한 수의 행이 저장되는 경우에는 범위를 제한한 스캔이 훨씬 효과적입니다. 하지만 동일한 쿼리에서 정렬 키 열이 다른 열보다 훨씬 높게 압축되는 경우에는 범위를 제한한 스캔 효과가 떨어질 수도 있습니다.

다음 표는 COLOR 열을 예로 들어 run length 인코딩의 적용 방식을 나타낸 것입니다.

원래 데이터 값	원래 크기(바이트)	압축된 값(토큰)	압축된 크기(바이트)
Blue	4	4	5
Blue	4		0
Green	5	5	6
Green	5		0
Green	5		0
Blue	4	{1,Blue}	5
Yellow	6	{4,Yellow}	7
Yellow	6		0
Yellow	6		0
Yellow	6		0
합계	51		23

Text255 and Text32k

Text255 및 Text32k 인코딩은 동일한 단어가 자주 반복되는 VARCHAR 열을 압축하는 데 유용합니다. 디스크에 저장되어 있는 열 값 블록마다 고유 단어의 딕셔너리가 별도로 생성됩니다. Amazon Redshift 디스크 블록은 1MB를 차지합니다. 딕셔너리에는 열에서 첫 고유 단어 245개가 저장됩니다. 이러한 단어들은 디스크에서 245개 값 중 하나를 표현하는 1바이트 인덱스 값으로 대체되고, 딕셔너리에 표현되지 않는 단어들은 모두 비압축 상태로 저장됩니다. 이 프로세스가 1MB 디스크 블록마다 반복됩니다. 인덱스로 대체된 단어가 열에 자주 나타나면 열의 압축비가 높아집니다.

Text32k 인코딩일 때도 원칙은 동일하지만 각 블록의 딕셔너리가 특정 수의 단어를 수집하지는 않습니다. 대신에 딕셔너리가 전체 입력 항목이 일부 오버헤드를 제외하고 32K에 도달할 때까지 발견하는 고유 단어를 각각 인덱싱합니다. 인덱스 값은 2개 바이트로 저장됩니다.

예를 들어 VENUE 테이블의 VENUENAME 열을 가정하겠습니다. 이 열에서 **Arena**, **Center**, **Theatre** 같은 단어들이 반복되어 text255 압축을 적용한다면 이러한 단어들이 각 블록에서 발견되는 첫 245개 단어에 포함될 가능성이 높습니다. 그렇다면 이 열에는 압축이 유리합니다. 이러한

단어들이 나올 때마다 스토리지를 1바이트(그렇지 않을 경우 각각 5, 6 또는 7바이트)만 차지하기 때문입니다.

ZSTD

ZSTD(Zstandard) 인코딩은 다양한 데이터 세트에서도 높은 압축비로 우수한 성능을 제공합니다. 제품 설명, 사용자 주석, 로그, JSON 문자열처럼 길고 짧은 문자열이 다양하게 저장되는 CHAR 및 VARCHAR 열에서 특히 효과적입니다. Delta 인코딩 또는 Mostly 인코딩 등 일부 알고리즘이 압축하지 않는 것보다 더 많은 스토리지 공간을 사용할 가능성이 있는 경우에도 ZSTD에서의 디스크 사용량이 늘어날 가능성이 높지 않습니다.

ZSTD는 SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE, TIMESTAMP 및 TIMESTAMPTZ 데이터 형식을 지원합니다.

다음 표는 지원되는 압축 인코딩과 인코딩을 지원하는 데이터 형식을 구분한 것입니다.

인코딩 유형	CREATE TABLE 및 ALTER TABLE 문의 키워드	데이터 타입
Raw(압축 없음)	RAW	모두
AZ64	AZ64	SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIMESTAMP, TIMESTAMPTZ
Byte dictionary	BYTEDICT	SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE PRECISION, CHAR, VARCHAR, DATE, TIMESTAMP, TIMESTAMP TZ
델타	델타 델타	SMALLINT, INT, BIGINT, DATE, TIMESTAMP, DECIMAL INT, BIGINT, DATE, TIMESTAMP, DECIMAL
LZO	LZO	SMALLINT, INTEGER, BIGINT, DECIMAL, CHAR, VARCHAR,

인코딩 유형	CREATE TABLE 및 ALTER TABLE 문의 키워드	데이터 타입
		DATE, TIMESTAMP, TIMESTAMP TZ, SUPER
Mostlyn	LZO	SMALLINT, INT, BIGINT, DECIMAL
	LZO	INT, BIGINT, DECIMAL
	LZO	BIGINT, DECIMAL
Run-length	RUNLENGTH	SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE, TIMESTAMP, TIMESTAMPTZ
텍스트	TEXT255	VARCHAR
	TEXT32K	VARCHAR
Zstandard	ZSTD	SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE, TIMESTAMP, TIMESTAMPTZ, SUPER

압축 인코딩 테스트

압축 인코딩을 수동으로 지정하는 경우 데이터를 이용해 여러 가지 인코딩을 테스트할 수 있습니다.

> **Note**
>
> 가능하다면 항상 COPY 명령을 사용하여 데이터를 로드하고, COPY 명령에서 데이터에 따른 최적의 인코딩을 선택하도록 허용하는 것이 좋습니다. 또는 ANALYZE COMPRESSION 명령을 사용하여 기존 데이터에 적합한 인코딩을 확인하는 방법도 있습니다. 자동 압축 적용에 대한 자세한 내용은 자동 압축을 사용하여 테이블 로드 섹션을 참조하세요.

데이터 압축에 대한 유의적 테스트를 위해서는 다수의 행이 필요합니다. 이번 예에서는 임의의 테이블을 생성한 후 이 테이블이 VENUE 테이블과 LISTING 테이블에서 선택하는 문을 사용하여 행을 삽입합니다. 일반적으로 두 테이블을 조인하는 WHERE 절을 생략합니다. 결과적으로 VENUE 테이블의 각 행이 LISTING 테이블의 모든 행으로 조인되어 전체 행의 수는 3,200만 개가 됩니다. 이를 데카르트 조인이라고 하며 일반적으로 권장되지 않습니다. 그러나 이는 이러한 용도로 많은 행을 생성하는 편리한 방법입니다. 테스트할 데이터가 기존 테이블에 로드되어 있으면 이번 단계를 생략해도 좋습니다.

샘플 데이터가 있는 테이블을 생성한 후 7개의 열이 있는 테이블을 생성합니다. 각각 다른 압축 인코딩(raw, bytedict, lzo, run length, text255, text32k 및 zstd)이 있습니다. 첫 번째 테이블에서 데이터를 선택할 때 사용한 INSERT 명령을 실행하여 정확히 동일한 데이터로 각 열을 채웁니다.

압축 인코딩을 테스트하려면 다음을 수행합니다.

1. (옵션) 먼저 데카르트 조인을 사용하여 다수의 행이 포함된 테이블을 하나 생성합니다. 기존 테이블을 테스트할 때는 이 단계를 생략해도 좋습니다.

```
create table cartesian_venue(
venueid smallint not null distkey sortkey,
venuename varchar(100),
venuecity varchar(30),
venuestate char(2),
venueseats integer);

insert into cartesian_venue
select venueid, venuename, venuecity, venuestate, venueseats
from venue, listing;
```

2. 비교할 인코딩을 적용하여 테이블을 생성합니다.

```
create table encodingvenue (
venueraw varchar(100) encode raw,
venuebytedict varchar(100) encode bytedict,
venuelzo varchar(100) encode lzo,
venuerunlength varchar(100) encode runlength,
venuetext255 varchar(100) encode text255,
venuetext32k varchar(100) encode text32k,
venuezstd varchar(100) encode zstd);
```

3. SELECT 절에 INSERT 문을 사용하여 동일한 데이터를 모든 열에 삽입합니다.

```
insert into encodingvenue
```

```
select venuename as venueraw, venuename as venuebytedict, venuename as venuelzo,
 venuename as venuerunlength, venuename as  venuetext32k, venuename as  venuetext255,
 venuename as venuezstd
from cartesian_venue;
```

4. 새로운 테이블에서 행의 수를 확인합니다.

```
select count(*) from encodingvenue

  count
----------
 38884394
(1 row)
```

5. STV_BLOCKLIST 시스템 테이블에 대해 쿼리를 실행하여 각 열에서 사용된 1MB 디스크 블록의 수를 비교합니다.

   MAX 집계 함수는 각 열마다 가장 높은 블록 수를 반환합니다. STV_BLOCKLIST 테이블에는 시스템에서 작성된 열 3개에 대한 세부 정보가 포함되어 있습니다. 다음 예는 WHERE 절에서 `col < 6`을 사용하여 시스템에서 작성된 열을 배제한 것입니다.

```
select col, max(blocknum)
from stv_blocklist b, stv_tbl_perm p
where (b.tbl=p.id) and name ='encodingvenue'
and col < 7
group by name, col
order by col;
```

   위 쿼리는 다음과 같은 결과를 반환합니다. 열은 0부터 번호가 매겨집니다. 클러스터의 구성 방식에 따라 결과적으로 다른 번호가 될 수는 있지만 상대적 크기는 비슷해야 합니다. 이 데이터 집합에서는 두 번째 열에 있는 BYTEDICT 인코딩이 최상의 결과를 이끌어내는 것을 알 수 있습니다. 이 접근 방식의 압축 비율은 20:1보다 좋습니다. LZO 인코딩과 ZSTD 인코딩 역시 우수한 결과를 산출하였습니다. 물론 데이터 세트가 달라지면 결과도 바뀝니다. 열에 더욱 긴 텍스트 문자열이 포함되어 있으면 LZO 인코딩이 종종 최상의 압축 결과를 산출하기도 합니다.

```
 col | max
-----+-----
   0 | 203
   1 |  10
   2 |  22
   3 | 204
```

```
   4 |  56
   5 |  72
   6 |  20
(7 rows)
```

기존 테이블에 이미 데이터가 있다면 ANALYZE COMPRESSION 명령을 사용하여 테이블에 적합한 인코딩을 확인할 수도 있습니다. 예를 들어 다음 예는 행의 수가 3,800만 개인 VENUE 테이블(CARTESIAN_VENUE)을 복사하는 데 가장 적합한 인코딩을 나타내고 있습니다. 예를 보면, ANALYZE COMPRESSION을 실행한 결과 VENUENAME 열에는 LZO 인코딩을 적용하는 것이 바람직합니다. ANALYZE COMPRESSION은 감소율을 포함한 다수의 요인을 기준으로 최적의 압축 인코딩을 선택합니다. 이번 예와 같은 경우에는 BYTEDICT의 압축 성능이 우수하지만 LZO 역시 90%가 넘는 압축비를 보이고 있습니다.

```
analyze compression cartesian_venue;

Table          | Column     | Encoding | Est_reduction_pct
---------------+------------+----------+------------------
reallybigvenue | venueid    | lzo      | 97.54
reallybigvenue | venuename  | lzo      | 91.71
reallybigvenue | venuecity  | lzo      | 96.01
reallybigvenue | venuestate | lzo      | 97.68
reallybigvenue | venueseats | lzo      | 98.21
```

예

다음 예제는 다양한 데이터 형식의 열을 포함시켜 CUSTOMER 테이블을 생성합니다. 이 CREATE TABLE 문은 각 열에 적합한 압축 인코딩을 조합할 수 있는 여러 가지 방법 중 한 가지를 나타내고 있습니다.

```
create table customer(
custkey int encode delta,
custname varchar(30) encode raw,
gender varchar(7) encode text255,
address varchar(200) encode text255,
city varchar(30) encode text255,
state char(2) encode raw,
zipcode char(5) encode bytedict,
start_date date encode delta32k);
```

다음 표는 CUSTOMER 테이블의 각 열마다 선택한 인코딩과 각 선택에 대한 설명을 나열한 것입니다.

열	데이터 유형	인코딩	설명
CUSTKEY	int	int	CUSTKEY 열은 연속된 고유의 정수 값으로 구성됩니다. 차이가 1바이트이기 때문에 DELTA는 좋은 선택입니다.
CUSTNAME	varchar(30)	raw	CUSTNAME 열에는 소수의 값이 반복되는 큰 도메인이 있습니다. 어떤 압축 인코딩을 적용하든 효과가 떨어질 수도 있습니다.
GENDER	varchar(7)	Text255	GENDER 열은 다수의 값이 반복되는 매우 작은 도메인입니다. 동일한 단어가 자주 반복되는 VARCHAR 열에서는 Text255 인코딩이 효과적입니다.
ADDRESS	varchar(200)	Text255	ADDRESS는 큰 도메인이지만 Street, Avenue, North, South 등 다수의 반복어가 포함되어 있습니다. 동일한 단어가 반복되는 VARCHAR 열을 압축하려면 Text255 및 Text32k 인코딩이 유용합니다. 여기에서는 열의 길이가 짧기 때문에 Text255가 좋은 선택입니다.

열	데이터 유형	인코딩	설명
CITY	varchar(30)	Text255	CITY 열은 일부 값이 반복되는 큰 도메인입니다. 몇 가지 도시 이름이 다른 도시에 비해 훨씬 더 공통적으로 사용됩니다. 따라서 여기에서도 ADDRESS와 같은 이유로 Text255가 좋은 선택입니다.
STATE	char(2)	raw	미국의 경우 STATE 열은 2자 값 50개로 구성된, 정확한 도메인입니다. BYTEDICT 인코딩이 일부 압축을 실행하겠지만 열 크기가 2자에 불과하기 때문에 데이터를 압축하지 않는 오버헤드를 감수할 정도로 압축 효과가 크지 않습니다.
ZIPCODE	char(5)	BYTEDICT	ZIPCODE는 50,000개 미만의 고유 값으로 구성되어 이미 알고 있는 도메인입니다. 몇 가지 우편 번호는 다른 우편 번호보다 훨씬 더 공통적으로 사용됩니다. BYTEDICT 인코딩은 열에 제한된 수의 고유 값이 저장될 때 매우 효과적입니다.

열	데이터 유형	인코딩	설명
START_DATE	date	날짜	Delta 인코딩은 날짜/시간 열에서도, 특히 행이 날짜 순서대로 로드되는 경우에 매우 유용합니다.

쿼리 최적화를 위한 데이터 배포

테이블에 데이터를 로드하면 Amazon Redshift가 테이블의 배포 스타일에 따라 테이블 행을 각 컴퓨팅 노드로 분산시킵니다. 쿼리를 실행하면 쿼리 옵티마이저가 필요에 따라 쿼리 행을 컴퓨팅 노드로 다시 분산시켜 조인 및 집계를 실행합니다. 테이블 배포 스타일을 선택하는 목적은 쿼리 실행 이전에 데이터의 분산 위치를 지정하여 재분산 단계의 영향을 최소화하는 데 있습니다.

> **Note**
>
> 이 섹션에서는 Amazon Redshift 데이터베이스의 데이터 배포 원칙을 소개합니다. `DISTSTYLE AUTO`로 테이블을 생성하는 것이 좋습니다. 그러면 Amazon Redshift가 자동 테이블 최적화를 사용하여 데이터 배포 스타일을 선택합니다. 자세한 내용은 자동 테이블 최적화 단원을 참조하십시오. 이 섹션의 나머지 부분에서는 배포 스타일에 대한 세부 정보를 제공합니다.

주제

- 데이터 분산 개념
- 분산 스타일
- 분산 스타일 보기
- 쿼리 패턴 평가
- 분산 스타일 지정
- 쿼리 계획 평가
- 쿼리 계획 예
- 분산 예제

데이터 분산 개념

다음은 Amazon Redshift에 대한 몇 가지 데이터 배포 개념입니다.

노드 및 조각

Amazon Redshift 클러스터는 노드 집합입니다. 클러스터를 구성하는 각 노드는 운영 체제, 전용 메모리 및 전용 디스크 스토리지를 자체적으로 가지고 있습니다. 여기에는 리더 노드라고 하는 노드가 하나 있습니다. 리더 노드는 데이터 및 쿼리 처리 태스크를 컴퓨팅 노드로 분산시킬 수 있도록 관리하는 역할을 합니다. 컴퓨팅 노드는 이러한 태스크를 수행하기 위한 리소스를 제공합니다.

컴퓨팅 노드의 디스크 스토리지는 다수의 조각으로 분할됩니다. 노드당 조각 수는 클러스터의 노드 크기에 따라 달라집니다. 각 노드는 병렬 쿼리 실행에 참여하여 조각에 대한 데이터 분산 작업을 최대한 균일하게 실행합니다. 각 노드 크기의 슬라이스 수에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 클러스터 및 노드 정보 섹션을 참조하세요.

데이터 재분산

테이블에 데이터를 로드하면 Amazon Redshift가 테이블의 배포 스타일에 따라 테이블 행을 각 노드 조각으로 분산시킵니다. 옵티마이저가 쿼리 계획에 따라 최적의 쿼리 실행을 위한 데이터 블록 위치를 결정합니다. 그런 다음 쿼리가 실행되는 동안 데이터가 물리적으로 이동되거나 재배포됩니다. 재분산에는 조인을 위해 특정 행을 노드로 전송하거나, 혹은 전체 테이블을 모든 노드로 브로드캐스팅하는 작업이 포함될 수 있습니다.

데이터 재분산은 쿼리 계획에서 차지하는 비용 부분이 매우 클 뿐만 아니라 여기에서 발생하는 네트워크 트래픽은 다른 데이터베이스 작업에도 영향을 미쳐 전반적인 시스템 성능이 느려질 수도 있습니다. 하지만 초기에 최적의 데이터 저장 위치를 예상하는 범위까지는 데이터 재분산의 영향을 최소화할 수 있습니다.

데이터 분산 목적

데이터를 테이블에 로드할 때는 Amazon Redshift가 테이블 생성 시 선택한 배포 스타일에 따라 테이블의 행을 컴퓨팅 노드 및 조각으로 분산시킵니다. 데이터 분산은 기본적으로 다음과 같은 두 가지 목적이 있습니다.

- 워크로드를 클러스터 노드에 균일하게 분산시킵니다. 불균일 분산, 즉 데이터 분산 스큐는 일부 노드가 상대적으로 더 많은 작업을 하게 되면서 쿼리 성능을 떨어뜨립니다.
- 쿼리 실행 시 데이터 이동을 최소화하려면 조인 또는 집계에 참여하는 행이 다른 테이블의 조인 행과 함께 노드에 공동 배치되어 있는 경우에는 옵티마이저가 쿼리 실행 시 공동 배치되어 있는 만큼 데이터를 재분산시킬 필요가 없습니다.

데이터베이스에 선택하는 분산 전략은 쿼리 성능, 스토리지 요건, 데이터 로딩 및 유지 관리에 미치는 비중이 매우 큽니다. 따라서 각 테이블마다 최적의 분산 스타일을 선택해야만 데이터 분산 밸런스를 맞춰 전체적인 시스템 성능을 크게 높일 수 있습니다.

분산 스타일

테이블을 생성할 때는 AUTO, EVEN, KEY, ALL 등 네 가지 분산 스타일 중 하나를 지정할 수 있습니다.

배포 스타일을 지정하지 않으면 Amazon Redshift에서는 AUTO 배포를 사용합니다.

AUTO 분산

AUTO 배포를 사용하면 Amazon Redshift에서는 테이블 데이터의 크기를 기반으로 최적의 배포 스타일을 할당합니다. 예를 들어 AUTO 배포 스타일을 지정하면 Amazon Redshift는 처음에 작은 테이블에 ALL 배포 스타일을 할당합니다. 테이블이 커지면 Amazon Redshift는 기본 키(또는 복합 기본 키의 열)를 배포 키로 선택하여 배포 스타일을 KEY로 변경할 수 있습니다. 테이블이 커지고 배포 키로 적합한 열이 없으면 Amazon Redshift는 배포 스타일을 EVEN으로 변경합니다. 배포 스타일 변경은 사용자 쿼리에 미치는 영향을 최소화하면서 백그라운드에서 이루어집니다.

Amazon Redshift가 테이블 배포 키를 변경하기 위해 자동으로 수행한 작업을 보려면 SVL_AUTO_WORKER_ACTION 섹션을 참조하세요. 테이블 배포 키 변경에 대한 현재 권장 사항을 보려면 SVV_ALTER_TABLE_RECOMMENDATIONS 섹션을 참조하세요.

테이블에 적용된 분산 스타일을 보려면 PG_CLASS_INFO 시스템 카탈로그 보기를 쿼리합니다. 자세한 내용은 분산 스타일 보기 단원을 참조하십시오. CREATE TABLE 문을 사용해 배포 스타일을 지정하지 않으면 Amazon Redshift에서는 AUTO 배포를 적용합니다.

EVEN 분산

리더 노드는 특정 열 값에 상관없이 행을 라운드 로빈 방식으로 조각에 분산시킵니다. EVEN 배포는 테이블이 조인에 참여하지 않는 경우 적합하며, KEY 배포와 ALL 배포 사이에 명확한 선택이 없는 경우에도 적합합니다.

KEY 분산

행이 열 1개의 값에 따라 분산됩니다. 리더 노드는 일치하는 값을 동일한 노드 조각에 할당합니다. 조인 키를 기준으로 테이블 페어를 분산시키면 리더 노드가 조인 열의 값에 따라 행을 조각에 공동 배치하기 때문에 공통 열에서 일치하는 값은 물리적으로 함께 저장됩니다. 이렇게 하면 공통 열의 일치하는 값이 물리적으로 함께 저장됩니다.

ALL 분산

전체 테이블의 복사본이 모든 노드로 분산됩니다. EVEN 분산이나 KEY 분산은 테이블 행의 일부를 각 노드에 할당하는 반면 ALL 분산은 테이블이 참여하는 조인마다 모든 행을 공동 배치합니다.

ALL 분산은 필요한 스토리지를 클러스터 노드 수와 곱하기 때문에 데이터를 다수의 테이블에 로드하거나, 업데이트하거나, 삽입하는 데 더 많은 시간이 걸립니다. 따라서 비교적 느리게 이동하는 테이블, 즉 업데이트가 자주 또는 광범위하게 이루어지지 않는 테이블에 한해 적합합니다. 쿼리 중에 작은 테이블을 재분산하는 비용이 적기 때문에 작은 차원 테이블을 DISTSTYLE ALL로 정의하는 이점이 크지 않습니다.

> **Note**
>
> 열에 대한 배포 스타일을 지정하면 Amazon Redshift가 클러스터 수준에서 데이터 배포를 처리합니다. Amazon Redshift는 데이터베이스 객체 내 데이터 분할 개념을 요구하거나 지원하지 않습니다. 따라서 테이블 공간을 생성하거나 테이블 분할 방식을 정의할 필요도 없습니다.

특정 시나리오에서는 이미 생성된 테이블의 분산 스타일을 변경할 수 있습니다. 자세한 내용은 ALTER TABLE 단원을 참조하십시오. 이미 생성된 테이블의 분산 스타일을 변경할 수 없는 시나리오에서는 테이블을 다시 생성하고 전체 복사를 통해 새 테이블을 채우십시오. 자세한 내용은 전체 복사 수행 섹션을 참조하세요.

분산 스타일 보기

테이블의 분산 스타일을 보려면 PG_CLASS_INFO 보기 또는 SVV_TABLE_INFO 보기를 쿼리합니다.

PG_CLASS_INFO의 RELEFFECTIVEDISTSTYLE 열은 테이블의 현재 분산 스타일을 나타냅니다. 테이블에서 자동 분산을 사용하는 경우 RELEFFECTIVEDISTSTYLE은 10, 11 또는 12입니다. 즉, 효과적인 분산 스타일이 AUTO (ALL), AUTO (EVEN) 또는 AUTO (KEY)임을 나타냅니다. 테이블에서 자동 분산을 사용하는 경우 분산 스타일은 처음에 AUTO(ALL)로 표시된 다음 테이블이 커지면 AUTO(EVEN) 또는 AUTO(KEY)로 변경될 수 있습니다.

다음 표는 RELEFFECTIVEDISTSTYLE 열에서 각 값에 따른 분산 스타일을 나타낸 것입니다.

RELEFFECTIVEDISTSTYLE	현재 분산 스타일
0	0
1	키

RELEFFECTIVEDISTSTYLE	현재 분산 스타일
8	ALL
10	AUTO(ALL)
11	AUTO(EVEN)
12	AUTO(KEY)

SVV_TABLE_INFO의 DISTSTYLE 열은 테이블의 현재 분산 스타일을 나타냅니다. 테이블에서 자동 분산을 사용하는 경우 DISTSTYLE은 AUTO(ALL), AUTO(EVEN) 또는 AUTO(KEY)입니다.

다음은 분산 스타일 3개와 자동 분산을 사용해 테이블 4개를 생성한 후 SVV_TABLE_INFO를 쿼리해 분산 스타일을 확인하는 예입니다.

```
create table public.dist_key (col1 int)
diststyle key distkey (col1);

insert into public.dist_key values (1);

create table public.dist_even (col1 int)
diststyle even;

insert into public.dist_even values (1);

create table public.dist_all (col1 int)
diststyle all;

insert into public.dist_all values (1);

create table public.dist_auto (col1 int);

insert into public.dist_auto values (1);

select "schema", "table", diststyle from SVV_TABLE_INFO
where "table" like 'dist%';

        schema   |    table        | diststyle
     ------------+-----------------+------------
      public     | dist_key        | KEY(col1)
```

```
public     | dist_even       | EVEN
public     | dist_all        | ALL
public     | dist_auto       | AUTO(ALL)
```

쿼리 패턴 평가

분산 스타일 선택은 데이터베이스 설계의 한 측면에 불과합니다. 분산 스타일은 전체 시스템의 맥락에서 클러스터 크기, 압축 인코딩 방법, 정렬 키, 테이블 제약 조건 등 다른 중요한 요인과 분산의 밸런스를 유지하면서 고려해야 합니다.

최대한 실제 데이터에 가까운 데이터를 이용하여 시스템을 테스트하십시오.

배포 스타일을 효과적으로 선택하기 위해서는 Amazon Redshift 애플리케이션의 쿼리 패턴을 알고 있어야 합니다. 시스템에서 가장 비용이 높은 쿼리를 식별한 후 이러한 쿼리들의 수요를 기초로 초기 데이터베이스를 설계해야 합니다. 쿼리의 총 비용을 결정하는 요소에는 쿼리를 실행하는 데 걸리는 시간과 쿼리가 소비하는 컴퓨팅 리소스가 포함됩니다. 쿼리 비용을 결정하는 다른 요소는 실행 빈도와 다른 쿼리 및 데이터베이스 작업에 미치는 부정적 영향입니다.

가장 비용이 높은 쿼리에서 사용되는 테이블을 식별한 후 쿼리 런타임 시 이러한 테이블이 어떠한 역할을 하는지 평가합니다. 그리고, 테이블의 조인 및 집계 방식을 고려합니다.

각 테이블의 분산 스타일을 선택할 때는 이번 단원의 지침을 따르십시오. 이미 그렇게 배포 스타일을 선택했다면 테이블을 생성한 후 실제 데이터에 최대한 가까운 데이터를 테이블에 로드합니다. 그런 다음 사용할 것으로 예상되는 쿼리 유형에 대해 테이블을 테스트합니다. 쿼리 실행 계획을 평가하여 조정 필요성을 판단할 수 있습니다. 로드 시간, 스토리지 공간 및 쿼리 런타임을 서로 비교하면서 시스템의 전체 요구 사항 간의 밸런스를 유지합니다.

분산 스타일 지정

이번 단원에서 분산 스타일을 지정하는 데 필요한 고려 사항과 권장 사항은 스타 스키마를 예로 사용합니다. 데이터베이스는 스타 스키마, 스타 스키마의 변형 또는 완전히 다른 스키마를 기초로 설계되기도 합니다. Amazon Redshift는 어떤 스키마를 선택하든 효과적으로 호환되도록 설계되었습니다. 이번 단원에서 언급하는 몇 가지 원칙은 어떤 설계 스키마에든 적용될 수 있습니다.

1. 기본 키와 외래 키를 모든 테이블에 지정하십시오.

   Amazon Redshift는 기본 키 및 외래 키 제약 조건을 강요하지 않지만 쿼리 옵티마이저는 쿼리 계획을 작성할 때 이러한 제약 조건을 사용합니다. 따라서 기본 키와 외래 키를 설정하는 경우에는 애플리케이션이 키의 유효성을 유지해야 합니다.
2. 공통 열을 기준으로 팩트 테이블과 가장 큰 차원 테이블을 분산시키십시오.

테이블 크기뿐만 아니라 가장 공통적인 조인에 참여하는 데이터 세트의 크기를 기준으로 가장 큰 차원을 선택합니다. 테이블이 WHERE 절을 사용하여 공통적으로 필터링되는 경우에는 행의 일부분만 조인에 참여합니다. 이러한 테이블은 더 많은 데이터를 제공하는 작은 크기의 테이블보다 재분산에 미치는 영향이 작습니다. 차원 테이블의 기본 키와 팩트 테이블의 해당 외래 키를 모두 DISTKEY로 지정합니다. 다수의 테이블이 동일한 분산 키를 사용하는 경우에는 모두 팩트 테이블과 함께 배치됩니다. 팩트 테이블은 분산 키가 1개로 제한됩니다. 따라서 다른 키로 조인하는 테이블은 팩트 테이블과 함께 배치되지 않습니다.

3. 나머지 차원 테이블의 분산 키를 지정하십시오.

   다른 테이블과 가장 공통적으로 조인하는 방식에 따라 기본 키 또는 외래 키로 테이블을 분산시킵니다.

4. ALL 분산 사용을 위한 일부 차원 테이블의 변경 여부를 평가하십시오.

   차원 테이블을 팩트 테이블이나 기타 중요한 조인 테이블과 함께 배치할 수 없는 경우에는 전체 테이블을 모든 노드로 분산시켜 쿼리 성능을 크게 높일 수 있습니다. ALL 분산을 사용하면 스토리지 공간 요건이 크게 늘어날 뿐만 아니라 로그 시간 및 유지 관리 작업도 증가합니다. 따라서 ALL 분산을 선택하려면 먼저 모든 인자에 가중치를 반영해야 합니다. 다음 단원에서는 EXPLAIN 계획을 평가하여 ALL 분산에 적합한 테이블 후보를 식별하는 방법에 대해서 설명하겠습니다.

5. 나머지 테이블에 AUTO 분산을 사용합니다.

   테이블이 대부분 비정규화되어 조인에 참여하지 않거나, 혹은 다른 분산 스타일을 명확히 구분하여 선택하지 않는 경우에는 AUTO 분산을 사용합니다.

Amazon Redshift에서 적절한 배포 스타일을 선택하도록 하려면 배포 스타일을 명시적으로 지정하지 않습니다.

쿼리 계획 평가

쿼리 계획을 사용하여 분산 스타일을 최적화하는 데 적합한 테이블 후보를 식별할 수 있습니다.

초기 설계 의사결정을 마친 후에는 테이블을 생성하여 데이터를 로드하고 나서 테이블을 테스트합니다. 테스트 데이터 세트는 실제 데이터에 최대한 가까운 것으로 사용합니다. 비교 기준으로 사용할 로드 시간을 측정합니다.

가장 비용이 높은 쿼리 중에서도 실행하려고 하는 대표적인 쿼리, 특히 조인 및 집계를 사용하는 쿼리를 평가합니다. 다양한 설계 옵션에 따라 런타임을 비교합니다. 런타임을 비교할 때 첫 번째로 실행하는 쿼리는 제외하세요. 첫 번째 런타임에는 컴파일 시간이 포함되어 있기 때문입니다.

DS_DIST_NONE

해당하는 조각이 컴퓨팅 노드에 공동 배치되기 때문에 재분산이 필요 없습니다. 일반적으로 팩트 테이블과 차원 테이블 1개 사이의 조인을 의미하는 DS_DIST_NONE 단계는 한 번만 있습니다.

DS_DIST_ALL_NONE

내부 조인 테이블이 DISTSTYLE ALL을 사용했기 때문에 재분산이 필요 없습니다. 전체 테이블이 모든 노드에 배치됩니다.

DS_DIST_INNER

내부 테이블이 재분산됩니다.

DS_DIST_OUTER

외부 테이블이 재분산됩니다.

DS_BCAST_INNER

전체 내부 테이블의 복사본이 모든 컴퓨팅 노드로 브로드캐스팅됩니다.

DS_DIST_ALL_INNER

외부 테이블이 DISTSTYLE ALL을 사용하기 때문에 내부 테이블 전체가 단일 조각으로 재분산됩니다.

DS_DIST_BOTH

두 테이블 모두 재분산됩니다.

DS_DIST_NONE과 DS_DIST_ALL_NONE은 유효합니다. 두 옵션은 모든 조인이 공동 배치되기 때문에 해당 단계에서 분산이 필요 없었다는 것을 의미합니다.

DS_DIST_INNER는 내부 테이블이 노드로 재분산되기 때문에 비교적 높은 비용의 단계가 될 것이라는 것을 의미합니다. 또한 외부 테이블이 이미 조인 키를 기준으로 적절하게 분산된다는 것을 나타내기도 합니다. 이 옵션을 DS_DIST_NONE으로 변환하려면 내부 테이블의 분산 키를 조인 키로 설정하십시오. 경우에 따라 외부 테이블이 조인 키에 배포되지 않기 때문에 조인 키에 내부 테이블을 배포할 수 없습니다. 이 경우 내부 테이블에 대해 ALL 분포를 사용할지 여부를 평가합니다. 테이블 업데이트가 자주 또는 광범위하게 이루어지지 않는 동시에 높은 재배포 비용을 감당할 정도로 충분히 크다면 배포 스타일을 ALL로 변경하고 다시 테스트합니다. ALL 분산은 로드 시간의 증가를 유발합니다. 따라서 다시 테스트할 때는 평가 요인에 로드 시간도 추가해야 합니다.

DS_DIST_ALL_INNER는 유효하지 않습니다. 이 옵션은 외부 테이블이 DISTSTYLE ALL을 사용하기 때문에 전체 외부 테이블의 복사본이 각 노드에 배치될 수 있도록 내부 테이블 전체가 단일 조각으로

재분산된다는 의미입니다. 이렇게 되면 모든 노드를 통한 병렬 런타임을 이용하는 대신 단일 노드를 통한 조인의 직렬 런타임이라는 비효율적인 결과가 발생합니다. DISTSTYLE ALL은 오직 내부 조인 테이블에서만 사용해야 합니다. 대신에 외부 테이블에서는 분산 키를 지정하거나 EVEN 분산을 사용하십시오.

DS_BCAST_INNER와 DS_DIST_BOTH는 유효하지 않습니다. 일반적으로 테이블이 분산 키를 기준으로 조인되지 않아서 이러한 재분산이 일어납니다. 팩트 테이블에 아직 분산 키가 없는 경우에는 조인 열을 두 테이블의 분산 키로 지정하십시오. 팩트 테이블의 다른 열에 이미 재분산 키가 있는 경우에는 이 조인의 공동 배치를 위한 분산 키 변경이 전체 성능 개선에 도움이 되는지 평가합니다. 외부 테이블의 분산 키 변경이 최적의 선택이 아니라면 DISTSTYLE ALL을 내부 테이블에 지정하여 공동 배치를 사용할 수 있습니다.

다음은 DS_BCAST_INNER 및 DS_DIST_NONE 레이블을 포함한 쿼리 계획의 일부를 나타낸 예입니다.

```
->  XN Hash Join DS_BCAST_INNER  (cost=112.50..3272334142.59 rows=170771 width=84)
        Hash Cond: ("outer".venueid = "inner".venueid)
        ->  XN Hash Join DS_BCAST_INNER  (cost=109.98..3167290276.71 rows=172456
 width=47)
              Hash Cond: ("outer".eventid = "inner".eventid)
              ->  XN Merge Join DS_DIST_NONE  (cost=0.00..6286.47 rows=172456 width=30)
                    Merge Cond: ("outer".listid = "inner".listid)
                    ->  XN Seq Scan on listing  (cost=0.00..1924.97 rows=192497
 width=14)
                    ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=24)
```

DISTSTYLE ALL을 사용하도록 차원 테이블을 변경하고 나면 아래와 같이 동일한 쿼리에 대한 쿼리 계획에서 DS_BCAST_INNER가 DS_DIST_ALL_NONE으로 바뀐 것을 알 수 있습니다. 또한 조인 단계의 상대적 비용에도 커다란 변화가 있습니다. 총 비용은 14142.59이며, 이전 쿼리의 경우 3272334142.59였습니다.

```
->  XN Hash Join DS_DIST_ALL_NONE  (cost=112.50..14142.59 rows=170771 width=84)
        Hash Cond: ("outer".venueid = "inner".venueid)
        ->  XN Hash Join DS_DIST_ALL_NONE  (cost=109.98..10276.71 rows=172456 width=47)
              Hash Cond: ("outer".eventid = "inner".eventid)
              ->  XN Merge Join DS_DIST_NONE  (cost=0.00..6286.47 rows=172456 width=30)
                    Merge Cond: ("outer".listid = "inner".listid)
                    ->  XN Seq Scan on listing  (cost=0.00..1924.97 rows=192497
 width=14)
```

```
                    ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=24)
```

쿼리 계획 예

다음 예는 쿼리 계획을 평가하여 분산 최적화의 기회를 찾는 방법을 나타내고 있습니다.

EXPLAIN 명령과 함께 다음 쿼리를 실행하여 쿼리 계획을 산출합니다.

```
explain
select lastname, catname, venuename, venuecity, venuestate, eventname,
month, sum(pricepaid) as buyercost, max(totalprice) as maxtotalprice
from category join event on category.catid = event.catid
join venue on venue.venueid = event.venueid
join sales on sales.eventid = event.eventid
join listing on sales.listid = listing.listid
join date on sales.dateid = date.dateid
join users on users.userid = sales.buyerid
group by lastname, catname, venuename, venuecity, venuestate, eventname, month
having sum(pricepaid)>9999
order by catname, buyercost desc;
```

TICKIT 데이터베이스에서는 SALES가 팩트 테이블이고, LISTING이 가장 큰 차원 테이블입니다. 두 테이블의 공동 배치를 위해 SALES는 LISTING의 외래 키인 LISTID를 기준으로 분산되고, LISTING은 기본 키인 LISTID를 기준으로 분산됩니다. 다음은 SALES 및 LISTING에 대한 CREATE TABLE 명령을 나타낸 예입니다.

```
create table sales(
 salesid integer not null,
 listid integer not null distkey,
 sellerid integer not null,
 buyerid integer not null,
 eventid integer not null encode mostly16,
 dateid smallint not null,
 qtysold smallint not null encode mostly8,
 pricepaid decimal(8,2) encode delta32k,
 commission decimal(8,2) encode delta32k,
 saletime timestamp,
 primary key(salesid),
 foreign key(listid) references listing(listid),
 foreign key(sellerid) references users(userid),
 foreign key(buyerid) references users(userid),
```

```
 foreign key(dateid) references date(dateid))
        sortkey(listid,sellerid);

create table listing(
 listid integer not null distkey sortkey,
 sellerid integer not null,
 eventid integer not null encode mostly16,
 dateid smallint not null,
 numtickets smallint not null encode mostly8,
 priceperticket decimal(8,2) encode bytedict,
 totalprice decimal(8,2) encode mostly32,
 listtime timestamp,
 primary key(listid),
 foreign key(sellerid) references users(userid),
 foreign key(eventid) references event(eventid),
 foreign key(dateid) references date(dateid));
```

다음 쿼리 계획에서는 SALES와 LISTING의 조인을 위한 Merge Join 단계가 재분산이 필요 없다는 의미의 DS_DIST_NONE을 표시하고 있습니다. 하지만 쿼리 계획 위를 보면 나머지 내부 조인은 쿼리 실행의 일부로서 내부 테이블의 브로드캐스팅을 의미하는 DS_BCAST_INNER를 표시하고 있습니다. 키 분산을 사용할 경우에는 테이블을 한 쌍만 공동 배치할 수 있기 때문에 테이블 5개를 다시 브로드캐스팅해야 합니다.

```
QUERY PLAN
XN Merge  (cost=1015345167117.54..1015345167544.46 rows=1000 width=103)
  Merge Key: category.catname, sum(sales.pricepaid)
  ->  XN Network  (cost=1015345167117.54..1015345167544.46 rows=170771 width=103)
        Send to leader
        ->  XN Sort  (cost=1015345167117.54..1015345167544.46 rows=170771 width=103)
              Sort Key: category.catname, sum(sales.pricepaid)
              ->  XN HashAggregate  (cost=15345150568.37..15345152276.08 rows=170771
 width=103)
                    Filter: (sum(pricepaid) > 9999.00)
                     ->  XN Hash Join DS_BCAST_INNER  (cost=742.08..15345146299.10
 rows=170771 width=103)
                           Hash Cond: ("outer".catid = "inner".catid)
                           ->  XN Hash Join DS_BCAST_INNER
 (cost=741.94..15342942456.61 rows=170771 width=97)
                                 Hash Cond: ("outer".dateid = "inner".dateid)
                                 ->  XN Hash Join DS_BCAST_INNER
 (cost=737.38..15269938609.81 rows=170766 width=90)
                                       Hash Cond: ("outer".buyerid = "inner".userid)
```

```
                                        ->  XN Hash Join DS_BCAST_INNER
(cost=112.50..3272334142.59 rows=170771 width=84)
                                              Hash Cond: ("outer".venueid =
"inner".venueid)
                                              ->  XN Hash Join DS_BCAST_INNER
(cost=109.98..3167290276.71 rows=172456 width=47)
                                                    Hash Cond: ("outer".eventid =
"inner".eventid)
                                                    ->  XN Merge Join DS_DIST_NONE
(cost=0.00..6286.47 rows=172456 width=30)
                                                          Merge Cond: ("outer".listid =
"inner".listid)
                                                          ->  XN Seq Scan on listing
(cost=0.00..1924.97 rows=192497 width=14)
                                                          ->  XN Seq Scan on sales
(cost=0.00..1724.56 rows=172456 width=24)
                                                    ->  XN Hash  (cost=87.98..87.98
rows=8798 width=25)
                                                          ->  XN Seq Scan on event
(cost=0.00..87.98 rows=8798 width=25)
                                              ->  XN Hash  (cost=2.02..2.02 rows=202
width=41)
                                                    ->  XN Seq Scan on venue
(cost=0.00..2.02 rows=202 width=41)
                                        ->  XN Hash  (cost=499.90..499.90 rows=49990
width=14)
                                              ->  XN Seq Scan on users
(cost=0.00..499.90 rows=49990 width=14)
                                  ->  XN Hash  (cost=3.65..3.65 rows=365 width=11)
                                        ->  XN Seq Scan on date  (cost=0.00..3.65
rows=365 width=11)
                              ->  XN Hash  (cost=0.11..0.11 rows=11 width=10)
                                    ->  XN Seq Scan on category  (cost=0.00..0.11 rows=11
width=10)
```

한 가지 해결책으로 DISTSTYLE ALL을 포함하도록 테이블을 변경할 수 있습니다.

```
ALTER TABLE users ALTER DISTSTYLE ALL;
ALTER TABLE venue ALTER DISTSTYLE ALL;
ALTER TABLE category ALTER DISTSTYLE ALL;
ALTER TABLE date ALTER DISTSTYLE ALL;
ALTER TABLE event ALTER DISTSTYLE ALL;
```

다시 EXPLAIN으로 동일한 쿼리를 실행한 후 새로운 쿼리 계획을 검사합니다. 이제 데이터가 DISTSTYLE ALL을 사용하여 모든 노드에 분산되었기 때문에 재분산이 필요 없다는 의미의 DS_DIST_ALL_NONE이 조인에 표시됩니다.

```
QUERY PLAN
XN Merge  (cost=1000000047117.54..1000000047544.46 rows=1000 width=103)
  Merge Key: category.catname, sum(sales.pricepaid)
  ->  XN Network  (cost=1000000047117.54..1000000047544.46 rows=170771 width=103)
        Send to leader
        ->  XN Sort  (cost=1000000047117.54..1000000047544.46 rows=170771 width=103)
              Sort Key: category.catname, sum(sales.pricepaid)
              ->  XN HashAggregate  (cost=30568.37..32276.08 rows=170771 width=103)
                    Filter: (sum(pricepaid) > 9999.00)
                    ->  XN Hash Join DS_DIST_ALL_NONE  (cost=742.08..26299.10
 rows=170771 width=103)
                          Hash Cond: ("outer".buyerid = "inner".userid)
                          ->  XN Hash Join DS_DIST_ALL_NONE  (cost=117.20..21831.99
 rows=170766 width=97)
                                Hash Cond: ("outer".dateid = "inner".dateid)
                                ->  XN Hash Join DS_DIST_ALL_NONE
 (cost=112.64..17985.08 rows=170771 width=90)
                                      Hash Cond: ("outer".catid = "inner".catid)
                                      ->  XN Hash Join DS_DIST_ALL_NONE
 (cost=112.50..14142.59 rows=170771 width=84)
                                            Hash Cond: ("outer".venueid =
 "inner".venueid)
                                            ->  XN Hash Join DS_DIST_ALL_NONE
 (cost=109.98..10276.71 rows=172456 width=47)
                                                  Hash Cond: ("outer".eventid =
 "inner".eventid)
                                                  ->  XN Merge Join DS_DIST_NONE
 (cost=0.00..6286.47 rows=172456 width=30)
                                                        Merge Cond: ("outer".listid =
 "inner".listid)
                                                        ->  XN Seq Scan on listing
 (cost=0.00..1924.97 rows=192497 width=14)
                                                        ->  XN Seq Scan on sales
 (cost=0.00..1724.56 rows=172456 width=24)
                                                  ->  XN Hash  (cost=87.98..87.98
 rows=8798 width=25)
                                                        ->  XN Seq Scan on event
 (cost=0.00..87.98 rows=8798 width=25)
```

```
                                          ->  XN Hash  (cost=2.02..2.02 rows=202
 width=41)
                                                ->  XN Seq Scan on venue
 (cost=0.00..2.02 rows=202 width=41)
                                    ->  XN Hash  (cost=0.11..0.11 rows=11 width=10)
                                          ->  XN Seq Scan on category
 (cost=0.00..0.11 rows=11 width=10)
                               ->  XN Hash  (cost=3.65..3.65 rows=365 width=11)
                                     ->  XN Seq Scan on date  (cost=0.00..3.65
 rows=365 width=11)
                          ->  XN Hash  (cost=499.90..499.90 rows=49990 width=14)
                                ->  XN Seq Scan on users  (cost=0.00..499.90 rows=49990
 width=14)
```

분산 예제

아래 예들은 CREATE TABLE 문에서 정의하는 옵션에 따른 데이터 분산 방식을 나타내고 있습니다.

DISTKEY 예제

TICKIT 데이터베이스의 USERS 테이블 스키마를 살펴봅니다. USERID가 SORTKEY 열과 DISTKEY 열로 정의되어 있습니다.

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'users';

    column     |          type          | encoding | distkey | sortkey
---------------+------------------------+----------+---------+---------
 userid        | integer                | none     | t       |       1
 username      | character(8)           | none     | f       |       0
 firstname     | character varying(30)  | text32k  | f       |       0

...
```

이 테이블에서 USERID를 분산 열로 사용하는 것은 좋은 선택입니다. SVV_DISKUSAGE 시스템 뷰에 대한 쿼리를 실행하면 테이블이 매우 균일하게 분산되어 있는 것을 볼 수 있습니다. 열 번호는 0부터 시작하기 때문에 USERID의 열 번호는 0입니다.

```
select slice, col, num_values as rows, minvalue, maxvalue
from svv_diskusage
where name='users' and col=0 and rows>0
order by slice, col;
```

```
slice| col | rows  | minvalue | maxvalue
-----+-----+-------+----------+----------
0    | 0   | 12496 | 4        | 49987
1    | 0   | 12498 | 1        | 49988
2    | 0   | 12497 | 2        | 49989
3    | 0   | 12499 | 3        | 49990
(4 rows)
```

테이블에 49,990개의 행이 포함되어 있습니다. rows(num_values) 열을 보면 각 조각마다 거의 동일한 수의 행이 포함되어 있는 것을 알 수 있습니다. minvalue 열과 maxvalue 열에는 각 조각마다 값의 범위가 표시되어 있습니다. 각 조각마다 거의 전체 범위의 값이 저장되어 있기 때문에 사용자 ID 범위를 필터링하는 쿼리에 모든 조각이 참여할 가능성이 큽니다.

이번 예에서는 작은 테스트 시스템의 분산에 대해서 설명합니다. 일반적으로 전체 조각 수는 이보다 훨씬 많습니다.

STATE 열을 사용해 공통적으로 조인 또는 분류하는 경우에는 STATE 열을 기준으로 분산시킬 수 있습니다. 다음은 USERS 테이블과 동일한 데이터로 새로운 테이블을 생성하지만 DISTKEY를 STATE 열로 설정할 경우를 보여주는 예입니다. 이 경우 배포는 균등하지 않습니다. 조각 0(13,587 행)에는 조각 3(10,150 행)보다 약 30% 많은 행이 저장되어 있습니다. 이보다 훨씬 큰 테이블에서 이러한 크기의 분산 스큐가 발생하면 쿼리 처리에 부정적인 영향을 미칠 수 있습니다.

```
create table userskey distkey(state) as select * from users;

select slice, col, num_values as rows, minvalue, maxvalue from svv_diskusage
where name = 'userskey' and col=0 and rows>0
order by slice, col;

slice | col | rows  | minvalue | maxvalue
------+-----+-------+----------+----------
    0 |   0 | 13587 |        5 |    49989
    1 |   0 | 11245 |        2 |    49990
    2 |   0 | 15008 |        1 |    49976
    3 |   0 | 10150 |        4 |    49986
(4 rows)
```

DISTSTYLE EVEN 예제

USERS 테이블과 동일한 데이터로 새로운 테이블을 생성하더라도 DISTSTYLE을 EVEN으로 설정하면 행이 항상 균일하게 조각으로 분산됩니다.

```
create table userseven diststyle even as
select * from users;

select slice, col, num_values as rows, minvalue, maxvalue from svv_diskusage
where name = 'userseven' and col=0 and rows>0
order by slice, col;

slice | col | rows  | minvalue | maxvalue
------+-----+-------+----------+----------
    0 |   0 | 12497 |        4 |    49990
    1 |   0 | 12498 |        8 |    49984
    2 |   0 | 12498 |        2 |    49988
    3 |   0 | 12497 |        1 |    49989
(4 rows)
```

하지만 분산이 특정 열을 기준으로 이루어지지 않기 때문에 특히 테이블이 다른 테이블에 조인되는 경우에는 쿼리 처리 성능이 떨어질 수 있습니다. 조인 열에 따른 분산 부재는 종종 효율적으로 실행될 수 있는 유형의 조인 작업에 영향을 미치기도 합니다. 두 테이블이 각각 조인 열을 기준으로 분산 및 정렬되면 조인, 집계, 분류 등의 작업이 최적화됩니다.

DISTSTYLE ALL 예제

USERS 테이블과 동일한 데이터로 새로운 테이블을 생성하더라도 DISTSTYLE을 ALL로 설정하면 모든 행이 각 노드의 첫 번째 조각으로 분산됩니다.

```
select slice, col, num_values as rows, minvalue, maxvalue from svv_diskusage
where name = 'usersall' and col=0 and rows > 0
order by slice, col;

slice | col | rows  | minvalue | maxvalue
------+-----+-------+----------+----------
    0 |   0 | 49990 |        4 |    49990
    2 |   0 | 49990 |        2 |    49990

(4 rows)
```

정렬 키

> **Note**
>
> SORTKEY AUTO로 테이블을 생성하는 것이 좋습니다. 그러면 Amazon Redshift가 자동 테이블 최적화를 사용하여 정렬 키를 선택합니다. 자세한 내용은 자동 테이블 최적화 단원을 참조하십시오. 이 섹션의 나머지 부분에서는 정렬 순서에 대한 세부 정보를 제공합니다.

테이블을 생성하면서 1개 이상의 열을 정렬 키로 정의할 수도 있습니다. 그러면 처음에 데이터를 빈 테이블에 로드할 때 열이 정렬 순서대로 디스크에 저장됩니다. 정렬 키 열에 대한 정보가 쿼리 플래너로 전달되고, 플래너는 이 정보를 사용해 데이터 정렬 방식의 활용 계획을 작성합니다. 자세한 내용은 CREATE TABLE 단원을 참조하십시오. 정렬 키 생성 모범 사례에 대한 자세한 내용은 최상의 정렬 키 선택 섹션을 참조하세요.

정렬은 범위 제한 조건자를 효율적으로 처리할 수 있는 방법입니다. Amazon Redshift는 열 기반 데이터를 1MB 디스크 블록에 저장합니다. 각 블록의 최소 값과 최대 값은 메타데이터로 저장됩니다. 쿼리가 범위 제한 조건자를 사용하는 경우에는 쿼리 프로세서가 테이블 스캔 도중 최소/최대 값을 사용하여 상당수의 블록을 빠르게 건너뛸 수 있습니다. 예를 들어 테이블에 날짜별로 정렬된 5년 치의 데이터를 저장되고 쿼리가 날짜 범위를 한 달로 지정한다고 가정합니다. 이 경우 스캔에서 디스크 블록의 최대 98%를 제거할 수 있습니다. 하지만 데이터가 정렬되어 있지 않다면 스캔해야 하는 디스크 블록의 수가 늘어납니다(모든 블록도 가능).

복합 또는 인터리브 정렬 키를 지정할 수 있습니다. 쿼리 조건자가 순서상 정렬 키 열의 하위 집합인 접두사를 사용할 때는 복합 정렬 키가 더욱 효율적입니다. 인터리브 정렬 키는 정렬 키에서 각 열마다 똑같은 가중치를 부여하기 때문에 쿼리 조건자가 정렬 키를 구성하는 열의 모든 하위 집합을 순서에 상관없이 사용할 수 있습니다.

선택한 정렬 키가 쿼리 성능에 미치는 영향을 알고 싶다면 EXPLAIN 명령을 사용하십시오. 자세한 내용은 쿼리 계획 및 실행 워크플로우 단원을 참조하십시오.

정렬 유형의 정의하려면 CREATE TABLE 또는 CREATE TABLE AS 문에서 INTERLEAVED 또는 COMPOUND 키워드를 사용하십시오. 기본 키워드는 COMPOUND입니다. INSERT, UPDATE 또는 DELETE 작업으로 테이블을 정기적으로 업데이트하는 경우 COMPOUND를 사용하는 것이 좋습니다. 인터리브 정렬 키는 최대 8개까지 열을 사용할 수 있습니다. 데이터와 클러스터 크기에 따라 VACUUM REINDEX는 인터리브된 정렬 키를 분석하기 위한 추가적인 패스를 만들기 때문에 VACUUM FULL보다 상당히 더 많은 시간이 소요됩니다. 인터리브된 정렬은 복합 정렬보다 많은 행을 다시 배열해야 하므로 인터리브된 테이블에서는 정렬 및 병합 작업이 더 오래 걸릴 수 있습니다.

테이블 정렬 키를 확인하려면 SVV_TABLE_INFO 시스템 뷰에 대한 쿼리를 실행하십시오.

주제

- 다차원 데이터 레이아웃 정렬(미리 보기)
- 복합 정렬 키
- 인터리브 정렬 키

다차원 데이터 레이아웃 정렬(미리 보기)

다음은 미리 보기로 출시된 테이블의 다차원 데이터 레이아웃 정렬에 대한 사전 릴리스 설명서입니다. 설명서 및 기능은 모두 변경될 수 있습니다. 이 기능은 테스트 클러스터에만 사용하고 프로덕션 환경에서는 사용하지 않는 것이 좋습니다. 미리 보기 이용 약관은 AWS Service Terms의 Beta Service Participation을 참조하세요.

> **Note**
>
> 이 기능은 미리 보기 클러스터 또는 미리 보기 작업 그룹을 통해서만 사용할 수 있습니다. 미리 보기 클러스터를 만들려면 Amazon Redshift 관리 안내서의 미리 보기 클러스터 생성을 참조하세요. 미리 보기 작업 그룹을 만들려면 Amazon Redshift 관리 안내서의 미리 보기 작업 그룹 만들기를 참조하세요.

다차원 데이터 레이아웃 정렬 키는 워크로드에서 발견되는 반복적인 조건자를 기반으로 하는 일종의 AUTO 정렬 키입니다. 워크로드에 반복적인 조건자가 있는 경우 Amazon Redshift는 반복되는 조건자를 충족하는 데이터 행을 콜로케이션으로 배치하여 테이블 스캔 성능을 개선할 수 있습니다. 다차원 데이터 레이아웃 정렬 키는 테이블의 데이터를 엄격한 열 순서로 저장하는 대신 워크로드에 나타나는 반복적인 조건자를 분석하여 데이터를 저장합니다. 워크로드에서 반복되는 조건자를 두 개 이상 찾을 수 있습니다. 워크로드에 따라 이러한 종류의 정렬 키를 사용하면 많은 조건자의 성능이 향상될 수 있습니다. Amazon Redshift는 AUTO 정렬 키로 정의된 테이블에 이 정렬 키 방법을 사용해야 하는지를 자동으로 결정합니다.

예를 들어 열 순서로 데이터가 정렬된 테이블이 있다고 가정하겠습니다. 워크로드의 조건을 만족하는지 확인하기 위해 많은 데이터 블록을 검사해야 할 수 있습니다. 그러나 데이터를 조건자 순서대로 디스크에 저장하면 쿼리를 충족하기 위해 스캔해야 하는 블록 수가 줄어듭니다. 이 경우에는 다차원 데이터 레이아웃 정렬 키를 사용하는 것이 좋습니다.

쿼리가 다차원 데이터 레이아웃 키를 사용하는지 확인하려면 SYS_QUERY_DETAIL 뷰의 step_attribute 열을 참조하세요. 값이 multi-dimensional일 경우 쿼리에 다차원 데이터 레이아웃이 사용된 것입니다. AUTO 정렬 키로 정의된 테이블이 다차원 데이터 레이아웃을 사용하는지 보려면 SVV_TABLE_INFO 뷰의 sortkey1 열을 참조하세요. 값이 padb_internal_mddl_key_col일 경우 테이블 정렬 키에 다차원 데이터 레이아웃이 사용된 것입니다.

Amazon Redshift가 다차원 데이터 레이아웃 정렬 키를 사용하지 못하게 하려면 SORTKEY AUTO가 아닌 다른 테이블 정렬 키 옵션을 선택하세요. SORTKEY 옵션에 대한 자세한 내용은 CREATE TABLE 섹션을 참조하세요.

복합 정렬 키

복합 키는 정렬 키를 정의할 때 나열되는 모든 열로 구성되며, 나열 순서를 따릅니다. 이 정렬 키는 쿼리의 필터가 필터나 조인 같이 정렬 키의 접두사를 사용하는 조건을 적용할 때 가장 유용합니다. 쿼리가 기본 열을 참조하지 않고 보조 정렬 열에만 의존할 경우에는 복합 정렬의 성능 이점이 줄어듭니다. 이 때는 COMPOUND기 기본 정렬 유형입니다.

복합 정렬 키는 조인 작업과 GROUP BY 및 ORDER BY 작업, 그리고 PARTITION BY와 ORDER BY를 사용하는 창 함수의 속도를 높이는 효과가 있습니다. 예를 들어 해시 조인보다 빠를 때가 많은 병합 조인은 데이터가 조인 열을 기준으로 분산 및 사전 정렬되어야 가능합니다. 그 밖에 압축 성능을 높이는 데도 복합 정렬 키가 효과적입니다.

이미 데이터가 로드되어 정렬까지 마친 테이블에 행을 추가하면 미정렬 영역이 증가하여 성능에 커다란 영향을 미칩니다. 이러한 영향은 테이블이 인터리브 정렬을 사용할 때, 특히 정렬 열에 날짜나 타임스탬프 열 같이 점차 증가하는 데이터가 포함되어 있으면 더욱 커집니다. 따라서 정기적으로, 특히 대용량 데이터를 로드한 후에는 VACUUM 작업을 실행하여 데이터를 다시 정렬 및 분석해야 합니다. 자세한 내용은 정렬되지 않은 리전 크기 줄이기 단원을 참조하십시오. 정리(VACUUM)를 통한 데이터 재정렬을 마친 후에는 ANALYZE 명령을 실행하여 쿼리 플래너의 통계 메타데이터를 업데이트하는 것이 좋습니다. 자세한 내용은 테이블 분석 단원을 참조하십시오.

인터리브 정렬 키

인터리브 정렬 키는 정렬 키에서 각 열, 즉 열의 하위 집합에 똑같은 가중치를 부여합니다. 다수의 쿼리가 다른 열을 필터로 사용하는 경우에는 인터리브 정렬 키를 통해 쿼리의 성능을 높일 수 있는 경우가 종종 있습니다. 그 밖에 쿼리가 보조 정렬 열에서 제한적인 조건자를 사용할 때도 인터리브 정렬이 복합 정렬과 비교하여 쿼리 성능을 크게 높여주는 효과가 있습니다.

> **Important**
>
> 자격 증명 열, 날짜, 타임스탬프와 같이 단순하게 증가하는 속성이 있는 열에서는 인터리브 정렬 키를 사용하지 마십시오.

단, 인터리브 정렬 키를 사용하여 향상되는 성능 이점과 로드 및 정리 시간이 늘어나는 문제를 서로 비교하여 검토해야 합니다.

인터리브 정렬은 WHERE 절에서 1개 이상의 정렬 키를 기준으로 필터링하여 선택의 폭이 매우 제한적인 쿼리를 실행할 때 가장 효과적입니다(예: `select c_name from customer where c_region = 'ASIA'`). 이때는 제한적으로 정렬되는 열의 수와 함께 인터리브 정렬의 이점도 커집니다.

인터리브 정렬은 대용량 테이블에서 더욱 효과적입니다. 각 슬라이스에 정렬이 적용됩니다. 따라서 인터리브 정렬은 테이블이 슬라이스당 여러 개의 1MB 블록을 요구할 만큼 충분히 클 때 가장 효과적입니다. 여기서 쿼리 프로세서는 제한적 조건자를 사용하여 블록을 상당 부분을 건너뛸 수 있습니다. 테이블이 사용하는 블록 수를 확인하려면 STV_BLOCKLIST 시스템 뷰에 대한 쿼리를 실행하십시오.

인터리브 정렬은 단일 열을 기준으로 정렬할 때 열 값에 긴 공통 접두사가 포함되어 있다면 복합 정렬보다 뛰어난 성능을 발휘할 수 있습니다. 예를 들어 URL은 공통적으로 "http://www"로 시작하기 마련입니다. 이때 복합 정렬 키는 접두사에서 사용할 수 있는 문자 수가 제한되어 있기 때문에 키가 중복되는 일이 다수 발생합니다. 하지만 인터리브 정렬은 영역 지도(zone map) 값에 내부 압축 방식을 사용하기 때문에 공통 접두사가 긴 열 값 사이에서도 쉽게 구분할 수 있습니다.

Amazon Redshift 프로비저닝된 클러스터를 Amazon Redshift Serverless로 마이그레이션할 때 Redshift 는 인터리브된 정렬 키와 DISTSTYLE KEY가 있는 테이블을 복합 정렬 키로 변환합니다. DISTSTYLE은 변경되지 않습니다. 배포 스타일에 대한 자세한 내용은 데이터 배포 스타일 사용을 참조하십시오.

VACUUM REINDEX

이미 데이터가 로드되어 정렬까지 마친 테이블에 행을 추가하면 시간이 지나면서 성능이 저하될 수 있습니다. 이러한 성능 저하는 복합 정렬과 인터리브 정렬에 모두 발생하지만 인터리브 테이블에 미치는 영향이 더욱 큽니다. VACUUM이 정렬 순서를 복원하기는 하지만 새로운 인터리브 데이터를 병합하려면 모든 데이터 블록을 수정해야 하는 경우도 있기 때문에 인터리브 테이블일 때 작업 시간이 더욱 오래 걸릴 수 있습니다.

Amazon Redshift는 테이블에 데이터를 처음 로드할 때 정렬 키 열의 값 분산을 분석한 후 해당 정보를 사용하여 정렬 키 열에서 최적의 인터리브를 구합니다. 이후 테이블 크기가 커지면 정렬 키 열의 값 분

산도 특히 날짜 또는 타임스탬프 열에서 변동, 즉 스큐가 일어날 수 있습니다. 스큐가 너무 커지면 성능까지 영향을 받게 됩니다. 이때 정렬 키를 다시 분석하여 성능을 복원하려면 REINDEX 키워드와 함께 VACUUM 명령을 실행하십시오. 단, VACUUM REINDEX는 데이터에 대한 추가 분석이 필요하기 때문에 인터리브 테이블에 대한 표준 VACUUM보다 시간이 오래 걸릴 수 있습니다. KEY 분산 스큐와 마지막 REINDEX 시간에 대한 자세한 내용은 SVV_INTERLEAVED_COLUMNS 시스템 뷰에 대한 쿼리를 실행하여 확인할 수 있습니다.

VACUUM 실행 주기와 VACUUM REINDEX 실행 시점에 대한 자세한 내용은 reindex 실행 여부 결정 섹션을 참조하세요.

테이블 제한 사항

고유성, 프라이머리 키 및 외래 키 제약 조건은 참고용일 뿐 표를 채울 때 Amazon Redshift에 적용되지는 않습니다. 예를 들어 종속성이 있는 테이블에 데이터를 삽입하면 제약 조건을 위반하더라도 삽입이 성공할 수 있습니다. 그래도 기본 키와 외래 키는 계획 힌트로 사용되며, ETL 프로세스 또는 애플리케이션의 다른 프로세스가 무결성을 적용하는 경우에는 선언되어야 합니다.

예를 들어 쿼리 플래너는 특정 통계 계산에서 기본 키와 외래 키를 사용합니다. 하위 쿼리 상관 관계 제거 기술에 영향을 미치는 고유성과 참조 관계를 추론하기 위해서입니다. 이를 통해 많은 수의 조인 순서를 지정하고 중복 조인을 제거할 수 있습니다.

플래너가 이러한 키 관계를 이용하지만 여기에는 Amazon Redshift 테이블의 모든 키가 로드된 상태로 유효하다는 가정을 전제로 합니다. 애플리케이션이 잘못된 외래 키 또는 기본 키를 허용하는 경우에는 일부 쿼리가 부정확한 결과를 반환할 수 있기 때문입니다. 예를 들어 SELECT DISTINCT 쿼리는 기본 키가 고유하지 않을 경우 중복 행을 반환할 수도 있습니다. 유효성이 의심된다면 키 제약 조건을 테이블에 정의하지 마십시오. 그러나 유효성이 보장되는 경우에는 프라이머리 및 외래 키와 고유성 제약 조건을 반드시 선언해야 합니다.

Amazon Redshift는 NOT NULL 열 제약 조건을 적용합니다.

테이블 제약 조건에 대한 자세한 내용은 CREATE TABLE 섹션을 참조하세요. 종속성이 있는 테이블을 삭제하는 방법에 대한 내용은 DROP TABLE 섹션을 참조하세요.

Amazon Redshift에서 데이터 로드

Amazon Redshift 데이터베이스에 데이터를 로드하는 방법에는 여러 가지가 있습니다. 로드하는 데 널리 사용되는 데이터 소스 중 하나는 Amazon S3 파일입니다. 다음 표에는 Amazon S3 소스로 시작하는 데 사용할 수 있는 몇 가지 방법이 요약되어 있습니다.

사용 방법	설명	필요한 경우
COPY 명령	배치 파일 모으기를 실행하여 Amazon S3 파일에서 데이터를 로드합니다. 이 방법은 Amazon Redshift의 병렬 처리 기능을 활용합니다. 자세한 내용은 COPY 명령으로 테이블 로드 단원을 참조하십시오.	배치 파일 모으기를 수동으로 시작하기 위한 기본 데이터 로드 요구 사항이 필요할 때 사용해야 합니다. 이 방법은 주로 사용자 지정 및 서드 파티 파일 모으기 파이프라인이나 일회성 또는 임시 파일 모으기 워크로드에 사용됩니다.
COPY... CREATE JOB 명령 (자동 복사)	추적된 Amazon S3 경로에 새 파일이 생성되면 COPY 명령을 자동으로 실행합니다. 자세한 내용은 Amazon S3 버킷에서 자동으로 파일을 복사하기 위해 S3 이벤트 통합 만들기 단원을 참조하십시오.	Amazon S3에 새 파일이 생성될 때 파일 모으기 파이프라인이 데이터를 자동으로 모아야 하는 경우 사용해야 합니다. Amazon Redshift는 데이터 중복을 방지하기 위해 모은 파일을 추적합니다. 이 방법을 사용하려면 Amazon S3 버킷 소유자의 구성이 필요합니다.
데이터 레이크 쿼리에서 로드	Amazon S3 파일에서 데이터 레이크 쿼리를 실행할 외부 테이블을 생성한 다음 INSERT INTO 명령을 실행하여 데이터 레이크 쿼리의 결과를 로컬 테이블에 로드합니다. 자세한 내용은 Redshift Spectrum용 외부 테이블 단원을 참조하십시오.	다음 시나리오 중 하나에서 사용해야 합니다. • AWS Glue 및 오픈 테이블 형식(예: Apache Iceberg, Apache Hudi 또는 Delta Lake)에서 로드합니다. • 소스 파일을 부분적으로 모아야 합니다(예: 특정 행을

사용 방법	설명	필요한 경우
		모으기 위해 WHERE 절을 실행하는 경우 필요). • 특정 열을 모으거나(예: SELECT 명령 실행) 이동 중에 기본 데이터 변환을 수행(예: 기본 작업 적용 또는 소스 파일의 값에 대한 UDF 직접 호출)하기 위해 더 많은 유연성이 필요합니다.
고려할 수 있는 기타 방법		
스트리밍 수집	스트리밍 모으기를 사용하면 지연 시간이 짧고 빠른 속도로 Amazon Kinesis Data Streams 및 Amazon Managed Streaming for Apache Kafka에서 Amazon Redshift 프로비저닝된 또는 Redshift Serverless 구체화된 뷰로 스트림 데이터를 모을 수 있습니다. 자세한 내용은 Amazon Kinesis Data Streams Streams에서 수집 시작 및 Apache Kafka 소스에서 스트리밍 수집 시작하기 단원을 참조하세요.	데이터를 Amazon S3의 파일로 먼저 스트리밍한 다음 Amazon S3에서 로드하는 사용 사례에서 고려해야 합니다. Amazon S3에 데이터를 보관할 필요가 없는 경우 Amazon Redshift로 데이터를 직접 스트리밍하는 것이 좋을 수 있습니다.
데이터 레이크 쿼리 실행	테이블의 콘텐츠를 로컬 테이블에 모으는 대신 데이터 레이크 테이블에서 직접 쿼리를 실행합니다. 자세한 내용은 Amazon Redshift Spectrum 단원을 참조하십시오.	사용 사례에서 Amazon Redshift의 로컬 테이블 쿼리 성능이 필요하지 않은 경우 사용해야 합니다.

사용 방법	설명	필요한 경우
Amazon Redshift 쿼리 에디터 v2를 사용하여 배치 로드	Amazon Redshift 쿼리 에디터 v2에서 배치 파일 모으기 워크로드를 시각적으로 준비하고 실행할 수 있습니다. 자세한 내용은 Amazon Redshift 관리 가이드의 S3에서 데이터 로드를 참조하세요.	쿼리 에디터 v2에서 COPY 문을 준비하고 시각적 도구를 통해 COPY 문 준비 프로세스를 간소화하려는 경우 사용해야 합니다.
Amazon Redshift 쿼리 에디터 v2를 사용하여 로컬 파일에서 데이터 로드	Amazon S3에 파일을 수동으로 업로드할 필요 없이 데스크톱에서 Amazon Redshift 테이블로 직접 파일을 업로드할 수 있습니다. 자세한 내용은 Amazon Redshift 관리 가이드의 로컬 파일 설정 및 워크플로우에서 데이터 로드를 참조하세요.	일회성 쿼리 목적으로 로컬 컴퓨터에서 빠르게 파일을 로드해야 할 경우 사용해야 합니다. 이 방법을 사용하면 Amazon Redshift 쿼리 에디터 v2가 고객 소유의 Amazon S3 버킷에 파일을 임시로 저장하고 이 Amazon S3 경로를 사용하여 복사 명령을 실행합니다.

COPY 명령은 테이블을 로드하는 가장 효율적인 방법입니다. INSERT 명령을 사용하여 데이터를 테이블에 추가할 수도 있지만 COPY를 사용하는 것보다 효율은 훨씬 떨어집니다. COPY 명령은 여러 데이터 파일 또는 여러 데이터 스트림에서 동시에 읽을 수 있습니다. Amazon Redshift는 Amazon Redshift 노드에 워크로드를 할당하고 노드 슬라이스 전체에 데이터 배포 및 행 정렬을 포함한 로드 작업을 병렬로 수행합니다.

> **Note**
>
> Amazon Redshift Spectrum 외부 테이블은 읽기 전용입니다. 외부 테이블로 복사 또는 삽입할 수 없습니다.

다른 AWS 리소스에 있는 데이터에 액세스하려면 이러한 리소스에 액세스하고 데이터 액세스에 필요한 작업을 수행할 권한이 Amazon Redshift에 있어야 합니다. AWS Identity and Access Management(IAM)를 사용하면 Amazon Redshift 리소스와 데이터에 대한 사용자의 액세스를 제한할 수 있습니다.

초기 데이터 로드 후 상당한 양의 데이터를 추가, 수정 또는 삭제하는 경우, VACUUM 명령을 실행해 데이터를 재구성하고 삭제 후 공간을 회수하는 후속 작업을 수행해야 합니다. 또한 ANALYZE 명령을 실행하여 테이블 통계를 업데이트해야 합니다.

주제

COPY 명령으로 테이블 로드

COPY 명령은 Amazon Redshift 대량 병렬 처리(MPP) 아키텍처를 활용하여 Amazon S3의 파일, DynamoDB 테이블 또는 하나 이상의 원격 호스트의 텍스트 출력에서 병렬로 데이터를 읽고 로드합니다.

COPY 명령의 모든 옵션을 익히기 전에 Amazon S3 데이터를 로드하는 기본 옵션을 익히는 것이 좋습니다. Amazon Redshift 시작 가이드에서는 COPY 명령을 통해 기본 IAM 역할을 사용하여 Amazon S3 데이터를 로드하는 간단한 방법을 보여줍니다. 자세한 내용은 4단계: Amazon S3에서 Amazon Redshift로 데이터 로드를 참조하세요.

> **Note**
>
> 대량의 데이터를 로드하려면 COPY 명령을 사용하는 것이 가장 좋습니다. 테이블을 채우는 데 개별적인 INSERT문을 사용하는 방식은 엄청나게 느릴 수 있습니다. 데이터가 다른 Amazon Redshift 데이터베이스 테이블에 이미 있는 경우에는, INSERT INTO ... SELECT 또는 CREATE TABLE AS를 사용하여 성능을 개선할 수 있습니다. 자세한 내용은 INSERT 또는 CREATE TABLE AS 섹션을 참조하세요.

다른 AWS 리소스에서 데이터를 로드하려면 리소스에 액세스하고 필요한 작업을 수행할 권한이 Amazon Redshift에 있어야 합니다.

COPY 명령을 사용하여 테이블로 데이터를 로드하는 권한을 부여하거나 취소하려면 INSERT 권한을 부여하거나 취소합니다.

데이터를 Amazon Redshift 테이블로 로드하려면 올바른 형식이어야 합니다. 이 섹션에서는 로드 전에 데이터를 준비 및 확인하고 실행 전에 COPY 문을 확인하는 방법을 살펴봅니다.

파일의 정보를 보호하기 위해 Amazon S3 버킷으로 업로드하기 전에 데이터 파일을 암호화할 수 있습니다. COPY는 로드를 수행하면서 데이터를 복호화합니다. 사용자에게 임시 보안 자격 증명을 제공하여 로드 데이터에 대한 액세스를 제한할 수도 있습니다. 임시 보안 자격 증명은 유효 기간이 짧고 만료 후 재사용이 불가능하기 때문에 보안을 강화하는 효과가 있습니다.

Amazon Redshift는 압축되지 않고 구분된 데이터를 빠르게 로드할 수 있도록 COPY에 특성이 내장되어 있습니다. 그러나 파일 업로드 시간을 절약하기 위해 gzip, lzop 또는 bzip2를 사용하여 파일을 압축할 수 있습니다.

COPY 쿼리에 ESCAPE, REMOVEQUOTES 및 FIXEDWIDTH와 같은 키워드가 있으면 압축되지 않은 데이터의 자동 분할이 지원되지 않습니다. 하지만 CSV 키워드는 지원됩니다.

AWS 클라우드 내에서 전송 중인 데이터 보안을 위해 Amazon Redshift는 COPY, UNLOAD, 백업 및 복원 작업을 수행할 때 하드웨어 가속화 SSL을 사용하여 Amazon S3 또는 Amazon DynamoDB와 통신합니다.

Amazon DynamoDB 테이블에서 직접 테이블을 로드하는 경우 사용하는 Amazon DynamoDB 프로비저닝 처리량의 양을 제어하는 옵션이 있습니다.

또는 COPY 명령이 입력 데이터를 분석하여 로드 프로세스의 일환으로 최적의 압축 인코딩을 테이블에 자동으로 적용하도록 할 수도 있습니다.

주제

- 좁은 테이블의 스토리지 최적화
- 열 기본값 로드
- 데이터 로드 문제 해결

자격 증명 및 액세스 권한

Amazon S3, Amazon DynamoDB, Amazon EMR, Amazon EC2 등의 다른 AWS 리소스를 사용하여 데이터를 로드 또는 언로드하려면 리소스에 액세스하고 데이터 액세스에 필요한 작업을 수행할 권한이 Amazon Redshift에 있어야 합니다. 예를 들어 Amazon S3에서 데이터를 로드하려면 COPY에 버킷에 대한 LIST 액세스 권한과 버킷 객체에 대한 GET 액세스 권한이 있어야 합니다.

리소스에 액세스할 수 있는 권한을 얻으려면 Amazon Redshift가 인증되어야 합니다. 역할 기반 액세스 제어 또는 키 기반 액세스 제어를 선택할 수 있습니다. 이 단원에서는 이 두 가지 방법의 개요를 살펴봅니다. 자세한 내용과 예는 다른 AWS 리소스에 대한 액세스 권한 섹션을 참조하세요.

역할 기반 액세스 제어

역할 기반 액세스 제어를 통해 Amazon Redshift는 AWS Identity and Access Management(IAM) 역할을 임시로 수임합니다. 그런 다음 역할에 부여되는 권한에 따라 Amazon Redshift가 필요한 AWS 리소스에 액세스할 수 있습니다.

역할 기반 액세스 제어는 AWS 자격 증명을 보호하는 것은 물론이고 AWS 리소스와 민감한 사용자 데이터에 대한 액세스를 더욱 안전하게, 그리고 세분화하여 제어할 수 있다는 점에서 더욱 바람직합니다.

역할 기반 액세스 제어를 사용하려면 먼저 Amazon Redshift 서비스 역할 유형을 사용하여 IAM 역할을 생성한 후 데이터 웨어하우스에 연결해야 합니다. 이 역할에는 최소한 COPY, UNLOAD 및 CREATE LIBRARY 작업을 위한 IAM 권한에 나열된 권한이 있어야 합니다. IAM 역할을 생성하여 클러스터에 연결하는 단계는 Amazon Redshift 관리 가이드의 Amazon Redshift 클러스터가 AWS 서비스에 액세스할 수 있도록 IAM 역할 생성 섹션을 참조하세요.

Amazon Redshift 관리 콘솔, CLI 또는 API를 사용하여 역할을 클러스터에 추가하거나, 클러스터와 연결된 역할을 확인할 수 있습니다. 자세한 내용은 Amazon Redshift 관리 가이드의 IAM 역할을 사용하여 COPY 및 UNLOAD 작업 권한 부여 섹션을 참조하세요.

IAM 역할을 생성하면 IAM이 생성된 역할의 Amazon 리소스 이름(ARN)을 반환합니다. IAM 역할을 사용하여 COPY 명령을 실행하려면 IAM_ROLE 파라미터 또는 CREDENTIALS 파라미터를 사용하여 역할 ARN을 제공합니다.

다음 COPY 명령 예는 역할 `MyRedshiftRole`과 함께 IAM_ROLE 파라미터를 인증에 사용합니다.

```
COPY customer FROM 's3://amzn-s3-demo-bucket/mydata'
IAM_ROLE 'arn:aws:iam::12345678901:role/MyRedshiftRole';
```

AWS 사용자는 최소한 COPY, UNLOAD 및 CREATE LIBRARY 작업을 위한 IAM 권한에 나열된 권한이 있어야 합니다.

키 기반 액세스 제어

키 기반 액세스 제어의 경우 데이터가 포함된 AWS 리소스에 액세스할 권한이 부여된 사용자에게 액세스 키 ID와 비밀 액세스 키를 제공합니다.

> **Note**
>
> 액세스 키 ID와 비밀 액세스 키를 일반 텍스트 형태로 입력하기보다는 IAM 역할을 인증에 사용하는 것을 강력 권장합니다. 키 기반 액세스 제어를 선택하는 경우에는 절대 AWS 계정(루트) 자격 증명을 사용하지 마세요. 항상 IAM 사용자를 먼저 생성한 후 해당 사용자의 액세스 키 ID와 비밀 액세스 키를 입력합니다. IAM 사용자를 생성하는 단계는 AWS 계정의 IAM 사용자 생성을 참조하세요.

입력 데이터 준비

입력 데이터가 데이터를 수신할 테이블 열과 호환되지 않는 경우, COPY 명령이 실패합니다.

입력 데이터가 유효한지 확인하려면 다음 지침을 사용하십시오.

- 데이터에는 최대 4바이트 길이의 UTF-8 문자만 포함될 수 있습니다.
- CHAR 및 VARCHAR 문자열이 해당 열의 길이보다 길지 않은지 확인합니다. VARCHAR 문자열은 문자가 아니라 바이트로 측정됩니다. 따라서 가령 각각 4바이트를 차지하는 중국어의 4자 문자열은 VARCHAR(16) 열이 필요합니다.
- 멀티바이트 문자는 VARCHAR 열에만 사용할 수 있습니다. 멀티바이트 문자의 길이가 4바이트를 넘지 않는지 확인합니다.
- CHAR 열의 데이터에 1바이트 문자만 포함되어 있는지 확인합니다.
- 레코드에서 마지막 필드를 표시하기 위해 특수 문자나 구문을 포함시키지 마십시오. 이 필드는 구분 기호일 수 있습니다.
- 데이터에 NUL(UTF-8 0000)이라고도 하는 null 종결자 또는 이진 0(0x000)이 포함된 경우, COPY 명령에서 NULL AS 옵션을 사용하여 이들 문자를 NULLS로 CHAR 또는 VARCHAR 열에 로드할 수 있

습니다. `null as '\0'` 또는 `null as '\000'`. NULL AS를 사용하지 않는 경우, null 종결자로 인해 COPY가 실패합니다.

- 문자열에 구분 기호와 삽입된 줄 바꿈 같은 특수 문자가 포함된 경우, COPY 명령과 함께 ESCAPE 옵션을 사용합니다.
- 작은따옴표와 큰따옴표의 짝이 모두 맞는지 확인합니다.
- 부동 소수점 문자열이 12.123 같은 표준 부동 소수점 형식이거나 1.0E4 같은 지수 형식인지 확인합니다.
- 모든 타임스탬프 및 날짜 문자열이 DATEFORMAT 및 TIMEFORMAT 문자열의 사양을 따르는지 확인합니다. 기본 타임스탬프 형식은 YYYY-MM-DD hh:mm:ss이고 기본 날짜 형식은 YYYY-MM-DD입니다.
- 개별 데이터 형식의 경계와 제한에 대한 자세한 내용은 데이터 타입 섹션을 참조하세요. 멀티바이트 문자 오류에 대한 자세한 내용은 멀티바이트 문자 로드 오류 섹션을 참조하세요.

Amazon S3에서 데이터 로드

COPY 명령은 Amazon Redshift 대량 병렬 처리(MPP) 아키텍처를 활용하여 Amazon S3 버킷에 있는 하나 이상의 파일에서 병렬로 데이터를 읽고 로드합니다. 파일이 압축된 경우 데이터를 여러 파일로 분할하여 병렬 처리를 최대한 활용할 수 있습니다. (이 규칙에는 예외가 있습니다. 자세한 내용은 데이터 파일 로드에서 설명합니다.) 또한 테이블에서 분산 키를 설정하면 병렬 처리를 최대한 활용할 수 있습니다. 기본 키에 대한 자세한 내용은 쿼리 최적화를 위한 데이터 배포 섹션을 참조하세요.

데이터는 행당 한 줄씩 대상 테이블로 로드됩니다. 데이터 파일의 필드는 순서대로 왼쪽에서 오른쪽으로 테이블 열에 일치됩니다. 데이터 파일의 필드는 고정 너비이거나 문자 구분일 수 있습니다. 기본 구분자는 파이프(|)입니다. 기본적으로 모든 테이블 열이 로드되지만 필요할 경우, 쉼표로 분리된 열 목록을 정의할 수 있습니다. COPY 명령에서 지정된 열 목록에 포함되지 않은 테이블 열은 기본값으로 로드됩니다. 자세한 내용은 열 기본값 로드 단원을 참조하십시오.

주제

- 압축 및 비압축 파일에서 데이터 로드
- COPY와 함께 사용할 파일을 Amazon S3에 업로드
- COPY 명령을 사용하여 Amazon S3에서 로드

압축 및 비압축 파일에서 데이터 로드

압축된 데이터를 로드할 때 각 테이블의 데이터를 여러 파일로 분할하는 것이 좋습니다. 압축되지 않은 구분된 데이터를 로드할 때 COPY 명령은 대규모 병렬 처리(MPP) 및 스캔 범위를 사용하여 Amazon S3 버킷의 대용량 파일에서 데이터를 로드합니다.

여러 압축 파일에서 데이터 로드

압축된 데이터가 있는 경우 각 테이블의 데이터를 여러 파일로 분할하는 것이 좋습니다. COPY 명령은 여러 파일에서 병렬로 데이터를 로드할 수 있습니다. 집합에 공통 접두사 또는 접두사 키를 지정하거나 매니페스트 파일에서 파일을 명시적으로 나열하여 여러 파일을 로드할 수 있습니다.

데이터를 파일로 분할할 때는 파일 수가 클러스터에 있는 조각 수의 배수가 되도록 하십시오. 이렇게 하면 Amazon Redshift가 데이터를 슬라이스 간에 균일하게 나눌 수 있습니다. 노드당 조각 수는 클러스터의 노드 크기에 따라 달라집니다. 예를 들어 각각의 dc2.large 컴퓨팅 노드에는 2개의 조각이 있고 각각의 dc2.8xlarge 컴퓨팅 노드에는 16개의 조각이 있습니다. 각 노드 크기의 슬라이스 수에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 클러스터 및 노드 정보 섹션을 참조하세요.

각 노드는 병렬 쿼리 실행에 참여하여 조각에 대한 데이터 분산 작업을 최대한 균일하게 실행합니다. 2개의 dc2.large 노드가 포함된 클러스터가 있는 경우 데이터를 4개의 파일 또는 4의 배수로 분할할 수 있습니다. Amazon Redshift는 워크로드를 분리할 때 파일 크기를 고려하지 않습니다. 따라서 압축 후 파일 크기가 1MB~1GB로 대략 같은지 확인해야 합니다.

로드 파일을 식별하기 위해 객체 접두사를 사용하려면 공통 접두사를 사용해 각 파일의 이름을 지정합니다. 예를 들어 `venue.txt` 파일은 다음과 같이 4개의 파일로 분할할 수 있습니다.

```
venue.txt.1
venue.txt.2
venue.txt.3
venue.txt.4
```

버킷의 폴더에 여러 파일을 넣고 폴더 이름을 접두사로 지정하면 COPY가 해당 폴더의 모든 파일을 로드합니다. 매니페스트 파일을 사용하여 로드될 파일을 명시적으로 나열하면 서로 다른 버킷이나 폴더에 있는 파일도 로드할 수 있습니다.

매니페스트 파일에 대한 자세한 내용은 Example: COPY from Amazon S3 using a manifest 섹션을 참조하세요.

압축되지 않은 구분된 파일에서 데이터 로드

압축되지 않은 구분된 데이터를 로드할 때 COPY 명령은 Amazon Redshift의 대규모 병렬 처리(MPP) 아키텍처를 사용합니다. Amazon Redshift는 병렬로 작동하는 조각을 자동으로 사용하여 Amazon S3 버킷의 대용량 파일에서 데이터 범위를 로드합니다. 병렬 로드가 발생하려면 파일을 구분해야 합니다. 예를 들어 파이프로 구분됩니다. CSV 파일에는 COPY 명령을 사용한 자동 병렬 데이터 로드도 사용할 수 있습니다. 테이블에서 배포 키를 설정하여 병렬 처리를 활용할 수도 있습니다. 기본 키에 대한 자세한 내용은 쿼리 최적화를 위한 데이터 배포 섹션을 참조하세요.

COPY 쿼리에 ESCAPE, REMOVEQUOTES 및 FIXEDWIDTH 키워드가 포함된 경우 자동 병렬 데이터 로드가 지원되지 않습니다.

하나 이상의 파일에서 대상 테이블로 데이터가 행당 한 줄씩 로드됩니다. 데이터 파일의 필드는 순서대로 왼쪽에서 오른쪽으로 테이블 열에 일치됩니다. 데이터 파일의 필드는 고정 너비이거나 문자 구분일 수 있습니다. 기본 구분자는 파이프(|)입니다. 기본적으로 모든 테이블 열이 로드되지만 필요할 경우, 쉼표로 분리된 열 목록을 정의할 수 있습니다. COPY 명령에서 지정된 열 목록에 포함되지 않은 테이블 열은 기본값으로 로드됩니다. 자세한 내용은 열 기본값 로드 단원을 참조하십시오.

데이터가 압축되지 않고 분리된 경우 다음 일반 프로세스를 따라 Amazon S3에서 데이터를 로드합니다.

1. Amazon S3에 파일을 업로드합니다.
2. COPY 명령을 실행하여 테이블을 로드합니다.
3. 데이터가 올바로 로드되었는지 확인합니다.

COPY 명령의 예는 COPY 예 단원을 참조하세요. Amazon Redshift에 로드된 데이터에 대한 자세한 내용은 STL_LOAD_COMMITS 및 STL_LOAD_ERRORS 시스템 테이블을 확인하세요.

각각에 포함된 노드와 슬라이스에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 클러스터 및 노드 정보 섹션을 참조하세요.

COPY와 함께 사용할 파일을 Amazon S3에 업로드

Amazon S3에 텍스트 파일을 업로드할 때 취할 수 있는 몇 가지 접근 방식이 있습니다.

- 압축 파일이 있는 경우 Amazon Redshift에서 병렬 처리를 최대한 활용하려면 대용량 파일을 분할하는 것이 좋습니다.
- 반면에 COPY는 압축되지 않은 텍스트로 구분된 대용량 파일 데이터를 자동으로 분할하여 병렬 처리를 용이하게 하고 대용량 파일의 데이터를 효과적으로 배포합니다.

데이터 파일을 보관할 Amazon S3 버킷을 생성한 다음 이 버킷에 데이터 파일을 업로드합니다. 버킷 생성 및 파일 업로드에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 Amazon S3 버킷을 사용한 작업을 참조하세요.

> **Important**
>
> 데이터 파일을 보관하는 Amazon S3 버킷은 REGION 옵션을 사용하여 Amazon S3 버킷이 위치한 리전을 지정하지 않는 한 클러스터와 동일한 AWS 리전에서 생성해야 합니다.

S3 IP 범위가 허용 목록에 추가되었는지 확인합니다. 필요한 S3 IP 범위에 대한 자세한 내용은 네트워크 격리를 참조하세요.

Amazon S3 콘솔을 사용하여 버킷을 생성할 때 리전을 선택하거나 Amazon S3 API 또는 CLI를 사용하여 버킷을 생성할 때 엔드포인트를 지정하여 특정 리전에서 Amazon S3 버킷을 생성할 수 있습니다.

데이터 로드 후 Amazon S3에 올바른 파일이 존재하는지 확인합니다.

주제

- 데이터 일관성 관리
- Amazon S3에 암호화된 데이터 업로드
- 버킷에 올바른 파일이 존재하는지 확인

데이터 일관성 관리

Amazon S3는 모든 AWS 리전의 Amazon S3 버킷에서 COPY, UNLOAD, INSERT(외부 테이블), CREATE EXTERNAL TABLE AS 및 Amazon Redshift Spectrum 작업에 강력한 쓰기 후 읽기 일관성을 제공합니다. 또한 Amazon S3 Select, Amazon S3 액세스 제어 목록, Amazon S3 객체 태그, 객체 메타데이터(예: HEAD 객체)에 대한 읽기 작업은 매우 일관적입니다. 데이터 일관성에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 Amazon S3 데이터 일관성 모델을 참조하세요.

Amazon S3에 암호화된 데이터 업로드

Amazon S3는 서버 측 암호화와 클라이언트 측 암호화를 모두 지원합니다. 이 항목에서는 서버 측 암호화와 클라이언트 측 암호화의 차이를 살펴보고, Amazon Redshift에서 클라이언트 측 암호화를 사용하는 단계를 설명합니다. 서버 측 암호화는 Amazon Redshift에 투명합니다.

서버 측 암호화

서버 측 암호화는 저장된 데이터 암호화입니다. 즉, Amazon S3는 데이터를 업로드할 때 데이터를 암호화하고 사용자가 데이터에 액세스할 때 데이터를 복호화합니다. COPY 명령을 사용하여 테이블을 로드하면 Amazon S3에서 서버 측 암호화된 객체나 암호화되지 않은 객체에서 로드하는 방식은 차이가 없습니다. 서버 측 암호화에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 서버 측 암호화 사용을 참조하세요.

클라이언트측 암호화

클라이언트 측 암호화에서는 클라이언트 애플리케이션이 데이터 암호화, 암호화 키, 관련 도구를 관리합니다. 클라이언트 측 암호화를 사용하여 Amazon S3 버킷에 데이터를 업로드한 다음 ENCRYPTED 옵션 및 프라이빗 암호화 키와 함께 COPY 명령을 사용하여 데이터를 로드하면 보안을 강화할 수 있습니다.

봉투 암호화를 사용하여 데이터를 암호화합니다. 봉투 암호화의 경우, 애플리케이션만이 모든 암호화를 처리합니다. 클라이언트 측 마스터 키 및 암호화되지 않은 데이터는 절대 AWS로 전송되지 않기 때문에 암호화 키를 안전하게 관리하는 것이 중요합니다. 암호화 키를 분실하면 데이터 암호를 해제할 수 없으며 AWS에서 암호화 키를 복구할 수 없습니다. 봉투 암호화는 빠른 대칭 암호화의 성능을 결합하는 동시에 비대칭 키를 사용하는 키 관리가 제공하는 보다 높은 수준의 보안을 유지합니다. 일회용 대칭 키(봉투 대칭 키)가 Amazon S3 암호화 클라이언트에 의해 생성되어 데이터를 암호화한 다음 이 키가 루트 키에 의해 암호화되고 Amazon S3에 데이터와 함께 저장됩니다. 로드 중 Amazon Redshift가 데이터에 액세스하면 암호화된 대칭 키가 검색되어 실제 키를 사용해 복호화된 다음 데이터가 복호화됩니다.

Amazon Redshift에서 Amazon S3 클라이언트 측 암호화 데이터로 작업하려면 사용하는 추가 요구 사항과 함께 Amazon Simple Storage Service 사용 설명서의 클라이언트측 암호화를 사용하여 데이터 보호에 약술된 단계를 따릅니다.

- 대칭 암호화 - AWS SDK for Java `AmazonS3EncryptionClient` 클래스는 대칭 키 암호화에 기반하는 앞서 설명한 봉투 암호화를 사용합니다. 클라이언트 측 암호화된 데이터를 업로드하려면 이 클래스를 사용하여 Amazon S3 클라이언트를 생성합니다.
- 256비트 AES 루트 대칭 키 - 루트 키는 봉투(envelope) 키를 암호화합니다. 루트 키를 `AmazonS3EncryptionClient` 클래스의 인스턴스에 전달합니다. 데이터를 Amazon Redshift에 복사할 때 필요하므로 이 키를 저장합니다.
- 암호화된 봉투(envelope) 키를 저장하기 위한 객체 메타데이터 - 기본적으로 Amazon S3는 봉투(envelope) 키를 `AmazonS3EncryptionClient` 클래스에 대한 객체 메타데이터로 저장합니다. 객체 메타데이터로 저장된 암호화된 엔벌로프 키는 해독 프로세스에서 사용됩니다.

> **Note**
>
> 처음으로 암호화 API를 사용할 때 암호 암호화 오류 메시지가 표시되는 경우 사용 중인 버전의 JDK에 암호화 및 암호 해독 변환에 대한 최대 키 길이를 128비트로 제한하는 Java Cryptography Extension(JCE) Jurisdiction Policy File이 있을 수 있습니다. 이 문제를 해결하는 방법에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 AWS Java용 SDK를 사용하여 클라이언트 측 암호화 지정을 참조하세요.

COPY 명령을 사용하여 클라이언트 측에서 암호화된 파일을 Amazon Redshift 테이블에 로드하는 방법은 Amazon S3에서 암호화된 데이터 파일 로드 섹션을 참조하세요.

예: 클라이언트 측에서 암호화된 데이터 업로드

AWS Java용 SDK를 사용하여 클라이언트 측 암호화 데이터를 업로드하는 방법의 예는 Amazon Simple Storage Service 사용 설명서의 클라이언트 측 암호화를 사용하여 데이터 보호를 참조하세요.

두 번째 옵션은 데이터를 Amazon Redshift에 업로드하기 위해 클라이언트 측 암호화 과정에서 선택해야 하는 설정을 보여줍니다. 구체적으로 이 예는 객체 메타데이터를 사용하여 암호화된 엔벌로프 키를 저장하고 256비트 AES 루트 대칭 키를 사용하는 방법을 보여줍니다.

이 예에서는 AWS Java용 SDK를 사용하여 256비트 AES 대칭 루트 키를 생성하고 이를 파일에 저장하는 예제 코드를 제공합니다. 그런 다음 먼저 클라이언트 측에서 샘플 데이터를 암호화하는 S3 암호화 클라이언트를 사용하여 Amazon S3에 객체를 업로드합니다. 이 예에서는 객체를 다운로드하고 데이터가 같은지도 확인합니다.

버킷에 올바른 파일이 존재하는지 확인

파일을 Amazon S3 버킷에 업로드한 후에는 버킷의 콘텐츠를 나열하여 올바른 파일이 모두 존재하고 원치 않는 파일이 없는지 확인하는 것이 좋습니다. 예를 들어 버킷 `amzn-s3-demo-bucket`에 `venue.txt.back` 파일이 보관된 경우, 해당 파일은 아마도 다음 명령에 의해 의도치 않게 로드될 것입니다.

```
COPY venue FROM 's3://amzn-s3-demo-bucket/venue' … ;
```

구체적으로 어떤 파일이 로드되는지 제어하려면 매니페스트 파일을 사용하여 데이터 파일을 명시적으로 나열합니다. 매니페스트 파일 사용에 대한 자세한 내용은 COPY 명령의 copy_from_s3_manifest_file 옵션과 COPY 예의 Example: COPY from Amazon S3 using a manifest를 참조하십시오.

버킷의 콘텐츠 나열에 대한 자세한 내용은 Amazon S3 Developer Guide의 Listing Object Keys 섹션을 참조하세요.

COPY 명령을 사용하여 Amazon S3에서 로드

COPY 명령을 사용하여 Amazon S3의 데이터 파일에서 병렬로 테이블을 로드합니다. Amazon S3 객체 접두사를 사용하거나 매니페스트 파일을 사용하여 로드할 파일을 지정할 수 있습니다.

접두사를 사용하여 로드할 파일을 지정하는 구문은 다음과 같습니다.

```
COPY <table_name> FROM 's3://<bucket_name>/<object_prefix>'
authorization;
```

매니페스트 파일은 로드할 데이터 파일을 나열하는 JSON 형식 파일입니다. 매니페스트 파일을 사용하여 로드할 파일을 지정하는 구문은 다음과 같습니다.

```
COPY <table_name> FROM 's3://<bucket_name>/<manifest_file>'
authorization
MANIFEST;
```

로드할 테이블이 데이터베이스에 이미 존재하고 있어야 합니다. 테이블 생성에 대한 자세한 내용은 SQL 참조의 CREATE TABLE 섹션을 참조하세요.

authorization 값은 Amazon Redshift가 Amazon S3 객체에 액세스하는 데 필요한 AWS 권한 부여를 제공합니다. 필요한 권한에 대한 자세한 내용은 COPY, UNLOAD 및 CREATE LIBRARY 작업을 위한 IAM 권한 섹션을 참조하세요. 선호되는 인증 방법은 IAM_ROLE 파라미터를 지정하고 IAM 역할의 Amazon 리소스 이름(ARN)에 필요한 권한을 제공하는 것입니다. 자세한 정보는 역할 기반 액세스 제어을 참조하십시오.

IAM_ROLE 파라미터를 사용하여 인증하려면 다음 구문을 참조하여 *<aws-account-id>* 및 *<role-name>*을 바꿉니다.

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-name>'
```

다음 예는 IAM 역할을 사용한 인증을 보여 줍니다.

```
COPY customer
FROM 's3://amzn-s3-demo-bucket/mydata'
```

```
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

기타 옵션에 대한 자세한 내용은 권한 부여 파라미터 섹션을 참조하세요.

실제로 테이블을 로드하지 않고 데이터를 확인하려면 COPY 명령과 함께 NOLOAD 옵션을 사용하십시오.

다음 예는 venue.txt라는 이름의 파일에서 파이프로 구분된 데이터의 첫 몇 행을 보여 줍니다.

```
1|Toyota Park|Bridgeview|IL|0
2|Columbus Crew Stadium|Columbus|OH|0
3|RFK Stadium|Washington|DC|0
```

파일을 Amazon S3에 업로드하기 전에 COPY 명령이 병렬 처리를 사용해 로드할 수 있도록 파일을 여러 파일로 분할합니다. 이때 파일 수는 클러스터 조각 수의 승수가 되어야 합니다. 이때는 압축 후 파일 크기가 1MB~1GB로 거의 같아질 수 있도록 로딩 데이터 파일을 분할합니다. 자세한 내용은 압축 및 비압축 파일에서 데이터 로드 단원을 참조하십시오.

예를 들어 venue.txt 파일은 다음과 같이 4개의 파일로 분할할 수 있습니다.

```
venue.txt.1
venue.txt.2
venue.txt.3
venue.txt.4
```

다음 COPY 명령은 Amazon S3 버킷 amzn-s3-demo-bucket에서 접두사 'venue'가 있는 데이터 파일에서 파이프로 구분된 데이터를 사용해 VENUE 테이블을 로드합니다.

> **Note**
>
> 다음 예의 Amazon S3 버킷 amzn-s3-demo-bucket은 존재하지 않습니다. 기존 Amazon S3 버킷의 실제 데이터를 사용하는 샘플 COPY 명령은 샘플 데이터 로드 섹션을 참조하세요.

```
COPY venue FROM 's3://amzn-s3-demo-bucket/venue'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
DELIMITER '|';
```

키 접두사 'venue'가 있는 Amazon S3 객체가 존재하지 않는 경우 로드가 실패합니다.

주제

매니페스트를 사용하여 데이터 파일 지정

매니페스트를 사용하여 COPY 명령이 데이터 로드에 필요한 파일만 모두 로드하도록 할 수 있습니다. 매니페스트를 이용하면 다른 버킷, 다른 리전이나 접두사가 다른 파일에서 파일을 불러올 수 있습니다. COPY 명령에 대해 객체 경로를 제공하는 대신 로드할 파일을 명시적으로 나열하는 JSON 형식 텍스트 파일의 이름을 제공합니다. 매니페스트의 URL은 접두사만이 아니라 파일의 버킷 이름과 전체 객체 경로를 지정해야 합니다.

매니페스트 파일에 대한 자세한 내용은 매니페스트를 사용한 데이터 파일 지정을 참조하십시오.

다음 예는 버킷이 서로 다르고 날짜 스탬프로 시작되는 파일 이름을 가진 파일을 로드하는 JSON을 보여 줍니다.

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket1/2013-10-04-custdata", "mandatory":true},
    {"url":"s3://amzn-s3-demo-bucket1/2013-10-05-custdata", "mandatory":true},
    {"url":"s3://amzn-s3-demo-bucket2/2013-10-04-custdata", "mandatory":true},
    {"url":"s3://amzn-s3-demo-bucket2/2013-10-05-custdata", "mandatory":true}
  ]
}
```

선택 사항인 mandatory 플래그는 파일이 없을 때 의 오류 반환 여부를 결정합니다. mandatory의 기본값은 false입니다. 필수 설정에 상관없이 파일이 발견되지 않으면 COPY는 종료됩니다.

다음은 앞 예에서 사용한 cust.manifest 매니페스트를 이용해 명령을 실행하는 예입니다.

```
COPY customer
FROM 's3://amzn-s3-demo-bucket/cust.manifest'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
MANIFEST;
```

UNLOAD에 의해 생성된 매니페스트 사용

MANIFEST 파라미터를 사용하여 UNLOAD 작업에 의해 생성된 매니페스트에는 COPY 작업에 필요하지 않은 키가 포함될 수 있습니다. 예를 들어 다음 UNLOAD매니페스트는 Amazon Redshift Spectrum 외부 테이블에 필요하고, ORC 또는 Parquet 파일 형식의 데이터 파일을 로드하는 데 필요한 meta 키를 포함합니다. meta 키에는 파일의 실제 크기(바이트 단위)인 값과 함께 content_length 키가 포함되어 있습니다. COPY 작업에는 url 키와 선택 사항인 mandatory 키만 필요합니다.

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket/unload/manifest_0000_part_00", "meta":
 { "content_length": 5956875 }},
    {"url":"s3://amzn-s3-demo-bucket/unload/unload/manifest_0001_part_00", "meta":
 { "content_length": 5997091 }}
 ]
}
```

매니페스트 파일에 대한 자세한 내용은 Example: COPY from Amazon S3 using a manifest 단원을 참조하세요.

Amazon S3에서 압축된 데이터 파일 로드

gzip, lzop 또는 bzip2를 사용해 압축된 데이터 파일을 로드하려면 해당 옵션인 GZIP, LZOP 또는 BZIP2를 포함시킵니다.

예를 들어 다음 명령은 lzop을 사용하여 압축한 파일에서 로드합니다.

```
COPY customer FROM 's3://amzn-s3-demo-bucket/customer.lzo'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
DELIMITER '|' LZOP;
```

> **Note**
>
> lzop 압축으로 데이터 파일을 압축하고 --filter 옵션을 사용하는 경우 COPY 명령은 이를 지원하지 않습니다.

Amazon S3에서 고정 너비 데이터 로드

고정 너비 데이터 파일은 데이터의 각 열의 길이가 균일합니다. 고정 너비 데이터 파일에서 각 필드의 길이와 위치는 정확히 동일합니다. 고정 너비 데이터 파일의 문자 데이터(CHAR 및 VARCHAR)의 경

우, 너비를 균일하게 유지하기 위해 자리 표시자로 선행 공백 또는 후행 공백을 포함시켜야 합니다. 정수의 경우, 자리 표시자로 선행 제로를 사용해야 합니다. 고정 너비 데이터 파일에는 열을 분리하는 구분 기호가 없습니다.

고정 너비 데이터 파일을 기존 테이블로 로드하려면 COPY 명령에서 FIXEDWIDTH 파라미터를 사용합니다. 데이터가 올바로 로드되려면 테이블 사양이 fixedwidth_spec의 값과 일치해야 합니다.

파일에서 테이블로 고정 너비 데이터를 로드하려면 다음 명령을 실행합니다.

```
COPY table_name FROM 's3://amzn-s3-demo-bucket/prefix'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
FIXEDWIDTH 'fixedwidth_spec';
```

fixedwidth_spec 파라미터는 콜론으로 분리된 각 열의 식별자와 각 열 너비가 포함된 문자열입니다. **column:width** 쌍은 쉼표로 구분됩니다. 식별자는 숫자, 문자 또는 숫자와 문자의 조합 중 어떤 것이든 가능합니다. 식별자는 테이블 자체와 아무 관련이 없으므로 사양에는 테이블과 동일한 순서로 열이 포함되어야 합니다.

다음 2개의 예에 나오는 사양은 동일하며, 첫 번째 사양은 숫자 식별자를 사용하고 두 번째 사양은 문자열 식별자를 사용합니다.

```
'0:3,1:25,2:12,3:2,4:6'
```

```
'venueid:3,venuename:25,venuecity:12,venuestate:2,venueseats:6'
```

다음 예는 상기 사양을 사용하여 VENUE 테이블에 로드할 수 있는 고정 너비 샘플 데이터를 보여 줍니다.

```
1  Toyota Park               Bridgeview  IL0
2  Columbus Crew Stadium     Columbus    OH0
3  RFK Stadium               Washington  DC0
4  CommunityAmerica Ballpark Kansas City KS0
5  Gillette Stadium          Foxborough  MA68756
```

다음 COPY 명령은 이 데이터 세트를 VENUE 테이블로 로드합니다.

```
COPY venue
```

```
FROM 's3://amzn-s3-demo-bucket/data/venue_fw.txt'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
FIXEDWIDTH 'venueid:3,venuename:25,venuecity:12,venuestate:2,venueseats:6';
```

Amazon S3에서 멀티바이트 데이터 로드

데이터에 중국어나 키릴 문자 같이 ASCII가 아닌 멀티바이트 문자가 포함되어 있는 경우에는 VARCHAR 열에 데이터를 로드해야 합니다. VARCHAR 데이터 형식은 4바이트 UTF-8 문자를 지원하지만 CHAR 데이터 형식에서는 1바이트 ASCII 문자만 허용되기 때문입니다. Amazon Redshift 테이블에 5바이트 이상의 문자는 로드할 수 없습니다. CHAR 및 VARCHAR에 대한 자세한 내용은 데이터 타입 섹션을 참조하세요.

입력 파일이 어떤 인코딩을 사용하는지 확인하려면 Linux *file* 명령을 사용합니다.

```
$ file ordersdata.txt
ordersdata.txt: ASCII English text
$ file uni_ordersdata.dat
uni_ordersdata.dat: UTF-8 Unicode text
```

Amazon S3에서 암호화된 데이터 파일 로드

COPY 명령을 사용하면 서버 측 암호화나 클라이언트 측 암호화 또는 이 둘을 모두 사용하여 Amazon S3에 업로드된 데이터 파일을 로드할 수 있습니다.

COPY 명령이 지원하는 Amazon S3 암호화 유형은 다음과 같습니다.

- Amazon S3 관리형 키를 사용한 서버 측 암호화(SSE-S3)
- AWS KMS keys(SSE-KMS)를 사용한 서버 측 암호화 지정
- 클라이언트 측 대칭 루트 키를 사용한 클라이언트 측 암호화

COPY 명령이 지원하지 않는 Amazon S3 암호화 유형은 다음과 같습니다.

- 고객 제공 키를 사용한 서버 측 암호화(SSE-C)
- AWS KMS key를 사용한 클라이언트 측 암호화
- 고객 제공 비대칭 루트 키를 사용한 클라이언트 측 암호화

Amazon S3 암호화에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 서버 측 암호화를 사용하여 데이터 보호 및 클라이언트 측 암호화를 사용하여 데이터 보호를 참조하세요.

UNLOAD 명령은 SSE-S3를 사용하여 자동으로 파일을 암호화합니다. 고객 관리형 대칭 키와 함께 SSE-KMS 또는 클라이언트측 암호화를 사용하여 언로드할 수도 있습니다. 자세한 내용은 암호화된 데이터 파일 언로드 단원을 참조하세요.

COPY 명령은 SSE-S3 및 SSE-KMS를 사용하여 암호화된 파일을 자동으로 인식하고 로드합니다. ENCRYPTED 옵션을 지정하고 키 값을 제공하여 클라이언트 측 대칭 루트 키를 사용해 암호화된 파일을 로드할 수 있습니다. 자세한 내용은 Amazon S3에 암호화된 데이터 업로드 단원을 참조하십시오.

클라이언트 측에서 암호화된 데이터 파일을 로드하려면 MASTER_SYMMETRIC_KEY 파라미터를 사용하여 루트 키 값을 제공하고 ENCRYPTED 옵션을 포함시킵니다.

```
COPY customer FROM 's3://amzn-s3-demo-bucket/encrypted/customer'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
MASTER_SYMMETRIC_KEY '<root_key>'
ENCRYPTED
DELIMITER '|';
```

gzip, lzop 또는 bzip2 압축된 암호화된 데이터 파일을 로드하려면 루트 키 값 및 ENCRYPTED 옵션과 함께 GZIP, LZOP 또는 BZIP2를 포함시킵니다.

```
COPY customer FROM 's3://amzn-s3-demo-bucket/encrypted/customer'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
MASTER_SYMMETRIC_KEY '<root_key>'
ENCRYPTED
DELIMITER '|'
GZIP;
```

Amazon EMR에서 데이터 로드

COPY 명령을 사용하면 클러스터의 Hadoop 분산 파일 시스템(HDFS)에 고정 너비 파일, 문자로 구분된 파일, CSV 파일 또는 JSON 형식 파일 형식으로 텍스트 파일을 쓰도록 구성된 Amazon EMR 클러스터에서 병렬로 데이터를 로드할 수 있습니다.

Amazon EMR에서 데이터를 로드하기 위한 프로세스

이 섹션에서는 Amazon EMR 클러스터에서 데이터를 로드하는 프로세스를 단계별로 살펴봅니다. 다음 단원에서는 각 단계에서 해야 할 일을 자세히 설명합니다.

- 1단계: IAM 권한 구성

Amazon EMR 클러스터를 생성하고 Amazon Redshift COPY 명령을 실행하는 사용자에게는 필요한 권한이 있어야 합니다.

- 2단계: Amazon EMR 클러스터 생성

  Hadoop 분산 파일 시스템(HDFS)으로 텍스트 파일을 출력하도록 클러스터를 구성합니다. Amazon EMR 클러스터 ID와 클러스터의 메인 퍼블릭 DNS(클러스터를 호스팅하는 Amazon EC2 인스턴스의 엔드포인트)가 필요합니다.

- 3단계: Amazon Redshift 클러스터 퍼블릭 키와 클러스터 노드 IP 주소 검색

  퍼블릭 키를 사용하면 Amazon Redshift 클러스터 노드가 호스트와의 SSH 연결을 설정할 수 있습니다. 각 클러스터 노드의 IP 주소를 사용하여 이러한 IP 주소를 사용하는 Amazon Redshift 클러스터로부터의 액세스를 허용하도록 호스트 보안 그룹을 구성합니다.

- 4단계: 각각의 Amazon EC2 호스트의 권한 부여된 키 파일에 Amazon Redshift 클러스터 퍼블릭 키 추가

  호스트가 Amazon Redshift 클러스터를 인식하고 SSH 연결을 수락하도록 호스트의 권한 부여된 키 파일에 Amazon Redshift 클러스터 퍼블릭 키를 추가합니다.

- 5단계: Amazon Redshift 클러스터의 모든 IP 주소를 수락하도록 호스트 구성

  Amazon EMR 인스턴스의 보안 그룹을 수정하여 Amazon Redshift IP 주소를 수락하도록 입력 규칙을 추가합니다.

- 6단계: COPY 명령을 실행하여 데이터 로드

  Amazon Redshift 데이터베이스에서 COPY 명령을 실행하여 Amazon Redshift 테이블로 데이터를 로드합니다.

1단계: IAM 권한 구성

Amazon EMR 클러스터를 생성하고 Amazon Redshift COPY 명령을 실행하는 사용자에게는 필요한 권한이 있어야 합니다.

IAM 권한을 구성하려면

1. Amazon EMR 클러스터를 생성할 사용자에 대해 다음의 권한을 추가합니다.

```
ec2:DescribeSecurityGroups
ec2:RevokeSecurityGroupIngress
```

```
ec2:AuthorizeSecurityGroupIngress
redshift:DescribeClusters
```

2. COPY 명령을 실행할 IAM 역할 또는 사용자에 대해 다음의 권한을 추가합니다.

```
elasticmapreduce:ListInstances
```

3. 다음 권한을 Amazon EMR 클러스터의 IAM 역할에 추가합니다.

```
redshift:DescribeClusters
```

2단계: Amazon EMR 클러스터 생성

COPY 명령은 Amazon EMR Hadoop 분산 파일 시스템(HDFS)에서 데이터를 로드합니다. Amazon EMR 클러스터를 생성할 때 클러스터의 HDFS로 데이터 파일을 출력하도록 클러스터를 구성합니다.

Amazon EMR 클러스터를 생성하려면

1. Amazon Redshift 클러스터와 동일한 AWS 리전에 Amazon EMR 클러스터를 생성합니다.

   Amazon Redshift 클러스터가 VPC에 있는 경우 Amazon EMR 클러스터는 동일한 VPC 그룹에 있어야 합니다. Amazon Redshift 클러스터가 EC2-Classic 모드를 사용하는 경우(즉, VPC에 없는 경우), Amazon EMR 클러스터도 EC2-Classic 모드를 사용해야 합니다. 자세한 내용은 Amazon Redshift 관리 가이드의 Virtual Private Cloud(VPC)의 클러스터 관리 섹션을 참조하세요.
2. 클러스터의 HDFS로 데이터 파일을 출력하도록 클러스터를 구성합니다. HDFS 파일 이름에는 별표(*)나 물음표(?)가 포함되면 안 됩니다.

   > **Important**
   >
   > 파일 이름에 별표(*)나 물음표(?)가 있으면 안 됩니다.

3. COPY 명령이 실행되는 동안 클러스터를 계속 사용할 수 있도록 Amazon EMR 클러스터 구성에서 자동 종료(Auto-terminate) 옵션에 대해 아니요(No)를 지정합니다.

   > **Important**
   >
   > COPY가 완료되기 전에 변경되거나 삭제된 데이터 파일이 있다면 예상치 못한 결과가 나오거나 COPY 작업이 실패할 수 있습니다.

4. 클러스터 ID와 메인 퍼블릭 DNS(클러스터를 호스팅하는 Amazon EC2 인스턴스의 엔드포인트)를 적어 둡니다. 이후 단계에서 이 정보를 사용하게 됩니다.

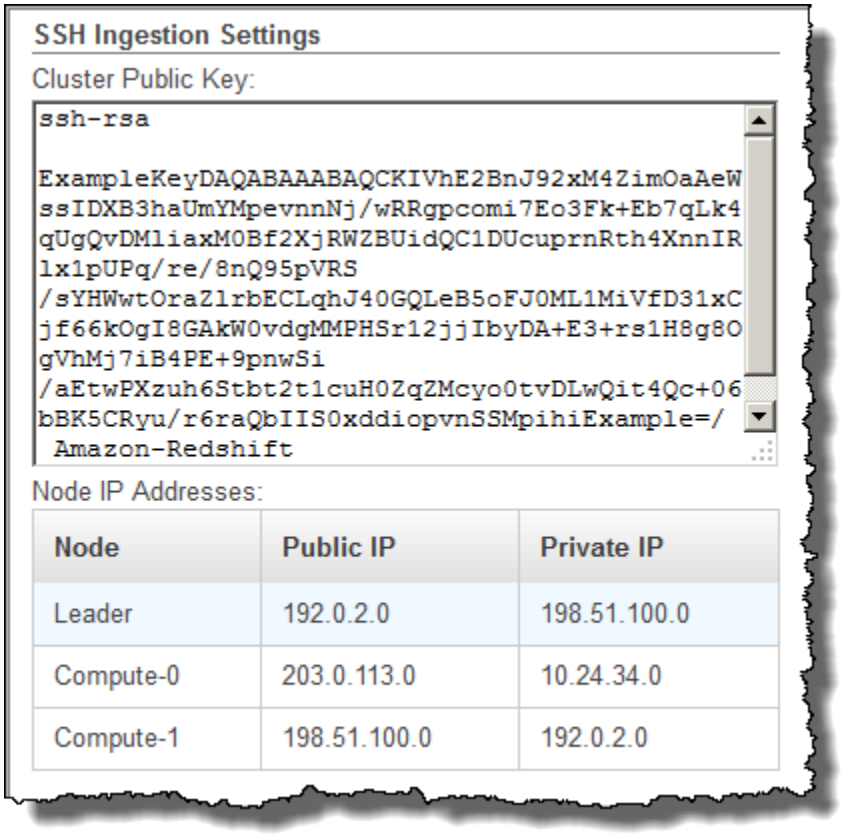
3단계: Amazon Redshift 클러스터 퍼블릭 키와 클러스터 노드 IP 주소 검색

각 클러스터 노드의 IP 주소를 사용하여 이러한 IP 주소를 사용하는 Amazon Redshift 클러스터로부터의 액세스를 허용하도록 호스트 보안 그룹을 구성합니다.

콘솔을 사용하여 Amazon Redshift 클러스터 퍼블릭 키와 클러스터의 클러스터 노드 IP 주소를 검색하려면

1. Amazon Redshift 관리 콘솔에 액세스합니다.
2. 탐색 창에서 Clusters(클러스터) 링크를 선택합니다.
3. 목록에서 해당 인스턴스를 선택합니다.
4. SSH 처리 설정 그룹을 찾습니다.

   클러스터 퍼블릭 키와 노드 IP 주소를 적어 둡니다. 이후 단계에서 사용하게 됩니다.


SSH Ingestion Settings

Cluster Public Key:

```
ssh-rsa

ExampleKeyDAQABAAABAQCKIVhE2BnJ92xM4ZimOaAeW
ssIDXB3haUmYMpevnnNj/wRRgpcomi7Eo3Fk+Eb7qLk4
qUgQvDMliaxM0Bf2XjRWZBUidQC1DUcuprnRth4XnnIR
lx1pUPq/re/8nQ95pVRS
/sYHWwtOraZlrbECLqhJ40GQLeB5oFJ0ML1MiVfD31xC
jf66kOgI8GAkW0vdgMMPHSr12jjIbyDA+E3+rs1H8g8O
gVhMj7iB4PE+9pnwSi
/aEtwPXzuh6Stbt2t1cuH0ZqZMcyo0tvDLwQit4Qc+06
bBK5CRyu/r6raQbIIS0xddiopvnSSMpihiExample=/
 Amazon-Redshift
```

Node IP Addresses:

Node	Public IP	Private IP
Leader	192.0.2.0	198.51.100.0
Compute-0	203.0.113.0	10.24.34.0
Compute-1	198.51.100.0	192.0.2.0

3단계의 프라이빗 IP 주소를 사용하여 Amazon Redshift로부터의 연결을 수락하도록 Amazon EC2 호스트를 구성합니다.

Amazon Redshift CLI를 사용하여 클러스터 퍼블릭 키와 클러스터의 클러스터 노드 IP 주소를 검색하려면 describe-clusters 명령을 실행합니다. 예:

```
aws redshift describe-clusters --cluster-identifier <cluster-identifier>
```

응답에는 ClusterPublicKey 값과 프라이빗 및 퍼블릭 IP 주소 목록이 포함되며, 다음과 비슷합니다.

```
{
    "Clusters": [
        {
            "VpcSecurityGroups": [],
            "ClusterStatus": "available",
            "ClusterNodes": [
                {
                    "PrivateIPAddress": "10.nnn.nnn.nnn",
                    "NodeRole": "LEADER",
                    "PublicIPAddress": "10.nnn.nnn.nnn"
                },
                {
                    "PrivateIPAddress": "10.nnn.nnn.nnn",
                    "NodeRole": "COMPUTE-0",
                    "PublicIPAddress": "10.nnn.nnn.nnn"
                },
                {
                    "PrivateIPAddress": "10.nnn.nnn.nnn",
                    "NodeRole": "COMPUTE-1",
                    "PublicIPAddress": "10.nnn.nnn.nnn"
                }
            ],
            "AutomatedSnapshotRetentionPeriod": 1,
            "PreferredMaintenanceWindow": "wed:05:30-wed:06:00",
            "AvailabilityZone": "us-east-1a",
            "NodeType": "dc2.large",
            "ClusterPublicKey": "ssh-rsa AAAABexamplepublickey...Y3TAl Amazon-
Redshift",
             ...
             ...
}
```

Amazon Redshift API를 사용하여 클러스터 퍼블릭 키와 클러스터의 클러스터 노드 IP 주소를 검색하려면 `DescribeClusters` 작업을 사용합니다. 자세한 내용은 Amazon Redshift CLI Guide의 describe-clusters 또는 Amazon Redshift API Guide의 DescribeClusters 섹션을 참조하세요.

4단계: 각각의 Amazon EC2 호스트의 권한 부여된 키 파일에 Amazon Redshift 클러스터 퍼블릭 키 추가

호스트가 Amazon Redshift를 인식하고 SSH 연결을 수락하도록 모든 Amazon EMR 클러스터 노드에 대해 각 호스트의 권한 부여된 키 파일에 클러스터 퍼블릭 키를 추가합니다.

호스트의 권한 부여된 키 파일에 Amazon Redshift 클러스터 퍼블릭 키를 추가하려면

1. SSH 연결을 사용하여 호스트에 액세스합니다.

   SSH를 사용하여 인스턴스에 연결에 대한 자세한 내용은 Amazon EC2 User Guide의 Connect to Your Instance 섹션을 참조하세요.
2. 콘솔 또는 CLI 응답 텍스트에서 Amazon Redshift 퍼블릭 키를 복사합니다.
3. 퍼블릭 키의 내용을 복사하여 호스트의 `/home/<ssh_username>/.ssh/authorized_keys` 파일에 붙여 넣습니다. 접두사 "`ssh-rsa`"와 접미사 `Amazon-Redshift`"를 포함하여 완전한 문자열을 포함시킵니다. 예:

   ```
   ssh-rsa AAAACTP3isxgGzVWoIWpbVvRCOzYdVifMrh… uA70BnMHCaMiRdmvsDOedZDOedZ Amazon-Redshift
   ```

5단계: Amazon Redshift 클러스터의 모든 IP 주소를 수락하도록 호스트 구성

호스트 인스턴스로의 인바운드 트래픽을 허용하려면 보안 그룹을 편집하고 각각의 Amazon Redshift 클러스터 노드에 하나의 Inbound 규칙을 추가합니다. 유형에서 포트 22의 TCP 프로토콜이 포함된 SSH를 선택합니다. 소스에서 3단계: Amazon Redshift 클러스터 퍼블릭 키와 클러스터 노드 IP 주소 검색에서 가져온 Amazon Redshift 클러스터 노드 프라이빗 IP 주소를 입력합니다. Amazon EC2 보안 그룹에 규칙 추가에 대한 자세한 내용은 Amazon EC2 User Guide의 Authorizing Inbound Traffic for Your Instances 섹션을 참조하세요.

6단계: COPY 명령을 실행하여 데이터 로드

COPY 명령을 실행하여 Amazon EMR 클러스터에 연결하고 데이터를 Amazon Redshift 테이블에 로드합니다. Amazon EMR 클러스터는 COPY 명령이 완료될 때까지 계속 실행되어야 합니다. 예를 들어 자동 종료되도록 클러스터를 구성하지 마십시오.

> **Important**
>
> COPY가 완료되기 전에 변경되거나 삭제된 데이터 파일이 있다면 예상치 못한 결과가 나오거나 COPY 작업이 실패할 수 있습니다.

COPY 명령에서 Amazon EMR 클러스터 ID와 HDFS 파일 경로 및 파일 이름을 지정합니다.

```
COPY sales
FROM 'emr://myemrclusterid/myoutput/part*' CREDENTIALS
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

와일드카드 문자 별표(*)와 물음표(?)를 파일 이름 인수의 일부로 사용할 수 있습니다. 예를 들어 `part*`는 파일 `part-0000`, `part-0001` 등등을 로드합니다. 폴더 이름만 지정하면 COPY가 폴더의 모든 파일을 로드하려고 합니다.

> **Important**
>
> 와일드카드 문자를 사용하거나 폴더 이름만을 사용하는 경우, 원치 않는 파일이 로드되지 않는지 확인하십시오. 그럴 경우, COPY 명령이 실패합니다. 예를 들어 일부 프로세스에서는 로그 파일이 출력 폴더로 로드되는 경우도 있습니다.

원격 호스트에서 데이터 로드

COPY 명령을 사용하면 Amazon EC2 인스턴스나 다른 컴퓨터 같은 하나 이상의 원격 호스트에서 병렬로 파일을 로드할 수 있습니다. COPY는 SSH를 사용하여 원격 호스트에 연결하고 원격 호스트에서 명령을 실행해 텍스트 출력을 생성합니다.

원격 호스트는 Amazon EC2 Linux 인스턴스이거나 SSH 연결을 허용하도록 구성된 다른 Unix 또는 Linux 컴퓨터일 수 있습니다. 이 안내서에서는 원격 호스트가 Amazon EC2 인스턴스라고 가정합니다. 다른 컴퓨터의 절차가 다를 경우, 안내서에서 차이점을 명시합니다.

Amazon Redshift는 여러 호스트에 연결할 수 있으며 각 호스트에 대해 여러 SSH 연결을 열 수 있습니다. Amazon Redshift는 각 연결을 통해 고유한 명령을 전송하여 호스트의 표준 출력에 대한 텍스트 출력을 생성합니다. 그러면 Amazon Redshift가 텍스트 파일을 읽을 때 텍스트 출력을 읽습니다.

시작하기 전 준비 사항

시작하기 전에 다음을 갖추고 있어야 합니다.

- SSH를 사용해 연결할 수 있는, Amazon EC2 인스턴스와 같은 하나 이상의 호스트 시스템.
- 호스트의 데이터 원본.

  Amazon Redshift 클러스터가 호스트에서 실행되어 텍스트 출력을 생성한다는 명령을 내립니다. 클러스터가 호스트에 연결된 후 COPY 명령이 명령을 실행하고, 호스트의 표준 출력에서 텍스트를 읽으며, 데이터를 병렬로 Amazon Redshift 테이블에 로드합니다. 텍스트 출력은 COPY 명령이 수집할 수 있는 형식이어야 합니다. 자세한 내용은 입력 데이터 준비 섹션을 참조하세요.
- 컴퓨터에서 호스트에 액세스합니다.

  Amazon EC2 인스턴스의 경우 SSH 연결을 사용하여 호스트에 액세스합니다. Amazon Redshift 클러스터의 퍼블릭 키를 호스트의 권한 부여된 키 파일에 추가하려면 호스트에 액세스해야 합니다.
- 실행 중인 Amazon Redshift 클러스터입니다.

  클러스터를 시작하는 방법에 대한 자세한 내용은 Amazon Redshift 시작 안내서를 참조하세요.

데이터 로드 프로세스

이 단원에서는 원격 호스트에서 데이터를 로드하는 프로세스를 단계별로 살펴봅니다. 다음 단원에서는 각 단계에서 해야 할 일을 자세히 설명합니다.

- 1단계: 클러스터 퍼블릭 키와 클러스터 노드 IP 주소 검색

  퍼블릭 키를 사용하면 Amazon Redshift 클러스터 노드가 원격 호스트와의 SSH 연결을 설정할 수 있습니다. 각 클러스터 노드의 IP 주소를 사용하여 이러한 IP 주소를 사용하는 Amazon Redshift 클러스터로부터의 액세스를 허용하도록 호스트 보안 그룹 또는 방화벽을 구성합니다.
- 2단계: 호스트의 권한 부여된 키 파일에 Amazon Redshift 클러스터 퍼블릭 키 추가

  호스트가 Amazon Redshift 클러스터를 인식하고 SSH 연결을 수락하도록 호스트의 권한 부여된 키 파일에 Amazon Redshift 클러스터 퍼블릭 키를 추가합니다.
- 3단계: Amazon Redshift 클러스터의 모든 IP 주소를 수락하도록 호스트를 구성

  Amazon EC2의 경우 인스턴스의 보안 그룹을 수정하여 Amazon Redshift IP 주소를 수락하도록 입력 규칙을 추가합니다. 다른 호스트의 경우 Amazon Redshift 노드가 원격 호스트와의 SSH 연결을 설정할 수 있도록 방화벽을 수정합니다.
- 4단계: 호스트의 퍼블릭 키 가져오기

  필요한 경우 Amazon Redshift가 호스트 식별에 퍼블릭 키를 사용해야 한다고 지정할 수 있습니다. 퍼블릭 키를 찾아 텍스트를 매니페스트 파일에 복사해야 합니다.

- 5단계: 매니페스트 파일 생성

  매니페스트는 Amazon Redshift가 호스트에 연결해 데이터를 가져오는 데 필요한 세부 정보가 포함된 JSON 형식의 텍스트 파일입니다.

- 6단계: Amazon S3 버킷에 매니페스트 파일 업로드

  Amazon Redshift는 매니페스트를 읽고 이 정보를 사용해 원격 호스트에 연결합니다. Amazon S3 버킷이 Amazon Redshift 클러스터와 동일한 리전에 속하지 않는 경우에는 REGION 옵션을 사용하여 데이터가 위치한 리전을 지정해야 합니다.

- 7단계: COPY 명령을 실행하여 데이터 로드

  Amazon Redshift 데이터베이스에서 COPY 명령을 실행하여 Amazon Redshift 테이블로 데이터를 로드합니다.

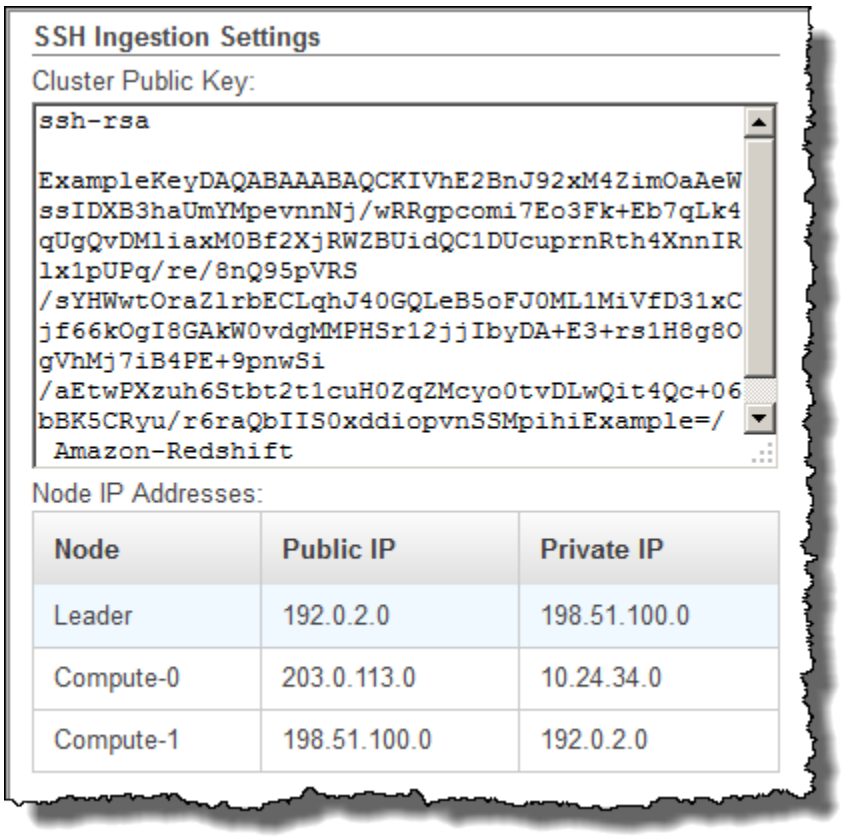
1단계: 클러스터 퍼블릭 키와 클러스터 노드 IP 주소 검색

각 클러스터 노드의 IP 주소를 사용하여 이러한 IP 주소를 사용하는 Amazon Redshift 클러스터로부터의 액세스를 허용하도록 호스트 보안 그룹을 구성합니다.

콘솔을 사용하여 클러스터 퍼블릭 키와 클러스터의 클러스터 노드 IP 주소를 검색하려면

1. Amazon Redshift 관리 콘솔에 액세스합니다.
2. 탐색 창에서 Clusters(클러스터) 링크를 선택합니다.
3. 목록에서 해당 인스턴스를 선택합니다.
4. SSH 처리 설정 그룹을 찾습니다.

   클러스터 퍼블릭 키와 노드 IP 주소를 적어 둡니다. 이후 단계에서 사용하게 됩니다.


SSH Ingestion Settings

Cluster Public Key:

```
ssh-rsa

ExampleKeyDAQABAAABAQCKIVhE2BnJ92xM4ZimOaAeW
ssIDXB3haUmYMpevnnNj/wRRgpcomi7Eo3Fk+Eb7qLk4
qUgQvDMliaxM0Bf2XjRWZBUidQC1DUcuprnRth4XnnIR
lx1pUPq/re/8nQ95pVRS
/sYHWwtOraZlrbECLqhJ40GQLeB5oFJ0ML1MiVfD31xC
jf66kOgI8GAkW0vdgMMPHSr12jjIbyDA+E3+rs1H8g8O
gVhMj7iB4PE+9pnwSi
/aEtwPXzuh6Stbt2t1cuH0ZqZMcyo0tvDLwQit4Qc+06
bBK5CRyu/r6raQbIIS0xddiopvnSSMpihiExample=/
 Amazon-Redshift
```

Node IP Addresses:

Node	Public IP	Private IP
Leader	192.0.2.0	198.51.100.0
Compute-0	203.0.113.0	10.24.34.0
Compute-1	198.51.100.0	192.0.2.0

3단계의 IP 주소를 사용하여 Amazon Redshift로부터의 연결을 수락하도록 호스트를 구성합니다. 연결하는 호스트의 유형이 무엇이고 호스트가 VPC에 있는지 여부에 따라 퍼블릭 IP 주소 또는 프라이빗 IP 주소를 사용합니다.

Amazon Redshift CLI를 사용하여 클러스터 퍼블릭 키와 클러스터의 클러스터 노드 IP 주소를 검색하려면 describe-clusters 명령을 실행합니다.

예:

```
aws redshift describe-clusters --cluster-identifier <cluster-identifier>
```

응답에는 ClusterPublicKey와 프라이빗 및 퍼블릭 IP 주소 목록이 포함되며, 다음과 비슷합니다.

```
{
    "Clusters": [
        {
            "VpcSecurityGroups": [],
            "ClusterStatus": "available",
            "ClusterNodes": [
```

```
                {
                    "PrivateIPAddress": "10.nnn.nnn.nnn",
                    "NodeRole": "LEADER",
                    "PublicIPAddress": "10.nnn.nnn.nnn"
                },
                {
                    "PrivateIPAddress": "10.nnn.nnn.nnn",
                    "NodeRole": "COMPUTE-0",
                    "PublicIPAddress": "10.nnn.nnn.nnn"
                },
                {
                    "PrivateIPAddress": "10.nnn.nnn.nnn",
                    "NodeRole": "COMPUTE-1",
                    "PublicIPAddress": "10.nnn.nnn.nnn"
                }
            ],
            "AutomatedSnapshotRetentionPeriod": 1,
            "PreferredMaintenanceWindow": "wed:05:30-wed:06:00",
            "AvailabilityZone": "us-east-1a",
            "NodeType": "dc2.large",
            "ClusterPublicKey": "ssh-rsa AAAABexamplepublickey...Y3TAl Amazon-
Redshift",
             ...
             ...
}
```

Amazon Redshift API를 사용하여 클러스터 퍼블릭 키와 클러스터의 클러스터 노드 IP 주소를 검색하려면 DescribeClusters 작업을 사용합니다. 자세한 내용은 Amazon Redshift CLI Guide의 describe-clusters 또는 Amazon Redshift API Guide의 DescribeClusters 섹션을 참조하세요.

2단계: 호스트의 권한 부여된 키 파일에 Amazon Redshift 클러스터 퍼블릭 키 추가

호스트가 Amazon Redshift를 인식하고 SSH 연결을 수락하도록 각 호스트의 권한 부여된 키 파일에 클러스터 퍼블릭 키를 추가합니다.

호스트의 권한 부여된 키 파일에 Amazon Redshift 클러스터 퍼블릭 키를 추가하려면

1. SSH 연결을 사용하여 호스트에 액세스합니다.

   SSH를 사용하여 인스턴스에 연결에 대한 자세한 내용은 Amazon EC2 User Guide의 Connect to Your Instance 섹션을 참조하세요.
2. 콘솔 또는 CLI 응답 텍스트에서 Amazon Redshift 퍼블릭 키를 복사합니다.

3. 퍼블릭 키의 내용을 복사하여 원격 호스트의 /home/<ssh_username>/.ssh/authorized_keys 파일에 붙여 넣습니다. <ssh_username>은 매니페스트 파일의 "username" 필드의 값과 일치해야 합니다. 접두사 "ssh-rsa"와 접미사 Amazon-Redshift"를 포함하여 완전한 문자열을 포함시킵니다. 예:

```
ssh-rsa AAAACTP3isxgGzVWoIWpbVvRCOzYdVifMrh… uA70BnMHCaMiRdmvsDOedZDOedZ Amazon-
Redshift
```

3단계: Amazon Redshift 클러스터의 모든 IP 주소를 수락하도록 호스트를 구성

Amazon EC2 인스턴스 또는 Amazon EMR 클러스터 작업 시 각각의 Amazon Redshift 클러스터 노드로부터의 트래픽을 허용하도록 호스트의 보안 그룹에 인바운드 규칙을 추가합니다. 유형에서 포트 22의 TCP 프로토콜이 포함된 SSH를 선택합니다. 소스에, 1단계: 클러스터 퍼블릭 키와 클러스터 노드 IP 주소 검색에서 가져온 Amazon Redshift 클러스터 노드 IP 주소를 입력합니다. Amazon EC2 보안 그룹에 규칙 추가에 대한 자세한 내용은 Amazon EC2 User Guide의 Authorizing Inbound Traffic for Your Instances 섹션을 참조하세요.

프라이빗 IP 주소를 사용하는 경우:

- Virtual Private Cloud(VPC)에 없는 Amazon Redshift 클러스터 및 Amazon EC2 -Classic 인스턴스가 있고, 이 둘 모두 동일한 AWS 리전에 있는 경우.
- VPC에 있는 Amazon Redshift 클러스터 및 Amazon EC2 -VPC 인스턴스가 있고, 이 둘 모두 동일한 AWS 리전 및 동일한 VPC에 있는 경우.

그렇지 않은 경우에는 퍼블릭 IP 주소를 사용합니다.

VPC에서 Amazon Redshift 사용에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 Virtual Private Cloud(VPC)의 클러스터 관리 섹션을 참조하세요.

4단계: 호스트의 퍼블릭 키 가져오기

필요한 경우 Amazon Redshift가 호스트를 식별할 수 있도록 매니페스트 파일에서 호스트의 퍼블릭 키를 제공할 수 있습니다. COPY 명령은 호스트 퍼블릭 키가 필요하지 않지만 보안상 이유로 '중간자' 공격 예방을 위해 퍼블릭 키를 사용하는 것이 좋습니다.

호스트의 퍼블릭 키는 다음 위치에서 찾을 수 있습니다. 여기서 <ssh_host_rsa_key_name>은 호스트 퍼블릭 키의 고유 이름입니다.

```
:  /etc/ssh/<ssh_host_rsa_key_name>.pub
```

> **Note**
>
> Amazon Redshift는 RSA 키만 지원합니다. DSA 키는 지원되지 않습니다.

5단계에서 매니페스트 파일을 만들 때 퍼블릭 키의 텍스트를 매니페스트 파일의 "Public Key" 필드에 붙여 넣습니다.

5단계: 매니페스트 파일 생성

COPY 명령은 SSH를 사용하여 다수의 호스트에 연결할 뿐만 아니라 각 호스트마다 SSH 연결을 생성할 수 있습니다. 이렇게 생성된 각 호스트 연결을 통해 명령을 실행하고, 명령을 통해 출력되는 데이터를 병렬 방식으로 테이블에 로드합니다. 매니페스트 파일은 Amazon Redshift가 호스트에 연결하는 데 사용하는 JSON 형식의 텍스트 파일입니다. 매니페스트 파일은 SSH 호스트 엔드포인트를 비롯해 호스트에서 데이터를 Amazon Redshift에 반환할 때 실행하는 명령을 지정합니다. 필요한 경우, 호스트 퍼블릭 키, 로그인 사용자 이름 및 각 항목의 필수 플래그를 포함시킬 수 있습니다.

로컬 컴퓨터에 매니페스트 파일을 생성합니다. 이후 단계에서 파일을 Amazon S3에 업로드합니다.

매니페스트 파일의 형식은 다음과 같습니다.

```
{
   "entries": [
     {"endpoint":"<ssh_endpoint_or_IP>",
       "command": "<remote_command>",
       "mandatory":true,
       "publickey": "<public_key>",
       "username": "<host_user_name>"},
     {"endpoint":"<ssh_endpoint_or_IP>",
       "command": "<remote_command>",
       "mandatory":true,
       "publickey": "<public_key>",
       "username": "host_user_name"}
    ]
}
```

매니페스트 파일에는 SSH 연결마다 "entries" 구문이 하나씩 포함됩니다. 각 항목은 단일 SSH 연결을 나타냅니다. 단일 호스트에 다중 연결을, 혹은 다수의 호스트에 다중 연결을 생성할 수 있습니다. 예시

와 같이 필드 이름과 값에는 큰따옴표가 필요합니다. 큰따옴표가 필요 없는 유일한 값은 필수 필드의 부울 값 **true** 또는 **false**입니다.

다음은 매니페스트 파일의 필드에 대한 설명입니다.

엔드포인트

호스트의 URL 주소 또는 IP 주소. 예: ec2-111-222-333.compute-1.amazonaws.com 또는 "22.33.44.56"

명령

호스트에 의해 실행되어 텍스트 또는 이진수(gzip, lzop 또는 bzip2) 출력을 생성하는 명령. 명령은 사용자 "host_user_name"이 실행 권한을 갖고 있는 어떤 명령도 될 수 있습니다. 명령은 파일 인쇄처럼 간단할 수도 있고 데이터베이스 쿼리나 스크립트 시작일 수도 있습니다. 출력(텍스트 파일, gzip 이진 파일, lzop 이진 파일 또는 bzip2 이진 파일)은 Amazon Redshift COPY 명령이 수집할 수 있는 형식이어야 합니다. 자세한 내용은 입력 데이터 준비 단원을 참조하십시오.

publickey

(옵션) 호스트의 퍼블릭 키입니다. 퍼블릭 키를 입력하면 Amazon Redshift가 호스트를 식별하는 데 이 키를 사용합니다. 퍼블릭 키를 입력하지 않으면 Amazon Redshift가 호스트를 식별하지 않습니다. 예를 들어 원격 호스트의 퍼블릭 키가 ssh-rsa AbcCbaxxx…xxxDHKJ root@amazon.com인 경우, public key 필드에 다음 텍스트를 입력합니다. AbcCbaxxx…xxxDHKJ

mandatory

(선택적) 연결이 실패하는 경우에 COPY 명령이 실패해야 하는지 여부를 나타냅니다. 기본값은 false입니다. Amazon Redshift가 하나 이상 연결에 성공하지 못하면 COPY 명령이 중단됩니다.

사용자 이름

(옵션) 호스트 시스템에 로그온하고 원격 명령을 실행하기 위해 사용할 사용자 이름입니다. 사용자 로그인 이름은 2단계에서 호스트의 인증 키 파일에 퍼블릭 키를 추가할 때 사용한 로그인 이름과 동일해야 합니다. 기본 사용자 이름은 "redshift"입니다.

다음 예는 동일 호스트에 대한 4개의 연결을 열어 각 연결을 통해 서로 다른 명령을 실행하는 완성된 매니페스트를 보여줍니다.

```
{
  "entries": [
```

```
       {"endpoint":"ec2-184-72-204-112.compute-1.amazonaws.com",
          "command": "cat loaddata1.txt",
          "mandatory":true,
          "publickey": "ec2publickeyportionoftheec2keypair",
          "username": "ec2-user"},
       {"endpoint":"ec2-184-72-204-112.compute-1.amazonaws.com",
          "command": "cat loaddata2.txt",
          "mandatory":true,
          "publickey": "ec2publickeyportionoftheec2keypair",
          "username": "ec2-user"},
       {"endpoint":"ec2-184-72-204-112.compute-1.amazonaws.com",
          "command": "cat loaddata3.txt",
          "mandatory":true,
          "publickey": "ec2publickeyportionoftheec2keypair",
          "username": "ec2-user"},
       {"endpoint":"ec2-184-72-204-112.compute-1.amazonaws.com",
          "command": "cat loaddata4.txt",
          "mandatory":true,
          "publickey": "ec2publickeyportionoftheec2keypair",
          "username": "ec2-user"}
     ]
}
```

6단계: Amazon S3 버킷에 매니페스트 파일 업로드

Amazon S3 버킷에 매니페스트 파일을 업로드합니다. Amazon S3 버킷이 Amazon Redshift 클러스터와 동일한 AWS 리전에 속하지 않는 경우에는 REGION 옵션을 사용하여 매니페스트가 위치한 AWS 리전을 지정해야 합니다. Amazon S3 버킷 생성 및 파일 업로드 방법에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서를 참조하세요.

7단계: COPY 명령을 실행하여 데이터 로드

COPY 명령을 실행하여 호스트에 연결하고 데이터를 Amazon Redshift 테이블에 로드합니다. COPY 명령에서 매니페스트 파일의 명시적 Amazon S3 객체 경로를 지정하고 SSH 옵션을 포함시킵니다. 예를 들면 다음과 같습니다.

```
COPY sales
FROM 's3://amzn-s3-demo-bucket/ssh_manifest'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
DELIMITER '|'
SSH;
```

> **Note**
>
> 자동 압축을 사용하는 경우 COPY 명령은 두 개의 데이터 읽기를 수행합니다. 즉, 원격 명령을 두 번 실행합니다. 첫 번째 읽기는 압축 분석용 샘플을 제공하기 위한 것이며, 두 번째 읽기가 실제로 데이터를 로드합니다. 원격 명령 2회 실행이 잠재적인 부작용으로 인해 문제를 일으킬 수 있는 경우, 자동 압축을 꺼야 합니다. 자동 압축을 끄려면 COMPUPDATE 옵션을 OFF로 설정하여 COPY 명령을 실행합니다. 자세한 내용은 자동 압축을 사용하여 테이블 로드 단원을 참조하십시오.

Amazon DynamoDB 테이블에서 데이터 로드

COPY 명령을 사용하여 단일 Amazon DynamoDB 테이블에서 데이터와 함께 테이블을 로드할 수 있습니다.

> **Important**
>
> 데이터를 제공하는 Amazon DynamoDB 테이블은 REGION 옵션을 사용하여 Amazon DynamoDB 테이블이 위치한 AWS 리전을 지정하지 않는 한 클러스터와 동일한 AWS 리전에서 생성해야 합니다.

COPY 명령은 Amazon Redshift 대량 병렬 처리(MPP) 아키텍처를 사용하여 Amazon DynamoDB 테이블에서 병렬로 데이터를 읽고 로드합니다. Amazon Redshift 테이블에서 배포 스타일을 설정하면 병렬 처리를 최대한 활용할 수 있습니다. 자세한 내용은 쿼리 최적화를 위한 데이터 배포 단원을 참조하십시오.

> **Important**
>
> COPY 명령이 Amazon DynamoDB 테이블에서 데이터를 읽을 때 그 결과로 이루어지는 데이터 전송은 해당 테이블의 프로비저닝 처리량의 일부입니다.

과도한 프로비저닝 읽기 처리량 사용을 피하려면 프로덕션 환경에 있는 Amazon DynamoDB 테이블에서 데이터를 로드하지 않는 것이 좋습니다. 프로덕션 테이블에서 데이터를 로드하는 경우에는 사용되지 않는 프로비저닝 처리량의 평균 비율보다 훨씬 낮게 READRATIO 옵션을 설정하는 것이 좋습니다. 낮은 READRATIO 설정은 조절 문제를 최소화하는 데 도움이 됩니다. Amazon DynamoDB 테이블의 전체 프로비저닝 처리량을 사용하려면 READRATIO를 100으로 설정합니다.

COPY 명령은 다음 규칙을 사용하여 DynamoDB 테이블에서 검색된 항목의 속성 이름을 기존 Amazon Redshift 테이블의 열 이름과 일치시킵니다.

- Amazon Redshift 테이블 열은 대/소문자를 구분하지 않고 Amazon DynamoDB 항목 속성과 일치합니다. DynamoDB 테이블의 항목에 대/소문자만 다른 여러 속성(예: Price와 PRICE)이 포함된 경우, COPY 명령이 실패합니다.
- Amazon DynamoDB 테이블의 속성과 일치하지 않는 Amazon Redshift 테이블 열은 COPY 명령의 EMPTYASNULL 옵션에서 지정한 값에 따라 NULL 또는 비어 있는 상태로 로드됩니다.
- Amazon Redshift 테이블의 열과 일치하지 않는 Amazon DynamoDB 속성은 삭제됩니다. 속성은 일치되기 전에 읽히며, 삭제된 속성도 해당 테이블의 프로비저닝 처리량의 일부를 사용합니다.
- 스칼라 STRING 및 NUMBER 데이터 형식의 Amazon DynamoDB 속성만 지원됩니다. Amazon DynamoDB BINARY 및 SET 데이터 형식은 지원되지 않습니다. COPY 명령이 지원되지 않는 데이터 형식을 가진 속성을 로드하려고 하는 경우, 명령은 실패합니다. 속성이 Amazon Redshift 테이블 열과 일치하지 않는 경우 COPY는 속성을 로드하려고 하지 않으며 오류를 발생시키지 않습니다.

COPY 명령은 다음 구문을 사용하여 Amazon DynamoDB 테이블에서 데이터를 로드합니다.

```
COPY <redshift_tablename> FROM 'dynamodb://<dynamodb_table_name>'
authorization
readratio '<integer>';
```

authorization 값은 Amazon DynamoDB 테이블에 액세스하는 데 필요한 AWS 자격 증명입니다. 이 자격 증명이 사용자와 일치하는 경우 해당 사용자에게 로드되는 Amazon DynamoDB 테이블에 대한 SCAN 및 DESCRIBE 권한이 있어야 합니다.

authorization 값은 클러스터가 Amazon DynamoDB 테이블에 액세스하는 데 필요한 AWS 권한 부여를 제공합니다. 권한에는 로드되는 Amazon DynamoDB 테이블에 대한 SCAN 및 DESCRIBE가 포함되어야 합니다. 필요한 권한에 대한 자세한 내용은 COPY, UNLOAD 및 CREATE LIBRARY 작업을 위한 IAM 권한 섹션을 참조하세요. 선호되는 인증 방법은 IAM_ROLE 파라미터를 지정하고 IAM 역할의 Amazon 리소스 이름(ARN)에 필요한 권한을 제공하는 것입니다. 자세한 내용은 역할 기반 액세스 제어 단원을 참조하십시오.

IAM_ROLE 파라미터를 사용하여 인증하려면 다음 구문을 참조하여 *<aws-account-id>* 및 *<role-name>*을 바꿉니다.

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-name>'
```

다음 예는 IAM 역할을 사용한 인증을 보여 줍니다.

```
COPY favoritemovies
FROM 'dynamodb://ProductCatalog'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

기타 옵션에 대한 자세한 내용은 권한 부여 파라미터 섹션을 참조하세요.

실제로 테이블을 로드하지 않고 데이터를 확인하려면 COPY 명령과 함께 NOLOAD 옵션을 사용하십시오.

다음 예에서는 DynamoDB 테이블 my-favorite-movies-table의 데이터와 함께 FAVORITEMOVIES 테이블을 로드합니다. 읽기 작업은 프로비저닝 처리량의 최대 50%를 사용합니다.

```
COPY favoritemovies FROM 'dynamodb://my-favorite-movies-table'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
READRATIO 50;
```

처리량을 극대화하기 위해 COPY 명령은 클러스터의 컴퓨팅 노드에서 병렬로 Amazon DynamoDB에서 데이터를 로드합니다.

자동 압축을 사용한 프로비저닝 처리량

기본적으로 COPY 명령은 압축 인코딩이 없는 빈 대상 테이블을 지정할 때마다 자동 압축을 적용합니다. 자동 압축 분석은 처음에 Amazon DynamoDB 테이블에서 많은 수의 행을 샘플링합니다. 샘플 크기는 COMPROWS 파라미터의 값에 따라 달라집니다. 기본값은 조각당 100,000행입니다.

샘플링 후 샘플 행은 삭제되며 전체 테이블이 로드됩니다. 그 결과, 많은 행을 두 번 읽게 됩니다. 자동 압축의 작동 원리에 대한 자세한 내용은 자동 압축을 사용하여 테이블 로드 섹션을 참조하세요.

> **Important**
>
> COPY 명령이 샘플링에 사용된 행을 포함하여 Amazon DynamoDB 테이블에서 데이터를 읽을 때 그 결과로 이루어지는 데이터 전송은 해당 테이블의 프로비저닝 처리량의 일부입니다.

Amazon DynamoDB에서 멀티바이트 데이터 로드

데이터에 중국어나 키릴 문자 같이 ASCII가 아닌 멀티바이트 문자가 포함되어 있는 경우에는 VARCHAR 열에 데이터를 로드해야 합니다. VARCHAR 데이터 형식은 4바이트 UTF-8 문자를 지원하지만 CHAR 데이터 형식에서는 1바이트 ASCII 문자만 허용되기 때문입니다. Amazon Redshift 테이블

에 5바이트 이상의 문자는 로드할 수 없습니다. CHAR 및 VARCHAR에 대한 자세한 내용은 데이터 타입 섹션을 참조하세요.

데이터가 올바로 로드되었는지 확인

로드 작업이 완료된 후 STL_LOAD_COMMITS 시스템 테이블을 쿼리하여 예상되는 파일이 로드되었는지 확인합니다. 로드에 문제가 있을 경우 전체 트랜잭션을 롤백할 수 있도록 COPY 명령과 로드 확인을 동일한 트랜잭션 내에서 실행합니다.

다음은 TICKIT 데이터베이스의 테이블을 로드하기 위한 항목을 반환하는 쿼리입니다.

```
SELECT query, trim(filename) AS filename, curtime, status
FROM stl_load_commits
WHERE filename like '%tickit%' order by query;


 query |         filename          |          curtime           | status
-------+---------------------------+----------------------------+--------
 22475 | tickit/allusers_pipe.txt  | 2013-02-08 20:58:23.274186 |      1
 22478 | tickit/venue_pipe.txt     | 2013-02-08 20:58:25.070604 |      1
 22480 | tickit/category_pipe.txt  | 2013-02-08 20:58:27.333472 |      1
 22482 | tickit/date2008_pipe.txt  | 2013-02-08 20:58:28.608305 |      1
 22485 | tickit/allevents_pipe.txt | 2013-02-08 20:58:29.99489  |      1
 22487 | tickit/listings_pipe.txt  | 2013-02-08 20:58:37.632939 |      1
 22489 | tickit/sales_tab.txt      | 2013-02-08 20:58:37.632939 |      1
(6 rows)
```

입력 데이터 확인

실제로 데이터를 로드하기 전에 Amazon S3 입력 파일 또는 Amazon DynamoDB 테이블의 데이터를 확인하려면 COPY 명령과 함께 NOLOAD 옵션을 사용합니다. 데이터를 로드할 때 사용할 동일한 COPY 명령 및 옵션과 함께 NOLOAD를 사용합니다. NOLOAD는 데이터를 데이터베이스에 로드하지 않고 모든 데이터의 무결성을 검사합니다. NOLOAD 옵션은 데이터 로드를 시도할 경우 발생하는 모든 오류를 표시합니다.

예를 들어 입력 파일에 잘못된 Amazon S3 경로를 지정하면 Amazon Redshift에서 다음 오류를 표시합니다.

```
ERROR:  No such file or directory
DETAIL:
```

```
-----------------------------------------------
Amazon Redshift error:  The specified key does not exist
code:      2
context:   S3 key being read :
location:  step_scan.cpp:1883
process:   xenmaster [pid=22199]
-----------------------------------------------
```

오류 메시지 문제를 해결하려면 로드 오류 참조 섹션을 참조하세요.

NOLOAD 옵션을 사용하는 예제는 NOLOAD 옵션을 사용한 CPPY 명령 섹션을 참조하세요.

자동 압축을 사용하여 테이블 로드

데이터 평가에 따라 수동으로 테이블의 열에 압축 인코딩을 적용할 수 있습니다. 또는 COMPUPDATE가 [켜짐(ON)]으로 설정된 COPY 명령을 사용하여 샘플 데이터를 기반으로 압축을 자동으로 분석하고 적용할 수 있습니다.

새 테이블을 만들고 로드할 때 자동 압축을 사용할 수 있습니다. COPY 명령은 압축 분석을 수행합니다. 이미 채워진 테이블에 대해 ANALYZE COMPRESSION 명령을 실행하여 테이블에서 데이터를 로드하거나 압축을 변경하지 않고 압축 분석을 수행할 수도 있습니다. 예를 들어 기존 데이터 정의 언어(DDL) 문을 유지하면서 나중에 사용하기 위해 테이블에서 압축을 분석하려는 경우 ANALYZE COMPRESSION을 실행할 수 있습니다.

자동 압축은 압축 인코딩을 선택할 때 전반적인 성능의 밸런스를 유지합니다. 범위가 제한된 스캔은 정렬 키 열이 동일한 쿼리의 다른 열보다 훨씬 높게 압축된 경우 성능이 떨어질 수 있습니다. 따라서 자동 압축은 정렬 키 열에서 데이터 분석 단계를 건너뛰고 사용자 정의 인코딩 유형을 유지합니다.

인코딩 유형을 명시적으로 정의하지 않은 경우 자동 압축은 RAW 인코딩을 선택합니다. ANALYZE COMPRESSION은 동일하게 동작합니다. 최적의 쿼리 성능을 얻으려면 정렬 키에 RAW를 사용해 보십시오.

자동 압축 작동 원리

COMPUPDATE 파라미터가 ON으로 설정된 경우 COPY 명령은 빈 대상 테이블에 COPY 명령을 실행할 때마다 항상 자동 압축을 적용하고 모든 테이블 열에는 RAW 인코딩이 있거나 인코딩이 없습니다.

현재의 압축 인코딩에 상관없이 빈 테이블에 자동 압축을 적용하려면 COMPUPDATE 옵션을 ON으로 설정하여 COPY 명령을 실행합니다. 자동 압축을 끄려면 COMPUPDATE 옵션을 OFF로 설정하여 COPY 명령을 실행합니다.

이미 데이터가 포함되어 있는 테이블에는 자동 압축을 적용할 수 없습니다.

> **Note**
>
> 자동 압축 분석이 유의미한 샘플을 생성하려면 로드 데이터에 충분한 행이 필요합니다(조각당 100,000행 이상).

자동 압축은 로드 트랜잭션의 일환으로 백그라운드에서 이러한 작업을 수행합니다.

1. 초기 행 샘플은 입력 파일에서 로드됩니다. 샘플 크기는 COMPROWS 파라미터의 값에 기반합니다. 기본값은 100,000입니다.
2. 각 열마다 압축 옵션이 선택됩니다.
3. 테이블에서 샘플 행이 제거됩니다.
4. 선택한 압축 인코딩을 사용하여 테이블이 다시 생성됩니다.
5. 전체 입력 파일이 로드되고 새 인코딩을 사용하여 압축됩니다.

COPY 명령을 실행한 후에 테이블이 완전히 로드되고 압축되며, 사용할 수 있게 됩니다. 나중에 더 많은 데이터를 로드하는 경우, 추가되는 행은 기존 인코딩에 따라 압축됩니다.

압축 분석만을 수행하려면 ANALYZE COMPRESSION을 실행하는 것이 전체 COPY를 실행하는 것보다 효율적입니다. 그런 다음 결과를 평가하여 자동 압축을 사용할지 수동으로 테이블을 다시 생성할지 결정합니다.

자동 압축은 COPY 명령에만 지원됩니다. 아니면 테이블을 만들 때 수동으로 압축 인코딩을 적용할 수 있습니다. 수동 압축 인코딩에 대한 자세한 내용은 열 압축으로 저장된 데이터 크기 축소 섹션을 참조하세요.

자동 압축 예

이 예에서는 TICKIT 데이터베이스에 BIGLIST라고 하는 LISTING 테이블의 사본이 포함되어 있고 이 테이블이 약 300만 개의 행과 함께 로드될 때 자동 압축을 적용하려 한다고 가정합니다.

테이블을 로드하고 자동으로 압축하려면

1. 테이블이 비어 있는지 확인합니다. 자동 압축은 비어 있는 테이블에만 적용할 수 있습니다.

```
TRUNCATE biglist;
```

2. 단일 COPY 명령을 사용하여 테이블을 로드합니다. 테이블이 비어 있어도 이전의 일부 인코딩이 지정되어 있을 수 있습니다. Amazon Redshift가 압축 분석을 수행하도록 촉진하려면 COMPUPDATE 파라미터를 ON으로 설정합니다.

```
COPY biglist FROM 's3://amzn-s3-demo-bucket/biglist.txt'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
DELIMITER '|' COMPUPDATE ON;
```

COMPROWS 옵션이 지정되지 않았기 때문에 기본적인 권장 샘플 크기인 조각당 100,000행이 사용됩니다.

3. 자동으로 선택된 인코딩 체계를 검토하려면 BIGLIST의 새 스키마를 살펴보십시오.

```
SELECT "column", type, encoding
from pg_table_def where tablename = 'biglist';


     Column     |            Type             | Encoding
----------------+-----------------------------+----------
 listid         | integer                     | az64
 sellerid       | integer                     | az64
 eventid        | integer                     | az64
 dateid         | smallint                    | none
 numtickets     | smallint                    | az64
 priceperticket | numeric(8,2)                | az64
 totalprice     | numeric(8,2)                | az64
 listtime       | timestamp without time zone | az64
```

4. 예상되는 수의 행이 로드되었는지 확인합니다.

```
select count(*) from biglist;

count
---------
3079952
(1 row)
```

나중에 COPY 또는 INSERT 문을 사용하여 이 테이블에 행을 추가하는 경우 동일한 압축 인코딩이 적용됩니다.

좁은 테이블의 스토리지 최적화

열은 극히 적지만 행의 수는 아주 많은 테이블의 경우, 3개의 숨겨진 메타데이터 ID 열(INSERT_XID, DELETE_XID, ROW_ID)이 테이블의 디스크 공간 중 불균형하게 많은 양을 사용합니다.

숨겨진 열의 압축을 최적화하기 위해서는 가능하다면 단일 COPY 트랜잭션에서 테이블을 로드하십시오. 여러 개의 별도의 COPY 명령을 사용하여 테이블을 로드하는 경우, INSERT_XID 열이 잘 압축되지 않습니다. 여러 COPY 명령을 사용하는 경우에는 vacuum 작업을 수행해야 하지만 INSERT_XID의 압축은 개선되지 않습니다.

열 기본값 로드

필요한 경우, COPY 명령에서 열 목록을 정의할 수 있습니다. 테이블의 열이 열 목록에서 누락된 경우, COPY는 CREATE TABLE 명령에서 지정된 DEFAULT 옵션이 제공하는 값 또는 DEFAULT 옵션이 지정되지 않은 경우에는 NULL을 사용하여 열을 로드합니다.

NOT NULL로 정의된 열에 COPY가 NULL을 할당하려고 하면 COPY 명령이 실패합니다. DEFAULT 옵션 할당에 대한 자세한 내용은 CREATE TABLE 섹션을 참조하세요.

Amazon S3의 데이터 파일에서 로드하는 경우 열 목록의 열은 데이터 파일의 필드와 동일한 순서여야 합니다. 데이터 파일의 필드에 열 목록의 해당 열이 없는 경우, COPY 명령이 실패합니다.

Amazon DynamoDB 테이블에서 로드할 때는 순서가 중요하지 않습니다. Amazon Redshift 테이블의 열과 일치하지 않는 Amazon DynamoDB 속성의 필드는 삭제됩니다.

COPY 명령을 사용하여 DEFAULT 값을 테이블로 로드할 때는 다음의 제한이 적용됩니다.

- 열 목록에 IDENTITY 열이 포함된 경우, COPY 명령에서 EXPLICIT_IDS 옵션도 지정해야 하며, 그렇지 않을 경우, COPY 명령이 실패합니다. 마찬가지로 열 목록에서 IDENTITY 열이 누락되어 있고 EXPLICIT_IDS 옵션이 지정된 경우, COPY 명령이 실패합니다.
- 일정한 열에 대해 평가된 DEFAULT 표현식은 로드된 모든 행에서 동일하므로 RANDOM() 함수를 사용하는 DEFAULT 표현식은 모든 행에 동일한 값을 할당합니다.
- CURRENT_DATE 또는 SYSDATE가 포함된 DEFAULT 표현식은 현재 트랜잭션의 타임스탬프로 설정됩니다.

예를 들어 COPY 예의 "기본값을 사용하여 파일에서 데이터 로드"를 참조하십시오.

데이터 로드 문제 해결

Amazon Redshift 테이블로 데이터를 로드할 때 Amazon S3의 오류, 잘못된 입력 데이터 및 COPY 명령 오류가 발생할 수 있습니다. 다음 섹션에서는 데이터 로드 오류 식별 및 해결에 대한 정보를 제공합니다.

주제

S3 이벤트 통합 및 COPY JOB 오류 문제 해결

다음 정보를 사용하여 일반적인 Amazon S3 이벤트 통합 문제 및 Amazon Redshift를 사용한 COPY JOB 문제를 해결하세요.

S3 이벤트 통합 만들기 실패

S3 이벤트 통합을 만들지 못한 경우 통합의 상태는 `Inactive`입니다. Amazon Redshift 데이터 웨어하우스에서 다음 사항이 올바른지 확인하세요.

- Amazon Redshift의 대상 네임스페이스에 대해 올바른 권한 있는 위탁자 및 통합 소스를 추가했습니다. S3 이벤트 통합을 만들기 위한 사전 조건 섹션을 참조하세요.
- 소스 Amazon S3 버킷에 올바른 리소스 기반 정책을 추가했습니다. S3 이벤트 통합을 만들기 위한 사전 조건 섹션을 참조하세요.

Amazon S3 데이터가 대상 데이터베이스에 표시되지 않음

COPY JOB의 데이터가 표시되지 않으면 다음을 확인합니다.

- SYS_COPY_JOB_DETAIL을 쿼리하여 Amazon S3 파일이 로드되었는지, 수집이 보류 중인지 또는 오류가 있는지 확인합니다. 자세한 내용은 SYS_COPY_JOB_DETAIL 단원을 참조하십시오.
- Amazon S3 파일이 없거나 예기치 않은 대기 시간이 있는 경우 STL_ERROR 또는 SYS_COPY_JOB_INFO를 참조합니다. 자격 증명 오류 또는 통합이 비활성 상태임을 암시하는 것이 있는지 찾습니다. 자세한 내용은 STL_ERROR 및 SYS_COPY_JOB_INFO 단원을 참조하세요.

S3ServiceException 오류

가장 일반적인 s3ServiceException 오류의 원인은 형식이 잘못 지정되었거나, 잘못된 자격 증명 문자열이 있거나, 서로 다른 AWS 리전에 있는 클러스터와 버킷이 있거나, Amazon S3 권한이 불충분한 경우입니다.

이 단원에서는 각각의 오류 유형에 대한 문제 해결 정보를 제공합니다.

잘못된 자격 증명 문자열

자격 증명 문자열의 형식이 잘못 지정된 경우, 다음과 같은 오류 메시지가 표시됩니다.

```
ERROR: Invalid credentials. Must be of the format: credentials
'aws_access_key_id=<access-key-id>;aws_secret_access_key=<secret-access-key>
[;token=<temporary-session-token>]'
```

자격 증명 문자열에 공백이나 줄 바꿈이 없고 작은따옴표로 묶여 있는지 확인합니다.

잘못된 액세스 키 ID

액세스 키 ID가 존재하지 않는 경우, 다음과 같은 오류 메시지가 표시됩니다.

```
[Amazon](500310) Invalid operation: S3ServiceException:The AWS Access Key Id you
 provided does not exist in our records.
```

이것은 복사 및 붙여넣기 오류인 경우가 많습니다. 액세스 키 ID를 올바로 입력했는지 확인합니다. 또한 임시 세션 키를 사용 중인 경우 token에 대한 값이 설정되어 있는지 확인합니다.

잘못된 보안 액세스 키

보안 액세스 키가 잘못된 경우, 다음과 같은 오류 메시지가 표시됩니다.

```
[Amazon](500310) Invalid operation: S3ServiceException:The request signature we
 calculated does not match the signature you provided.
Check your key and signing method.,Status 403,Error SignatureDoesNotMatch
```

이것은 복사 및 붙여넣기 오류인 경우가 많습니다. 보안 액세스 키를 올바로 입력했고 액세스 키 ID에 맞는 키인지 확인합니다.

버킷이 다른 리전에 있음

COPY 명령에서 지정된 Amazon S3 버킷은 클러스터와 같은 AWS 리전에 있어야 합니다. Amazon S3 버킷과 클러스터가 서로 다른 리전에 있는 경우 다음과 비슷한 오류 메시지가 표시됩니다.

```
ERROR: S3ServiceException:The bucket you are attempting to access must be addressed
 using the specified endpoint.
```

Amazon S3 관리 콘솔을 사용하여 버킷을 생성할 때 리전을 선택하거나 Amazon S3 API 또는 CLI를 사용하여 버킷을 생성할 때 엔드포인트를 지정하여 특정 리전에서 Amazon S3 버킷을 생성할 수 있습니다. 자세한 내용은 COPY와 함께 사용할 파일을 Amazon S3에 업로드 단원을 참조하십시오.

Amazon S3 리전에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 버킷 액세스를 참조하세요.

또는 COPY 명령과 함께 REGION 옵션을 사용하여 리전을 지정할 수 있습니다.

액세스 거부됨

사용자에게 충분한 권한이 없는 경우, 다음과 같은 오류 메시지가 표시됩니다.

```
ERROR: S3ServiceException:Access Denied,Status 403,Error AccessDenied
```

가능한 원인 중 하나는 보안 인증 정보에 의해 식별되는 사용자가 Amazon S3 버킷에 대한 LIST 및 GET 액세스 권한이 없을 수 있다는 것입니다. 기타 원인을 보려면 Amazon Simple Storage Service 사용 설명서의 Amazon S3의 액세스 거부(403 Forbidden) 오류 해결을 참조하세요.

버킷에 대한 사용자 액세스를 관리하는 방법에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 Amazon S3의 ID 및 액세스 관리를 참조하세요.

데이터 로드 문제 해결을 위한 시스템 테이블

다음 Amazon Redshift 시스템 테이블은 데이터 로드 문제 해결에 도움이 될 수 있습니다.

- 특정 로드 도중 발생한 오류를 발견하려면 STL_LOAD_ERRORS를 쿼리합니다.
- 특정 파일의 로드 시간을 확인하거나 특정 파일이 읽혔는지 보려면 STL_FILE_SCAN을 쿼리합니다.
- Amazon S3에서 데이터를 전송하는 중에 발생한 오류의 세부 정보를 찾으려면 STL_S3CLIENT_ERROR를 쿼리합니다.

로드 오류를 찾고 진단하려면

1. 뷰를 생성하거나 로드 오류에 대한 세부 정보를 반환하는 쿼리를 정의합니다. 다음 예는 STL_LOAD_ERRORS 테이블을 STV_TBL_PERM 테이블에 조인하여 테이블 ID를 실제 테이블 이름과 일치시킵니다.

```
create view loadview as
(select distinct tbl, trim(name) as table_name, query, starttime,
trim(filename) as input, line_number, colname, err_code,
trim(err_reason) as reason
from stl_load_errors sl, stv_tbl_perm sp
where sl.tbl = sp.id);
```

2. COPY가 데이터에 관한 유용한 정보를 반환할 수 있도록 COPY 명령에서 MAXERRORS 옵션을 충분히 큰 값으로 설정합니다. COPY에서 오류가 발생하는 경우, STL_LOAD_ERRORS 테이블에서 세부 정보를 참조하라고 오류 메시지에 표시됩니다.
3. 오류 세부 정보를 보려면 LOADVIEW 뷰를 쿼리합니다. 예:

```
select * from loadview where table_name='venue';
```

```
  tbl   | table_name | query |         starttime
--------+------------+-------+----------------------------
 100551 | venue      | 20974 | 2013-01-29 19:05:58.365391

|     input      | line_number | colname | err_code |       reason
+----------------+-------------+---------+----------+--------------------
| venue_pipe.txt |           1 |       0 |     1214 | Delimiter not found
```

4. 뷰가 반환하는 정보를 바탕으로 입력 파일이나 로드 스크립트의 문제를 수정합니다. 주의할 몇 가지 일반적인 오류는 다음과 같습니다.
   - 테이블의 데이터 형식과 입력 데이터 필드의 값이 불일치.
   - 테이블의 열 수와 입력 데이터의 필드 수가 불일치.
   - 따옴표의 짝이 맞지 않습니다. Amazon Redshift는 작은따옴표와 큰따옴표를 모두 지원합니다. 그러나 이들 따옴표는 적절히 짝이 맞아야 합니다.
   - 입력 파일의 잘못된 형식의 날짜/시간 데이터.
   - 입력 파일에서 범위를 벗어난 값(숫자 열의 경우).
   - 열의 고유 값 수가 압축 인코딩의 제한을 초과.

멀티바이트 문자 로드 오류

CHAR 데이터 형식인 열은 1바이트 UTF-8 문자 127바이트까지 또는 ASCII 문자 세트이기도 한 7F 16진수만을 허용합니다. VARCHAR 열은 최대 4바이트까지 멀티바이트 UTF-8 문자를 허용합니다. 자세한 내용은 문자 형식 단원을 참조하십시오.

로드 데이터의 줄에 열 데이터 형식에 맞지 않는 문자가 포함되어 있는 경우, COPY는 오류를 반환하고 STL_LOAD_ERRORS 시스템 로그 테이블에 오류 번호 1220을 사용하여 행을 기록합니다. ERR_REASON 필드에는 잘못된 문자의 바이트 시퀀스가 16진수로 포함되어 있습니다.

로드 데이터의 유효하지 않은 문자를 수정하는 대신 로드 프로세스 도중 유효하지 않은 문자를 대체할 수 있습니다. 유효하지 않은 UTF-8 문자를 대체하려면 COPY 명령에서 ACCEPTINVCHARS 옵션을 지정합니다. ACCEPTINVCHARS 옵션이 설정되면 지정한 문자가 코드 포인트를 대체합니다. ACCEPTINVCHARS 옵션이 설정되지 않은 경우 Amazon Redshift는 문자를 유효한 UTF-8로 수락합니다. 자세한 내용은 ACCEPTINVCHARS 단원을 참조하십시오.

다음 코드 포인트 목록은 유효한 UTF-8이며 ACCEPTINVCHARS 옵션이 설정되지 않은 경우 COPY 작업은 오류를 반환하지 않습니다. 그러나 이러한 코드 포인트는 유효하지 않은 문자입니다. ACCEPTINVCHARS 옵션을 사용하여 코드 포인트를 지정한 문자로 바꿀 수 있습니다. 이러한 코드 포인트에는 `0xFDD0~0xFDEF`의 값 범위와 `FFFE` 또는 `FFFF`로 끝나는 `0x10FFFF`까지의 값이 포함됩니다.

- `0xFFFE`, `0x1FFFE`, `0x2FFFE`, ..., `0xFFFFE`, `0x10FFFE`
- `0xFFFF`, `0x1FFFF`, `0x2FFFF`, ..., `0xFFFFF`, `0x10FFFF`

다음 예는 COPY가 UTF-8 문자 `e0 a1 c7a4`를 CHAR 열로 로드하려 할 때 발생하는 오류 이유를 보여줍니다.

```
Multibyte character not supported for CHAR
(Hint: Try using  VARCHAR). Invalid char: e0 a1 c7a4
```

오류가 VARCHAR 데이터 형식과 관련된 경우, 유효하지 않은 UTF-8 16진수 시퀀스뿐 아니라 오류 코드도 오류 이유에 포함됩니다. 다음 예는 COPY가 UTF-8 a4를 VARCHAR 필드로 로드하려 할 때 발생하는 오류 이유를 보여줍니다.

```
String contains invalid or unsupported UTF-8 codepoints.
Bad UTF-8 hex sequence: a4 (error 3)
```

다음 표에는 VARCHAR 로드 오류의 설명과 해결 방법 제안이 나와 있습니다. 이런 오류 중 하나가 발생하면 문자를 유효한 UTF-8 코드 시퀀스로 대체하거나 문자를 삭제하십시오.

오류 코드	설명
1	UTF-8 바이트 시퀀스가 VARCHAR가 지원하는 최대 4바이트를 초과합니다.
2	UTF-8 바이트 시퀀스가 불완전합니다. COPY가 문자열 끝 앞에서 멀티바이트 문자의 예상되는 연속 바이트 수를 찾지 못했습니다.
3	UTF-8 1바이트 문자가 범위를 초과합니다. 시작 바이트는 254, 255이거나 128과 191 사이의 문자여서는 안 됩니다(128과 191도 포함).
4	바이트 시퀀스의 후행 바이트 값이 범위를 초과합니다. 연속 바이트는 128에서 191 사이여야 합니다(128과 191도 포함).
5	UTF-8 문자가 서로게이트로 예약되어 있습니다. 서로게이트 코드 포인트(U+D800 ~ U+DFFF)가 유효하지 않습니다.
8	바이트 시퀀스가 최대 UTF-8 코드 포인트를 초과합니다.
9	UTF-8 바이트 시퀀스에 일치하는 코드 포인트가 없습니다.

로드 오류 참조

파일에서 데이터를 로드하는 중에 오류가 발생하면 STL_LOAD_ERRORS 테이블을 쿼리하여 오류를 파악하고 가능한 설명을 확인합니다. 다음 표에는 데이터 로드 중에 발생할 수 있는 모든 오류 코드가 나와 있습니다.

로드 오류 코드

오류 코드	설명
1200	알 수 없는 구문 분석 오류. 고객 지원 센터에 문의.
1201	입력 파일에서 필드 구분 기호를 찾을 수 없습니다.
1202	입력 데이터의 열이 DDL에서 정의된 것보다 많습니다.

오류 코드	설명
1203	입력 데이터의 열이 DDL에서 정의된 것보다 적습니다.
1204	입력 데이터가 데이터 형식에 허용되는 범위를 초과했습니다.
1205	날짜 형식이 유효하지 않습니다. 유효한 형식은 DATEFORMAT 및 TIMEFORMAT 문자열 섹션을 참조하세요.
1206	타임스탬프 형식이 유효하지 않습니다. 유효한 형식은 DATEFORMAT 및 TIMEFORMAT 문자열 섹션을 참조하세요.
1207	데이터에 0-9의 예상 범위를 벗어난 값이 포함되었습니다.
1208	FLOAT 데이터 형식 오류.
1209	DECIMAL 데이터 형식 오류.
1210	BOOLEAN 데이터 형식 오류.
1211	입력 줄에 포함된 데이터가 없습니다.
1212	로드 파일을 찾을 수 없습니다.
1213	NOT NULL로 지정된 필드에 포함된 데이터가 없습니다.
1214	구분 기호를 찾을 수 없음.
1215	CHAR 필드 오류.
1216	입력 줄이 유효하지 않습니다.
1217	ID 열 값이 유효하지 않습니다.
1218	NULL AS '\0'을 사용할 때 null 종결자(NUL 또는 UTF-8 0000)가 포함된 필드에 2바이트 이상이 포함되었습니다.
1219	UTF-8 16진수에 잘못된 숫자가 포함되어 있습니다.
1220	문자열에 잘못되거나 지원되지 않는 UTF-8 코드 포인트가 포함되어 있습니다.
1221	파일의 인코딩이 COPY 명령에서 지정된 것과 동일하지 않습니다.

오류 코드	설명
1222	정수 값 오버플로 오류입니다.
1223	데이터 유형이 유효하지 않습니다.
1224	입력 데이터가 잘못된 JSON 형식 또는 SUPER 데이터 유형입니다.
8001	MANIFEST 파라미터를 사용한 COPY에는 Amazon S3 객체의 전체 경로가 필요합니다.
9005	잘못된 종료 키가 지정되었습니다.

Amazon S3 버킷에서 자동으로 파일을 복사하기 위해 S3 이벤트 통합 만들기

> **Note**
>
> 자동 복사용 미리 보기 릴리스가 종료되었습니다. 따라서 미리 보기 클러스터는 미리 보기 기간 종료 30일 후에 자동으로 제거됩니다. 자동 복사를 계속 사용하려면 다른 Amazon Redshift 클러스터에서 기존 자동 복사 작업을 다시 만드는 것이 좋습니다. 미리 보기 클러스터를 최신 Amazon Redshift 버전으로 업그레이드하는 것은 지원되지 않습니다.

자동 복사 작업을 사용하여 Amazon S3에 저장된 파일에서 Amazon Redshift 테이블로 데이터를 로드할 수 있습니다. Amazon Redshift는 COPY 명령에 지정된 경로에 새 Amazon S3 파일이 추가되는 시기를 감지합니다. 그러면 외부 데이터 수집 파이프라인을 만들지 않고도 COPY 명령이 자동으로 실행됩니다. Amazon Redshift는 로드된 파일을 추적합니다. Amazon Redshift는 COPY 명령당 함께 배치되는 파일 수를 결정합니다. 시스템 뷰에서 결과 COPY 명령을 볼 수 있습니다.

자동 COPY JOB을 만드는 첫 번째 단계는 S3 이벤트 통합을 만드는 것입니다. Amazon S3 소스 버킷에 새 파일이 나타나면 Amazon Redshift는 COPY 명령을 사용하여 데이터베이스로의 파일 로드를 관리합니다.

S3 이벤트 통합을 만들기 위한 사전 조건

s3 이벤트 통합을 설정하려면 다음 사전 조건이 충족되었는지 확인합니다.

- Amazon S3 버킷에 여러 Amazon S3 권한을 허용하는 버킷 정책이 있어야 합니다. 예를 들어 다음 예시 정책은 *us-east-1*에서 호스팅되는 리소스 버킷 amzn-s3-demo-bucket에 대한 권한을 허용합니다. Amazon S3 버킷과 통합이 모두 동일한 AWS 리전에 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Auto-Copy-Policy-01",
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
                },
            "Action": [
                "s3:GetBucketNotification",
                "s3:PutBucketNotification",
                "s3:GetBucketLocation"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
            "Condition": {
                "StringLike": {
                    "aws:SourceArn": "arn:aws:redshift:us-
east-1:123456789012:integration:*",
                    "aws:SourceAccount": "123456789012"
                }
            }
        }
    ]
}
```

- 대상 Amazon Redshift 프로비저닝된 클러스터 또는 Redshift Serverless 네임스페이스에 버킷에 대한 권한이 있어야 합니다. 클러스터 또는 서버리스 네임스페이스와 연결된 IAM 역할에 적절한 권한을 허용하는 IAM 정책이 있는지 확인합니다. 이 정책은 버킷 리소스에 대한 s3:GetObject(예: *amzn-s3-demo-bucket*) 버킷 리소스에 대한 s3:ListBucket 및 해당 콘텐츠(예: *amzn-s3-demo-bucket*/*)를 모두 허용해야 합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AutoCopyReadId",
            "Effect": "Allow",
```

```
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}
```

역할에 대한 신뢰 관계가 있는 IAM 역할에 정책을 추가하는 방법은 다음과 같습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "redshift.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

대상 데이터 웨어하우스가 프로비저닝된 클러스터인 경우 Amazon Redshift 콘솔에서 클러스터 세부 정보의 클러스터 권한 탭을 사용하여 IAM 역할을 프로비저닝된 클러스터에 연결할 수 있습니다. 프로비저닝된 클러스터에 역할을 연결하는 방법에 대한 자세한 내용은 Amazon Redshift 관리 안내서의 IAM 역할을 클러스터와 연결을 참조하세요.

대상 데이터 웨어하우스가 Redshift Serverless인 경우 Redshift Serverless 콘솔에서 네임스페이스 세부 정보의 보안 및 암호화 탭을 사용하여 IAM 역할을 서버리스 네임스페이스에 연결할 수 있습니다. 역할을 서버리스 네임스페이스에 연결하는 방법에 대한 자세한 내용은 Amazon Redshift 관리 안내서의 Amazon Redshift Serverless에 권한 부여를 참조하세요.

- Amazon Redshift 데이터 웨어하우스에는 Amazon S3 버킷을 허용하는 리소스 정책도 있어야 합니다. Amazon Redshift 콘솔을 사용하는 경우 s3 이벤트 통합을 만들 때 Amazon Redshift는이 정책을 Amazon Redshift 데이터 웨어하우스에 추가할 수 있도록 수정 옵션을 제공합니다. 직접 리소스 정책

을 업데이트하려면 put-resource-policy AWS CLI 명령을 사용할 수 있습니다. 예를 들어 Amazon S3 버킷과의 S3 이벤트 통합을 위해 Amazon Redshift 프로비저닝된 클러스터에 리소스 정책을 연결하려면 다음과 유사한 AWS CLI 명령을 실행합니다. 다음 예시에서는 사용자 계정 *123456789012*에 대해 *us-east-1* AWS 리전에 프로비저닝된 클러스터 네임스페이스에 대한 정책을 보여줍니다. 버킷의 이름은 *amzn-s3-demo-bucket*입니다.

```
aws redshift put-resource-policy \
--policy file://rs-rp.json \
--resource-arn "arn:aws:redshift: us-east-1:123456789012:namespace/cc4ffe56-
ad2c-4fd1-a5a2-f29124a56433"
```

여기서 rs-rp.json에 다음 사항이 포함됩니다.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Principal": {
    "Service": "redshift.amazonaws.com"
   },
   "Action": "redshift:AuthorizeInboundIntegration",
   "Resource": "arn:aws:redshift:us-east-1:123456789012:namespace/cc4ffe56-ad2c-4fd1-
a5a2-f29124a56433",
   "Condition": {
    "StringEquals": {
     "aws:SourceArn": "arn:aws:s3:::amzn-s3-demo-bucket"
    }
   }
  },
  {
   "Effect": "Allow",
   "Principal": {
    "AWS": "arn:aws:iam::123456789012:role/myRedshiftRole"
   },
   "Action": "redshift:CreateInboundIntegration",
   "Resource": "arn:aws:redshift:us-east-1:123456789012:namespace/cc4ffe56-ad2c-4fd1-
a5a2-f29124a56433",
   "Condition": {
    "StringEquals": {
     "aws:SourceArn": "arn:aws:s3:::amzn-s3-demo-bucket"
    }
```

```
   }
  }
 ]
}
```

Amazon S3 버킷과 S3 이벤트 통합을 위해 Redshift Serverless 네임스페이스에 리소스 정책을 연결하려면 다음과 유사한 AWS CLI 명령을 실행합니다. 다음 예시에서는 사용자 계정 *123456789012*의 *us-east-1*에 있는 서버리스 네임스페이스에 AWS 리전 대한 정책을 보여줍니다. 버킷의 이름은 *amzn-s3-demo-bucket*입니다.

```
aws redshift put-resource-policy \
--policy file://rs-rp.json \
--resource-arn "arn:aws:redshift-serverless:us-east-1:123456789012:namespace/
namespace-1"
```

여기서 `rs-rp.json`에 다음 사항이 포함됩니다.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Principal": {
    "Service": "redshift.amazonaws.com"
   },
   "Action": "redshift:AuthorizeInboundIntegration",
   "Resource": "arn:aws:redshift-serverless:us-east-1:123456789012:namespace/
namespace-1",
   "Condition": {
    "StringEquals": {
     "aws:SourceArn": "arn:aws:s3:::amzn-s3-demo-bucket"
    }
   }
  },
  {
   "Effect": "Allow",
   "Principal": {
    "AWS": "arn:aws:iam::123456789012:user/myUser"
   },
   "Action": "redshift:CreateInboundIntegration",
   "Resource": "arn:aws:redshift-serverless:us-east-1:123456789012:namespace/
namespace-1",
```

```
    "Condition": {
     "StringEquals": {
      "aws:SourceArn": "arn:aws:s3:::amzn-s3-demo-bucket"
     }
    }
   }
  ]
}
```

S3 이벤트 통합 만들기

복사 작업을 설정하려면 먼저 S3 이벤트 통합을 정의합니다.

Amazon Redshift console

Amazon Redshift 콘솔에서 Amazon S3 이벤트 통합을 만드는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 S3 이벤트 통합을 선택합니다.
3. Amazon S3 이벤트 통합 만들기를 선택하여 마법사를 열어 자동 복사에 사용할 S3 이벤트 통합을 만듭니다. 소스 Amazon S3 버킷과 대상 Amazon Redshift 데이터 웨어하우스는 동일한 AWS 리전에 있어야 합니다. 통합을 만드는 단계를 진행할 때 다음 정보를 지정합니다.

   - 통합 이름 - 현재 AWS 리전에서 AWS 계정이 소유한 모든 통합의 고유 식별자입니다.
   - 설명 - 나중에 참조할 수 있도록 Amazon S3 이벤트 통합을 설명하는 텍스트입니다.
   - 소스 S3 버킷 - 현재 AWS 계정 및 AWS 리전에 있는 Amazon S3 버킷이며 Amazon Redshift로 데이터를 수집하는 소스입니다.
   - Amazon Redshift 데이터 웨어하우스 - 통합으로부터 데이터를 수신하는 대상 Amazon Redshift 프로비저닝된 클러스터 또는 Redshift Serverless 작업 그룹입니다.

     대상 Amazon Redshift가 동일한 계정에 있는 경우 대상을 선택할 수 있습니다. 대상이 다른 계정에 있는 경우 Amazon Redshift 데이터 웨어하우스 ARN을 지정합니다. 대상에는 승인된 위탁자 및 통합 소스가 포함된 리소스 정책이 있어야 합니다. 대상에 올바른 리소스 정책이 없고 대상이 동일한 계정에 있는 경우 수정 옵션을 선택하여 통합 생성 프로세스 중에 리소스 정책을 자동으로 적용할 수 있습니다. 대상이 다른 AWS 계정에 있는 경우 Amazon Redshift 웨어하우스에서 리소스 정책을 수동으로 적용해야 합니다.
4. 최대 50개의 태그 키와 선택적 값 입력 - 통합에 대한 추가 메타데이터를 제공합니다.

5. S3 이벤트 통합 만들기를 선택할 수 있는 검토 페이지가 표시됩니다.

AWS CLI

AWS CLI를 사용하여 Amazon S3 이벤트 통합을 만들려면 다음 옵션과 함께 `create-integration` 명령을 사용합니다.

- `integration-name` - 통합 이름을 지정합니다.
- `source-arn` - Amazon S3 소스 버킷의 ARN을 지정합니다.
- `target-arn` - Amazon Redshift 프로비저닝 클러스터 또는 Redshift Serverless 작업 그룹 대상의 네임스페이스 ARN을 지정합니다.

다음 예제에서는 통합 이름, 소스 ARN 및 대상 ARN을 제공하여 통합을 생성합니다. 통합은 암호화되지 않습니다.

```
aws redshift create-integration \
--integration-name s3-integration \
--source-arn arn:aws:s3:us-east-1::s3-example-bucket \
--target-arn arn:aws:redshift:us-east-1:123456789012:namespace:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222
          {
    "IntegrationArn": "arn:aws:redshift:us-east-1:123456789012:integration:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "IntegrationName": "s3-integration",
    "SourceArn": "arn:aws:s3:::s3-example-bucket",
    "SourceType": "s3-event-notifications",
    "TargetArn": "arn:aws:redshift:us-east-1:123456789012:namespace:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "Status": "creating",
    "Errors": [],
    "CreateTime": "2024-10-09T19:08:52.758000+00:00",
    "Tags": []
}
```

다음 AWS CLI 명령을 사용하여 S3 이벤트 통합을 관리할 수도 있습니다.

- `delete-integration` - 통합 ARN을 지정하여 S3 이벤트 통합을 삭제합니다.
- `modify-integration` - 통합 ARN을 지정하여 S3 이벤트 통합의 이름 또는 설명(또는 둘 다)을 변경합니다.

- `describe-integrations` - 통합 ARN을 지정하여 S3 이벤트 통합의 속성을 봅니다.

이러한 명령에 대한 자세한 내용은 Amazon Redshift CLI Guide를 참조하세요.

그런 다음 Amazon Redshift는 연결된 소스 및 대상, 상태, 연결된 자동 복사 작업의 상태에 대한 정보와 함께 S3 이벤트 통합을 만듭니다. S3 이벤트 통합을 선택하고 세부 정보를 표시하도록 통합을 선택하여 Amazon Redshift 콘솔에서 S3 이벤트 통합에 대한 정보를 볼 수 있습니다. 통합은 내 계정 내에서 만들어진 것과 다른 계정에서 만들어진 것으로 구분됩니다. 내 계정 내 목록에는 소스와 대상이 동일한 계정에 있는 통합이 표시됩니다. 다른 계정에서 목록에는 다른 계정에서 소스를 소유한 통합이 표시됩니다.

S3 이벤트 통합을 삭제하면 해당 COPY JOB 상태가 1(활성)에서 0(비활성/보류)로 변경됩니다. 그러나 해당 COPY JOB은 자동으로 삭제되지 않습니다. 나중에 이름이 같은 COPY JOB을 생성하려고 하면 충돌이 발생할 수 있습니다.

COPY 작업 만들기 및 모니터링

통합이 만들어진 후 만든 통합에 대한 S3 이벤트 통합 세부 정보 페이지에서 자동 복사 작업 만들기를 선택하여 통합에 대한 자동 복사 작업을 만들 수 있는 Amazon Redshift Query Editor V2로 이동합니다. Amazon Redshift는 COPY JOB CREATE 문에서 FROM 절의 버킷을 S3 이벤트 통합에 사용되는 버킷과 매칭합니다. Amazon Redshift Query Editor V2에 대한 자세한 내용은 Amazon Redshift 관리 안내서의 Amazon Redshift Query Editor V2를 사용하여 데이터베이스 쿼리를 참조하세요. 예를 들어 Query Editor V2에서 다음 COPY 명령을 실행하여 Amazon S3 버킷 `s3://amzn-s3-demo-bucket/staging-folder`을 Amazon S3 이벤트 통합에 매칭하는 자동 COPY JOB을 만듭니다.

```
COPY public.target_table
FROM 's3://amzn-s3-demo-bucket/staging-folder'
IAM_ROLE 'arn:aws:iam::123456789012:role/MyLoadRoleName'
JOB CREATE my_copy_job_name
AUTO ON;
```

COPY 작업을 한 번 정의합니다. 향후 실행에도 동일한 파라미터가 사용됩니다.

COPY JOB을 정의하고 관리하려면 권한이 있어야 합니다. COPY JOB에 대한 권한 부여 및 취소에 대한 자세한 내용은 GRANT 및 REVOKE 섹션을 참조하세요. COPY JOB에 대한 범위 지정 권한 부여 및 취소에 대한 자세한 내용은 범위가 지정된 권한 부여 및 범위가 지정된 권한 취소 섹션을 참조하세요.

CREATE, LIST, SHOW, DROP, ALTER 및 RUN 작업에 대한 옵션을 사용하여 로드 작업을 관리합니다. 자세한 내용은 COPY JOB 단원을 참조하십시오.

COPY JOB 상태 및 진행률을 확인하기 위해 시스템 뷰를 쿼리할 수 있습니다. 뷰는 다음과 같이 제공됩니다.

- SYS_COPY_JOB - 현재 정의된 각 COPY JOB에 대한 행이 포함됩니다.
- SYS_COPY_JOB_DETAIL - 각 COPY JOB에 대해 보류 중인 파일, 오류 파일 및 수집된 파일에 대한 세부 정보를 포함합니다.
- SYS_COPY_JOB_INFO - COPY 작업에 대해 로깅된 메시지를 포함합니다.
- SYS_LOAD_HISTORY - COPY 명령의 세부 정보가 포함됩니다.
- SYS_LOAD_ERROR_DETAIL - COPY 명령 오류의 세부 정보가 포함됩니다.
- SVV_COPY_JOB_INTEGRATIONS - S3 이벤트 통합에 대한 세부 정보를 포함합니다.
- STL_LOAD_ERRORS - COPY 명령의 오류가 포함됩니다.
- STL_LOAD_COMMITS - COPY 명령 데이터 로드 문제를 해결하는 데 사용되는 정보가 포함됩니다.

S3 이벤트 통합 오류 문제 해결에 대한 자세한 내용은 S3 이벤트 통합 및 COPY JOB 오류 문제 해결 섹션을 참조하세요.

COPY JOB에 의해 로드된 파일 목록을 가져오려면 먼저 *<job_id>*를 대체하고 다음 SQL을 실행하세요.

```
SELECT job_id, job_name, data_source, copy_query, filename, status, curtime
FROM sys_copy_job copyjob
JOIN stl_load_commits loadcommit
ON copyjob.job_id = loadcommit.copy_job_id
WHERE job_id = <job_id>;
```

자동 복사를 위한 S3 이벤트 통합을 만들 때 제한 사항

자동 복사를 사용할 때는 다음을 고려하세요.

- AWS 계정에서 각 클러스터 또는 작업 그룹에 대해 최대 200개의 COPY JOB을 만들 수 있습니다.
- 각 Amazon S3 버킷에 대해 최대 35개의 복사 작업을 만들 수 있습니다.
- 각 Amazon Redshift 대상에 대해 최대 50개의 S3 이벤트 통합을 만들 수 있습니다.
- 버킷 이름에 마침표(.)가 있는 소스 Amazon S3 버킷과 S3 이벤트 통합을 만들 수 없습니다.

DML 명령을 사용하여 테이블 로드

Amazon Redshift는 테이블에서 행을 수정하는 데 사용할 수 있는 표준 데이터 조작 언어(DML) 명령(INSERT, UPDATE, DELETE)을 지원합니다. TRUNCATE 명령을 사용하여 빠른 대량 삭제를 수행할 수도 있습니다.

> **Note**
>
> 대량의 데이터를 로드하려면 COPY 명령을 사용하는 것이 가장 좋습니다. 테이블을 채우는 데 개별적인 INSERT문을 사용하는 방식은 엄청나게 느릴 수 있습니다. 데이터가 다른 Amazon Redshift 데이터베이스 테이블에 이미 있는 경우에는, INSERT INTO ... SELECT FROM 또는 CREATE TABLE AS를 사용하여 성능을 개선할 수 있습니다. 자세한 내용은 INSERT 또는 CREATE TABLE AS 섹션을 참조하세요.

테이블에서 변경 전에 비해 상당히 많은 수의 행을 삽입, 업데이트 또는 삭제하는 경우, 작업을 마치면 테이블에 대해 ANALYZE 및 VACUUM 명령을 실행합니다. 애플리케이션에서 작은 변경이 시간이 지나면서 누적되는 경우, 정기적으로 ANALYZE 및 VACUUM 명령이 실행되도록 예약하는 것이 좋습니다. 자세한 내용은 테이블 분석 및 테이블 Vacuum 섹션을 참조하세요.

주제

- 새 데이터 업데이트 및 삽입

새 데이터 업데이트 및 삽입

MERGE 명령을 사용하여 기존 테이블에 새 데이터를 효율적으로 추가할 수 있습니다. 스테이징 테이블을 만든 다음 이 섹션에 설명된 방법 중 하나를 사용하여 스테이징 테이블에서 대상 테이블을 업데이트하여 병합 작업을 수행합니다. MERGE 명령에 대한 자세한 내용은 MERGE 섹션을 참조하세요.

병합 예에서는 TICKIT 데이터 세트라고 하는 Amazon Redshift용 샘플 데이터 세트를 사용합니다. 필수 조건으로 일반 데이터베이스 작업 시작하기에서 제공되는 지침에 따라 TICKIT 테이블과 데이터를 설정할 수 있습니다. 샘플 데이터 세트에 대한 자세한 내용은 샘플 데이터베이스에서 확인할 수 있습니다.

병합 방법 1: 기존 행 교체

대상 테이블의 모든 열을 덮어쓰는 경우 병합을 수행하는 가장 빠른 방법은 기존 행을 바꾸는 것입니다. 이렇게 하면 업데이트할 행을 내부 조인을 사용하여 삭제하기 때문에 대상 테이블을 한 번만 스캔합니다. 행은 삭제된 후에 스테이징 테이블에서 단일 삽입 작업을 통해 새 행으로 교체됩니다.

다음이 모두 참인 경우, 이 방법을 사용하십시오.

- 대상 테이블과 스테이징 테이블에 동일한 열이 포함되어 있습니다.
- 대상 테이블 열의 모든 데이터를 스테이징 테이블의 모든 열로 교체하려 합니다.
- 병합에서 스테이징 테이블의 모든 열을 사용합니다.

이러한 기준 중 하나라도 해당되지 않는 경우 다음 섹션에 설명된 병합 방법 2: MERGE를 사용하지 않고 열 목록 지정을 사용하세요.

스테이징 테이블의 열 일부를 사용하지 않는 경우, 변경되지 않는 행을 WHERE 절을 이용해 배제하여 DELETE 및 INSERT 문을 필터링합니다. 하지만 스테이징 테이블의 열 대부분이 병합에 참여하지 않는다면 이 단원 후반부에서 설명하는 것처럼 별도의 단계에서 UPDATE와 INSERT를 수행하는 것이 좋습니다.

병합 방법 2: MERGE를 사용하지 않고 열 목록 지정

전체 행을 덮어쓰는 대신 대상 테이블의 특정 열을 업데이트하려면 이 방법을 사용합니다. 이 방법은 추가 업데이트 단계가 필요하고 MERGE 명령을 사용하지 않기 때문에 이전 방법보다 시간이 오래 걸립니다. 다음 중 참인 것이 있다면 이 방법을 사용하십시오.

- 대상 테이블의 모든 열을 업데이트하는 것이 아닙니다.
- 스테이징 테이블의 행 대부분이 업데이트에서 사용되지 않습니다.

주제

임시 스테이징 테이블 생성

스테이징 테이블은 업데이트와 삽입을 포함하여 대상 테이블을 변경하는 데 사용될 모든 데이터가 보관되는 임시 테이블입니다.

병합 작업에는 스테이징 테이블과 대상 테이블 간의 조인이 필요합니다. 조인 행을 배치하려면 스테이징 테이블의 분산 키를 대상 테이블의 분산 키와 동일한 열로 설정합니다. 예를 들어 대상 테이블이 외래 키 열을 분산 키로 사용하는 경우, 스테이징 테이블의 분산 키에 동일한 열을 사용하십시오. CREATE TABLE LIKE 문을 사용하여 스테이징 테이블을 생성하는 경우, 스테이징 테이블은 상위 테이블에서 분산 키를 상속합니다. CREATE TABLE AS 문을 사용하는 경우에는 새 테이블이 분산 키를 상속하지 않습니다. 자세한 내용은 쿼리 최적화를 위한 데이터 배포 섹션을 참조하세요.

분산 키가 기본 키와 동일하지 않고 병합 작업의 일환으로 분산 키가 업데이트되지 않는 경우, 분산 키 열에서 중복 조인 조건자를 추가하여 배치된 조인을 활성화합니다. 예:

```
where target.primarykey = stage.primarykey
and target.distkey = stage.distkey
```

쿼리가 배치된 조인을 사용할지 확인하려면 EXPLAIN과 함께 쿼리를 실행하고 모든 조인에서 DS_DIST_NONE을 확인합니다. 자세한 내용은 쿼리 계획 평가 섹션을 참조하세요.

기존 행을 교체하여 병합 작업 수행

해당 절차에 상세히 설명된 병합 작업을 실행할 때 모든 단계를 포함하되 단일 트랜잭션에서 임시 스테이징 테이블을 생성하거나 삭제할 때는 예외입니다. 임의 단계가 실패하면 트랜잭션이 롤백됩니다. 단일 트랜잭션을 사용하면 커밋의 수도 줄어들어 시간과 자원을 절약할 수 있습니다.

기존 행을 교체하여 병합 작업을 수행하려면

1. 다음 유사 코드에 나온 것처럼 스테이징 테이블을 생성한 다음 병합될 데이터로 테이블을 채웁니다.

   ```
   CREATE temp table stage (like target);

   INSERT INTO stage
   SELECT * FROM source
   WHERE source.filter = 'filter_expression';
   ```

2. MERGE를 사용하여 스테이징 테이블과 내부 조인을 수행하여 스테이징 테이블과 일치하는 대상 테이블의 행을 업데이트한 다음 스테이징 테이블과 일치하지 않는 나머지 모든 행을 대상 테이블에 삽입합니다.

업데이트 및 삽입 작업은 하나의 MERGE 명령으로 실행하는 것이 좋습니다.

```
MERGE INTO target
USING stage [optional alias] on (target.primary_key = stage.primary_key)
WHEN MATCHED THEN
UPDATE SET col_name1 = stage.col_name1 , col_name2= stage.col_name2, col_name3 =
 {expr}
WHEN NOT MATCHED THEN
INSERT (col_name1 , col_name2, col_name3) VALUES (stage.col_name1, stage.col_name2,
 {expr});
```

3. 스테이징 테이블을 삭제합니다.

```
DROP TABLE stage;
```

MERGE 명령을 사용하지 않고 열 목록을 지정하여 병합 작업 수행

절차에 자세히 설명된 병합 작업을 실행할 때는 모든 단계를 단일 트랜잭션에 포함시키십시오. 임의 단계가 실패하면 트랜잭션이 롤백됩니다. 단일 트랜잭션을 사용하면 커밋의 수도 줄어들어 시간과 자원을 절약할 수 있습니다.

열 목록을 지정하여 병합 작업을 수행하려면

1. 전체 작업을 단일 트랜잭션 블록에 넣습니다.

```
BEGIN transaction;
…
END transaction;
```

2. 다음 유사 코드에 나온 것처럼 스테이징 테이블을 생성한 다음 병합될 데이터로 테이블을 채웁니다.

```
create temp table stage (like target);
insert into stage
select * from source
where source.filter = 'filter_expression';
```

3. 스테이징 테이블과의 내부 조인을 사용하여 대상 테이블을 업데이트합니다.

   - UPDATE 절에서 업데이트될 열을 명시적으로 나열합니다.

- 스테이징 테이블과의 내부 조인을 수행합니다.
- 분산 키가 기본 키와 다르고 분산 키가 업데이트되지 않는 경우, 분산 키에서 중복 조인을 추가합니다. 쿼리가 배치된 조인을 사용할지 확인하려면 EXPLAIN과 함께 쿼리를 실행하고 모든 조인에서 DS_DIST_NONE을 확인합니다. 자세한 내용은 쿼리 계획 평가 섹션을 참조하세요.
- 대상 테이블이 타임스탬프 기준으로 정렬된 경우 대상 테이블에서 범위 제한 스캔을 활용하기 위해 술어를 추가합니다. 자세한 내용은 Amazon Redshift 쿼리 설계 모범 사례 단원을 참조하십시오.
- 병합에서 사용하지 않는 행이 있는 경우, 변경할 행을 필터링하기 위해 절을 추가합니다. 예를 들어 하나 이상의 열에서 부등식 필터를 추가하여 변경되지 않은 행을 제외합니다.
- 문제가 생기면 모든 것을 롤백할 수 있도록 업데이트 작업, 삭제 작업, 삽입 작업을 단일 트랜잭션 블록에 넣습니다.

예:

```
begin transaction;

update target
set col1 = stage.col1,
col2 = stage.col2,
col3 = 'expression'
from stage
where target.primarykey = stage.primarykey
and target.distkey = stage.distkey
and target.col3 > 'last_update_time'
and (target.col1 != stage.col1
or target.col2 != stage.col2
or target.col3 = 'filter_expression');
```

4. 대상 테이블과의 내부 조인을 사용하여 스테이징 테이블에서 필요 없는 행을 삭제합니다. 대상 테이블의 행 중에는 스테이징 테이블의 행과 이미 일치하는 행도 있고 이전 단계에서 업데이트된 행도 있습니다. 어느 경우건 이 행들은 삽입에 필요하지 않습니다.

```
delete from stage
using target
where stage.primarykey = target.primarykey;
```

5. 스테이징 테이블에서 나머지 행을 삽입합니다. VALUES 절에 2단계에서 UPDATE 문에 사용한 것과 동일한 열 목록을 사용합니다.

```
insert into target
(select col1, col2, 'expression'
from stage);

end transaction;
```

6. 스테이징 테이블을 삭제합니다.

```
drop table stage;
```

병합 예

다음 예는 병합을 수행하여 SALES 테이블을 업데이트합니다. 첫 번째 예는 대상 테이블에서 모든 행을 삭제한 다음 준비 열에서 삽입하는 보다 간단한 방법을 사용합니다. 두 번째 예는 대상 테이블의 선택된 열에서의 업데이트가 필요하므로 추가 업데이트 단계가 포함됩니다.

병합 예에서는 TICKIT 데이터 세트라고 하는 Amazon Redshift용 샘플 데이터 세트를 사용합니다. 필수 조건으로 일반 데이터베이스 작업 시작하기 가이드의 지침에 따라 TICKIT 테이블과 데이터를 설정할 수 있습니다. 샘플 데이터 세트에 대한 자세한 내용은 샘플 데이터베이스에서 확인할 수 있습니다.

데이터 원본 샘플 병합

이 단원의 예에는 업데이트와 삽입이 모두 포함된 샘플 데이터 원본이 필요합니다. 예를 위해 SALES 테이블의 데이터를 사용하는 SALES_UPDATE라는 이름의 샘플 테이블을 만듭니다. 12월 신규 영업 활동을 나타내는 무작위 데이터로 새 테이블을 채웁니다. 다음에 나오는 예에서는 SALES_UPDATE 샘플 테이블을 사용해 스테이징 테이블을 만듭니다.

```
-- Create a sample table as a copy of the SALES table.

create table tickit.sales_update as
select * from tickit.sales;

-- Change every fifth row to have updates.

update tickit.sales_update
set qtysold = qtysold*2,
pricepaid = pricepaid*0.8,
commission = commission*1.1
where saletime > '2008-11-30'
and mod(sellerid, 5) = 0;
```

```
-- Add some new rows to have inserts.
-- This example creates a duplicate of every fourth row.

insert into tickit.sales_update
select (salesid + 172456) as salesid, listid, sellerid, buyerid, eventid, dateid,
 qtysold, pricepaid, commission, getdate() as saletime
from tickit.sales_update
where saletime > '2008-11-30'
and mod(sellerid, 4) = 0;
```

매칭 키를 기반으로 기존 행을 교체하는 병합 예

다음 스크립트는 SALES_UPDATE 테이블을 사용해 12월 영업 활동의 새 데이터로 SALES 테이블에서 병합 작업을 수행합니다. 이 예제에서는 업데이트가 있는 SALES 테이블의 행을 바꿉니다. 이 예제에서는 판매 수량 및 가격 지불 열을 업데이트하지만 수수료 및 급여 시간은 변경하지 않습니다.

```
MERGE into tickit.sales
USING tickit.sales_update sales_update
on ( sales.salesid = sales_update.salesid
and sales.listid = sales_update.listid
and sales_update.saletime > '2008-11-30'
and (sales.qtysold != sales_update.qtysold
or sales.pricepaid != sales_update.pricepaid))
WHEN MATCHED THEN
update SET qtysold = sales_update.qtysold,
pricepaid = sales_update.pricepaid
WHEN NOT MATCHED THEN
INSERT (salesid, listid, sellerid, buyerid, eventid, dateid, qtysold , pricepaid,
 commission, saletime)
values (sales_update.salesid, sales_update.listid, sales_update.sellerid,
 sales_update.buyerid, sales_update.eventid,
sales_update.dateid, sales_update.qtysold , sales_update.pricepaid,
 sales_update.commission, sales_update.saletime);

-- Drop the staging table.
drop table tickit.sales_update;

-- Test to see that commission and salestime were not impacted.
SELECT sales.salesid, sales.commission, sales.salestime, sales_update.commission,
 sales_update.salestime
FROM tickit.sales
INNER JOIN tickit.sales_update sales_update
```

```
ON
sales.salesid = sales_update.salesid
AND sales.listid = sales_update.listid
AND sales_update.saletime > '2008-11-30'
AND (sales.commission != sales_update.commission
OR sales.salestime != sales_update.salestime);
```

MERGE를 사용하지 않고 열 목록을 지정하는 병합 예

다음 예는 병합 작업을 수행하여 12월 영업 활동의 새 데이터로 SALES를 업데이트합니다. 변경되지 않은 행과 함께 업데이트와 삽입이 모두 포함된 샘플 데이터가 필요합니다. 이 예에서는 QTYSOLD 및 PRICEPAID 열은 업데이트하되 COMMISSION과 SALETIME은 변경하지 않으려 합니다. 다음 스크립트는 SALES_UPDATE 테이블을 사용해 SALES 테이블에서 병합 작업을 수행합니다.

```
-- Create a staging table and populate it with rows from SALES_UPDATE for Dec
create temp table stagesales as select * from sales_update
where saletime > '2008-11-30';

-- Start a new transaction
begin transaction;

-- Update the target table using an inner join with the staging table
-- The join includes a redundant predicate to collocate on the distribution key -- A
 filter on saletime enables a range-restricted scan on SALES

update sales
set qtysold = stagesales.qtysold,
pricepaid = stagesales.pricepaid
from stagesales
where sales.salesid = stagesales.salesid
and sales.listid = stagesales.listid
and stagesales.saletime > '2008-11-30'
and (sales.qtysold != stagesales.qtysold
or sales.pricepaid != stagesales.pricepaid);

-- Delete matching rows from the staging table
-- using an inner join with the target table

delete from stagesales
using sales
where sales.salesid = stagesales.salesid
and sales.listid = stagesales.listid;
```

```
-- Insert the remaining rows from the staging table into the target table
insert into sales
select * from stagesales;

-- End transaction and commit
end transaction;

-- Drop the staging table
drop table stagesales;
```

전체 복사 수행

전체 복사는 대량 삽입을 사용하여 테이블을 다시 생성하고 다시 채움으로써 테이블을 자동으로 정렬합니다. 테이블에 정렬되지 않은 큰 리전이 있는 경우, 전체 복사가 vacuum보다 훨씬 빠릅니다. 딥 카피 작업 중에는 추적할 수 있는 경우에만 동시 업데이트를 수행하는 것이 좋습니다. 프로세스가 완료되면 델타 업데이트를 새 테이블로 옮깁니다. VACUUM 작업은 동시 업데이트를 자동으로 지원합니다.

다음 방법 중 하나를 선택해 원본 테이블의 사본을 만들 수 있습니다.

- 원본 테이블 DDL을 사용합니다.

  CREATE TABLE DDL을 사용할 수 있는 경우, 가장 빠르고 선호되는 방법입니다. 새 테이블을 만들면 기본 키와 외래 키를 비롯해 모든 테이블 및 열 속성을 지정할 수 있습니다. SHOW TABLE 함수를 사용하여 원본 DDL을 찾을 수 있습니다.
- CREATE TABLE LIKE를 사용합니다.

  원본 DDL을 사용할 수 없는 경우, CREATE TABLE LIKE를 사용해 원본 테이블을 다시 생성할 수 있습니다. 새 테이블은 상위 테이블의 인코딩, 배포 키, 정렬 키 및 null이 아닌 속성을 상속합니다. 새 테이블은 상위 테이블의 기본 키 및 외래 키 속성은 상속하지 않지만 ALTER TABLE을 사용해 추가할 수 있습니다.
- 임시 테이블을 생성하고 원본 테이블을 자릅니다.

  상위 테이블의 기본 키와 외래 키 특성을 유지해야 하는 경우. 상위 테이블에 종속성이 있는 경우 CREATE TABLE ... AS(CTAS)를 사용하여 임시 테이블을 만들 수 있습니다. 그런 다음 원본 테이블을 자르고 임시 테이블을 이용해 원본 테이블을 채웁니다.

  임시 테이블을 사용하면 영구 테이블을 사용하는 경우에 비해 성능이 대폭 향상되지만 데이터가 손실될 위험이 있습니다. 임시 테이블은 자신이 생성된 세션이 끝날 때 자동으로 삭제됩니다. TRUNCATE는 트랜잭션 블록 내에 있더라도 즉시 커밋됩니다. TRUNCATE가 성공했지만 후속

INSERT가 완료되기 전에 세션이 종료되면 데이터가 손실됩니다. 데이터 손실을 허용할 수 없다면 영구 테이블을 사용합니다.

테이블의 복사본을 생성한 후 새 테이블에 대한 액세스 권한을 부여해야 할 수 있습니다. GRANT를 사용하여 액세스 권한을 정의할 수 있습니다. 테이블의 모든 액세스 권한을 보고 부여하려면 다음 중 하나여야 합니다.

- 슈퍼 사용자
- 복사할 테이블의 소유자
- 테이블의 권한을 볼 수 있는 ACCESS SYSTEM TABLE 권한과, 모든 관련 권한을 부여할 권한이 있는 사용자

또한 딥 카피가 존재하는 스키마에 대한 사용 권한을 부여해야 할 수도 있습니다. 딥 카피의 스키마가 원본 테이블의 스키마와 다르며 `public` 스키마도 아닌 경우에는 사용 권한을 부여해야 합니다. 사용 권한을 보고 부여하려면 다음 중 하나여야 합니다.

- 슈퍼 사용자
- 딥 카피의 스키마에 대한 USAGE 권한을 부여할 수 있는 사용자

원본 테이블 DDL을 사용하여 전체 복사를 수행하려면

1. (선택적) `v_generate_tbl_ddl`이라는 스크립트를 실행하여 테이블 DDL을 다시 생성합니다.
2. 원본 CREATE TABLE DDL을 사용하여 테이블 사본을 만듭니다.
3. INSERT INTO … SELECT 문을 사용하여 원본 테이블의 데이터로 사본을 채웁니다.
4. 이전 테이블에 부여된 권한을 확인합니다. 이러한 권한은 SVV_RELATION_PRIVILEGES 시스템 보기에서 확인할 수 있습니다.
5. 필요한 경우 이전 테이블의 권한을 새 테이블에 부여합니다.
6. 원래 테이블에 권한이 있는 모든 그룹과 사용자에게 사용 권한을 부여합니다. 딥 카피 테이블이 `public` 스키마에 있거나 원본 테이블과 동일한 스키마에 있는 경우에는 이 단계를 수행하지 않아도 됩니다.
7. 원본 테이블을 삭제합니다.
8. ALTER TABLE 문을 사용하여 사본의 이름을 원본 테이블 이름으로 변경합니다.

다음 예는 sample_copy라는 이름의 SAMPLE 복제본을 사용하여 SAMPLE 테이블에서 전체 복사를 수행합니다.

```
--Create a copy of the original table in the sample_namespace namespace using the
 original CREATE TABLE DDL.
create table sample_namespace.sample_copy ( … );

--Populate the copy with data from the original table in the public namespace.
insert into sample_namespace.sample_copy (select * from public.sample);

--Check SVV_RELATION_PRIVILEGES for the original table's privileges.
select * from svv_relation_privileges where namespace_name = 'public' and relation_name
 = 'sample' order by identity_type, identity_id, privilege_type;

--Grant the original table's privileges to the copy table.
grant DELETE on table sample_namespace.sample_copy to group group1;
grant INSERT, UPDATE on table sample_namespace.sample_copy to group group2;
grant SELECT on table sample_namespace.sample_copy to user1;
grant INSERT, SELECT, UPDATE on table sample_namespace.sample_copy to user2;

--Grant usage permission to every group and user that has privileges in the original
 table.
grant USAGE on schema sample_namespace to group group1, group group2, user1, user2;

--Drop the original table.
drop table public.sample;

--Rename the copy table to match the original table's name.
alter table sample_namespace.sample_copy rename to sample;
```

CREATE TABLE LIKE를 사용하여 전체 복사를 수행하려면

1. CREATE TABLE LIKE를 사용하여 새 테이블을 생성합니다.
2. INSERT INTO … SELECT 문을 사용하여 현재 테이블에서 새 테이블로 행을 복사합니다.
3. 이전 테이블에 부여된 권한을 확인합니다. 이러한 권한은 SVV_RELATION_PRIVILEGES 시스템 보기에서 확인할 수 있습니다.
4. 필요한 경우 이전 테이블의 권한을 새 테이블에 부여합니다.
5. 원래 테이블에 권한이 있는 모든 그룹과 사용자에게 사용 권한을 부여합니다. 딥 카피 테이블이 public 스키마에 있거나 원본 테이블과 동일한 스키마에 있는 경우에는 이 단계를 수행하지 않아도 됩니다.

6. 현재 테이블을 삭제합니다.
7. ALTER TABLE 문을 사용하여 새 테이블의 이름을 원본 테이블 이름으로 변경합니다.

다음 예는 CREATE TABLE LIKE를 사용하여 SAMPLE 테이블에서 전체 복사를 수행합니다.

```
--Create a copy of the original table in the sample_namespace namespace using CREATE
 TABLE LIKE.
create table sameple_namespace.sample_copy (like public.sample);

--Populate the copy with data from the original table.
insert into sample_namespace.sample_copy (select * from public.sample);

--Check SVV_RELATION_PRIVILEGES for the original table's privileges.
select * from svv_relation_privileges where namespace_name = 'public' and relation_name
 = 'sample' order by identity_type, identity_id, privilege_type;

--Grant the original table's privileges to the copy table.
grant DELETE on table sample_namespace.sample_copy to group group1;
grant INSERT, UPDATE on table sample_namespace.sample_copy to group group2;
grant SELECT on table sample_namespace.sample_copy to user1;
grant INSERT, SELECT, UPDATE on table sample_namespace.sample_copy to user2;

--Grant usage permission to every group and user that has privileges in the original
 table.
grant USAGE on schema sample_namespace to group group1, group group2, user1, user2;

--Drop the original table.
drop table public.sample;

--Rename the copy table to match the original table's name.
alter table sample_namespace.sample_copy rename to sample;
```

임시 테이블을 생성하고 원본 테이블을 잘라 전체 복사를 수행하려면

1. CREATE TABLE AS를 사용하여 원본 테이블의 행으로 임시 테이블을 생성합니다.
2. 현재 테이블을 자릅니다.
3. INSERT INTO … SELECT 문을 사용하여 임시 테이블에서 원본 테이블로 행을 복사합니다.
4. 임시 테이블을 삭제합니다.

다음 예는 임시 테이블을 생성하고 원본 테이블을 잘라 SALES 테이블에서 전체 복사를 수행합니다. 원본 테이블은 그대로 유지되므로 복사 테이블에 권한을 부여하지 않아도 됩니다.

```
--Create a temp table copy using CREATE TABLE AS.
create temp table salestemp as select * from sales;

--Truncate the original table.
truncate sales;

--Copy the rows from the temporary table to the original table.
insert into sales (select * from salestemp);

--Drop the temporary table.
drop table salestemp;
```

테이블 분석

ANALYZE 작업은 쿼리 플래너가 최적의 계획을 선택하는 데 사용하는 통계 메타데이터를 업데이트합니다.

대부분의 경우 ANALYZE 명령을 명시적으로 실행할 필요가 없습니다. Amazon Redshift는 워크로드의 변경 사항을 모니터링하고 백그라운드에서 통계를 자동으로 업데이트합니다. 또한 COPY 명령은 빈 테이블로 데이터 로드 시 분석을 자동으로 수행합니다.

테이블 또는 전체 데이터베이스를 명시적으로 분석하려면 ANALYZE 명령을 실행합니다.

자동 분석

Amazon Redshift는 데이터베이스를 지속적으로 모니터링하고 백그라운드에서 분석 작업을 자동으로 수행합니다. 시스템 성능에 대한 영향을 최소화하기 위해 자동 분석은 워크로드가 적을 때 실행됩니다.

자동 분석은 기본적으로 활성화되어 있습니다. 자동 분석을 끄려면 클러스터의 파라미터 그룹을 수정하여 `auto_analyze` 파라미터를 **`false`**로 설정합니다.

처리 시간을 줄이고 전반적인 시스템 성능을 개선하기 위해 Amazon Redshift에서는 수정 범위가 작은 테이블에 대한 자동 분석은 건너뜁니다.

분석 작업은 통계가 최신 상태인 테이블은 건너뜁니다. ETL(추출, 변환, 로드) 워크플로우의 일부로 ANALYZE를 실행하면 자동 분석이 통계가 최신 상태인 테이블을 건너뜁니다. 마찬가지로, 자동 분석이 테이블의 통계를 업데이트한 경우에는 명시적 ANALYZE가 해당 테이블을 건너뜁니다.

새 테이블 데이터의 분석

기본적으로 COPY 명령은 데이터를 빈 테이블로 로드한 후 ANALYZE를 수행합니다. STATUPDATE를 ON으로 설정하면 테이블이 비어 있는지 여부에 상관없이 ANALYZE를 수행하도록 지정할 수 있습니다. STATUPDATE를 OFF로 지정하면 ANALYZE가 수행되지 않습니다. 테이블 소유자 또는 수퍼유저만이 ANALYZE 명령을 실행하거나 STATUPDATE가 ON으로 설정된 상태에서 COPY 명령을 실행할 수 있습니다.

Amazon Redshift는 다음 명령을 사용하여 생성하는 새 테이블을 분석합니다.

- CREATE TABLE AS (CTAS)
- CREATE TEMP TABLE AS
- SELECT INTO

처음 데이터가 로드된 후 분석되지 않은 새 테이블에 대해 쿼리를 실행하면 Amazon Redshift에서는 경고 메시지를 반환합니다. 후속 업데이트 또는 로드 후 테이블을 쿼리하면 경고가 발생하지 않습니다. 분석되지 않은 테이블을 참조하는 쿼리에 대해 EXPLAIN 명령을 실행하면 동일한 경고 메시지가 반환됩니다.

비어 있지 않은 테이블에 데이터를 추가해 테이블의 크기가 크게 변경되면 항상 통계를 명시적으로 업데이트할 수 있습니다. ANALYZE 명령을 실행하거나 COPY 명령과 함께 STATUPDATE ON 옵션을 사용하여 업데이트할 수 있습니다. 마지막 ANALYZE 이후 삽입되거나 삭제된 행의 수에 대한 세부 정보를 보려면 PG_STATISTIC_INDICATOR 시스템 카탈로그 테이블을 쿼리합니다.

ANALYZE 명령의 범위는 다음 중 하나로 지정할 수 있습니다.

- 전체 현재 데이터베이스
- 단일 테이블
- 단일 테이블의 하나 이상의 특정 열
- 쿼리에서 조건자로 사용될 수 있는 열

ANALYZE 명령은 테이블에서 행의 샘플을 가져오고, 몇 가지 계산을 하며, 그 결과로 나온 열 통계를 저장합니다. 기본적으로 Amazon Redshift는 DISTKEY 열에 대해 샘플 패스를 실행하고, 테이블의 다른 모든 열에 대해 또 다른 샘플 패스를 실행합니다. 열의 하위 세트에 대한 통계를 생성하려면 쉼표로 구분된 열 목록을 지정할 수 있습니다. PREDICATE COLUMNS 절과 함께 ANALYZE를 실행해 조건자로 사용되지 않는 열은 건너뛸 수 있습니다.

ANALYZE 작업은 리소스를 많이 사용하므로 실제로 통계 업데이트가 필요한 테이블과 열에서만 실행하십시오. 정기적으로 또는 같은 일정으로 모든 테이블의 모든 열을 분석할 필요는 없습니다. 데이터가 대폭 변경되는 경우, 다음에서 자주 사용되는 열을 분석합니다.

- 정렬 및 그룹화 작업
- 조인
- 쿼리 조건자

Amazon Redshift는 처리 시간을 줄이고 전체 시스템 성능을 높이기 위해 analyze_threshold_percent 파라미터에서 결정하는 대로 변경되는 행 비율이 낮은 테이블은 ANALYZE를 건너뜁니다. 기본적으로 분석 임계값은 10퍼센트로 설정됩니다. SET 명령을 실행하면 현재 세션의 분석 임계값을 변경할 수 있습니다.

빈번히 분석할 필요성이 낮은 열은 큰 VARCHAR 열 등 실제로 결코 쿼리되지 않는 사실과 측정값 및 관련 속성을 나타내는 열입니다. 예를 들어 TICKIT 데이터베이스의 LISTING 테이블을 고려해 보십시오.

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'listing';


column         |        type        | encoding | distkey | sortkey
---------------+--------------------+----------+---------+---------
listid         | integer            | none     | t       | 1
sellerid       | integer            | none     | f       | 0
eventid        | integer            | mostly16 | f       | 0
dateid         | smallint           | none     | f       | 0
numtickets     | smallint           | mostly8  | f       | 0
priceperticket | numeric(8,2)       | bytedict | f       | 0
totalprice     | numeric(8,2)       | mostly32 | f       | 0
listtime       | timestamp with...  | none     | f       | 0
```

이 테이블이 많은 수의 새 레코드와 함께 매일 로드된다면 쿼리에서 조인 키로 자주 사용되는 LISTID 테이블을 정기적으로 분석해야 합니다. TOTALPRICE와 LISTTIME이 쿼리에서 자주 사용되는 제약이라면 평일에는 항상 이 열들과 분산 키를 분석할 수 있습니다.

```
analyze listing(listid, totalprice, listtime);
```

애플리케이션에서 판매자와 이벤트가 훨씬 정적이고 날짜 ID가 2년 또는 3년만을 포함하는 고정된 날짜 집합을 가리킨다고 가정해 보겠습니다. 이 경우, 이러한 열의 고유한 값은 크게 변하지 않습니다. 하지만 각 고유 값의 인스턴스 수는 꾸준히 증가합니다.

그 밖에도 NUMTICKETS 및 PRICEPERTICKET 지표에 대한 쿼리 주기가 TOTALPRICE 열에 비해 드문 경우를 생각해보겠습니다. 이러한 경우에는 주말마다 한 번 전체 테이블에서 ANALYZE 명령을 실행해 매일 분석되지 않는 5개 열의 통계를 업데이트할 수 있습니다.

조건자 열

열 목록을 지정하는 대신 편하게 조건자로 사용될 수 있는 열만 분석하는 방법도 있습니다. 쿼리를 실행할 때 join, filter condition 또는 group by 절에서 사용되는 열은 모두 시스템 카탈로그에서 조건자 열로 표시됩니다. PREDICATE COLUMNS 절을 사용하여 ANALYZE를 실행하면 분석 작업에 다음 기준을 만족하는 열만 포함됩니다.

- 조건자 열로 표시되는 열
- 분산 키인 열
- 정렬 키에 포함되는 열

테이블에서 조건자로 표시되는 열이 하나도 없으면 PREDICATE COLUMNS로 지정되더라도 ANALYZE에 모든 열이 포함됩니다. 조건자 열로 표시되는 열이 없는 이유는 테이블에 대한 쿼리를 아직 실행하지 않았기 때문일 수 있습니다.

워크로드의 쿼리 패턴이 비교적 안정적일 때는 PREDICATE COLUMNS를 사용할 수 있습니다. 다른 열이 조건자로 사용되는 경우가 빈번하여 쿼리 패턴이 가변적일 때 PREDICATE COLUMNS를 사용하면 일시적으로 시간이 경과된(stale) 통계가 나타날 수 있습니다. 시간이 경과된 통계는 최적화되지 못한 쿼리 런타임 계획으로 이어져 런타임이 길어지게 됩니다. 하지만 다음에 PREDICATE COLUMNS를 사용하여 ANALYZE를 실행할 때는 새로운 조건자 열이 포함됩니다.

조건자 열에 대한 세부 정보를 보려면 다음 SQL을 사용하여 PREDICATE_COLUMNS라는 이름으로 뷰를 생성하십시오.

```
CREATE VIEW predicate_columns AS
WITH predicate_column_info as (
SELECT ns.nspname AS schema_name, c.relname AS table_name, a.attnum as col_num,
 a.attname as col_name,
        CASE
            WHEN 10002 = s.stakind1 THEN array_to_string(stavalues1, '||')
            WHEN 10002 = s.stakind2 THEN array_to_string(stavalues2, '||')
            WHEN 10002 = s.stakind3 THEN array_to_string(stavalues3, '||')
```

```
            WHEN 10002 = s.stakind4 THEN array_to_string(stavalues4, '||')
            ELSE NULL::varchar
        END AS pred_ts
   FROM pg_statistic s
   JOIN pg_class c ON c.oid = s.starelid
   JOIN pg_namespace ns ON c.relnamespace = ns.oid
   JOIN pg_attribute a ON c.oid = a.attrelid AND a.attnum = s.staattnum)
SELECT schema_name, table_name, col_num, col_name,
       pred_ts NOT LIKE '2000-01-01%' AS is_predicate,
       CASE WHEN pred_ts NOT LIKE '2000-01-01%' THEN (split_part(pred_ts,
 '||',1))::timestamp ELSE NULL::timestamp END as first_predicate_use,
       CASE WHEN pred_ts NOT LIKE '%||2000-01-01%' THEN (split_part(pred_ts,
 '||',2))::timestamp ELSE NULL::timestamp END as last_analyze
FROM predicate_column_info;
```

LISING 테이블에 대해 다음 쿼리를 실행한다고 가정하겠습니다. 단, LISTID, LISTTIME 및 EVENTID는 join, filter 및 group by 절에 사용됩니다.

```
select s.buyerid,l.eventid, sum(l.totalprice)
from listing l
join sales s on l.listid = s.listid
where l.listtime > '2008-12-01'
group by l.eventid, s.buyerid;
```

다음 예와 같이 PREDICATE_COLUMNS 뷰에 대한 쿼리를 실행하면 LISTID, EVENTID 및 LISTTIME이 조건자 열로 표시되어 있는 것을 볼 수 있습니다.

```
select * from predicate_columns
where table_name = 'listing';
```

```
schema_name | table_name | col_num | col_name       | is_predicate |
 first_predicate_use | last_analyze
------------+------------+---------+----------------+--------------
+---------------------+---------------------
public      | listing    |       1 | listid         | true         | 2017-05-05
 19:27:59 | 2017-05-03 18:27:41
public      | listing    |       2 | sellerid       | false        |
  | 2017-05-03 18:27:41
public      | listing    |       3 | eventid        | true         | 2017-05-16
 20:54:32 | 2017-05-03 18:27:41
public      | listing    |       4 | dateid         | false        |
  | 2017-05-03 18:27:41
```

```
public      | listing    |       5 | numtickets     | false        |
  | 2017-05-03 18:27:41
public      | listing    |       6 | priceperticket | false        |
  | 2017-05-03 18:27:41
public      | listing    |       7 | totalprice     | false        |
  | 2017-05-03 18:27:41
public      | listing    |       8 | listtime       | true         | 2017-05-16
 20:54:32 | 2017-05-03 18:27:41
```

통계를 최신 상태로 유지하면 쿼리 플래너가 최적의 계획을 선택할 수 있으므로 쿼리 성능이 향상됩니다. Amazon Redshift는 백그라운드에서 통계를 자동으로 새로 고치며 사용자가 ANALYZE 명령을 명시적으로 실행할 수도 있습니다. ANALYZE를 명시적으로 실행하도록 선택한 경우 다음을 수행합니다.

- 쿼리를 실행하기 전에 ANALYZE 명령을 실행합니다.
- 정기적인 로드 또는 업데이트 사이클이 끝날 때마다 일상적으로 데이터베이스에 대해 ANALYZE 명령을 실행합니다.
- 새로 생성한 모든 테이블과 상당한 변경이 있는 모든 기존 테이블이나 열에 대해 ANALYZE 명령을 실행합니다.
- 쿼리에서의 사용 및 변경 경향에 따라 테이블과 열의 형식마다 일정을 달리하여 ANALYZE 작업을 실행하는 것을 고려해 보십시오.
- ANALYZE를 실행할 때 시간과 클러스터 리소스를 절약하려면 PREDICATE COLUMNS 절을 사용하십시오.

프로비저닝된 클러스터 또는 서버리스 네임스페이스에 스냅샷을 복원하거나 일시 중지된 프로비저닝된 클러스터를 다시 시작한 후에는 ANALYZE 명령을 명시적으로 실행할 필요가 없습니다. 이러한 경우 Amazon Redshift는 시스템 테이블 정보를 보존하므로 수동 ANALYZE 명령이 필요하지 않습니다. Amazon Redshift는 필요에 따라 자동 분석 작업을 계속 실행합니다.

분석 작업은 통계가 최신 상태인 테이블은 건너뜁니다. ETL(추출, 변환, 로드) 워크플로우의 일부로 ANALYZE를 실행하면 자동 분석이 통계가 최신 상태인 테이블을 건너뜁니다. 마찬가지로, 자동 분석이 테이블의 통계를 업데이트한 경우에는 명시적 ANALYZE가 해당 테이블을 건너뜁니다.

ANALYZE 명령 이력

테이블이나 데이터베이스에서 언제 마지막으로 ANALYZE 명령이 실행되었는지 알면 도움이 됩니다. ANALYZE 명령이 실행될 때 Amazon Redshift는 다음과 비슷한 여러 쿼리를 실행합니다.

```
padb_fetch_sample: select * from table_name
```

STL_ANALYZE에 대한 쿼리를 실행하여 분석 작업 이력을 확인합니다. Amazon Redshift가 자동 분석을 사용해 테이블을 분석하면 is_background 열이 t(true)로 설정됩니다. 그렇지 않으면 f(false)로 설정됩니다. 다음은 STV_TBL_PERM을 조인하여 테이블 이름과 런타임 세부 정보를 표시하는 예입니다.

```
select distinct a.xid, trim(t.name) as name, a.status, a.rows, a.modified_rows,
 a.starttime, a.endtime
from stl_analyze a
join stv_tbl_perm t  on t.id=a.table_id
where name = 'users'
order by starttime;


xid    | name  | status          | rows  | modified_rows | starttime           |
 endtime
-------+-------+-----------------+-------+---------------+---------------------
+---------------------
  1582 | users | Full            | 49990 |         49990 | 2016-09-22 22:02:23 |
 2016-09-22 22:02:28
244287 | users | Full            | 24992 |         74988 | 2016-10-04 22:50:58 |
 2016-10-04 22:51:01
244712 | users | Full            | 49984 |         24992 | 2016-10-04 22:56:07 |
 2016-10-04 22:56:07
245071 | users | Skipped         | 49984 |             0 | 2016-10-04 22:58:17 |
 2016-10-04 22:58:17
245439 | users | Skipped         | 49984 |          1982 | 2016-10-04 23:00:13 |
 2016-10-04 23:00:13
(5 rows)
```

또는 모든 트랜잭션에서 실행된, ANALYZE 명령이 포함된 모든 문을 반환하는 보다 복잡한 쿼리를 실행할 수도 있습니다.

```
select xid, to_char(starttime, 'HH24:MM:SS.MS') as starttime,
datediff(sec,starttime,endtime ) as secs, substring(text, 1, 40)
from svl_statementtext
where sequence = 0
and xid in (select xid from svl_statementtext s where s.text like 'padb_fetch_sample
%' )
order by xid desc, starttime;
```

```
xid  |  starttime   | secs |                  substring
-----+--------------+------+-------------------------------------------
1338 | 12:04:28.511 |    4 | Analyze date
1338 | 12:04:28.511 |    1 | padb_fetch_sample: select count(*) from
1338 | 12:04:29.443 |    2 | padb_fetch_sample: select * from date
1338 | 12:04:31.456 |    1 | padb_fetch_sample: select * from date
1337 | 12:04:24.388 |    1 | padb_fetch_sample: select count(*) from
1337 | 12:04:24.388 |    4 | Analyze sales
1337 | 12:04:25.322 |    2 | padb_fetch_sample: select * from sales
1337 | 12:04:27.363 |    1 | padb_fetch_sample: select * from sales
...
```

테이블 Vacuum

Amazon Redshift는 백그라운드의 테이블에서 VACUUM DELETE 작업을 자동으로 정렬하고 수행할 수 있습니다. 로드 또는 일련의 증분적 업데이트 후 테이블을 정리하기 위해 전체 데이터베이스나 개별 테이블에 대해 VACUUM 명령을 실행할 수도 있습니다.

> **Note**
>
> 필요한 테이블 권한이 있는 사용자만 테이블을 유효하게 정리(vacuum)할 수 있습니다. 필요한 테이블 권한 없이 VACUUM을 실행할 경우 작업은 성공적으로 완료되지만 아무런 효과도 없습니다. VACUUM을 유효하게 실행하기 위한 유효한 테이블 권한 목록은 VACUUM 섹션을 참조하세요.
>
> 이러한 이유로 필요에 따라 개별 테이블에 대해 VACUUM을 실행하는 것이 좋습니다. 또한 전체 데이터베이스에 대해 vacuum을 실행하면 비용이 많이 필요하므로 이 방법을 권장합니다.

자동 테이블 정렬

Amazon Redshift는 백그라운드에서 데이터를 자동으로 정렬하여 정렬 키 순서대로 테이블 데이터를 유지합니다. Amazon Redshift는 스캔 쿼리를 추적하여 정렬이 유용한 테이블 섹션을 결정합니다.

시스템의 로드에 따라 Amazon Redshift가 자동으로 정렬을 시작합니다. 이 자동 정렬은 데이터를 정렬 키 순서로 유지하기 위해 VACUUM 명령을 실행할 필요성을 줄입니다. 예를 들어 대량의 데이터 로드 후 정렬 키 순서로 전체 정렬된 데이터가 필요한 경우에도 VACUUM 명령을 수동으로 실행할 수 있습니다. 테이블이 VACUUM SORT 실행을 통해 이익을 얻을 수 있는지 확인하려면 SVV_TABLE_INFO의 `vacuum_sort_benefit` 열을 모니터링하십시오.

Amazon Redshift는 각 테이블에서 정렬 키를 사용하는 스캔 쿼리를 추적합니다. Amazon Redshift는 각 테이블에 대한 데이터 스캔 및 필터링 개선의 최대 비율을 추정합니다(테이블이 전체 정렬된 경우). 이 추정값은 SVV_TABLE_INFO의 vacuum_sort_benefit 열에서 볼 수 있습니다. 이 열을 unsorted 열과 함께 사용하면 테이블에서 VACUUM SORT를 수동으로 실행하여 쿼리가 이익을 얻을 수 있는 시기를 결정할 수 있습니다. 이 unsorted 열은 테이블의 물리적 정렬 순서를 반영합니다. vacuum_sort_benefit 열은 VACUUM SORT를 수동으로 실행하여 테이블 정렬의 영향을 지정합니다.

예를 들어 다음 쿼리를 고려해 보십시오.

```
select "table", unsorted,vacuum_sort_benefit from svv_table_info order by 1;
```

```
 table | unsorted | vacuum_sort_benefit
-------+----------+---------------------
 sales |    85.71 |                5.00
 event |    45.24 |               67.00
```

테이블 "sales"의 경우 테이블의 ~86%가 물리적으로 정렬되지 않은 경우에도 86%의 정렬되지 않은 테이블로부터 얻는 쿼리 성능 영향은 5%에 불과합니다. 이는 쿼리에서 테이블의 일부에만 액세스하거나 테이블에 액세스하는 쿼리가 거의 없기 때문일 수 있습니다. “event” 테이블의 경우, 테이블은 ~45%가 물리적으로 정렬되지 않았습니다. 그러나 쿼리 성능 영향이 67%라는 의미는 쿼리에서 테이블의 많은 부분에 액세스하거나 테이블에 액세스하는 쿼리 수가 많음을 나타냅니다. "event" 테이블은 VACUUM SORT를 실행할 때 이익을 얻을 가능성이 있습니다.

자동 vacuum 삭제

삭제를 수행하면 행이 삭제 표시되지만 제거되지는 않습니다. Amazon Redshift는 데이터베이스 테이블에서 삭제된 행 수를 기준으로 백그라운드에서 VACUUM DELETE 작업을 자동으로 실행합니다. Amazon Redshift는 로드가 감소한 기간 동안 VACUUM DELETE가 실행되도록 예약하고 부하가 많은 기간 동안 작업을 일시 중지합니다.

주제

- VACUUM 빈도
- 정렬 단계 및 병합 단계
- vacuum 임계값
- vacuum 유형

- Vacuum 시간 최소화

VACUUM 빈도

일관된 쿼리 성능을 유지하기 위해서는 필요할 때마다 자주 vacuum을 수행해야 합니다. VACUUM 명령을 얼마나 자주 실행할지 결정할 때는 다음 요인을 고려하십시오.

- 저녁이나 지정된 데이터베이스 관리 기간 같이 클러스터에서의 활동이 최소일 것으로 예상되는 기간에 VACUUM을 실행합니다.
- 유지 관리 기간 외 시간에 VACUUM 명령을 실행합니다. 자세한 내용은 유지 관리 기간 관련 일정을 참조하세요.
- 정렬되지 않은 리전이 많으면 vacuum 시간이 길어집니다. vacuum을 연기하면 다시 정리해야 하는 데이터가 많아지기 때문에 vacuum에 더 많은 시간이 소요됩니다.
- VACUUM은 I/O를 많이 사용하는 작업이므로 vacuum 완료까지 걸리는 시간이 길어질수록 클러스터에서 실행되는 동시 쿼리와 그 밖의 데이터베이스 작업에 미치는 영향이 커집니다.
- 인터리브 정렬을 사용하는 테이블의 경우, VACUUM 시간이 길어집니다. 인터리브 테이블을 다시 정렬해야 하는지 평가하려면 SVV_INTERLEAVED_COLUMNS 보기를 쿼리합니다.

정렬 단계 및 병합 단계

Amazon Redshift는 2단계로 vacuum 작업을 수행합니다. 먼저 정렬되지 않은 리전의 행을 정렬한 다음 필요할 경우 테이블 끝에 있는 새로 정렬된 행을 기존의 행과 병합합니다. 큰 테이블을 vacuum하는 경우, vacuum 작업은 증분적 정렬 후 병합으로 구성되는 일련의 단계로 진행됩니다. 작업이 실패하거나 vacuum 도중 Amazon Redshift가 오프라인 상태가 되는 경우 부분적으로 vacuum된 테이블이나 데이터베이스는 일관된 상태가 되지만 vacuum 작업을 수동으로 다시 시작해야 합니다. 증분적 정렬은 손실되지만 실패 전에 커밋된 병합된 행들은 다시 vacuum할 필요가 없습니다. 정렬되지 않은 리전이 클 경우, 상당한 시간을 손해 볼 수 있습니다. 정렬 및 병합 단계에 대한 자세한 내용은 병합된 행의 볼륨 줄이기 섹션을 참조하세요.

테이블이 vacuum되는 동안 사용자는 테이블에 액세스할 수 있습니다. 테이블이 vacuum되는 중에도 쿼리 및 쓰기 작업을 수행할 수 있지만 DML과 vacuum이 동시에 실행되고 있을 때는 두 작업 모두 시간이 더 소요될 수 있습니다. vacuum 도중에 UPDATE 및 DELETE 문을 실행하는 경우 시스템 성능이 저하될 수 있습니다. 증분적 병합은 동시적인 UPDATE 작업과 DELETE 작업을 일시적으로 차단하며 UPDATE 작업과 DELETE 작업은 해당 테이블에서 증분적 병합 단계를 차단합니다. ALTER TABLE 같은 DDL 작업은 테이블의 vacuum 작업이 끝날 때까지 차단됩니다.

> **Note**
>
> VACUUM에 대한 다양한 수정자가 작동 방식을 제어합니다. 이를 사용하여 현재 요구 사항에 맞게 vacuum 작업을 조정할 수 있습니다. 예를 들어 VACUUM RECLUSTER를 사용하면 전체 병합 작업을 수행하지 않아 vacuum 작업이 단축됩니다. 자세한 내용은 VACUUM 단원을 참조하십시오.

vacuum 임계값

기본적으로 VACUUM은 테이블 행의 95% 이상이 이미 정렬된 테이블에 대해서는 정렬 단계를 건너뜁니다. 정렬 단계를 건너뛰면 VACUUM 성능을 상당히 개선할 수 있습니다. 단일 테이블의 기본 정렬 임계값을 변경하려면 VACUUM 명령을 실행할 때 테이블 이름과 TO threshold PERCENT 파라미터를 포함시킵니다.

vacuum 유형

다양한 vacuum 유형에 대한 자세한 내용은 VACUUM 섹션을 참조하세요.

Vacuum 시간 최소화

Amazon Redshift는 백그라운드에서 자동으로 데이터를 정렬하고 VACUUM DELETE를 실행합니다. 이렇게 하면 VACUUM 명령을 실행할 필요성이 줄어듭니다. Vacuum은 시간이 많이 걸릴 수 있는 프로세스입니다. 데이터의 성격에 따라 다르지만, Vacuum 시간을 최소화하려면 다음과 같은 방식을 따르는 것이 좋습니다.

주제

- reindex 실행 여부 결정
- 정렬되지 않은 리전 크기 줄이기
- 병합된 행의 볼륨 줄이기
- 정렬 키 순서로 데이터 로드
- 시계열 테이블을 사용하여 저장된 데이터 줄이기

reindex 실행 여부 결정

인터리브 정렬 스타일을 사용하면 종종 쿼리 성능을 크게 높일 수 있지만 정렬 키 열의 값 분산이 변경되면 시간이 지남에 따라 성능이 저하될 수 있습니다.

COPY 또는 CREATE TABLE AS를 사용하여 비어 있는 인터리브 테이블에 처음 로드할 때는 Amazon Redshift가 인터리브 인덱스를 자동으로 구축합니다. INSERT를 사용하여 인터리브 테이블에 처음 로드하는 경우에는 이후 VACUUM REINDEX를 실행하여 인터리브 인덱스를 초기화해야 합니다.

시간이 지나면서 새로운 정렬 키 값이 포함된 행을 추가하여 정렬 키 열의 값 분산이 바뀌면 성능이 떨어질 수 있습니다. 하지만 새로운 행이 기존 정렬 키 값의 범위만 벗어나지 않는다면 인덱스를 다시 빌드할 필요가 없습니다. VACUUM SORT ONLY 또는 VACUUM FULL을 실행하여 정렬 순서를 복원하십시오.

쿼리 엔진은 정렬 순서를 사용하여 쿼리 처리를 위해 스캔해야 하는 데이터 블록을 효율적으로 선택할 수 있습니다. 인터리브 정렬의 경우 Amazon Redshift는 정렬 키 열 값을 분석하여 최적의 정렬 순서를 결정합니다. 행이 추가되어 키 값의 분산이 변경되거나 스큐되면 정렬 전략이 더 이상 최적이 아니게 되며, 정렬의 성능상 이득이 저하됩니다. 정렬 키 분산을 다시 분석하기 위해 VACUUM REINDEX를 실행할 수 있습니다. reindex 작업은 시간이 많이 걸리므로 reindex가 테이블에 이득이 될지 결정하려면 SVV_INTERLEAVED_COLUMNS 뷰를 쿼리하십시오.

예를 들어 다음 쿼리는 인터리브 정렬 키를 사용하는 테이블의 세부 정보를 보여 줍니다.

```
select tbl as tbl_id, stv_tbl_perm.name as table_name,
col, interleaved_skew, last_reindex
from svv_interleaved_columns, stv_tbl_perm
where svv_interleaved_columns.tbl = stv_tbl_perm.id
and interleaved_skew is not null;


 tbl_id | table_name | col | interleaved_skew | last_reindex
--------+------------+-----+------------------+---------------------
 100048 | customer   |   0 |             3.65 | 2015-04-22 22:05:45
 100068 | lineorder  |   1 |             2.65 | 2015-04-22 22:05:45
 100072 | part       |   0 |             1.65 | 2015-04-22 22:05:45
 100077 | supplier   |   1 |             1.00 | 2015-04-22 22:05:45
(4 rows)
```

interleaved_skew의 값은 스큐의 양을 나타내는 비율입니다. 값이 1이면 적용된 제한이 없는 것입니다. 스큐가 1.4보다 크면 일반적으로 스큐가 기본 집합에 내재하지 않는 한 VACUUM REINDEX로 성능이 개선됩니다.

last_reindex의 날짜 값을 사용하면 마지막 reindex 후 얼마나 지났는지 확인할 수 있습니다.

정렬되지 않은 리전 크기 줄이기

이미 데이터가 포함되어 있는 테이블에 대량의 새 데이터를 로드하거나 일상적 유지 관리 작업의 일환으로 테이블을 vacuum하지 않으면 정렬되지 않은 리전이 늘어납니다. vacuum 작업이 오래 실행되는 것을 방지하려면 다음과 같은 습관을 들이십시오.

- 정기적인 일정에 따라 vacuum 작업을 실행합니다.

  테이블을 조금씩 증분하여 로드하는 경우(테이블의 전체 행 수의 작은 비율에 해당하는 일일 업데이트 등), 정기적으로 VACUUM을 실행하면 개별 vacuum 작업이 빠르게 진행되는 데 도움이 됩니다.
- 가장 큰 로드를 먼저 실행합니다.

  COPY 작업을 여러 번 사용하여 새 테이블을 로드해야 하는 경우, 가장 큰 로드를 먼저 실행합니다. 새 테이블이나 잘린 테이블에 첫 로드를 실행할 때는 정렬된 리전으로 모든 데이터가 직접 로드되므로 vacuum이 필요하지 않습니다.
- 모든 행을 삭제하는 대신 테이블을 자릅니다.

  테이블에 행을 삭제해도 vacuum 작업을 수행할 때까지는 행이 차지하던 공간이 회수되지 않습니다. 그러나 테이블을 자르면 테이블이 비워 지고 디스크 공간이 회수되므로 vacuum이 필요하지 않습니다. 또는 테이블을 삭제하고 다시 만드십시오.
- 테스트 테이블을 자르거나 삭제합니다.

  테스트 목적으로 소수의 행을 테이블에 로드하는 경우, 로드가 끝난 후 행을 삭제하지 마십시오. 그 대신 테이블을 자르고 후속 프로덕션 로드 작업의 일환으로 이 행들을 다시 로드합니다.
- 전체 복사를 수행합니다.

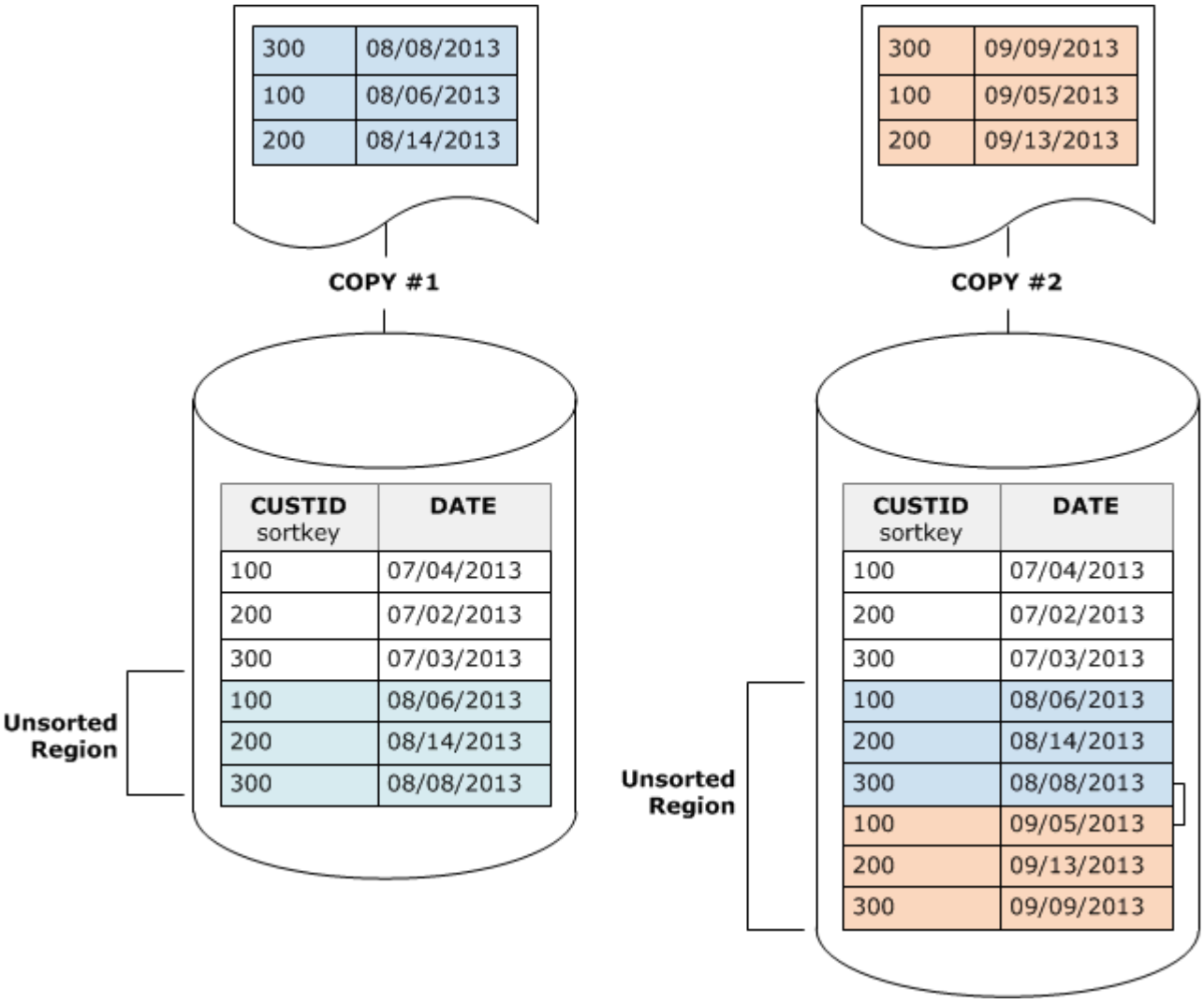
  복합 정렬 키를 사용하는 테이블에 정렬되지 않은 큰 리전이 있는 경우, 전체 복사가 vacuum보다 훨씬 빠릅니다. 전체 복사는 대량 삽입을 사용하여 테이블을 다시 생성하고 다시 채움으로써 테이블을 자동으로 다시 정렬합니다. 테이블에 정렬되지 않은 큰 리전이 있는 경우, 전체 복사가 vacuum보다 훨씬 빠릅니다. 반면 전체 복사 작업 중에는 vacuum 중에는 가능한 동시 업데이트를 할 수 없다는 단점이 있습니다. 자세한 내용은 Amazon Redshift 쿼리 설계 모범 사례 단원을 참조하십시오.

병합된 행의 볼륨 줄이기

vacuum 작업이 새 행을 테이블의 정렬된 리전에 병합해야 하는 경우, 테이블이 커질수록 vacuum에 필요한 시간이 늘어납니다. 병합해야 하는 행의 수를 줄이면 vacuum 성능을 높일 수 있습니다.

vacuum 전에 테이블은 테이블 헤드에 있는 정렬된 리전과 그 뒤에 오는, 행이 추가되거나 업데이트될 때마다 증가하는 정렬되지 않은 리전으로 구성됩니다. COPY 작업에 의해 행 세트가 추가되면 새로운 행 세트는 테이블 끝에 있는 정렬되지 않은 리전에 추가될 때 정렬 키에서 정렬됩니다. 새 행들은 자체 세트 안에서는 정렬되어 있지만 정렬되지 않은 리전 안에서는 그렇지 않습니다.

다음 다이어그램은 2회 연속 COPY 작업 후의 정렬되지 않은 리전을 보여 주며, 여기서 정렬 키는 CUSTID입니다. 이 예에는 간단히 하기 위해 복합 정렬 키가 나와 있지만 정렬되지 않은 리전의 효과가 더 크다는 점만 제외하면 인터리브 정렬 키에도 동일한 원리가 적용됩니다.



vacuum은 2단계로 테이블의 정렬 순서를 복원합니다.

1. 정렬되지 않은 리전을 새로 정렬된 리전으로 정렬합니다.

   첫 번째 단계는 비교적 리소스를 적게 사용하는데 정렬되지 않은 리전만 다시 작성되기 때문입니다. 새로 정렬된 리전의 정렬 키 값의 범위가 기존 범위보다 높다면 새 행들만 다시 작성하면 되며,

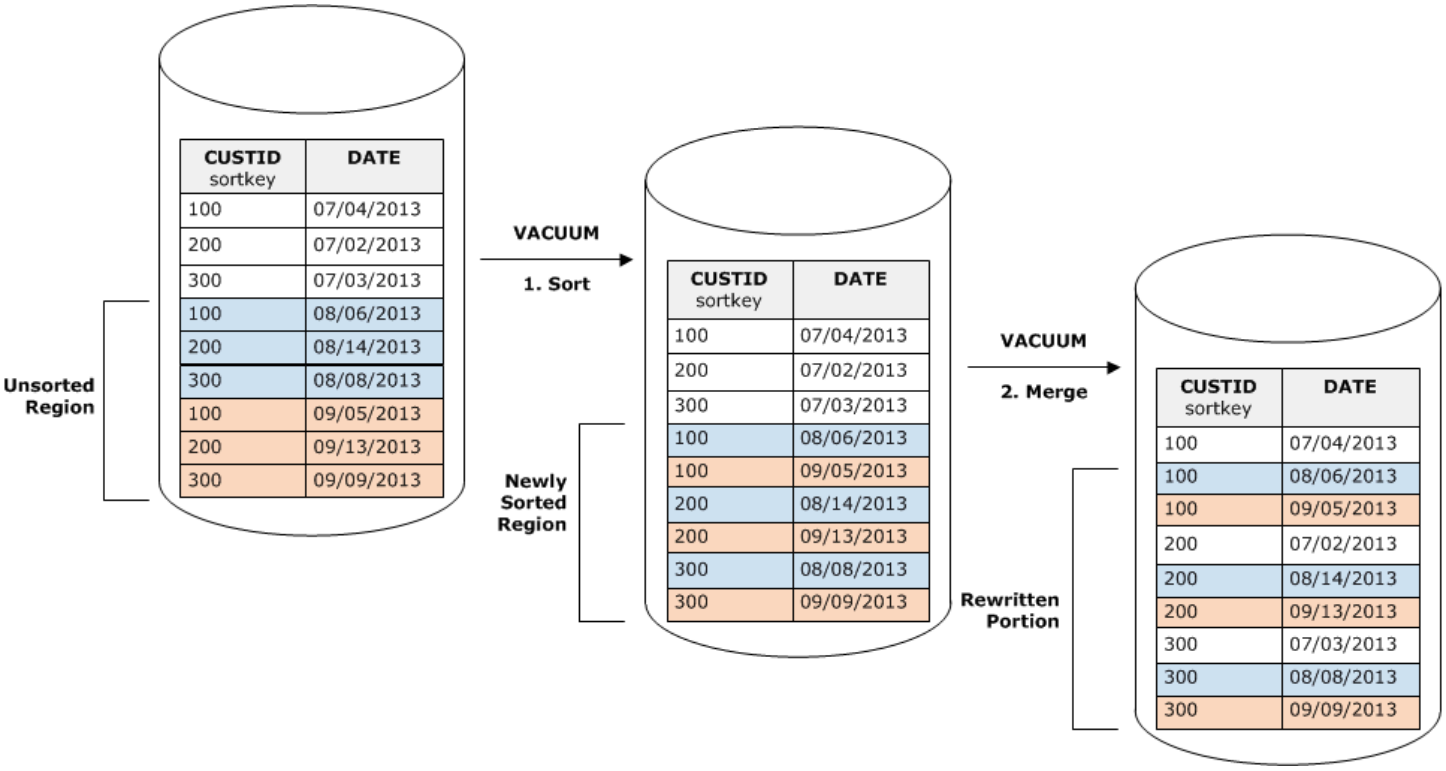
vacuum이 완료됩니다. 예를 들어 정렬된 리전에 ID 값 1~500이 포함되어 있고 후속 복사 작업이 500보다 큰 키 값을 추가하는 경우, 정렬되지 않은 리전을 재작성하면 됩니다.

2. 새로 정렬된 리전과 이전에 정렬된 리전을 병합합니다.

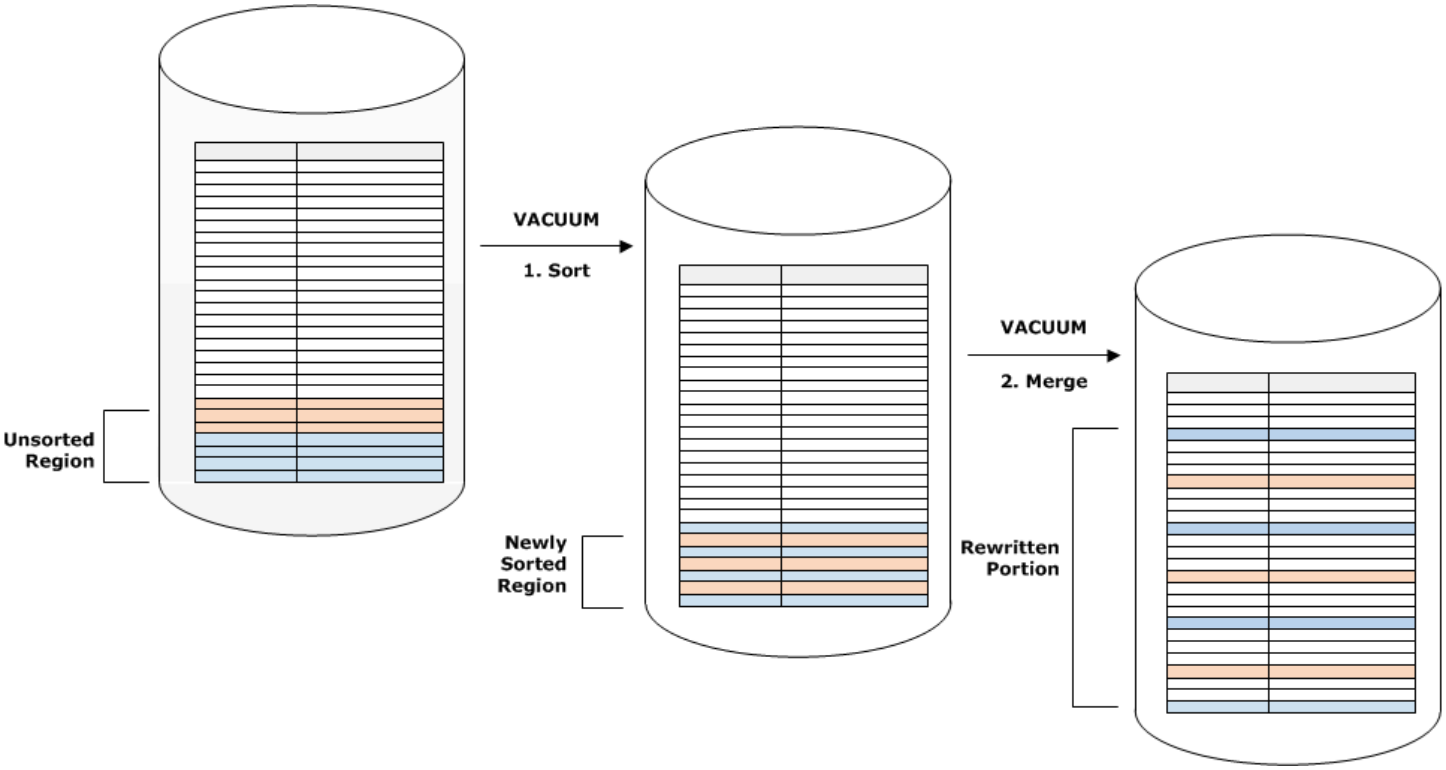
   새로 정렬된 리전의 키가 정렬된 리전의 키와 중첩되는 경우, VACUUM은 행을 병합해야 합니다. vacuum은 새로 정렬된 리전의 시작 부분(가장 낮은 정렬 키)에서 시작해 이전에 정렬된 리전과 새로 정렬된 리전의 병합된 행들을 새 블록 세트에 작성합니다.

새 정렬 키 범위와 기존 정렬 키의 중첩 정도는 이전에 정렬된 리전을 어느 정도까지 다시 작성해야 하는지 결정합니다. 정렬되지 않은 키가 기존 정렬 범위 전체에 흩어져 있는 경우, vacuum은 테이블의 기존 부분을 다시 작성해야 할 수 있습니다.

다음 다이어그램은 CUSTID가 정렬 키인 테이블에 추가되는 행들을 vacuum이 어떻게 정렬하고 병합하는지 보여 줍니다. 각각의 복사 작업이 기존 키와 중첩되는 키 값을 가진 새로운 행 세트를 추가하기 때문에 거의 테이블 전체를 다시 작성해야 합니다. 이 다이어그램에는 단일 정렬 및 병합이 나와 있지만 실제로 큰 vacuum은 일련의 증분적 정렬 및 병합 단계로 구성됩니다.



새로운 행 세트의 정렬 키 범위가 기존 키의 범위와 중첩되는 경우, 병합 단계 비용은 테이블이 커지는 데 따라 테이블 크기에 비례하여 계속 증가하는 반면 정렬 단계의 비용은 정렬되지 않은 리전의 크기에 여전히 비례합니다. 이런 경우, 다음 다이어그램에서 보듯 병합 단계의 비용은 정렬 단계의 비용과의 비교가 무색합니다.



테이블 중에서 다시 병합된 비율을 확인하려면 vacuum 작업이 완료된 후 SVV_VACUUM_SUMMARY를 쿼리합니다. 다음 쿼리는 시간이 흐르면서 CUSTSALES가 계속 커짐에 따른 6회 연속 vacuum의 효과를 보여 줍니다.

```
select * from svv_vacuum_summary
where table_name = 'custsales';


 table_name | xid  | sort_      | merge_     | elapsed_   | row_  | sortedrow_ | block_
  | max_merge_
            |      | partitions | increments | time       | delta | delta      | delta
  | partitions
 -----------+------+------------+------------+------------+-------+------------
+---------+---------------
  custsales | 7072 |          3 |          2 |  143918314 |     0 |   88297472 |   1524
  |      47
  custsales | 7122 |          3 |          3 |  164157882 |     0 |   88297472 |    772
  |      47
  custsales | 7212 |          3 |          4 |  187433171 |     0 |   88297472 |    767
  |      47
  custsales | 7289 |          3 |          4 |  255482945 |     0 |   88297472 |    770
  |      47
```

```
 custsales | 7420 |          3 |          5 |  316583833 |     0 |   88297472 |    769
 |      47
 custsales | 9007 |          3 |          6 |  306685472 |     0 |   88297472 |    772
 |      47
(6 rows)
```

merge_increments 열은 각각의 vacuum 작업을 위해 병합된 데이터 양을 표시합니다. 연속적 vacuum 동안 병합 증분의 수가 테이블 크기 증가에 비례하여 증가하는 경우, 이는 기존의 정렬된 리전과 새로 정렬된 리전이 중첩되기 때문에 각각의 vacuum 작업이 다시 병합하는 이 테이블의 행이 증가하고 있다는 뜻입니다.

정렬 키 순서로 데이터 로드

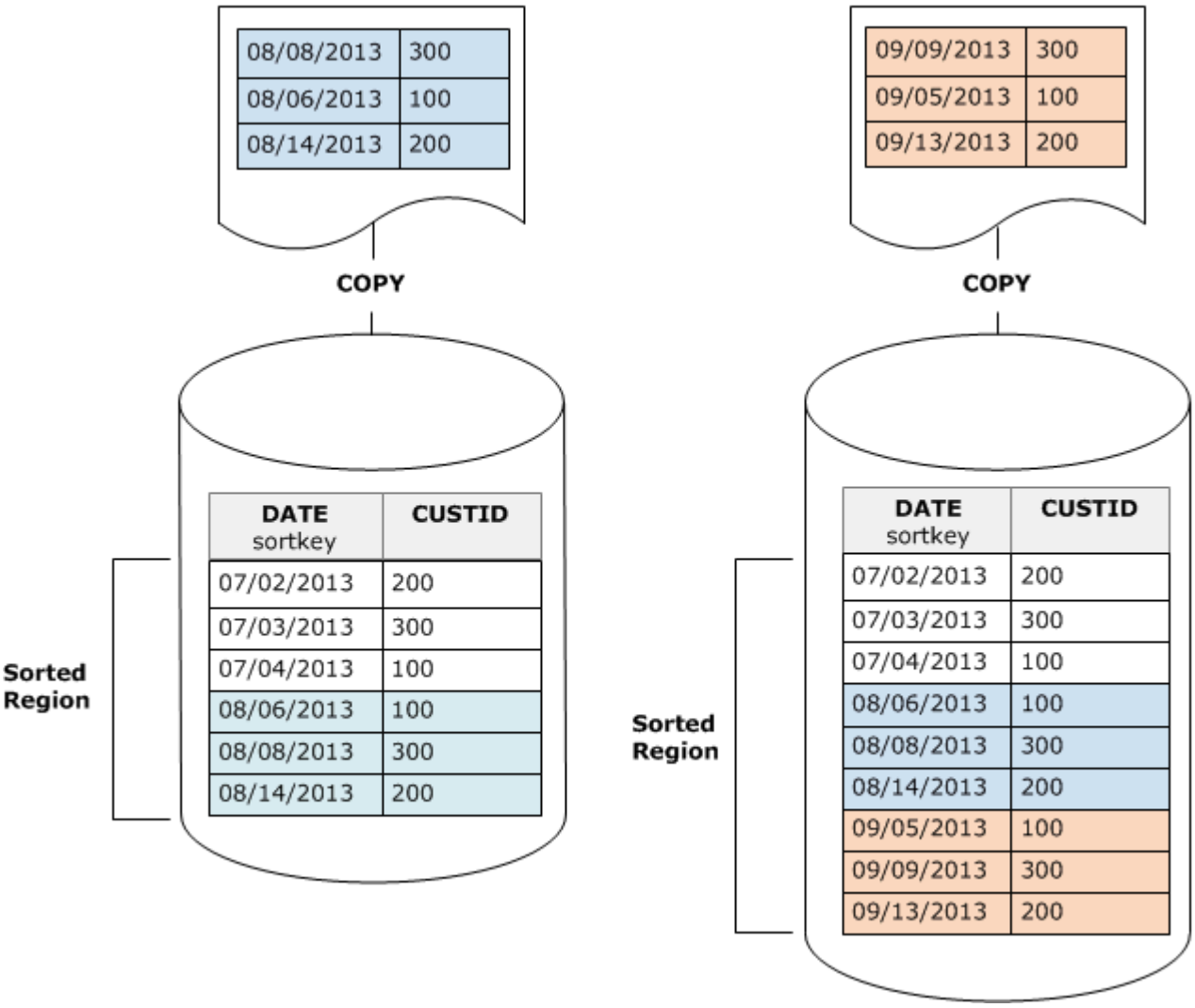
COPY 명령을 사용하여 데이터를 정렬 키 순서로 로드하면 vacuum 필요성을 줄이거나 심지어 없앨 수도 있습니다.

다음이 모두 참인 경우, COPY는 새 행을 테이블의 정렬된 리전에 자동으로 추가합니다.

- 테이블이 하나의 정렬 열에만 복합 정렬 키를 사용합니다.
- 정렬 열은 NOT NULL입니다.
- 테이블이 100% 정렬되어 있거나 비어 있습니다.
- 모든 새 행은 삭제 대기 상태인 행을 포함하여 기존 행보다 정렬 순서가 더 높습니다. 이 인스턴스에서 Amazon Redshift는 정렬 키의 첫 8바이트를 사용하여 정렬 순서를 결정합니다.

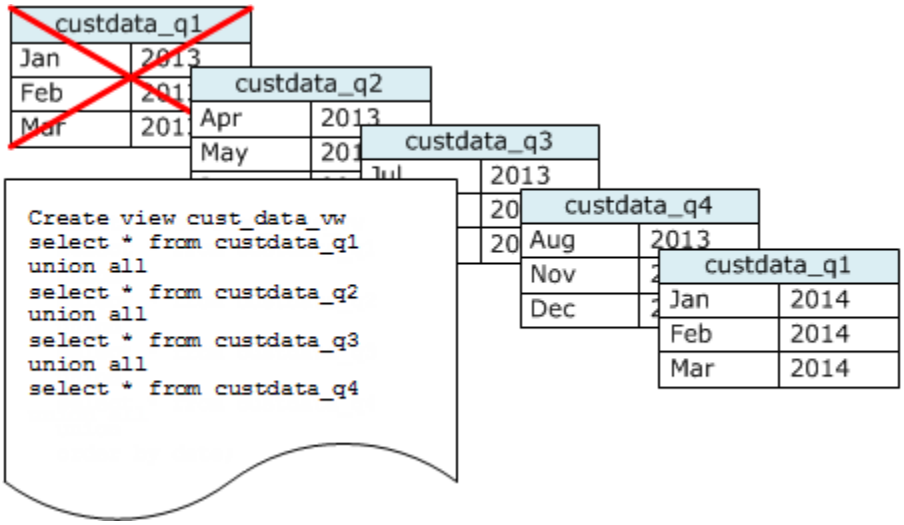
예를 들어 고객 ID와 시간을 사용하여 고객 이벤트를 기록하는 테이블이 있다고 가정해 보십시오. 고객 ID에서 정렬하는 경우, 앞의 예에서 보듯 증분적 로드에 의해 추가된 새 행의 정렬 키 범위가 기존 범위와 중첩되어 vacuum 작업이 리소스를 많이 사용하게 됩니다.

정렬 키를 타임스탬프 열로 설정하면 다음 다이어그램에 나온 것처럼 새 행들이 테이블 끝에서 정렬 순서로 추가되어 vacuum 필요성이 줄어들거나 심지어 없어집니다.



시계열 테이블을 사용하여 저장된 데이터 줄이기

롤링 기간 동안 데이터를 유지하는 경우, 다음 다이어그램에 나온 것처럼 시계열 테이블을 사용하십시오.



데이터 세트를 추가할 때마다 새 테이블을 만든 다음 시계열에서 가장 오래된 테이블을 삭제합니다. 두 가지 장점이 있습니다.

- DROP TABLE 작업이 대량 DELETE보다 훨씬 효율적이기 때문에 행 삭제에 따른 비용 추가를 피할 수 있습니다.
- 테이블이 타임스탬프를 기준으로 정렬되면 vacuum이 필요 없습니다. 각 테이블에 한 달 동안의 데이터가 포함된 경우, 테이블이 타임스탬프를 기준으로 정렬되어 있지 않더라도 vacuum이 다시 써야 하는 데이터 양은 한 달치에 불과합니다.

데이터가 여러 테이블에 저장되어 있다는 사실을 감추는, 보고 쿼리가 사용할 UNION ALL 뷰를 생성할 수 있습니다. 쿼리가 정렬 키에서 필터를 적용하는 경우, 쿼리 플래너는 사용되지 않는 모든 테이블을 효율적으로 건너뜁니다. 다른 유형의 쿼리에는 UNION ALL이 비효율적일 수 있으므로 테이블을 사용하는 모든 쿼리의 컨텍스트에서 쿼리 성능을 평가해야 합니다.

동시 쓰기 작업 관리

Amazon Redshift는 테이블이 증분적으로 로드되거나 수정되는 동안 테이블을 읽을 수 있도록 허용합니다.

일부 전통적인 데이터 웨어하우징 및 비즈니스 인텔리전스 애플리케이션에서는 야간 로드가 완료될 때에만 사용자들이 데이터베이스를 사용할 수 있습니다. 이런 경우, 분석 쿼리가 실행되고 보고서가 생성되는 정상 업무 시간 중에는 업데이트가 허용되지 않습니다. 다만 하루 중 오랫동안 또는 하루 종일 라이브 상태로 유지되는 애플리케이션의 수가 늘어나면서 로드 기간이라는 개념은 낡은 것이 되고 있습니다.

Amazon Redshift는 테이블이 증분적으로 로드되거나 수정되는 동안 테이블을 읽을 수 있도록 함으로써 이런 유형의 애플리케이션을 지원합니다. 쿼리는 다음 버전이 커밋되기를 기다리는 것이 아니라 가장 최근에 커밋된 데이터 버전 또는 스냅샷을 단순히 확인합니다. 특정 쿼리가 다른 쓰기 작업으로부터의 커밋을 기다리도록 하려면 그에 따라 일정을 세워야 합니다.

다음 주제에서는 트랜잭션, 데이터베이스 스냅샷, 업데이트, 동시 동작과 관련하여 몇 가지 주요 개념과 사용 사례를 설명합니다.

주제

직렬화 가능 격리

일부 애플리케이션은 동시 쿼리 및 로드뿐 아니라 여러 테이블에 또는 같은 테이블 동시에 쓸 수 있는 능력이 필요합니다. 이 맥락에서 동시에는 정확히 같은 시간에 실행되도록 예약된 것을 말하는 것이 아니라 중첩된다는 뜻입니다. 첫 번째 트랜잭션이 커밋되기 전에 두 번째 트랜잭션이 시작된다면 두 트랜잭션은 동시적이라고 간주됩니다. 동시 작업은 같은 사용자 또는 서로 다른 사용자에 의해 제어되는 서로 다른 세션에서 비롯될 수 있습니다.

> **Note**
>
> Amazon Redshift는 별도로 실행되는 각각의 SQL 명령이 개별적으로 커밋되는 기본 자동 커밋 동작을 지원합니다. 명령 세트를 트랜잭션 블록(BEGIN 및 END 문에 의해 정의됨)으로 묶는 경우, 이 블록은 하나의 트랜잭션으로 커밋되므로 필요할 경우, 롤백할 수 있습니다. 이 동작에 대한 예외가 TRUNCATE 및 VACUUM 명령으로서 현재 트랜잭션의 대기 중인 모든 변경을 자동으로 커밋합니다.
>
> 일부 SQL 클라이언트는 BEGIN 및 COMMIT 명령을 자동으로 실행하므로 클라이언트는 명령문 그룹이 트랜잭션으로 실행되는지 각 개별 명령문이 자체 트랜잭션으로 실행되는지 여부를 제어합니다. 사용 중인 인터페이스의 설명서를 확인하십시오. 예를 들어 Amazon Redshift JDBC 드라이버를 사용할 경우 세미콜론으로 구분된 여러 SQL 명령이 포함된 쿼리 문자열이 있는 JDBC `PreparedStatement`는 모든 명령문을 단일 트랜잭션으로 실행합니다. 반대로 SQL Workbench/J를 사용하고 AUTO COMMIT ON을 설정한 경우 여러 명령문을 실행하면 각 명령문이 자체 트랜잭션으로 실행됩니다.

Amazon Redshift에서 동시 쓰기 작업은 테이블에서의 쓰기 잠금 및 직렬화 가능 격리 원칙을 사용하여 보호적 방식으로 지원됩니다. 직렬화 가능 격리는 테이블에 대해 실행 중인 트랜잭션이 해당 테이블에 대해 실행 중인 유일한 트랜잭션이라는 환상을 보존합니다. 예를 들어 동시에 실행 중인 T1과 T2라는 2개의 트랜잭션은 적어도 다음 중 하나와 동일한 결과를 생성해야 합니다.

- T1과 T2가 이 순서로 순차 실행됩니다.
- T2와 T1이 이 순서로 순차 실행됩니다.

동시 트랜잭션은 서로에게 보이지 않으며, 서로의 변경을 감지할 수 없습니다. 각각의 동시 트랜잭션은 트랜잭션 시작 시 데이터베이스의 스냅샷을 생성합니다. 데이터베이스 스냅샷은 대부분의 SELECT 문과 COPY, DELETE, INSERT, UPDATE, TRUNCATE 같은 DML 명령 및 다음의 DDL 명령이 처음 발생할 때 트랜잭션 내에서 생성됩니다.

- ALTER TABLE(열 추가 또는 삭제)
- CREATE TABLE
- DROP TABLE
- TRUNCATE TABLE

동시 트랜잭션의 순차 실행 중 동시 실행과 동일한 결과를 생성하는 실행이 있는 경우 이러한 트랜잭션은 "직렬화 가능"한 것으로 간주되며 안전하게 실행할 수 있습니다. 동일한 결과를 생성할 수 있는 이러한 트랜잭션의 순차 실행이 없는 경우 직렬화 기능을 중단하는 문을 실행하는 트랜잭션은 중지되고 롤백됩니다.

시스템 카탈로그 테이블(PG)과 기타 Amazon Redshift 시스템 테이블(STL 및 STV)은 트랜잭션에서 잠기지 않습니다. 따라서 DDL 및 TRUNCATE 작업에서 발생하는 데이터베이스 객체의 변경 사항은 동시 트랜잭션에 대한 커밋 시 볼 수 있습니다.

예를 들어 2개의 동시 트랜잭션 T1과 T2가 시작될 때 데이터베이스에 테이블 A가 존재한다고 가정해 보십시오. T2가 PG_TABLES 카탈로그 테이블에서 선택하여 테이블 목록을 반환한다고 가정합니다. 그런 다음 T1이 테이블 A를 삭제하고 커밋한 다음 T2가 테이블을 다시 나열합니다. 이제 테이블 A가 더 이상 나열되지 않습니다. T2가 삭제된 테이블을 쿼리하려고 하는 경우 Amazon Redshift는 "관계가 존재하지 않음(relation does not exist)" 오류를 반환합니다. 테이블 목록을 T2에 반환하거나 테이블 A가 존재하는지 확인하는 카탈로그 쿼리에는 사용자 테이블에서 수행되는 작업과 동일한 격리 규칙이 적용되지 않습니다.

이런 테이블의 업데이트를 위한 트랜잭션은 읽기 커밋된 격리 모드에서 실행됩니다. PG-prefix 카탈로그 테이블은 스냅샷 격리를 지원하지 않습니다.

시스템 테이블과 카탈로그 테이블의 직렬화 가능 격리

데이터베이스 스냅샷은 사용자가 생성한 테이블 또는 Amazon Redshift 시스템 테이블(STL 또는 STV)을 참조하는 SELECT 쿼리의 트랜잭션에서도 생성됩니다. 테이블을 참조하지 않는 SELECT 쿼리는 새 트랜잭션 데이터베이스 스냅샷을 생성하지 않습니다. 시스템 카탈로그 테이블(PG)에서만 작동하는 INSERT, DELETE 및 UPDATE 문도 새 트랜잭션 데이터베이스 스냅샷을 생성하지 않습니다.

직렬화 가능 격리 오류 수정 방법

ERROR:1023 DETAIL: Redshift의 테이블에서 직렬화 가능한 격리 위반

Amazon Redshift에서 직렬화 가능 격리 오류를 감지하면 다음과 같은 오류 메시지가 표시됩니다.

```
ERROR:1023 DETAIL: Serializable isolation violation on table in Redshift
```

직렬화 가능 격리 오류를 해결하기 위해 다음과 같은 방법을 시도할 수 있습니다.

- 취소된 트랜잭션을 재시도합니다.

  Amazon Redshift에서 동시 워크로드를 직렬화할 수 없음을 감지했습니다. 이는 일반적으로 오류가 발생한 트랜잭션을 재시도하여 해결할 수 있는 애플리케이션 논리의 차이를 나타냅니다. 문제가 지속되면 다른 방법 중 하나를 시도하십시오.
- 동일한 원자성 트랜잭션에 있을 필요가 없는 모든 작업을 트랜잭션 외부로 이동합니다.

  이 방법은 두 개의 트랜잭션 내에 있는 개별 작업이 다른 트랜잭션의 결과에 영향을 미칠 수 있는 방식으로 상호 참조하는 경우에 사용할 수 있습니다. 예를 들어 다음 두 세션은 각자 트랜잭션을 시작합니다.

  ```
  Session1_Redshift=# begin;
  ```

  ```
  Session2_Redshift=# begin;
  ```

  각 트랜잭션에 있는 SELECT 문의 결과는 다른 트랜잭션에 있는 INSERT 문에 영향을 받을 수 있습니다. 다시 말해 다음과 같은 문을 어떤 순서로든 직렬 방식으로 실행한다고 가정합시다. 어떤 경우에도 트랜잭션이 동시에 실행된 경우보다 한 행을 더 반환하는 SELECT 문 중 하나를 얻게 됩니다. 작업을 직렬로 실행하여 동시에 실행할 때와 동일한 결과를 산출할 수 있는 순서는 없습니다. 따라서 마지막으로 실행되는 작업으로 인해 직렬화 가능 격리 오류가 발생합니다.

  ```
  Session1_Redshift=# select * from tab1;
  ```

```
Session1_Redshift=# insert into tab2 values (1);
```

```
Session2_Redshift=# insert into tab1 values (1);
Session2_Redshift=# select * from tab2;
```

많은 경우 SELECT 문의 결과는 중요하지 않습니다. 다시 말해 트랜잭션 내 작업의 원자성은 중요하지 않습니다. 이러한 경우에는 다음 예와 같이 SELECT 문을 트랜잭션 외부로 이동합니다.

```
Session1_Redshift=# begin;
Session1_Redshift=# insert into tab1 values (1)
Session1_Redshift=# end;
Session1_Redshift=# select * from tab2;
```

```
Session2_Redshift # select * from tab1;
Session2_Redshift=# begin;
Session2_Redshift=# insert into tab2 values (1)
Session2_Redshift=# end;
```

이 예의 경우 트랜잭션 내 교차 참조는 없습니다. 두 가지 INSERT 문은 서로에게 영향을 미치지 않습니다. 이 예의 경우 트랜잭션이 직렬로 실행되어 동시에 실행될 때와 동일한 결과를 산출하는 순서가 최소 한 개 있습니다. 이는 트랜잭션을 직렬화할 수 있음을 뜻합니다.

- 각 세션에서 모든 테이블을 잠가 직렬화를 강제 실행하십시오.

  LOCK 명령은 직렬화 가능 격리 오류를 발생시킬 수 있는 작업을 차단합니다. LOCK 명령 사용 시 다음을 반드시 수행하십시오.

  - 트랜잭션 내 읽기 전용 SELECT 문의 영향을 받는 테이블을 포함해 트랜잭션의 영향을 받는 모든 테이블을 잠급니다.
  - 작업이 수행되는 순서에 상관없이 테이블을 동일한 순서로 잠급니다.
  - 작업을 수행하기 전에 트랜잭션을 시작하는 시점에 모든 테이블을 잠급니다.
- 동시 트랜잭션에는 스냅샷 격리를 사용하세요.

  ALTER DATABASE 명령을 스냅샷 격리와 함께 사용하십시오. ALTER DATABASE의 SNAPSHOT 파라미터에 대한 자세한 내용은 파라미터 섹션을 참조하세요.

ERROR:1018 DETAIL: 관계가 존재하지 않음

여러 세션에서 동시 Amazon Redshift 작업을 실행하면 다음과 같은 오류 메시지가 표시됩니다.

```
ERROR: 1018 DETAIL: Relation does not exist.
```

Amazon Redshift의 트랜잭션은 스냅샷 격리를 따릅니다. 트랜잭션이 시작된 후 Amazon Redshift는 데이터베이스의 스냅샷을 만듭니다. 트랜잭션의 전체 수명 주기 동안 트랜잭션은 스냅샷에 반영된 데이터베이스 상태에서 작동합니다. 트랜잭션이 스냅샷에 존재하지 않는 테이블에서 읽는 경우 이전에 표시된 1018 오류 메시지가 나타납니다. 트랜잭션이 스냅샷을 만든 후 다른 동시 트랜잭션이 테이블을 생성하더라도 트랜잭션은 새로 생성된 테이블에서 읽을 수 없습니다.

이 직렬화 격리 오류를 해결하기 위해 테이블이 존재함을 알고 있는 시점으로 트랜잭션 시작을 이동하려고 할 수 있습니다.

테이블이 다른 트랜잭션에 의해 생성된 경우 이 시점은 적어도 해당 트랜잭션이 커밋된 이후입니다. 또한 테이블을 삭제했을 수 있는 동시 트랜잭션이 커밋되지 않았는지 확인합니다.

```
session1 = # BEGIN;
session1 = # DROP TABLE A;
session1 = # COMMIT;
```

```
session2 = # BEGIN;
```

```
session3 = # BEGIN;
session3 = # CREATE TABLE A (id INT);
session3 = # COMMIT;
```

```
session2 = # SELECT * FROM A;
```

session2에 의해 읽기 작업으로 실행되는 마지막 작업이 직렬화 가능한 격리 오류를 발생시킵니다. 이 오류는 session2가 스냅샷을 만들고 테이블이 커밋된 session1에 의해 이미 삭제된 경우 발생합니다. 즉, 동시 session3이 테이블을 생성하더라도 session2는 테이블이 스냅샷에 없기 때문에 테이블을 볼 수 없습니다.

이 오류를 해결하기 위해 다음과 같이 세션을 재정렬할 수 있습니다.

```
session1 = # BEGIN;
session1 = # DROP TABLE A;
session1 = # COMMIT;
```

```
session3 = # BEGIN;
session3 = # CREATE TABLE A (id INT);
```

```
session3 = # COMMIT;
```

```
session2 = # BEGIN;
session2 = # SELECT * FROM A;
```

이제 session2가 스냅샷을 만들 때 session3은 이미 커밋되었으며 테이블은 데이터베이스에 있습니다. Session2는 오류 없이 테이블에서 읽을 수 있습니다.

쓰기 및 읽기/쓰기 작업

다양한 형식의 명령을 언제 어떻게 실행할지 결정하면 동시 쓰기 및 읽기-쓰기 작업의 특정 동작을 관리할 수 있습니다. 이 논의와 관련된 명령은 다음과 같습니다.

- COPY 명령 - 로드를 수행(초기 또는 증분적 로드)
- INSERT 명령 - 하나 이상의 행을 한 번에 추가
- UPDATE 명령 - 기존 행을 수정
- DELETE 명령 - 행을 제거

COPY 및 INSERT 작업은 순수한 쓰기 작업이지만 DELETE 및 UPDATE 작업은 읽기/쓰기 작업입니다. (행을 삭제 또는 업데이트하려면 먼저 행을 읽어야 합니다.) 동시 쓰기 작업의 결과는 동시에 실행 중인 특정 명령에 따라 달라집니다. 동일 테이블에 대한 COPY 및 INSERT 작업은 잠금이 풀릴 때까지 대기 상태로 보류되었다가 정상적으로 진행됩니다.

UPDATE 작업과 DELETE 작업은 쓰기를 수행하기 전에 초기 테이블 읽기에 의존하므로 서로 다르게 동작합니다. 동시 트랜잭션은 서로에게 보이지 않으므로 UPDATE와 DELETE 모두 마지막 커밋으로부터 데이터 스냅샷을 읽어야 합니다. 첫 번째 UPDATE 또는 DELETE가 잠금을 풀면 두 번째 UPDATE 또는 DELETE는 작업할 데이터가 잠재적으로 부실한지 여부를 확인해야 합니다. 데이터는 부실해지지 않는데, 첫 번째 트랜잭션이 잠금을 풀 때까지 두 번째 트랜잭션은 데이터의 스냅샷을 가져오지 않기 때문입니다.

동시 쓰기 트랜잭션의 잠재적 교착 상황

트랜잭션에 둘 이상의 테이블의 업데이트가 포함되는 경우, 동시에 실행되는 두 트랜잭션이 같은 테이블 세트에 쓰기를 시도하다 교착될 가능성이 늘 존재합니다. 트랜잭션은 커밋하거나 롤백할 때 테이블 잠금을 한 번에 하나씩 풀지 않고 모든 잠금을 한 번에 풉니다.

예를 들어 트랜잭션 T1과 T2가 대략 같은 시간에 시작된다고 가정해 보십시오. T1이 테이블 A에 쓰기를 시작하고 T2가 테이블 B에 쓰기를 시작하는 경우, 두 트랜잭션은 충돌 없이 진행될 수 있습니다. 하

지만 T1이 테이블 A에 대한 쓰기를 마치고 테이블 B에 대한 쓰기를 시작해야 하는 경우에는 T2가 아직 테이블 B를 잠그고 있기 때문에 계속 진행할 수 없습니다. 반대로 T2가 테이블 B에 대한 쓰기를 마치고 테이블 A에 대한 쓰기를 시작해야 하는 경우, T1이 테이블 A를 잠그고 있기 때문에 계속 진행할 수 없습니다. 어느 트랜잭션도 쓰기 작업이 커밋될 때까지 잠금을 풀 수 없으므로 어느 트랜잭션도 진행될 수 없는 것입니다.

이런 종류의 교착을 피하기 위해서는 동시 쓰기 작업을 신중하게 예약해야 합니다. 예를 들어 트랜잭션에서 항상 같은 순서로 테이블을 업데이트해야 하며, 잠금을 지정하는 경우에는 DML 작업을 수행하기 전에 같은 순서로 테이블을 잠가야 합니다.

동시 쓰기 예

다음 유사 코드 예는 동시에 실행되는 트랜잭션이 어떻게 진행되는지 또는 대기하는지 보여 줍니다.

같은 테이블에 대한 동시 COPY 작업

트랜잭션 1은 LISTING 테이블에 행을 복사합니다.

```
begin;
copy listing from ...;
end;
```

트랜잭션 2는 별도의 세션에서 동시에 시작되며, LISTING 테이블에 더 많은 행을 복사하려 시도합니다. 트랜잭션 2는 트랜잭션 1이 LISTING 테이블에서 쓰기 잠금을 풀 때까지 기다려야 하며, 그런 다음에야 진행될 수 있습니다.

```
begin;
[waits]
copy listing from ;
end;
```

트랜잭션 하나 또는 둘 모두에 COPY 명령 대신 INSERT 명령이 포함된 경우, 똑같은 동작이 발생합니다.

같은 테이블로부터의 동시 DELETE 작업

트랜잭션 1은 테이블에서 행을 삭제합니다.

```
begin;
delete from listing where ...;
end;
```

트랜잭션 2는 동시에 시작되어 같은 테이블에서 행을 삭제하려 시도합니다. 트랜잭션 2는 트랜잭션 1이 완료되기를 기다렸다가 행 삭제를 시도하므로 성공합니다.

```
begin
[waits]
delete from listing where ;
end;
```

트랜잭션 하나 또는 둘 모두에 DELETE 명령 대신 같은 테이블에 대한 UPDATE 명령이 포함된 경우, 똑같은 동작이 발생합니다.

읽기 작업과 쓰기 작업이 혼합된 동시 트랜잭션

이 예에서 트랜잭션 1은 USERS 테이블에서 행을 삭제하고 이 테이블을 다시 로드하여 COUNT(*) 쿼리를 실행한 다음 ANALYZE하고 마지막으로 커밋합니다.

```
begin;
delete one row from USERS table;
copy ;
select count(*) from users;
analyze ;
end;
```

그러는 동안 트랜잭션 2가 시작됩니다. 이 트랜잭션은 추가적인 행을 USERS 테이블에 복사해 테이블을 분석한 다음 첫 번째 트랜잭션과 동일한 COUNT(*) 쿼리를 실행하려 시도합니다.

```
begin;
[waits]
copy users from ...;
select count(*) from users;
analyze;
end;
```

두 번째 트랜잭션은 첫 번째 트랜잭션이 완료될 때까지 기다려야 하므로 성공합니다. 이 트랜잭션의 COUNT 쿼리는 완료한 로드를 기준으로 카운트를 반환합니다.

튜토리얼: Amazon S3에서 데이터 로드

이 튜토리얼에서는 Amazon S3 버킷에 있는 데이터 파일에서 Amazon Redshift 데이터베이스 테이블로 데이터를 로드하는 프로세스를 처음부터 끝까지 살펴봅니다.

이 자습서에서는 다음을 수행합니다.

- 쉼표로 구분된 값(CSV) 형식, 문자로 구분된 형식, 고정 너비 형식을 사용하는 데이터 파일을 다운로드합니다.
- Amazon S3 버킷을 생성한 다음 이 버킷에 데이터 파일을 업로드합니다.
- Amazon Redshift 클러스터를 시작하고 데이터베이스 테이블을 생성합니다.
- COPY 명령을 사용하여 Amazon S3의 데이터 파일에서 테이블을 로드합니다.
- 로드 오류 문제를 해결하고 COPY 명령을 수정하여 오류를 수정합니다.

사전 조건

필요한 사전 조건은 다음과 같습니다.

- Amazon Redshift 클러스터를 시작하고 Amazon S3에 버킷을 생성하기 위한 AWS 계정.
- Amazon S3에서 테스트 데이터를 로드하기 위한 AWS 자격 증명(IAM 역할). 새로운 IAM 역할이 필요한 경우 IAM 역할 생성으로 이동합니다.
- Amazon Redshift 콘솔 쿼리 편집기와 같은 SQL 클라이언트.

이 자습서는 자습서만으로 학습이 가능하도록 만들어졌습니다. Amazon Redshift 데이터베이스의 설계 및 사용 방법을 더 깊이 이해하려면 이 튜토리얼 외에 다음 튜토리얼을 공부하는 것이 좋습니다.

- Amazon Redshift 시작 안내서에서는 Amazon Redshift 클러스터를 생성하고 샘플 데이터를 로드하는 프로세스를 안내합니다.

개요

INSERT 명령 또는 COPY 명령을 사용하여 Amazon Redshift 테이블에 데이터를 추가할 수 있습니다. Amazon Redshift 데이터 웨어하우스의 규모와 속도에서는 COPY 명령이 INSERT 명령보다 몇 배 더 빠르고 효율적입니다.

COPY 명령은 Amazon Redshift 대량 병렬 처리(MPP) 아키텍처를 사용하여 여러 데이터 원본에서 병렬로 데이터를 읽고 로드합니다. Amazon S3, Amazon EMR 또는 SSH(Secure Shell) 연결을 통해 액세스 가능한 원격 호스트에 있는 데이터 파일에서 로드할 수 있습니다. 또는 Amazon DynamoDB 테이블에서 직접 로드할 수 있습니다.

이 튜토리얼에서는 COPY 명령을 사용하여 Amazon S3에서 데이터를 로드합니다. 여기 설명된 여러 원리는 다른 데이터 원본에서의 로드에도 적용됩니다.

COPY 명령을 사용하는 방법에 대해 자세히 알아보려면 다음 리소스를 참조하십시오.

- 데이터 로드에 대한 Amazon Redshift 모범 사례
- Amazon EMR에서 데이터 로드
- 원격 호스트에서 데이터 로드
- Amazon DynamoDB 테이블에서 데이터 로드

1단계: 클러스터 생성

사용할 클러스터가 이미 있다면 이 단계를 건너뛰어도 됩니다.

이 자습서에서는 연습용으로 4노드 클러스터를 사용합니다.

클러스터 생성

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.

   탐색 메뉴에서 프로비저닝된 클러스터 대시보드를 선택합니다.

   > **Important**
   >
   > 사용자에게 클러스터 작업을 수행하는 데 필요한 권한이 있는지 확인해야 합니다. 필요한 권한을 부여하는 방법에 대한 자세한 내용은 Amazon Redshift의 AWS 서비스 액세스 권한 부여를 참조하십시오.

2. 오른쪽 상단에서 클러스터를 생성하려는 AWS 리전을 선택합니다. 본 튜토리얼의 목적에 맞게 [미국 서부(오레곤)(US West (Oregon))]를 선택합니다.
3. 탐색 메뉴에서 클러스터(Clusters)를 선택한 다음 클러스터 생성(Create cluster)을 선택합니다. 클러스터 생성 페이지가 표시됩니다.
4. 클러스터 생성(Create cluster) 페이지에서 클러스터의 파라미터를 입력합니다. 다음 값을 변경하는 것을 제외하고 파라미터에 대해 고유한 값을 선택합니다.

   - 노드 유형으로 **`dc2.large`**를 선택합니다.
   - 노드 수(Number of nodes)로 **`4`**를 선택합니다.

- Cluster permissions(클러스터 권한) 섹션의 Available IAM roles(사용 가능한 IAM 역할)에서 IAM 역할을 선택합니다. 이 역할은 이전에 생성하고 Amazon S3에 액세스할 수 있는 역할이어야 합니다. 그런 다음 IAM 역할 연결(Associate IAM role)을 선택하여 클러스터에 대한 연결된 IAM 역할(Associated IAM roles) 목록에 추가합니다.

5. 클러스터 생성을 선택합니다.

Amazon Redshift 시작 안내서 단계에 따라 SQL 클라이언트에서 클러스터에 연결한 후 연결을 테스트합니다. 그 밖에 테이블을 생성하거나 데이터를 업로드하거나 예제 쿼리를 실행하는 나머지 시작하기 단계는 완료할 필요가 없습니다.

2단계: 데이터 파일 다운로드

이 단계에서는 샘플 데이터 파일 세트를 컴퓨터에 다운로드합니다. 다음 단계에서는 파일을 Amazon S3 버킷에 업로드합니다.

데이터 파일을 다운로드하려면

1. 압축된 파일 LoadingDataSampleFiles.zip을 다운로드합니다.
2. 파일을 컴퓨터의 폴더에 압축 해제합니다.
3. 폴더에 다음 파일이 포함되어 있는지 확인합니다.

```
customer-fw-manifest
customer-fw.tbl-000
customer-fw.tbl-000.bak
customer-fw.tbl-001
customer-fw.tbl-002
customer-fw.tbl-003
customer-fw.tbl-004
customer-fw.tbl-005
customer-fw.tbl-006
customer-fw.tbl-007
customer-fw.tbl.log
dwdate-tab.tbl-000
dwdate-tab.tbl-001
dwdate-tab.tbl-002
dwdate-tab.tbl-003
dwdate-tab.tbl-004
dwdate-tab.tbl-005
dwdate-tab.tbl-006
dwdate-tab.tbl-007
```

```
part-csv.tbl-000
part-csv.tbl-001
part-csv.tbl-002
part-csv.tbl-003
part-csv.tbl-004
part-csv.tbl-005
part-csv.tbl-006
part-csv.tbl-007
```

3단계: Amazon S3 버킷에 파일 업로드

이 단계에서는 Amazon S3 버킷을 생성하고 이 버킷에 데이터 파일을 업로드합니다.

Amazon S3 버킷에 파일을 업로드하려면

1. Amazon S3에서 버킷을 생성합니다.

   버킷 생성에 대한 자세한 내용을 알아보려면 Amazon Simple Storage Service 사용 설명서의 버킷 생성을 참조하세요.

   a. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/s3/에서 Amazon S3 콘솔을 엽니다.
   b. 버킷 생성을 선택합니다.
   c. AWS 리전을 선택합니다.

      클러스터와 같은 리전에 버킷을 생성합니다. 클러스터가 미국 서부(오레곤) 리전에 있는 경우 미국 서부(오레곤) 리전(us-west-2)(US West (Oregon) Region (us-west-2))을 선택합니다.
   d. 버킷 생성 대화 상자의 버킷 이름 상자에 버킷 이름을 입력합니다.

      선택한 버킷 이름은 Amazon S3의 모든 기존 버킷 이름을 통틀어 고유해야 합니다. 고유성을 보장하기 위한 한 가지 방법은 버킷 이름의 접두사로 조직 이름을 사용하는 것입니다. 버킷 이름은 특정 규칙을 준수해야 합니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 버킷 규제 및 제한을 참조하세요.
   e. 나머지 옵션의 경우 권장 기본값을 선택합니다.
   f. 버킷 생성을 선택합니다.

      Amazon S3에서 버킷을 생성하면 콘솔의 [버킷(Buckets)] 패널에 빈 버킷이 표시됩니다.
2. 폴더를 만듭니다.

a. 새 버킷의 이름을 선택합니다.

b. 폴더 생성 버튼을 선택합니다.

c. 새 폴더의 이름을 **load**로 지정합니다.

> **Note**
>
> 만든 버킷이 샌드박스에 없습니다. 이 연습에서는 실제 버킷에 객체를 추가합니다. 버킷에 객체를 저장한 시간에 대해 명목 금액이 청구됩니다. Amazon S3 요금에 대한 자세한 내용은 Amazon S3 요금 섹션을 참조하세요.

3. 새 Amazon S3 버킷에 데이터 파일을 업로드합니다.

a. 데이터 폴더의 이름을 선택합니다.

b. 업로드 마법사에서 파일 추가를 선택합니다.

Amazon S3 콘솔 지침에 따라 다운로드하고 추출한 모든 파일을 업로드합니다.

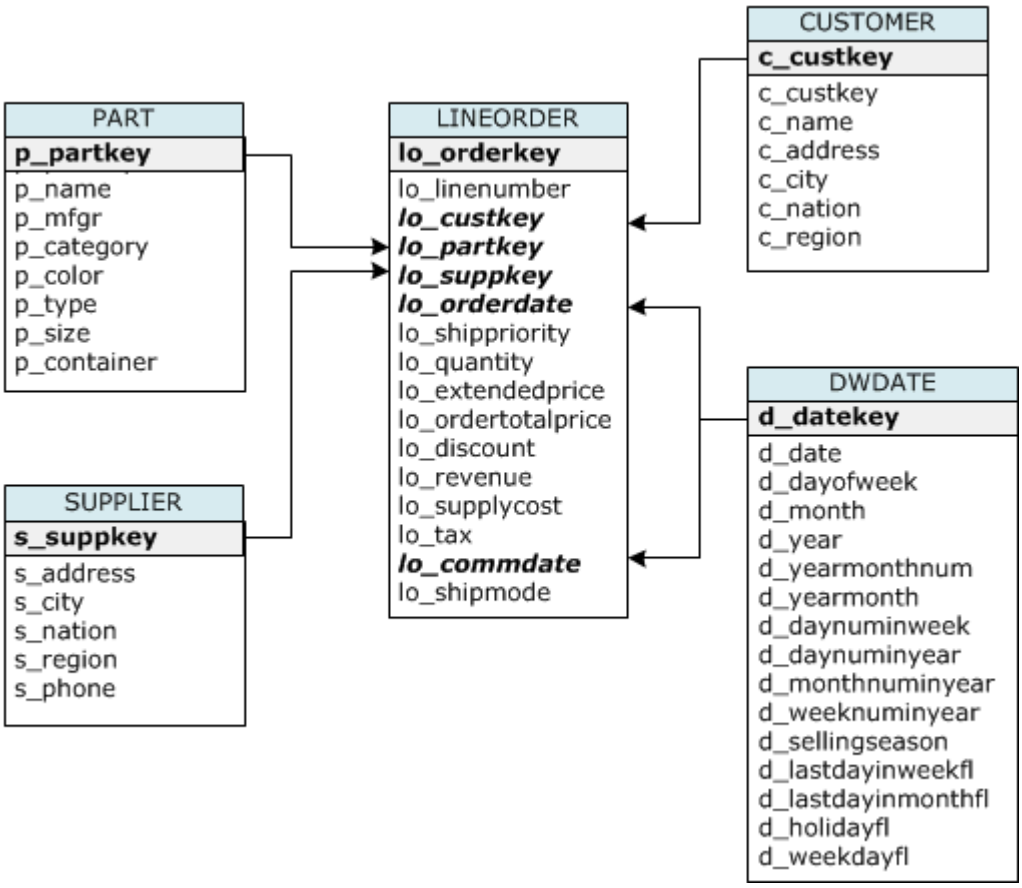
c. 업로드를 선택합니다.

사용자 자격 증명

Amazon Redshift COPY 명령에 Amazon S3 버킷의 파일 객체를 읽을 수 있는 액세스 권한이 있어야 합니다. Amazon S3 버킷 생성에 사용한 사용자 자격 증명이 Amazon Redshift COPY 명령 실행에 사용하는 자격 증명과 같다면 COPY 명령은 필요한 모든 권한을 가지게 됩니다. 다른 사용자 자격 증명을 사용하려는 경우 Amazon S3 액세스 제어를 사용하여 액세스 권한을 부여하면 됩니다. Amazon Redshift COPY 명령이 Amazon S3 버킷의 파일 객체에 액세스하기 위해서는 적어도 ListBucket 및 GetObject 권한이 필요합니다. Amazon S3 리소스 액세스에 대한 자세한 내용은 Amazon S3 리소스에 대한 액세스 권한 관리 섹션을 참조하세요.

4단계: 샘플 테이블 생성

이 자습서에서는 Star Schema Benchmark(SSB) 스키마에 따라 테이블 세트를 사용합니다. 다음 다이어그램은 SSB 데이터 모델을 나타냅니다.



SSB 테이블이 현재 데이터베이스에 이미 존재할 수 있습니다. 이 경우 데이터베이스에서 이 테이블을 삭제한 후 다음 단계에서 CREATE TABLE 명령을 사용해 테이블을 만들어야 합니다. 이 자습서에서 사용하는 테이블은 기존 테이블과 속성이 다를 수 있습니다.

샘플 테이블을 만들려면

1. SSB 테이블을 삭제하려면 SQL 클라이언트에서 다음 명령을 실행합니다.

```
drop table part cascade;
drop table supplier;
drop table customer;
drop table dwdate;
drop table lineorder;
```

2. SQL 클라이언트에서 다음 CREATE TABLE 명령을 실행합니다.

```
CREATE TABLE part
(
  p_partkey     INTEGER NOT NULL,
  p_name        VARCHAR(22) NOT NULL,
```

```
  p_mfgr        VARCHAR(6),
  p_category    VARCHAR(7) NOT NULL,
  p_brand1      VARCHAR(9) NOT NULL,
  p_color       VARCHAR(11) NOT NULL,
  p_type        VARCHAR(25) NOT NULL,
  p_size        INTEGER NOT NULL,
  p_container   VARCHAR(10) NOT NULL
);

CREATE TABLE supplier
(
  s_suppkey   INTEGER NOT NULL,
  s_name      VARCHAR(25) NOT NULL,
  s_address   VARCHAR(25) NOT NULL,
  s_city      VARCHAR(10) NOT NULL,
  s_nation    VARCHAR(15) NOT NULL,
  s_region    VARCHAR(12) NOT NULL,
  s_phone     VARCHAR(15) NOT NULL
);

CREATE TABLE customer
(
  c_custkey      INTEGER NOT NULL,
  c_name         VARCHAR(25) NOT NULL,
  c_address      VARCHAR(25) NOT NULL,
  c_city         VARCHAR(10) NOT NULL,
  c_nation       VARCHAR(15) NOT NULL,
  c_region       VARCHAR(12) NOT NULL,
  c_phone        VARCHAR(15) NOT NULL,
  c_mktsegment   VARCHAR(10) NOT NULL
);

CREATE TABLE dwdate
(
  d_datekey            INTEGER NOT NULL,
  d_date               VARCHAR(19) NOT NULL,
  d_dayofweek          VARCHAR(10) NOT NULL,
  d_month              VARCHAR(10) NOT NULL,
  d_year               INTEGER NOT NULL,
  d_yearmonthnum       INTEGER NOT NULL,
  d_yearmonth          VARCHAR(8) NOT NULL,
  d_daynuminweek       INTEGER NOT NULL,
  d_daynuminmonth      INTEGER NOT NULL,
  d_daynuminyear       INTEGER NOT NULL,
```

```
  d_monthnuminyear     INTEGER NOT NULL,
  d_weeknuminyear      INTEGER NOT NULL,
  d_sellingseason      VARCHAR(13) NOT NULL,
  d_lastdayinweekfl    VARCHAR(1) NOT NULL,
  d_lastdayinmonthfl   VARCHAR(1) NOT NULL,
  d_holidayfl          VARCHAR(1) NOT NULL,
  d_weekdayfl          VARCHAR(1) NOT NULL
);
CREATE TABLE lineorder
(
  lo_orderkey          INTEGER NOT NULL,
  lo_linenumber        INTEGER NOT NULL,
  lo_custkey           INTEGER NOT NULL,
  lo_partkey           INTEGER NOT NULL,
  lo_suppkey           INTEGER NOT NULL,
  lo_orderdate         INTEGER NOT NULL,
  lo_orderpriority     VARCHAR(15) NOT NULL,
  lo_shippriority      VARCHAR(1) NOT NULL,
  lo_quantity          INTEGER NOT NULL,
  lo_extendedprice     INTEGER NOT NULL,
  lo_ordertotalprice   INTEGER NOT NULL,
  lo_discount          INTEGER NOT NULL,
  lo_revenue           INTEGER NOT NULL,
  lo_supplycost        INTEGER NOT NULL,
  lo_tax               INTEGER NOT NULL,
  lo_commitdate        INTEGER NOT NULL,
  lo_shipmode          VARCHAR(10) NOT NULL
);
```

5단계: COPY 명령 실행

COPY 명령을 실행하여 SSB 스키마에서 각각의 테이블을 로드합니다. COPY 명령 예에서는 다양한 파일 형식에서 로드하는 방법, 몇 가지 COPY 명령 옵션을 사용하는 방법, 로드 오류 문제를 해결하는 방법을 보여 줍니다.

COPY 명령 구문

기본적인 COPY 명령 구문은 다음과 같습니다.

```
COPY table_name [ column_list ] FROM data_source CREDENTIALS access_credentials
 [options]
```

COPY 명령을 실행하기 위해서는 다음 값을 제공해야 합니다.

테이블 이름

COPY 명령의 대상 테이블. 이 테이블은 사전에 데이터베이스에 존재해야 하며, 임시 테이블일 수도 있고, 영구 테이블일 수도 있습니다. COPY 명령은 새로운 입력 데이터를 테이블의 기존 행에 추가합니다.

열 목록

기본적으로 COPY는 원본 데이터에서 테이블 열로 필드를 순서대로 로드합니다. 필요할 경우, 열 이름이 쉼표로 분리된 열 목록을 지정하여 데이터 필드를 특정 열에 매핑할 수 있습니다. 이 자습서에서는 열 목록을 사용하지 않습니다. 자세한 내용은 COPY 명령 참조의 Column List 섹션을 참조하세요.

데이터 원본

COPY 명령을 사용하여 Amazon S3 버킷, Amazon EMR 클러스터, SSH 연결을 사용하는 원격 호스트 또는 Amazon DynamoDB 테이블에서 데이터를 로드할 수 있습니다. 이 튜토리얼에서는 Amazon S3 버킷의 데이터 파일에서 데이터를 로드합니다. Amazon S3에서 로드할 때 버킷 이름과 데이터 파일 위치를 제공해야 합니다. 이렇게 하려면 데이터 파일의 객체 경로 또는 각 데이터 파일과 해당 위치를 명시적으로 나열하는 매니페스트 파일의 위치를 제공합니다.

- 키 접두사

  Amazon S3에 저장된 객체는 버킷 이름, 폴더 이름(있는 경우), 객체 이름이 포함된 객체 키에 의해 고유하게 식별됩니다. 키 접두사는 접두사가 동일한 객체 집합을 말합니다. 객체 경로는 키 접두사로서 해당 키 접두사를 공유하는 모든 객체를 로드할 때 COPY 명령이 사용합니다. 예를 들어 키 접두사 `custdata.txt`는 단일 파일 또는 `custdata.txt.001`, `custdata.txt.002` 등등의 파일 집합을 가리킬 수 있습니다.
- 매니페스트 파일

  경우에 따라 여러 버킷 또는 폴더와 같이 접두사가 다른 파일을 로드해야 할 수도 있습니다. 또는 접두사를 공유하는 파일을 제외해야 할 수도 있습니다. 이 경우 매니페스트 파일을 사용할 수 있습니다. 매니페스트 파일은 각각의 로드 파일과 그 고유한 객체 키를 명시적으로 나열합니다. 이 자습서 뒷부분에서는 매니페스트 파일을 사용하여 PART 테이블을 로드합니다.

보안 인증 정보

로드할 데이터가 포함된 AWS 리소스에 액세스하려면 충분한 권한이 있는 사용자의 AWS 액세스 보안 인증 정보를 제공해야 합니다. 이러한 자격 증명에는 IAM 역할 Amazon 리소스 이름(ARN)이 포함됩니

다. Amazon S3에서 데이터를 로드하려면 자격 증명에 ListBucket 및 GetObject 권한이 있어야 합니다. 데이터가 암호화된 경우 추가 자격 증명이 필요합니다. 자세한 내용은 COPY 명령 참조의 권한 부여 파라미터 섹션을 참조하세요. 액세스 관리에 대한 자세한 내용은 Amazon S3 리소스에 대한 액세스 권한 관리 섹션을 참조하세요.

옵션

COPY 명령에 여러 파라미터를 지정하면 파일 형식을 지정하고, 데이터 형식을 관리하고, 오류를 관리하고, 다른 기능을 제어할 수 있습니다. 이 자습서에서 사용하는 COPY 명령 옵션과 기능은 다음과 같습니다.

- 키 접두사

  키 접두사를 지정하여 여러 파일에서 로드하는 방법에 대한 자세한 내용은 NULL AS를 사용하여 PART 테이블 로드 섹션을 참조하세요.
- CSV 형식

  CSV 형식의 데이터를 로드하는 방법에 대한 자세한 내용은 NULL AS를 사용하여 PART 테이블 로드 섹션을 참조하세요.
- NULL AS

  NULL AS 옵션을 사용하여 PART를 로드하는 방법에 대한 자세한 내용은 NULL AS를 사용하여 PART 테이블 로드 섹션을 참조하세요.
- 문자로 구분된 형식

  DELIMITER 옵션을 사용하는 방법에 대한 자세한 내용은 DELIMITER 및 REGION 옵션 섹션을 참조하세요.
- REGION

  REGION 옵션을 사용하는 방법에 대한 자세한 내용은 DELIMITER 및 REGION 옵션 섹션을 참조하세요.
- 고정 형식 너비

  고정 너비 데이터에서 CUSTOMER 테이블을 로드하는 방법에 대한 자세한 내용은 MANIFEST를 사용하여 CUSTOMER 테이블 로드 섹션을 참조하세요.
- MAXERROR

  MAXERROR 옵션을 사용하는 방법에 대한 자세한 내용은 MANIFEST를 사용하여 CUSTOMER 테이블 로드 섹션을 참조하세요.

- ACCEPTINVCHARS

  ACCEPTINVCHARS 옵션을 사용하는 방법에 대한 자세한 내용은 MANIFEST를 사용하여 CUSTOMER 테이블 로드 섹션을 참조하세요.
- MANIFEST

  MANIFEST 옵션을 사용하는 방법에 대한 자세한 내용은 MANIFEST를 사용하여 CUSTOMER 테이블 로드 섹션을 참조하세요.
- DATEFORMAT

  DATEFORMAT 옵션을 사용하는 방법에 대한 자세한 내용은 DATEFORMAT을 사용하여 DWDATE 테이블 로드 섹션을 참조하세요.
- GZIP, LZOP 및 BZIP2

  파일 압축 방법에 대한 자세한 내용은 여러 데이터 파일 로드 섹션을 참조하세요.
- COMPUPDATE

  COMPUPDATE 옵션을 사용하는 방법에 대한 자세한 내용은 여러 데이터 파일 로드 섹션을 참조하세요.
- 여러 파일

  여러 파일을 로드하는 방법에 대한 자세한 내용은 여러 데이터 파일 로드 섹션을 참조하세요.

SSB 테이블 로드

다음 COPY 명령을 사용하여 SSB 스키마에서 각각의 테이블을 로드합니다. 각 테이블에 대한 명령은 다양한 COPY 옵션과 문제 해결 방법을 보여 줍니다.

SSB 테이블을 로드하려면 다음 단계를 따릅니다.

1. 버킷 이름 및 AWS 자격 증명 대체
2. NULL AS를 사용하여 PART 테이블 로드
3. MANIFEST를 사용하여 CUSTOMER 테이블 로드
4. DATEFORMAT을 사용하여 DWDATE 테이블 로드

버킷 이름 및 AWS 자격 증명 대체

이 자습서의 COPY 명령은 다음 형식으로 제시되어 있습니다.

```
copy table from 's3://<your-bucket-name>/load/key_prefix'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
options;
```

각 COPY 명령에 대해 다음을 수행합니다.

1. *<your-bucket-name>*을 클러스터와 같은 리전에 있는 버킷 이름으로 대체합니다.

   이 단계에서는 버킷과 클러스터가 같은 리전에 있다고 가정합니다. 또는 COPY 명령과 함께 REGION 옵션을 사용하여 리전을 지정할 수 있습니다.
2. *<aws-account-id>*와 *<role-name>*을 사용자의 AWS 계정와 IAM 역할로 바꿉니다. 자격 증명 문자열에서 작은따옴표로 묶이는 구간에는 공백이나 줄 바꿈이 있어서는 안 됩니다. ARN은 샘플과 형식이 약간 다를 수 있습니다. COPY 명령을 실행할 때 정확한지 확인하려면 IAM 콘솔에서 역할에 대한 ARN을 복사하는 것이 가장 좋습니다.

NULL AS를 사용하여 PART 테이블 로드

이 단계에서는 CSV 및 NULL AS 옵션을 사용하여 PART 테이블을 로드합니다.

COPY 명령은 여러 파일에서 데이터를 병렬로 로드할 수 있으므로 단일 파일에서 로드하는 것보다 훨씬 빠릅니다. 이 원리를 보여 주기 위해 이 자습서에서는 각 테이블의 데이터를 아주 작은 여덟 개의 파일로 분할합니다. 이후 단계에서는 단일 파일에서 로드할 때와 여러 파일에서 로드할 때의 시간 차이를 비교합니다. 자세한 내용은 파일에서 데이터 로드 단원을 참조하십시오.

키 접두사

파일 세트의 키 접두사를 지정하거나 매니페스트 파일에서 파일을 명시적으로 나열하여 여러 파일에서 데이터를 로드할 수 있습니다. 이 단계에서는 키 접두사를 사용합니다. 매니페스트 파일은 이후 단계에서 사용합니다. 키 접두사 `'s3://amzn-s3-demo-bucket/load/part-csv.tbl'`는 `load` 폴더에 있는 다음 파일 세트를 로드합니다.

```
part-csv.tbl-000
part-csv.tbl-001
part-csv.tbl-002
part-csv.tbl-003
part-csv.tbl-004
part-csv.tbl-005
part-csv.tbl-006
part-csv.tbl-007
```

CSV 형식

쉼표로 분리된 값(comma separated value)을 뜻하는 CSV는 스프레드시트 데이터 가져오기 및 내보내기에 사용되는 공통 형식입니다. CSV는 쉼표로 구분된(comma-delimited) 형식보다 더 유연한데, 필드 내에서 따옴표로 묶인 문자열을 포함시킬 수 있기 때문입니다. CSV 형식에서 COPY의 기본 인용 부호는 큰따옴표(")이지만 QUOTE AS 옵션을 사용하면 다른 인용 부호를 지정할 수 있습니다. 필드 안에서 인용 부호를 사용할 때는 추가 인용 부호로 이스케이프 처리해야 합니다.

PART 테이블용 CSV 형식 데이터 파일에서 발췌한 다음 예는 큰따옴표로 묶인 문자열("LARGE ANODIZED BRASS")을 보여줍니다. 또한 따옴표로 묶인 문자열 안에서 2개의 큰따옴표로 묶인 문자열("MEDIUM ""BURNISHED"" TIN")도 보여줍니다.

```
15,dark sky,MFGR#3,MFGR#47,MFGR#3438,indigo,"LARGE ANODIZED BRASS",45,LG CASE
22,floral beige,MFGR#4,MFGR#44,MFGR#4421,medium,"PROMO, POLISHED BRASS",19,LG DRUM
23,bisque slate,MFGR#4,MFGR#41,MFGR#4137,firebrick,"MEDIUM ""BURNISHED"" TIN",42,JUMBO
 JAR
```

PART 테이블용 데이터에는 COPY가 실패하도록 하는 문자가 포함되어 있습니다. 이 연습에서는 오류 문제를 해결하고 수정합니다.

CSV 형식인 데이터를 로드하려면 COPY 명령에 csv를 추가합니다. 다음 명령을 실행하여 PART 테이블을 로드합니다.

```
copy part from 's3://<your-bucket-name>/load/part-csv.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
csv;
```

다음과 비슷한 오류 메시지가 나타날 수 있습니다.

```
An error occurred when executing the SQL command:
copy part from 's3://amzn-s3-demo-bucket/load/part-csv.tbl'
credentials' ...

ERROR: Load into table 'part' failed.  Check 'stl_load_errors' system table for
 details. [SQL State=XX000]

Execution time: 1.46s

1 statement(s) failed.
1 statement(s) failed.
```

오류에 대한 자세한 내용을 보려면 STL_LOAD_ERRORS 테이블을 쿼리하십시오. 다음 쿼리는 SUBSTRING 함수를 사용해 읽기 쉽게 열을 줄이고 LIMIT 10을 사용해 반환되는 행의 수를 줄입니다. 버킷 이름의 길이를 감안하여 `substring(filename,22,25)`에서 값을 조정할 수 있습니다.

```
select query, substring(filename,22,25) as filename,line_number as line,
substring(colname,0,12) as column, type, position as pos, substring(raw_line,0,30) as
 line_text,
substring(raw_field_value,0,15) as field_text,
substring(err_reason,0,45) as reason
from stl_load_errors
order by query desc
limit 10;
```

```
 query  |    filename      | line |  column   |    type    | pos |
--------+------------------------+-----------+------------+------------+-----+----
 333765 | part-csv.tbl-000 |    1 |           |            |   0 |

 line_text        | field_text |                    reason
------------------+------------+-----------------------------------------------
 15,NUL next,     |            | Missing newline: Unexpected character 0x2c f
```

NULL AS

`part-csv.tbl` 데이터 파일은 NUL 종결자 문자(`\x000` 또는 `\x0`)를 사용하여 NULL 값을 표시합니다.

> **Note**
>
> 철자가 매우 비슷함에도 불구하고 NUL과 NULL은 서로 다릅니다. NUL은 레코드 끝(EOR)을 표시하는 데 종종 사용되는 코드포인트 `x000`이 포함된 UTF-8 문자입니다. NULL은 데이터 없음을 나타내는 SQL 값입니다.

기본적으로 COPY는 NUL 종결자 문자를 EOR 문자로 취급하고 레코드를 종결하므로 종종 예상치 못한 오류가 발생합니다. 텍스트 데이터에서 NULL을 나타내는 단일 표준 방법은 없습니다. 그러므로 테이블을 로드할 때 NULL AS COPY 명령 옵션을 사용하여 NULL로 대체할 문자를 지정할 수 있습니다. 이 예에서는 COPY가 NUL 종결자 문자를 NULL 문자로 취급하도록 합니다.

> **Note**
>
> NULL 값을 받는 테이블 열은 null을 허용하는 열로 구성되어야 합니다. 다시 말해 CREATE TABLE 사양에서 NOT NULL 제약을 포함해서는 안 됩니다.

NULL AS 옵션을 사용하여 PART를 로드하려면 다음 COPY 명령을 실행합니다.

```
copy part from 's3://<your-bucket-name>/load/part-csv.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
csv
null as '\000';
```

COPY가 NULL 값을 로드했는지 확인하려면 다음 명령을 사용하여 NULL이 포함된 행만 선택합니다.

```
select p_partkey, p_name, p_mfgr, p_category from part where p_mfgr is null;
```

```
 p_partkey |  p_name  | p_mfgr | p_category
-----------+----------+--------+------------
        15 | NUL next |        | MFGR#47
        81 | NUL next |        | MFGR#23
       133 | NUL next |        | MFGR#44
(2 rows)
```

DELIMITER 및 REGION 옵션

DELIMITER 및 REGION 옵션은 데이터를 로드하는 방법을 이해하는 데 중요합니다.

문자로 구분된 형식

문자로 구분된 파일의 필드는 파이프 문자(|), 쉼표(,) 또는 탭(\t) 같은 특정 문자로 분리되어 있습니다. 문자로 구분된 파일은 비인쇄 ASCII 문자 중 하나를 포함한 어떤 단일 ASCII 문자도 구분 기호로 사용할 수 있습니다. DELIMITER 옵션을 사용하여 구분 기호 문자를 지정합니다. 기본 구분자는 파이프 문자(|)입니다.

SUPPLIER 테이블용 데이터에서 발췌한 다음 예는 파이프로 구분된 형식을 사용합니다.

```
1|1|257368|465569|41365|19950218|2-HIGH|0|17|2608718|9783671|4|2504369|92072|2|
19950331|TRUCK
```

```
1|2|257368|201928|8146|19950218|2-HIGH|0|36|6587676|9783671|9|5994785|109794|6|
19950416|MAIL
```

REGION

가능하다면 항상 Amazon Redshift 클러스터와 동일한 AWS 리전에 로드 데이터를 위치시켜야 합니다. 데이터와 클러스터가 같은 리전에 있으면 대기 시간이 줄어들고 리전 간 데이터 전송 비용을 피할 수 있습니다. 자세한 내용은 데이터 로드에 대한 Amazon Redshift 모범 사례 단원을 참조하십시오.

다른 AWS 리전에서 데이터를 로드해야 하는 경우 REGION 옵션을 사용하여 로드 데이터가 위치한 AWS 리전을 지정합니다. 리전을 지정하는 경우, 매니페스트 파일을 포함한 모든 로드 데이터가 명명된 리전에 있어야 합니다. 자세한 내용은 REGION 단원을 참조하십시오.

예를 들어 클러스터가 미국 동부(버지니아 북부) 리전에 있고 Amazon S3 버킷이 미국 서부(오리건) 리전에 있는 경우, 다음 COPY 명령은 파이프로 구분된 데이터에서 SUPPLIER 테이블을 로드하는 방법을 보여줍니다.

```
copy supplier from 's3://amzn-s3-demo-bucket/ssb/supplier.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
delimiter '|'
gzip
region 'us-west-2';
```

MANIFEST를 사용하여 CUSTOMER 테이블 로드

이 단계에서는 FIXEDWIDTH, MAXERROR, ACCEPTINVCHARS 및 MANIFEST 옵션을 사용하여 CUSTOMER 테이블을 로드합니다.

이 연습의 샘플 데이터에는 COPY가 로드를 시도할 때 실패를 초래하는 문자가 포함되어 있습니다. MAXERRORS 옵션과 STL_LOAD_ERRORS 시스템 테이블을 사용하여 로드 오류 문제를 해결한 다음 ACCEPTINVCHARS 및 MANIFEST 옵션을 사용하여 오류를 제거합니다.

고정 너비 형식

고정 너비 형식은 구분 기호로 필드를 분리하지 않고 각 필드를 고정된 수의 문자로 정의합니다. CUSTOMER 테이블용 데이터에서 발췌한 다음 예는 고정 너비 형식을 사용합니다.

```
1   Customer#000000001   IVhzIApeRb            MOROCCO  0MOROCCO  AFRICA      25-705
2   Customer#000000002   XSTf4,NCwDVaWNe6tE    JORDAN   6JORDAN   MIDDLE EAST 23-453
3   Customer#000000003   MG9kdTD               ARGENTINA5ARGENTINAAMERICA     11-783
```

레이블/폭 페어의 순서는 테이블 열의 순서와 정확히 일치해야 합니다. 자세한 내용은 FIXEDWIDTH 단원을 참조하십시오.

CUSTOMER 테이블 데이터용 고정 너비 사양 문자열은 다음과 같습니다.

```
fixedwidth 'c_custkey:10, c_name:25, c_address:25, c_city:10, c_nation:15,
c_region :12, c_phone:15,c_mktsegment:10'
```

고정 너비 데이터에서 CUSTOMER 테이블을 로드하려면 다음 명령을 실행합니다.

```
copy customer
from 's3://<your-bucket-name>/load/customer-fw.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
fixedwidth 'c_custkey:10, c_name:25, c_address:25, c_city:10, c_nation:15,
 c_region :12, c_phone:15,c_mktsegment:10';
```

다음과 비슷한 오류 메시지가 표시됩니다.

```
An error occurred when executing the SQL command:
copy customer
from 's3://amzn-s3-demo-bucket/load/customer-fw.tbl'
credentials'...

ERROR: Load into table 'customer' failed.  Check 'stl_load_errors' system table for
 details. [SQL State=XX000]

Execution time: 2.95s

1 statement(s) failed.
```

MAXERROR

기본적으로 COPY 실행 시 처음 오류가 발생할 때 명령이 실패하고 오류 메시지가 반환됩니다. 테스트 도중 시간을 절약하기 위해 MAXERROR 옵션을 사용하여 실패 전에 지정된 수의 오류를 건너뛰도록 COPY에 명령할 수 있습니다. CUSTOMER 테이블 데이터 로드를 처음 테스트할 때 오류를 예상하기 때문에 COPY 명령에 `maxerror 10`을 추가합니다.

FIXEDWIDTH 및 MAXERROR 옵션을 사용하여 테스트하려면 다음 명령을 실행합니다.

```
copy customer
from 's3://<your-bucket-name>/load/customer-fw.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
```

```
fixedwidth 'c_custkey:10, c_name:25, c_address:25, c_city:10, c_nation:15,
 c_region :12, c_phone:15,c_mktsegment:10'
maxerror 10;
```

이번에는 오류 메시지 대신 다음과 비슷한 경고 메시지가 나타납니다.

```
Warnings:
Load into table 'customer' completed, 112497 record(s) loaded successfully.
Load into table 'customer' completed, 7 record(s) could not be loaded.  Check
 'stl_load_errors' system table for details.
```

경고는 COPY 실행 시 일곱 번 오류가 발생했음을 나타냅니다. 오류를 확인하려면 다음 예에 나온 것처럼 STL_LOAD_ERRORS 테이블을 쿼리하십시오.

```
select query, substring(filename,22,25) as filename,line_number as line,
substring(colname,0,12) as column, type, position as pos, substring(raw_line,0,30) as
 line_text,
substring(raw_field_value,0,15) as field_text,
substring(err_reason,0,45) as error_reason
from stl_load_errors
order by query desc, filename
limit 7;
```

STL_LOAD_ERRORS 쿼리 결과가 다음과 비슷해야 합니다.

```
 query  |         filename          | line |  column   |    type    | pos |
 line_text           | field_text |              error_reason
--------+---------------------------+------+-----------
+------------+-----+-------------------------------+------------
+--------------------------------------------
 334489 | customer-fw.tbl.log       |    2 | c_custkey | int4       |  -1 | customer-
fw.tbl               | customer-f | Invalid digit, Value 'c', Pos 0, Type: Integ
 334489 | customer-fw.tbl.log       |    6 | c_custkey | int4       |  -1 | Complete
                  | Complete   | Invalid digit, Value 'C', Pos 0, Type: Integ
 334489 | customer-fw.tbl.log       |    3 | c_custkey | int4       |  -1 | #Total rows
                  | #Total row | Invalid digit, Value '#', Pos 0, Type: Integ
 334489 | customer-fw.tbl.log       |    5 | c_custkey | int4       |  -1 | #Status
                  | #Status    | Invalid digit, Value '#', Pos 0, Type: Integ
 334489 | customer-fw.tbl.log       |    1 | c_custkey | int4       |  -1 | #Load file
                  | #Load file | Invalid digit, Value '#', Pos 0, Type: Integ
 334489 | customer-fw.tbl000        |    1 | c_address | varchar    |  34 | 1
 Customer#000000001  | .Mayag.ezR | String contains invalid or unsupported UTF8
```

```
 334489 | customer-fw.tbl000        |    1 | c_address | varchar    |  34 | 1
 Customer#000000001  | .Mayag.ezR | String contains invalid or unsupported UTF8
(7 rows)
```

결과를 검사하면 error_reasons 열에 2개의 메시지가 있음을 볼 수 있습니다.

- ```
  Invalid digit, Value '#', Pos 0, Type: Integ
  ```

  이 오류의 원인은 customer-fw.tbl.log 파일입니다. 문제는 이것이 데이터 파일이 아니라 로그 파일이며, 로드되어서는 안 된다는 것입니다. 매니페스트 파일을 사용하면 잘못된 파일이 로드되는 것을 피할 수 있습니다.

- ```
  String contains invalid or unsupported UTF8
  ```

  VARCHAR 데이터 형식은 최대 3바이트의 멀티바이트 UTF-8 문자를 지원합니다. 지원되지 않거나 잘못된 문자가 로드 데이터에 포함된 경우, ACCEPTINVCHARS 옵션을 사용하여 잘못된 각각의 문자를 지정된 다른 문자로 대체할 수 있습니다.

로드의 또 다른 문제는 감지하기가 더 어렵습니다. 로드로 인해 예기치 않은 결과가 발생했습니다. 이 문제를 조사하려면 다음 명령을 실행하여 CUSTOMER 테이블을 쿼리합니다.

```
select c_custkey, c_name, c_address
from customer
order by c_custkey
limit 10;
```

```
 c_custkey |          c_name           |         c_address
-----------+---------------------------+---------------------------
         2 | Customer#000000002        | XSTf4,NCwDVaWNe6tE
         2 | Customer#000000002        | XSTf4,NCwDVaWNe6tE
         3 | Customer#000000003        | MG9kdTD
         3 | Customer#000000003        | MG9kdTD
         4 | Customer#000000004        | XxVSJsL
         4 | Customer#000000004        | XxVSJsL
         5 | Customer#000000005        | KvpyuHCplrB84WgAi
         5 | Customer#000000005        | KvpyuHCplrB84WgAi
         6 | Customer#000000006        | sKZz0CsnMD7mp4Xd0YrBvx
         6 | Customer#000000006        | sKZz0CsnMD7mp4Xd0YrBvx
(10 rows)
```

행들이 고유해야 하지만 중복된 행이 있습니다.

예상치 못한 결과를 검사하는 또 다른 방법은 로드된 행의 수를 확인하는 것입니다. 이 경우, 100,000개의 행이 로드되어야 하지만 로드 메시지는 112,497개의 레코드를 로드했다고 보고했습니다. 이 여분의 행은 불필요한 파일인 customer-fw.tbl0000.bak을 COPY가 로드했기 때문에 로드되었습니다.

이 연습에서는 잘못된 파일이 로드되는 것을 방지하기 위해 매니페스트 파일을 사용합니다.

ACCEPTINVCHARS

기본적으로 COPY는 열의 데이터 형식이 지원하지 않는 문자가 있으면 행을 건너뛰고 오류를 반환합니다. 잘못된 UTF-8 문자에 대한 자세한 내용은 멀티바이트 문자 로드 오류 섹션을 참조하세요.

MAXERRORS 옵션을 사용하여 오류를 무시하고 로드를 계속한 다음 STL_LOAD_ERRORS를 쿼리해 잘못된 문자를 찾은 후 데이터 파일을 수정할 수 있습니다. 하지만 MAXERRORS는 로드 문제 해결에 사용하는 것이 가장 좋으며, 일반적으로 프로덕션 환경에서는 사용하지 말아야 합니다.

잘못된 문자를 관리하는 데는 일반적으로 ACCEPTINVCHARS 옵션을 택하는 것이 더 좋습니다. ACCEPTINVCHARS는 COPY에게 잘못된 각 문자를 지정된 유효한 문자로 대체하고 로드 작업을 계속하라고 명령합니다. NULL을 제외한 어떤 ASCII 문자도 대체 문자로 지정할 수 있습니다. 기본 대체 문자는 물음표(?)입니다. COPY는 멀티바이트 문자를 같은 길이의 대체 문자열로 대체합니다. 예를 들어 4바이트 문자는 '????'로 대체됩니다.

COPY는 유효하지 않은 UTF-8 문자가 포함된 행 수를 반환합니다. 또한 영향을 받는 각 행에 대해 STL_REPLACEMENTS 시스템 테이블에 노드 조각당 최대 100행까지 항목을 추가합니다. 이후 추가되는 잘못된 UTF-8 문자 역시 변경되지만 이러한 문자의 변경 이벤트는 기록되지 않습니다.

ACCEPTINVCHARS는 VARCHAR 열에서만 유효합니다.

이 단계에서는 대체 문자 '^'과 함께 ACCEPTINVCHARS를 추가합니다.

MANIFEST

키 접두사를 사용하여 Amazon S3에서 COPY를 실행할 때는 원치 않는 테이블이 로드될 위험이 있습니다. 예를 들어 's3://amzn-s3-demo-bucket/load/ 폴더에는 키 접두사 customer-fw.tbl: customer-fw.tbl0000, customer-fw.tbl0001 등을 공유하는 8개의 데이터 파일이 포함되어 있습니다. 하지만 같은 폴더에 불필요한 파일인 customer-fw.tbl.log와 customer-fw.tbl-0001.bak도 포함되어 있습니다.

올바른 파일만 모두 로드하려면 매니페스트 파일을 사용하십시오. 매니페스트는 로드할 각 원본 파일의 고유한 객체 키를 명시적으로 나열하는 JSON 형식의 텍스트 파일입니다. 파일 객체는 서로 다른 폴더나 버킷에 있을 수 있지만 리전은 같아야 합니다. 자세한 내용은 MANIFEST 단원을 참조하십시오.

다음은 customer-fw-manifest 텍스트를 보여줍니다.

```
{
  "entries": [
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-000"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-001"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-002"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-003"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-004"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-005"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-006"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-007"}
    ]
}
```

매니페스트 파일을 사용하여 CUSTOMER 테이블의 데이터를 로드하려면

1. 텍스트 편집기에서 파일을 엽니다.customer-fw-manifest
2. *<your-bucket-name>*을 버킷의 이름으로 바꿉니다.
3. 파일을 저장합니다.
4. 파일을 버킷의 로드 폴더에 업로드합니다.
5. 다음 COPY 명령을 실행합니다.

```
copy customer from 's3://<your-bucket-name>/load/customer-fw-manifest'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
fixedwidth 'c_custkey:10, c_name:25, c_address:25, c_city:10, c_nation:15,
 c_region :12, c_phone:15,c_mktsegment:10'
maxerror 10
acceptinvchars as '^'
manifest;
```

DATEFORMAT을 사용하여 DWDATE 테이블 로드

이 단계에서는 DELIMITER 및 DATEFORMAT 옵션을 사용하여 DWDATE 테이블을 로드합니다.

DATE 및 TIMESTAMP 열을 로드할 때 COPY는 기본 형식(날짜는 YYYY-MM-DD, 타임스탬프는 YYYY-MM-DD HH:MI:SS)을 예상합니다. 로드 데이터가 기본 형식을 사용하지 않는 경우, DATEFORMAT 및 TIMEFORMAT을 사용하여 형식을 지정할 수 있습니다.

발췌한 다음 예는 DWDATE 테이블의 날짜 형식을 보여 줍니다. 열 2의 날짜 형식이 일관되지 않다는 점을 눈여겨보십시오.

```
19920104 1992-01-04          Sunday  January 1992 199201 Jan1992 1 4 4 1...
19920112 January 12, 1992 Monday  January 1992 199201 Jan1992 2 12 12 1...
19920120 January 20, 1992 Tuesday     January 1992 199201 Jan1992 3 20 20 1...
```

DATEFORMAT

날짜 형식은 하나만 지정할 수 있습니다. 로드 데이터에 서로 다른 열에서 일관되지 않은 형식이 포함되어 있거나 형식이 로드 시에 알려지지 않은 경우, `'auto'` 인수와 함께 DATEFORMAT을 사용합니다. `'auto'`가 지정되면 COPY는 유효한 날짜 또는 시간 형식을 인식하여 이를 기본 형식으로 변환합니다. `'auto'` 옵션은 DATEFORMAT 및 TIMEFORMAT 문자열 사용 시 지원되지 않는 몇 가지 형식을 인식합니다. 자세한 내용은 DATEFORMAT 및 TIMEFORMAT 옵션의 자동 인식 사용 단원을 참조하십시오.

DWDATE 테이블을 로드하려면 다음 COPY 명령을 실행합니다.

```
copy dwdate from 's3://<your-bucket-name>/load/dwdate-tab.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
delimiter '\t'
dateformat 'auto';
```

여러 데이터 파일 로드

GZIP 및 COMPUPDATE 옵션을 사용하여 테이블을 로드할 수 있습니다.

단일 데이터 파일 또는 여러 파일에서 테이블을 로드할 수 있습니다. 이렇게 하면 두 방법의 로드 시간을 비교할 수 있습니다.

GZIP, LZOP 및 BZIP2

gzip, lzop 또는 bzip2 압축 형식을 사용하여 파일을 압축할 수 있습니다. 압축된 파일에서 로드하면 COPY가 로드 프로세스 중에 파일 압축을 해제합니다. 파일을 압축하면 스토리지 공간이 절약되고 업로드 시간이 단축됩니다.

COMPUPDATE

압축 인코딩 없는 빈 테이블을 로드하는 경우, COPY는 로드 데이터를 분석하여 최적의 인코딩을 결정합니다. 그런 다음 로드를 시작하기 전에 이러한 인코딩을 사용하도록 테이블을 변경합니다. 이 분석 프로세스는 시간이 걸리지만 테이블당 최대 한 번 실행됩니다. 시간을 절약하려면 COMPUPDATE를 꺼서 이 단계를 건너뛸 수 있습니다. COPY 시간의 정확한 평가가 가능하도록 하려면 이 단계에서 COMPUPDATE를 끄십시오.

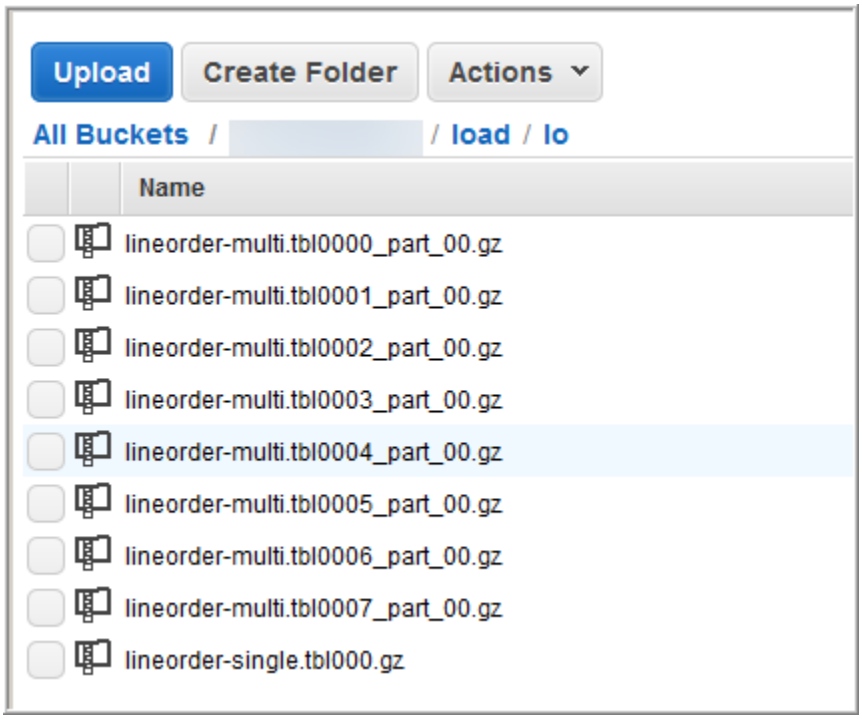
여러 파일

COPY 명령은 단일 파일에서 로드하는 대신 여러 파일에서 데이터를 병렬로 로드할 때 데이터를 매우 효율적으로 로드할 수 있습니다. 파일 수가 클러스터에 있는 조각 수의 배수가 되도록 데이터를 파일로 분할할 수 있습니다. 이 경우 Amazon Redshift는 워크로드를 분할하고 슬라이스 간에 데이터를 균등하게 배포합니다. 노드당 조각 수는 클러스터의 노드 크기에 따라 달라집니다. 각 노드 크기의 슬라이스 수에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 클러스터 및 노드 정보 섹션을 참조하세요.

예를 들어 이 자습서에서 사용하는 클러스터의 컴퓨팅 노드는 각각 2개의 조각을 가질 수 있습니다. 따라서 4노드 클러스터의 조각은 총 8개입니다. 앞 단계에서는 파일이 매우 작음에도 불구하고 여덟 개의 파일에 로드 데이터가 포함되었습니다. 큰 단일 파일에서 로드할 때와 여러 파일에서 로드할 때의 시간 차이를 비교할 수 있습니다.

1,500만 개의 레코드를 포함하고 약 1.2GB를 차지하는 파일도 Amazon Redshift의 규모에서는 매우 작습니다. 그러나 이 파일들은 여러 파일에서 로드할 경우의 성능상 장점을 보여 주는 데는 충분합니다.

다음 이미지는 LINEORDER를 위한 데이터 파일을 보여줍니다.



여러 파일에서의 COPY 성능을 평가하려면

1. 실험실 테스트에서 다음 명령을 실행하여 단일 파일에서 COPY 작업을 수행했습니다. 이 명령은 가상의 버킷을 보여줍니다.

```
copy lineorder from 's3://amzn-s3-demo-bucket/load/lo/lineorder-single.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
gzip
compupdate off
region 'us-east-1';
```

2. 결과는 다음과 같았습니다. 실행 시간을 적어 둡니다.

```
Warnings:
Load into table 'lineorder' completed, 14996734 record(s) loaded successfully.

0 row(s) affected.
copy executed successfully

Execution time: 51.56s
```

3. 그런 다음, 다음 명령을 실행하여 여러 파일에서 COPY 명령을 실행했습니다.

```
copy lineorder from 's3://amzn-s3-demo-bucket/load/lo/lineorder-multi.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
gzip
compupdate off
region 'us-east-1';
```

4. 결과는 다음과 같았습니다. 실행 시간을 적어 둡니다.

```
Warnings:
Load into table 'lineorder' completed, 14996734 record(s) loaded successfully.

0 row(s) affected.
copy executed successfully

Execution time: 17.7s
```

5. 실행 시간을 비교합니다.

이 실험에서는 1,500만 개의 레코드를 로드하는 시간이 51.56초에서 17.7초로 줄어들어 65.7% 단축되었습니다.

이 결과는 4노드 클러스터 사용을 기준으로 한 것입니다. 클러스터에 더 많은 노드가 있는 경우, 시간이 더욱 단축됩니다. 수십에서 수백 개의 노드가 있는 일반적인 Amazon Redshift 클러스터의 경우 이 차이는 더 크게 벌어집니다. 단일 노드 클러스터인 경우, 실행 시간의 차이가 거의 없습니다.

6단계: 데이터베이스 vacuum 및 분석

상당히 많은 행을 추가, 삭제 또는 수정할 때마다 VACUUM 명령을 실행한 다음 ANALYZE 명령을 실행해야 합니다. vacuum은 삭제된 행에서 공간을 회수하고 정렬 순서를 복원합니다. ANALYZE 명령은 통계 메타데이터를 업데이트하므로 쿼리 옵티마이저가 더 정확한 쿼리 계획을 생성할 수 있습니다. 자세한 내용은 테이블 Vacuum 단원을 참조하십시오.

데이터를 정렬 키 순서로 로드하는 경우에는 vacuum이 빠릅니다. 이 자습서에서 상당히 많은 행을 추가했지만 빈 테이블에 추가한 것입니다. 따라서 재정렬할 필요가 없고 삭제한 행도 없습니다. COPY는 빈 테이블을 로드한 후 자동으로 통계를 업데이트하므로 통계는 최신 상태입니다. 다만 깔끔한 정리를 위해 데이터베이스에 대해 vacuum을 수행하고 분석하여 이 자습서를 마치십시오.

데이터베이스를 vacuum하고 분석하려면 다음 명령을 실행합니다.

```
vacuum;
analyze;
```

7단계: 리소스 정리

클러스터는 실행하는 동안 계속해서 요금이 발생합니다. 이 튜토리얼을 완료하면 Amazon Redshift 시작 안내서의 5단계: 액세스 취소 및 샘플 클러스터 삭제 단계에 따라 환경을 이전 상태로 되돌려야 합니다.

클러스터를 유지하면서 SSB 테이블에서 사용한 스토리지를 복원하고 싶다면 다음 명령을 실행하세요.

```
drop table part;
drop table supplier;
drop table customer;
drop table dwdate;
drop table lineorder;
```

다음

요약

요약

이 튜토리얼에서는 데이터 파일을 Amazon S3에 업로드한 다음 COPY 명령을 사용하여 파일에서 Amazon Redshift 테이블로 데이터를 로드했습니다.

데이터를 로드할 때 사용한 형식은 다음과 같습니다.

- 문자로 구분된 형식
- CSV
- 고정 너비 형식

STL_LOAD_ERRORS 시스템 테이블을 사용하여 로드 오류 문제를 해결한 다음 REGION, MANIFEST, MAXERROR, ACCEPTINVCHARS, DATEFORMAT 및 NULL AS 옵션을 사용하여 오류를 해결했습니다.

데이터 로드에 다음의 모범 사례를 적용했습니다.

- COPY 명령을 사용하여 데이터 로드
- 파일에서 데이터 로드
- 단일 COPY 명령을 사용하여 여러 파일에서 로드
- 데이터 파일 압축
- 로드 전후의 데이터 파일 확인

Amazon Redshift 모범 사례에 대한 자세한 내용은 다음 링크를 참조하세요.

- 데이터 로드에 대한 Amazon Redshift 모범 사례
- Amazon Redshift 테이블 설계 모범 사례
- Amazon Redshift 쿼리 설계 모범 사례

Amazon Redshift에서 데이터 언로드

데이터베이스 테이블에서 Amazon S3 버킷의 파일로 데이터를 언로드할 때는 SELECT 문에서 UNLOAD 명령을 사용하면 됩니다. 텍스트 데이터는 로드할 때 사용한 데이터 형식에 상관없이 구분 형식 또는 고정 폭 형식으로 언로드할 수 있습니다. 또한 GZIP 압축 파일의 생성 여부도 지정할 수 있습니다.

Amazon S3 버킷에 대한 사용자의 액세스 권한은 임시 보안 자격 증명을 사용하여 제한할 수 있습니다.

주제

Amazon S3로 데이터 언로드

Amazon Redshift는 데이터를 병렬 방식으로 쉽게 다시 로드할 수 있도록 select 문의 결과를 다수의 파일, 즉 노드 조각당 1개 이상의 파일로 분할합니다. 또는 PARALLEL OFF 옵션을 추가하여 UNLOAD가 결과를 1개 이상의 파일에 직렬 방식으로 작성하도록 지정할 수도 있습니다. Amazon S3의 파일 크기는 MAXFILESIZE 파라미터를 지정하여 제한할 수 있습니다. UNLOAD는 Amazon S3 서버 측 암호화(SSE-S3)를 사용하여 데이터 파일을 자동으로 암호화합니다.

Amazon Redshift에서 지원되는 UNLOAD 명령에서는 모든 select 문을 사용할 수 있습니다. 단, 바깥쪽 select에서 LIMIT 절을 사용하는 SELECT는 예외입니다. 예를 들어 특정 열을 추가하거나, 혹은 WHERE 절을 사용하여 다수의 테이블을 조인하는 SELECT 문을 사용할 수 있습니다. 쿼리에 인용 부호(리터럴 값을 묶는 등)가 포함되어 있으면 쿼리 텍스트에서 이스케이프 처리해야 합니다(\'). 자세한 내용은 SELECT&; Command Reference를 참조하십시오. LIMIT 절의 사용에 대한 자세한 내용은 UNLOAD 명령의 사용 노트 섹션을 참조하세요.

예를 들어 다음 UNLOAD 명령은 VENUE 테이블의 내용을 Amazon S3 버킷인 `s3://amzn-s3-demo-bucket/tickit/unload/`로 전송합니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/tickit/unload/venue_'
```

```
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

위 예에서 생성한 파일 이름에는 접두사 'venue_'가 추가됩니다.

```
venue_0000_part_00
venue_0001_part_00
venue_0002_part_00
venue_0003_part_00
```

기본적으로 UNLOAD는 클러스터의 조각 수에 따라 다수의 파일에 병렬 방식으로 데이터를 작성합니다. 데이터를 단일 파일에 작성할 때는 PARALLEL OFF를 지정하십시오. 그러면 UNLOAD가 데이터를 직렬 방식으로 작성하여 절대적으로 ORDER BY 절(사용하는 경우)에 따라 정렬됩니다. 데이터 파일의 최대 크기는 6.2GB입니다. 데이터 크기가 최대 파일 크기보다 크면 UNLOAD가 각각 6.2GB까지 파일을 추가로 생성합니다.

다음은 VENUE의 내용을 단일 파일로 작성하는 예입니다. 파일 크기가 6.2GB보다 작기 때문에 파일이 하나만 필요합니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/tickit/unload/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
parallel off;
```

> **Note**
>
> UNLOAD 명령은 병렬 처리를 사용하도록 설계되었습니다. 따라서 대부분 경우, 특히 COPY 명령에서 파일을 사용하여 테이블을 로드한다면 PARALLEL을 활성화하는 것이 좋습니다.

VENUE 테이블의 전체 데이터 크기가 5GB라고 가정했을 때 다음 예에서는 VENUE의 데이터를 각각 100MB씩 50개 파일로 작성합니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/tickit/unload/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
parallel off
maxfilesize 100 mb;
```

이때 접두사를 Amazon S3 경로 문자열에 추가하면 UNLOAD가 파일 이름에 접두사를 사용합니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/tickit/unload/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

UNLOAD 명령에서 MANIFEST 옵션을 지정하면 언로드 파일이 나열되는 매니페스트 파일을 생성할 수 있습니다. 매니페스트란 Amazon S3에 작성한 개별 파일의 URL을 명시적으로 나열한 JSON 형식의 텍스트 파일을 말합니다.

다음은 MANIFEST 옵션을 추가하는 예입니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/tickit/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest;
```

다음은 언로드 파일이 4개인 매니페스트 예입니다.

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket/tickit/venue_0000_part_00"},
    {"url":"s3://amzn-s3-demo-bucket/tickit/venue_0001_part_00"},
    {"url":"s3://amzn-s3-demo-bucket/tickit/venue_0002_part_00"},
    {"url":"s3://amzn-s3-demo-bucket/tickit/venue_0003_part_00"}
  ]
}
```

COPY에서 MANIFEST 옵션을 지정하면 동일한 파일을 로드하는 데 매니페스트 파일을 사용할 수도 있습니다. 자세한 내용은 매니페스트를 사용하여 데이터 파일 지정 단원을 참조하십시오.

UNLOAD 작업을 마치면 UNLOAD가 파일을 작성한 Amazon S3 버킷으로 이동하여 데이터가 정확하게 언로드되었는지 확인합니다. 조각마다 0부터 시작하여 번호가 매겨진 파일이 다수 표시됩니다. MANIFEST 옵션을 지정하였다면 여기에서 'manifest'로 끝나는 파일도 볼 수 있습니다. 예:

```
amzn-s3-demo-bucket/tickit/venue_0000_part_00
amzn-s3-demo-bucket/tickit/venue_0001_part_00
amzn-s3-demo-bucket/tickit/venue_0002_part_00
amzn-s3-demo-bucket/tickit/venue_0003_part_00
amzn-s3-demo-bucket/tickit/venue_manifest
```

UNLOAD가 완료된 후 Amazon S3 목록 작업을 호출하여 Amazon S3에 기록된 파일 목록을 프로그래밍 방식으로 가져올 수 있습니다. STL_UNLOAD_LOG를 쿼리할 수도 있습니다.

다음은 UNLOAD에서 생성된 파일의 경로 이름을 반환하는 쿼리입니다. PG_LAST_QUERY_ID 함수는 가장 최근 쿼리를 반환합니다.

```
select query, substring(path,0,40) as path
from stl_unload_log
where query=2320
order by path;

query |             path
-------+---------------------------------------
  2320 | s3://amzn-s3-demo-bucket/venue0000_part_00
  2320 | s3://amzn-s3-demo-bucket/venue0001_part_00
  2320 | s3://amzn-s3-demo-bucket/venue0002_part_00
  2320 | s3://amzn-s3-demo-bucket/venue0003_part_00
(4 rows)
```

데이터 용량이 너무 크면 Amazon Redshift가 각 조각마다 여러 부분으로 파일을 분할할 수도 있습니다. 예:

```
venue_0000_part_00
venue_0000_part_01
venue_0000_part_02
venue_0001_part_00
venue_0001_part_01
venue_0001_part_02
...
```

다음은 select 문에 인용 부호로 묶인 문자열이 포함되어 인용 부호가 이스케이프 처리(=\'OH\' ')되는 UNLOAD 명령입니다.

```
unload ('select venuename, venuecity from venue where venuestate=\'OH\' ')
to 's3://amzn-s3-demo-bucket/tickit/venue/ '
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

기본적으로 UNLOAD는 대상 버킷의 기존 파일을 덮어쓰지 않고 중단됩니다. 매니페스트 파일을 포함하여 기존 파일을 덮어쓰려면 ALLOWOVERWRITE 옵션을 지정합니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/venue_pipe_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest
```

```
allowoverwrite;
```

암호화된 데이터 파일 언로드

UNLOAD는 AWS 관리형 암호화 키를 사용하는 Amazon S3 서버 측 암호화(SSE-S3)를 통해 파일을 자동 생성합니다. AWS Key Management Service 키(SSE-KMS)를 사용하는 서버 측 암호화를 지정하거나 고객 관리형 키를 사용하는 클라이언트 측 암호화를 지정할 수도 있습니다. UNLOAD는 고객 제공 키를 사용하는 Amazon S3 서버 측 암호화는 지원하지 않습니다. 자세한 내용은 서버 측 암호화를 사용하여 데이터 보호를 참조하세요.

AWS KMS 키를 사용하는 서버 측 암호화를 통해 Amazon S3로 언로드하려면 다음 예와 같이 KMS_KEY_ID 파라미터를 지정하여 키 ID를 입력하세요.

```
unload ('select venuename, venuecity from venue')
to 's3://amzn-s3-demo-bucket/encrypted/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
KMS_KEY_ID '1234abcd-12ab-34cd-56ef-1234567890ab'
encrypted;
```

사용자 고유의 암호화 키를 입력하고 싶다면 UNLOAD 명령에서 ENCRYPTED 옵션을 사용하여 클라이언트 측에서 암호화된 데이터 파일을 Amazon S3에 생성할 수 있습니다. 그러면 UNLOAD가 Amazon S3 클라이언트 측 암호화에서 사용하는 것과 동일한 봉투 암호화 프로세스를 사용합니다. 그런 다음 COPY 명령에서 ENCRYPTED 옵션을 사용하여 암호화된 파일을 로드할 수 있습니다.

프로세스는 다음과 같습니다.

1. 프라이빗 암호화 키, 즉 루트 대칭 키로 사용할 base64 인코딩 256비트 AES 키를 생성합니다.
2. 루트 대칭 키와 ENCRYPTED 옵션을 추가하여 UNLOAD 명령을 실행합니다.
3. UNLOAD가 1회용 대칭키(봉투 대칭 키로도 불림)와 데이터 암호화에 사용되는 초기화 벡터(IV)를 생성합니다.
4. UNLOAD가 루트 대칭 키를 사용하여 봉투 대칭 키를 암호화합니다.
5. UNLOAD가 암호화된 데이터 파일을 Amazon S3에 저장하고, 각 파일과 함께 암호화된 봉투 키와 IV를 객체 메타데이터로 저장합니다. 암호화된 봉투 키는 객체 메타데이터 `x-amz-meta-x-amz-key`로 저장되고, IV는 객체 메타데이터 `x-amz-meta-x-amz-iv`로 저장됩니다.

봉투 암호화 프로세스에 대한 자세한 내용은 Client-side data encryption with the AWS SDK for Java and Amazon S3 문서를 참조하세요.

암호화된 데이터 파일을 언로드하려면 루트 키 값을 자격 증명 문자열에 입력한 후 ENCRYPTED 옵션을 추가합니다. MANIFEST 옵션을 사용하면 매니페스트 파일도 암호화됩니다.

```
unload ('select venuename, venuecity from venue')
to 's3://amzn-s3-demo-bucket/encrypted/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key '<root_key>'
manifest
encrypted;
```

암호화되면서 GZIP으로 압축된 데이터 파일을 언로드하려면 루트 키 값 및 ENCRYPTED 옵션과 함께 GZIP 옵션을 추가합니다.

```
unload ('select venuename, venuecity from venue')
to 's3://amzn-s3-demo-bucket/encrypted/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key '<root_key>'
encrypted gzip;
```

암호화된 데이터 파일을 로드하려면 MASTER_SYMMETRIC_KEY 파라미터에 동일한 루트 키 값을 입력한 후 ENCRYPTED 옵션을 추가합니다.

```
copy venue from 's3://amzn-s3-demo-bucket/encrypted/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key '<root_key>'
encrypted;
```

구분 또는 고정 폭 형식의 데이터 언로드

데이터를 구분 형식 또는 고정 폭 형식으로 언로드할 수 있습니다. 기본 출력은 파이프 구분('|' 문자 사용) 형식입니다.

다음은 쉼표를 구분자로 지정하는 예입니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/tickit/venue/comma'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter ',';
```

그 결과 출력되는 파일의 모습은 다음과 같습니다.

```
20,Air Canada Centre,Toronto,ON,0
60,Rexall Place,Edmonton,AB,0
100,U.S. Cellular Field,Chicago,IL,40615
200,Al Hirschfeld Theatre,New York City,NY,0
240,San Jose Repertory Theatre,San Jose,CA,0
300,Kennedy Center Opera House,Washington,DC,0
...
```

동일한 결과 집합을 탭으로 구분된 파일로 언로드하려면 다음과 같이 명령을 실행합니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/tickit/venue/tab'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter as '\t';
```

그 밖에 FIXEDWIDTH 명세를 사용할 수도 있습니다. 이 명세는 각 테이블 열의 식별자와 열의 폭(문자 수)으로 구성됩니다. UNLOAD 명령은 데이터를 자르기보다는 중단되기 때문에 폭을 지정할 때는 적어도 열에서 가장 긴 항목만큼 길어야 합니다. 고정 폭 데이터의 언로드는 구분 데이터의 언로드와 비슷하지만 결과적으로 구분 문자가 출력되지 않는다는 점이 다릅니다. 예:

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/tickit/venue/fw'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
fixedwidth '0:3,1:100,2:30,3:2,4:6';
```

예:

```
20 Air Canada Centre         Toronto      ON0
60 Rexall Place              Edmonton     AB0
100U.S. Cellular Field       Chicago      IL40615
200Al Hirschfeld Theatre     New York CityNY0
240San Jose Repertory TheatreSan Jose     CA0
300Kennedy Center Opera HouseWashington   DC0
```

FIXEDWIDTH 명세에 대한 자세한 내용은 UNLOAD 명령을 참조하십시오.

언로드된 데이터 다시 로드

언로드 작업 결과를 다시 로드하려면 COPY 명령을 사용할 수 있습니다.

다음은 VENUE 테이블이 매니페스트 파일을 사용해 언로드된 후 길이가 잘려서 다시 로드되는 간단한 예입니다.

```
unload  ('select * from venue order by venueid')
to 's3://amzn-s3-demo-bucket/tickit/venue/reload_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest
delimiter '|';

truncate venue;

copy venue
from 's3://amzn-s3-demo-bucket/tickit/venue/reload_manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest
delimiter '|';
```

다시 로드된 VENUE 테이블의 모습은 다음과 같습니다.

```
select * from venue order by venueid limit 5;

 venueid |         venuename          |  venuecity  | venuestate | venueseats
---------+----------------------------+-------------+------------+-----------
       1 | Toyota Park                | Bridgeview  | IL         |          0
       2 | Columbus Crew Stadium      | Columbus    | OH         |          0
       3 | RFK Stadium                | Washington  | DC         |          0
       4 | CommunityAmerica Ballpark | Kansas City | KS         |          0
       5 | Gillette Stadium           | Foxborough  | MA         |      68756
(5 rows)
```

Amazon Redshift의 사용자 정의 함수

SQL SELECT 절 또는 Python 프로그램을 이용해 사용자 지정 스칼라 사용자 정의 함수(UDF)를 만들 수 있습니다. 새 함수는 데이터베이스에 저장되고 실행할 충분한 권한이 있는 사용자가 사용할 수 있습니다. 기존 Amazon Redshift 함수를 실행하는 것과 거의 동일한 방식으로 사용자 정의 스칼라 UDF를 실행합니다.

Python UDF에는 표준 Python 함수를 사용하는 것은 물론이고 사용자 지정 Python 모듈을 가져올 수도 있습니다. 자세한 내용은 UDF에 대한 Python 언어 지원 단원을 참조하십시오. 참고로 Python 3는 Python UDF에는 사용할 수 없습니다. Amazon Redshift UDF에 대한 Python 3 지원을 얻으려면 Scalar Lambda UDF을 사용하세요.

Lambda에 정의된 사용자 정의 함수를 SQL 쿼리의 일부로 사용하는 AWS Lambda UDF를 생성할 수도 있습니다. Lambda UDF를 사용하면 복잡한 UDF를 작성하고 서드 파티 구성 요소와 통합할 수 있습니다. 또한 현재 Python과 SQL UDF의 몇 가지 한계를 극복하는 데 도움이 될 수 있습니다. 예를 들어 네트워크 및 스토리지 리소스에 액세스하고 보다 본격적인 SQL 문을 작성하는 데 도움이 될 수 있습니다. Java, Go, PowerShell, Node.js, C#, Python 및 Ruby와 같이 Lambda에서 지원하는 모든 프로그래밍 언어로 Lambda UDF를 생성할 수 있습니다. 또는 사용자 정의 런타임을 사용할 수 있습니다.

기본적으로 모든 사용자가 UDF를 실행할 수 있습니다. 권한에 대한 자세한 내용은 UDF 보안 및 권한 섹션을 참조하세요.

주제

UDF 보안 및 권한

UDF를 새로 만들려면 SQL 또는 plpythonu(Python)에 대한 언어에 사용 권한이 필요합니다. 기본적으로 USAGE ON LANGUAGE SQL이 PUBLIC에 허용되지만, 특정 사용자 또는 그룹에게 USAGE ON LANGUAGE PLPYTHONU 권한을 명시적으로 허용해야만 합니다.

SQL에 대한 사용을 취소하려면 먼저 PUBLIC에서의 사용을 취소해야 합니다. 그런 다음 SQL UDF 생성이 허용된 사용자 또는 그룹에게만 SQL에 대한 사용 권한을 허용합니다. 다음 예에서는 PUBLIC의 SQL 사용 권한을 취소합니다. 그런 다음 사용자 그룹 udf_devs에 사용 권한을 부여합니다.

```
revoke usage on language sql from PUBLIC;
grant usage on language sql to group udf_devs;
```

UDF를 실행하려면 각 함수마다 실행 권한이 필요합니다. 기본적으로 새로운 UDF를 실행하는 권한은 PUBLIC에 허용됩니다. 사용을 제한하려면 함수에 대해 PUBLIC에서 이 권한을 취소합니다. 그런 다음 개인 혹은 그룹에 권한을 허용합니다.

다음 예는 PUBLIC의 f_py_greater 함수 실행 권한을 취소합니다. 그런 다음 사용자 그룹 udf_devs에 사용 권한을 부여합니다.

```
revoke execute on function f_py_greater(a float, b float) from PUBLIC;
grant execute on function f_py_greater(a float, b float) to group udf_devs;
```

기본적으로 수퍼유저는 모든 권한을 갖습니다.

자세한 내용은 GRANT 및 REVOKE 단원을 참조하세요.

UDF 이름 충돌 방지

UDF를 실행하기 전에 명명 규칙을 고려하여 잠재적 충돌이나 뜻밖의 결과를 피할 수 있습니다. 함수 이름은 오버로딩될 수 있기 때문에 이전이나 이후의 Amazon Redshift 함수 이름과 충돌을 일으키기 쉽습니다. 이번 주제에서는 오버로딩에 대해서 설명하면서 충돌을 피할 수 있는 전략을 제시하겠습니다.

함수 이름 오버로드

함수는 이름과 서명으로 구분합니다. 여기에서 서명이란 입력 인수의 수와 데이터 형식을 말합니다. 서명만 다르다면 동일한 스키마의 함수 2개가 이름이 같을 수도 있습니다. 다시 말해서 함수 이름은 중복 정의, 즉 오버로딩이 가능합니다.

쿼리를 실행하면 쿼리 엔진이 입력 인수의 수와 데이터 형식을 기준으로 호출할 함수를 결정합니다. 이때 오버로딩을 사용하면 다수의 인수를 바꿔가며 CREATE FUNCTION 명령에서 허용하는 최대 수까지 함수를 시뮬레이션할 수 있습니다.

기본 제공 Amazon Redshift 함수와의 충돌 방지

접두사 f_를 사용하여 모든 UDF의 이름을 지정하는 것이 좋습니다. Amazon Redshift는 UDF에 한해 접두사 f_를 예약하기 때문에 UDF 이름에 접두사 f_,\를 사용하면 UDF 이름이 이전은 물론이고 이후로도 Amazon Redshift 내장 SQL 함수 이름과 충돌을 일으킬 일은 없습니다. 예를 들어 새로운 UDF의 이름을 f_sum으로 지정하면 Amazon Redshift SUM 함수와 충돌이 일어나지 않습니다. 마찬가지로 새로운 함수의 이름을 f_fibonacci로 지정하면 향후 버전에서 FIBONACCI라는 함수가 Amazon Redshift에 추가되더라도 충돌에 대한 걱정을 할 필요 없습니다.

UDF와 내장 함수가 서로 다른 스키마에 속하는 경우에는 함수 이름의 오버로딩 없이도 기존 Amazon Redshift 내장 SQL 함수와 동일한 이름과 서명으로 UDF를 생성할 수 있습니다. 내장 함수는 시스템 카탈로그 스키마인 pg_catalog에 존재하기 때문에 퍼블릭이나 사용자 정의 스키마처럼 다른 스키마의 함수와 동일한 이름으로 UDF를 생성할 수 있습니다. 경우에 따라 스키마 이름으로 명시적으로 규정되지 않은 함수를 호출할 수 있습니다. 이 경우 Amazon Redshift는 기본적으로 pg_catalog 스키마를 먼저 검색합니다. 따라서 기본 제공 함수는 같은 이름의 새 UDF보다 먼저 실행됩니다.

마지막에 pg_catalog를 배치하도록 검색 경로를 설정하여 이 동작을 변경할 수 있습니다. 이렇게 하면 UDF가 기본 제공 함수보다 우선하지만 실행하면 예기치 않은 결과가 발생할 수 있습니다. 따라서 예약된 접두사인 f_를 사용하는 것처럼 고유한 명명 전략을 도입하는 것이 훨씬 안정적입니다. 자세한 내용은 SET 및 search_path 섹션을 참조하세요.

Scalar SQL UDF

스칼라 SQL UDF에는 함수 호출 시 실행되어 단일 값을 반환하는 SQL SELECT 절이 포함됩니다. CREATE FUNCTION 명령에서 사용하는 파라미터는 다음과 같습니다.

- (옵션) 입력 인수. 각 인수마다 데이터 형식이 있어야 합니다.
- 반환 데이터 형식 1개
- SQL SELECT 절 1개. SELECT 절에서는 함수 정의에서 인수의 순서에 따라 $1, $2, ... 등을 이용한 입력 인수를 참조하십시오.

입력 및 반환 데이터 형식은 모든 표준 Amazon Redshift 데이터 형식일 수 있습니다.

SELECT 절에 FROM 절을 포함하지 마십시오. 대신 SQL UDF를 호출하는 SQL 문에 FROM 절을 포함하십시오.

SELECT 절은 다음 유형의 절을 포함할 수 없습니다.

- FROM

- INTO
- WHERE
- GROUP BY
- ORDER BY
- LIMIT

스칼라 SQL 함수 예

다음은 2개의 수를 비교하여 더 큰 값을 반환하는 함수의 생성 예입니다. 자세한 내용은 CREATE FUNCTION 단원을 참조하십시오.

```
create function f_sql_greater (float, float)
  returns float
stable
as $$
  select case when $1 > $2 then $1
    else $2
  end
$$ language sql;
```

다음은 새로운 f_sql_greater 함수를 호출하여 SALES 테이블에 대한 쿼리를 실행한 후 COMMISSION 또는 PRICEPAID의 20% 중에서 더 큰 값을 반환하는 쿼리입니다.

```
select f_sql_greater(commission, pricepaid*0.20) from sales;
```

Scalar Python UDF

스칼라 Python UDF에는 함수 호출 시 실행되어 단일 값을 반환하는 Python 프로그램이 포함됩니다. CREATE FUNCTION 명령에서 사용하는 파라미터는 다음과 같습니다.

- (옵션) 입력 인수. 각 인수마다 이름과 데이터 형식이 있어야 합니다.
- 반환 데이터 형식 1개
- 실행 가능한 Python 프로그램 1개

입력 및 반환 데이터 형식은 SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE 또는 TIMESTAMP일 수 있습니다. 뿐만 아니라,

Python UDF는 런타임에 제공되는 인수를 기준으로 표준 데이터 형식으로 Amazon Redshift가 자동 변환하는 ANYELEMENT의 데이터 형식을 사용할 수 있습니다. 자세한 내용은 ANYELEMENT 데이터 형식 섹션을 참조하세요.

Amazon Redshift 쿼리가 스칼라 UDF를 호출하면 런타임에 다음과 같은 단계가 실행됩니다.

1. 함수가 입력 인수를 Python 데이터 형식으로 변환합니다.

   Python 데이터 형식에 Amazon Redshift 데이터 형식 매핑은 Python UDF 데이터 형식 섹션을 참조하세요.
2. 함수가 Python 프로그램을 실행하여 변환된 입력 인수를 전달합니다.
3. Python 코드가 단일 값을 반환합니다. 반환 값의 데이터 형식은 함수 정의에서 지정한 RETURNS 데이터 형식과 일치해야 합니다.
4. 함수가 Python 반환 값을 Amazon Redshift 데이터 형식으로 변환한 후 값을 쿼리로 반환합니다.

> **Note**
>
> Python 3는 Python UDF에는 사용할 수 없습니다. Amazon Redshift UDF에 대한 Python 3 지원을 얻으려면 Scalar Lambda UDF을 사용하세요.

스칼라 Python UDF 예시

다음은 2개의 수를 비교하여 더 큰 값을 반환하는 함수의 생성 예입니다. 예를 보면 2개의 달러 기호($$) 사이에 있는 코드가 들여쓰기되어 있는데, 이것이 Python의 요건입니다. 자세한 내용은 CREATE FUNCTION 단원을 참조하십시오.

```
create function f_py_greater (a float, b float)
  returns float
stable
as $$
  if a > b:
    return a
  return b
$$ language plpythonu;
```

다음은 새로운 `f_greater` 함수를 호출하여 SALES 테이블에 대한 쿼리를 실행한 후 COMMISSION 또는 PRICEPAID의 20% 중에서 더 큰 값을 반환하는 쿼리입니다.

```
select f_py_greater (commission, pricepaid*0.20) from sales;
```

Python UDF 데이터 형식

Python UDF는 입력 인수와 함수의 반환 값으로 표준 Amazon Redshift 데이터 형식을 모두 사용할 수 있습니다. 표준 데이터 형식 외에도 ANYELEMENT 데이터 형식을 지원합니다. Amazon Redshift는 이 데이터 형식을 런타임에 입력되는 인수에 따라 표준 데이터 형식으로 자동 변환합니다. 스칼라 UDF는 ANYELEMENT 데이터 형식을 반환할 수 있습니다. 자세한 내용은 ANYELEMENT 데이터 형식 단원을 참조하십시오.

실행 중 Amazon Redshift는 처리를 위해 Amazon Redshift 데이터 형식의 인수를 Python 데이터 형식으로 변환합니다. 그런 다음 Python 데이터 형식의 반환 값을 해당 Amazon Redshift 데이터 형식으로 변환합니다. Amazon Redshift 데이터 형식에 대한 자세한 내용은 데이터 타입 섹션을 참조하세요.

다음 표는 Amazon Redshift 데이터 형식을 Python 데이터 형식으로 매핑한 것입니다.

Amazon Redshift 데이터 형식	Python 데이터 형식
smallint 정수 bigint bigint long	int
decimal 또는 numeric	decimal
double real	float
boolean	bool
char varchar	문자열
타임스탬프	datetime

ANYELEMENT 데이터 형식

ANYELEMENT는 다형성 데이터 형식입니다. 이 말은 함수가 인수의 데이터 형식으로 ANYELEMENT를 사용하도록 선언되면 함수를 호출하여 해당 인수를 입력할 때 모든 표준 Amazon Redshift 데이터 형식이 허용된다는 것을 의미합니다. ANYELEMENT 인수는 함수 호출 시 실제로 전달되는 데이터 형식으로 설정됩니다.

함수가 ANYELEMENT 데이터 형식을 다수 사용하면 함수 호출 시 모두 동일한 실제 데이터 형식으로 확인되어야 합니다. ANYELEMENT 인수 데이터 형식은 모두 ANYELEMENT로 전달되는 첫 번째 인수의 실제 데이터 형식으로 설정됩니다. 예를 들어 `f_equal(anyelement, anyelement)`로 선언된 함수는 데이터 형식이 동일할 경우에 한해 입력 값을 2개 갖습니다.

함수의 반환 값이 ANYELEMENT로 선언되면 입력 인수는 1개 이상이 ANYELEMENT이어야 합니다. 반환 값의 실제 데이터 형식은 ANYELEMENT 입력 인수로 입력되는 실제 데이터 형식과 동일합니다.

UDF에 대한 Python 언어 지원

Python 프로그래밍 언어를 기반으로 사용자 지정 UDF를 생성할 수 있습니다. 다음 모듈을 제외하고, UDF에서 Python 2.7 Standard Library를 사용할 수 있습니다.

- ScrolledText
- Tix
- Tkinter
- tk
- turtle
- smtpd

다음 모듈은 Python Standard Library 외에 Amazon Redshift에도 구현되어 있습니다.

- numpy 1.8.2
- pandas 0.14.1
- python-dateutil 2.2
- pytz 2014.7
- scipy 0.12.1
- six 1.3.0

- wsgiref 0.1.2

또한 사용자 지정 Python 모듈을 가져와서 UDF에서 CREATE LIBRARY 명령을 실행하는 데 사용할 수 있습니다. 자세한 내용은 예제: 사용자 지정 Python 라이브러리 모듈 가져오기 단원을 참조하십시오.

> **Important**
>
> Amazon Redshift는 UDF를 통해 파일 시스템에 접근하려는 모든 네트워크 액세스와 쓰기 액세스를 차단합니다.

> **Note**
>
> Python 3는 Python UDF에는 사용할 수 없습니다. Amazon Redshift UDF에 대한 Python 3 지원을 얻으려면 Scalar Lambda UDF을 사용하세요.

예제: 사용자 지정 Python 라이브러리 모듈 가져오기

스칼라 함수는 Python 언어 구문으로 정의됩니다. Python 표준 라이브러리 모듈과 Amazon Redshift 사전 설치된 모듈을 사용할 수 있습니다. 또한 사용자 정의 Python 라이브러리 모듈을 생성하고 라이브러리를 클러스터로 가져오거나 Python 또는 서드 파티의 기존 라이브러리를 사용할 수도 있습니다.

Python Standard Library 모듈 또는 Amazon Redshift에 사전 설치되어 있는 Python 모듈과 동일한 이름의 모듈이 포함되어 있는 라이브러리는 생성할 수 없습니다. 기존에 사용자가 설치한 라이브러리가 생성할 라이브러리와 동일한 Python 패키지를 사용하는 경우에는 기존 라이브러리를 삭제한 후 새로운 라이브러리를 설치해야 합니다.

사용자 지정 라이브러리를 설치하려면 수퍼유저이거나 `USAGE ON LANGUAGE plpythonu` 권한을 가지고 있어야 합니다. 하지만 함수를 생성할 수 있는 권한을 가진 사용자라면 누구나 설치된 라이브러리를 사용할 수 있습니다. 클러스터에 설치된 라이브러리 정보는 PG_LIBRARY 시스템 카탈로그에 대한 쿼리를 실행하여 확인할 수 있습니다.

사용자 지정 Python 모듈을 클러스터로 가져오기

이번 단원에서는 사용자 지정 Python 모듈을 클러스터로 가져오는 예에 대해서 살펴봅니다. 이번 섹션에서 언급하는 단계를 따르려면 라이브러리 패키지를 업로드할 Amazon S3 버킷이 필요합니다. 업

로드된 패키지는 클러스터에 설치됩니다. 버킷 생성에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 버킷 생성을 참조하세요.

이번 예에서는 위치 및 거리 데이터를 가지고 작업할 UDF를 생성한다고 가정합니다. 먼저 SQL 클라이언트 도구에서 Amazon Redshift 클러스터에 연결한 후 다음 명령을 실행하여 함수를 생성합니다.

```
CREATE FUNCTION f_distance (x1 float, y1 float, x2 float, y2 float) RETURNS float
 IMMUTABLE as $$
    def distance(x1, y1, x2, y2):
        import math
        return math.sqrt((y2 - y1) ** 2 + (x2 - x1) ** 2)

    return distance(x1, y1, x2, y2)
$$ LANGUAGE plpythonu;

CREATE FUNCTION f_within_range (x1 float, y1 float, x2 float, y2 float) RETURNS bool
 IMMUTABLE as $$
    def distance(x1, y1, x2, y2):
        import math
        return math.sqrt((y2 - y1) ** 2 + (x2 - x1) ** 2)

    return distance(x1, y1, x2, y2) < 20
$$ LANGUAGE plpythonu;
```

위 예를 보면 일부 코드 라인이 중복되어 있습니다. UDF는 다른 UDF의 내용을 참조할 수 없을 뿐만 아니라 두 함수 모두 동일한 기능을 요구하기 때문에 이러한 중복은 필요합니다. 하지만 여러 함수에서 코드를 중복시킬 필요 없이 사용자 지정 라이브러리를 생성한 후 두 함수에서 사용하도록 구성하는 방법도 있습니다.

이를 위해서는 먼저 다음 단계에 따라 라이브러리 패키지를 생성해야 합니다.

1. geometry라는 이름의 폴더를 생성합니다. 이 폴더는 라이브러리의 최상위 패키지입니다.
2. geometry 폴더에서 `__init__.py`라는 이름의 파일을 생성합니다. 파일 이름을 보면 이중 밑줄 문자가 2개 포함되어 있습니다. 이 파일은 패키지가 초기화할 수 있다는 것을 Python에게 알려주는 역할을 합니다.
3. 다시 한 번 geometry 폴더에서 trig라는 이름의 폴더를 생성합니다. 이 폴더는 라이브러리의 하위 패키지입니다.
4. trig 폴더에서 `__init__.py`라는 이름의 파일을 한 번 더, 그리고 `line.py`라는 이름의 파일을 생성합니다. 이 폴더에서 `__init__.py` 파일은 하위 패키지가 초기화될 수 있다는 것을 Python에게 알려주는 역할을 하며, `line.py`는 라이브러리 코드가 저장되는 파일입니다.

폴더 및 파일 구조는 다음과 같아야 합니다.

```
geometry/
   __init__.py
   trig/
      __init__.py
      line.py
```

패키지 구조에 대한 자세한 내용은 Python 웹사이트의 Python 튜토리얼에서 모듈 섹션을 참조하세요.

5. 다음은 라이브러리의 클래스 및 멤버 함수가 포함된 코드입니다. 이 코드를 복사해서 `line.py` 파일에 붙여넣습니다.

```
class LineSegment:
  def __init__(self, x1, y1, x2, y2):
    self.x1 = x1
    self.y1 = y1
    self.x2 = x2
    self.y2 = y2
  def angle(self):
    import math
    return math.atan2(self.y2 - self.y1, self.x2 - self.x1)
  def distance(self):
    import math
    return math.sqrt((self.y2 - self.y1) ** 2 + (self.x2 - self.x1) ** 2)
```

패키지 생성을 마쳤으면 이제 다음 단계에 따라 패키지를 준비하여 Amazon S3에 업로드합니다.

1. geometry 폴더의 내용을 geometry.zip이라는 이름의 .zip 파일로 압축합니다. 이때 geometry 폴더 자체는 제외하고 다음과 같이 폴더의 내용만 압축 파일에 포함되어야 합니다.

```
geometry.zip
   __init__.py
   trig/
      __init__.py
      line.py
```

2. geometry.zip을 Amazon S3 버킷에 업로드합니다.

> **Important**
>
> Amazon S3 버킷이 Amazon Redshift 클러스터와 동일한 리전에 속하지 않는 경우에는 REGION 옵션을 사용하여 데이터가 위치한 리전을 지정해야 합니다. 자세한 내용은 CREATE LIBRARY 단원을 참조하십시오.

3. SQL 클라이언트 도구에서 다음 명령을 실행하여 라이브러리를 설치합니다. *<bucket_name>*은 버킷 이름으로, *<access key id>*와 *<secret key>*는 각각 AWS Identity and Access Management(IAM) 사용자 자격 증명의 액세스 키와 비밀 액세스 키로 바꿉니다.

```
CREATE LIBRARY geometry LANGUAGE plpythonu FROM 's3://<bucket_name>/geometry.zip'
 CREDENTIALS 'aws_access_key_id=<access key id>;aws_secret_access_key=<secret key>';
```

라이브러리를 클러스터에 설치한 후에는 라이브러리를 사용하도록 함수를 구성해야 합니다. 이를 위해 다음 명령을 실행합니다.

```
CREATE OR REPLACE FUNCTION f_distance (x1 float, y1 float, x2 float, y2 float) RETURNS
 float IMMUTABLE as $$
    from trig.line import LineSegment

    return LineSegment(x1, y1, x2, y2).distance()
$$ LANGUAGE plpythonu;

CREATE OR REPLACE FUNCTION f_within_range (x1 float, y1 float, x2 float, y2 float)
 RETURNS bool IMMUTABLE as $$
    from trig.line import LineSegment

    return LineSegment(x1, y1, x2, y2).distance() < 20
$$ LANGUAGE plpythonu;
```

위의 명령에서 `import trig/line`으로 이번 단원의 첫 번째 함수에서 보였던 중복 코드가 제거되었습니다. 이 라이브러리에서 제공하는 기능은 여러 UDF에서 재사용할 수 있습니다. 모듈을 가져오려면 하위 패키지 경로와 모듈 이름(`trig/line`)만 지정하면 됩니다.

Python UDF 제한 사항

이번 주제에서 다루는 제약 조건 이내에서 Amazon Redshift 내장 스칼라 함수를 사용하는 곳이라면 어디에서든지 UDF를 사용할 수 있습니다. 자세한 내용은 SQL 함수 참조 단원을 참조하십시오.

Amazon Redshift Python UDF의 제약 조건은 다음과 같습니다.

- Python UDF는 네트워크에 액세스하거나, 혹은 파일 시스템 읽기 또는 쓰기를 할 수 없습니다.
- 사용자가 설치한 Python 라이브러리의 전체 크기가 100MB를 초과할 수 없습니다.
- Amazon Redshift는 자동 워크로드 관리(WLM)를 사용하는 프로비저닝된 클러스터와 서버리스 작업 그룹에 대해 한 번에 하나의 Python UDF만 실행할 수 있습니다. 둘 이상의 UDF를 동시에 실행하려고 하면 Amazon Redshift는 워크로드 관리 대기열에서 실행되도록 나머지 Python UDF를 대기열에 대기시킵니다. 자동 WLM을 사용할 때 SQL UDF에는 동시성 제한이 없습니다.
- 프로비저닝된 클러스터에 수동 WLM을 사용하는 경우 클러스터당 동시에 실행할 수 있는 Python UDF의 수는 클러스터 전체 동시성 수준의 4분의 1로 제한됩니다. 예를 들어, 동시성이 15인 프로비저닝된 클러스터는 최대 3개의 Python UDF를 동시에 실행할 수 있습니다.
- Python UDF를 사용할 때 Amazon Redshift는 SUPER 및 HLLSKETCH 데이터 유형을 지원하지 않습니다.

Python UDF에서 오류 및 경고 로깅

UDF에서 Python 로깅 모듈을 사용하여 사용자 정의 오류 및 경고 메시지를 생성할 수 있습니다. 쿼리를 실행한 후에는 SVL_UDF_LOG 시스템 뷰에 대한 쿼리를 실행하여 로그 메시지를 가져올 수도 있습니다.

> **Note**
>
> UDF 로깅은 클러스터 리소스를 소비하기 때문에 시스템 성능에 영향을 미칠 수도 있습니다. 따라서 로깅은 개발 및 문제 해결 목적으로만 실행하는 것이 좋습니다.

쿼리를 실행하면 로그 핸들러가 해당하는 함수 이름, 노드 및 조각과 함께 메시지를 SVL_UDF_LOG 시스템 뷰에 작성합니다. 각 조각마다 메시지별로 SVL_UDF_LOG에 작성되는 행은 1개입니다. 메시지는 4096바이트로 잘립니다. UDF 로그는 조각당 500개 행으로 제한됩니다. 그래서 로그가 가득 차면 로그 핸들러가 더욱 오래된 메시지를 삭제하고 경고 메시지를 SVL_UDF_LOG에 추가합니다.

> **Note**
>
> Amazon Redshift UDF 로그 핸들러는 줄 바꿈(\n), 파이프(|) 문자 및 백슬래시(\) 문자를 백슬래시(\) 문자로 이스케이프 처리합니다.

UDF 로그 레벨은 기본적으로 WARNING으로 설정됩니다. 로그 레벨이 WARNING, ERROR 및 CRITICAL인 메시지가 기록됩니다. 심각도가 INFO, DEBUG 및 NOTSET로 낮은 메시지는 무시됩니다. UDF 로그 레벨을 설정하려면 Python logger 메서드를 사용하십시오. 예를 들어 다음과 같은 메서드는 로그 레벨을 INFO로 설정됩니다.

```
logger.setLevel(logging.INFO)
```

Python 로깅 모듈의 사용에 대한 자세한 내용은 Python 설명서에서 Logging facility for Python 섹션을 참조하세요.

다음은 f_pyerror라는 이름의 함수를 생성한 다음 Python 로깅 모듈을 가져와서 로거를 인스턴스화하고 오류를 기록하는 예입니다.

```
CREATE OR REPLACE FUNCTION f_pyerror()
RETURNS INTEGER
VOLATILE AS
$$
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)
logger.info('Your info message here')
return 0
$$ language plpythonu;
```

다음은 SVL_UDF_LOG에 대한 쿼리를 실행하여 이전 예에서 기록된 메시지를 확인하는 예입니다.

```
select funcname, node, slice, trim(message) as message
from svl_udf_log;

  funcname  | query | node | slice |   message
------------+-------+------+-------+-----------------
  f_pyerror | 12345 |     1|     1 | Your info message here
```

Scalar Lambda UDF

Amazon Redshift는 AWS Lambda에 정의된 사용자 정의 함수를 SQL 쿼리의 일부로 사용할 수 있습니다. Java, Go, PowerShell, Node.js, C#, Python 및 Ruby와 같이 Lambda에서 지원하는 모든 프로그래밍 언어로 스칼라 Lambda UDF를 작성할 수 있습니다. 또는 사용자 정의 런타임을 사용할 수 있습니다.

CREATE EXTERNAL FUNCTION 명령은 다음 파라미터를 생성합니다.

- (옵션) 데이터 형식이 있는 인수 목록입니다.
- 반환 데이터 형식 1개
- Amazon Redshift에서 호출하는 외부 함수의 한 함수 이름입니다.
- Amazon Redshift 클러스터에서 Lambda를 수임하고 호출할 권한이 있는 한 IAM 역할입니다.
- Lambda UDF가 호출하는 한 Lambda 함수 이름입니다.

CREATE EXTERNAL FUNCTION에 대한 자세한 내용은 CREATE EXTERNAL FUNCTION 섹션을 참조하세요.

이 함수의 입력 및 반환 데이터 형식은 모든 표준 Amazon Redshift 데이터 형식일 수 있습니다.

Amazon Redshift는 외부 함수가 일괄 처리된 인수와 결과를 보내고 받을 수 있도록 합니다.

Lambda UDF는 Lambda에서 정의 및 관리되며 Amazon Redshift에서 이러한 UDF를 호출하기 위한 액세스 권한을 제어할 수 있습니다. 동일한 쿼리에서 여러 Lambda 함수를 호출하거나 동일한 함수를 여러 번 호출할 수 있습니다.

스칼라 함수가 지원되는 SQL 문의 절에서 Lambda UDF를 사용합니다. SELECT, UPDATE, INSERT 또는 DELETE와 같은 모든 SQL 문에서 Lambda UDF를 사용할 수도 있습니다.

> **Note**
>
> Lambda UDF를 사용하면 Lambda 서비스에서 추가 요금이 발생할 수 있습니다. 발생 여부는 Lambda 요청 수(UDF 호출) 및 Lambda 프로그램 실행의 총 기간과 같은 요인에 따라 다릅니다. 그러나 Amazon Redshift에서 Lambda UDF를 사용하기 위한 추가 요금은 없습니다. AWS Lambda 요금에 대한 자세한 내용은 AWS Lambda 요금을 참조하세요.
>
> Lambda 요청 수는 Lambda UDF가 사용되는 특정 SQL 문 절에 따라 다릅니다. 예를 들어 함수가 다음과 같은 WHERE 절에서 사용된다고 가정합니다.
>
> ```
> SELECT a, b FROM t1 WHERE lambda_multiply(a, b) = 64; SELECT a, b
> FROM t1 WHERE a*b = lambda_multiply(2, 32)
> ```
>
> 이 경우 Amazon Redshift는 각각에 대해 첫 번째 SELECT 문을 호출하고 두 번째 SELECT 문을 한 번만 호출합니다.
>
> 그러나 쿼리의 프로젝션 부분에서 UDF를 사용하면 결과 집합의 모든 정규화된 또는 집계된 행에 대해 한 번만 Lambda 함수를 호출할 수 있습니다.

UDF 보안 및 권한

Lambda UDF를 생성하려면 LANGUAGE EXFUNC에 대한 사용 권한이 있는지 확인합니다. 특정 사용자, 그룹 또는 퍼블릭에 명시적으로 USAGE ON LANGUAGE EXFUNC를 부여하거나 USAGE ON LANGUAGE EXFUNC를 취소해야 합니다.

다음 예는 EXFUNC에 대한 사용 권한을 PUBLIC에 부여합니다.

```
grant usage on language exfunc to PUBLIC;
```

다음 예는 PUBLIC에서의 exfunc 사용 권한을 취소한 후 사용자 그룹 lambda_udf_devs에 사용을 부여합니다.

```
revoke usage on language exfunc from PUBLIC;
grant usage on language exfunc to group lambda_udf_devs;
```

Lambda UDF를 실행하려면 호출된 각 함수에 대한 권한이 있는지 확인합니다. 기본적으로 새로운 Lambda UDF를 실행하는 권한은 PUBLIC에 허용됩니다. 사용을 제한하려면 함수에 대해 PUBLIC에서 이 권한을 취소합니다. 그런 다음 특정 사용자나 그룹에 권한을 허용합니다.

다음 예는 PUBLIC의 exfunc_sum 함수 실행 권한을 취소합니다. 그런 다음 사용자 그룹 lambda_udf_devs에 사용 권한을 부여합니다.

```
revoke execute on function exfunc_sum(int, int) from PUBLIC;
grant execute on function exfunc_sum(int, int) to group lambda_udf_devs;
```

기본적으로 수퍼유저는 모든 권한을 갖습니다.

권한 부여 및 취소에 대한 자세한 내용은 GRANT 및 REVOKE 섹션을 참조하세요.

Lambda UDF에 대한 권한 부여 파라미터 구성

CREATE EXTERNAL FUNCTION 명령은 AWS Lambda에서 Lambda 함수를 호출할 수 있는 권한이 필요합니다. 권한 부여를 시작하려면 CREATE EXTERNAL FUNCTION 명령을 실행할 때 AWS Identity and Access Management(IAM) 역할을 지정합니다. IAM 역할에 대한 자세한 내용은 IAM 사용 설명서의 IAM 역할 단원을 참조하세요.

클러스터에 연결된 Lambda 함수를 호출할 수 있는 권한이 있는 기존 IAM 역할이 있는 경우 명령에 대한 IAM_ROLE 파라미터에서 역할 Amazon 리소스 이름(ARN)을 대체할 수 있습니다. 다음 섹션에서는 CREATE EXTERNAL FUNCTION 명령에서 IAM 역할을 사용하는 단계를 설명합니다.

Lambda용 IAM 역할 생성

IAM 역할에는 Lambda 함수를 호출할 수 있는 권한이 필요합니다. IAM 역할을 생성하는 동안 다음 방법 중 하나로 권한을 제공합니다.

- IAM 역할을 생성하는 동안 [권한 정책 연결(Attach permissions policy)] 페이지에서 `AWSLambdaRole` 정책을 연결합니다. `AWSLambdaRole` 정책은 최소 요구 사항인 Lambda 함수를 호출할 수 있는 권한을 부여합니다. 자세한 내용과 기타 정책은 AWS Lambda 개발자 안내서의 AWS Lambda에 대한 자격 증명 기반 IAM 정책 섹션을 참조하세요.
- 모든 리소스 또는 해당 함수의 ARN이 있는 특정 Lambda 함수의 `lambda:InvokeFunction` 권한으로 IAM 역할에 연결할 사용자 정의 정책을 생성합니다. 정책을 생성하는 방법에 대한 자세한 내용은 IAM User Guide의 Creating IAM policies 섹션을 참조하세요.

  다음 예제 정책은 특정 Lambda 함수에서 Lambda 호출을 사용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Invoke",
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:my-function"
        }
    ]
}
```

  Lambda 함수의 리소스에 대한 자세한 내용은 IAM API Reference의 Resources and conditions for Lambda actions 섹션을 참조하세요.

  필요한 권한으로 사용자 지정 정책을 생성한 후 IAM 역할을 생성하는 동안 [권한 정책 연결(Attach permissions policy)] 페이지에서 IAM 역할에 정책을 연결할 수 있습니다.

IAM 역할을 생성하는 단계는 Amazon Redshift 관리 가이드의 사용자를 대신하여 기타 AWS 서비스에 액세스하도록 Amazon Redshift에 권한 부여 섹션을 참조하세요.

새 IAM 역할을 생성하지 않으려면 앞에서 언급한 권한을 기존 IAM 역할에 추가할 수 있습니다.

IAM 역할을 클러스터와 연결

클러스터에 IAM 역할 연결 Amazon Redshift 관리 콘솔, CLI 또는 API를 사용하여 역할을 클러스터에 추가하거나, 클러스터와 연결된 역할을 확인할 수 있습니다. 자세한 내용은 Amazon Redshift 관리 가이드의 클러스터와 IAM 역할 연결 섹션을 참조하세요.

명령에 IAM 역할 포함

CREATE EXTERNAL FUNCTION 명령에 IAM 역할 ARN을 포함합니다. IAM 역할을 생성하면 IAM이 생성된 역할의 Amazon 리소스 이름(ARN)을 반환합니다. IAM 역할을 지정하려면 역할 ARN에 `IAM_ROLE` 파라미터를 제공합니다. 다음은 `IAM_ROLE` 파라미터에 대한 구문을 보여줍니다.

```
IAM_ROLE 'arn:aws:iam::aws-account-id:role/role-name'
```

동일한 리전 내의 다른 계정에 있는 Lambda 함수를 호출하려면 Amazon Redshift에서 IAM 역할 연결을 참조하세요.

Amazon Redshift와 AWS Lambda 간의 JSON 인터페이스 사용

Amazon Redshift는 통신하는 모든 Lambda 함수에 대해 공통 인터페이스를 사용합니다.

다음 표는 JSON 페이로드에 대해 예상할 수 있는 지정된 Lambda 함수의 입력 필드 목록을 보여줍니다.

필드 이름	설명	값 범위
request_id	각 호출 요청을 고유하게 식별하는 UUID(Universally Unique Identifier)입니다.	유효한 UUID입니다.
cluster	클러스터의 전체 Amazon 리소스 이름(ARN)입니다.	유효한 클러스터 ARN입니다.
사용자	호출하는 사용자의 이름입니다.	유효한 사용자 이름입니다.
데이터베이스	쿼리가 실행 중인 데이터베이스의 이름입니다.	유효한 데이터베이스 이름입니다.

필드 이름	설명	값 범위
external_function	호출하는 외부 함수의 정규화된 이름입니다.	유효한 정규화된 함수 이름입니다.
query_id	호출하는 쿼리의 쿼리 ID입니다.	유효한 쿼리 ID입니다.
num_records	페이로드의 인수 수입니다.	값 1 - 2^64입니다.
인수	지정된 형식의 데이터 페이로드입니다.	배열 형식의 데이터는 JSON 배열이어야 합니다. 인수의 수가 1보다 크면 각 요소는 배열인 레코드입니다. Amazon Redshift는 배열을 사용하여 페이로드의 레코드 순서를 유지합니다.

JSON 배열의 순서에 따라 일괄 처리 순서가 결정됩니다. Lambda 함수는 인수를 반복적으로 처리하고 정확한 수의 레코드를 생성해야 합니다. 다음은 페이로드의 예입니다.

```
{
  "request_id" : "23FF1F97-F28A-44AA-AB67-266ED976BF40",
  "cluster" : "arn:aws:redshift:xxxx",
  "user" : "adminuser",
  "database" : "db1",
  "external_function": "public.foo",
  "query_id" : 5678234,
  "num_records" : 4,
  "arguments" : [
     [ 1, 2 ],
     [ 3, null],
     null,
     [ 4, 6]
   ]
 }
```

Lambda 함수의 반환 출력에는 다음 필드가 포함됩니다.

필드 이름	설명	값 범위
success	함수의 성공 또는 실패 표시입니다.	값 "true" 또는 "false"입니다.
error_msg	성공 값이 "false"이면 오류 메시지(함수가 실패한 경우)입니다. 그렇지 않으면 이 필드는 무시됩니다.	유효한 메시지입니다.
num_records	페이로드의 레코드 수입니다.	값 1 - 2^64입니다.
결과	지정된 형식의 호출 결과입니다.	N/A

다음은 Lambda 함수 출력의 예입니다.

```
{
  "success": true,   // true indicates the call succeeded
  "error_msg" : "my function isn't working",  // shall only exist when success != true
  "num_records": 4,      // number of records in this payload
  "results" : [
     1,
     4,
     null,
     7
   ]
}
```

SQL 쿼리에서 Lambda 함수를 호출할 때 Amazon Redshift는 다음 사항을 고려하여 연결 보안을 보장합니다.

- GRANT 및 REVOKE 권한. UDF 보안 및 권한에 관한 자세한 내용은 UDF 보안 및 권한를 참조하시기 바랍니다.
- Amazon Redshift는 지정된 Lambda 함수에 최소 데이터 집합만 제출합니다.
- Amazon Redshift는 지정된 IAM 역할을 가진 지정된 Lambda 함수만 호출합니다.

사용자 정의 함수(UDF)의 사용 사례 예제

사용자 정의 함수를 사용하면 Amazon Redshift를 다른 구성 요소와 통합하여 비즈니스 문제를 해결할 수 있습니다. 다음은 다른 사용자가 사용 사례에 UDF를 어떻게 사용하는지 보여주는 몇 가지 예입니다.

- Amazon Redshift Lambda UDF를 사용하여 외부 구성 요소 액세스 - Amazon Redshift Lambda UDF가 작동하는 방식을 설명하고 Lambda UDF를 생성하는 방법을 안내합니다.
- Amazon Redshift, Amazon Translate 및 Amazon Comprehend에서 SQL 함수를 사용하여 텍스트 Amazon 번역 및 분석 - 몇 번의 클릭만으로 텍스트 필드를 번역, 편집 및 분석할 수 있는 사전 구축된 Amazon Redshift Lambda UDF를 제공합니다.
- Amazon Redshift에서 Amazon Location Service 액세스 - Amazon Redshift Lambda UDF를 사용하여 Amazon Location Service와 통합하는 방법을 설명합니다.
- Amazon Redshift 및 Protegrity를 사용한 데이터 토큰화 - Amazon Redshift Lambda UDF를 Protegrity 서버리스 제품과 통합하는 방법을 설명합니다.
- Amazon Redshift UDF - Amazon Redshift SQL, Lambda 및 Python UDF의 모음입니다.

Amazon Redshift에서 저장 프로시저 생성

이 주제에서는 Amazon Redshift에서 저장 프로시저를 생성하고 사용하는 방법을 설명합니다. 저장 프로시저는 여러 프로그램에서 사용할 수 있는 SQL 문 모음입니다.

PostgreSQL 프로시저 언어 PL/pgSQL을 사용하여 Amazon Redshift 저장 프로시저를 정의함으로써 SQL 쿼리 및 논리 연산을 수행할 수 있습니다. 프로시저는 데이터베이스에 저장되고 충분한 데이터베이스 권한이 있는 모든 사용자가 사용할 수 있습니다.

사용자 정의 함수(UDF)와 달리, 저장 프로시저는 SELECT 쿼리 외에도 데이터 정의 언어(DDL) 및 데이터 조작 언어(DML)를 통합할 수 있습니다. 저장 프로시저는 값을 반환할 필요가 없습니다. 루프 및 조건부 표현식을 포함한 프로시저 언어를 사용하여 논리 흐름을 제어할 수 있습니다.

저장 프로시저를 생성 및 관리하기 위한 SQL 명령에 대한 자세한 내용은 다음 명령 주제를 참조하십시오.

- CREATE PROCEDURE
- ALTER PROCEDURE
- DROP PROCEDURE
- SHOW PROCEDURE
- CALL
- GRANT
- REVOKE
- ALTER DEFAULT PRIVILEGES

주제

- Amazon Redshift의 저장 프로시저 개요
- PL/pgSQL 언어 참조

Amazon Redshift의 저장 프로시저 개요

이 주제에서는 저장 프로시저의 용도와 사용에 대해 자세히 설명합니다.

저장 프로시저는 일반적으로 데이터 변환 로직, 데이터 검증 및 비즈니스별 로직을 캡슐화하는 데 사용됩니다. 여러 SQL 단계를 저장 프로시저로 결합함으로써 애플리케이션과 데이터베이스 간 왕복 횟수를 줄일 수 있습니다.

세분화된 액세스 제어를 통해 사용자에게 기본 테이블에 대한 액세스 권한을 부여하지 않고도 함수를 수행하도록 저장 프로시저를 생성할 수 있습니다. 예를 들어 소유자 또는 수퍼유저만 테이블을 자를 수 있고, 사용자가 데이터를 테이블에 삽입하려면 쓰기 권한이 필요합니다. 사용자에게 기본 테이블에 대한 권한을 부여하는 대신에, 작업을 수행하는 저장 프로시저를 생성할 수 있습니다. 그런 다음 사용자에게 저장 프로시저를 실행할 권한을 부여합니다.

DEFINER 보안 속성이 있는 저장 프로시저는 저장 프로시저의 소유자 권한으로 실행됩니다. 기본적으로 저장 프로시저에는 INVOKER 보안이 있습니다. 이는 프로시저가 프로시저를 호출하는 사용자의 권한을 사용함을 뜻합니다.

저장 프로시저를 생성하려면 CREATE PROCEDURE 명령을 사용합니다. 프로시저를 실행하려면 CALL 명령을 사용합니다. 이 단원의 뒷부분에 예제가 나와 있습니다.

> **Note**
>
> 일부 클라이언트는 Amazon Redshift 저장 프로시저를 생성할 때 다음 오류를 표시할 수 있습니다.
>
> ```
> ERROR: 42601: [Amazon](500310) unterminated dollar-quoted string at or near "$$
> ```
>
> 이 오류는 클라이언트가 문을 구분하는 세미콜론과 달러 기호($) 인용으로 CREATE PROCEDURE 문을 올바르게 구문 분석할 수 없어 발생합니다. 그 결과 문의 일부만 Amazon Redshift 서버로 전송됩니다. 보통 클라이언트의 `Run as batch` 또는 `Execute selected` 옵션을 사용하여 이 오류를 해결할 수 있습니다.
> 예를 들어 Aginity 클라이언트를 사용하는 경우 `Run entire script as batch` 옵션을 사용합니다. SQL Workbench/J를 사용하는 경우 버전 124를 사용하는 것이 좋습니다. SQL Workbench/J 버전 125를 사용하는 경우 차선책으로서 대체 구분 기호를 지정하는 것을 고려합니다.
> CREATE PROCEDURE에는 세미콜론(;)으로 구분된 SQL 문이 포함되어 있습니다. 슬래시(/) 등의 대체 구분 기호를 정의하고 CREATE PROCEDURE 문의 끝에 이를 배치하면 처리를 위해 문이 Amazon Redshift 서버에 전송됩니다. 다음은 한 예입니다.
>
> ```
> CREATE OR REPLACE PROCEDURE test()
> AS $$
> BEGIN
>   SELECT 1 a;
> END;
> $$
> LANGUAGE plpgsql
> ```

```
;
/
```

자세한 내용은 SQL Workbench/J 설명서의 Alternate delimiter를 참조하십시오. 또는 Amazon Redshift 콘솔의 쿼리 편집기 또는 TablePlus와 같이 CREATE PROCEDURE 문의 구문 분석을 더 잘 지원하는 클라이언트를 사용합니다.

주제

- 저장 프로시저 명명
- 저장 프로시저의 보안 및 권한
- 저장 프로시저에서 결과 세트 반환
- 트랜잭션 관리
- 오류 트래핑
- 저장 프로시저 로깅
- 저장 프로시저 제한 사항

다음 예제에서는 출력 인수가 없는 프로시저를 보여 줍니다. 기본적으로 인수는 입력(IN) 인수입니다.

```
CREATE OR REPLACE PROCEDURE test_sp1(f1 int, f2 varchar)
AS $$
BEGIN
  RAISE INFO 'f1 = %, f2 = %', f1, f2;
END;
$$ LANGUAGE plpgsql;

call test_sp1(5, 'abc');
INFO: f1 = 5, f2 = abc
CALL
```

Note

저장 프로시저를 작성할 때는 민감한 값을 보호하기 위한 모범 사례를 참조하는 것이 좋습니다.
민감한 정보를 저장 프로시저 로직 내에 하드 코딩하지 마세요. 예를 들어, 저장 프로시저 본문의 CREATE USER 문에 사용자 암호를 할당하지 마십시오. 하드 코딩된 값이 카탈로그 테이블

에 스키마 메타 데이터로 기록될 수 있기 때문에 보안 위험이 따릅니다. 암호와 같은 민감한 값은 파라미터를 사용하여 저장 프로시저에 인수로 전달하십시오.
저장 프로시저에 대한 자세한 내용은 CREATE PROCEDURE 및 Amazon Redshift에서 저장 프로시저 생성 섹션을 참조하세요. 카탈로그 테이블에 대한 자세한 내용은 시스템 카탈로그 테이블 섹션을 참조하세요.

다음 예제에서는 출력 인수가 있는 프로시저를 보여 줍니다. 인수는 입력(IN), 입력 및 출력(INOUT), 출력(OUT)입니다.

```
CREATE OR REPLACE PROCEDURE test_sp2(f1 IN int, f2 INOUT varchar(256), out_var OUT
 varchar(256))
AS $$
DECLARE
  loop_var int;
BEGIN
  IF f1 is null OR f2 is null THEN
    RAISE EXCEPTION 'input cannot be null';
  END IF;
  DROP TABLE if exists my_etl;
  CREATE TEMP TABLE my_etl(a int, b varchar);
    FOR loop_var IN 1..f1 LOOP
        insert into my_etl values (loop_var, f2);
        f2 := f2 || '+' || f2;
    END LOOP;
  SELECT INTO out_var count(*) from my_etl;
END;
$$ LANGUAGE plpgsql;


call test_sp2(2,'2019');

         f2          | column2
---------------------+---------
 2019+2019+2019+2019 | 2
(1 row)
```

저장 프로시저 명명

이 주제에서는 저장 프로시저 이름에 대한 세부 정보를 설명합니다.

이름이 동일하나 입력 인수 데이터 유형 또는 서명이 다른 프로시저를 정의하는 경우, 새 프로시저를 생성합니다. 이렇게 하면 프로시저 이름이 오버로드됩니다. 자세한 내용은 프로시저 이름 오버로드 단원을 참조하십시오. Amazon Redshift는 출력 인수를 기반으로 하는 프로시저 오버로드를 사용하지 않습니다. 이름 및 입력 인수 데이터 유형이 동일하지만 출력 인수 유형은 다른 두 프로시저를 가질 수 없습니다.

소유자 또는 수퍼유저는 저장 프로시저의 본문을 동일한 서명이 있는 새 본문으로 바꿀 수 있습니다. 저장 프로시저의 서명 또는 반환 형식을 변경하려면 저장 프로시저를 삭제한 후 다시 생성합니다. 자세한 내용은 DROP PROCEDURE 및 CREATE PROCEDURE을 참조하십시오.

저장 프로시저를 실행하기 전에 저장 프로시저의 명명 규칙을 고려하여 잠재적 충돌이나 뜻밖의 결과를 피할 수 있습니다. 프로시저 이름을 오버로드할 수 있으므로 기존 및 향후 Amazon Redshift 프로시저 이름과 충돌할 수 있습니다.

프로시저 이름 오버로드

프로시저는 이름과 서명으로 구분합니다. 여기에서 서명이란 입력 인수의 수와 인수의 데이터 형식을 말합니다. 서명만 다르다면 동일한 스키마의 프로시저 2개가 이름이 같을 수도 있습니다. 다시 말해 프로시저 이름을 오버로드할 수 있습니다.

프로시저를 실행하면 쿼리 엔진이 입력 인수의 수와 데이터 형식을 기준으로 호출할 프로시저를 결정합니다. 이때 오버로딩을 사용하면 다수의 인수를 바꿔가며 CREATE PROCEDURE 명령에서 허용하는 최대 수까지 프로시저를 시뮬레이션할 수 있습니다. 자세한 내용은 CREATE PROCEDURE 단원을 참조하십시오.

이름 충돌 방지

접두사 sp_를 사용하여 모든 절차의 이름을 지정하는 것이 좋습니다. Amazon Redshift는 저장 프로시저 전용으로 sp_ 접두사를 예약합니다. 프로시저 이름에 sp_ 접두사를 붙이면 프로시저 이름이 기존 또는 향후 Amazon Redshift 프로시저 이름과 충돌하지 않게 할 수 있습니다.

저장 프로시저의 보안 및 권한

이 주제에서는 저장 프로시저를 만들고 실행하는 데 필요한 데이터베이스 자격 증명에 대해 설명합니다.

기본적으로 모든 사용자는 프로시저를 생성할 권한이 있습니다. 프로시저를 생성하려면 언어 PL/pgSQL에 대한 USAGE 권한이 있어야 하며, 이 권한은 기본적으로 PUBLIC에 부여됩니다. 기본적으로 수퍼유저 및 소유자만 프로시저를 호출할 권한이 있습니다. 수퍼유저는 사용자가 저장 프로시저를 생성하지 못하도록 사용자의 PL/pgSQL에서 REVOKE USAGE를 실행할 수 있습니다.

프로시저를 호출하려면 프로시저에 대한 EXECUTE 권한을 부여받아야 합니다. 기본적으로 새 프로시저에 대한 EXECUTE 권한은 프로시저 소유자 및 수퍼유저에게 부여됩니다. 자세한 내용은 GRANT 단원을 참조하십시오.

기본적으로 프로시저를 생성하는 사용자가 소유자입니다. 소유자는 기본적으로 해당 프로시저에 대한 CREATE, DROP, EXECUTE 권한이 있습니다. 수퍼유저는 모든 권한을 갖습니다.

SECURITY 속성은 데이터베이스 객체에 액세스할 프로시저의 권한을 제어합니다. 저장 프로시저를 생성할 때 SECURITY 속성을 DEFINER 또는 INVOKER로 설정할 수 있습니다. SECURITY INVOKER를 지정하는 경우, 프로시저는 프로시저를 호출하는 사용자의 권한을 사용합니다. SECURITY DEFINER를 지정하는 경우, 프로시저는 프로시저 소유자의 권한을 사용합니다. INVOKER가 기본값입니다.

SECURITY DEFINER 프로시저는 이를 소유한 사용자의 권한으로 실행되므로, 프로시저가 잘못 사용되지 않도록 주의하세요. SECURITY DEFINER 프로시저가 잘못 사용되지 되지 않도록 하려면 다음을 수행합니다.

- PUBLIC이 아니라 특정 사용자에게 SECURITY DEFINER 프로시저에 대한 EXECUTE를 부여합니다.
- 프로시저가 스키마 이름을 사용하여 액세스해야 하는 모든 데이터베이스 객체를 한정합니다. 예를 들어 단순히 `mytable` 대신 `myschema.mytable`을 사용합니다.
- 객체 이름을 스키마로 한정할 수 없는 경우, SET 옵션을 사용하여 프로시저를 생성할 때 `search_path`를 설정합니다. `search_path`를 설정하여 신뢰할 수 없는 사용자가 작성할 수 있는 스키마를 제외합니다. 그러면 이 프로시저의 호출자가 프로시저에서 사용할 객체를 마스킹하는 객체(예: 테이블 또는 뷰)를 생성하지 못하게 할 수 있습니다. SET 옵션에 대한 자세한 내용은 CREATE PROCEDURE 섹션을 참조하세요.

다음 예제에서는 `search_path`를 `admin`으로 설정하여 `user_creds` 테이블이 `admin` 스키마에서 액세스되고 퍼블릭 또는 호출자의 `search_path`에 있는 다른 스키마에서 액세스하지 못하게 합니다.

```
CREATE OR REPLACE PROCEDURE sp_get_credentials(userid int, o_creds OUT varchar)
AS $$
BEGIN
  SELECT creds INTO o_creds
  FROM user_creds
  WHERE user_id = $1;
END;
$$ LANGUAGE plpgsql
SECURITY DEFINER
```

```
-- Set a secure search_path
SET search_path = admin;
```

저장 프로시저에서 결과 세트 반환

이 주제에서는 저장 프로시저가 데이터를 반환하는 방법을 설명합니다.

커서 또는 임시 테이블을 사용하여 결과 세트를 반환할 수 있습니다.

커서 반환

커서를 반환하려면 `refcursor` 데이터 형식으로 정의된 INOUT 인수를 사용하여 프로시저를 생성합니다. 프로시저를 호출할 때 커서에 이름을 지정합니다. 그러면 커서에서 이름을 기준으로 결과를 가져올 수 있습니다.

다음 예제에서는 `refcursor` 데이터 형식을 사용하여 `rs_out`라는 INOUT 인수로 `get_result_set`라는 프로시저를 생성합니다. 이 프로시저는 SELECT 문을 사용하여 커서를 엽니다.

```
CREATE OR REPLACE PROCEDURE get_result_set (param IN integer, rs_out INOUT refcursor)
AS $$
BEGIN
  OPEN rs_out FOR SELECT * FROM fact_tbl where id >= param;
END;
$$ LANGUAGE plpgsql;
```

다음 CALL 명령은 이름이 `mycursor`인 커서를 엽니다. 트랜잭션 내에서만 커서를 사용합니다.

```
BEGIN;
CALL get_result_set(1, 'mycursor');
```

커서가 열리면 다음 예제와 같이 커서에서 가져올 수 있습니다.

```
FETCH ALL FROM mycursor;

    id | secondary_id | name
-------+--------------+---------
     1 |            1 | Joe
     1 |            2 | Ed
     2 |            1 | Mary
     1 |            3 | Mike
```

```
(4 rows)
```

결국 트랜잭션이 커밋되거나 롤백됩니다.

```
COMMIT;
```

저장 프로시저에 의해 반환된 커서에는 DECLARE CURSOR에서 설명한 것과 동일한 제약 및 성능 고려 사항이 적용됩니다. 자세한 내용은 커서 제약 조건 단원을 참조하십시오.

다음 예제에서는 JDBC의 `refcursor` 데이터 형식을 사용하여 `get_result_set` 저장 프로시저의 호출을 보여 줍니다. 리터럴 `'mycursor'`(커서 이름)는 `prepareStatement`로 전달됩니다. 그런 다음 `ResultSet`에서 결과를 가져옵니다.

```
static void refcursor_example(Connection conn) throws SQLException {
    conn.setAutoCommit(false);
    PreparedStatement proc = conn.prepareStatement("CALL get_result_set(1,
 'mycursor')");
    proc.execute();
    ResultSet rs = statement.executeQuery("fetch all from mycursor");
    while (rs.next()) {
      int n = rs.getInt(1);
      System.out.println("n " + n);
    }
```

임시 테이블 사용

결과를 반환하기 위해 핸들을 결과 행이 포함된 임시 테이블로 반환할 수 있습니다. 클라이언트는 이름을 파라미터로 저장 프로시저에 제공할 수 있습니다. 저장 프로시저 내부에서 동적 SQL을 사용하여 임시 테이블에서 작업할 수 있습니다. 다음은 그 한 예입니다.

```
CREATE PROCEDURE get_result_set(param IN integer, tmp_name INOUT varchar(256)) as $$
DECLARE
  row record;
BEGIN
  EXECUTE 'drop table if exists ' || tmp_name;
  EXECUTE 'create temp table ' || tmp_name || ' as select * from fact_tbl where id <= '
 || param;
END;
$$ LANGUAGE plpgsql;

CALL get_result_set(2, 'myresult');
 tmp_name
```

```
-----------
 myresult
(1 row)

SELECT * from myresult;
 id | secondary_id | name
----+--------------+------
  1 |            1 | Joe
  2 |            1 | Mary
  1 |            2 | Ed
  1 |            3 | Mike
(4 rows)
```

트랜잭션 관리

기본 트랜잭션 관리 동작 또는 NONATOMIC 동작을 사용하여 저장 프로시저를 만들 수 있습니다.

기본 모드 저장 프로시저 트랜잭션 관리

기본 트랜잭션 모드 자동 커밋 동작은 별도로 실행되는 각각의 SQL 명령이 개별적으로 커밋되게 합니다. 저장 프로시저 호출은 단일 SQL 명령으로 처리됩니다. 프로시저 내부의 SQL 문은 호출이 시작되면 암시적으로 시작되고 호출이 끝나면 종료되는 트랜잭션 블록에 있는 것처럼 동작합니다. 다른 프로시저에 대한 중첩 호출은 다른 SQL 문처럼 취급되고 호출자와 동일한 트랜잭션의 컨텍스트 내에서 작동합니다. 자동 커밋 동작에 대한 자세한 내용은 직렬화 가능 격리 섹션을 참조하세요.

그러나 사용자 지정 트랜잭션 블록(BEGIN...COMMIT에 의해 정의됨) 내에서 저장 프로시저를 호출한다고 가정합니다. 이 경우 저장 프로시저의 모든 문은 사용자 지정 트랜잭션의 컨텍스트에서 실행됩니다. 프로시저는 종료 시 암시적으로 커밋하지 않습니다. 호출자는 프로시저 커밋 또는 롤백을 제어합니다.

저장 프로시저를 실행하는 동안 오류가 발생하면 현재 트랜잭션의 모든 변경 사항이 롤백됩니다.

저장 프로시저에서 다음과 같은 트랜잭션 제어 문을 사용할 수 있습니다.

- COMMIT - 현재 트랜잭션에서 수행된 모든 작업을 커밋하고 새 트랜잭션을 암시적으로 시작합니다. 자세한 내용은 COMMIT 단원을 참조하십시오.
- ROLLBACK - 현재 트랜잭션에서 수행된 작업을 롤백하고 새 트랜잭션을 암시적으로 시작합니다. 자세한 내용은 ROLLBACK 단원을 참조하십시오.

TRUNCATE는 저장 프로시저 내에서 발행할 수 있는 또 다른 문으로 트랜잭션 관리에 영향을 줍니다. Amazon Redshift에서 TRUNCATE는 커밋을 암시적으로 실행합니다. 이 동작은 저장 프로시저의 컨텍

스트에서 동일하게 유지됩니다. 저장 프로시저 내에서 TRUNCATE 문이 실행되면 현재 트랜잭션을 커밋하고 새 트랜잭션을 시작합니다. 자세한 내용은 TRUNCATE 단원을 참조하십시오.

COMMIT, ROLLBACK 또는 TRUNCATE 문을 따르는 모든 문은 새 트랜잭션의 컨텍스트에서 실행됩니다. COMMIT, ROLLBACK 또는 TRUNCATE 문이 나타나거나 저장 프로시저가 종료될 때까지 실행됩니다.

저장 프로시저 내에서 COMMIT, ROLLBACK 또는 TRUNCATE 문을 사용하는 경우, 다음 제약이 적용됩니다.

- 트랜잭션 블록 내에서 저장 프로시저가 호출된 경우, COMMIT, ROLLBACK 문을 TRUNCATE 문을 실행할 수 없습니다. 이 제약은 저장 프로시저의 자체 본문 및 중첩된 모든 프로시저 호출 내에 적용됩니다.
- 저장 프로시저가 `SET config` 옵션으로 생성된 경우, COMMIT, ROLLBACK 문을 TRUNCATE 문을 실행할 수 없습니다. 이 제약은 저장 프로시저의 자체 본문 및 중첩된 모든 프로시저 호출 내에 적용됩니다.
- COMMIT, ROLLBACK 또는 TRUNCATE 문이 처리되면 (명시적으로 또는 암시적으로) 열린 모든 커서가 자동으로 닫힙니다. 명시적 및 암시적 커서에 대한 제약은 저장 프로시저 제한 사항 섹션을 참조하세요.

또한 동적 SQL을 사용하여 COMMIT 또는 ROLLBACK을 실행할 수 없습니다. 그러나 동적 SQL을 사용하여 TRUNCATE를 실행할 수 있습니다. 자세한 내용은 Dynamic SQL 단원을 참조하십시오.

저장 프로시저를 작업할 때 PL/pgSQL의 BEGIN 및 END 문은 그룹화에만 사용된다는 점을 고려하십시오. 트랜잭션을 시작하거나 종료하지 않습니다. 자세한 내용은 차단 단원을 참조하십시오.

다음 예제에서는 명시적 트랜잭션 블록 내에서 저장 프로시저를 호출할 때 트랜잭션 동작을 보여 줍니다. 저장 프로시저 외부에서 실행된 두 삽입 문과 내부에서 실행된 한 삽입 문은 모두 동일한 트랜잭션(3382)의 일부입니다. 사용자가 명시적 커밋을 실행하면 트랜잭션이 커밋됩니다.

```
CREATE OR REPLACE PROCEDURE sp_insert_table_a(a int) LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO test_table_a values (a);
END;
$$;

Begin;
  insert into test_table_a values (1);
  Call sp_insert_table_a(2);
```

```
  insert into test_table_a values (3);
Commit;

select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  | pid |  type   |               stmt_text
--------+------+-----+---------+----------------------------------------
    103 | 3382 | 599 | UTILITY | Begin;
    103 | 3382 | 599 | QUERY   | insert into test_table_a values (1);
    103 | 3382 | 599 | UTILITY | Call sp_insert_table_a(2);
    103 | 3382 | 599 | QUERY   | INSERT INTO test_table_a values ( $1 )
    103 | 3382 | 599 | QUERY   | insert into test_table_a values (3);
    103 | 3382 | 599 | UTILITY | COMMIT
```

반대로 동일한 문이 명시적 트랜잭션 블록 외부에서 실행되고 세션이 자동 커밋을 ON으로 설정한 경우를 예로 들겠습니다. 이 경우 각 문은 자체 트랜잭션에서 실행됩니다.

```
insert into test_table_a values (1);
Call sp_insert_table_a(2);
insert into test_table_a values (3);

select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  | pid |  type   |
             stmt_text
--------+------+-----+---------
+-----------------------------------------------------------------------------------------------------------------
    103 | 3388 | 599 | QUERY   | insert into test_table_a values (1);
    103 | 3388 | 599 | UTILITY | COMMIT
    103 | 3389 | 599 | UTILITY | Call sp_insert_table_a(2);
    103 | 3389 | 599 | QUERY   | INSERT INTO test_table_a values ( $1 )
    103 | 3389 | 599 | UTILITY | COMMIT
    103 | 3390 | 599 | QUERY   | insert into test_table_a values (3);
    103 | 3390 | 599 | UTILITY | COMMIT
```

다음 예제에서는 test_table_a에 삽입한 후 TRUNCATE 문을 실행합니다. TRUNCATE 문은 현재 트랜잭션(3335)을 커밋하고 새 트랜잭션(3336)을 시작하는 암시적 커밋을 실행합니다. 프로시저가 종료되면 새 트랜잭션이 커밋됩니다.

```
CREATE OR REPLACE PROCEDURE sp_truncate_proc(a int, b int) LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO test_table_a values (a);
  TRUNCATE test_table_b;
  INSERT INTO test_table_b values (b);
END;
$$;

Call sp_truncate_proc(1,2);

select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  |  pid  |  type   |
                                        stmt_text
--------+------+-------+---------
+-----------------------------------------------------------------------------------------------------
    103 | 3335 | 23636 | UTILITY | Call sp_truncate_proc(1,2);
    103 | 3335 | 23636 | QUERY   | INSERT INTO test_table_a values ( $1 )
    103 | 3335 | 23636 | UTILITY | TRUNCATE test_table_b
    103 | 3335 | 23636 | UTILITY | COMMIT
    103 | 3336 | 23636 | QUERY   | INSERT INTO test_table_b values ( $1 )
    103 | 3336 | 23636 | UTILITY | COMMIT
```

다음 예제에서는 중첩 호출에서 TRUNCATE를 실행합니다. TRUNCATE는 트랜잭션(3344)의 외부 및 내부 프로시저에서 지금까지 완료된 모든 작업을 커밋합니다. 새 트랜잭션(3345)을 시작합니다. 외부 프로시저가 종료되면 새 트랜잭션이 커밋됩니다.

```
CREATE OR REPLACE PROCEDURE sp_inner(c int, d int) LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO inner_table values (c);
  TRUNCATE outer_table;
  INSERT INTO inner_table values (d);
END;
$$;

CREATE OR REPLACE PROCEDURE sp_outer(a int, b int, c int, d int) LANGUAGE plpgsql
AS $$
BEGIN
```

```
  INSERT INTO outer_table values (a);
  Call sp_inner(c, d);
  INSERT INTO outer_table values (b);
END;
$$;

Call sp_outer(1, 2, 3, 4);

select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  |  pid  |  type   |
                                          stmt_text
--------+------+-------+---------
+-----------------------------------------------------------------------------------------------
    103 | 3344 | 23636 | UTILITY | Call sp_outer(1, 2, 3, 4);
    103 | 3344 | 23636 | QUERY   | INSERT INTO outer_table values ( $1 )
    103 | 3344 | 23636 | UTILITY | CALL sp_inner( $1 , $2 )
    103 | 3344 | 23636 | QUERY   | INSERT INTO inner_table values ( $1 )
    103 | 3344 | 23636 | UTILITY | TRUNCATE outer_table
    103 | 3344 | 23636 | UTILITY | COMMIT
    103 | 3345 | 23636 | QUERY   | INSERT INTO inner_table values ( $1 )
    103 | 3345 | 23636 | QUERY   | INSERT INTO outer_table values ( $1 )
    103 | 3345 | 23636 | UTILITY | COMMIT
```

다음 예제에서는 TRUNCATE 문이 커밋되었을 때 커서 cur1이 닫혔다는 것을 보여 줍니다.

```
CREATE OR REPLACE PROCEDURE sp_open_cursor_truncate()
LANGUAGE plpgsql
AS $$
DECLARE
  rec RECORD;
  cur1 cursor for select * from test_table_a order by 1;
BEGIN
  open cur1;
  TRUNCATE table test_table_b;
  Loop
    fetch cur1 into rec;
    raise info '%', rec.c1;
    exit when not found;
  End Loop;
END
```

```
$$;

call sp_open_cursor_truncate();
ERROR: cursor "cur1" does not exist
CONTEXT: PL/pgSQL function "sp_open_cursor_truncate" line 8 at fetch
```

다음 예제에서는 TRUNCATE 문을 실행하고 명시적 트랜잭션 블록 내에서 호출할 수 없습니다.

```
CREATE OR REPLACE PROCEDURE sp_truncate_atomic() LANGUAGE plpgsql
AS $$
BEGIN
  TRUNCATE test_table_b;
END;
$$;

Begin;
  Call sp_truncate_atomic();
ERROR: TRUNCATE cannot be invoked from a procedure that is executing in an atomic
 context.
HINT: Try calling the procedure as a top-level call i.e. not from within an explicit
 transaction block.
Or, if this procedure (or one of its ancestors in the call chain) was created with SET
 config options, recreate the procedure without them.
CONTEXT: SQL statement "TRUNCATE test_table_b"
PL/pgSQL function "sp_truncate_atomic" line 2 at SQL statement
```

다음 예에서는 슈퍼 사용자 또는 테이블 소유자가 아닌 사용자가 테이블에 TRUNCATE 문을 실행할 수 있음을 보여줍니다. 사용자는 `Security Definer` 저장 프로시저를 사용하여 이 작업을 수행합니다. 이 예에서는 다음 작업을 보여줍니다.

- user1이 테이블 `test_tbl`을 생성합니다.
- user1이 저장 프로시저 `sp_truncate_test_tbl`을 생성합니다.
- user1은 저장 프로시저에 대한 EXECUTE 권한을 user2에게 부여합니다.
- user2는 저장 프로시저를 실행하여 테이블 `test_tbl`을 잘라냅니다. 이 예에서는 TRUNCATE 명령 전후의 행 수를 보여줍니다.

```
set session_authorization to user1;
create table test_tbl(id int, name varchar(20));
insert into test_tbl values (1,'john'), (2, 'mary');
CREATE OR REPLACE PROCEDURE sp_truncate_test_tbl() LANGUAGE plpgsql
```

```
AS $$
DECLARE
  tbl_rows int;
BEGIN
  select count(*) into tbl_rows from test_tbl;
  RAISE INFO 'RowCount before Truncate: %', tbl_rows;
  TRUNCATE test_tbl;
  select count(*) into tbl_rows from test_tbl;
  RAISE INFO 'RowCount after Truncate: %', tbl_rows;
END;
$$ SECURITY DEFINER;
grant execute on procedure sp_truncate_test_tbl() to user2;
reset session_authorization;


set session_authorization to user2;
call sp_truncate_test_tbl();
INFO:  RowCount before Truncate: 2
INFO:  RowCount after Truncate: 0
CALL
reset session_authorization;
```

다음 예에서는 COMMIT을 두 번 실행합니다. 첫 번째 COMMIT는 트랜잭션 10363에서 수행된 모든 작업을 커밋하고 트랜잭션 10364를 암시적으로 시작합니다. 트랜잭션 10364는 두 번째 COMMIT 문으로 커밋됩니다.

```
CREATE OR REPLACE PROCEDURE sp_commit(a int, b int) LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO test_table values (a);
  COMMIT;
  INSERT INTO test_table values (b);
  COMMIT;
END;
$$;

call sp_commit(1,2);

select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;
 userid |  xid  | pid  |  type   |
                               stmt_text
```

```
--------+-------+------+---------
+-----------------------------------------------------------------------------------------------
    100 | 10363 | 3089 | UTILITY | call sp_commit(1,2);
    100 | 10363 | 3089 | QUERY   | INSERT INTO test_table values ( $1 )
    100 | 10363 | 3089 | UTILITY | COMMIT
    100 | 10364 | 3089 | QUERY   | INSERT INTO test_table values ( $1 )
    100 | 10364 | 3089 | UTILITY | COMMIT
```

다음 예제에서는 sum_vals가 2보다 큰 경우 ROLLBACK 문을 실행합니다. 첫 번째 ROLLBACK 문은 트랜잭션 10377에서 수행된 모든 작업을 롤백하고 새 트랜잭션 10378을 시작합니다. 프로시저가 종료되면 트랜잭션 10378이 커밋됩니다.

```
CREATE OR REPLACE PROCEDURE sp_rollback(a int, b int) LANGUAGE plpgsql
AS $$
DECLARE
  sum_vals int;
BEGIN
  INSERT INTO test_table values (a);
  SELECT sum(c1) into sum_vals from test_table;
  IF sum_vals > 2 THEN
    ROLLBACK;
  END IF;

  INSERT INTO test_table values (b);
END;
$$;

call sp_rollback(1, 2);

select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

userid |  xid  | pid  |  type   |
                               stmt_text
--------+-------+------+---------
+-----------------------------------------------------------------------------------------------
    100 | 10377 | 3089 | UTILITY | call sp_rollback(1, 2);
    100 | 10377 | 3089 | QUERY   | INSERT INTO test_table values ( $1 )
    100 | 10377 | 3089 | QUERY   | SELECT sum(c1) from test_table
    100 | 10377 | 3089 | QUERY   | Undoing 1 transactions on table 133646 with current
 xid 10377 : 10377
    100 | 10378 | 3089 | QUERY   | INSERT INTO test_table values ( $1 )
```

```
    100 | 10378 | 3089 | UTILITY | COMMIT
```

NONATOMIC 모드 저장 프로시저 트랜잭션 관리

NONATOMIC 모드에서 생성된 저장 프로시저는 기본 모드에서 생성된 프로시저와는 다른 트랜잭션 제어 동작을 가집니다. 저장 프로시저 외부에 있는 SQL 명령의 자동 커밋 동작과 유사하게, NONATOMIC 프로시저 내의 각 SQL 문은 자체 트랜잭션에서 실행되고 자동으로 커밋됩니다. 사용자가 NONATOMIC 저장 프로시저 내에서 명시적 트랜잭션 블록을 시작하는 경우 블록 내의 SQL 문은 자동으로 커밋되지 않습니다. 트랜잭션 블록은 트랜잭션 블록 내 문의 커밋 또는 롤백을 제어합니다.

NONATOMIC 저장 프로시저에서는 START TRANSACTION 문을 사용하여 프로시저 내에서 명시적 트랜잭션 블록을 열 수 있습니다. 하지만 이미 열려 있는 트랜잭션 블록이 있는 경우 Amazon Redshift는 하위 트랜잭션을 지원하지 않으므로 이 명령문은 아무런 동작도 하지 않습니다. 이전 트랜잭션이 계속됩니다.

NONATOMIC 프로시저에서 cursor FOR 루프를 사용하는 경우 쿼리 결과를 반복하기 전에 명시적 트랜잭션 블록을 열어야 합니다. 그러지 않으면 루프 내의 SQL 문이 자동으로 커밋될 때 커서가 닫힙니다.

NONATOMIC 모드 동작을 사용할 때 고려해야 할 몇 가지 사항은 다음과 같습니다.

- 열린 트랜잭션 블록이 없고 세션에 자동 커밋이 켜짐으로 설정된 경우 저장 프로시저 내의 각 SQL 문이 자동으로 커밋됩니다.
- 저장 프로시저가 트랜잭션 블록 내에서 호출되는 경우 COMMIT/ROLLBACK/TRUNCATE 문을 실행하여 트랜잭션을 종료할 수 있습니다. 이는 기본 모드에서는 불가능합니다.
- START TRANSACTION 문을 실행하여 저장 프로시저 내에서 트랜잭션 블록을 시작할 수 있습니다.

다음 예제는 NONATOMIC 저장 프로시저를 사용할 때의 트랜잭션 동작을 보여줍니다. 다음 예제의 모든 세션에서는 자동 커밋이 켜짐으로 설정되어 있습니다.

다음 예제에서 NONATOMIC 저장 프로시저에는 2개의 INSERT 문이 있습니다. 트랜잭션 블록 외부에서 프로시저를 호출하면 프로시저 내의 모든 INSERT 문이 자동으로 커밋됩니다.

```
CREATE TABLE test_table_a(v int);
CREATE TABLE test_table_b(v int);

CREATE OR REPLACE PROCEDURE sp_nonatomic_insert_table_a(a int, b int) NONATOMIC AS
$$
BEGIN
    INSERT INTO test_table_a values (a);
```

```
    INSERT INTO test_table_b values (b);
END;
$$
LANGUAGE plpgsql;

Call sp_nonatomic_insert_table_a(1,2);

Select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  |    pid     |  type   |               stmt_text
--------+------+------------+---------+----------------------------------------
      1 | 1792 | 1073807554 | UTILITY | Call sp_nonatomic_insert_table_a(1,2);
      1 | 1792 | 1073807554 | QUERY   | INSERT INTO test_table_a values ( $1 )
      1 | 1792 | 1073807554 | UTILITY | COMMIT
      1 | 1793 | 1073807554 | QUERY   | INSERT INTO test_table_b values ( $1 )
      1 | 1793 | 1073807554 | UTILITY | COMMIT
(5 rows)
```

그러나 BEGIN..COMMIT 블록 내에서 프로시저를 호출하면 모든 문이 동일한 트랜잭션의 일부가 됩니다(xid=1799).

```
Begin;
  INSERT INTO test_table_a values (10);
  Call sp_nonatomic_insert_table_a(20,30);
  INSERT INTO test_table_b values (40);
Commit;

Select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  |    pid     |  type   |                stmt_text
--------+------+------------+---------+-----------------------------------------
      1 | 1799 | 1073914035 | UTILITY | Begin;
      1 | 1799 | 1073914035 | QUERY   | INSERT INTO test_table_a values (10);
      1 | 1799 | 1073914035 | UTILITY | Call sp_nonatomic_insert_table_a(20,30);
      1 | 1799 | 1073914035 | QUERY   | INSERT INTO test_table_a values ( $1 )
      1 | 1799 | 1073914035 | QUERY   | INSERT INTO test_table_b values ( $1 )
      1 | 1799 | 1073914035 | QUERY   | INSERT INTO test_table_b values (40);
      1 | 1799 | 1073914035 | UTILITY | COMMIT
(7 rows)
```

이 예제에서는 START TRANSACTION...COMMIT 사이에 두 개의 INSERT 문이 있습니다. 프로시저가 트랜잭션 블록 외부에서 호출되면 두 INSERT 문은 동일한 트랜잭션에 있게 됩니다(xid=1866).

```
CREATE OR REPLACE PROCEDURE sp_nonatomic_txn_block(a int, b int) NONATOMIC AS
$$
BEGIN
    START TRANSACTION;
    INSERT INTO test_table_a values (a);
    INSERT INTO test_table_b values (b);
    COMMIT;
END;
$$
LANGUAGE plpgsql;

Call sp_nonatomic_txn_block(1,2);

Select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  |    pid     |  type   |               stmt_text
--------+------+------------+---------+----------------------------------------
      1 | 1865 | 1073823998 | UTILITY | Call sp_nonatomic_txn_block(1,2);
      1 | 1866 | 1073823998 | QUERY   | INSERT INTO test_table_a values ( $1 )
      1 | 1866 | 1073823998 | QUERY   | INSERT INTO test_table_b values ( $1 )
      1 | 1866 | 1073823998 | UTILITY | COMMIT
(4 rows)
```

BEGIN...COMMIT 블록 내에서 프로시저를 호출하면 이미 열려 있는 트랜잭션이 있기 때문에 프로시저 내의 START TRANSACTION은 아무런 동작도 하지 않습니다. 프로시저 내의 COMMIT은 현재 트랜잭션(xid=1876)을 커밋하고 새 트랜잭션을 시작합니다.

```
Begin;
  INSERT INTO test_table_a values (10);
  Call sp_nonatomic_txn_block(20,30);
  INSERT INTO test_table_b values (40);
Commit;

Select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  |    pid     |  type   |               stmt_text
```

```
--------+------+------------+---------+---------------------------------------
      1 | 1876 | 1073832133 | UTILITY | Begin;
      1 | 1876 | 1073832133 | QUERY   | INSERT INTO test_table_a values (10);
      1 | 1876 | 1073832133 | UTILITY | Call sp_nonatomic_txn_block(20,30);
      1 | 1876 | 1073832133 | QUERY   | INSERT INTO test_table_a values ( $1 )
      1 | 1876 | 1073832133 | QUERY   | INSERT INTO test_table_b values ( $1 )
      1 | 1876 | 1073832133 | UTILITY | COMMIT
      1 | 1878 | 1073832133 | QUERY   | INSERT INTO test_table_b values (40);
      1 | 1878 | 1073832133 | UTILITY | COMMIT
(8 rows)
```

이 예제에서는 커서 루프를 사용하는 방법을 보여줍니다. 테이블 test_table_a에는 세 개의 값이 있습니다. 목표는 세 값을 반복하여 test_table_b 테이블에 삽입하는 것입니다. 다음과 같은 방식으로 NONATOMIC 저장 프로시저를 만들면 첫 번째 루프에서 INSERT 문을 실행한 후 커서 'cur1'이 존재하지 않는다는 오류가 발생합니다. 이는 INSERT의 자동 커밋이 열린 커서를 닫기 때문입니다.

```
insert into test_table_a values (1), (2), (3);

CREATE OR REPLACE PROCEDURE sp_nonatomic_cursor() NONATOMIC
LANGUAGE plpgsql
AS $$
DECLARE
  rec RECORD;
  cur1 cursor for select * from test_table_a order by 1;
BEGIN
  open cur1;
  Loop
    fetch cur1 into rec;
    exit when not found;
    raise info '%', rec.v;
    insert into test_table_b values (rec.v);
  End Loop;
END
$$;

CALL sp_nonatomic_cursor();

INFO:  1
ERROR:  cursor "cur1" does not exist
CONTEXT:  PL/pgSQL function "sp_nonatomic_cursor" line 7 at fetch
```

커서 루프가 작동하도록 하려면 START TRANSACTION...COMMIT 사이에 루프를 배치하세요.

```
insert into test_table_a values (1), (2), (3);

CREATE OR REPLACE PROCEDURE sp_nonatomic_cursor() NONATOMIC
LANGUAGE plpgsql
AS $$
DECLARE
  rec RECORD;
  cur1 cursor for select * from test_table_a order by 1;
BEGIN
  START TRANSACTION;
  open cur1;
  Loop
    fetch cur1 into rec;
    exit when not found;
    raise info '%', rec.v;
    insert into test_table_b values (rec.v);
  End Loop;
  COMMIT;
END
$$;

CALL sp_nonatomic_cursor();

INFO:  1
INFO:  2
INFO:  3
CALL
```

오류 트래핑

이 주제에서는 Amazon Redshift에서 오류를 처리하는 방법을 설명합니다.

저장 프로시저의 쿼리 또는 명령에 오류가 발생하면 후속 쿼리가 실행되지 않고 트랜잭션이 롤백됩니다. 그러나 EXCEPTION 블록을 사용하여 오류를 처리할 수 있습니다.

> **Note**
>
> 기본 동작은 저장 프로시저에 추가 오류 생성 조건이 없는 경우에도 오류로 인해 후속 쿼리가 실행되지 않는 것입니다.

```
[ <<label>> ]
[ DECLARE
  declarations ]
BEGIN
  statements
EXCEPTION
  WHEN OTHERS THEN
    statements
END;
```

예외가 발생하고 예외 처리 블록을 추가하면 RAISE 문과 대부분의 다른 PL/pgSQL 문을 작성할 수 있습니다. 예를 들어 사용자 정의 메시지로 예외를 발생시키거나 기록을 로깅 테이블에 삽입할 수 있습니다.

예외 처리 블록에 들어갈 때 현재 트랜잭션이 롤백되고 블록의 명령문을 실행하기 위해 새 트랜잭션이 생성됩니다. 블록의 문이 오류 없이 실행되면 트랜잭션이 커밋되고 예외가 다시 발생합니다. 마지막으로 저장 프로시저가 종료됩니다.

예외 블록에서 지원되는 유일한 조건은 쿼리 취소를 제외한 모든 오류 유형과 일치하는 OTHERS입니다. 또한 예외 처리 블록에서 오류가 발생하면 외부 예외 처리 블록에서 이를 catch할 수 있습니다.

NONATOMIC 프로시저 내에서 오류가 발생하여 예외 블록에서 처리되는 경우 오류가 다시 발생하지 않습니다. 예외 처리 블록에서 포착한 예외를 발생시키려면 PL/pgSQL 문 RAISE를 참조하세요. 이 문은 예외 처리 블록에서만 유효합니다. 자세한 정보는 RAISE 섹션을 참조하세요.

저장 프로시저에서 오류가 나타난 후 발생하는 상황을 CONTINUE 핸들러를 사용하여 제어

CONTINUE 핸들러는 NONATOMIC 저장 프로시저 내의 실행 흐름을 제어하는 일종의 예외 핸들러입니다. 핸들러를 사용하면 기존 문 블록을 종료하지 않고도 예외를 파악하고 처리할 수 있습니다. 일반적으로 저장 프로시저에서 오류가 발생하면 흐름이 중단되고 오류가 호출자에게 반환됩니다. 하지만 일부 사용 사례에서는 오류 상태가 흐름이 중단될 만큼 심각하지 않을 수도 있습니다. 별도의 트랜잭션에서 선택한 오류 처리 로직을 사용하여 오류를 적절하게 처리한 다음 오류 다음에 오는 문을 계속 실행하는 것이 좋습니다. 다음은 구문을 보여줍니다.

```
[ DECLARE
  declarations ]
BEGIN
  statements
EXCEPTION
  [ CONTINUE_HANDLER | EXIT_HANDLER ] WHEN OTHERS THEN
    handler_statements
```

```
END;
```

다양한 유형의 오류에 대한 정보를 수집하는 데 도움이 되는 몇 가지 시스템 테이블이 있습니다. 자세한 내용은 STL_LOAD_ERRORS, STL_ERROR, SYS_STREAM_SCAN_ERRORS 단원을 참조하세요. 오류 해결에 사용할 수 있는 추가 시스템 테이블도 있습니다. 이에 대한 자세한 정보는 시스템 테이블 및 뷰 참조에서 찾아볼 수 있습니다.

예

다음 예는 예외 처리 블록에서 문을 작성하는 방법을 보여줍니다. 저장 프로시저는 기본 트랜잭션 관리 동작을 사용합니다.

```
CREATE TABLE employee (firstname varchar, lastname varchar);
INSERT INTO employee VALUES ('Tomas','Smith');
CREATE TABLE employee_error_log (message varchar);

CREATE OR REPLACE PROCEDURE update_employee_sp() AS
$$
BEGIN
    UPDATE employee SET firstname = 'Adam' WHERE lastname = 'Smith';
    EXECUTE 'select invalid';
EXCEPTION WHEN OTHERS THEN
    RAISE INFO 'An exception occurred.';
    INSERT INTO employee_error_log VALUES ('Error message: ' || SQLERRM);
END;
$$
LANGUAGE plpgsql;

CALL update_employee_sp();

INFO:  An exception occurred.
ERROR:  column "invalid" does not exist
CONTEXT:  SQL statement "select invalid"
PL/pgSQL function "update_employee_sp" line 3 at execute statement
```

이 예에서 update_employee_sp를 호출하면 예외 발생(An exception occurred.)이라는 정보 메시지가 나타나고 로깅 테이블의 employee_error_log 로그에 오류 메시지가 삽입됩니다. 저장 프로시저가 종료되기 전에 원래 예외가 다시 발생합니다. 다음 쿼리는 예제를 실행한 결과로 생성된 레코드를 보여줍니다.

```
SELECT * from employee;
```

```
firstname | lastname
-----------+----------
 Tomas     | Smith

SELECT * from employee_error_log;

          message
-----------------------------------------------
 Error message: column "invalid" does not exist
```

형식 관련 도움말 및 추가 수준 목록 등 RAISE에 대한 자세한 내용은 지원되는 PL/pgSQL 문 섹션을 참조하세요.

다음 예는 예외 처리 블록에서 문을 작성하는 방법을 보여줍니다. 저장 프로시저는 NONATOMIC 트랜잭션 관리 동작을 사용합니다. 이 예제에서는 프로시저 호출이 완료된 후 호출자에게 다시 오류가 발생하지 않습니다. 다음 문의 오류로 인해 UPDATE 문이 롤백되지 않습니다. 정보 메시지가 표시되고 오류 메시지가 로깅 테이블에 삽입됩니다.

```
CREATE TABLE employee (firstname varchar, lastname varchar);
INSERT INTO employee VALUES ('Tomas','Smith');
CREATE TABLE employee_error_log (message varchar);

-- Create the SP in NONATOMIC mode
CREATE OR REPLACE PROCEDURE update_employee_sp_2() NONATOMIC AS
$$
BEGIN
    UPDATE employee SET firstname = 'Adam' WHERE lastname = 'Smith';
    EXECUTE 'select invalid';
EXCEPTION WHEN OTHERS THEN
    RAISE INFO 'An exception occurred.';
    INSERT INTO employee_error_log VALUES ('Error message: ' || SQLERRM);
END;
$$
LANGUAGE plpgsql;

CALL update_employee_sp_2();
INFO:  An exception occurred.
CALL

SELECT * from employee;

 firstname | lastname
-----------+----------
```

```
 Adam      | Smith
(1 row)

SELECT * from employee_error_log;

                    message
-----------------------------------------------
 Error message: column "invalid" does not exist
(1 row)
```

이 예에서는 2개의 하위 블록으로 프로시저를 만드는 방법을 보여줍니다. 저장 프로시저가 호출되면 첫 번째 하위 블록의 오류가 해당 예외 처리 블록에서 처리됩니다. 첫 번째 하위 블록이 완료된 후 프로시저는 두 번째 하위 블록을 계속 실행합니다. 결과를 보면 프로시저 호출이 완료될 때 오류가 발생하지 않음을 알 수 있습니다. employee 테이블에 대한 UPDATE 및 INSERT 작업이 커밋됩니다. 두 예외 블록의 오류 메시지가 로깅 테이블에 삽입됩니다.

```
CREATE TABLE employee (firstname varchar, lastname varchar);
INSERT INTO employee VALUES ('Tomas','Smith');
CREATE TABLE employee_error_log (message varchar);

CREATE OR REPLACE PROCEDURE update_employee_sp_3() NONATOMIC AS
$$
BEGIN
    BEGIN
        UPDATE employee SET firstname = 'Adam' WHERE lastname = 'Smith';
        EXECUTE 'select invalid1';
    EXCEPTION WHEN OTHERS THEN
        RAISE INFO 'An exception occurred in the first block.';
        INSERT INTO employee_error_log VALUES ('Error message: ' || SQLERRM);
    END;
    BEGIN
        INSERT INTO employee VALUES ('Edie','Robertson');
        EXECUTE 'select invalid2';
    EXCEPTION WHEN OTHERS THEN
        RAISE INFO 'An exception occurred in the second block.';
        INSERT INTO employee_error_log VALUES ('Error message: ' || SQLERRM);
    END;
END;
$$
LANGUAGE plpgsql;

CALL update_employee_sp_3();
INFO:  An exception occurred in the first block.
```

```
INFO:  An exception occurred in the second block.
CALL

SELECT * from employee;

 firstname | lastname
-----------+-----------
 Adam      | Smith
 Edie      | Robertson
(2 rows)

SELECT * from employee_error_log;

                    message
-------------------------------------------------
 Error message: column "invalid1" does not exist
 Error message: column "invalid2" does not exist
(2 rows)
```

다음 예시는 CONTINUE 예외 핸들러를 사용하는 방법을 보여줍니다. 이 샘플은 두 개의 테이블을 만들어 저장 프로시저에서 사용합니다. CONTINUE 핸들러는 NONATOMIC 트랜잭션 관리 동작을 사용하여 저장 프로시저의 실행 흐름을 제어합니다.

```
CREATE TABLE tbl_1 (a int);
CREATE TABLE tbl_error_logging(info varchar, err_state varchar, err_msg varchar);

CREATE OR REPLACE PROCEDURE sp_exc_handling_1() NONATOMIC AS
$$
BEGIN
    INSERT INTO tbl_1 VALUES (1);
    -- Expect an error for the insert statement following, because of the invalid value
    INSERT INTO tbl_1 VALUES ("val");
    INSERT INTO tbl_1 VALUES (2);
EXCEPTION CONTINUE_HANDLER WHEN OTHERS THEN
    INSERT INTO tbl_error_logging VALUES ('Encountered error', SQLSTATE, SQLERRM);
END;
$$ LANGUAGE plpgsql;
```

저장 프로시저를 호출합니다.

```
CALL sp_exc_handling_1();
```

흐름은 다음과 같이 진행됩니다.

1. 호환되지 않는 데이터 유형을 열에 삽입하려고 하면 오류가 발생합니다. 제어가 EXCEPTION 블록으로 전달됩니다. 예외 처리 블록이 입력될 때 현재 트랜잭션이 롤백되고 문을 실행하기 위해 새 암시적 트랜잭션이 생성됩니다.
2. CONTINUE_HANDLER의 문이 오류 없이 실행되면 제어가 문 바로 뒤에 있는 문으로 전달되어 예외가 발생합니다. CONTINUE_HANDLER의 문에서 새 예외가 발생하는 경우 EXCEPTION 블록 내에서 예외 핸들러를 사용하여 해당 예외를 처리할 수 있습니다.

샘플 저장 프로시저를 호출한 후 테이블에 다음 레코드가 포함됩니다.

- `SELECT * FROM tbl_1;`을 실행하면 두 개의 레코드가 반환됩니다. 여기에는 1 및 2 값이 포함됩니다.
- `SELECT * FROM tbl_error_logging;`을 실행하면 오류 발생, 42703, 'val' 열이 tbl_1에 존재하지 않음이라는 값이 있는 레코드 하나가 반환됩니다.

다음 추가 오류 처리 예시에서는 EXIT 핸들러와 CONTINUE 핸들러를 모두 사용합니다. 그러면 데이터 테이블과 로깅 테이블이라는 두 개의 테이블이 생성됩니다. 또한 오류 처리를 보여주는 저장 프로시저가 생성됩니다.

```
CREATE TABLE tbl_1 (a int);
CREATE TABLE tbl_error_logging(info varchar, err_state varchar, err_msg varchar);

CREATE OR REPLACE PROCEDURE sp_exc_handling_2() NONATOMIC AS
$$
BEGIN
    INSERT INTO tbl_1 VALUES (1);
    BEGIN
        INSERT INTO tbl_1 VALUES (100);
        -- Expect an error for the insert statement following, because of the invalid
 value
        INSERT INTO tbl_1 VALUES ("val");
        INSERT INTO tbl_1 VALUES (101);
    EXCEPTION EXIT_HANDLER WHEN OTHERS THEN
        INSERT INTO tbl_error_logging VALUES ('Encountered error', SQLSTATE, SQLERRM);
    END;
    INSERT INTO tbl_1 VALUES (2);
    -- Expect an error for the insert statement following, because of the invalid value
    INSERT INTO tbl_1 VALUES ("val");
```

```
    INSERT INTO tbl_1 VALUES (3);
EXCEPTION CONTINUE_HANDLER WHEN OTHERS THEN
    INSERT INTO tbl_error_logging VALUES ('Encountered error', SQLSTATE, SQLERRM);
END;
$$ LANGUAGE plpgsql;
```

저장 프로시저를 만든 후 다음을 사용하여 호출합니다.

```
CALL sp_exc_handling_2();
```

BEGIN과 END라는 내부 세트로 괄호가 묶인 내부 예외 블록에서 오류가 발생하면 EXIT 핸들러가 오류를 처리합니다. 외부 블록에서 발생하는 모든 오류는 CONTINUE 핸들러에서 처리합니다.

샘플 저장 프로시저를 호출한 후 테이블에 다음 레코드가 포함됩니다.

- `SELECT * FROM tbl_1;`을 실행하면 값이 1, 2, 3, 100인 레코드 네 개가 반환됩니다.
- `SELECT * FROM tbl_error_logging;`을 실행하면 두 개의 레코드가 반환됩니다. 값은 오류 발생, 42703, 'val' 열이 tbl_1에 존재하지 않음입니다.

tbl_error_logging 테이블이 존재하지 않는 경우 예외가 발생합니다.

다음 예시는 FOR 루프와 함께 CONTINUE 예외 핸들러를 사용하는 방법을 보여줍니다. 이 샘플은 세 개의 테이블을 만들어 저장 프로시저 내의 FOR 루프에서 사용합니다. FOR 루프는 결과 세트 변형입니다. 즉, 쿼리 결과를 반복합니다.

```
CREATE TABLE tbl_1 (a int);
INSERT INTO tbl_1 VALUES (1), (2), (3);
CREATE TABLE tbl_2 (a int);
CREATE TABLE tbl_error_logging(info varchar, err_state varchar, err_msg varchar);

CREATE OR REPLACE PROCEDURE sp_exc_handling_loop() NONATOMIC AS
$$
DECLARE
 rec RECORD;
BEGIN
    FOR rec IN SELECT a FROM tbl_1
    LOOP
        IF rec.a = 2 THEN
            -- Expect an error for the insert statement following, because of the
 invalid value
```

```
            INSERT INTO tbl_2 VALUES("val");
        ELSE
            INSERT INTO tbl_2 VALUES (rec.a);
        END IF;
    END LOOP;
EXCEPTION CONTINUE_HANDLER WHEN OTHERS THEN
    INSERT INTO tbl_error_logging VALUES ('Encountered error', SQLSTATE, SQLERRM);
END;
$$ LANGUAGE plpgsql;
```

저장 프로시저를 호출합니다.

```
CALL sp_exc_handling_loop();
```

샘플 저장 프로시저를 호출한 후 테이블에 다음 레코드가 포함됩니다.

- `SELECT * FROM tbl_2;`을 실행하면 두 개의 레코드가 반환됩니다. 여기에는 1 및 3 값이 포함됩니다.
- `SELECT * FROM tbl_error_logging;`을 실행하면 오류 발생, 42703, 'val' 열이 tbl_2에 존재하지 않음이라는 값이 있는 레코드 하나가 반환됩니다.

CONTINUE 핸들러와 관련된 사용 참고 사항:

- CONTINUE_HANDLER 및 EXIT_HANDLER 키워드는 NONATOMIC 저장 프로시저에서만 사용할 수 있습니다.
- CONTINUE_HANDLER 및 EXIT_HANDLER 키워드는 선택 사항입니다. EXIT_HANDLER가 기본값입니다.

저장 프로시저 로깅

이 주제에서는 Amazon Redshift가 저장 프로시저 로깅에 사용하는 저장 프로시저 및 뷰에 대해 설명합니다.

저장 프로시저에 대한 세부 정보가 다음 시스템 테이블 및 뷰에 기록됩니다.

- SVL_STORED_PROC_CALL - 저장 프로시저 호출의 시작 시간과 종료 시간, 호출이 완료 전에 종료되었는지 여부에 대한 세부 정보가 기록됩니다. 자세한 내용은 SVL_STORED_PROC_CALL 단원을 참조하십시오.

- SVL_STORED_PROC_MESSAGES - RAISE 쿼리에서 방출되는 저장 프로시저의 메시지는 해당 로깅 수준으로 기록됩니다. 자세한 내용은 SVL_STORED_PROC_MESSAGES 단원을 참조하십시오.
- SVL_QLOG - 저장 프로시저에서 호출된 각 쿼리에 대해 프로시저 호출의 쿼리 ID가 기록됩니다. 자세한 내용은 SVL_QLOG 단원을 참조하십시오.
- STL_UTILITYTEXT - 저장 프로시저 호출이 완료된 후 기록됩니다. 자세한 내용은 STL_UTILITYTEXT 단원을 참조하십시오.
- PG_PROC_INFO - 이 시스템 카탈로그 뷰는 저장 프로시저에 대한 정보를 보여줍니다. 자세한 내용은 PG_PROC_INFO 단원을 참조하십시오.

저장 프로시저 제한 사항

이 주제에서는 Amazon Redshift 저장 프로시저에 대한 제한 사항을 설명합니다.

Amazon Redshift 저장 프로시저를 사용하는 경우 다음 고려 사항이 적용됩니다.

저장 프로시저 지원에 대한 Amazon Redshift와 PostgreSQL 간의 차이점

다음은 Amazon Redshift와 PostgreSQL의 저장 프로시저 지원 간 차이점입니다.

- Amazon Redshift는 하위 트랜잭션을 지원하지 않으므로 예외 처리 블록에 대한 지원이 제한됩니다.

고려 사항 및 제한 사항

다음은 Amazon Redshift의 저장 프로시저에 대한 고려 사항입니다.

- 데이터베이스의 최대 저장 프로시저 수는 10,000개입니다.
- 프로시저 소스 코드의 최대 크기는 2MB입니다.
- 사용자 세션 하나에서 동시에 열 수 있는 명시적 및 묵시적 커서의 최대 수는 1개입니다. SQL 문의 결과 세트를 반복하는 FOR 루프는 암시적 커서를 엽니다. 중첩 커서는 지원되지 않습니다.
- 명시적 및 암시적 커서는 결과 집합 크기의 제한이 표준 Amazon Redshift 커서와 동일합니다. 자세한 내용은 커서 제약 조건 단원을 참조하십시오.
- 중첩 호출의 최대 수준 수는 16입니다.
- 프로시저 파라미터의 최대 수는 입력 인수의 경우 32, 출력 인수의 경우 32입니다.
- 저장 프로시저의 최대 변수 수는 1,024입니다.
- 자체 트랜잭션 컨텍스트가 필요한 모든 SQL 명령은 저장 프로시저 내부에서 지원되지 않습니다. 그러한 예는 다음과 같습니다.

- PREPARE
- CREATE/DROP DATABASE
- CREATE EXTERNAL TABLE
- VACUUM
- SET LOCAL
- ALTER TABLE APPEND

- Java Database Connectivity(JDBC) 드라이버를 통한 `registerOutParameter` 메서드 호출은 `refcursor` 데이터 형식에서 지원되지 않습니다. 데이터 형식 사용 예는 `refcursor` 섹션을 참조하세요..저장 프로시저에서 결과 세트 반환

PL/pgSQL 언어 참조

Amazon Redshift의 저장 프로시저는 PostgreSQL PL/pgSQL 프로시저 언어를 기반으로 하지만 몇 가지 중요한 차이점이 있습니다. 이 참조에서는 Amazon Redshift에서 구현한 PL/pgSQL 구문에 대한 세부 정보를 확인할 수 있습니다. PL/pgSQL에 대한 자세한 내용은 PostgreSQL 설명서의 PL/pgSQL - SQL procedural language 섹션을 참조하세요.

주제

- PL/pgSQL 참조 규칙
- PL/pgSQL의 구조
- 지원되는 PL/pgSQL 문

PL/pgSQL 참조 규칙

이 단원에서는 PL/pgSQL 저장 프로시저 언어의 구문을 작성할 때 사용되는 규칙을 찾을 수 있습니다.

문자	설명
CAPS	대문자로 된 단어는 키워드입니다.
[ ]	대괄호는 선택적 인수를 나타냅니다. 다수의 인수가 대괄호로 묶이면 인수를 얼마든지 선택할 수 있다는 것을 의미합니다. 또한 별도의 라인에서 대괄호로 묶이는 인수는 Amazon Redshift 구문 분석기에서 인수가 구문에 나열된 순서를 그대로 따른다는 것을 의미합니다.

문자	설명
{ }	중괄호는 중괄호 안의 인수 중 하나를 선택해야 함을 나타냅니다.
|	파이프는 인수 중에서 하나를 선택할 수 있다는 것을 의미합니다.
### ####	빨간색 기울임꼴 단어는 자리 표시자를 나타냅니다. 빨간색 기울임꼴 단어 자리에 적절한 값을 넣습니다.
. . .	줄임표는 앞의 요소를 반복할 수 있음을 나타냅니다.
'	작은따옴표로 묶인 단어는 따옴표를 입력해야 함을 나타냅니다.

PL/pgSQL의 구조

PL/pgSQL은 다른 프로시저 언어와 여러 동일한 구문를 가진 프로시저 언어입니다.

주제

- 차단
- 변수 선언
- 별칭 선언
- 기본 제공 변수
- 레코드 형식

차단

PL/pgSQL은 블록 구조 언어입니다. 프로시저의 전체 본문은 블록으로 정의되어 있으며, 변수 선언 및 PL/pgSQL 문이 포함되어 있습니다. 문은 중첩 블록, 즉 하위 블록일 수도 있습니다.

선언 및 문은 세미콜론으로 끝납니다. 세미콜론이 있는 블록 또는 하위 블록에서는 END 키워드를 따릅니다. DECLARE 및 BEGIN 키워드 뒤에는 세미콜론을 사용하지 마십시오.

모든 키워드 및 식별자는 대문자와 소문자 혼합으로 작성할 수 있습니다. 큰 따옴표로 묶이지 않은 한 식별자는 소문자로 암시적으로 변환됩니다.

이중 하이픈(--)은 행의 끝까지 확장되는 주석을 시작합니다. /*는 다음 */까지 확장되는 블록 주석을 시작합니다. 블록 주석은 중첩할 수 없습니다. 그러나 이중 하이픈 주석을 블록 주석으로 묶을 수 있고, 이중 하이픈은 블록 주석 구분 기호 /* 및 */를 숨길 수 있습니다.

블록의 문 섹션의 모든 문은 하위 블록일 수 있습니다. 하위 블록을 사용하여 논리적 그룹화를 하거나 변수를 작은 문 그룹으로 지역화할 수 있습니다.

```
[ <<label>> ]
[ DECLARE
  declarations ]
BEGIN
  statements
END [ label ];
```

블록 앞에 오는 선언 섹션에 선언된 변수는 블록이 입력될 때마다 기본값으로 초기화됩니다. 다시 말해 함수 호출당 한 번만 초기화되지 않습니다.

다음은 그 한 예입니다.

```
CREATE PROCEDURE update_value() AS $$
DECLARE
  value integer := 20;
BEGIN
  RAISE NOTICE 'Value here is %', value;  -- Value here is 20
  value := 50;
  --
  -- Create a subblock
  --
  DECLARE
    value integer := 80;
  BEGIN
    RAISE NOTICE 'Value here is %', value;  -- Value here is 80
  END;

  RAISE NOTICE 'Value here is %', value;  -- Value here is 50
END;
$$ LANGUAGE plpgsql;
```

레이블을 사용하여 EXIT 문에 사용할 블록을 식별하거나 블록에 선언된 변수의 이름을 한정합니다.

PL/pgSQL의 문을 그룹화하기 위한 BEGIN/END 사용을 트랜잭션 제어를 위한 데이터베이스 명령과 혼동하지 마십시오. PL/pgSQL의 BEGIN 및 END는 그룹화에만 사용되며, 트랜잭션을 시작하거나 종료하지 않습니다.

변수 선언

블록의 DECLARE 섹션에서 루프 변수를 제외하고 블록의 모든 변수를 선언합니다. 변수는 모든 유효한 Amazon Redshift 데이터 형식을 사용할 수 있습니다. 지원되는 데이터 형식은 데이터 타입 섹션을 참조하세요.

PL/pgSQL 변수는 Amazon Redshift 지원 데이터 형식과 RECORD 및 refcursor일 수 있습니다. RECORD에 대한 자세한 내용은 레코드 형식 섹션을 참조하세요. refcursor에 대한 자세한 내용은 커서 섹션을 참조하세요.

```
DECLARE
name [ CONSTANT ] type [ NOT NULL ] [ { DEFAULT | := } expression ];
```

다음으로 변수 선언 예를 찾을 수 있습니다.

```
customerID integer;
numberofitems numeric(6);
link varchar;
onerow RECORD;
```

정수 범위를 반복하는 FOR 루프의 루프 변수는 자동으로 정수 변수로 선언됩니다.

DEFAULT 절이 제공된 경우 블록이 입력되면 변수에 할당되는 초기 값을 지정합니다. DEFAULT 절이 제공되지 않은 경우 변수는 SQL NULL 상태로 초기화됩니다. CONSTANT 옵션은 변수가 할당되지 못하게 합니다. 따라서 값이 블록 기간 동안 일정하게 유지됩니다. NOT NULL이 지정되지 않은 경우 null 값을 할당하면 실행 시간 오류가 발생합니다. NOT NULL로 선언된 모든 변수는 null이 아닌 기본값이 지정되어야 합니다.

기본값은 블록이 입력될 때마다 평가됩니다. 예를 들어 timestamp 유형의 변수에 now()를 할당하면 변수가 함수가 미리 컴파일된 시간이 아니라 현재 함수 호출의 시간을 갖게 됩니다.

```
quantity INTEGER DEFAULT 32;
url VARCHAR := 'http://mysite.com';
user_id CONSTANT INTEGER := 10;
```

refcursor 데이터 형식은 저장 프로시저 내 커서 변수의 데이터 형식입니다. refcursor 값은 저장 프로시저 내에서 반환될 수 있습니다. 자세한 내용은 저장 프로시저에서 결과 세트 반환 단원을 참조하십시오.

별칭 선언

저장 프로시저의 서명에 인수 이름이 생략된 경우, 해당 인수의 별칭을 선언할 수 있습니다.

```
name ALIAS FOR $n;
```

기본 제공 변수

다음 기본 제공 변수가 지원됩니다.

- FOUND
- SQLSTATE
- SQLERRM
- GET DIAGNOSTICS integer_var := ROW_COUNT;

FOUND는 부울 형식의 특수 변수입니다. FOUND는 각 프로시저 호출 내에서 false를 시작합니다. FOUND는 다음 형식의 문으로 설정됩니다.

- SELECT INTO

  행을 반환하면 FOUND를 true로 설정하고, 행을 반환하지 않으면 false로 설정합니다.
- UPDATE, INSERT, DELETE

  하나 이상의 행이 영향을 받으면 FOUND를 true로 설정하고, 행이 영향을 받지 않으면 false로 설정합니다.
- FETCH

  행을 반환하면 FOUND를 true로 설정하고, 행을 반환하지 않으면 false로 설정합니다.
- FOR 문

  FOR 문이 한 번 이상 반복되면 FOUND를 true로 설정하고, 그렇지 않으면 false로 설정합니다. 이는 FOR 문의 세 가지 변형인 정수 FOR 루프, 레코드 세트 FOR 루프, 동적 레코드 세트 FOR 루프 모두에 적용됩니다.

  FOR 루프가 종료되면 FOUND가 설정됩니다. 루프 런타임 내부에서 FOUND는 FOR 문에 의해 수정되지 않습니다. 그러나 루프 본문 내 다른 문의 실행에 의해 변경될 수 있습니다.

다음은 그 한 예입니다.

```
CREATE TABLE employee(empname varchar);
CREATE OR REPLACE PROCEDURE show_found()
AS  $$
DECLARE
  myrec record;
BEGIN
  SELECT INTO myrec * FROM employee WHERE empname = 'John';
  IF NOT FOUND THEN
    RAISE EXCEPTION 'employee John not found';
  END IF;
END;
$$ LANGUAGE plpgsql;
```

예외 핸들러 내의 특수 변수 SQLSTATE에는 발생한 예외에 해당하는 오류 코드가 포함되어 있습니다. 특수 변수 SQLERRM에는 해당 예외와 관련된 오류 메시지가 포함되어 있습니다. 이러한 변수는 예외 핸들러 외부에서 정의되지 않으며 사용하면 오류가 발생합니다.

다음은 그 한 예입니다.

```
CREATE OR REPLACE PROCEDURE sqlstate_sqlerrm() AS
$$
BEGIN
  UPDATE employee SET firstname = 'Adam' WHERE lastname = 'Smith';
  EXECUTE 'select invalid';
  EXCEPTION WHEN OTHERS THEN
  RAISE INFO 'error message SQLERRM %', SQLERRM;
  RAISE INFO 'error message SQLSTATE %', SQLSTATE;
END;
$$ LANGUAGE plpgsql;
```

ROW_COUNT는 GET DIAGNOSTICS 명령과 함께 사용됩니다. 이는 SQL 엔진으로 전송된 마지막 SQL 명령에서 처리된 행 수를 보여 줍니다.

다음은 그 한 예입니다.

```
CREATE OR REPLACE PROCEDURE sp_row_count() AS
$$
DECLARE
  integer_var int;
BEGIN
  INSERT INTO tbl_row_count VALUES(1);
  GET DIAGNOSTICS integer_var := ROW_COUNT;
```

```
  RAISE INFO 'rows inserted = %', integer_var;
END;
$$ LANGUAGE plpgsql;
```

레코드 형식

RECORD 형식은 데이터 형식이 아니라, 자리 표시자에 불과합니다. 레코드 형식 변수는 SELECT 또는 FOR 명령 중에 할당된 행의 실제 행 구조를 가정합니다. 레코드 변수의 하위 구조는 값이 할당될 때마다 변경될 수 있습니다. 레코드 변수가 처음 할당될 때까지 하위 구조가 없습니다. 그 안에 있는 필드에 액세스하려고 시도하면 실행 시간 오류가 발생합니다.

```
name RECORD;
```

다음은 그 한 예입니다.

```
CREATE TABLE tbl_record(a int, b int);
INSERT INTO tbl_record VALUES(1, 2);
CREATE OR REPLACE PROCEDURE record_example()
LANGUAGE plpgsql
AS $$
DECLARE
  rec RECORD;
BEGIN
  FOR rec IN SELECT a FROM tbl_record
  LOOP
    RAISE INFO 'a = %', rec.a;
  END LOOP;
END;
$$;
```

지원되는 PL/pgSQL 문

PL/pgSQL 문은 루프 및 조건 표현식을 비롯한 프로시저 구문으로 SQL 명령을 보완하여 논리 흐름을 제어합니다. COPY, UNLOAD, INSERT 등의 데이터 조작 언어(DML)와 CREATE TABLE 등의 데이터 정의 언어(DDL)를 포함한 대부분의 SQL 명령을 사용할 수 있습니다. 포괄적인 SQL 명령 목록은 SQL 명령 섹션을 참조하세요. 또한 Amazon Redshift에서 다음 PL/pgSQL 문이 지원됩니다.

주제

- 대입
- SELECT INTO

- No-op
- Dynamic SQL
- 반환
- 조건부: IF
- 조건부: CASE
- Loops
- 커서
- RAISE
- 트랜잭션 제어

대입

대입문은 값을 변수에 할당합니다. 표현식은 단일 값을 반환해야 합니다.

```
identifier := expression;
```

:= 대신에 대입에 표준이 아닌 =를 사용하는 것도 허용됩니다.

표현식의 데이터 형식이 변수의 데이터 형식과 일치하지 않거나 변수에 크기 또는 정밀도가 있는 경우, 결과 값이 암시적으로 변환됩니다.

다음에 예가 나와 있습니다.

```
customer_number := 20;
tip := subtotal * 0.15;
```

SELECT INTO

SELECT INTO 문은 여러 열(단 행은 하나만)의 결과를 레코드 변수 또는 스칼라 변수 목록에 할당합니다.

```
SELECT INTO target select_expressions FROM ...;
```

앞의 구문에서 *target*는 레코드 변수이거나 간단한 변수 및 레코드 필드의 쉼표로 구분된 목록일 수 있습니다. *select_expressions* 목록과 명령의 나머지는 정규 SQL에서와 동일합니다.

변수 목록이 *target*로 사용되는 경우, 선택한 값이 대상의 구조와 정확히 일치해야 합니다. 그렇지 않으면 실행 시간 오류가 발생합니다. 레코드 변수가 대상인 경우, 쿼리 결과 열의 행 형식으로 자동 구성됩니다.

INTO 절은 SELECT 문의 거의 모든 곳에서 나타날 수 있습니다. 보통 SELECT 절 바로 뒤나 FROM 절 바로 앞에 나타납니다. 즉, *select_expressions* 목록 바로 앞이나 바로 뒤에 나타납니다.

쿼리가 행을 반환하지 않으면 NULL 값이 *target*에 할당됩니다. 쿼리가 행을 여러 개 반환하면 첫 번째 행이 *target*에 할당되고 나머지는 삭제됩니다. 명령문에 ORDER BY가 포함되어 있지 않으면 첫 번째 행은 결정적이지 않습니다.

대입이 행을 하나 이상 반환했는지 여부를 확인하려면 특수 FOUND 변수를 사용합니다.

```
SELECT INTO customer_rec * FROM cust WHERE custname = lname;
IF NOT FOUND THEN
  RAISE EXCEPTION 'employee % not found', lname;
END IF;
```

레코드 결과가 null인지 여부를 테스트하려면 IS NULL 조건부를 사용하면 됩니다. 추가 행이 삭제되었는지 확인할 수 있는 방법은 없습니다. 다음 예제에서는 행이 반환되지 않은 경우를 처리합니다.

```
CREATE OR REPLACE PROCEDURE select_into_null(return_webpage OUT varchar(256))
AS $$
DECLARE
  customer_rec RECORD;
BEGIN
  SELECT INTO customer_rec * FROM users WHERE user_id=3;
  IF customer_rec.webpage IS NULL THEN
    -- user entered no webpage, return "http://"
    return_webpage = 'http://';
  END IF;
END;
$$ LANGUAGE plpgsql;
```

No-op

no-op 문(NULL;)은 아무 것도 수행하지 않는 자리 표시자 문입니다. no-op 문은 IF-THEN-ELSE 체인의 한 브랜치가 비어 있음을 나타낼 수 있습니다.

```
NULL;
```

Dynamic SQL

PL/pgSQL 저장 프로시저에서 실행될 때마다 다른 테이블 또는 다른 데이터 형식을 포함할 수 있는 동적 명령을 생성하려면 EXECUTE 문을 사용합니다.

```
EXECUTE command-string [ INTO target ];
```

위의 *command-string*은 실행할 명령을 포함하는 문자열(텍스트 형식)을 생성하는 표현식입니다. 이 *command-string* 값은 SQL 엔진으로 전송됩니다. PL/pgSQL 변수의 대체는 명령 문자열에서 수행되지 않습니다. 변수 값은 생성될 때 명령 문자열에 삽입되어야 합니다.

> **Note**
>
> 동적 SQL 내에서 COMMIT 및 ROLLBACK 문을 사용할 수 없습니다. 저장 프로시저 내의 COMMIT 및 ROLLBACK 문 사용에 대한 자세한 내용은 트랜잭션 관리를 참조하십시오.

동적 명령을 작업할 경우 보통 작은따옴표의 이스케이핑을 처리해야 합니다. 달러 인용을 사용하여 함수 본문에서 고정 텍스트를 따옴표로 묶는 것이 좋습니다. 생성된 쿼리에 삽입할 동적 값에는 따옴표가 포함될 수 있으므로 특별한 처리가 필요합니다. 다음 예제에서는 함수의 달러 인용을 전체로 가정하므로, 따옴표가 큰따옴표일 필요가 없습니다.

```
EXECUTE 'UPDATE tbl SET '
  || quote_ident(colname)
  || ' = '
  || quote_literal(newvalue)
  || ' WHERE key = '
  || quote_literal(keyvalue);
```

앞의 예제에서는 함수 quote_ident(text) 및 quote_literal(text)을 보여 줍니다. 이 예제에서는 열 및 테이블 식별자가 포함된 변수를 quote_ident 함수로 전달합니다. 또한 생성된 명령에 리터럴 문자열이 포함된 변수를 quote_literal 함수로 전달합니다. 두 함수 모두 각각 큰따옴표 또는 작은따옴표로 묶인 입력 텍스트를 반환하기 위한 적절한 단계를 거치며, 포함된 특수 문자는 적절하게 이스케이프됩니다.

달러 인용은 고정 텍스트 인용 시에만 유용합니다. 앞의 예제를 다음 형식으로 작성하지 마십시오.

```
EXECUTE 'UPDATE tbl SET '
  || quote_ident(colname)
```

```
  || ' = $$'
  || newvalue
  || '$$ WHERE key = '
  || quote_literal(keyvalue);
```

newvalue의 내용에 $$가 포함되면 이 예제가 중단되기 때문입니다. 선택할 수 있는 다른 달러 인용 구분 기호에도 동일한 문제가 적용됩니다. 미리 알려지지 않은 텍스트를 안전하게 인용하려면 quote_literal 함수를 사용합니다.

반환

RETURN 문은 저장 프로시저에서 호출자에게 다시 반환합니다.

```
RETURN;
```

다음은 그 한 예입니다.

```
CREATE OR REPLACE PROCEDURE return_example(a int)
AS $$
BEGIN
  FOR b in 1..10 LOOP
    IF b < a THEN
      RAISE INFO 'b = %', b;
    ELSE
      RETURN;
    END IF;
  END LOOP;
END;
$$ LANGUAGE plpgsql;
```

조건부: IF

IF 조건부 문은 Amazon Redshift가 사용하는 PL/pgSQL 언어에서 다음 형식을 취할 수 있습니다.

- IF ... THEN

  ```
  IF boolean-expression THEN
    statements
  END IF;
  ```

  다음은 그 한 예입니다.

```
IF v_user_id <> 0 THEN
  UPDATE users SET email = v_email WHERE user_id = v_user_id;
END IF;
```

- IF ... THEN ... ELSE

```
IF boolean-expression THEN
  statements
ELSE
  statements
END IF;
```

다음은 그 한 예입니다.

```
IF parentid IS NULL OR parentid = ''
THEN
  return_name = fullname;
  RETURN;
ELSE
  return_name = hp_true_filename(parentid) || '/' || fullname;
  RETURN;
END IF;
```

- IF ... THEN ... ELSIF ... THEN ... ELSE

키워드 ELSIF는 ELSEIF로 표기할 수도 있습니다.

```
IF boolean-expression THEN
  statements
[ ELSIF boolean-expression THEN
  statements
[ ELSIF boolean-expression THEN
  statements
    ...] ]
[ ELSE
  statements ]
END IF;
```

다음은 그 한 예입니다.

```
IF number = 0 THEN
```

```
  result := 'zero';
ELSIF number > 0 THEN
  result := 'positive';
ELSIF number < 0 THEN
  result := 'negative';
ELSE
  -- the only other possibility is that number is null
  result := 'NULL';
END IF;
```

조건부: CASE

CASE 조건부 문은 Amazon Redshift가 사용하는 PL/pgSQL 언어에서 다음 형식을 취할 수 있습니다.

- 단순 CASE

```
CASE search-expression
WHEN expression [, expression [ ... ]] THEN
  statements
[ WHEN expression [, expression [ ... ]] THEN
  statements
  ... ]
[ ELSE
  statements ]
END CASE;
```

단순 CASE 문은 피연산자의 동등성을 기반으로 조건부 실행을 제공합니다.

search-expression 값은 한 번 평가되고 WHEN 절의 각 *expression*과 연속적으로 비교됩니다. 일치하는 항목이 있으면 해당 *statements*가 실행된 후 제어가 END CASE 뒤의 다음 문으로 전달됩니다. 후속 WHEN 표현식은 평가되지 않습니다. 일치하는 항목이 없으면 ELSE *statements*가 실행됩니다. 그러나 ELSE가 없으면 CASE_NOT_FOUND 예외가 발생합니다.

다음은 그 한 예입니다.

```
CASE x
WHEN 1, 2 THEN
  msg := 'one or two';
ELSE
  msg := 'other value than one or two';
```

```
END CASE;
```

- 검색 CASE

```
CASE
WHEN boolean-expression THEN
  statements
[ WHEN boolean-expression THEN
  statements
  ... ]
[ ELSE
  statements ]
END CASE;
```

검색된 형태의 CASE는 부울 표현식의 사실성을 기반으로 조건부 실행을 제공합니다.

각 WHEN 절의 *boolean-expression*은 true를 생성하는 항목이 발견될 때까지 평가됩니다. 그런 다음 해당 문이 실행되고, 제어가 END CASE 뒤의 다음 문으로 전달됩니다. 후속 WHEN *expressions*는 평가되지 않습니다. true 결과가 없으면 ELSE *statements*가 실행됩니다. 그러나 ELSE가 없으면 CASE_NOT_FOUND 예외가 발생합니다.

다음은 그 한 예입니다.

```
CASE
WHEN x BETWEEN 0 AND 10 THEN
  msg := 'value is between zero and ten';
WHEN x BETWEEN 11 AND 20 THEN
  msg := 'value is between eleven and twenty';
END CASE;
```

Loops

loop 문은 Amazon Redshift가 사용하는 PL/pgSQL 언어에서 다음 형식을 취할 수 있습니다.

- 단순 루프

```
[<<label>>]
LOOP
  statements
END LOOP [ label ];
```

단순 루프는 EXIT 또는 RETURN 문에 의해 종료될 때까지 무기한으로 반복되는 무조건부 루프를 정의합니다. 선택적 레이블은 중첩 루프 내의 EXIT 및 CONTINUE 문에서 EXIT 및 CONTINUE 문이 참조하는 루프를 지정하는 데 사용할 수 있습니다.

다음은 그 한 예입니다.

```
CREATE OR REPLACE PROCEDURE simple_loop()
LANGUAGE plpgsql
AS $$
BEGIN
  <<simple_while>>
  LOOP
    RAISE INFO 'I am raised once';
    EXIT simple_while;
    RAISE INFO 'I am not raised';
  END LOOP;
  RAISE INFO 'I am raised once as well';
END;
$$;
```

- 종료 루프

```
EXIT [ label ] [ WHEN expression ];
```

*label*이 없는 경우 가장 안쪽의 루프가 종료되고 END LOOP 다음의 문이 다음에 실행됩니다. *label*이 있는 경우 중첩 루프 또는 블록의 현재 또는 일부 외부 수준의 레이블이어야 합니다. 그런 다음 명명된 루프 또는 블록이 종료되고 제어가 해당 루프 또는 블록 END 다음에 있는 문으로 계속됩니다.

WHEN이 지정된 경우 *expression*이 true인 경우에만 루프 종료가 발생합니다. 그렇지 않은 경우 제어가 EXIT 뒤의 문으로 전달됩니다.

모든 형식의 루프에서 EXIT를 사용할 수 있으며, 무조건부 루프에서 사용하는 것에 국한되지 않습니다.

BEGIN 블록과 함께 사용하는 경우, EXIT는 블록 종료 후 제어를 다음 문으로 전달합니다. 이 목적으로 레이블을 사용해야 합니다. 레이블이 지정되지 않은 EXIT는 BEGIN 블록과 일치하는 것으로 간주되지 않습니다.

다음은 그 한 예입니다.

```
CREATE OR REPLACE PROCEDURE simple_loop_when(x int)
LANGUAGE plpgsql
AS $$
DECLARE i INTEGER := 0;
BEGIN
  <<simple_loop_when>>
  LOOP
    RAISE INFO 'i %', i;
    i := i + 1;
    EXIT simple_loop_when WHEN (i >= x);
  END LOOP;
END;
$$;
```

- 계속 루프

```
CONTINUE [ label ] [ WHEN expression ];
```

*label*이 제공되지 않은 경우, 실행이 가장 안쪽에 있는 루프의 다음 반복으로 이동합니다. 즉, 루프 본문에 남아 있는 모든 문을 건너뜁니다. 그런 다음 제어는 루프 제어 표현식(있는 경우)으로 돌아가 다른 루프 반복이 필요한지 여부를 결정합니다. *label*이 있는 경우, 실행이 계속되는 루프의 레이블을 지정합니다.

WHEN이 지정된 경우 *expression*이 true인 경우에만 루프의 다음 반복이 시작됩니다. 그렇지 않은 경우 제어가 CONTINUE 뒤의 문으로 전달됩니다.

모든 형식의 루프에서 CONTINUE를 사용할 수 있으며, 무조건부 루프에서 사용하는 것에 국한되지 않습니다.

```
CONTINUE mylabel;
```

- WHILE 루프

```
[<<label>>]
WHILE expression LOOP
  statements
END LOOP [ label ];
```

WHILE 문은 *boolean-expression*이 true로 평가되는 한 일련의 문을 반복합니다. 표현식은 루프 본문의 각 항목 바로 앞에서 검사됩니다.

다음은 그 한 예입니다.

```
WHILE amount_owed > 0 AND gift_certificate_balance > 0 LOOP
  -- some computations here
END LOOP;

WHILE NOT done LOOP
  -- some computations here
END LOOP;
```

- FOR 루프(정수 변형)

```
[<<label>>]
FOR name IN [ REVERSE ] expression .. expression LOOP
  statements
END LOOP [ label ];
```

FOR 루프(정수 변형)는 정수 값 범위를 반복하는 루프를 생성합니다. 변수 이름은 정수 유형으로 자동 정의되며 루프 내부에만 있습니다. 변수 이름의 모든 기존 정의는 루프 내에서 무시됩니다. 범위의 하한값과 상한값을 제공하는 두 표현식은 루프를 입력할 때 한 번 평가됩니다. REVERSE를 지정하면 각 반복 후에 단계 값이 추가되는 것이 아니라 빠집니다.

하한값이 상한값보다 크면(REVERSE의 경우 작으면) 루프 본문이 실행되지 않습니다. 오류는 발생하지 않습니다.

레이블이 FOR 루프에 연결되어 있는 경우, 해당 레이블을 사용하여 정규화된 이름이 있는 정수 루프 변수를 참조할 수 있습니다.

다음은 그 한 예입니다.

```
FOR i IN 1..10 LOOP
  -- i will take on the values 1,2,3,4,5,6,7,8,9,10 within the loop
END LOOP;

FOR i IN REVERSE 10..1 LOOP
  -- i will take on the values 10,9,8,7,6,5,4,3,2,1 within the loop
END LOOP;
```

- FOR 루프(결과 세트 변형)

```
[<<label>>]
FOR target IN query LOOP
  statements
END LOOP [ label ];
```

*target*는 레코드 변수 또는 스칼라 변수의 쉼표로 구분된 목록입니다. 대상은 쿼리에서 생성된 각 행에 연속적으로 할당되고, 행별로 루프 본문이 실행됩니다.

FOR 루프(결과 세트 변형)를 통해 저장 프로시저는 쿼리 결과를 반복하고 그에 따라 해당 데이터를 조작할 수 있습니다.

다음은 그 한 예입니다.

```
CREATE PROCEDURE cs_refresh_reports() AS $$
DECLARE
  reports RECORD;
BEGIN
  FOR reports IN SELECT * FROM cs_reports ORDER BY sort_key LOOP
    -- Now "reports" has one record from cs_reports
    EXECUTE 'INSERT INTO ' || quote_ident(reports.report_name) || ' ' ||
 reports.report_query;
  END LOOP;
  RETURN;
END;
$$ LANGUAGE plpgsql;
```

- 동적 SQL을 사용하는 FOR 루프

```
[<<label>>]
FOR record_or_row IN EXECUTE text_expression LOOP
  statements
END LOOP;
```

동적 SQL을 사용하는 FOR 루프를 통해 저장 프로시저는 동적 쿼리 결과를 반복하고 그에 따라 해당 데이터를 조작할 수 있습니다.

다음은 그 한 예입니다.

```
CREATE OR REPLACE PROCEDURE for_loop_dynamic_sql(x int)
LANGUAGE plpgsql
```

```
AS $$
DECLARE
  rec RECORD;
  query text;
BEGIN
  query := 'SELECT * FROM tbl_dynamic_sql LIMIT ' || x;
  FOR rec IN EXECUTE query
  LOOP
    RAISE INFO 'a %', rec.a;
  END LOOP;
END;
$$;
```

커서

한 번에 전체 쿼리를 실행하는 대신에 커서를 설정할 수 있습니다. 커서는 쿼리를 캡슐화하고 쿼리 결과를 한 번에 몇 행씩 읽습니다. 이렇게 하는 이유는 결과에 다수의 행이 포함된 경우 메모리 오버런을 방지하기 위함입니다. 또 다른 이유는 호출자가 행을 읽을 수 있도록 저장 프로시저가 생성한 커서 참조를 반환하는 것입니다. 이를 통해 저장 프로시저에서 대량의 행 세트를 효율적으로 반환할 수 있습니다.

NONATOMIC 저장 프로시저에서 커서를 사용하려면 START TRANSACTION...COMMIT 사이에 커서 루프를 배치하세요.

커서를 설정하려면 먼저 커서 변수를 선언합니다. PL/pgSQL의 커서에 대한 모든 액세스는 항상 특수 데이터 형식 `refcursor`인 커서 변수를 통과합니다. `refcursor` 데이터 형식에는 커서 참조가 있습니다.

형식 `refcursor`의 변수로 선언하여 커서 변수를 생성할 수 있습니다. 또는 다음과 같은 커서 선언 구문을 사용할 수 있습니다.

```
name CURSOR [ ( arguments ) ] FOR query ;
```

앞의 예에서 *##*(지정된 경우)는 *##*에서 파라미터 값으로 대체될 이름을 각각 정의하는 쉼표로 구분된 *## ### ##* 페어 목록입니다. 이러한 이름을 대체할 실제 값은 커서가 열릴 때 나중에 지정됩니다.

다음에 예가 나와 있습니다.

```
DECLARE
  curs1 refcursor;
  curs2 CURSOR FOR SELECT * FROM tenk1;
```

```
  curs3 CURSOR (key integer) IS SELECT * FROM tenk1 WHERE unique1 = key;
```

이러한 세 변수는 모두 데이터 형식이 refcursor이지만, 첫 번째 변수를 모든 쿼리에 사용할 수 있습니다. 이와 달리 두 번째 변수에는 완전히 지정된 쿼리가 이미 바인딩되어 있고, 마지막 변수에는 파라미터가 있는 쿼리가 바인딩되어 있습니다. 커서가 열리면 key 값은 정수 파라미터 값으로 대체됩니다. 변수 curs1은 어떤 특정 쿼리에도 바인딩되어 있지 않으므로 언바운드되었다고 합니다.

커서를 사용하여 행을 검색하려면 먼저 커서를 열어야 합니다. PL/pgSQL에는 세 가지 형태의 OPEN 문이 있습니다. 이중 두 개는 바인딩되지 않은 커서 변수를 사용하고 세 번째는 바인딩된 커서 변수를 사용합니다.

- 선택을 위해 열기: 커서 변수가 열리고 지정된 쿼리가 실행됩니다. 커서는 아직 열 수 없습니다. 또한 바인딩되지 않은 커서(즉, 단순 refcursor 변수로)로 선언되어 있어야 합니다. SELECT 쿼리는 PL/pgSQL의 다른 SELECT 문과 동일한 방식으로 취급됩니다.

  ```
  OPEN cursor_name FOR SELECT ...;
  ```

  다음은 그 한 예입니다.

  ```
  OPEN curs1 FOR SELECT * FROM foo WHERE key = mykey;
  ```

- 실행을 위해 열기: 커서 변수가 열리고 지정된 쿼리가 실행됩니다. 커서는 아직 열 수 없습니다. 또한 바인딩되지 않은 커서(즉, 단순 refcursor 변수로)로 선언되어 있어야 합니다. 쿼리가 EXECUTE 명령에서와 동일한 방식으로 문자열 표현식으로 지정됩니다. 쿼리가 실행마다 다를 수 있으므로 이 접근 방식은 유연성을 제공합니다.

  ```
  OPEN cursor_name FOR EXECUTE query_string;
  ```

  다음은 그 한 예입니다.

  ```
  OPEN curs1 FOR EXECUTE 'SELECT * FROM ' || quote_ident($1);
  ```

- 바인딩된 커서 열기: 이 형태의 OPEN은 쿼리가 선언될 때 쿼리가 바인딩된 커서 변수를 여는 데 사용됩니다. 커서는 아직 열 수 없습니다. 실제 인수 값 표현식의 목록은 커서가 인수를 사용하도록 선언된 경우에만 나타나야 합니다. 이러한 값은 쿼리에서 대체됩니다.

  ```
  OPEN bound_cursor_name [ ( argument_values ) ];
  ```

다음은 그 한 예입니다.

```
OPEN curs2;
OPEN curs3(42);
```

커서가 열린 후 다음에 설명된 문을 사용하여 작업할 수 있습니다. 이러한 문은 커서을 연 동일한 저장 프로시저에서 발생하지 않아도 됩니다. 저장 프로시저에서 `refcursor` 값을 반환하고 호출자가 커서에서 작업하도록 할 수 있습니다. 모든 포털은 트랜잭션 종료 시 암시적으로 닫힙니다. 따라서 트랜잭션이 종료되기 전까지만 `refcursor` 값을 사용하여 열린 커서를 참조할 수 있습니다.

- FETCH는 다음 행을 커서에서 대상으로 가져옵니다. 이 대상은 SELECT INTO와 마찬가지로 행 변수, 레코드 변수 또는 간단한 변수의 쉼표로 구분된 목록이 될 수 있습니다. SELECT INTO와 마찬가지로 특수 변수 FOUND를 확인하여 행을 가져왔는지 여부를 알 수 있습니다.

  ```
  FETCH cursor INTO target;
  ```

  다음은 그 한 예입니다.

  ```
  FETCH curs1 INTO rowvar;
  ```

- CLOSE는 열린 커서의 기본 포털을 닫습니다. 이 문을 사용하여 트랜잭션의 종료 이전에 리소스를 해제할 수 있습니다. 또한 이 문을 사용하여 커서 변수가 다시 열리도록 할 수 있습니다.

  ```
  CLOSE cursor;
  ```

  다음은 그 한 예입니다.

  ```
  CLOSE curs1;
  ```

RAISE

`RAISE level` 문을 사용하여 메시지를 보고하고 오류를 발생시킵니다.

```
RAISE level 'format' [, variable [, ...]];
```

가능한 수준은 NOTICE, INFO, LOG, WARNING, EXCEPTION입니다. EXCEPTION은 현재 트랜잭션을 정상적으로 취소하는 오류를 발생시킵니다. 다른 수준은 다른 우선 순위 수준의 메시지만 생성합니다.

형식 문자열 내부의 %는 다음 선택적 인수의 문자열 표현으로 대체됩니다. %%를 작성하여 리터럴 %를 내보냅니다. 현재 선택적 인수는 표현식이 아닌 간단한 변수여야 하고, 형식은 간단한 문자열 리터럴이어야 합니다.

다음 예제에서는 v_job_id 값이 문자열에서 %를 대체합니다.

```
RAISE NOTICE 'Calling cs_create_job(%)', v_job_id;
```

RAISE 문을 사용하여 예외 처리 블록에서 포착된 예외를 다시 발생시킵니다. 이 문은 NONATOMIC 모드 저장 프로시저의 예외 처리 블록에서만 유효합니다.

```
RAISE;
```

트랜잭션 제어

Amazon Redshift가 사용하는 PL/pgSQL 언어의 트랜잭션 제어 문을 사용할 수 있습니다. 저장 프로시저 내의 COMMIT, ROLLBACK, TRUNCATE 사용에 대한 자세한 내용은 트랜잭션 관리를 참조하십시오.

NONATOMIC 모드 저장 프로시저에서는 트랜잭션 블록을 시작하는 데 START TRANSACTION을 사용합니다.

```
START TRANSACTION;
```

> **Note**
>
> PL/pgSQL 문 START TRANSACTION은 다음과 같은 점에서 SQL 문 START TRANSACTION과 다릅니다.
>
> - 저장 프로시저 내에서 START TRANSACTION은 BEGIN과 동의어가 아닙니다.
> - PL/pgSQL 문은 선택적 격리 수준 및 액세스 권한 키워드를 지원하지 않습니다.

Amazon Redshift의 구체화된 뷰

이 섹션에서는 Amazon Redshift에서 구체화된 뷰를 생성하고 사용하는 방법을 설명합니다. 구체화된 뷰는 쿼리 결과를 저장하는 데이터베이스 객체로, 성능과 효율성을 개선하는 데 사용할 수 있습니다.

데이터 웨어하우스 환경에서 애플리케이션은 대형 테이블에 대해 복잡한 쿼리를 수행해야 하는 경우가 많습니다. 수십억 개의 행이 포함된 테이블에서 다중 테이블 조인 및 집계를 수행하는 SELECT 문을 예로 들 수 있습니다. 이러한 쿼리를 처리하면 결과를 컴퓨팅하는 데 시스템 리소스 및 시간이 많이 필요할 수 있습니다.

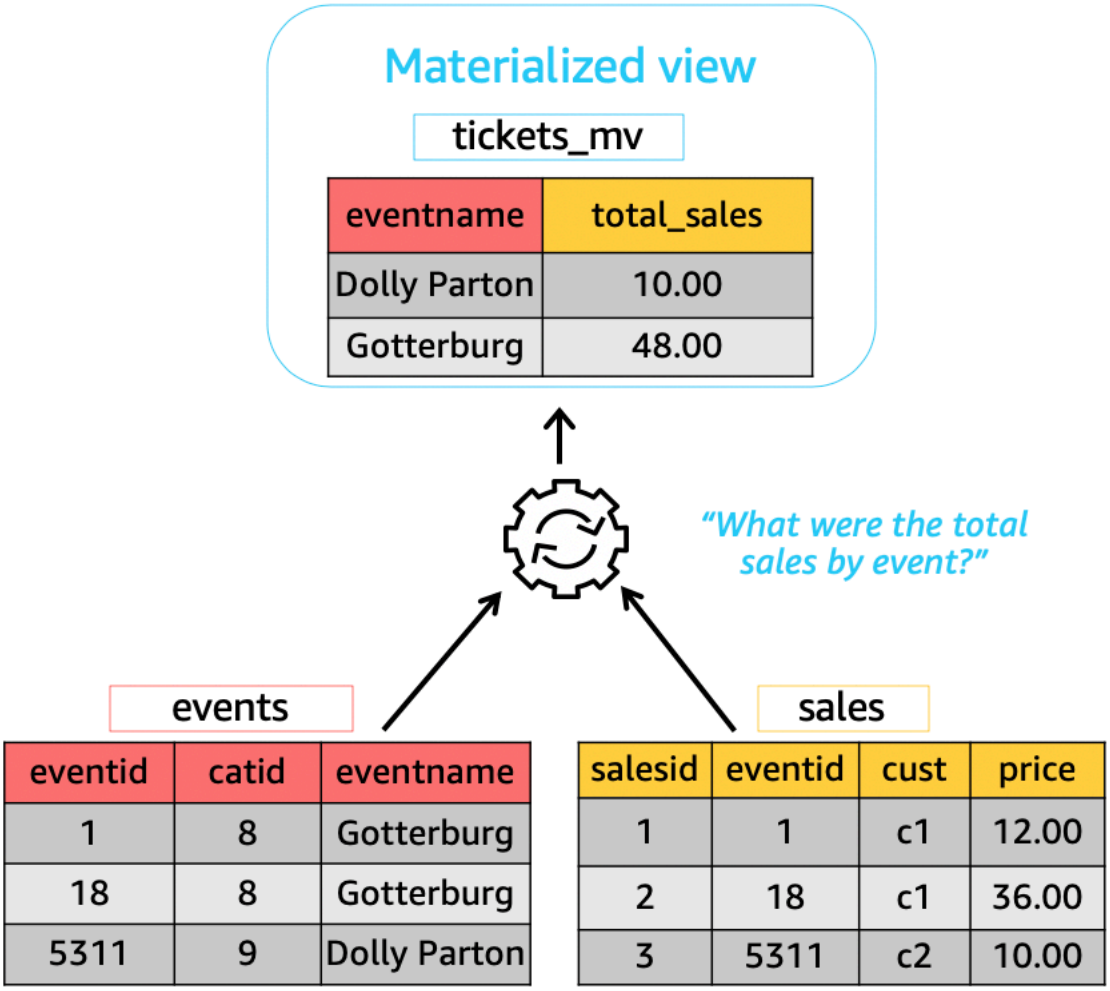
Amazon Redshift의 구체화된 뷰를 통해 이러한 문제를 해결할 수 있습니다. 구체화된 보기에는 하나 이상의 기본 테이블에 대한 SQL 쿼리를 기반으로 사전 계산된 결과 집합이 포함됩니다. 데이터베이스의 다른 테이블이나 뷰를 쿼리할 때와 같은 방식으로 SELECT 문을 실행하여 구체화된 뷰를 쿼리할 수 있습니다. Amazon Redshift는 기본 테이블에 전혀 액세스할 필요 없이 구체화된 뷰에서 미리 계산된 결과를 반환합니다. 사용자 입장에서는 기본 테이블에서 동일한 데이터를 검색할 때보다 쿼리 결과가 훨씬 빠르게 반환됩니다.

구체화된 보기는 예측 가능하고 반복되는 쿼리 속도를 높이는 데 특히 유용합니다. 애플리케이션은 대용량 테이블에 대해 리소스 집약적인 쿼리(예: 집계 또는 다중 조인)를 수행하는 대신 구체화된 뷰를 쿼리하여 사전 계산된 결과 집합을 검색할 수 있습니다. 예를 들어 쿼리 집합이 Amazon QuickSight와 같은 대시시보드를 채우는 데 사용되는 시나리오를 생각해 봅니다. 이 사용 사례는 쿼리를 예측하고 반복할 수 있기 때문에 구체화된 보기에 이상적입니다.

다른 구체화된 뷰와 관련하여 구체화된 뷰를 정의할 수 있습니다. 구체화된 뷰에 구체화된 뷰를 사용하여 구체화된 뷰의 기능을 확장합니다. 이 접근 방식에서 기존의 구체화된 뷰는 쿼리가 데이터를 검색하기 위한 기본 테이블과 동일한 역할을 합니다.

이 접근 방식은 다른 집계 또는 GROUP BY 옵션에 대해 미리 계산된 조인을 재사용하는 데 특히 유용합니다. 예를 들어 고객 정보(수백만 개의 행 포함)를 항목 주문 세부 정보(수십억 개의 행 포함)와 조인하는 구체화된 뷰를 사용합니다. 이는 요청 시 반복적으로 계산하는 데 비용이 많이 드는 쿼리입니다. 이 구체화된 뷰 위에 생성된 구체화된 뷰에 대해 다른 GROUP BY 옵션을 사용하고 다른 테이블과 조인할 수 있습니다. 이렇게 하면 매번 값비싼 기본 조인을 실행하는 데 사용되는 계산 시간이 절약됩니다. STV_MV_DEPS 테이블은 다른 구체화된 뷰에 대한 구체화된 뷰의 종속성을 보여줍니다.

구체화된 뷰를 생성할 때 Amazon Redshift는 사용자 지정 SQL 문을 실행하여 기본 테이블에서 데이터를 수집하고 결과 집합을 저장합니다. 다음 그림에서는 SQL 쿼리가 2개의 기본 테이블 `events` 및 `sales`를 사용하여 정의하는 구체화된 뷰 `tickets_mv`에 대한 개요를 제공합니다.



그런 다음 쿼리에 이러한 구체화된 뷰를 사용하여 속도를 높일 수 있습니다. 또한 Amazon Redshift는 쿼리가 구체화된 뷰를 명시적으로 참조하지 않는 경우에도 구체화된 뷰를 사용하도록 이러한 쿼리를 자동으로 다시 작성할 수 있습니다. 쿼리의 자동 재작성은 구체화된 뷰를 사용하도록 쿼리를 변경할 수 없을 때 성능을 향상시키는 데 특히 유용합니다.

구체화된 뷰에서 데이터를 업데이트하기 위해 언제든지 REFRESH MATERIALIZED VIEW 문을 사용하여 구체화된 뷰를 수동으로 새로 고칠 수 있습니다. Amazon Redshift가 기본 테이블에서 발생한 변경 사항을 식별한 다음 해당 변경 사항을 구체화된 뷰에 적용합니다. 쿼리를 자동으로 재작성하려면 구체화된 보기가 최신 상태여야 하므로 구체화된 보기 소유자는 기본 테이블이 변경될 때마다 구체화된 보기를 새로 고쳐야 합니다.

Amazon Redshift는 자동 재작성을 위해 구체화된 보기를 최신 상태로 유지하는 몇 가지 방법을 제공합니다. 구체화된 뷰의 기본 테이블이 업데이트될 때 구체화된 뷰를 새로 고치도록 자동 새로 고침 옵션으로 구체화된 뷰를 구성할 수 있습니다. 이 자동 새로 고침 작업은 다른 워크로드에 대한 중단을 최소화하기 위해 클러스터 리소스를 사용할 수 있을 때 실행됩니다. 자동 새로 고침 예약은 워크로드에

따라 달라지므로 Amazon Redshift가 구체화된 뷰를 새로 고치는 시기를 더 잘 제어할 수 있습니다. Amazon Redshift 스케줄러 API 및 콘솔 통합을 사용하여 구체화된 뷰 새로 고침 작업을 예약할 수 있습니다. 쿼리 예약에 대한 자세한 내용은 Amazon Redshift 콘솔에서 쿼리 예약 섹션을 참조하세요.

이렇게 하면 구체화된 뷰의 최신 데이터에 대한 서비스 수준 계약(SLA) 요구 사항이 있는 경우에 특히 유용합니다. 자동으로 새로 고칠 수 있는 구체화된 뷰를 수동으로 새로 고칠 수도 있습니다. 구체화된 뷰를 생성하는 방법에 대한 자세한 내용은 CREATE MATERIALIZED VIEW 섹션을 참조하세요.

SELECT 문을 실행하여 구체화된 보기를 쿼리할 수 있습니다. 구체화된 뷰를 쿼리하는 방법에 대한 자세한 내용은 구체화된 뷰 쿼리 섹션을 참조하세요. 기본 테이블에서 데이터를 삽입, 업데이트 및 삭제하면 결과 집합은 결국 효력을 상실합니다. 구체화된 보기를 언제든지 새로 고쳐 기본 테이블의 최신 변경 사항으로 업데이트할 수 있습니다. 구체화된 뷰를 새로 고치는 방법에 대한 자세한 내용은 REFRESH MATERIALIZED VIEW 섹션을 참조하세요.

구체화된 보기를 생성 및 관리할 때 사용되는 SQL 명령에 대한 자세한 내용은 다음 명령 주제를 참조하십시오.

- CREATE MATERIALIZED VIEW
- ALTER MATERIALIZED VIEW
- REFRESH MATERIALIZED VIEW
- DROP MATERIALIZED VIEW

구체화된 보기를 모니터링하기 위한 시스템 테이블 및 보기에 대한 자세한 내용은 다음 주제를 참조하십시오.

- STV_MV_INFO
- STL_MV_STATE
- SVL_MV_REFRESH_STATUS
- STV_MV_DEPS

주제

- 구체화된 뷰 쿼리
- 구체화된 뷰를 사용하기 위한 자동 쿼리 재작성
- Amazon Redshift Spectrum의 외부 데이터 레이크 테이블에 대한 구체화된 뷰
- 구체화된 뷰 새로 고침
- 자동화된 구체화된 보기

- 구체화된 뷰의 사용자 정의 함수(UDF) 사용
- 구체화된 뷰로 스트리밍 모으기

구체화된 뷰 쿼리

구체화된 보기는 테이블 또는 표준 보기와 같이 구체화된 보기 이름을 데이터 원본으로 참조하여 모든 SQL 쿼리에서 사용할 수 있습니다.

쿼리가 구체화된 보기에 액세스하면 가장 최근 새로 고침으로 구체화된 보기에 저장된 데이터만 볼 수 있습니다. 따라서 쿼리는 구체화된 보기의 해당 기본 테이블에서 모든 최신 변경 사항을 볼 수 없습니다.

다른 사용자가 구체화된 보기를 쿼리하려는 경우 구체화된 보기의 소유자는 해당 사용자에게 SELECT 권한을 부여합니다. 다른 사용자가 기본 테이블에 대해 SELECT 권한을 가질 필요는 없습니다. 구체화된 보기의 소유자는 구체화된 보기를 쿼리하지 못하도록 다른 사용자의 SELECT 권한을 취소할 수 있습니다. 다른 사용자에게는 구체화된 뷰의 기본 테이블이 포함된 스키마에 대한 USAGE 권한이 여전히 필요합니다.

구체화된 보기의 소유자가 더 이상 기본 테이블에 대한 SELECT 권한을 가지고 있지 않은 경우입니다.

- 소유자가 더 이상 구체화된 보기를 쿼리할 수 없습니다.
- 구체화된 보기에 대한 SELECT 권한이 있는 다른 사용자가 더 이상 구체화된 보기를 쿼리할 수 없습니다.

다음 예제는 `tickets_mv` 구체화된 보기를 쿼리합니다. 구체화된 뷰를 생성하는 데 사용되는 SQL 명령에 대한 자세한 내용은 CREATE MATERIALIZED VIEW 섹션을 참조하세요.

```
SELECT sold
FROM tickets_mv
WHERE catgroup = 'Concerts';
```

쿼리 결과는 사전 계산되므로 기본 테이블(`category`, `event` 및 `sales`)에 액세스할 필요가 없습니다. Amazon Redshift는 `tickets_mv`에서 직접 결과를 반환할 수 있습니다.

구체화된 뷰를 사용하기 위한 자동 쿼리 재작성

Amazon Redshift에서 구체화된 뷰의 자동 쿼리 재작성을 사용하여 Amazon Redshift가 구체화된 뷰를 사용하도록 쿼리를 재작성하도록 할 수 있습니다. 이렇게 하면 구체화된 뷰를 명시적으로 참조하지 않

는 쿼리의 경우에도 쿼리 워크로드가 가속화됩니다. Amazon Redshift는 쿼리를 재작성할 때 최신 구체화된 보기만 사용합니다.

사용 노트

쿼리 자동 재작성이 쿼리에 사용되는지 확인하려면 쿼리 계획 또는 STL_EXPLAIN을 검사합니다. 다음은 SELECT 문과 원래 쿼리 계획의 EXPLAIN 출력을 보여줍니다.

```
SELECT catgroup, SUM(qtysold) AS sold
FROM category c, event e, sales s
WHERE c.catid = e.catid AND e.eventid = s.eventid
GROUP BY 1;

EXPLAIN
 XN HashAggregate  (cost=920021.24..920021.24 rows=1 width=35)
   ->  XN Hash Join DS_BCAST_INNER  (cost=440004.53..920021.22 rows=4 width=35)
         Hash Cond: ("outer".eventid = "inner".eventid)
         ->  XN Seq Scan on sales s  (cost=0.00..7.40 rows=740 width=6)
         ->  XN Hash  (cost=440004.52..440004.52 rows=1 width=37)
               ->  XN Hash Join DS_BCAST_INNER  (cost=0.01..440004.52 rows=1 width=37)
                     Hash Cond: ("outer".catid = "inner".catid)
                     ->  XN Seq Scan on event e  (cost=0.00..2.00 rows=200 width=6)
                     ->  XN Hash  (cost=0.01..0.01 rows=1 width=35)
                           ->  XN Seq Scan on category c  (cost=0.00..0.01 rows=1
 width=35)
```

다음은 성공적인 자동 재작성 후 EXPLAIN 출력을 보여줍니다. 이 출력에는 원래 쿼리 계획의 일부를 대체하는 쿼리 계획의 구체화된 뷰에 대한 스캔이 포함됩니다.

```
* EXPLAIN
     XN HashAggregate  (cost=11.85..12.35 rows=200 width=41)
       ->  XN Seq Scan on mv_tbl__tickets_mv__0 derived_table1  (cost=0.00..7.90
 rows=790 width=41)
```

자동, 예약 또는 수동과 같은 새로 고침 전략에 관계없이 쿼리의 자동 재작성을 위해 최신(새로운) 구체화된 뷰만 고려됩니다. 따라서 원래 쿼리는 최신 결과를 반환합니다. 구체화된 뷰가 쿼리에서 명시적으로 참조되는 경우 Amazon Redshift는 구체화된 뷰에 현재 저장된 데이터에 액세스합니다. 이 데이터는 구체화된 뷰의 기본 테이블에서 최신 변경 사항을 반영하지 않을 수 있습니다.

클러스터 버전 1.0.20949 이상에서 생성된 구체화된 뷰의 자동 쿼리 재작성을 사용할 수 있습니다.

FALSE로 SET mv_enable_aqmv_for_session을 사용하여 세션 수준에서 자동 쿼리 재작성을 중지할 수 있습니다.

제한 사항

다음은 구체화된 뷰의 자동 쿼리 재작성 사용에 대한 제한 사항입니다.

- 자동 쿼리 재작성은 다음 중 하나를 참조하거나 포함하지 않는 구체화된 뷰에서 작동합니다.
  - 하위 쿼리
  - 왼쪽, 오른쪽 또는 전체 외부 조인
  - 설정 연산
  - SUM, COUNT, MIN, MAX, AVG를 제외한 모든 집계 함수. (이런 집계 함수에서만 자동 쿼리 재작성이 작동합니다.)
  - DISTINCT가 포함된 모든 집계 함수
  - 모든 윈도 함수
  - SELECT DISTINCT 또는 HAVING 절
  - 외부 테이블
  - 다른 구체화된 뷰
- 자동 쿼리 재작성은 사용자 정의 Amazon Redshift 테이블을 참조하는 SELECT 쿼리를 재작성합니다. Amazon Redshift는 다음 쿼리를 재작성하지 않습니다.
  - CREATE TABLE AS 문
  - SELECT INTO 문
  - 카탈로그 또는 시스템 테이블에 대한 쿼리
  - 외부 조인 또는 SELECT DISTINCT 절이 있는 쿼리
- 쿼리가 자동으로 재작성되지 않는 경우 지정된 구체화된 뷰에 대한 SELECT 권한이 있고 mv_enable_aqmv_for_session 옵션이 TRUE로 설정되어 있는지 확인합니다.

  STV_MV_INFO를 검사하여 구체화된 뷰가 쿼리 자동 재작성에 적합한지 확인할 수도 있습니다. 자세한 내용은 STV_MV_INFO 단원을 참조하십시오.

Amazon Redshift Spectrum의 외부 데이터 레이크 테이블에 대한 구체화된 뷰

구체화된 뷰는 외부 데이터 레이크 테이블에 대한 증분 유지 관리를 제공할 수 있습니다. 증분 유지 관리에서는 Amazon Redshift가 마지막 새로 고침 이후 기본 테이블의 데이터 변경 사항만으로 구체화된 뷰의 데이터를 업데이트합니다. 증분 유지 관리는 기본 테이블에서 데이터를 변경할 때마다 구체화된 뷰를 완전히 다시 계산하는 것보다 비용 효율적입니다.

하나 이상의 외부 테이블에서 구체화된 뷰를 사용하는 경우 구체화된 뷰는 다음 항목에 대해 증분 방식으로 생성됩니다.

- 지원되는 모든 형식의 데이터 파일(Parquet, Avro, CSV 등)이 포함된.파티셔닝 및 파티셔닝되지 않은 표준 데이터 레이크 테이블.
- Copy-on-write 및 merge-on-read가 포함된 파티셔닝 및 파티셔닝되지 않는 Apache Iceberg 테이블.
- 동일한 데이터베이스의 Amazon Redshift 테이블과 조인된 Amazon Redshift Spectrum 테이블.

구체화된 뷰는 다음 항목에 대해 증분 방식으로 새로 고쳐집니다.

- 구체화된 뷰가 집계를 수행하지 않는 경우 S3 DELETE 또는 PUT 덮어쓰기(데이터 파일 삭제) 후 표준 데이터 레이크 테이블.
- INSERT, DELETE, UPDATE 또는 테이블 압축 후 Apache Iceberg 테이블.

Amazon Redshift Spectrum에 대한 자세한 내용은 Amazon Redshift Spectrum 섹션을 참조하세요.

제한 사항

구체화된 뷰에 대한 일반적인 제한은 데이터 레이크 테이블의 구체화된 뷰에 여전히 적용됩니다. 자세한 내용은 구체화된 뷰 새로 고침 단원을 참조하십시오. 또한 외부 데이터 레이크 테이블에서 구체화된 뷰를 사용할 때는 다음 제한 사항을 고려하세요.

- 구체화된 뷰는 다음 항목에 대해 비증분식으로 생성됩니다.
  - Hudi 또는 Delta 레이크 테이블.
  - 스펙트럼 중첩 데이터 액세스.
- 다음 항목에 대한 구체화된 뷰 새로 고침은 전체 재계산으로 돌아갑니다.
  - 구체화된 뷰가 집계를 수행하는 경우 필요한 스냅샷이 만료된 후 Apache Iceberg 테이블.

  - 구체화된 뷰가 집계를 수행하는 경우 Amazon S3에서 데이터 파일을 삭제하거나 업데이트한 후 표준 데이터 레이크 테이블.
  - 트랜잭션 블록 내에서 두 번 이상 새로 고쳐진 표준 데이터 레이크 테이블.
  - 매니페스트에서 관리하는 표준 데이터 레이크 테이블. 매니페스트에 대한 자세한 내용은 매니페스트를 사용한 데이터 파일 지정을 참조하세요.
  - Amazon Redshift는 특히 조인을 포함하고 마지막 새로 고침 이후 둘 이상의 기본 테이블이 업데이트된 구체화된 뷰의 경우 성능이 더 좋을 것으로 예상되면 전체 재계산으로 돌아갑니다.
- Apache Iceberg 테이블에서 구체화된 뷰 새로 고침은 단일 데이터 파일에서 삭제된 최대 4백만 개의 위치만 처리할 수 있습니다. 이 한도에 도달한 후 구체화된 뷰를 계속 새로 고치려면 Apache Iceberg 기본 테이블을 압축해야 합니다.
- Apache Iceberg 테이블에서는 구체화된 뷰 생성 및 새로 고침에 동시성 규모 조정이 지원되지 않습니다.
- 자동 제어 기능은 지원되지 않습니다. 여기에는 자동화된 구체화된 뷰 , 자동 새로 고침 및 자동 쿼리 다시 쓰기가 포함됩니다.
- 구체화된 증분 뷰가 새로 고쳐지면 IAM 권한은 Amazon Redshift 기본 테이블의 액세스된 부분에만 적용됩니다.

구체화된 뷰 새로 고침

이 주제에서는 기본 테이블에서 구체화된 뷰의 데이터를 새로 고치는 방법을 설명합니다.

구체화된 보기를 생성할 때 그 콘텐츠는 해당 시점에서 기본 데이터베이스 테이블의 상태를 반영합니다. 애플리케이션이 기본 테이블의 데이터를 변경하더라도 구체화된 보기의 데이터는 변경되지 않습니다. 구체화된 뷰에서 데이터를 업데이트하기 위해 언제든지 `REFRESH MATERIALIZED VIEW` 문을 사용하여 구체화된 뷰를 수동으로 새로 고칠 수 있습니다. 이 문을 사용하면 Amazon Redshift가 기본 테이블 또는 테이블에서 발생한 변경 사항을 식별하고 해당 변경 사항을 구체화된 뷰에 적용합니다.

Amazon Redshift에는 구체화된 뷰 새로 고침을 위한 두 가지 전략이 있습니다.

- 대부분의 경우 Amazon Redshift는 증분 새로 고침을 수행할 수 있습니다. 증분 새로 고침에서는 Amazon Redshift가 마지막 새로 고침 이후 기본 테이블의 데이터 변경 사항을 빠르게 식별하고 구체화된 뷰의 데이터를 업데이트합니다. 증분 새로 고침은 구체화된 뷰를 정의할 때 쿼리에 사용되는 다음 SQL 구문에 지원됩니다.
  - SELECT, FROM, [INNER] JOIN, WHERE, GROUP BY 또는 HAVING 절을 포함하는 구문.
  - SUM, MIN, MAX, AVG, COUNT와 같은 집계를 포함하는 구문.

  - 대부분의 내장 SQL 함수, 특히 변경할 수 없는 함수(입력 인수가 동일하고 항상 동일한 출력을 생성하는 경우).

  증분 새로 고침은 데이터 공유 테이블을 기반으로 한 구체화된 뷰에서 지원됩니다.
- 증분 새로 고침이 불가능하면 Amazon Redshift가 전체 새로 고침을 수행합니다. 전체 새로 고침은 기본 SQL 문을 다시 실행하여 구체화된 보기의 모든 데이터를 대체합니다.
- Amazon Redshift는 구체화된 뷰를 정의하는 데 사용되는 SELECT 쿼리에 따라 구체화된 뷰에 대한 새로 고침 방법을 자동으로 선택합니다.

구체화된 뷰에 대한 구체화된 뷰를 새로 고치는 것은 계단식 프로세스가 아닙니다. 즉, 구체화된 뷰 B에 종속된 구체화된 뷰 A가 있다고 가정합니다. 이 경우 REFRESH MATERIALIZED VIEW A가 호출되면 B가 오래된 경우에도 현재 버전의 B를 사용하여 A가 새로 고쳐집니다. A를 완전히 최신 상태로 만들려면 A를 새로 고치기 전에 먼저 별도의 트랜잭션에서 B를 새로 고칩니다.

다음 예에서는 프로그래밍 방식으로 구체화된 뷰에 대한 전체 새로 고침 계획을 생성하는 방법을 보여줍니다. 구체화된 뷰 v를 새로 고치려면 먼저 구체화된 뷰 u를 새로 고칩니다. 구체화된 뷰 w를 새로 고치려면 먼저 구체화된 뷰 u를 새로 고친 다음 구체화된 뷰 v를 새로 고칩니다.

```
CREATE TABLE t(a INT);
CREATE MATERIALIZED VIEW u AS SELECT * FROM t;
CREATE MATERIALIZED VIEW v AS SELECT * FROM u;
CREATE MATERIALIZED VIEW w AS SELECT * FROM v;

WITH RECURSIVE recursive_deps (mv_tgt, lvl, mv_dep) AS
( SELECT trim(name) as mv_tgt, 0 as lvl, trim(ref_name) as mv_dep
  FROM stv_mv_deps
  UNION ALL
  SELECT R.mv_tgt, R.lvl+1 as lvl, trim(S.ref_name) as mv_dep
  FROM stv_mv_deps S, recursive_deps R
  WHERE R.mv_dep = S.name
)

SELECT mv_tgt, mv_dep from recursive_deps
ORDER BY mv_tgt, lvl DESC;

 mv_tgt | mv_dep
--------+--------
 v      | u
 w      | u
 w      | v
```

```
(3 rows)
```

다음 예에서는 오래된 구체화된 뷰에 종속된 구체화된 뷰에 대해 REFRESH MATERIALIZED VIEW를 실행할 때 정보 메시지를 보여줍니다.

```
create table a(a int);
```

```
create materialized view b as select * from a;
```

```
create materialized view c as select * from b;
```

```
insert into a values (1);
```

```
refresh materialized view c;

INFO:  Materialized view c is already up to date.  However, it depends on another
 materialized view that is not up to date.
```

```
REFRESH MATERIALIZED VIEW b;
INFO:  Materialized view b was incrementally updated successfully.
```

```
REFRESH MATERIALIZED VIEW c;
INFO:  Materialized view c was incrementally updated successfully.
```

Amazon Redshift에는 현재 구체화된 뷰의 증분 새로 고침에 대한 다음과 같은 제한 사항이 있습니다.

Amazon Redshift는 다음 SQL 요소를 사용하는 쿼리로 정의된 구체화된 뷰에 대해 증분 새로 고침을 지원하지 않습니다.

- OUTER JOIN(RIGHT, LEFT 또는 FULL).
- 설정 연산 UNION, INTERSECT, EXCEPT 및 MINUS.
- 집계 함수 MEDIAN, PERCENTILE_CONT, LISTAGG, STDDEV_SAMP, STDDEV_POP, APPROXIMATE COUNT, APPROXIMATE PERCENTILE 및 비트 단위 집계 함수.

> **Note**
>
> COUNT, SUM, AVG 집계 함수가 지원됩니다.

- DISTINCT COUNT, DISTINCT SUM 등과 같은 DISTINCT 집계 함수.
- 창 함수.
- 일반 하위 표현식 최적화와 같은 쿼리 최적화를 위해 임시 테이블을 사용하는 쿼리입니다.
- 하위 쿼리.
- 구체화된 뷰를 정의하는 쿼리에서 다음 형식을 참조하는 외부 테이블
  - Delta Lake
  - Hudi

  위에 나열된 형식 외의 다른 형식을 사용하는 정의된 구체화된 뷰에 대한 증분 새로 고침이 지원됩니다. 자세한 내용은 Amazon Redshift Spectrum의 외부 데이터 레이크 테이블에 대한 구체화된 뷰 단원을 참조하십시오.

구체화된 뷰 자동 새로 고침

Amazon Redshift는 구체화된 뷰가 자동 새로 고침 옵션을 사용하여 생성되거나 변경될 때 기본 테이블의 최신 데이터로 구체화된 뷰를 자동으로 새로 고칠 수 있습니다. Amazon Redshift는 기본 테이블이 변경된 후 가능한 한 빨리 구체화된 뷰를 자동으로 새로 고칩니다.

클러스터의 활성 워크로드에 미치는 영향을 최소화하면서 가장 중요한 구체화된 뷰의 새로 고침을 완료하기 위해 Amazon Redshift는 여러 요소를 고려합니다. 이러한 요소에는 현재 시스템 로드, 새로 고침에 필요한 리소스, 사용 가능한 클러스터 리소스 및 구체화된 뷰가 사용되는 빈도가 포함됩니다.

Amazon Redshift는 자동 새로 고침보다 워크로드의 우선 순위를 지정하고 사용자 워크로드의 성능을 유지하기 위해 자동 새로 고침을 중지할 수 있습니다. 이 접근 방식은 일부 구체화된 뷰의 새로 고침을 지연시킬 수 있습니다. 경우에 따라 구체화된 뷰에 대해 보다 결정적인 새로 고침 동작이 필요할 수 있습니다. 이 경우에는 REFRESH MATERIALIZED VIEW에 설명된 대로 수동 새로 고침을 사용하거나 Amazon Redshift 스케줄러 API 작업 또는 콘솔을 사용하여 예약된 새로 고침을 사용하는 것이 좋습니다.

CREATE MATERIALIZED VIEW를 사용하여 구체화된 뷰에 대한 자동 새로 고침을 설정할 수 있습니다. 또한 AUTO REFRESH 절을 사용하여 구체화된 뷰를 자동으로 새로 고칠 수 있습니다. 구체화된 뷰 생성에 대한 자세한 내용은 CREATE MATERIALIZED VIEW 섹션을 참조하세요. ALTER MATERIALIZED VIEW로 현재 구체화된 보기에 자동 새로 고침을 사용할 수 있습니다.

구체화된 뷰를 새로 고칠 때 다음 사항을 고려하세요.

- 구체화된 뷰에 자동 새로 고침을 사용하지 않아도 REFRESH MATERIALIZED VIEW 명령을 사용하여 구체화된 뷰를 명시적으로 새로 고칠 수 있습니다.

- Amazon Redshift는 외부 테이블에 정의된 구체화된 뷰를 자동으로 새로 고치지 않습니다.
- 새로 고침 상태의 경우 사용자가 시작했거나 자동으로 새로 고친 쿼리를 기록하는 SVL_MV_REFRESH_STATUS를 확인할 수 있습니다.
- 재계산 전용 구체화된 보기에서 REFRESH를 실행하려면 스키마에 대한 CREATE 권한이 있는지 확인합니다. 자세한 내용은 GRANT 단원을 참조하십시오.

자동화된 구체화된 보기

이 주제에서는 Amazon Redshift가 자동화된 구체화된 뷰를 사용하여 성능을 개선하는 방법을 설명합니다. Amazon Redshift는 데이터베이스 활동 및 성능을 기반으로 구체화된 뷰를 자동으로 생성합니다. Amazon Redshift는 기본적으로 자동화된 구체화된 뷰를 사용합니다.

구체화된 보기는 Amazon Redshift에서 쿼리 성능 개선을 위한 강력한 도구입니다. 미리 계산된 결과 집합을 저장하여 이를 수행합니다. 유사한 쿼리는 기존 결과 집합에서 레코드를 검색할 수 있으므로 매번 동일한 논리를 다시 실행할 필요가 없습니다. 개발자와 분석가는 워크로드를 분석한 후 구체화된 보기를 생성하여 어떤 쿼리에 도움이 될지, 각 구체화된 보기의 유지 관리 비용이 가치가 있는지 여부를 결정합니다. 워크로드가 증가하거나 변경됨에 따라 이러한 구체화된 보기를 검토하여 실질적인 성능 이점을 계속 제공하는지 확인해야 합니다.

Redshift의 AutoMV(Automated Materialized Views) 기능은 사용자가 생성한 구체화된 보기와 동일한 성능 이점을 제공합니다. Amazon Redshift는 기계 학습을 사용하여 워크로드를 지속적으로 모니터링하고, 유용한 경우 새로운 구체화된 뷰를 생성합니다. AutoMV는 구체화된 보기를 생성하고 최신 상태로 유지하는 비용과 쿼리 대기 시간에 대한 예상 이점의 균형을 맞춥니다. 시스템은 또한 이전에 생성된 AutoMV를 모니터링하고 더 이상 유용하지 않을 경우 이를 삭제합니다.

AutoMV 동작 및 기능은 사용자가 생성한 구체화된 보기와 동일합니다. 동일한 기준과 제한 사항을 사용하여 자동으로 증분 방식으로 새로 고쳐집니다. 사용자가 생성한 구체화된 뷰와 마찬가지로 구체화된 뷰를 사용하기 위한 자동 쿼리 재작성는 시스템 생성 AutoMV의 이점을 얻을 수 있는 쿼리를 식별합니다. 이러한 쿼리를 자동으로 다시 작성하여 AutoMV를 사용하므로 쿼리 성능이 향상됩니다. 개발자는 AutoMV를 활용하기 위해 쿼리를 수정할 필요가 없습니다.

> **Note**
>
> 자동 구체화된 뷰는 간헐적으로 새로 고쳐집니다. AutoMV를 사용하도록 재작성된 쿼리는 항상 최신 결과를 반환합니다. Redshift가 데이터가 최신이 아님을 감지하면 자동화된 구체화된

뷰에서 읽기 위해 쿼리가 다시 작성되지 않습니다. 대신 쿼리는 기본 테이블에서 최신 데이터를 선택합니다.

반복적으로 사용되는 쿼리가 있는 모든 워크로드는 AutoMV의 이점을 누릴 수 있습니다. 일반적인 사용 사례는 다음과 같습니다.

- 대시보드 - 대시보드는 주요 비즈니스 지표(KPI), 이벤트, 추세 및 기타 지표에 대한 빠른 보기를 제공하는 데 널리 사용됩니다. 대시보드는 차트 및 테이블과 함께 공통 레이아웃을 사용하는 경우가 많지만 필터링 또는 드릴다운과 같은 차원 선택 작업에 대해 다른 보기를 표시합니다. 대시보드에는 종종 다른 파라미터와 함께 반복적으로 사용되는 공통 쿼리 집합이 있습니다. 대시보드 쿼리는 자동화된 구체화된 보기에서 큰 이점을 얻을 수 있습니다.
- 보고서 - 보고 쿼리는 비즈니스 요구 사항 및 보고서 유형에 따라 다양한 빈도로 예약할 수 있습니다. 또한 자동화되거나 온디맨드일 수 있습니다. 보고 쿼리의 일반적인 특징은 오래 실행되고 리소스를 많이 사용한다는 것입니다. AutoMV를 사용하면 이러한 쿼리를 실행할 때마다 다시 계산할 필요가 없으므로 Redshift에서 각 쿼리의 런타임과 리소스 사용률이 감소합니다.

자동화된 구체화된 뷰를 끄려면 `auto_mv` 파라미터 그룹을 `false`로 업데이트합니다. 자세한 내용은 Amazon Redshift 클러스터 관리 안내서의 Amazon Redshift 파라미터 그룹 섹션을 참조하세요.

자동화되고 구체화된 보기의 SQL 범위 및 고려 사항

- 자동화된 구체화된 뷰는 쿼리 또는 하위 쿼리에 의해 시작되고 생성될 수 있습니다. 단, 쿼리가 SUM, COUNT, MIN, MAX 또는 AVG 집계 함수 중 하나 또는 `GROUP BY` 절을 포함해야 합니다. 하지만 다음은 포함할 수 없습니다.
  - 왼쪽, 오른쪽 또는 전체 외부 조인
  - SUM, COUNT, MIN, MAX, AVG 이외의 집계 함수. (이런 함수에서 자동 쿼리 재작성이 작동합니다.)
  - DISTINCT를 포함하는 모든 집계 함수
  - 모든 윈도 함수
  - SELECT DISTINCT 또는 HAVING 절
  - 다른 구체화된 뷰

  기준을 충족하는 쿼리가 자동화된 구체화된 뷰의 생성을 시작한다는 보장은 없습니다. 시스템은 워크로드에 대한 예상 이점과 유지 관리할 리소스 비용(시스템 새로 고침 비용 포함)을 기반으로 뷰를 생성할 후보를 결정합니다. 생성되는 각 구체화된 뷰는 자동 쿼리 재작성에서 사용할 수 있습니다.

- AutoMV가 하위 쿼리 또는 집합 연산자의 개별 구간에 의해 시작될 수 있지만 생성되는 구체화된 뷰에는 하위 쿼리 또는 집합 연산자가 포함되지 않습니다.
- AutoMV가 쿼리에 사용되었는지 확인하려면 EXPLAIN 계획을 보고 출력에서 `%_auto_mv_%`를 찾습니다. 자세한 내용은 EXPLAIN을 참조하세요.
- 자동화된 구체화된 뷰는 데이터 공유 및 페더레이션된 테이블과 같은 외부 테이블에서는 지원되지 않습니다.

자동화되고 구체화된 보기 제한 사항

다음은 자동화되고 구체화된 보기와 관련한 작업의 제한 사항입니다.

- 최대 AutoMV 수 - 자동화된 구체화된 뷰의 제한은 클러스터의 데이터베이스당 200개입니다.
- 스토리지 공간 및 용량 - AutoMV의 중요한 특징은 예비 백그라운드 주기를 사용하여 수행되어 사용자 워크로드가 영향을 받지 않도록 한다는 것입니다. 클러스터가 사용 중이거나 스토리지 공간이 부족한 경우 AutoMV는 작업을 중단합니다. 특히 총 클러스터 용량의 80%에서는 자동화된 구체화된 새로운 뷰가 생성되지 않습니다. 총 용량의 90%에서는 성능 저하 없이 사용자 워크로드를 계속 진행할 수 있도록 삭제할 수 있습니다. 클러스터 용량 결정에 대한 자세한 내용은 STV_NODE_STORAGE_CAPACITY 단원을 참조하세요.

자동 구체화된 뷰에 대한 요금 청구

Amazon Redshift의 자동 최적화 기능은 자동 구체화된 뷰를 생성하고 새로 고칩니다. 이 프로세스의 컴퓨팅 리소스에는 요금이 부과되지 않습니다. 자동 구체화된 뷰의 스토리지에는 일반 스토리지 요금이 청구됩니다. 자세한 내용은 Amazon Redshift 요금을 참조하세요.

추가적인 리소스

다음 블로그 게시물은 자동 구체화된 뷰에 대한 자세한 설명을 제공합니다. 자동 구체화된 뷰를 생성, 유지 관리 및 삭제하는 방법을 자세히 설명합니다. 또한 이러한 결정을 이끄는 기본 알고리즘인 자동 구체화된 뷰를 이용해 Amazon Redshift 쿼리 성능 최적화에 대해서도 설명합니다.

이 비디오는 구체화된 뷰에 대한 설명으로 시작하며 구체화된 뷰가 성능을 개선하고 리소스를 절약하는 방법을 보여줍니다. 그런 다음 프로세스 흐름 애니메이션과 실시간 데모를 통해 자동 구체화된 뷰를 심층적으로 설명합니다.

구체화된 뷰의 사용자 정의 함수(UDF) 사용

Amazon Redshift 구체화된 뷰에서 스칼라 UDF를 사용할 수 있습니다. 이를 Python 또는 SQL로 정의하고 구체화된 뷰 정의에서 참조합니다.

구체화된 뷰에서의 UDF 참조

다음 절차는 구체화된 뷰 정의에서 간단한 산술 비교를 수행하는 UDF를 사용하는 방법을 보여줍니다.

1. 구체화된 뷰 정의에서 사용할 테이블을 생성합니다.

```
CREATE TABLE base_table (a int, b int);
```

2. 정수가 비교 정수보다 큰지 여부를 나타내는 부울 값을 반환하는 Python에서 스칼라 사용자 정의 함수를 만듭니다.

```
CREATE OR REPLACE FUNCTION udf_python_bool(x1 int, x2 int) RETURNS bool IMMUTABLE
AS $$
  return x1 > x2
$$ LANGUAGE plpythonu;
```

   선택적으로 SQL과 기능이 유사한 UDF를 생성하여 결과를 첫 번째 함수와 비교할 수 있습니다.

```
CREATE OR REPLACE FUNCTION udf_sql_bool(int, int) RETURNS bool IMMUTABLE
AS $$
  select $1 > $2;
$$ LANGUAGE SQL;
```

3. 생성한 테이블에서 선택하고 UDF를 참조하는 구체화된 뷰를 생성합니다.

```
CREATE MATERIALIZED VIEW mv_python_udf AS SELECT udf_python_bool(a, b) AS a FROM
 base_table;
```

   선택적으로 SQL UDF를 참조하는 구체화된 뷰를 생성할 수 있습니다.

```
CREATE MATERIALIZED VIEW mv_sql_udf AS SELECT udf_sql_bool(a, b) AS a FROM
 base_table;
```

4. 테이블에 데이터를 추가하고 구체화된 뷰를 새로 고칩니다.

```
INSERT INTO base_table VALUES (1,2), (1,3), (4,2);
```

```
REFRESH MATERIALIZED VIEW mv_python_udf;
```

선택적으로 SQL UDF를 참조하는 구체화된 뷰를 새로 고칠 수 있습니다.

```
REFRESH MATERIALIZED VIEW mv_sql_udf;
```

5. 구체화된 뷰의 데이터를 쿼리합니다.

```
SELECT * FROM mv_python_udf ORDER BY a;
```

쿼리의 결과는 다음과 같습니다.

```
a
-----
false
false
true
```

열 a(4)의 값이 열 b(2)의 값보다 크기 때문에 마지막 값 집합에 대해 `true`가 반환됩니다.

6. 선택적으로 SQL UDF를 참조하는 구체화된 뷰를 쿼리할 수 있습니다. SQL 함수의 결과는 Python 버전의 결과와 일치합니다.

```
SELECT * FROM mv_sql_udf ORDER BY a;
```

쿼리의 결과는 다음과 같습니다.

```
a
-----
false
false
true
```

이렇게 하면 비교할 마지막 값 집합에 대해 `true`가 반환됩니다.

7. CASCADE와 함께 DROP 문을 사용하여 사용자 정의 함수와 이를 참조하는 구체화된 뷰를 삭제합니다.

```
DROP FUNCTION udf_python_bool(int, int) CASCADE;
```

```
DROP FUNCTION udf_sql_bool(int, int) CASCADE;
```

구체화된 뷰로 스트리밍 모으기

이 주제에서는 스트리밍 데이터에 빠르게 액세스하기 위해 구체화된 뷰를 사용하는 방법을 설명합니다.

스트리밍 모으기를 사용하면 지연 시간이 짧고 빠른 속도로 Amazon Kinesis Data Streams 또는 Amazon Managed Streaming for Apache Kafka에서 Amazon Redshift 프로비저닝된 또는 Redshift Serverless 데이터베이스로 스트림 데이터를 모을 수 있습니다. 데이터는 용도에 맞게 구성된 Redshift 구체화된 뷰에 배치됩니다. 따라서 외부 데이터에 빠르게 액세스할 수 있습니다. 스트리밍 모으기는 데이터 액세스 시간을 단축하고 스토리지 비용을 절감합니다. 소수의 SQL 명령 모음을 사용하여 Amazon Redshift 클러스터 또는 Amazon Redshift Serverless 작업 그룹에 대해 구성할 수 있습니다. 그런 다음 매번 구체화된 뷰 새로 고침을 사용하여 초당 수백 메가바이트의 데이터를 모을 수 있습니다.

스트리밍 서비스에서 Redshift로 데이터가 이동하는 방법

스트리밍 모으기의 작동 방식과 프로세스에서 사용되는 데이터베이스 객체를 이해하는 데 도움이 됩니다. 데이터는 데이터 스트림 제공업체에서 Amazon Redshift 프로비저닝된 클러스터 또는 Amazon Redshift Serverless 작업 그룹으로 직접 이동합니다. Amazon S3 버킷과 같은 임시 랜딩 영역이 없습니다. 프로비저닝된 클러스터 또는 작업 그룹은 스트림 소비자입니다. Redshift 데이터베이스에서는 스트림에서 읽은 데이터가 구체화된 뷰에 표시됩니다. 데이터는 도착하는 대로 처리됩니다. 예를 들어 SQL을 통해 JSON 값을 사용하고 구체화된 뷰의 데이터 열에 매핑할 수 있습니다. 구체화된 뷰가 새로 고쳐지면 Redshift는 뷰가 스트림과 함께 최신 상태로 유지될 때까지 할당된 Kinesis 데이터 샤드 또는 Kafka 파티션의 데이터를 사용합니다.

Amazon Redshift 스트리밍 모으기의 사용 사례에는 지속적으로 생성되고 시작 시점에서 짧은 기간(지연 시간) 내에 처리되어야 하는 데이터가 포함됩니다. 이를 실시간에 가까운 분석이라고 합니다. 소스에는 IT 디바이스, 시스템 텔레메트리 디바이스 및 사용량이 많은 웹 사이트 또는 애플리케이션의 클릭스트림 데이터가 포함될 수 있습니다.

성능 향상을 위한 데이터 구문 분석 모범 사례

스트리밍 모으기를 구성할 때 들어오는 데이터를 구문 분석하는 여러 가지 방법이 있습니다. 관행에는 데이터가 도착할 때 비즈니스 로직을 수행하거나 형식을 지정하는 것이 포함될 수 있습니다. 오류 또는 데이터 손실을 방지하기 위해 다음과 같은 모범 사례를 따르는 것이 좋습니다. 이는 내부 테스트를 통해 도출되었으며 고객이 구성 및 구문 분석 문제를 해결하는 데 도움이 됩니다.

- 스트리밍된 데이터에서 값 추출 - 구체화된 뷰 정의에서 JSON_EXTRACT_PATH_TEXT 함수를 사용하여 스트리밍되는 JSON을 구문 분석하거나 나누는 경우 성능과 지연 시간에 큰 영향을 미칠 수 있습니다. 설명하자면, JSON_EXTRACT_PATH_TEXT를 사용하여 추출한 각 열에 대해 수신되는 JSON이 다시 구문 분석됩니다. 그 후에는 데이터 형식 변환, 필터링, 비즈니스 로직 계산이 발생합니다. 즉, 예를 들어 JSON 데이터에서 10개의 열을 추출하면 각 JSON 레코드가 10번 구문 분석되며 여기에는 추가 로직이 포함됩니다. 이에 따라 수집 지연 시간이 길어집니다. 또한 JSON_PARSE 함수를 사용하여 JSON 레코드를 Redshift의 SUPER 데이터 형식으로 변환하는 대안적인 접근 방식도 권장합니다. 스트리밍된 데이터가 구체화된 뷰에 도착한 후 PartiQL을 사용하여 SUPER의 JSON 데이터 표현에서 개별 문자열을 추출합니다. 자세한 내용은 비정형 데이터 쿼리를 참조하세요.

  또한 JSON_EXTRACT_PATH_TEXT의 최대 데이터 크기는 64KB입니다. 따라서 JSON 레코드가 64KB보다 큰 경우 JSON_EXTRACT_PATH_TEXT로 처리 시 오류가 발생합니다.

- Amazon Kinesis Data Streams 스트림 또는 Amazon MSK 주제를 여러 구체화된 뷰에 매핑 - 단일 스트림 또는 주제에서 데이터를 모으기 위해 구체화된 뷰를 여러 개 생성하지 않는 것이 좋습니다. 각각의 구체화된 뷰가 Kinesis Data Streams 스트림 또는 Kafka 주제의 파티션에 있는 각 샤드에 대한 소비자를 생성하기 때문입니다. 이로 인해 스트림 또는 주제의 처리량이 제한되거나 초과될 수 있습니다. 또한 동일한 데이터를 여러 번 모으므로 비용이 더 많이 들 수도 있습니다. 스트리밍 모으기를 구성할 때는 각 스트림 또는 주제에 대해 하나의 구체화된 뷰를 생성하는 것이 좋습니다.

  사용 사례에서 하나의 KDS 스트림 또는 MSK 주제의 데이터를 여러 구체화된 뷰에 모아야 하는 경우 그렇게 하기 전에 AWS 빅 데이터 블로그, 특히 Amazon MSK에서 Amazon Redshift 스트리밍 모으기를 사용하여 거의 실시간 분석을 구현하는 모범 사례를 참조하세요.

스트리밍 모으기 동작 및 데이터 유형

다음 표에는 다양한 데이터 유형에 대한 기술적 동작 세부 정보 및 크기 제한이 설명되어 있습니다. 스트리밍 모으기를 위한 구체화된 뷰를 구성하기 전에 이러한 내용을 숙지하는 것이 좋습니다.

기능 또는 특성	설명
Kafka 주제 길이 제한	이름이 128자(따옴표 제외)보다 긴 Kafka 주제를 사용할 수 없습니다. 자세한 내용은 이름 및 식별자 단원을 참조하세요.
구체화된 뷰에서의 증분 새로 고침 및 JOIN	구체화된 보기는 점진적으로 유지 관리할 수 있어야 합니다. Kinesis 또는 Amazon MSK는 기본적으로 24시간 또는 7일 동안의 스트림 또는 주제 기록을 보존하지 않기 때문에 전체 재계산이 불가능합니다. Kinesis 또는 Amazon MSK에서 더 긴 데이터 보존 기간을 설정할 수 있습니다. 그러나 이렇게 하면 더 많은 유지 관리 및 비용이 발생할 수 있습니다. 또한 JOIN은 현재 Kinesis 스트림 또는 Amazon MSK 주제에서 생성된 구체화된 뷰에서 지원되지 않습니다. 스트림 또는 주제에서 구체화된 뷰를 생성한 후 스트리밍 구체화된 뷰를 다른 구체화된 뷰, 테이블 또는 뷰에 조인하기 위해 다른 구체화된 뷰를 만들 수 있습니다. 자세한 내용은 구체화된 보기 새로 고침 섹션을 참조하세요.
레코드 구문 분석	Amazon Redshift 스트리밍 수집은 KPL 주요 개념 - 집계에 의해 집계된 레코드 구문 분석을 지원하지 않습니다. 집계된 레코드는 수집되지만 이진 프로토콜 버퍼 데이터로 저장됩니다. (자세한 내용은 프로토콜 버퍼를 참조하세요.) 데이터를 Kinesis로 푸시하는 방법에 따라 이 기능을 꺼야 할 수 있습니다.
압축 해제	VARBYTE는 압축 해제를 지원하지 않습니다. 이로 인해 압축 데이터가 포함된 레코드는 Redshift 내에서 쿼리할 수 없습니다. 데이터를 Kinesis 스트림 또는 Amazon MSK 주제에 추가하기 전에 압축을 풉니다.
최대 레코드 크기	Amazon Redshift가 Kinesis 또는 Amazon MSK에서 수집할 수 있는 레코드의 최대 크기는 16,777,216바이트(16MiB)이며, 이는 Amazon Redshift의 VARBYTE 데이터 유형에서 지원하는 최대 크기입니다. 기본적으로 Kinesis 데이터 스트림 또는 Amazon MSK 주제에서 생성된 Amazon Redshift 스트리밍 구체화된 뷰는 데이터 열의 크기를 각각 1,048,576바이트(1MiB) 및 16,777,216바이트(16MiB)로 설정합니다.

기능 또는 특성	설명
	Note 1MiB는 Kinesis 데이터 스트림에 배치할 수 있는 현재 레코드의 최대 크기입니다. Kinesis 크기 제한에 관한 자세한 내용은 Amazon Kinesis Data Streams 개발자 안내서의 할당량 및 제한을 참조하시기 바랍니다.
오류 레코드	데이터 크기가 최댓값을 초과하여 레코드를 Redshift로 모을 수 없는 경우 해당 레코드를 건너뜁니다. 이 경우에도 구체화된 뷰 새로 고침은 여전히 성공하며 각 오류 레코드의 세그먼트가 SYS_STREAM_SCAN_ERRORS 시스템 테이블에 기록됩니다. 계산 오류 또는 형식 변환 오류와 같은 비즈니스 논리에서 발생하는 오류는 건너뛰지 않습니다. 구체화된 뷰 정의에 로직을 추가하기 전에 로직을 주의 깊게 테스트하세요.
Amazon MSK 다중 VPC 프라이빗 연결	Amazon MSK 다중 VPC 프라이빗 연결은 현재 Redshift 스트리밍 수집에 지원되지 않습니다. 그 대신 VPC 피어링을 사용하여 VPC를 연결하거나 AWS Transit Gateway를 사용하여 중앙 허브를 통해 VPC와 온프레미스 네트워크를 연결할 수 있습니다. 둘 중 하나를 사용하면 Redshift가 Amazon MSK 클러스터와 통신하거나 다른 VPC의 Amazon MSK Serverless와 통신할 수 있습니다.
자동 새로 고침 사용량 및 활성화	구체화된 뷰에 대한 자동 새로 고침 쿼리는 다른 사용자 워크로드로 처리됩니다. 자동 새로 고침은 데이터가 도착하면 스트림에서 데이터를 로드합니다. 스트리밍 수집을 위해 생성된 구체화된 뷰에 대해 자동 새로 고침을 명시적으로 설정할 수 있습니다. 이렇게 하려면 구체화된 뷰 정의에서 `AUTO REFRESH`를 지정합니다. 기본값은 수동 새로 고침입니다. 스트리밍 수집을 위해 기존 구체화된 뷰에 대한 자동 새로 고침을 지정하려면, `ALTER MATERIALIZED VIEW` 를 실행하여 새로 고침을 켜야 합니다. 자세한 내용은 구체화된 뷰 생성 또는 구체화된 뷰 변경을 참조하세요.

기능 또는 특성	설명
스트리밍 모으기와 Amazon Redshift Serverless	프로비저닝된 클러스터의 Amazon Redshift 스트리밍 모으기에 적용되는 설정 및 구성 지침이 Amazon Redshift Serverless의 스트리밍 모으기에도 적용됩니다. 자동 새로 고침 및 기타 워크로드로 스트리밍 모으기를 지원하려면 필요한 수준의 RPU를 지정하는 것이 중요합니다. 자세한 내용은 Amazon Redshift Serverless에 대한 청구를 참조하세요.
Amazon MSK 클러스터와 다른 가용 영역에 있는 Amazon Redshift 노드	스트리밍 모으기를 구성할 때 Amazon MSK에 대해 랙 인식이 활성화된 경우 Amazon Redshift는 동일한 가용 영역에 있는 Amazon MSK 클러스터에 연결을 시도합니다. 모든 노드가 Amazon Redshift 클러스터와 다른 가용 영역에 있는 경우 교차 가용 영역 데이터 전송 비용이 발생할 수 있습니다. 이를 방지하려면 Redshift 프로비저닝 클러스터 또는 작업 그룹과 동일한 AZ에 하나 이상의 Amazon MSK 브로커 클러스터 노드를 유지하세요.
새로 고침 시작 위치	구체화된 뷰를 생성한 후 초기 새로 고침은 Kinesis 스트림의 `TRIM_HORIZON` 또는 Amazon MSK 주제의 오프셋 0에서 시작됩니다.
데이터 형식	지원되는 데이터 형식은 `VARBYTE`에서 변환할 수 있는 형식으로 제한됩니다. 자세한 내용은 VARBYTE 형식 및 VARBYTE 연산자 단원을 참조하세요.
테이블에 레코드 추가	`ALTER TABLE APPEND`를 실행하여 기존 소스 구체화된 뷰에서 대상 테이블에 행을 추가할 수 있습니다. 구체화된 뷰가 스트리밍 수집을 위해 구성된 경우에만 작동합니다. 자세한 내용은 ALTER TABLE APPEND을 참조하세요.

기능 또는 특성	설명
TRUNCATE 또는 DELETE 실행	다음 방법을 사용하여 스트리밍 모으기에 사용되는 구체화된 뷰에서 레코드를 제거할 수 있습니다. • TRUNCATE - 이 명령은 스트리밍 모으기를 위해 구성된 구체화된 뷰에서 행을 모두 삭제합니다. 테이블 스캔은 수행하지 않습니다. 자세한 내용은 TRUNCATE를 참조하세요. • DELETE - 이 명령은 스트리밍 모으기를 위해 구성된 구체화된 뷰에서 행을 모두 삭제합니다. 자세한 내용은 DELETE를 참조하세요.

Amazon Kinesis Data Streams Streams에서 수집 시작

이 주제에서는 구체화된 뷰를 사용하여 Kinesis Data Streams의 스트리밍 데이터를 사용하는 방법을 설명합니다.

Amazon Redshift 스트리밍 수집을 설정하려면 스트리밍 데이터 원본에 매핑되는 외부 스키마를 생성하고 외부 스키마를 참조하는 구체화된 보기를 생성해야 합니다. Amazon Redshift 스트리밍 수집은 Kinesis Data Streams 지원을 소스로 제공합니다. 따라서 스트리밍 수집을 구성하기 전에 Kinesis Data Streams 소스를 사용할 수 있어야 합니다. 소스가 없는 경우 Kinesis 설명서의 Amazon Kinesis Data Streams 시작하기 지침을 따르거나 AWS 관리 콘솔을 통해 스트림 생성 지침을 사용하여 콘솔에 만듭니다.

Amazon Redshift 스트리밍 수집은 구체화된 보기를 사용합니다. 이 보기는 REFRESH 실행 중에 스트림에서 직접 업데이트됩니다. 구체화된 보기는 스트림 데이터 원본에 매핑됩니다. 구체화된 보기 정의의 일부로 스트림 데이터에 대한 필터링 및 집계를 수행할 수 있습니다. 스트리밍 수집 구체화된 보기(기본구체화된 보기)는 하나의 스트림만 참조할 수 있지만 기본 구체화된 보기 및 기타 구체화된 보기 또는 테이블과 조인하는 구체화된 보기를 추가로 생성할 수 있습니다.

> **Note**
>
> 스트리밍 수집 및 Amazon Redshift Serverless -이 주제의 구성 단계는 프로비저닝된 Amazon Redshift 클러스터와 Amazon Redshift Serverless 모두에 적용됩니다. 자세한 내용은 스트리밍 모으기 동작 및 데이터 유형 단원을 참조하십시오.

Kinesis Data Streams 스트림을 사용할 수 있다고 가정하면 첫 번째 단계는 CREATE EXTERNAL SCHEMA를 사용하여 Amazon Redshift Redshift에서 스키마를 정의하고 Kinesis Data Streams 리소스를 참조하는 것입니다. 그런 다음 스트림의 데이터에 액세스하려면 구체화된 보기에서 STREAM을 정의합니다. 반정형 SUPER 형식에 스트림 레코드를 저장하거나 Redshift 데이터 유형으로 변환되는 데이터를 생성하는 스키마를 정의할 수 있습니다. 구체화된 보기를 쿼리할 때 반환된 레코드는 스트림의 특정 시점 보기입니다.

1. Amazon Redshift 클러스터 또는 Amazon Redshift Serverless 작업 그룹이 이 역할을 수임하도록 허용하는 신뢰 정책으로 IAM 역할을 생성합니다. IAM 역할에 대한 신뢰 정책을 구성하는 방법에 대한 자세한 내용은 Amazon Redshift가 사용자를 대신하여 다른 AWS 서비스에 액세스할 수 있도록 권한 부여 섹션을 참조하세요. 역할이 생성된 후에는 Amazon Kinesis 데이터 스트림과의 통신 권한을 제공하는 다음 IAM 정책이 있어야 합니다.

   Kinesis 데이터 스트림의 암호화되지 않은 스트림에 대한 IAM 정책

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadStream",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStreamSummary",
                "kinesis:GetShardIterator",
                "kinesis:GetRecords",
                "kinesis:ListShards",
                "kinesis:DescribeStream"
            ],
            "Resource": "arn:aws:kinesis:*:0123456789:stream/*"
        },
        {
            "Sid": "ListStream",
            "Effect": "Allow",
            "Action": "kinesis:ListStreams",
            "Resource": "*"
        }
    ]
}
```

   Kinesis 데이터 스트림의 암호화된 스트림에 대한 IAM 정책

```
{
 "Version": "2012-10-17",
 "Statement": [{
   "Sid": "ReadStream",
   "Effect": "Allow",
   "Action": [
    "kinesis:DescribeStreamSummary",
    "kinesis:GetShardIterator",
    "kinesis:GetRecords",
    "kinesis:ListShards",
    "kinesis:DescribeStream"
   ],
   "Resource": "arn:aws:kinesis:*:0123456789:stream/*"
  },
  {
   "Sid": "DecryptStream",
   "Effect": "Allow",
   "Action": [
    "kms:Decrypt"
   ],
   "Resource": "arn:aws:kms:us-
east-1:0123456789:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  },
        {
   "Sid": "ListStream",
   "Effect": "Allow",
   "Action": "kinesis:ListStreams",
   "Resource": "*"
  }
 ]
}
```

2. VPC를 확인하고 Amazon Redshift 클러스터 또는 Amazon Redshift Serverless에 NAT 게이트웨이 또는 인터넷 게이트웨이를 사용하여 인터넷을 통해 Kinesis Data Streams 엔드포인트에 도달하는 경로가 있는지 확인하세요. Redshift와 Kinesis Data Streams 간의 트래픽을 AWS 네트워크 내에 유지하려면 Kinesis 인터페이스 VPC 엔드포인트를 사용하는 것이 좋습니다. 자세한 내용은 인터페이스 VPC 엔드포인트와 함께 Amazon Kinesis Data Streams Kinesis Data Streams 사용 단원을 참조하세요.

3. Amazon Redshift에서 외부 스키마를 생성하여 Kinesis의 데이터를 스키마에 매핑합니다.

```
CREATE EXTERNAL SCHEMA kds
```

```
FROM KINESIS
IAM_ROLE { default | 'iam-role-arn' };
```

Kinesis Data Streams의 스트리밍 수집에는 인증 유형이 필요하지 않습니다. CREATE EXTERNAL SCHEMA 명령문에 정의된 IAM 역할을 사용하여 Kinesis Data Streams 요청을 수행합니다.

선택 사항: REGION 키워드를 사용하여 Amazon Kinesis Data Streams 또는 Amazon MSK 스트림이 있는 리전을 지정합니다.

```
CREATE EXTERNAL SCHEMA kds
FROM KINESIS
REGION 'us-west-2'
IAM_ROLE { default | 'iam-role-arn' };
```

이 샘플에서 리전은 소스 스트림의 위치를 지정합니다. IAM_ROLE은 샘플입니다.

4. 스트림 데이터를 소비할 구체화된 보기를 생성합니다. 다음과 같은 명령문을 사용하면 레코드를 구문 분석할 수 없는 경우 오류가 발생합니다. 오류 레코드를 건너뛰지 않으려면 다음과 같은 명령을 사용하세요.

```
CREATE MATERIALIZED VIEW my_view AUTO REFRESH YES AS
SELECT *
FROM kds.my_stream_name;
```

Kinesis Stream 이름은 대/소문자를 구분하며 대문자와 소문자가 모두 포함될 수 있습니다. 대문자 이름을 가진 스트림에서 수집하려면 데이터베이스 수준에서 enable_case_sensitive_identifier 구성을 true로 설정하면 됩니다. 자세한 내용은 이름 및 식별자 및 enable_case_sensitive_identifier 섹션을 참조하세요.

자동 새로 고침을 켜려면 AUTO REFRESH YES를 사용합니다. 기본 동작은 수동 새로 고침입니다. 참고로 CAN_JSON_PARSE를 사용하면 구문 분석할 수 없는 레코드는 건너뛸 수 있습니다.

메타데이터 열에는 다음이 포함됩니다.

메타데이터 열	데이터 유형	설명
approximate_arrival_timestamp	시간대 미포함 TIMESTAMP	레코드가 Kinesis 스트림에 삽입된 대략적인 시간

메타데이터 열	데이터 유형	설명
partition_key	varchar(256)	샤드에 레코드를 할당하기 위해 Kinesis에서 사용하는 키
shard_id	char(20)	레코드가 검색된 스트림 내 샤드의 고유 식별자
sequence_number	varchar(128)	Kinesis 샤드에 있는 레코드의 고유 식별자
refresh_time	시간대 미포함 TIMESTAMP	새로 고침의 시작 시간
kinesis_data	varbyte	Kinesis 스트림의 레코드

구체화된 뷰 정의에 비즈니스 로직이 있는 경우 비즈니스 로직 오류로 인해 경우에 따라 스트리밍 수집이 차단될 수 있다는 점에 유의해야 합니다. 이로 인해 구체화된 뷰를 삭제하고 다시 생성해야 할 수도 있습니다. 이를 방지하려면 로직을 가능한 한 단순하게 유지하고 수집 후 데이터에 대한 대부분의 비즈니스 로직 검사를 수행하는 것이 좋습니다.

5. 뷰를 새로 고치면 Redshift Redshift가 스트림에서 읽고 데이터를 구체화된 뷰로 로드하도록 호출합니다.

```
REFRESH MATERIALIZED VIEW my_view;
```

6. 구체화된 보기의 데이터를 쿼리합니다.

```
select * from my_view;
```

Apache Kafka 소스에서 스트리밍 수집 시작하기

이 주제에서는 구체화된 뷰를 사용하여 Amazon MSK, Apache Kafka 또는 Confluent Cloud의 스트리밍 데이터를 사용하는 방법을 설명합니다.

Amazon Redshift 스트리밍 수집의 목적은 스트리밍 서비스에서 Amazon Redshift 또는 Amazon Redshift Serverless로 스트림 데이터를 직접 수집하는 프로세스를 단순화하는 것입니다. 이는 Amazon MSK Provisioned 및 Amazon MSK Serverless, 오픈 소스 Apache Kafka 및 Confluent Cloud

에서 작동합니다. Amazon Redshift 스트리밍 수집을 사용하면 Redshift로 스트림 데이터를 수집하기 전에 Amazon S3에서 Apache Kafka 주제를 준비할 필요가 없습니다.

기술적인 수준에서 스트리밍 수집은 짧은 지연 시간의 고속 수집으로 스트림 또는 주제 데이터를 Amazon Redshift 구체화된 뷰에 제공합니다. 설정 후 구체화된 뷰 새로 고침을 사용하여 대용량 데이터를 가져올 수 있습니다.

Amazon Redshift 스트리밍 수집을 구성하려면 먼저 Apache Kafka 소스를 사용할 수 있어야 합니다. 소스가 없는 경우 다음 지침에 따라 소스를 만듭니다.

- Amazon MSK - Getting Started Using Amazon MSK
- Apache Kafka - Apache Kafka Quickstart
- Confluent Cloud - Quick Start for Confluent Cloud

Kafka에서 스트리밍 수집 설정

다음 절차에 따라 AWS 관리형이 아닌 Apache Kafka 소스(Apache Kafka 및 Confluent Cloud)에서 Amazon Redshift로 스트리밍 수집을 설정합니다.

주제

- 인증 설정
- VPC 설정
- 구체화된 뷰 만들기

인증 설정

이 섹션에서는 Amazon Redshift 애플리케이션이 Amazon MSK 소스에 액세스하도록 허용하는 인증 설정을 설명합니다.

애플리케이션의 역할을 생성한 후 다음 정책 중 하나를 연결하여 Amazon MSK, Apache Kafka 또는 Confluent Cloud 클러스터에 대한 액세스를 허용합니다. mTLS 인증의 경우 Amazon Redshift가 사용하는 인증서를 ACM 또는 Secrets Manager에 저장할 수 있으므로 인증서가 저장되는 위치와 일치하는 정책을 선택해야 합니다.

AUTHENTICATION IAM(Amazon MSK만 해당):

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "MSKIAMpolicy",
            "Effect": "Allow",
            "Action": [
                "kafka-cluster:ReadData",
                "kafka-cluster:DescribeTopic",
                "kafka-cluster:Connect"
            ],
            "Resource": [
                "arn:aws:kafka:*:0123456789:cluster/MyTestCluster/*",
                "arn:aws:kafka:*:0123456789:topic/MyTestCluster/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kafka-cluster:AlterGroup",
                "kafka-cluster:DescribeGroup"
            ],
            "Resource": [
                "arn:aws:kafka:*:0123456789:group/MyTestCluster/*"
            ]
        }
    ]
}
```

AUTHENTICATION MTLS: AWS Certificate Manager에 저장된 인증서 사용

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "MSKmTLSACMpolicy",
            "Effect": "Allow",
            "Action": [
                "acm:ExportCertificate"
            ],
            "Resource": [
                "arn:aws:acm:us-east-1:444455556666:certificate/certificate_ID"
            ]
        }
    ]
}
```

AUTHENTICATION MTLS: AWS Secrets Manager에 저장된 인증서 사용

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "MSKmTLSSecretsManagerpolicy",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [
                "arn:aws:secretsmanager:us-east-1:444455556666:secret:secret_ID"
            ]
        }
    ]
}
```

Amazon MSK

AUTHENTICATION NONE을 사용하여 Amazon MSK 소스에 연결할 경우 IAM 역할이 필요하지 않습니다. 그러나 AUTHENTICATION IAM 또는 MTLS를 사용하여 Amazon MSK 클러스터로 인증하는 경우 Amazon Redshift 클러스터 또는 Amazon Redshift Serverless 네임스페이스에 적절한 권한을 갖추고 연결된 IAM 역할이 있어야 합니다. Amazon Redshift 클러스터 또는 Amazon Redshift Serverless 네임스페이스에서 이 역할을 수임하도록 허용하는 신뢰 정책을 포함하여 IAM 역할을 만듭니다. 역할을 만든 후 다음 권한 중 하나를 추가하여 IAM 또는 MTLS를 지원합니다. mTLS 인증의 경우 Amazon Redshift가 사용하는 인증서는 AWS Certificate Manager 또는 AWS Secrets Manager에 저장할 수 있으므로 인증서가 저장되는 위치와 일치하는 정책을 선택해야 합니다. Amazon Redshift 프로비저닝 클러스터 또는 Redshift Serverless 네임스페이스에 대한 역할을 연결합니다. IAM 역할에 대한 신뢰 정책을 구성하는 방법에 대한 자세한 내용은 Amazon Redshift가 사용자를 대신하여 다른 AWS 서비스에 액세스할 수 있도록 권한 부여 섹션을 참조하세요.

다음 테이블에는 Amazon MSK에서 스트리밍 수집을 위해 설정할 수 있는 무료 구성 옵션이 나와 있습니다.

Amazon Redshift 구성	Amazon MSK 구성	Redshift와 Amazon MSK 간에 열 포트
AUTHENTICATION NONE	TLS 전송 비활성화됨	9092

Amazon Redshift 구성	Amazon MSK 구성	Redshift와 Amazon MSK 간에 열 포트
AUTHENTICATION NONE	TLS 전송 활성화됨	9094
AUTHENTICATION IAM	IAM	9098/9198
AUTHENTICATION MTLS	TLS 전송 활성화됨	9094

Amazon Redshift 인증은 CREATE EXTERNAL SCHEMA 문에서 설정합니다.

> **Note**
>
> Amazon MSK 클러스터에 mTLS(상호 전송 계층 보안) 인증이 활성화되어 있는 경우, 인증 없음을 사용하도록 Amazon Redshift를 구성하면 인증되지 않은 액세스를 위해 포트 9094를 사용하도록 지시합니다. 그러나 이 포트는 mTLS 인증에서 사용 중이므로 실패합니다. 따라서 mTLS를 사용할 경우 AUTHENTICATION MTLS로 전환하는 것이 좋습니다.

Apache Kafka or Confluent Cloud

Apache Kafka 및 Confluent Cloud의 경우 Amazon Redshift는 다음 연결 프로토콜을 지원합니다.

- Apache Kafka에 연결할 때 인증에 TLS 전송과 함께 mTLS 또는 일반 텍스트를 사용할 수 있습니다.
- Confluent Cloud에 연결할 때는 인증에 mTLS만 사용할 수 있습니다.

Amazon Redshift는 Apache Kafka 또는 Confluent Cloud에 연결하기 위해 다음과 같은 암호화 프로토콜을 지원합니다.

Apache Kafka 및 Confluent Cloud에 지원되는 인증 방법

Amazon Redshift	Kafka 보안 프로토콜	Apache Kafka 지원	Confluent 클라우드 지원
AUTHENTICATION NONE	PLAINTEXT	No	No

Amazon Redshift	Kafka 보안 프로토콜	Apache Kafka 지원	Confluent 클라우드 지원
AUTHENTICATION NONE	SSL	예	No
AUTHENTICATION IAM	SASL_SSL	No	No
AUTHENTICATION MTLS	SSL	예(인증서 포함)	예(인증서 포함)

Amazon Redshift는 SASL/SCRAM 또는 SASL/PLAINTEXT를 지원하지 않습니다.

VPC 설정

인증 리소스를 만든 후 VPC를 확인하고 Amazon Redshift 클러스터 또는 Amazon Redshift Serverless 작업 그룹에 Apache Kafka 소스로 이동하는 경로가 있는지 확인합니다.

> **Note**
>
> Amazon MSK의 경우 Amazon MSK 클러스터에 대한 인바운드 보안 그룹 규칙은 Amazon Redshift 클러스터 또는 Redshift Serverless 작업 그룹의 보안 그룹을 허용해야 합니다. 지정 포트는 Amazon MSK 클러스터에서 구성된 인증 방법에 따라 다릅니다. 자세한 내용은 포트 정보 및 AWS 내에서 그러나 VPC 외부에서 액세스를 참조하세요.

다음으로, Amazon Redshift 클러스터 또는 Amazon Redshift Serverless 작업 그룹에서 향상된 VPC 라우팅을 사용하도록 설정합니다. 자세한 내용은 향상된 VPC 라우팅을 참조하세요.

구체화된 뷰 만들기

이 섹션에서는 Amazon Redshift가 Apache Kafka 스트리밍 데이터에 액세스하는 데 사용하는 구체화된 뷰를 설정합니다.

사용 가능한 Apache Kafka 클러스터가 있다고 가정하고 첫 번째 단계는 `CREATE EXTERNAL SCHEMA`를 사용하여 Redshift에서 스키마를 정의하고 클러스터를 데이터 소스로 참조하는 것입니다. 그런 다음 주제의 데이터에 액세스하려면 구체화된 뷰에서 `STREAM`을 정의합니다. 기본값 Amazon

Redshift VARBYTE 데이터 유형을 사용하여 주제에서 레코드를 저장하거나, 반정형 SUPER 형식에 데이터를 변환하는 스키마를 정의할 수 있습니다. 구체화된 뷰를 쿼리할 때 반환된 레코드는 주제의 특정 시점 보기입니다.

1. Amazon Redshift에서 Apacke Kafka 클러스터에 매핑할 외부 스키마를 만듭니다. 구문은 다음과 같습니다.

```
CREATE EXTERNAL SCHEMA MySchema
FROM KAFKA
[ IAM_ROLE [ default | 'iam-role-arn' ] ]
AUTHENTICATION [ none | iam | mtls ]
[AUTHENTICATION_ARN 'acm-certificate-arn' | SECRET_ARN 'ssm-secret-arn' ];
```

   FROM 절에서 KAFKA는 스키마가 Apache Kafka 소스의 데이터를 매핑함을 나타냅니다.

   AUTHENTICATION은 스트리밍 수집을 위한 인증 유형을 나타냅니다. 사용 가능한 3가지 유형은 다음과 같습니다.

   - none - 필요한 인증이 없다고 지정합니다. 이는 MSK에 대해 인증되지 않은 액세스에 해당합니다. 이는 Apache Kafka의 SSL 인증에 해당합니다. 이 인증 방법은 Confluent Cloud에서 지원되지 않습니다.
   - iam - IAM 인증을 지정합니다. Amazon MSK에서는 IAM 인증만 사용할 수 있습니다. 이 항목을 선택할 때는 IAM 역할에 IAM 인증 권한이 있는지 확인해야 합니다. 필수 IAM 정책 설정에 관한 자세한 내용은 Kafka에서 스트리밍 수집 설정 섹션을 참조하시기 바랍니다.
   - mtls - 클라이언트와 서버 간의 인증을 용이하게 하여 상호 전송 계층 보안을 통해 안전한 통신을 제공하도록 지정합니다. 이 경우 클라이언트는 Redshift이고 서버는 Apache Kafka입니다. mTLS를 사용하여 스트리밍 수집 구성에 관한 자세한 내용은 Apache Kafka 소스에서 Redshift 스트리밍 수집을 위한 mTLS 인증 섹션을 참조하시기 바랍니다.

   사용자 이름과 암호를 사용한 Amazon MSK 인증은 스트리밍 수집에서 지원되지 않으니 참고합니다.

   AUTHENTICATION_ARN 파라미터는 암호화된 연결을 설정하는 데 사용하는 ACM 상호 전송 계층 보안(mTLS) 인증서의 ARN을 지정합니다.

   SECRET_ARN 파라미터는 Amazon Redshift에서 mTLS에 사용할 인증서가 포함된 AWS Secrets Manager 보안 암호의 ARN을 지정합니다.

다음 예시는 외부 스키마를 만들 경우 Amazon MSK 클러스터에 대한 브로커 URI를 설정하는 방법을 보여줍니다.

IAM 인증 사용:

```
CREATE EXTERNAL SCHEMA my_schema
FROM KAFKA
IAM_ROLE 'arn:aws:iam::012345678901:role/my_role'
AUTHENTICATION IAM
URI 'b-1.myTestCluster.123z8u.c2.kafka.us-
west-1.amazonaws.com:9098,b-2.myTestCluster.123z8u.c2.kafka.us-
west-1.amazonaws.com:9098'
```

인증 없음 사용

```
CREATE EXTERNAL SCHEMA my_schema
FROM KAFKA
AUTHENTICATION none
URI 'b-1.myTestCluster.123z8u.c2.kafka.us-
west-1.amazonaws.com:9092,b-2.myTestCluster.123z8u.c2.kafka.us-
west-1.amazonaws.com:9092'
```

mTLS 사용:

```
CREATE EXTERNAL SCHEMA my_schema
FROM KAFKA
IAM_ROLE 'arn:aws:iam::012345678901:role/my_role'
AUTHENTICATION MTLS
URI 'b-1.myTestCluster.123z8u.c2.kafka.us-west-1.amazonaws.com:9094,b-
 2.myTestCluster.123z8u.c2.kafka.us-west-1.amazonaws.com:9094'
AUTHENTICATION_ARN 'acm-certificate-arn' | [ SECRET_ARN 'ssm-secret-arn' ];
```

외부 스키마 생성에 대한 자세한 내용은 CREATE EXTERNAL SCHEMA를 참조하세요.

2. 스트림 데이터를 소비할 구체화된 뷰를 생성합니다. 다음과 샘플과 같은 SQL 명령을 사용합니다.

```
CREATE MATERIALIZED VIEW MyView AUTO REFRESH YES AS
SELECT *
FROM MySchema."mytopic";
```

Kafka 주제 이름은 대소문자를 구분하며 대문자와 소문자를 모두 포함할 수 있습니다. 대문자 이름을 가진 주제에서 수집하려면 세션 또는 데이터베이스 수준에서 `enable_case_sensitive_identifier` 구성을 `true`로 설정하면 됩니다. 자세한 내용은 이름 및 식별자 및 enable_case_sensitive_identifier 섹션을 참조하세요.

자동 새로 고침을 켜려면 `AUTO REFRESH YES`를 사용합니다. 기본 동작은 수동 새로 고침입니다.

3. 메타데이터 열에는 다음이 포함됩니다.

메타데이터 열	데이터 유형	설명
kafka_partition	bigint	Kafka 주제에 있는 레코드의 파티션 ID
kafka_offset	bigint	주어진 파티션에 대한 Kafka 주제의 레코드 오프셋
kafka_timestamp_type	char(1)	Kafka 레코드에 사용되는 타임스탬프 유형: • C - 클라이언트 측의 레코드 생성 시간(CREATE_TIME) • L - Kafka 서버 측의 레코드 추가 시간(LOG_APPEND_TIME) • U - 레코드 생성 시간을 사용할 수 없음(NO_TIMESTAMP_TYPE)
kafka_timestamp	시간대 미포함 TIMESTAMP	timestamp 값의 형식
kafka_key	varbyte	Kafka 레코드의 핵심
kafka_value	varbyte	Kafka로부터 받은 레코드
kafka_headers	super	Kafka로부터 받은 레코드의 헤더

메타데이터 열	데이터 유형	설명
refresh_time	시간대 미포함 TIMESTAMP	새로 고침의 시작 시간

중요한 것은 구체화된 뷰 정의에 비즈니스 로직이 있는 경우 비즈니스 로직 오류로 인해 경우에 따라 스트리밍 수집이 실패할 수 있다는 것입니다. 이로 인해 구체화된 뷰를 삭제하고 다시 생성해야 할 수도 있습니다. 이를 방지하려면 비즈니스 로직을 단순하게 유지하고 수집 후 데이터에 대한 추가 로직을 실행하는 것이 좋습니다.

4. 뷰를 새로 고치면 Amazon Redshift Redshift가 주제에서 읽고 데이터를 구체화된 뷰로 로드하도록 트리거합니다.

```
REFRESH MATERIALIZED VIEW MyView;
```

5. 구체화된 보기의 데이터를 쿼리합니다.

```
select * from MyView;
```

구체화된 뷰는 REFRESH 실행 시 주제에서 직접 업데이트됩니다. Kafka 주제 데이터 소스에 매핑되는 구체화된 뷰를 생성합니다. 구체화된 뷰 정의의 일부로 데이터에 대한 필터링 및 합계를 수행할 수 있습니다. 스트리밍 수집 구체화된 뷰(기본 구체화된 뷰)는 하나의 Kafka 주제만 참조할 수 있지만 기본 구체화된 뷰 및 기타 구체화된 뷰 또는 테이블과 조인하는 구체화된 뷰를 추가로 생성할 수 있습니다.

스트리밍 수집 제한 사항에 대한 자세한 내용은 스트리밍 모으기 동작 및 데이터 유형 단원을 참조하세요.

Apache Kafka 소스에서 Redshift 스트리밍 수집을 위한 mTLS 인증

상호 전송 계층 보안(mTLS)에서는 서버가 정보를 보내는 클라이언트를 인증하고 클라이언트가 서버를 인증할 수 있는 수단을 제공합니다. mTLS 사용의 이점은 여러 산업의 수직적 애플리케이션의 다양한 사용 사례에서 신뢰할 수 있는 인증을 사용할 수 있다는 것입니다. 여기에는 금융, 소매, 기관 및 의료 산업의 사용 사례가 포함됩니다. Redshift로 스트리밍 수집을 하는 경우 인증이 서버 간에 이뤄집니다. 이는 Amazon MSK, Apache Kafka 또는 Confluent Cloud와 Amazon Redshift 프로비저닝된 클러스터 또는 Amazon Redshift Serverless 작업 그룹일 수 있습니다.

이 주제에서는 mTLS를 사용하여 Redshift 클라이언트와 Apache Kafka 서버 간에 인증하는 외부 스키마를 만드는 방법을 보여주는 절차 및 SQL 명령 예시를 제공합니다. 이 주제에서는 Apache Kafka 소

스에서 스트리밍 수집을 설정하는 전체 단계를 단계별로 보완해 줍니다. 자세한 내용은 Apache Kafka 소스에서 스트리밍 수집 시작하기 단원을 참조하십시오.

스트리밍 수집에서 mTLS를 사용하기 위한 사전 조건

이 섹션에서는 AWS Certificate Manager 또는 AWS Secrets Manager를 사용하여 스트리밍 수집에 mTLS를 사용하기 위한 사전 조건 단계를 제공합니다.

예비 단계로, 다른 기능 중에서 보안 통신 채널을 통해 보안 통신을 사용하는 인증서를 발급하는 데 사용할 수 있는 프라이빗 인증 기관(PCA)을 보유하거나 만들어야 합니다. AWS Private Certificate Authority (프라이빗 CA)는 이 함수를 수행하는 사용 가능한 서비스입니다. 자세한 내용은 AWS Private Certificate Authority 사용 설명서에 나와 있는 프라이빗 CA 생성을 참조하시기 바랍니다. 프라이빗 CA를 만든 후 루트 CA 인증서를 내보내고 확장명을 .pem으로 하여 파일에 저장합니다.

CA 인증서를 사용하는 클러스터를 만들려면 다음을 수행합니다.

Amazon MSK

1. mtls 클라이언트 인증을 지원하는 Amazon MSK 클러스터를 만듭니다. Amazon MSK 클러스터 구성에 관한 자세한 내용은 Amazon Managed Streaming for Apache Kafka Developer Guide에 나와 있는 Mutual TLS client authentication for Amazon MSK 섹션을 참조하시기 바랍니다.
2. Amazon MSK 클러스터의 보안 설정을 편집하고 AWS Certificate Manager(ACM)을 사용하여 TLS 클라이언트 인증을 켜고 이전에 만든 AWS Private CA(PCA)를 선택합니다. 자세한 내용은 Amazon Managed Streaming for Apache Kafka 개발자 안내서에 나와 있는 클러스터의 보안 설정 업데이트를 참조하시기 바랍니다.

Confluent Cloud

1. 전용 Confluent Cloud 클러스터를 만듭니다. 가급적이면 Amazon Redshift 클러스터와 동일한 AWS 리전에 만드는 것이 좋습니다. Confluent Cloud 클러스터 만들기에 대한 자세한 내용은 Create a Kafka cluster in Confluent Cloud를 참조하세요.
2. 앞서 만들어 내보낸 AWS Private CA 루트 CA 인증서 pem 파일을 업로드합니다. 자세한 내용은 Manage certificate authority for mTLS authentication for Confluent Cloud를 참조하세요. Confluent Cloud는 이 인증서를 사용하여 Amazon Redshift 클라이언트 인증서를 확인합니다.

AWS Certificate Manager으로 스트리밍 수집 시 mTLS 사용

다음 절차에서는 인증서 스토리지 및 관리용 AWS Certificate Manager(ACM)를 활용하여 Redshift 스트리밍 수집 시 mTLS를 구성하는 방법을 보여줍니다.

1. ACM을 통해 프라이빗 인증서를 요청합니다. 이렇게 할 때 사전 조건 섹션에서 만든 PCA를 인증 기관으로 선택합니다. ACM은 보안 통신 용도로 서명된 인증서와 연결된 프라이빗 키를 저장합니다. 자세한 내용은 AWS Certificate Manager 사용 설명서에 나와 있는 인증서 발급 및 관리를 참조하시기 바랍니다.
2. Redshift 클러스터 또는 Amazon Redshift Serverless 작업 그룹을 관리하는 데 사용하는 IAM 역할의 경우 acm:ExportCertificate 인증서를 내보낼 수 있는 권한을 연결합니다. 스트리밍 수집 시 필요한 IAM 리소스 설정에 관한 자세한 내용은 Kafka에서 스트리밍 수집 설정 섹션을 참조하시기 바랍니다. 다음 단계에서 동일한 IAM 역할을 지정하여 외부 스키마를 만듭니다.

   > **Note**
   >
   > AWS Certificate Manager에 대한 요청에는 VPC에서 인터넷 게이트웨이(IGW) 또는 NAT 게이트웨이(NGW)가 필요합니다. VPC에 IGW 또는 NGW가 없는 경우 다음을 수행합니다.
   >
   > - ACM 대신 Secrets Manager를 사용하여 인증서를 저장합니다.
   > - Secrets Manager VPC 엔드포인트를 VPC에 연결합니다.
   >
   > 스트리밍 수집에 mTLS와 함께 Secrets Manager를 사용하는 방법에 대한 자세한 내용은 다음 AWS Secrets Manager으로 스트리밍 수집 시 mTLS 사용 섹션을 참조하세요.

3. Amazon MSK, Apache Kafka 또는 Confluent Cloud 클러스터에 대한 부트스트랩 브로커 URI를 가져옵니다. Amazon MSK용 부트스트랩 브로커 URI 가져오기에 대한 자세한 내용은 Amazon Managed Streaming for Apache Kafka Developer Guide의 Getting the bootstrap brokers for an Amazon MSK cluster 섹션을 참조하세요.
4. 다음 예시와 같은 SQL 명령을 실행하여 `mtls`를 사용하여 클러스터를 Redshift 외부 스키마에 매핑하는 외부 스키마를 만듭니다.

   Amazon MSK

   ```
   CREATE EXTERNAL SCHEMA my_schema
   FROM KAFKA
   IAM_ROLE 'arn:aws:iam::012345678901:role/my_role'
   ```

```
AUTHENTICATION mtls
URI 'b-1.myTestCluster.123z8u.c2.kafka.us-
west-1.amazonaws.com:9094,b-2.myTestCluster.123z8u.c2.kafka.us-
west-1.amazonaws.com:9094'
AUTHENTICATION_ARN 'arn:aws:acm:Region:444455556666:certificate/certificate_ID';
```

Apache Kafka or Confluent Cloud

```
CREATE EXTERNAL SCHEMA my_schema
FROM KAFKA
IAM_ROLE 'arn:aws:iam::012345678901:role/my_role'
AUTHENTICATION mtls
URI 'lkc-2v531.domz6wj0p.us-west-1.aws.confluent.cloud:9092'
AUTHENTICATION_ARN 'arn:aws:acm:region:444455556666:certificate/certificate_ID';
```

중요한 파라미터:

- IAM_ROLE - 스트리밍 수집 시 클러스터와 연결된 IAM 역할입니다.
- URI - 클러스터에 대한 부트스트랩 브로커 URI입니다. Amazon MSK의 경우 포트 9094는 TLS 암호화를 위해 브로커와 통신하도록 지정되어 있으니 참고하시기 바랍니다.
- AUTHENTICATION_ARN – ACM 인증서의 ARN입니다. ARN은 발급된 인증서를 선택하면 ACM 콘솔에서 사용할 수 있습니다.

이러한 구성 단계를 수행한 후 샘플에 정의된 스키마를 참조하는 Redshift 구체화된 뷰를 생성한 다음, REFRESH MATERIALIZED VIEW를 사용하여 데이터를 스트리밍할 수 있습니다. 자세한 내용은 Apache Kafka 소스에서 스트리밍 수집 시작하기 단원을 참조하십시오.

AWS Secrets Manager으로 스트리밍 수집 시 mTLS 사용

AWS Certificate Manager에서 인증서를 참조하지 않으려면 인증서 관리용으로 AWS Secrets Manager를 사용하여 Redshift 스트리밍 수집을 위해 mTLS를 구성할 수 있습니다. 다음 단계에서는 Secrets Manager를 사용하여 mTLS를 구성하는 방법을 설명합니다.

1. 원하는 도구를 사용하여 인증서 서명 요청 및 프라이빗 키를 만듭니다. 그런 다음 클러스터에 대한 인증서를 만드는 데 사용한 것과 동일한 AWS 프라이빗 CA(PCA)를 사용하여 서명 요청을 사용해 서명된 인증서를 생성합니다. 인증서 발급에 관한 자세한 내용은 AWS Private Certificate Authority API 참조의 IssueCertificate를 참조하시기 바랍니다.

2. AWS Private Certificate Authority을 사용하여 인증서를 추출합니다. 자세한 내용은 AWS Private Certificate Authority 사용 설명서의 Retrieve a private certificate 섹션을 참조하시기 바랍니다.
3. AWS Secrets Manager의 이전 단계에서 만든 인증서와 프라이빗 키를 저장합니다. `Other type of secret`을 선택하고 일반 텍스트 형식을 사용합니다. 키-값 페어는 다음 예제처럼 `{"certificate":"<cert value>","privateKey":"<pkey value>"}` 형식과 일치해야 합니다. AWS Secrets Manager에서 보안 암호 생성에 관한 자세한 내용은 AWS Secrets Manager 사용 설명서의 AWS Secrets Manager으로 암호 생성 및 관리를 참조하시기 바랍니다.

```
{"certificate":"-----BEGIN CERTIFICATE-----
klhdslkfjahksgdfkgioeuyihbflahabhbdslv6akybeoiwv1hoaiusdhbahsbdi
H4hAX8/eE96qCcjkpfT84EdvHzp6fC+/WwM0oXlwUEWlvfMCXNaG5D8SqRq3qA==
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
klhdslkfjahksgdfkgioeuyihbflahabhbdslv6akybeoiwv1hoaiusdhbahsbdi
wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
-----END CERTIFICATE-----",
"privateKey":"-----BEGIN PRIVATE KEY-----
klhdslkfjahksgdfkgioeuyihbflahabhbdslv6akybeoiwv1hoaiusdhbahsbdi
7OD4m1dBEs3Fj++hDMH9rYRp99RqtCOf0EWOUe139KOilOsW+cyhAoc9Ci2+Jo/k
17u2N1iGILMQEZuCRtnJOkFYkw==
-----END PRIVATE KEY-----"}
```

4. 권한 정책을 연결하여 Amazon Redshift 클러스터 또는 Amazon Redshift Serverless 작업 그룹을 관리하는 데 사용하는 IAM 역할에 대한 보안 암호를 검색합니다. 이 권한은 `secretsmanager:GetSecretValue`입니다. 자세한 내용은 인증 설정 단원을 참조하십시오. IAM 정책 관리에 대한 자세한 내용은 IAM 정책 편집을 참조하세요. 다음 단계에서 동일한 IAM 역할을 지정하여 외부 스키마를 만듭니다.
5. Redshift에서 SQL을 실행하여 외부 스키마를 만듭니다. AUTHENTICATION 유형(`mtls`)을 사용합니다. 또한 AWS Secrets Manager에서 클러스터의 URI와 보안 암호 ARN을 지정합니다.

```
CREATE EXTERNAL SCHEMA my_schema
FROM KAFKA
IAM_ROLE 'arn:aws:iam::012345678901:role/my_role'
AUTHENTICATION mtls
URI 'b-1.myTestCluster.123z8u.c2.kafka.us-
west-1.amazonaws.com:9094,b-2.myTestCluster.123z8u.c2.kafka.us-
west-1.amazonaws.com:9094'
SECRET_ARN 'arn:aws:secretsmanager:us-east-1:012345678910:secret:myMTLSSecret';
```

중요한 파라미터:

- IAM_ROLE - 스트리밍 수집 시 클러스터와 연결된 IAM 역할입니다.
- URI - 클러스터에 대한 부트스트랩 브로커 URI입니다. Amazon MSK의 경우 포트 9094는 TLS 암호화를 위해 브로커와 통신하도록 지정되어 있으니 참고하시기 바랍니다.
- SECRET_ARN - mTLS에 사용할 인증서를 포함하는 Secrets Manager의 보안 암호 ARN입니다.

기존 외부 스키마에 대한 mTLS 인증 사용

스트리밍 수집 시 사용하는 기존 외부 스키마가 있고 인증을 위해 상호 TLS를 구현하려는 경우 ACM에서 mTLS 인증 및 ACM 인증서 ARN을 지정하는 다음과 같은 명령을 실행할 수 있습니다.

```
ALTER EXTERNAL SCHEMA schema_name
AUTHENTICATION mtls
AUTHENTICATION_ARN 'arn:aws:acm:Region:444455556666:certificate/certificate_ID';
```

또는 AWS Secrets Manager의 보안 암호 ARN을 참조하여 mTLS 인증을 지정할 수 있습니다.

```
ALTER EXTERNAL SCHEMA schema_name
AUTHENTICATION mtls
SECRET_ARN 'arn:aws:secretsmanager:us-east-1:012345678910:secret:myMTLSSecret';
```

Kinesis를 사용한 스트리밍 데이터 수집

다음 절차는 ev_station_data로 명명된 Kinesis 스트림에서 데이터를 수집하는 방법을 보여줍니다. 이 데이터는 다양한 EV 충전 스테이션의 소비 데이터를 JSON 형식으로 포함합니다. 스키마가 잘 정의되어 있습니다. 이 예에서는 데이터를 원시 JSON으로 저장하는 방법과 수집될 때 JSON 데이터를 Amazon Redshift 데이터 유형으로 변환하는 방법을 보여 줍니다.

생산자 설정

1. Amazon Kinesis Data Streams 사용하여 다음 단계에 따라 `ev_station_data`라는 이름의 스트림을 생성합니다. 용량 모드(Capacity mode)에 온디맨드(On-demand)를 선택합니다. 자세한 내용은 AWS 관리 콘솔을 통해 스트림 생성을 참조하세요.
2. 이 Amazon Kinesis 데이터 생성기는 스트림에 사용할 테스트 데이터를 생성하는 데 도움이 됩니다. 시작하려면 도구에 설명된 단계에 따라 다음 데이터 템플릿을 사용하여 데이터를 생성합니다.

   ```
   {
   ```

```
   "_id" : "{{random.uuid}}",
   "clusterID": "{{random.number(
        {   "min":1,
            "max":50
        }
    )}}",
    "connectionTime": "{{date.now("YYYY-MM-DD HH:mm:ss")}}",
    "kWhDelivered": "{{commerce.price}}",
    "stationID": "{{random.number(
        {   "min":1,
            "max":467
        }
    )}}",
      "spaceID": "{{random.word}}-{{random.number(
        {   "min":1,
            "max":20
        }
    )}}",

   "timezone": "America/Los_Angeles",
   "userID": "{{random.number(
        {   "min":1000,
            "max":500000
        }
    )}}"
}
```

스트림 데이터의 각 JSON 객체에는 다음과 같은 속성이 있습니다.

```
{
    "_id": "12084f2f-fc41-41fb-a218-8cc1ac6146eb",
    "clusterID": "49",
    "connectionTime": "2022-01-31 13:17:15",
    "kWhDelivered": "74.00",
    "stationID": "421",
    "spaceID": "technologies-2",
    "timezone": "America/Los_Angeles",
    "userID": "482329"
}
```

Amazon Redshift 설정

다음 단계는 데이터를 수집하도록 구체화된 보기를 구성하는 방법을 보여줍니다.

1. 외부 스키마를 만들어 Kinesis 데이터를 Redshift 객체로 매핑합니다.

```
CREATE EXTERNAL SCHEMA evdata FROM KINESIS
IAM_ROLE 'arn:aws:iam::0123456789:role/redshift-streaming-role';
```

   IAM 역할을 구성하는 방법에 대한 자세한 내용은 Amazon Kinesis Data Streams Streams에서 수집 시작 섹션을 참조하세요.

2. 스트림 데이터를 소비할 구체화된 보기를 생성합니다. 다음 예에서는 JSON 소스 데이터를 수집하기 위해 구체화된 보기를 정의하는 두 가지 방법을 모두 보여 줍니다.

   먼저 스트림 레코드를 반정형 SUPER 형식으로 저장합니다. 이 예에서 JSON 소스는 Redshift 유형으로 변환하지 않고 Redshift에 저장됩니다.

```
CREATE MATERIALIZED VIEW ev_station_data AS
    SELECT approximate_arrival_timestamp,
    partition_key,
    shard_id,
    sequence_number,
    case when can_json_parse(kinesis_data) then json_parse(kinesis_data) else null
 end as payload,
    case when not can_json_parse(kinesis_data) then kinesis_data else null end as
 failed_payload
    FROM evdata."ev_station_data" ;
```

   반대로, 다음 구체화된 보기 정의에서 구체화된 보기에는 Redshift에 정의된 스키마가 있습니다. 구체화된 보기는 스트림의 UUID 값에 분산되며 approximatearrivaltimestamp 값으로 정렬됩니다.

```
CREATE MATERIALIZED VIEW ev_station_data_extract DISTKEY(6) sortkey(1) AUTO REFRESH
 YES AS
    SELECT refresh_time,
    approximate_arrival_timestamp,
    partition_key,
    shard_id,
    sequence_number,

 json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'_id',true)::character(36)
 as ID,
```

```
json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'clusterID',true)::varchar(30)
as clusterID,

json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'connectionTime',true)::varchar(
as connectionTime,

json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'kWhDelivered',true)::DECIMAL(10
as kWhDelivered,

json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'stationID',true)::DECIMAL(10,2)
as stationID,

json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'spaceID',true)::varchar(100)
as spaceID,
   json_extract_path_text(from_varbyte(kinesis_data,
'utf-8'),'timezone',true)::varchar(30)as timezone,

json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'userID',true)::varchar(30)
as userID
   FROM evdata."ev_station_data"
   WHERE LENGTH(kinesis_data) < 65355;
```

스트림 쿼리

1. 새로 고쳐진 구체화된 보기를 쿼리하여 사용량 통계를 가져옵니다.

```
SELECT to_timestamp(connectionTime, 'YYYY-MM-DD HH24:MI:SS') as connectiontime
,SUM(kWhDelivered) AS Energy_Consumed
,count(distinct userID) AS #Users
from ev_station_data_extract
group by  to_timestamp(connectionTime, 'YYYY-MM-DD HH24:MI:SS')
order by 1 desc;
```

2. 결과 보기.

```
connectiontime         energy_consumed #users
2022-02-08 16:07:21+00   4139         10
2022-02-08 16:07:20+00   5571         10
2022-02-08 16:07:19+00   8697         20
2022-02-08 16:07:18+00   4408         10
2022-02-08 16:07:17+00   4257         10
```

```
2022-02-08 16:07:16+00   6861         10
2022-02-08 16:07:15+00   5643         10
2022-02-08 16:07:14+00   3677         10
2022-02-08 16:07:13+00   4673         10
2022-02-08 16:07:11+00   9689         20
```

AWS Glue Data Catalog 뷰

이 주제에서는 AWS Glue Data Catalog에서 뷰를 만드는 방법을 설명합니다. 데이터 카탈로그의 뷰를 사용하여 동일한 스키마로 다양한 데이터 소스의 데이터에 액세스할 수 있습니다.

데이터 카탈로그에서 뷰를 생성하면 Amazon Athena 및 Amazon EMR Spark와 같은 엔진 전반에서 사용할 단일 공통 뷰 스키마와 메타데이터 객체를 만들 수 있습니다. 이렇게 하면 데이터 레이크와 데이터 웨어하우스 전반에서 사용 사례에 맞게 동일한 뷰를 사용할 수 있습니다. 데이터 카탈로그의 뷰는 정의자 뷰로 범주화된다는 점에서 특별합니다. 즉, 뷰를 쿼리하는 사용자가 아니라 뷰를 생성한 사용자가 액세스 권한을 정의합니다. 다음은 데이터 카탈로그에서 뷰를 만들 때의 사용 사례와 이점입니다.

- 사용자에게 필요한 권한에 따라 데이터 액세스를 제한하는 뷰를 만듭니다. 예를 들어 데이터 카탈로그의 뷰를 사용하여 인사 관리(HR) 부서 소속이 아닌 직원은 개인 식별 정보(PII)를 보지 못하게 할 수 있습니다.
- 사용자가 불완전한 레코드에 액세스할 수 없게 합니다. 데이터 카탈로그의 뷰에 특정 필터를 적용하여 데이터 카탈로그의 뷰에 있는 데이터 레코드가 항상 완전한지 확인할 수 있습니다.
- 데이터 카탈로그 뷰에는 뷰를 만드는 데 사용된 쿼리 정의가 완료되어야 뷰를 만들 수 있다는 보안상의 이점이 있습니다. 이러한 보안상의 이점은 데이터 카탈로그의 뷰가 악의적인 사용자의 SQL 명령에 영향을 받지 않는다는 것을 의미합니다.
- 데이터 카탈로그의 뷰는 일반 뷰와 동일한 이점을 지원합니다. 예를 들어 사용자에게 기본 테이블에 대한 사용 권한을 제공하지 않고도 뷰 액세스는 허용할 수 있습니다.

데이터 카탈로그에서 뷰를 만들려면 Spectrum 외부 테이블, Lake Formation에서 관리하는 데이터 공유 또는 Apache Iceberg 테이블이 있어야 합니다.

데이터 카탈로그 뷰의 정의는 AWS Glue Data Catalog에 저장됩니다. AWS Lake Formation을 사용하여 리소스 부여, 열 부여 또는 태그 기반 액세스 제어를 통해 액세스 권한을 부여할 수 있습니다. Lake Formation에서 액세스 권한 부여 및 취소에 대한 자세한 내용은 데이터 카탈로그 리소스에 대한 권한 부여 및 취소를 참조하세요.

사전 조건

데이터 카탈로그에서 뷰를 생성하기 전에 다음 사전 조건을 충족해야 합니다.

- IAM 역할에 다음과 같은 신뢰 정책이 있어야 합니다.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
               "Service": [
                    "glue.amazonaws.com",
                    "lakeformation.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

- 또한 다음 역할 전달 정책이 필요합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1",
            "Action": [
                "iam:PassRole"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "glue.amazonaws.com",
                        "lakeformation.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

- 마지막으로, 다음 권한이 필요합니다.
  - `Glue:GetDatabase`
  - `Glue:GetDatabases`

- Glue:CreateTable
- Glue:GetTable
- Glue:UpdateTable
- Glue:DeleteTable
- Glue:GetTables
- Glue:SearchTables
- Glue:BatchGetPartition
- Glue:GetPartitions
- Glue:GetPartition
- Glue:GetTableVersion
- Glue:GetTableVersions

종합 예제

먼저 데이터 카탈로그 데이터베이스를 기반으로 외부 스키마를 만드세요.

```
CREATE EXTERNAL SCHEMA IF NOT EXISTS external_schema FROM DATA CATALOG DATABASE
 'external_data_catalog_db'
IAM_ROLE 'arn:aws:iam::123456789012:role/sample-role';
```

이제 데이터 카탈로그 뷰를 만들 수 있습니다.

```
CREATE EXTERNAL PROTECTED VIEW external_schema.remote_view
AS SELECT * FROM external_schema.remote_table;
```

그런 다음 뷰 쿼리를 시작할 수 있습니다.

```
SELECT * FROM external_schema.remote_view;
```

데이터 카탈로그의 뷰와 관련된 SQL 명령에 대한 자세한 내용은 CREATE EXTERNAL VIEW, ALTER EXTERNAL VIEW, DROP EXTERNAL VIEW를 참조하세요.

보안 로깅

Redshift는 쿼리가 다중 언어 Glue 뷰를 참조할 때 Redshift 시스템 로그에 기록된 메타데이터를 마스킹합니다. 다중 언어란 뷰가 Redshift 및 Amazon EMR과 같은 다양한 쿼리 엔진의 SQL 언어를 지원한다는 의미입니다. 다음 테이블의 데이터는 쿼리 ID가 동일한 모든 쿼리에 대해 마스킹됩니다. 다음 테이블에는 보안 로깅이 적용된 시스템 뷰와 열이 나열되어 있습니다.

시스템 테이블	중요한 열
SYS_EXTERNAL_QUERY_DETAIL	열: source_type, total_partitions, qualified_partitions, scanned_files, returned_rows, returned_bytes, file_format, file_location, external_query_text, warning_message 자세한 내용은 SYS_EXTERNAL_QUERY_DETAIL 단원을 참조하십시오.
SYS_EXTERNAL_QUERY_ERROR	열: file_location, rowid, column_name, original_value, modified_value, trigger, action, action_value, error_code 자세한 내용은 SYS_EXTERNAL_QUERY_ERROR 단원을 참조하십시오.
SYS_QUERY_DETAIL	열: step_name, table_id, table_name, input_bytes, input_rows, output_bytes, output_rows, blocks_read, blocks_write, local_read_IO, remote_read_IO, spilled_block_local_disk, spilled_block_remote_disk 자세한 내용은 SYS_QUERY_DETAIL 단원을 참조하십시오.
SYS_QUERY_HISTORY	열: returned_rows, returned_bytes 자세한 내용은 SYS_QUERY_HISTORY 단원을 참조하십시오.
STL_AGGR	열: rows, bytes, tbl, type 자세한 내용은 STL_AGGR 단원을 참조하십시오.
STL_BCAST	열: rows, bytes, packets 자세한 내용은 STL_BCAST 단원을 참조하십시오.
STL_DDLTEXT	열: text 자세한 내용은 STL_DDLTEXT 단원을 참조하십시오.
STL_DELETE	열: rows, tbl 자세한 내용은 STL_DELETE 단원을 참조하십시오.
STL_DIST	열: rows, bytes, packets 자세한 내용은 STL_DIST 단원을 참조하십시오.

시스템 테이블	중요한 열
STL_EXPLAIN	열: plannode, info 자세한 내용은 STL_EXPLAIN 단원을 참조하십시오.
STL_HASH	열: rows, bytes, tbl, est_rows 자세한 내용은 STL_HASH 단원을 참조하십시오.
STL_HASHJOIN	열: rows, tbl, num_parts, join_type 자세한 내용은 STL_HASHJOIN 단원을 참조하십시오.
STL_INSERT	열: rows, tbl 자세한 내용은 STL_INSERT 단원을 참조하십시오.
STL_LIMIT	열: rows 자세한 내용은 STL_LIMIT 단원을 참조하십시오.
STL_MERGE	열: rows 자세한 내용은 STL_MERGE 단원을 참조하십시오.
STL_MERGEJOIN	열: rows, tbl 자세한 내용은 STL_MERGEJOIN 단원을 참조하십시오.
STL_NESTLOOP	열: rows, tbl 자세한 내용은 STL_NESTLOOP 단원을 참조하십시오.
STL_PARSE	열: rows 자세한 내용은 STL_PARSE 단원을 참조하십시오.
STL_PLAN_INFO	열: rows, bytes 자세한 내용은 STL_PLAN_INFO 단원을 참조하십시오.
STL_PROJECT	열: rows, tbl 자세한 내용은 STL_PROJECT 단원을 참조하십시오.
STL_QUERY	열: querytxt 자세한 내용은 STL_QUERY 단원을 참조하십시오.
STL_QUERY_METRICS	열: max_rows, rows, max_blocks_read, blocks_read, max_block s_to_disk, blocks_to_disk, max_query_scan_size, query_scan_size 자세한 내용은 STL_QUERY_METRICS 단원을 참조하십시오.
STL_QUERYTEXT	열: text 자세한 내용은 STL_QUERYTEXT 단원을 참조하십시오.
STL_RETURN	열: rows, bytes 자세한 내용은 STL_RETURN 단원을 참조하십시오.
STL_SAVE	열: rows, bytes, tbl 자세한 내용은 STL_SAVE 단원을 참조하십시오.
STL_SCAN	열: rows, bytes, fetches, type, tbl, rows_pre_filter, perm_table_name, scanned_mega_value 자세한 내용은 STL_SCAN 단원을 참조하십시오.

시스템 테이블	중요한 열
STL_SORT	열: rows, bytes, tbl 자세한 내용은 STL_SORT 단원을 참조하십시오.
STL_TR_CONFLICT	열: table_id 자세한 내용은 STL_TR_CONFLICT 단원을 참조하십시오.
STL_UNDONE	열: table_id 자세한 내용은 STL_UNDONE 단원을 참조하십시오.
STL_UNIQUE	열: rows, type, bytes 자세한 내용은 STL_UNIQUE 단원을 참조하십시오.
STL_UTILITYTEXT	열: text 자세한 내용은 STL_UTILITYTEXT 단원을 참조하십시오.
STL_WINDOW	열: rows 자세한 내용은 STL_WINDOW 단원을 참조하십시오.
STV_BLOCKLIST	열: col, tbl, num_values, minvalue, maxvalue 자세한 내용은 STV_BLOCKLIST 단원을 참조하십시오.
STV_EXEC_STATE	열: rows, bytes, label 자세한 내용은 STV_EXEC_STATE 단원을 참조하십시오.
STV_LOCKS	열: table_id 자세한 내용은 STV_LOCKS 단원을 참조하십시오.
STV_QUERY_METRICS	열: rows, max_rows, blocks_read, max_blocks_read, max_blocks_to_disk, blocks_to_disk, max_query_scan_size, query_scan_size 자세한 내용은 STV_QUERY_METRICS 단원을 참조하십시오.
STV_STARTUP_RECOVERY_STATE	열: table_id, table_name 자세한 내용은 STV_STARTUP_RECOVERY_STATE 단원을 참조하십시오.
STV_TBL_PERM	열: id, rows, sorted_rows, temp, block_count, query_scan_size 자세한 내용은 STV_TBL_PERM 단원을 참조하십시오.
STV_TBL_TRANS	열: id, rows, size 자세한 내용은 STV_TBL_TRANS 단원을 참조하십시오.
SVCS_EXPLAIN	열: plannode, info 자세한 내용은 SVCS_EXPLAIN 단원을 참조하십시오.

시스템 테이블	중요한 열
SVCS_PLAN_INFO	열: rows, bytes 자세한 내용은 SVCS_PLAN_INFO 단원을 참조하십시오.
SVCS_QUERY_SUMMARY	열: step, rows, bytes, rate_row, rate_byte, label, rows_pre_filter 자세한 내용은 SVCS_QUERY_SUMMARY 단원을 참조하십시오.
SVCS_S3LIST	열: bucket, prefix, max_file_size, avg_file_size 자세한 내용은 SVCS_QUERY_SUMMARY 단원을 참조하십시오.
SVCS_S3LOG	컬럼: message 자세한 내용은 SVCS_QUERY_SUMMARY 단원을 참조하십시오.
SVCS_S3PARTITION_SUMMARY	열: total_partitions, qualified_partitions, min_assigned_partitions, max_assigned_partitions, avg_assigned_partitions 자세한 내용은 SVCS_S3PARTITION_SUMMARY 단원을 참조하십시오.
SVCS_S3QUERY_SUMMARY	열: external_table_name, file_format, s3_scanned_rows, s3_scanned_bytes, s3query_returned_rows, s3query_returned_bytes 자세한 내용은 SVCS_S3QUERY_SUMMARY 단원을 참조하십시오.
SVL_QUERY_METRICS	열: step_label, scan_row_count, join_row_count, nested_loop_join_row_count, return_row_count, spectrum_scan_row_count, spectrum_scan_size_mb 자세한 내용은 SVL_QUERY_METRICS 단원을 참조하십시오.
SVL_QUERY_METRICS_SUMMARY	열: step_label, scan_row_count, join_row_count, nested_loop_join_row_count, return_row_count, spectrum_scan_row_count, spectrum_scan_size_mb 자세한 내용은 SVL_QUERY_METRICS_SUMMARY 단원을 참조하십시오.
SVL_QUERY_REPORT	열: rows, bytes, label, rows_pre_filter 자세한 내용은 SVL_QUERY_REPORT 단원을 참조하십시오.
SVL_QUERY_SUMMARY	열: rows, bytes, rows_pre_filter 자세한 내용은 SVL_QUERY_SUMMARY 단원을 참조하십시오.

시스템 테이블	중요한 열
SVL_S3LIST	열: bucket, prefix, max_file_size, avg_file_size 자세한 내용은 SVL_S3LIST 단원을 참조하십시오.
SVL_S3LOG	컬럼: message 자세한 내용은 SVL_S3LOG 단원을 참조하십시오.
SVL_S3PARTITION	열: rows, bytes, label, rows_pre_filter 자세한 내용은 SVL_S3PARTITION 단원을 참조하십시오.
SVL_S3PARTITION_SUMMARY	열: total_partitions, qualified_partitions, min_assigned_partitions, max_assigned_partitions, avg_assigned_partitions 자세한 내용은 SVL_S3PARTITION_SUMMARY 단원을 참조하십시오.
SVL_S3QUERY	열: external_table_name, file_format, s3_scanned_rows, s3_scanned_bytes, s3query_returned_rows, s3query_returned_bytes 자세한 내용은 SVL_S3QUERY 단원을 참조하십시오.
SVL_S3QUERY_SUMMARY	열: external_table_name, file_format, s3_scanned_rows, s3_scanned_bytes, s3query_returned_rows, s3query_returned_bytes 자세한 내용은 SVL_S3QUERY_SUMMARY 단원을 참조하십시오.
SVL_S3RETRIES	열: file_size, location, message 자세한 내용은 SVL_S3RETRIES 단원을 참조하십시오.
SVL_SPECTRUM_SCAN_ERROR	열: location, rowid, colname, original_value, modified_value 자세한 내용은 SVL_SPECTRUM_SCAN_ERROR 단원을 참조하십시오.
SVL_STATEMENTTEXT	열: type, text 자세한 내용은 SVL_STATEMENTTEXT 단원을 참조하십시오.
SVL_STORED_PROC_CALL	열: querytxt 자세한 내용은 SVL_STORED_PROC_CALL 단원을 참조하십시오.
SVL_STORED_PROC_MESSAGES	열: querytext 자세한 내용은 SVL_STORED_PROC_MESSAGES 단원을 참조하십시오.
SVL_UDF_LOG	열: funcname 자세한 내용은 SVL_UDF_LOG 단원을 참조하십시오.

시스템 테이블	중요한 열
SVV_DISKUSAGE	열: name, col, tbl, blocknum, num_values, minvalue, maxvalue 자세한 내용은 SVV_DISKUSAGE 단원을 참조하십시오.
SVV_QUERY_STATE	열: rows, bytes, label 자세한 내용은 SVV_QUERY_STATE 단원을 참조하십시오.
SVV_TABLE_INFO	열: table_id, table 자세한 내용은 SVV_TABLE_INFO 단원을 참조하십시오.
SVV_TRANSACTIONS	열: relation 자세한 내용은 SVV_TRANSACTIONS 단원을 참조하십시오.

고려 사항 및 제한

다음은 데이터 카탈로그에서 생성된 뷰에 적용되는 고려 사항과 제한 사항입니다.

- 다른 뷰를 기반으로 하는 데이터 카탈로그 뷰는 만들 수 없습니다.
- 데이터 카탈로그 뷰에는 기본 테이블이 10개만 허용됩니다.
- 뷰 정의자는 기본 테이블에 대한 모든 `SELECT GRANTABLE` 권한을 가지고 있어야 합니다.
- 뷰에는 Lake Formation 객체와 내장 기능만 포함할 수 있습니다. 다음과 같은 객체는 뷰 안에 넣을 수 없습니다.
  - 시스템 테이블
  - 사용자 정의 함수(UDF)
  - Lake Formation 관리형 데이터 공유에 포함되지 않은 Redshift 테이블, 뷰, 구체화된 뷰, 지연 바인딩 뷰
- 뷰에는 중첩된 Redshift Spectrum 테이블을 포함할 수 없습니다.
- AWS Glue를 통한 뷰의 기본 객체 표현은 뷰와 동일한 AWS 계정 및 리전에 있어야 합니다.

Amazon Redshift에서 공간 데이터 쿼리

공간 데이터는 정의된 공간(공간 참조 시스템)에서 지오메트리의 위치와 모양을 설명합니다. Amazon Redshift는 공간 데이터와 선택적으로 데이터의 공간 참조 시스템 식별자(SRID)를 포함하는 GEOMETRY 및 GEOGRAPHY 데이터 유형의 공간 데이터를 지원합니다.

공간 데이터에는 지리학적 특성을 나타내는 데 사용할 수 있는 기하학적 데이터가 포함됩니다. 이러한 형식의 데이터의 예로는 일기 예보, 지도 안내, 지리적 위치, 매장 위치 및 항공사 노선이 있는 트윗 등이 있습니다. 공간 데이터는 비즈니스 분석, 보고 및 예측에 중요한 역할을 합니다.

Amazon Redshift SQL 함수를 사용하여 공간 데이터를 쿼리할 수 있습니다. 공간 데이터에는 객체의 기하학적 값이 포함됩니다.

GEOMETRY 데이터 유형 작업은 데카르트 평면에서 작동합니다. 공간 참조 시스템 식별자(SRID)는 객체 내부에 저장되지만, 이 SRID는 단지 좌표계의 식별자일 뿐 GEOMETRY 객체를 처리하는 알고리즘에서는 아무런 역할을 하지 않는다. 반대로 GEOGRAPHY 데이터 유형에 대한 작업은 객체 내부의 좌표를 회전 타원체의 구형 좌표로 취급합니다. 이 회전 타원체는 지리 공간 참조 시스템을 참조하는 SRID에 의해 정의됩니다. 기본적으로 GEOGRAPHY 데이터 유형은 WGS(World Geodetic System) 84를 참조하는 공간 참조(SRID) 4326으로 생성됩니다. SRID에 대한 자세한 내용은 Wikipedia의 공간 참조 시스템을 참조하세요.

ST_Transform 함수를 사용하여 다양한 공간 참조 시스템의 좌표를 변환할 수 있습니다. 좌표 변환이 완료된 후 입력 GEOMETRY가 지리적 SRID로 인코딩되는 한 둘 사이에 간단한 캐스트를 사용할 수도 있습니다. 이 캐스트는 추가 변환 없이 좌표를 복사합니다. 예:

```
SELECT ST_AsEWKT(ST_GeomFromEWKT('SRID=4326;POINT(10 20)')::geography);
```

```
st_asewkt
------------------------
 SRID=4326;POINT(10 20)
```

GEOMETRY 및 GEOGRAPHY 데이터 유형의 차이점을 더 잘 이해하려면 WGS(World Geodetic System) 84를 사용하여 베를린 공항(BER)과 샌프란시스코 공항(SFO) 간의 거리를 계산하는 것이 좋습니다. GEOGRAPHY 데이터 유형을 사용하는 경우 결과는 미터 단위입니다. SRID 4326과 함께 GEOMETRY 데이터 유형을 사용하는 경우 결과는 도 단위이며 1도의 거리는 지구 지오메트리의 위치에 따라 다르기 때문에 미터로 변환할 수 없습니다.

GEOGRAPHY 데이터 유형에 대한 계산은 왜곡 없이 국가의 정확한 면적과 같은 현실적인 둥근 지구 계산에 주로 사용됩니다. 그러나 계산하는 데 훨씬 더 많은 비용이 듭니다. 따라서 ST_Transform은 좌표를 적절한 로컬 투영 좌표계로 변환하고 GEOMETRY 데이터 유형에 대한 계산을 더 빠르게 수행할 수 있습니다.

공간 데이터를 사용하여 다음을 수행하기 위해 쿼리를 실행할 수 있습니다.

- 두 지점 사이의 거리를 파악합니다.
- 한 영역(폴리곤)에 다른 영역이 포함되는지 확인합니다.
- 한 라인스트링이 다른 라인스트링 또는 다각형과 교차하는지 확인합니다.

GEOMETRY 데이터 형식을 사용하여 공간 데이터의 값을 보유할 수 있습니다. Amazon Redshift의 GEOMETRY 값은 2차원(2D), 3차원(3DZ), 측정값이 있는 2차원(3DM) 및 4차원(4D) 지오메트리 기본 데이터 형식을 정의할 수 있습니다.

- 2차원(2D) 지오메트리는 평면에서 2개의 직교 좌표(x, y)로 지정됩니다.
- 3차원(3DZ) 지오메트리는 공간에서 3개의 데카르트 좌표(x, y, z)로 지정됩니다.
- 측정값이 있는 2차원(3DM) 지오메트리는 3개의 좌표(x, y, m)로 지정됩니다. 여기서 처음 2개는 평면의 데카르트 좌표이고 3번째는 측정값입니다.
- 4차원(4D) 지오메트리는 4개의 좌표(x, y, z, m)로 지정되며, 처음 3개는 공간의 데카르트 좌표이고 4번째는 측정값입니다.

지오메트리 기본 데이터 형식에 대한 자세한 내용은 Wikipedia의 Well-known text representation of geometry를 참조하세요.

GEOGRAPHY 데이터 형식을 사용하여 공간 데이터의 값을 보유할 수 있습니다. Amazon Redshift의 GEOGRAPHY 값은 2차원(2D), 3차원(3DZ), 측정값이 있는 2차원(3DM) 및 4차원(4D) 지오메트리 기본 데이터 형식을 정의할 수 있습니다.

- 2차원(2D) 지오메트리는 회전 타원체의 경도 및 위도 좌표로 지정됩니다.
- 3차원(3DZ) 지오메트리는 회전 타원체의 경도, 위도 및 고도 좌표로 지정됩니다.
- 측정값이 있는 2차원(3DM) 지오메트리는 3개의 좌표(경도, 위도, 측정값)로 지정됩니다. 여기서 처음 2개는 구의 각 좌표이고 세 번째 좌표는 측정값입니다.
- 4차원(4D) 지오메트리는 4개의 좌표(경도, 위도, 고도, 측정값)로 지정됩니다. 여기서 처음 3개는 경도, 위도 및 고도이고 4번째 좌표는 측정값입니다.

지리 좌표계에 대한 자세한 내용은 Wikipedia의 지리 좌표계와 구면 좌표계를 참조하세요.

GEOMETRY 및 GEOGRAPHY 데이터 유형의 하위 유형은 다음과 같습니다.

- POINT
- LINESTRING
- POLYGON
- MULTIPOINT
- MULTILINESTRING
- MULTIPOLYGON
- GEOMETRYCOLLECTION

다음과 같은 기하학적 데이터 표현을 지원하는 Amazon Redshift SQL 함수가 있습니다.

- GeoJSON
- WKT(Well-known Text)
- EWKT(Extended Well-Known Text)
- WKB(Well-Known Binary) 표현
- EWKB(Extended Well-Known Binary)

GEOMETRY 및 GEOGRAPHY 데이터 유형 간에 캐스팅할 수 있습니다.

다음 SQL은 라인스트링을 GEOMETRY에서 GEOGRAPHY로 캐스팅합니다.

```
SELECT ST_AsEWKT(ST_GeomFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)')::geography);
```

```
 st_asewkt
---------------------------------------------
 SRID=4326;LINESTRING(110 40,2 3,-10 80,-7 9)
```

다음 SQL은 라인스트링을 GEOGRAPHY에서 GEOMETRY로 캐스팅합니다.

```
SELECT ST_AsEWKT(ST_GeogFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)')::geometry);
```

```
 st_asewkt
--------------------------------------------
 SRID=4326;LINESTRING(110 40,2 3,-10 80,-7 9)
```

Amazon Redshift는 공간 데이터 쿼리를 위해 많은 SQL 함수를 제공합니다. `ST_IsValid` 함수를 제외하고 `GEOMETRY` 객체를 인수로 수락하는 공간 함수는 이 `GEOMETRY` 객체가 유효한 지오메트리일 것으로 예상합니다. `GEOMETRY` 또는 `GEOGRAPHY` 객체가 유효하지 않으면 공간 함수의 동작이 정의되지 않습니다. 유효성에 대한 자세한 내용은 기하학적 유효성 섹션을 참조하세요.

공간 데이터를 쿼리하는 SQL 함수에 대한 자세한 내용은 공간 함수 섹션을 참조하세요.

공간 데이터 로드에 대한 자세한 내용은 GEOMETRY 또는 GEOGRAPHY 데이터 유형의 열 로드 섹션을 참조하세요.

주제

- 튜토리얼: Amazon Redshift에서 공간 SQL 함수 사용
- Amazon Redshift에 shapefile 로드
- Amazon Redshift 공간 데이터 용어
- Amazon Redshift에서 공간 데이터를 사용할 때의 고려 사항

튜토리얼: Amazon Redshift에서 공간 SQL 함수 사용

이 튜토리얼에서는 Amazon Redshift에서 일부 공간 SQL 함수를 사용하는 방법을 설명합니다.

이렇게 하려면 공간 SQL 함수를 사용하여 2개의 테이블을 쿼리합니다. 이 튜토리얼은 임대 숙박 시설의 위치 데이터를 독일 베를린의 우편 번호와 연관시키는 공개 데이터 집합의 데이터를 사용합니다.

주제

- 사전 조건
- 1단계: 테이블 생성 및 테스트 데이터 로드
- 2단계: 공간 데이터 쿼리
- 3단계: 리소스 정리

사전 조건

이 튜토리얼을 이해하려면 다음 리소스가 필요합니다.

- 액세스하고 업데이트할 수 있는 기존 Amazon Redshift 클러스터 및 데이터베이스. 기존 클러스터에서 테이블을 생성하고, 샘플 데이터를 로드하고, SQL 쿼리를 실행하여 공간 함수를 시연합니다. 클러스터는 노드가 2개 이상 있어야 합니다. 클러스터를 생성하는 방법에 대해 알아보려면 Amazon Redshift 시작 안내서의 단계를 따르세요.
- Amazon Redshift 쿼리 편집기를 사용하려면 클러스터가 쿼리 편집기를 지원하는 AWS 리전에 있어야 합니다. 자세한 정보는 Amazon Redshift 관리 안내서의 쿼리 편집기를 사용하여 데이터베이스 쿼리를 참조하세요.
- Amazon S3에서 테스트 데이터를 로드할 수 있도록 하는 Amazon Redshift 클러스터에 대한 AWS 자격 증명. Amazon S3와 같은 다른 AWS 서비스에 액세스하는 방법에 대한 자세한 내용은 Amazon Redshift에 대한 AWS 서비스 액세스 권한 부여를 참조하십시오.
- Amazon S3 데이터를 읽기 위해 관리형 정책 `AmazonS3ReadOnlyAccess`가 연결된 `mySpatialDemoRole`이라는 AWS Identity and Access Management(IAM) 역할. Amazon S3 버킷에서 데이터를 로드할 수 있는 권한이 있는 역할을 생성하려면 Amazon Redshift 관리 가이드의 IAM 역할을 사용하여 COPY, UNLOAD 및 CREATE EXTERNAL SCHEMA 작업에 대한 권한 부여 섹션을 참조하세요.
- IAM 역할 `mySpatialDemoRole`을 생성한 후 해당 역할은 Amazon Redshift 클러스터와 연결되어 있어야 합니다. 해당 연결을 생성하는 방법에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 IAM 역할을 사용하여 COPY, UNLOAD 및 CREATE EXTERNAL SCHEMA 작업에 대한 권한 부여 섹션을 참조하세요.

1단계: 테이블 생성 및 테스트 데이터 로드

이 튜토리얼에서 사용하는 원본 데이터는 `accommodations.csv` 및 `zipcodes.csv` 파일에 있습니다.

`accommodations.csv` 파일은 insideairbnb.com의 오픈 소스 데이터입니다. `zipcodes.csv` 파일은 독일 베를린-브란덴부르크 국립 통계 연구소(Amt für Statistik Berlin-Brandenburg)의 오픈 소스 데이터인 우편 번호를 제공합니다. 두 데이터 원본 모두 Creative Commons 라이선스에 따라 제공됩니다. 데이터는 독일 베를린 리전으로 제한됩니다. 이러한 파일은 이 튜토리얼에서 사용할 Amazon S3 퍼블릭 버킷에 있습니다.

필요에 따라 다음 Amazon S3 링크에서 원본 데이터를 다운로드할 수 있습니다.

- `accommodations` 테이블의 원본 데이터.
- `zipcode` 테이블의 원본 데이터.

다음 절차에 따라 테이블을 생성하고 테스트 데이터를 로드합니다.

테이블을 생성하고 테스트 데이터를 로드하려면

1. Amazon Redshift 쿼리 편집기를 엽니다. 쿼리 에디터 작업에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 쿼리 에디터를 사용하여 데이터베이스 쿼리 섹션을 참조하세요.
2. 이 튜토리얼에서 사용하는 테이블이 데이터베이스에 이미 있는 경우 해당 테이블을 삭제합니다. 자세한 내용은 3단계: 리소스 정리 단원을 참조하십시오.
3. 각 숙박 시설의 지리학적 위치(경도 및 위도), 목록 이름 및 기타 비즈니스 데이터를 저장할 `accommodations` 테이블을 생성합니다.

   이 튜토리얼에서는 독일 베를린의 방 임대를 살펴봅니다. `shape` 열에는 숙박 시설 위치의 지리학적 점이 저장됩니다. 다른 열에는 임대에 대한 정보가 들어 있습니다.

   `accommodations` 테이블을 생성하려면 Amazon Redshift 쿼리 편집기에서 다음 SQL 문을 실행합니다.

   ```
   CREATE TABLE public.accommodations (
     id INTEGER PRIMARY KEY,
     shape GEOMETRY,
     name VARCHAR(100),
     host_name VARCHAR(100),
     neighbourhood_group VARCHAR(100),
     neighbourhood VARCHAR(100),
     room_type VARCHAR(100),
     price SMALLINT,
     minimum_nights SMALLINT,
     number_of_reviews SMALLINT,
     last_review DATE,
     reviews_per_month NUMERIC(8,2),
     calculated_host_listings_count SMALLINT,
     availability_365 SMALLINT
   );
   ```
4. 쿼리 편집기에서 베를린 우편 번호를 저장할 `zipcode` 테이블을 생성합니다.

   우편 번호는 `wkb_geometry` 열에 다각형으로 정의됩니다. 나머지 열은 우편 번호에 대한 추가 공간 메타데이터를 설명합니다.

   `zipcode` 테이블을 생성하려면 Amazon Redshift 쿼리 편집기에서 다음 SQL 문을 실행합니다.

```
CREATE TABLE public.zipcode (
  ogc_field INTEGER PRIMARY KEY NOT NULL,
  wkb_geometry GEOMETRY,
  gml_id VARCHAR(256),
  spatial_name VARCHAR(256),
  spatial_alias VARCHAR(256),
  spatial_type VARCHAR(256)
 );
```

5. 샘플 데이터를 사용하여 테이블을 로드합니다.

   이 튜토리얼의 샘플 데이터는 인증된 모든 AWS 사용자에게 읽기 권한을 허용하는 Amazon S3 버킷에서 제공됩니다. Amazon S3에 대한 액세스를 허용하는 유효한 AWS 자격 증명을 제공해야 합니다.

   테이블에 테스트 데이터를 로드하려면 다음 COPY 명령을 실행합니다. *account-number*를 AWS 계정 번호로 바꿉니다. 자격 증명 문자열에서 작은따옴표로 묶이는 구간에는 공백이나 줄 바꿈이 있을 수 없습니다.

```
COPY public.accommodations
FROM 's3://redshift-downloads/spatial-data/accommodations.csv'
DELIMITER ';'
IGNOREHEADER 1 REGION 'us-east-1'
CREDENTIALS 'aws_iam_role=arn:aws:iam::account-number:role/mySpatialDemoRole';
```

```
COPY public.zipcode
FROM 's3://redshift-downloads/spatial-data/zipcode.csv'
DELIMITER ';'
IGNOREHEADER 1 REGION 'us-east-1'
CREDENTIALS 'aws_iam_role=arn:aws:iam::account-number:role/mySpatialDemoRole';
```

6. 다음 명령을 실행하여 각 테이블이 제대로 로드되었는지 확인합니다.

```
select count(*) from accommodations;
```

```
select count(*) from zipcode;
```

   다음 결과는 테스트 데이터의 각 테이블에 있는 행 수를 보여줍니다.

테이블 이름	행
숙박 시설	22,248
우편 번호	190

2단계: 공간 데이터 쿼리

테이블이 생성되고 로드된 후 SQL SELECT 문을 사용하여 테이블을 쿼리할 수 있습니다. 다음 쿼리는 검색할 수 있는 일부 정보를 보여줍니다. 요구 사항을 충족하기 위해 공간 함수를 사용하는 다른 많은 쿼리를 작성할 수 있습니다.

공간 데이터를 쿼리하려면

1. 쿼리하여 다음과 같이 accommodations 테이블에 저장된 총 목록 수를 구합니다. 공간 참조 시스템은 고유한 공간 참조 식별자 4326을 갖는 WGS(World Geodetic System) 84입니다.

```
SELECT count(*) FROM public.accommodations WHERE ST_SRID(shape) = 4326;
```

```
 count
-------
 22248
```

2. 몇 가지 추가 속성과 함께 WKT(Well-Known Text) 형식의 지오메트리 객체를 가져옵니다. 또한 이 우편 번호 데이터가 공간 참조 ID(SRID) 4326을 사용하는 WGS(World Geodetic System) 84에도 저장되어 있는지 검증할 수 있습니다. 공간 데이터는 상호 운용 가능하도록 동일한 공간 참조 시스템에 저장되어야 합니다.

```
SELECT ogc_field, spatial_name, spatial_type, ST_SRID(wkb_geometry),
 ST_AsText(wkb_geometry)
FROM public.zipcode
ORDER BY spatial_name;
```

```
ogc_field  spatial_name  spatial_type  st_srid  st_astext
-----------------------------------------------------------------
```

```
0           10115        Polygon        4326     POLYGON((...))
4           10117        Polygon        4326     POLYGON((...))
8           10119        Polygon        4326     POLYGON((...))
...
(190 rows returned)
```

3. 베를린의 자치구인 베를린 미테(10117)의 다각형(GeoJSON 형식), 차원, 이 다각형의 점 수를 선택합니다.

```
SELECT ogc_field, spatial_name, ST_AsGeoJSON(wkb_geometry),
 ST_Dimension(wkb_geometry), ST_NPoints(wkb_geometry)
FROM public.zipcode
WHERE spatial_name='10117';
```

```
ogc_field  spatial_name  spatial_type
 st_dimension  st_npoint
--------------------------------------------------------------------------------------------
4           10117         {"type":"Polygon", "coordinates":[[[...]]]}    2
    331
```

4. 다음 SQL 명령을 실행하여 브란덴부르크 문에서 500m 이내에 몇 개의 숙박 시설이 있는지 봅니다.

```
SELECT count(*)
FROM public.accommodations
WHERE ST_DistanceSphere(shape, ST_GeomFromText('POINT(13.377704 52.516431)', 4326))
 < 500;
```

```
count
------
  29
```

5. 다음 쿼리를 실행하여 근처로 나열되는 숙박 시설에 저장된 데이터에서 브란덴부르크 문의 대략적인 위치를 구합니다.

   이 쿼리에는 하위 선택이 필요합니다. 요청한 위치가 숙박 시설에 더 가까워 이전 쿼리와 동일하지 않기 때문에 숙박 시설 수가 다릅니다.

```
WITH poi(loc) as (
```

```
  SELECT st_astext(shape) FROM accommodations WHERE name LIKE '%brandenburg gate%'
)
SELECT count(*)
FROM accommodations a, poi p
WHERE ST_DistanceSphere(a.shape, ST_GeomFromText(p.loc, 4326)) < 500;
```

```
count
------
  60
```

6. 다음 쿼리를 실행하여 가격에 따라 내림차순으로 정렬된 브란덴부르크 문 주변의 모든 숙박 시설에 대한 세부 정보를 표시합니다.

```
SELECT name, price, ST_AsText(shape)
FROM public.accommodations
WHERE ST_DistanceSphere(shape, ST_GeomFromText('POINT(13.377704 52.516431)', 4326))
 < 500
ORDER BY price DESC;
```

```
name                                                    price  st_astext
-------------------------------------------------------------------------------------------
DUPLEX APARTMENT/PENTHOUSE in 5* LOCATION! 7583         300
 POINT(13.3826510209548 52.5159819722552)
DUPLEX-PENTHOUSE IN FIRST LOCATION! 7582                300
 POINT(13.3799997083855 52.5135918444834)
...
(29 rows returned)
```

7. 다음 쿼리를 실행하여 우편 번호와 함께 가장 비싼 숙박 시설을 검색합니다.

```
SELECT
  a.price, a.name, ST_AsText(a.shape),
  z.spatial_name, ST_AsText(z.wkb_geometry)
FROM accommodations a, zipcode z
WHERE price = 9000 AND ST_Within(a.shape, z.wkb_geometry);
```

```
price   name                                st_astext
     spatial_name      st_astext
```

```
----------------------------------------------------------------------------------------------------
9000    Ueber den Dächern Berlins Zentrum    POINT(13.334436985013
 52.4979779501538)    10777             POLYGON((13.3318284987227
 52.4956021172799,...
```

8. 하위 쿼리를 사용하여 숙박 시설의 최대, 최소 또는 중간 가격을 계산합니다.

   다음 쿼리는 우편 번호별로 숙박 시설의 중간 가격을 나열합니다.

```
SELECT
  a.price, a.name, ST_AsText(a.shape),
  z.spatial_name, ST_AsText(z.wkb_geometry)
FROM accommodations a, zipcode z
WHERE
  ST_Within(a.shape, z.wkb_geometry) AND
  price = (SELECT median(price) FROM accommodations)
ORDER BY a.price;
```

```
price name                              st_astext
 spatial_name   st_astext
----------------------------------------------------------------------------------------------------

45    "Cozy room Berlin-Mitte"        POINT(13.3864349535358 52.5292016386514)
 10115          POLYGON((13.3658598465795 52.535659581048,...
...
(723 rows returned)
```

9. 다음 쿼리를 실행하여 베를린에 나열된 숙박 시설 수를 검색합니다. 핫스팟을 찾기 위해 이들은 우편 번호별로 그룹화하고 공급량별로 정렬합니다.

```
SELECT z.spatial_name as zip, count(*) as numAccommodations
FROM public.accommodations a, public.zipcode z
WHERE ST_Within(a.shape, z.wkb_geometry)
GROUP BY zip
ORDER BY numAccommodations DESC;
```

```
zip   numaccommodations
----------------------------
10245 872
10247 832
```

```
10437 733
10115 664
...
(187 rows returned)
```

3단계: 리소스 정리

클러스터는 실행하는 동안 계속해서 요금이 발생합니다. 이 튜토리얼을 마치면 샘플 클러스터를 삭제할 수 있습니다.

클러스터를 유지하면서 테스트 데이터 테이블에서 사용한 스토리지를 복원하고 싶다면 다음 명령을 실행하여 테이블을 삭제합니다.

```
drop table public.accommodations cascade;
```

```
drop table public.zipcode cascade;
```

Amazon Redshift에 shapefile 로드

COPY 명령을 사용하여 Amazon S3에 저장된 Esri shapefile을 Amazon Redshift 테이블로 수집할 수 있습니다. shapefile에는 지리학적 특성의 기하학적 위치와 속성 정보가 벡터 형식으로 저장됩니다. shapefile 형식은 점, 선 및 다각형과 같은 공간 객체를 공간적으로 설명할 수 있습니다. shapefile에 대한 자세한 내용은 Wikipedia의 Shapefile을 참조하세요.

COPY 명령은 데이터 형식 파라미터 SHAPEFILE을 지원합니다. 기본적으로 shapefile의 첫 번째 열은 GEOMETRY 또는 IDENTITY 열입니다. 모든 후속 열은 shapefile에 지정된 순서를 따릅니다. 그러나 COPY 열 매핑을 사용하여 순서를 정의할 수 있으므로 대상 테이블이 이 정확한 레이아웃에 있을 필요는 없습니다. COPY 명령 shapefile 지원에 대한 자세한 내용은 SHAPEFILE 섹션을 참조하세요.

경우에 따라 결과 지오메트리 크기가 Amazon Redshift에 지오메트리를 저장하기 위한 최댓값보다 클 수 있습니다. 이러한 경우 COPY 옵션 SIMPLIFY 또는 SIMPLIFY AUTO를 사용하여 수집하는 동안 다음과 같이 지오메트리를 단순화할 수 있습니다.

- Ramer-Douglas-Peucker 알고리즘과 주어진 허용치를 사용하여 수집하는 동안 모든 지오메트리를 단순화하려면 SIMPLIFY *tolerance*를 지정합니다.
- Ramer-Douglas-Peucker 알고리즘을 사용하여 최대 크기보다 큰 지오메트리만 단순화하려면 허용치 없이 SIMPLIFY AUTO를 지정합니다. 이 접근 방식은 최대 크기 제한 내에서 객체를 저장할 수 있을 만큼 충분히 큰 최소 허용치를 계산합니다.

- Ramer-Douglas-Peucker 알고리즘과 자동 계산된 허용치를 사용하여 최대 크기보다 큰 지오메트리만 단순화하려면 `SIMPLIFY AUTO max_tolerance`를 지정합니다. 이 접근 방식은 허용치가 최대 허용치를 초과하지 않도록 합니다.

GEOMETRY 데이터 값의 최대 크기에 대한 자세한 내용은 Amazon Redshift에서 공간 데이터를 사용할 때의 고려 사항 섹션을 참조하세요.

허용치가 충분히 낮아 레코드가 GEOMETRY 데이터 값의 최대 크기 아래로 축소될 수 없는 경우도 있습니다. 이러한 경우 COPY 명령의 MAXERROR 옵션을 사용하여 수집 오류를 모두 또는 일정 수까지 무시할 수 있습니다.

또한 COPY 명령은 GZIP shapefile 로드를 지원합니다. 이를 수행하려면 COPY GZIP 파라미터를 지정합니다. 이 옵션을 사용하면 모든 shapefile 구성 요소가 독립적으로 압축되고 동일한 압축 접미사를 공유해야 합니다.

shapefile이 포함된 프로젝션 설명 파일(.prj)이 있는 경우 Redshift는 이 파일을 사용하여 SRID(공간 참조 시스템 ID)를 결정합니다. SRID가 유효하면 결과 형상에 이 SRID가 할당됩니다. 입력 형상과 연결된 SRID 값이 없으면 결과 형상의 SRID 값은 0입니다. `SET read_srid_on_shapefile_ingestion`을 OFF로 설정하여 세션 수준에서 공간 참조 시스템 ID의 자동 검색을 사용 중지할 수 있습니다.

계산된 허용치와 함께 단순화된 레코드를 보려면 SYS_SPATIAL_SIMPLIFY 또는 SVL_SPATIAL_SIMPLIFY 시스템 뷰를 쿼리합니다. `SIMPLIFY tolerance`를 지정하면 이 뷰에 각 COPY 작업에 대한 레코드가 포함됩니다. 그렇지 않으면 단순화된 각 지오메트리에 대한 레코드가 포함됩니다. 자세한 내용은 SYS_SPATIAL_SIMPLIFY 또는 SVL_SPATIAL_SIMPLIFY을 참조하세요.

shapefile 로드의 예는 Amazon Redshift에 shapefile 로드 섹션을 참조하세요.

Amazon Redshift 공간 데이터 용어

다음 용어는 몇 가지 Amazon Redshift 공간 함수를 설명하는 데 사용됩니다.

경계 상자

지오메트리 또는 지리의 경계 상자는 지오메트리 또는 지리의 모든 점 좌표 범위의 외적(차원 간)으로 정의됩니다. 2차원 지오메트리의 경우 경계 상자는 지오메트리의 모든 점을 완전히 포함하는 직사각형입니다. 예를 들어 다각형 `POLYGON((0 0,1 0,0 2,0 0))`의 경계 상자는 점 (0, 0) 및 (1, 2)에 의해 왼쪽 하단 및 오른쪽 상단 모서리로 정의되는 직사각형입니다. Amazon Redshift는 기하학적 술어와

공간 조인의 속도를 높이기 위해 경계 상자를 미리 계산하고 지오메트리 내부에 저장합니다. 예를 들어 두 지오메트리의 경계 상자가 교차하지 않으면 이 두 지오메트리는 교차할 수 없으며 ST_Intersects 술어를 사용하는 공간 조인의 결과 집합에 있을 수 없습니다.

공간 함수를 사용하여 경계 상자에 대한 추가(AddBBox), 삭제(DropBBox) 및 지원 결정(SupportsBBox)을 할 수 있습니다. Amazon Redshift는 모든 지오메트리 하위 유형에 대한 경계 상자의 사전 계산을 지원합니다.

다음 예에서는 테이블의 기존 지오메트리를 업데이트하여 경계 상자와 함께 저장하는 방법을 보여줍니다. 클러스터가 클러스터 버전 1.0.26809 이상인 경우 기본적으로 미리 계산된 경계 상자를 사용하여 모든 새 지오메트리가 생성됩니다.

```
UPDATE my_table SET geom = AddBBox(geom) WHERE SupportsBBox(geom) = false;
```

기존 지오메트리를 업데이트한 후 업데이트된 테이블에서 VACUUM 명령을 실행하는 것이 좋습니다. 자세한 내용은 VACUUM 단원을 참조하십시오.

세션 중 지오메트리를 경계 상자로 인코딩할지 여부를 설정하려면 default_geometry_encoding 섹션을 참조하세요.

기하학적 유효성

Amazon Redshift에서 사용하는 기하학적 알고리즘은 입력 지오메트리가 유효한 지오메트리라고 가정합니다. 알고리즘에 대한 입력이 유효하지 않으면 결과가 정의되지 않습니다. 다음 섹션에서는 각 지오메트리 하위 유형에 대해 Amazon Redshift에서 사용하는 기하학적 유효성 정의를 설명합니다.

Point

다음 조건 중 하나에 해당하면 점이 유효한 것으로 간주됩니다.

- 점이 빈 점입니다.
- 모든 점 좌표가 유한 부동 소수점 숫자입니다.

점은 빈 점일 수 있습니다.

라인스트링

다음 조건 중 하나에라도 해당하면 라인스트링이 유효한 것으로 간주됩니다.

- 라인스트링이 비어 있습니다. 즉, 점을 포함하지 않습니다.
- 비어 있지 않은 라인스트링의 모든 점에 유한 부동 소수점 숫자인 좌표가 있습니다.

- 라인스트링은 비어 있지 않은 경우 1차원이어야 합니다. 즉, 점으로 퇴화할 수 없습니다.

라인스트링은 빈 점을 포함할 수 없습니다.

라인스트링에 중복된 연속 점이 있을 수 있습니다.

라인스트링에 자체 교차점이 있을 수 있습니다.

다각형

다음 조건 중 하나에라도 해당하면 다각형이 유효한 것으로 간주됩니다.

- 다각형이 비어 있습니다. 즉, 링을 포함하지 않습니다.
- 비어 있지 않은 경우 다음 모든 조건에 해당하면 다각형이 유효합니다.
  - 다각형의 모든 링이 유효합니다. 다음 모든 조건에 해당하면 링이 유효한 것으로 간주됩니다.
    - 링의 모든 점에 유한 부동 소수점 숫자인 좌표가 있습니다.
    - 링이 닫혔습니다. 즉, 첫 번째 점과 마지막 점이 일치합니다.
    - 링에 자체 교차점이 없습니다.
    - 링이 2차원입니다.
  - 다각형의 링이 일관된 방향을 갖습니다. 즉, 링을 가로지르는 경우 다각형의 내부는 오른쪽 또는 왼쪽에 있습니다. 이는 다각형의 외부 링이 시계 방향 또는 시계 반대 방향인 경우 모든 다각형의 내부 링은 동일한 시계 반대 방향 또는 시계 방향이어야 함을 의미합니다.
  - 모든 내부 링이 다각형의 외부 링 내에 있어야 합니다.
  - 내부 링은 중첩될 수 없습니다. 즉, 한 내부 링이 다른 내부 링 안에 있을 수 없습니다.
  - 내부 및 외부 링은 유한한 수의 점에서만 교차할 수 있습니다.
  - 다각형의 내부는 단순히 연결되어야 합니다.

다각형은 빈 점을 포함할 수 없습니다.

다중 점

다음 조건 중 하나에라도 해당하면 다중 점이 유효한 것으로 간주됩니다.

- 다중 점이 비어 있습니다. 즉, 점을 포함하지 않습니다.
- 다중 점이 비어 있지 않고 모든 점이 점 유효성 정의에 따라 유효합니다.

다중 점은 하나 이상의 빈 점을 포함할 수 있습니다.

다중 점에 중복 점이 있을 수 있습니다.

다중 라인스트링

다음 조건 중 하나에라도 해당하면 다중 라인스트링이 유효한 것으로 간주됩니다.

- 다중 라인스트링이 비어 있습니다. 즉, 라인스트링을 포함하지 않습니다.
- 비어 있지 않은 다중 라인스트링의 모든 라인스트링이 라인스트링 유효성 정의에 따라 유효합니다.

빈 라인스트링만으로 구성된 비어 있지 않은 다중 라인스트링은 유효한 것으로 간주됩니다.

다중 라인스트링의 빈 라인스트링은 유효성에 영향을 미치지 않습니다.

다중 라인스트링에 중복된 연속 점이 있는 라인스트링이 있을 수 있습니다.

다중 라인스트링에 자체 교차점이 있을 수 있습니다.

다중 라인스트링은 빈 점을 포함할 수 없습니다.

다중 다각형

다음 조건 중 하나에라도 해당하면 다중 다각형이 유효한 것으로 간주됩니다.

- 다중 다각형이 다각형을 포함하지 않습니다(비어 있음).
- 다중 다각형이 비어 있지 않으며 다음에 모두 해당합니다.
  - 다중 다각형의 모든 다각형이 유효합니다.
  - 다중 다각형의 두 다각형은 무한한 수의 점에서 교차할 수 없습니다. 특히 이는 두 다각형의 내부가 교차할 수 없고 유한한 수의 점에서만 접촉할 수 있음을 의미합니다.

다중 다각형의 빈 다각형은 다중 다각형을 무효화하지 않습니다.

다중 다각형은 빈 점을 포함할 수 없습니다.

지오메트리 컬렉션

다음 조건 중 하나에라도 해당하면 지오메트리 컬렉션이 유효한 것으로 간주됩니다.

- 지오메트리 컬렉션이 비어 있습니다. 즉, 지오메트리를 포함하지 않습니다.
- 비어 있지 않은 지오메트리 컬렉션의 모든 지오메트리는 유효합니다.

이 정의는 재귀적 방식이지만 중첩 지오메트리 컬렉션에 계속 적용됩니다.

지오메트리 컬렉션에는 빈 점과 빈 점이 있는 다중 점이 포함될 수 있습니다.

기하학적 단순성

Amazon Redshift에서 사용하는 기하학적 알고리즘은 입력 지오메트리가 유효한 지오메트리라고 가정합니다. 알고리즘에 대한 입력이 유효하지 않으면 단순성 확인이 정의되지 않습니다. 다음 섹션에서는 각 지오메트리 하위 유형에 대해 Amazon Redshift에서 사용하는 기하학적 단순성 정의를 설명합니다.

Point

다음 조건 중 하나에라도 해당하면 유효 점이 단순한 것으로 간주됩니다.

- 유효 점이 항상 단순한 것으로 간주됩니다.
- 빈 점이 단순한 것으로 간주됩니다.

라인스트링

다음 조건 중 하나에라도 해당하면 유효 라인스트링이 단순한 것으로 간주됩니다.

- 라인스트링이 비어 있습니다.
- 라인스트링이 비어 있지 않고 다음 모든 조건에 해당합니다.
  - 중복된 연속 점이 없습니다.
  - 일치할 수 있는 첫 번째 점과 마지막 점을 제외하고 자체 교차점이 없습니다. 즉, 라인스트링이 경계 점을 제외하고는 자체 교차점을 가질 수 없습니다.

다각형

유효 다각형은 중복된 연속 점을 포함하지 않으면 단순한 것으로 간주됩니다.

다중 점

다음 조건 중 하나에라도 해당하면 유효 다중 점이 단순한 것으로 간주됩니다.

- 다중 점이 비어 있습니다. 즉, 점을 포함하지 않습니다.
- 다중 점의 비어 있지 않은 두 점은 일치하지 않습니다.

다중 라인스트링

다음 조건 중 하나에라도 해당하면 유효 다중 라인스트링이 단순한 것으로 간주됩니다.

- 다중 라인스트링이 비어 있습니다.
- 다중 라인스트링이 비어 있지 않고 다음 모든 조건에 해당합니다.
  - 해당 라인스트링이 모두 단순합니다.
  - 다중 라인스트링의 두 라인스트링은 해당 라인스트링의 경계 점을 제외하고는 교차하지 않습니다.

빈 라인스트링으로 구성된 비어 있지 않은 다중 라인스트링은 비어 있는 것으로 간주됩니다.

다중 라인스트링의 빈 라인스트링은 단순성에 영향을 미치지 않습니다.

다중 라인스트링의 닫힌 라인스트링은 다중 라인스트링의 다른 라인스트링과 교차할 수 없습니다.

다중 라인스트링에 중복된 연속 점이 있는 라인스트링이 있을 수 없습니다.

다중 다각형

유효 다중 다각형은 중복된 연속 점을 포함하지 않으면 단순한 것으로 간주됩니다.

지오메트리 컬렉션

다음 조건 중 하나에라도 해당하면 유효 지오메트리 컬렉션이 단순한 것으로 간주됩니다.

- 지오메트리 컬렉션이 비어 있습니다. 즉, 지오메트리를 포함하지 않습니다.
- 비어 있지 않은 지오메트리 컬렉션의 모든 지오메트리는 단순합니다.

이 정의는 재귀적 방식이지만 중첩 지오메트리 컬렉션에 계속 적용됩니다.

H3

H3는 공간 좌표를 제곱미터 해상도까지 인덱싱하는 방법을 제공하는 계층적 지리공간 인덱싱 그리드 시스템입니다. 인덱싱된 데이터는 서로 다른 데이터 세트 간에 결합하고 다양한 정밀도 수준으로 집계할 수 있습니다. H3를 사용하면 가장 가까운 이웃, 최단 경로, 그래디언트 평활화 등 그리드를 기반으로 다양한 알고리즘과 최적화를 수행할 수 있습니다. H3 인덱스는 육각형 또는 오각형일 수 있는 셀을 나타냅니다. 해상도가 주어지면 공간이 계층적으로 세분됩니다. H3는 0에서 15까지의 해상도를 포함하여 16개의 해상도를 지원합니다. `0`이 가장 거칠고 15가 가장 정교합니다.

Amazon Redshift는 다음과 같은 H3 공간 함수를 제공합니다.

- H3_FromLongLat
- H3_FromPoint
- H3_Polyfill

Amazon Redshift에서 공간 데이터를 사용할 때의 고려 사항

Amazon Redshift와 함께 공간 데이터를 사용할 때의 고려 사항은 다음과 같습니다.

- `GEOMETRY` 또는 `GEOGRAPHY` 객체의 최대 크기는 1,048,447바이트입니다.

- Amazon Redshift Spectrum은 기본적으로 공간 데이터를 지원하지 않습니다. 따라서 GEOMETRY 또는 GEOGRAPHY 열이 있는 외부 테이블을 생성하거나 변경할 수 없습니다.
- Python 사용자 정의 함수(UDF)의 데이터 유형은 GEOMETRY 또는 GEOGRAPHY 데이터 유형을 지원하지 않습니다.
- GEOMETRY 또는 GEOGRAPHY 열을 Amazon Redshift 테이블의 정렬 키 또는 배포 키로 사용할 수 없습니다.
- SQL ORDER BY, GROUP BY 또는 DISTINCT 절에서는 GEOMETRY 또는 GEOGRAPHY 열을 사용할 수 없습니다.
- 많은 SQL 함수에서는 GEOMETRY 또는 GEOGRAPHY 열을 사용할 수 없습니다.
- GEOMETRY 또는 GEOGRAPHY 열에서 모든 형식으로 UNLOAD 작업을 수행할 수 없습니다. GEOMETRY 또는 GEOGRAPHY 열을 텍스트 또는 쉼표로 구분된 값(CSV) 파일로 UNLOAD할 수 있습니다. 이렇게 하면 GEOMETRY 또는 GEOGRAPHY 데이터가 16진수 EWKB 형식으로 작성됩니다. EWKB 데이터의 크기가 4MB를 초과하면 나중에 데이터를 테이블에 로드할 수 없으므로 경고가 발생합니다.
- 지원되는 GEOMETRY 또는 GEOGRAPHY 데이터 압축 인코딩은 RAW입니다.
- JDBC 또는 ODBC 드라이버를 사용하는 경우 사용자 지정 유형 매핑을 사용하십시오. 이 경우 클라이언트 애플리케이션에는 ResultSet 객체의 파라미터가 GEOMETRY 또는 GEOGRAPHY 객체인 정보가 있어야 합니다. ResultSetMetadata 작업은 VARCHAR 유형을 반환합니다.
- SHAPEFILE에서 지리적 날짜를 복사하려면 먼저 GEOMETRY 열로 수집한 다음 객체를 GEOGRAPHY 객체로 캐스팅합니다.

다음 비 공간 함수는 GEOMETRY 또는 GEOGRAPHY 유형의 입력이나 GEOMETRY 또는 GEOGRAPHY 유형의 열을 사용할 수 있습니다.

- 집계 함수 COUNT
- 조건부 표현식 COALESCE 및 NVL
- CASE 표현식
- GEOMETRY 및 GEOGRAPHY의 기본 인코딩은 RAW입니다. 자세한 내용은 압축 인코딩 단원을 참조하십시오.

Amazon Redshift에서 연합 쿼리를 사용하여 데이터 쿼리

Amazon Redshift에서 연합 쿼리를 사용하면 운영 데이터베이스, 데이터 웨어하우스 및 데이터 레이크에서 데이터를 쿼리하고 분석할 수 있습니다. 연합 쿼리 기능을 사용하면 외부 데이터베이스의 실시간 데이터에 대한 Amazon Redshift의 쿼리를 Amazon Redshift 및 Amazon S3 환경의 쿼리와 통합할 수 있습니다. 연합 쿼리는 Amazon RDS for PostgreSQL, Amazon Aurora PostgreSQL 호환 버전, Amazon RDS for MySQL 및 Amazon Aurora MySQL 호환 버전의 외부 데이터베이스와 함께 작동할 수 있습니다.

연합 쿼리를 사용하면 실시간 데이터를 BI(비즈니스 인텔리전스) 및 보고 애플리케이션의 일부로 통합할 수 있습니다. 예를 들어 Amazon Redshift에 대한 데이터 수집을 더 쉽게 만들려면 연합 쿼리를 사용하여 다음을 수행할 수 있습니다.

- 운영 데이터베이스를 직접 쿼리합니다.
- 변환을 신속하게 적용합니다.
- 복잡한 추출, 변환, 로드(ETL) 파이프라인 없이 대상 테이블에 데이터를 로드합니다.

네트워크를 통한 데이터 이동을 줄이고 성능을 향상시키기 위해 Amazon Redshift는 연합 쿼리에 대한 컴퓨팅의 일부를 원격 운영 데이터베이스에 직접 배포합니다. 또한 Amazon Redshift는 병렬 처리 용량을 사용하여 필요에 따라 이러한 쿼리 실행을 지원합니다.

페더레이션 쿼리를 실행하면 Amazon Redshift는 먼저 리더 노드에서 RDS 또는 Aurora DB 클러스터 DB 인스턴스로 클라이언트 연결을 설정하여 테이블 메타데이터를 검색합니다. Amazon Redshift는 컴퓨팅 노드에서 조건자를 푸시다운하여 하위 쿼리를 실행하고 결과 행을 검색합니다. 그런 다음 Amazon Redshift는 추가 처리를 위해 컴퓨팅 노드 간에 결과 행을 배포합니다.

Amazon Aurora PostgreSQL 데이터베이스 또는 Amazon RDS for PostgreSQL 데이터베이스로 전송된 쿼리에 대한 세부 정보가 시스템 뷰 SVL_FEDERATED_QUERY에 기록됩니다.

주제

- Amazon Redshift를 사용하여 페더레이션 데이터에 액세스할 때의 고려 사항

PostgreSQL에 대한 연합 쿼리 사용 시작하기

연합 쿼리를 생성하려면 다음 일반 워크플로우를 따릅니다.

1. Amazon Redshift 클러스터에서 Amazon RDS 또는 Aurora PostgreSQL DB 인스턴스로의 연결을 설정합니다.

   이를 위해 RDS PostgreSQL 또는 Aurora PostgreSQL DB 인스턴스가 Amazon Redshift 클러스터로부터의 연결을 허용할 수 있는지 확인합니다. Amazon Redshift 클러스터와 Amazon RDS 또는 Aurora PostgreSQL 인스턴스가 동일한 Virtual Private Cloud(VPC) 및 서브넷 그룹에 있는 것이 좋습니다. 이런 방식으로 RDS 또는 Aurora PostgreSQL DB 인스턴스에 대한 보안 그룹의 인바운드 규칙에 Amazon Redshift 클러스터의 보안 그룹을 추가할 수 있습니다.

   또한 RDS 또는 Aurora PostgreSQL 인스턴스에 Amazon Redshift가 연결할 수 있게 하는 VPC 피어링 또는 기타 네트워킹을 설정할 수 있습니다. VPC 네트워킹에 대한 자세한 내용은 다음을 참조하세요.

   - Amazon VPC 피어링 설명서의 VPC 피어링이란?
   - Amazon RDS 사용 설명서의 VPC에서 DB 인스턴스를 사용한 작업

   > **Note**
   >
   > Enhanced VPC Routing을 활성화해야 하는 경우가 있습니다. 예를 들어 Amazon Redshift 클러스터가 RDS 또는 Aurora PostgreSQL 인스턴스와 다른 VPC에 있거나, 동일한 VPC에 있어도 라우팅에 의해 요구되는 경우가 있습니다. 그렇지 않으면 연합 쿼리를 실행할 때 시간 제한 오류가 발생할 수 있습니다.

2. RDS PostgreSQL 및 Aurora PostgreSQL 데이터베이스에 대한 보안 암호를 AWS Secrets Manager에서 설정합니다. 그런 다음 AWS Identity and Access Management(IAM) 액세스 정책 및 역할에서 보안 암호를 참조합니다. 자세한 내용은 연합 쿼리 사용을 위해 비밀 및 IAM 역할 생성 단원을 참조하십시오.

   > **Note**
   >
   > 클러스터에서 향상된 VPC 라우팅을 사용하는 경우 AWS Secrets Manager에 대한 인터페이스 VPC 엔드포인트를 구성해야 할 수 있습니다. 이는 Amazon Redshift 클러스터의 VPC와 서브넷이 퍼블릭 AWS Secrets Manager 엔드포인트에 액세스할 수 없는 경우에 필요합

니다. VPC 인터페이스 엔드포인트를 사용하면 VPC의 Amazon Redshift 클러스터와 AWS Secrets Manager 간 통신이 VPC에서 엔드포인트 인터페이스로 비공개로 라우팅됩니다. 자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 생성을 참조하세요.

3. 이전에 생성한 IAM 역할을 Amazon Redshift 클러스터에 적용합니다. 자세한 내용은 연합 쿼리 사용을 위해 비밀 및 IAM 역할 생성 단원을 참조하십시오.
4. 외부 스키마를 사용하여 RDS PostgreSQL 및 Aurora PostgreSQL 데이터베이스에 연결합니다. 자세한 내용은 CREATE EXTERNAL SCHEMA 단원을 참조하십시오. 연합 쿼리를 사용하는 방법에 대한 예제는 연합 쿼리 사용 예 단원을 참조하세요.
5. RDS PostgreSQL 및 Aurora PostgreSQL 데이터베이스를 참조하는 외부 스키마를 참조 중인 SQL 쿼리를 실행합니다.

AWS CloudFormation으로 PostgreSQL에 대한 연합 쿼리 사용 시작하기

연합 쿼리를 사용하여 운영 데이터베이스 간에 쿼리할 수 있습니다. 이 시작 안내서에서는 샘플 AWS CloudFormation 스택을 사용하여 Amazon Redshift 클러스터에서 Aurora PostgreSQL 서버리스 데이터베이스로의 연합 쿼리를 사용하여 설정을 자동화할 수 있습니다. 리소스를 프로비저닝하기 위해 SQL 문을 실행할 필요 없이 빠르게 시작하고 실행할 수 있습니다.

스택은 샘플 데이터가 있는 테이블을 포함하는 Aurora PostgreSQL 인스턴스를 참조하는 외부 스키마를 생성합니다. Redshift 클러스터에서 외부 스키마의 테이블을 쿼리할 수 있습니다.

대신 CloudFormation을 사용하지 않고 SQL 문을 실행하여 외부 스키마를 설정하여 연합 쿼리를 시작하려는 경우 PostgreSQL에 대한 연합 쿼리 사용 시작하기 섹션을 참조하세요.

연합 쿼리를 위해 CloudFormation 스택을 실행하기 전에 Data API가 설정된 Amazon Aurora PostgreSQL 호환 버전 서버리스 데이터베이스가 있는지 확인합니다. 데이터베이스 속성에서 Data API를 설정할 수 있습니다. 설정을 찾을 수 없으면 Aurora PostgreSQL의 서버리스 인스턴스를 실행하고 있는지 다시 확인하세요. 또한 RA3 노드를 사용하는 Amazon Redshift 클러스터가 있는지 확인하세요. Redshift 클러스터와 서버리스 Aurora PostgreSQL 인스턴스가 동일한 Virtual Private Cloud(VPC) 및 서브넷 그룹에 있는 것이 좋습니다. 이런 방식으로 Aurora PostgreSQL 데이터베이스 인스턴스에 대한 보안 그룹의 인바운드 규칙에 Amazon Redshift 클러스터의 보안 그룹을 추가할 수 있습니다.

Amazon Redshift 클러스터 설정 시작하기에 대한 자세한 내용은 Amazon Redshift 프로비저닝된 데이터 웨어하우스 시작하기를 참조하세요. CloudFormation으로 리소스 설정에 대한 자세한 내용은 AWS

CloudFormation이란 무엇인가요?를 참조하세요. Aurora DB 클러스터 데이터베이스 설정에 대한 자세한 내용은 Aurora DB 클러스터 Serverless v1 DB 클러스터 생성을 참조하세요.

Redshift 연합 쿼리를 위한 CloudFormation 스택 시작

다음 절차에 따라 Amazon Redshift용 CloudFormation 스택을 시작하여 연합 쿼리를 사용합니다. 이렇게 하기 전에 Amazon Redshift 클러스터와 서버리스 Aurora PostgreSQL 인스턴스를 설정했는지 확인합니다.

연합 쿼리용 CloudFormation 스택을 시작하려면

1. 여기에서 CFN 스택 시작(Launch CFN stack)을 클릭하여 AWS Management Console에서 CloudFormation 서비스를 시작합니다.

   메시지가 나타나면 로그인합니다.

   Amazon S3에 저장된 CloudFormation 템플릿 파일을 참조하여 스택 생성 프로세스가 시작됩니다. CloudFormation 템플릿은 스택을 구성하는 AWS 리소스를 선언하는 JSON 형식의 텍스트 파일입니다.
2. 다음(Next)을 선택하여 스택 세부 정보를 입력합니다.
3. 파라미터(Parameters)에서 클러스터에 대해 다음을 입력합니다.

   - Amazon Redshift 클러스터 이름(예: **`ra3-consumer-cluster`**)
   - 특정 데이터베이스 이름(예: **`dev`**)
   - 데이터베이스 사용자의 이름입니다(예: **`consumeruser`**).

   또한 사용자, 데이터베이스 이름, 포트 및 엔드포인트를 포함하여 Aurora DB 클러스터 데이터베이스에 대한 파라미터를 입력합니다. 스택은 여러 데이터베이스 객체를 생성하므로 테스트 클러스터와 테스트 서버리스 데이터베이스를 사용하는 것이 좋습니다.

   Next(다음)를 선택합니다.

   스택 옵션이 나타납니다.
4. 다음(Next)을 선택하여 기본 설정을 적용합니다.
5. 기능(Capabilities)에서 AWS CloudFormation 에서 IAM 리소스를 생성할 수 있음을 승인합니다(I acknowledge that AWS CloudFormation might create IAM resources)를 선택합니다.
6. 스택 생성을 선택합니다.

스택 생성을 선택합니다. CloudFormation은 약 10분 정도 소요되는 템플릿 리소스를 프로비저닝하고 외부 스키마를 생성합니다.

스택이 생성되는 동안 오류가 발생하면 다음을 수행합니다.

- 오류 해결에 도움이 되는 정보는 CloudFormation 이벤트(Events) 탭을 참조합니다.
- Redshift 클러스터에 대해 올바른 이름, 데이터베이스 이름 및 데이터베이스 사용자 이름을 입력했는지 확인합니다. 또한 Aurora PostgreSQL 인스턴스의 파라미터도 확인합니다.
- 클러스터에 RA3 노드가 있는지 확인합니다.
- 데이터베이스와 Redshift 클러스터가 동일한 서브넷 및 보안 그룹에 있는지 확인합니다.

외부 스키마에서 데이터 쿼리

다음 절차를 사용하려면 설명된 클러스터와 데이터베이스에서 쿼리를 실행하는 데 필요한 권한이 있는지 확인해야 합니다.

연합 쿼리를 사용하여 외부 데이터베이스를 쿼리하려면

1. Redshift 쿼리 편집기와 같은 클라이언트 도구를 사용하여 스택을 생성할 때 입력한 Redshift 데이터베이스에 연결합니다.
2. 스택에서 생성한 외부 스키마를 쿼리합니다.

   ```
   select * from svv_external_schemas;
   ```

   SVV_EXTERNAL_SCHEMAS 보기는 사용 가능한 외부 스키마에 대한 정보를 반환합니다. 이 경우 스택에 의해 생성된 외부 스키마 `myfederated_schema`가 반환됩니다. 설정한 경우 다른 외부 스키마가 반환될 수도 있습니다. 보기는 스키마의 연결된 데이터베이스도 반환합니다. 데이터베이스는 스택을 생성할 때 입력한 Aurora DB 클러스터 데이터베이스입니다. 스택은 Aurora DB 클러스터 데이터베이스에 `category`라는 테이블과 `sales`라는 또 다른 테이블을 추가합니다.
3. Aurora PostgreSQL 데이터베이스를 참조하는 외부 스키마의 테이블에서 SQL 쿼리를 실행합니다. 다음 예에서는 쿼리를 보여줍니다.

   ```
   SELECT count(*) FROM myfederated_schema.category;
   ```

   `category` 테이블은 여러 레코드를 반환합니다. `sales` 테이블에서 레코드를 반환할 수도 있습니다.

```
SELECT count(*) FROM myfederated_schema.sales;
```

더 많은 예제는 연합 쿼리 사용 예를 참조합니다.

MySQL에 대한 연합 쿼리 사용 시작하기

MySQL 데이터베이스에 대한 연합 쿼리를 생성하려면 다음과 같은 일반적인 접근 방식을 따릅니다.

1. Amazon Redshift 클러스터에서 Amazon RDS 또는 Aurora MySQL DB 인스턴스로의 연결을 설정합니다.

   이를 위해 RDS MySQL 또는 Aurora MySQL DB 인스턴스가 Amazon Redshift 클러스터로부터의 연결을 허용할 수 있는지 확인합니다. Amazon Redshift 클러스터와 Amazon RDS 또는 Aurora MySQL 인스턴스가 동일한 Virtual Private Cloud(VPC) 및 서브넷 그룹에 있는 것이 좋습니다. 이런 방식으로 RDS 또는 Aurora MySQL DB 인스턴스에 대한 보안 그룹의 인바운드 규칙에 Amazon Redshift 클러스터의 보안 그룹을 추가할 수 있습니다.

   또한 RDS 또는 Aurora MySQL 인스턴스에 Amazon Redshift가 연결할 수 있게 하는 VPC 피어링 또는 기타 네트워킹을 설정할 수 있습니다. VPC 네트워킹에 대한 자세한 내용은 다음을 참조하세요.

   - Amazon VPC 피어링 설명서의 VPC 피어링이란?
   - Amazon RDS 사용 설명서의 VPC에서 DB 인스턴스를 사용한 작업

   > **Note**
   >
   > Amazon Redshift 클러스터가 RDS 또는 Aurora MySQL 인스턴스와 다른 VPC에 있는 경우 Enhanced VPC Routing을 사용합니다. 그렇지 않으면 연합 쿼리를 실행할 때 시간 제한 오류가 발생할 수 있습니다.

2. AWS Secrets Manager에서 RDS MySQL 및 Aurora MySQL 데이터베이스에 대한 보안 암호를 설정합니다. 그런 다음 AWS Identity and Access Management(IAM) 액세스 정책 및 역할에서 보안 암호를 참조합니다. 자세한 내용은 연합 쿼리 사용을 위해 비밀 및 IAM 역할 생성 단원을 참조하십시오.

   > **Note**
   >
   > 클러스터에서 향상된 VPC 라우팅을 사용하는 경우 AWS Secrets Manager에 대한 인터페이스 VPC 엔드포인트를 구성해야 할 수 있습니다. 이는 Amazon Redshift 클러스터의 VPC와 서브넷이 퍼블릭 AWS Secrets Manager 엔드포인트에 액세스할 수 없는 경우에 필요합

니다. VPC 인터페이스 엔드포인트를 사용하면 VPC의 Amazon Redshift 클러스터와 AWS Secrets Manager 간 통신이 VPC에서 엔드포인트 인터페이스로 비공개로 라우팅됩니다. 자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 생성을 참조하세요.

3. 이전에 생성한 IAM 역할을 Amazon Redshift 클러스터에 적용합니다. 자세한 내용은 연합 쿼리 사용을 위해 비밀 및 IAM 역할 생성 단원을 참조하십시오.
4. 외부 스키마를 사용하여 RDS MySQL 및 Aurora MySQL 데이터베이스에 연결합니다. 자세한 내용은 CREATE EXTERNAL SCHEMA 단원을 참조하십시오. 연합 쿼리 사용 방법에 대한 예는 MySQL에서 연합 쿼리를 사용하는 예 섹션을 참조하세요.
5. RDS MySQL 및 Aurora MySQL 데이터베이스를 참조하는 외부 스키마를 참조 중인 SQL 쿼리를 실행합니다.

연합 쿼리 사용을 위해 비밀 및 IAM 역할 생성

다음 단계에서는 연합 쿼리에 사용할 비밀 및 IAM 역할을 생성하는 방법을 보여 줍니다.

사전 조건

연합 쿼리에 사용할 보안 암호와 IAM 역할을 생성하려면 다음과 같은 사전 조건이 충족되어야 합니다.

- 사용자 이름 및 암호 인증을 사용하는 RDS PostgreSQL, Aurora PostgreSQL DB 인스턴스, RDS MySQL 또는 Aurora MySQL DB 인스턴스.
- 연합 쿼리를 지원하는 클러스터 유지 관리 버전을 가진 Amazon Redshift 클러스터

AWS Secrets Manager를 사용하여 비밀(사용자 이름 및 암호)을 생성하려면

1. RDS 또는 Aurora DB 클러스터 인스턴스를 소유한 계정으로 Secrets Manager 콘솔에 로그인합니다.
2. 새 비밀 저장(Store a new secret)을 선택합니다.
3. RDS 데이터베이스 자격 증명(Credentials for RDS database) 타일을 선택합니다. 사용자 이름 및 암호에 인스턴스의 값을 입력합니다. 암호화 키(Encryption key) 값을 확인하거나 선택합니다. 그런 다음 비밀에 액세스할 RDS 데이터베이스를 선택합니다.

> **Note**
>
> 기본 암호화 키(DefaultEncryptionKey)를 사용하는 것이 좋습니다. 사용자 지정 암호화 키를 사용하는 경우 비밀에 액세스하는 데 사용되는 IAM 역할을 키 사용자로 추가해야 합니다.

4. 비밀의 이름을 입력하고 기본 선택 사항으로 생성 단계를 계속한 다음 저장(Store)을 선택합니다.
5. 비밀을 확인하고 비밀을 식별하기 위해 생성한 비밀 ARN 값을 기록해 둡니다.

비밀을 사용하여 보안 정책 생성

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. 다음과 유사한 JSON으로 정책을 생성합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessSecret",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecretVersionIds"
            ],
            "Resource": "arn:aws:secretsmanager:us-west-2:123456789012:secret:my-
rds-secret-VNenFy"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword",
                "secretsmanager:ListSecrets"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

비밀을 검색하려면 나열 및 읽기 작업을 수행해야 합니다. 리소스를 생성한 특정 비밀로 제한하는 것이 좋습니다. 이를 위해 비밀의 Amazon 리소스 이름(ARN)을 사용하여 리소스를 제한합니다. IAM 콘솔의 시각적 편집기를 사용하여 권한과 리소스를 지정할 수도 있습니다.

3. 정책 이름을 지정하고 생성을 완료합니다.
4. IAM 역할(IAM roles)로 이동합니다.
5. Redshift - 사용자 지정에 대한 IAM 역할을 생성합니다.
6. 방금 생성한 IAM 정책을 기존 IAM 역할에 연결하거나 새 IAM 역할을 생성하고 정책을 연결합니다.
7. IAM 역할의 신뢰 관계(Trust relationships) 탭에서 신뢰 엔터티 `redshift.amazonaws.com`이(가) 포함되어 있음을 확인합니다.
8. 생성한 역할 ARN을 기록해 둡니다. 이 ARN은 비밀에 액세스할 수 있습니다.

Amazon Redshift 클러스터에 IAM 역할을 연결하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 클러스터(Clusters)를 선택합니다. 현재 AWS 리전의 계정에 대한 클러스터가 나열됩니다.
3. 클러스터에 대한 자세한 내용을 보려면 목록에서 클러스터 이름을 선택합니다.
4. 작업(Actions)에서 IAM 역할 관리(Manage IAM roles)를 선택합니다. IAM 역할 관리(Manage IAM roles) 페이지가 나타납니다.
5. 클러스터에 IAM 역할을 추가합니다.

연합 쿼리 사용 예

다음 예에서는 연합 쿼리를 실행하는 방법을 보여줍니다. Amazon Redshift 데이터베이스에 연결된 SQL 클라이언트를 사용하여 SQL을 실행합니다.

PostgreSQL에서 연합 쿼리를 사용하는 예

다음 예에서는 Amazon Redshift 데이터베이스, Aurora PostgreSQL 데이터베이스 및 Amazon S3를 참조하는 연합 쿼리를 설정하는 방법을 보여줍니다. 이 예에서는 연합 쿼리의 작동 방식을 보여줍니다.

사용자 환경에서 실행하려면 사용자 환경에 맞게 변경합니다. 이 작업을 수행하기 위한 사전 조건은 PostgreSQL에 대한 연합 쿼리 사용 시작하기 단원을 참조하세요.

Aurora PostgreSQL 데이터베이스를 참조하는 외부 스키마를 생성합니다.

```
CREATE EXTERNAL SCHEMA apg
FROM POSTGRES
DATABASE 'database-1' SCHEMA 'myschema'
URI 'endpoint to aurora hostname'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-SecretsManager-RO'
SECRET_ARN 'arn:aws:secretsmanager:us-west-2:123456789012:secret:federation/test/
dataplane-apg-creds-YbVKQw';
```

Amazon S3를 참조하고 Amazon Redshift Spectrum을 사용하는 다른 외부 스키마를 생성합니다. 또한 public에 스키마를 사용할 수 있는 권한을 부여합니다.

```
CREATE EXTERNAL SCHEMA s3
FROM DATA CATALOG
DATABASE 'default' REGION 'us-west-2'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-S3';

GRANT USAGE ON SCHEMA s3 TO public;
```

Amazon Redshift 테이블의 행 수를 표시합니다.

```
SELECT count(*) FROM public.lineitem;

  count
----------
25075099
```

Aurora PostgreSQL 테이블의 행 수를 표시합니다.

```
SELECT count(*) FROM apg.lineitem;

count
-------
11760
```

Amazon S3의 행 수를 표시합니다.

```
SELECT count(*) FROM s3.lineitem_1t_part;

   count
------------
6144008876
```

Amazon Redshift, Aurora PostgreSQL 및 Amazon S3에서 테이블의 뷰를 생성합니다. 이 보기는 연합 쿼리를 실행하는 데 사용됩니다.

```
CREATE VIEW lineitem_all AS
  SELECT
 l_orderkey,l_partkey,l_suppkey,l_linenumber,l_quantity,l_extendedprice,l_discount,l_tax,l_retu
        l_shipdate::date,l_commitdate::date,l_receiptdate::date,
 l_shipinstruct ,l_shipmode,l_comment
  FROM s3.lineitem_1t_part
  UNION ALL SELECT * FROM public.lineitem
  UNION ALL SELECT * FROM apg.lineitem
     with no schema binding;
```

결과를 제한하는 술어와 함께 `lineitem_all` 보기의 행 수를 표시합니다.

```
SELECT count(*) from lineitem_all WHERE l_quantity = 10;

   count
-----------
123373836
```

매년 1월의 특정 항목에 대한 판매량을 알아봅니다.

```
SELECT extract(year from l_shipdate) as year,
       extract(month from l_shipdate) as month,
       count(*) as orders
FROM lineitem_all
WHERE extract(month from l_shipdate) = 1
AND l_quantity < 2
GROUP BY 1,2
ORDER BY 1,2;

 year | month | orders
------+-------+---------
 1992 |     1 |  196019
```

```
 1993 |     1 | 1582034
 1994 |     1 | 1583181
 1995 |     1 | 1583919
 1996 |     1 | 1583622
 1997 |     1 | 1586541
 1998 |     1 | 1583198
 2016 |     1 |   15542
 2017 |     1 |   15414
 2018 |     1 |   15527
 2019 |     1 |     151
```

대/소문자가 혼합된 이름을 사용하는 예

대/소문자가 혼합된 데이터베이스, 스키마, 테이블 또는 열 이름이 있는 지원되는 PostgreSQL 원격 데이터베이스를 쿼리하려면 `enable_case_sensitive_identifier`를 `true`로 설정합니다. 이 세션 파라미터에 대한 자세한 내용은 enable_case_sensitive_identifier 섹션을 참조하세요.

```
SET enable_case_sensitive_identifier TO TRUE;
```

일반적으로 데이터베이스 및 스키마 이름은 소문자입니다. 다음 예에서는 데이터베이스 및 스키마 이름이 소문자이고 테이블 및 열 이름이 대/소문자가 혼합된 지원되는 PostgreSQL 원격 데이터베이스에 연결하는 방법을 보여줍니다.

소문자 데이터베이스 이름(`dblower`)과 소문자 스키마 이름(`schemalower`)이 있는 Aurora PostgreSQL 데이터베이스를 참조하는 외부 스키마를 생성합니다.

```
CREATE EXTERNAL SCHEMA apg_lower
FROM POSTGRES
DATABASE 'dblower' SCHEMA 'schemalower'
URI 'endpoint to aurora hostname'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-SecretsManager-RO'
SECRET_ARN 'arn:aws:secretsmanager:us-west-2:123456789012:secret:federation/test/
dataplane-apg-creds-YbVKQw';
```

쿼리가 실행되는 세션에서 `enable_case_sensitive_identifier`를 `true`로 설정합니다.

```
SET enable_case_sensitive_identifier TO TRUE;
```

연합 쿼리를 실행하여 PostgreSQL 데이터베이스의 모든 데이터를 선택합니다. 테이블(`MixedCaseTab`)과 열(`MixedCaseName`)에 대/소문자가 혼합된 이름이 있습니다. 결과는 하나의 행(`Harry`)입니다.

```
select * from apg_lower."MixedCaseTab";
```

```
 MixedCaseName
-------
 Harry
```

다음 예에서는 대/소문자가 혼합된 데이터베이스, 스키마, 테이블 및 열 이름이 있는 지원되는 PostgreSQL 원격 데이터베이스에 연결하는 방법을 보여줍니다.

외부 스키마를 생성하기 전에 enable_case_sensitive_identifier를 true로 설정합니다. 외부 스키마를 생성하기 전에 enable_case_sensitive_identifier를 true로 설정하지 않으면 데이터베이스가 존재하지 않는다는 오류가 발생합니다.

대/소문자가 혼합된 데이터베이스 이름(UpperDB)과 스키마 이름(UpperSchema)이 있는 Aurora PostgreSQL 데이터베이스를 참조하는 외부 스키마를 생성합니다.

```
CREATE EXTERNAL SCHEMA apg_upper
FROM POSTGRES
DATABASE 'UpperDB' SCHEMA 'UpperSchema'
URI 'endpoint to aurora hostname'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-SecretsManager-RO'
SECRET_ARN 'arn:aws:secretsmanager:us-west-2:123456789012:secret:federation/test/
dataplane-apg-creds-YbVKQw';
```

연합 쿼리를 실행하여 PostgreSQL 데이터베이스의 모든 데이터를 선택합니다. 테이블(MixedCaseTab)과 열(MixedCaseName)에 대/소문자가 혼합된 이름이 있습니다. 결과는 하나의 행(Harry)입니다.

```
select * from apg_upper."MixedCaseTab";
```

```
 MixedCaseName
-------
 Harry
```

MySQL에서 연합 쿼리를 사용하는 예

다음 예에서는 Aurora MySQL 데이터베이스를 참조하는 연합 쿼리를 설정하는 방법을 보여줍니다. 이 예제에서는 연합 쿼리의 작동 방식을 보여 줍니다. 사용자 환경에서 실행하려면 사용자 환경에 맞게 변경합니다. 이 작업을 수행하기 위한 사전 조건은 MySQL에 대한 연합 쿼리 사용 시작하기 단원을 참조하세요.

이 예는 다음 사전 조건에 따라 다릅니다.

- Aurora MySQL 데이터베이스에 대해 Secrets Manager에서 설정한 보안 암호입니다. 이 보안 암호는 IAM 액세스 정책 및 역할에서 참조됩니다. 자세한 내용은 연합 쿼리 사용을 위해 비밀 및 IAM 역할 생성 단원을 참조하십시오.
- Amazon Redshift와 Aurora MySQL을 연결하도록 설정된 보안 그룹입니다.

Aurora MySQL 데이터베이스를 참조하는 외부 스키마를 생성합니다.

```
CREATE EXTERNAL SCHEMA amysql
FROM MYSQL
DATABASE 'functional'
URI 'endpoint to remote hostname'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-SecretsManager-RO'
SECRET_ARN 'arn:aws:secretsmanager:us-west-2:123456789012:secret:federation/test/
dataplane-apg-creds-YbVKQw';
```

Aurora MySQL 테이블의 예제 SQL 선택을 실행하여 Aurora MySQL의 직원 테이블에서 한 행을 표시합니다.

```
SELECT level FROM amysql.employees LIMIT 1;

 level
-------
     8
```

Amazon Redshift와 지원되는 PostgreSQL 및 MySQL 데이터베이스 간의 데이터 형식 차이점

다음 표에서는 Amazon Redshift 데이터 형식을 대응하는 Amazon RDS PostgreSQL 또는 Aurora PostgreSQL 데이터 형식에 매핑하는 방법을 보여줍니다.

Amazon Redshift 데이터 형식	RDS PostgreSQL 또는 Aurora PostgreSQL 데이터 형식	설명
SMALLINT	SMALLINT	2바이트 부호화 정수
INTEGER	INTEGER	4바이트 부호화 정수
BIGINT	BIGINT	8바이트 부호화 정수
DECIMAL	DECIMAL	정밀도를 선택할 수 있는 정확한 숫자
REAL	REAL	단정밀도 부동 소수점 수
DOUBLE PRECISION	DOUBLE PRECISION	배정밀도 부동 소수점 수
BOOLEAN	BOOLEAN	논리적 부울(true/false)
CHAR	CHAR	고정 길이 문자열
VARCHAR	VARCHAR	사용자 정의 제한이 포함된 가변 길이 문자열
날짜	날짜	날짜(년, 월, 일)
TIMESTAMP	TIMESTAMP	날짜/시간(시간대 제외)
TIMESTAMPTZ	TIMESTAMPTZ	날짜/시간(시간대 포함)
GEOMETRY	PostGIS GEOMETRY	공간 데이터

다음 RDS PostgreSQL 및 Aurora PostgreSQL 데이터 형식은 Amazon Redshift에서 VARCHAR(64K)로 변환됩니다.

- JSON, JSONB
- 배열
- BIT, BIT VARYING
- BYTEA
- 복합 유형
- 날짜 및 시간 형식 INTERVAL, TIME, TIME WITH TIMEZONE
- 열거 유형
- 화폐 유형
- 네트워크 주소 유형
- 숫자 유형 SERIAL, BIGSERIAL, SMALLSERIAL 및 MONEY
- 객체 식별자 유형
- pg_lsn 유형
- 유사 유형
- 범위 유형
- 텍스트 검색 유형
- TXID_SNAPSHOT
- UUID
- XML 형식

다음 표에서는 Amazon Redshift 데이터 형식을 대응하는 Amazon RDS MySQL 또는 Aurora MySQL 데이터 형식에 매핑하는 방법을 보여줍니다.

Amazon Redshift 데이터 형식	RDS MySQL 또는 Aurora MySQL 데이터 형식	설명
BOOLEAN	TINYINT(1)	논리적 부울(true 또는 false)
SMALLINT	TINYINT(UNSIGNED)	2바이트 부호화 정수
SMALLINT	SMALLINT	2바이트 부호화 정수
INTEGER	SMALLINT UNSIGNED	4바이트 부호화 정수

Amazon Redshift 데이터 형식	RDS MySQL 또는 Aurora MySQL 데이터 형식	설명
INTEGER	MEDIUMINT (UNSIGNED)	4바이트 부호화 정수
INTEGER	INT	4바이트 부호화 정수
BIGINT	INT UNSIGNED	8바이트 부호화 정수
BIGINT	BIGINT	8바이트 부호화 정수
DECIMAL	BIGINT UNSIGNED	정밀도를 선택할 수 있는 정확한 숫자
DECIMAL	DECIMAL(M,D)	정밀도를 선택할 수 있는 정확한 숫자
REAL	FLOAT	단정밀도 부동 소수점 수
DOUBLE PRECISION	DOUBLE	배정밀도 부동 소수점 수
CHAR	CHAR	고정 길이 문자열
VARCHAR	VARCHAR	사용자 정의 제한이 포함된 가변 길이 문자열
날짜	날짜	날짜(년, 월, 일)
TIME	TIME	시간(시간대 제외)
TIMESTAMP	TIMESTAMP	날짜/시간(시간대 제외)
TIMESTAMP	DATETIME	시간(시간대 제외)
VARCHAR(4)	YEAR	연도를 나타내는 가변 길이 문자

TIME 데이터가 범위(00:00:00~24:00:00)를 벗어나면 오류가 발생합니다.

다음 RDS MySQL 및 Aurora MySQL 데이터 형식은 Amazon Redshift에서 VARCHAR(64K)로 변환됩니다.

- BIT
- BINARY
- VARBINARY
- TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB
- TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT
- ENUM
- SET
- SPATIAL

Amazon Redshift를 사용하여 페더레이션 데이터에 액세스할 때의 고려 사항

일부 Amazon Redshift 기능은 연합 데이터에 대한 액세스를 지원하지 않습니다. 다음과 같은 관련 제한 사항 및 고려 사항을 찾을 수 있습니다.

Amazon Redshift에서 연합 쿼리를 사용할 때 제한 사항과 고려 사항은 다음과 같습니다.

- 연합 쿼리는 외부 데이터 원본에 대한 읽기 액세스를 지원합니다. 외부 데이터 원본에서 데이터베이스 객체를 작성하거나 생성할 수 없습니다.
- 경우에 따라 Amazon Redshift 이외 AWS 리전에서 Amazon RDS 또는 Aurora DB 클러스터 데이터베이스에 액세스할 수 있습니다. 이러한 경우 일반적으로 AWS 리전 간 데이터 전송에 대한 네트워크 지연 및 청구 요금이 발생합니다. Amazon Redshift 클러스터와 동일한 AWS 리전에 로컬 엔드포인트가 있는 Aurora 글로벌 데이터베이스를 사용하는 것이 좋습니다. Aurora 글로벌 데이터베이스는 두 AWS 리전 간 스토리지 기반 복제에 전용 인프라를 사용하며 일반적인 지연 시간은 1초 미만입니다.
- Amazon RDS 또는 Aurora DB 클러스터 액세스 비용을 고려하세요. 예를 들어 이 기능을 사용하여 Aurora DB 클러스터에 액세스할 경우 Aurora DB 클러스터 요금은 IOPS를 기준으로 부과됩니다.
- 페더레이션 쿼리는 RDS 또는 Aurora DB 클러스터에서 Amazon Redshift에 대한 액세스를 사용하지 않습니다.

- 페더레이션 쿼리는 Amazon Redshift와 Amazon RDS 또는 Aurora DB 클러스터 둘 다 사용할 수 있는 AWS 리전에서만 사용할 수 있습니다.
- 연합 쿼리는 현재 `ALTER SCHEMA`을(를) 지원하지 않습니다. 스키마를 변경하려면 `DROP`을(를) 사용한 다음 `CREATE EXTERNAL SCHEMA`을(를) 사용합니다.
- 연합 쿼리는 동시성 확장과 함께 작동하지 않습니다.
- 연합 쿼리는 현재 PostgreSQL 외부 데이터 래퍼를 통한 액세스를 지원하지 않습니다.
- RDS MySQL 또는 Aurora MySQL에 대한 연합 쿼리는 READ COMMITTED 수준에서 트랜잭션 격리를 지원합니다.
- 지정하지 않으면 Amazon Redshift는 포트 3306에서 RDS for MySQL 또는 Aurora MySQL에 연결합니다. MySQL용 외부 스키마를 생성하기 전에 MySQL 포트 번호를 확인합니다.
- 지정하지 않으면 Amazon Redshift는 포트 5432에서 RDS PostgreSQL 또는 Aurora PostgreSQL에 연결합니다. PostgreSQL용 외부 스키마를 생성하기 전에 PostgreSQL 포트 번호를 확인합니다.
- MySQL에서 TIMESTAMP 및 DATE 데이터 유형을 가져올 때 0 값은 NULL로 처리됩니다.
- Aurora DB 클러스터 데이터베이스 리더 엔드포인트를 사용하는 경우 '잘못된 스냅샷' 오류가 발생할 수 있습니다. 다음 방법 중 하나를 사용하여 이를 방지할 수 있습니다.
  - Aurora DB 클러스터 엔드포인트를 사용하는 대신 특정 Aurora DB 클러스터 인스턴스 엔드포인트를 사용합니다. 이 메서드는 PostgreSQL 데이터베이스의 결과에 대해 REPEATABLE READ 트랜잭션 격리를 사용합니다.
  - Aurora DB 클러스터 리더 엔드포인트를 사용하고 세션에 대해 `pg_federation_repeatable_read`를 false로 설정합니다. 이 메서드는 PostgreSQL 데이터베이스의 결과에 대해 READ COMMITTED 트랜잭션 격리를 사용합니다. Aurora DB 클러스터 리더 엔드포인트에 대한 자세한 내용은 Amazon Aurora 사용 설명서의 Aurora DB 클러스터 엔드포인트의 유형을 참조하세요. `pg_federation_repeatable_read`에 대한 자세한 내용은 pg_federation_repeatable_read 섹션을 참조하십시오.

다음은 PostgreSQL 데이터베이스에 대한 연합 쿼리로 작업할 때 트랜잭션에 대한 고려 사항입니다.

- 쿼리가 연합 테이블로 구성된 경우 리더 노드는 원격 데이터베이스에서 READ ONLY REPEATABLE READ 트랜잭션을 시작합니다. 이 트랜잭션은 Amazon Redshift 트랜잭션 기간 동안 유지됩니다.
- 리더 노드는 `pg_export_snapshot`을(를) 호출하여 원격 데이터베이스의 스냅샷을 생성하고 영향을 받는 테이블에 대한 읽기 잠금을 만듭니다.
- 컴퓨팅 노드는 트랜잭션을 시작하고 리더 노드에서 생성된 스냅샷을 사용하여 원격 데이터베이스에 쿼리를 실행합니다.

지원되는 페더레이션된 데이터베이스 버전

Amazon Redshift 외부 스키마는 외부 RDS PostgreSQL 또는 Aurora PostgreSQL에서 데이터베이스를 참조할 수 있습니다. 이 경우 다음과 같은 제한 사항이 적용됩니다.

- Aurora DB 클러스터를 참조하는 외부 스키마를 생성할 때 Aurora PostgreSQL 데이터베이스의 버전은 9.6 이상이어야 합니다.
- Amazon RDS를 참조하는 외부 스키마를 생성할 때 Amazon RDS PostgreSQL 데이터베이스의 버전은 9.6 이상이어야 합니다.

Amazon Redshift 외부 스키마는 외부 RDS MySQL 또는 Aurora MySQL에서 데이터베이스를 참조할 수 있습니다. 이 경우 다음과 같은 제한 사항이 적용됩니다.

- Aurora DB 클러스터를 참조하는 외부 스키마를 생성할 때 Aurora MySQL 데이터베이스의 버전은 5.6 이상이어야 합니다.
- Amazon RDS를 참조하는 외부 스키마를 생성할 때 RDS MySQL 데이터베이스의 버전이 5.6 이상이어야 합니다.

Amazon Redshift Spectrum

이 섹션에서는 Redshift Spectrum을 사용하여 Amazon S3의 데이터를 효율적으로 읽는 방법에 대해 설명합니다.

Amazon Redshift Spectrum을 사용하면 데이터를 Amazon Redshift 테이블에 로드하지 않고도 Amazon S3의 파일에서 정형 및 비정형 데이터를 효율적으로 쿼리하고 가져올 수 있습니다. Redshift Spectrum 쿼리는 대량 병렬 처리를 채택해 큰 데이터 집합에 대해 매우 빠르게 실행됩니다. 대부분의 처리가 Redshift Spectrum 계층에서 이루어지며, 데이터가 대부분 Amazon S3에 그대로 남습니다. 또한 다수의 클러스터가 Amazon S3의 동일한 데이터 집합에 대해 동시에 쿼리를 실행할 수 있기 때문에 각 클러스터의 데이터를 일일이 복사할 필요가 없습니다.

주제

Amazon Redshift Spectrum 개요

이 주제에서는 Redshift Spectrum을 사용하여 Amazon S3에서 효율적으로 읽는 방법에 대해 자세히 설명합니다.

Amazon Redshift Spectrum은 클러스터와 독립적인 전용 Amazon Redshift 서버에 상주합니다. Amazon Redshift는 조건자 필터링 및 집계 같은 많은 컴퓨팅 집약적 작업을 Redshift Spectrum 계층까지 푸시합니다. 따라서 Redshift Spectrum 쿼리는 다른 쿼리보다 클러스터의 처리 용량을 훨씬 적게 사용합니다. Redshift Spectrum은 또 지능적으로 조정됩니다. Redshift Spectrum은 쿼리 요구에 기반하여 대량 병렬 처리를 활용하기 위해 수천 개의 인스턴스를 사용할 수 있습니다.

파일의 구조를 정의하고 외부 데이터 카탈로그에 테이블로 등록해 Redshift Spectrum 테이블을 만듭니다. 외부 데이터 카탈로그는 AWS Glue, Amazon Athena와 함께 제공되는 데이터 카탈로그 또는 자체 Apache Hive 메타스토어일 수 있습니다. 데이터 정의 언어(DDL) 명령을 사용하거나 외부 데이터 카탈로그에 연결된 다른 도구를 사용하여 Amazon Redshift에서 외부 테이블을 생성하고 관리할 수 있습니다. 외부 데이터 카탈로그의 변경 사항은 Amazon Redshift 클러스터에서 즉시 사용할 수 있습니다.

필요할 경우, 하나 이상의 열에서 외부 테이블을 파티셔닝할 수 있습니다. 외부 테이블의 일부로 파티션을 정의하면 성능을 개선할 수 있습니다. 개선이 가능한 이유는 쿼리를 위한 데이터가 포함되지 않은 파티션을 Amazon Redshift 쿼리 옵티마이저가 제거하기 때문입니다.

Spectrum 테이블에 대한 구체화된 뷰는 비용과 성능을 크게 개선할 수 있습니다. 자세한 내용은 Amazon Redshift Spectrum의 외부 데이터 레이크 테이블에 대한 구체화된 뷰 단원을 참조하십시오.

Redshift Spectrum 테이블이 정의된 후 다른 Amazon Redshift 테이블과 똑같이 테이블을 쿼리하고 조인할 수 있습니다. Redshift Spectrum은 외부 테이블에 대한 업데이트 작업을 지원하지 않습니다. Redshift Spectrum 테이블을 여러 Amazon Redshift 클러스터에 추가하여 같은 AWS 리전의 어느 클러스터에서나 Amazon S3에 있는 동일한 데이터를 쿼리할 수 있습니다. Amazon S3 데이터 파일을 업데이트하면 어느 Amazon Redshift 클러스터에서나 해당 데이터를 즉시 사용할 수 있습니다.

액세스하는 AWS Glue 데이터 카탈로그가 보안을 강화하기 위해 암호화될 수 있습니다. AWS Glue 카탈로그가 암호화되어 있는 경우 AWS Glue 카탈로그에 액세스하려면 AWS Glue에 AWS Key Management Service(AWS KMS) 키가 필요합니다. 일부 AWS 리전에서는 AWS Glue 카탈로그 암호화를 사용할 수 없습니다. 지원되는 AWS 리전 목록은 AWS Glue Developer Guide의 Encryption and Secure Access for AWS Glue 섹션을 참조하세요. AWS Glue Data Catalog 암호화에 대한 자세한 내용은 AWS Glue Developer Guide의 Encrypting Your AWS Glue Data Catalog 섹션을 참조하세요.

> **Note**
>
> PG_TABLE_DEF, STV_TBL_PERM, PG_CLASS 또는 information_schema와 같은 표준 Amazon Redshift 테이블에 사용하는 것과 동일한 리소스를 사용하여 Redshift Spectrum 테이블에 대한 세부 정보를 볼 수 없습니다. 비즈니스 인텔리전스 또는 분석 도구가 Redshift Spectrum 외부 테이블을 인식하지 못하는 경우 애플리케이션이 SVV_EXTERNAL_TABLES 및 SVV_EXTERNAL_COLUMNS를 쿼리하도록 구성합니다.

Amazon Redshift Spectrum 리전

Redshift 스펙트럼은 리전별 설명서에 달리 명시되지 않는 한 Amazon Redshift Redshift를 사용할 수 있는 AWS 리전에서 사용할 수 있습니다. 상업용 리전에서의 AWS 리전 가용성은 Amazon Web Services 일반 참조의 Redshift API에 대한 서비스 엔드포인트를 참조하세요.

Amazon Redshift Spectrum 제한 사항

이 주제에서는 Redshift Spectrum 사용 시 적용되는 제한 사항을 설명합니다.

Redshift Spectrum을 사용할 때 다음을 고려하세요.

- Amazon Redshift 클러스터와 Amazon S3 버킷이 동일한 AWS 리전에 있어야 합니다.
- Redshift Spectrum은 대한 향상된 VPC 라우팅을 지원하지 않습니다. Amazon S3 데이터에 액세스하려면 추가 구성 단계를 수행해야 할 수 있습니다. 자세한 내용은 Amazon Redshift 관리 안내서의 Redshift Spectrum 및 향상된 VPC 라우팅을 참조하세요.
- Redshift 스펙트럼은 Amazon S3 액세스 포인트 별칭을 지원합니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 액세스 포인트에 버킷 스타일 별칭 사용을 참조하세요. 그러나 Redshift Spectrum은 Amazon S3 액세스 포인트 별칭이 있는 VPC를 지원하지 않습니다. 자세한 내용은 Amazon Redshift 관리 안내서의 Redshift Spectrum 및 향상된 VPC 라우팅을 참조하세요.
- 외부 테이블에서는 업데이트 또는 삭제 작업을 수행할 수 없습니다. 지정된 스키마에 새 외부 테이블을 생성하기 위해 CREATE EXTERNAL TABLE을 사용할 수 있습니다. CREATE EXTERNAL TABLE에 대한 자세한 내용은 CREATE EXTERNAL TABLE 섹션을 참조하세요. 외부 카탈로그의 기존 외부 테이블에 SELECT 쿼리의 결과를 삽입하기 위해 INSERT(외부 테이블)를 사용할 수 있습니다. INSERT(외부 테이블)에 대한 자세한 내용은 INSERT(외부 테이블) 섹션을 참조하세요.
- AWS Lake Formation에 대해 사용 설정된 AWS Glue Data Catalog를 사용하지 않는 한 외부 테이블에 대한 사용자 권한을 제어할 수 없습니다. 그 대신 외부 테이블에서 권한을 허용하고 취소할 수는 있습니다. AWS Lake Formation 작업에 대한 자세한 내용은 Redshift Spectrum 및 AWS Lake Formation 섹션을 참조하세요.
- Redshift Spectrum 쿼리를 실행하려면 데이터베이스 사용자가 데이터베이스에서 임시 테이블을 생성할 수 있는 권한을 보유해야 합니다. 다음 예에서는 `spectrumdb` 데이터베이스에 대한 임시 권한을 `spectrumusers` 사용자 그룹에 부여합니다.

  ```
  grant temp on database spectrumdb to group spectrumusers;
  ```

  자세한 내용은 GRANT 단원을 참조하십시오.

- Athena Data Catalog 또는 AWS Glue Data Catalog를 메타데이터 스토어로 사용하는 경우 Amazon Redshift 관리 가이드의 할당량 및 제한 섹션을 참조하세요.
- Redshift Spectrum은 Kerberos가 포함된 Amazon EMR을 지원하지 않습니다.

Amazon Redshift Spectrum 시작하기

이 튜토리얼에서는 Amazon Redshift Spectrum을 사용하여 Amazon S3의 파일에서 직접 데이터를 쿼리하는 방법에 대해 알아봅니다. 클러스터와 SQL 클라이언트가 이미 있다면 간단한 설정만으로 이 튜토리얼을 마칠 수 있습니다.

> **Note**
>
> Redshift Spectrum 쿼리에는 추가 요금이 발생합니다. 이 자습서에서 샘플 쿼리를 실행하는 요금은 정상 요금입니다. 요금에 대한 자세한 내용은 Redshift Spectrum 요금을 참조하세요.

사전 조건

Redshift Spectrum을 사용하려면 Amazon Redshift 클러스터와 SQL 명령을 실행할 수 있도록 클러스터에 연결된 SQL 클라이언트가 있어야 합니다. 클러스터와 Amazon S3의 데이터 파일은 같은 AWS 리전에 있어야 합니다.

Amazon Redshift 클러스터를 만드는 방법에 대한 자세한 내용은 Amazon Redshift 시작 안내서에서 Amazon Redshift 프로비저닝된 데이터 웨어하우스 시작하기를 참조하세요. 클러스터에 연결하는 방법에 대한 자세한 내용은 Amazon Redshift 시작 안내서의 Amazon Redshift 데이터 웨어하우스에 연결 단원을 참조하세요.

이어지는 예시 중 일부에서 샘플 데이터는 미국 동부(버지니아 북부) 리전(`us-east-1`)에 있으므로 `us-east-1`에 있는 클러스터가 필요합니다. 또는 Amazon S3를 사용하여 다음 버킷 및 폴더의 데이터 객체를 클러스터가 위치한 AWS 리전의 버킷에 복사할 수 있습니다.

- `s3://redshift-downloads/tickit/spectrum/customers/*`
- `s3://redshift-downloads/tickit/spectrum/sales_partition/*`
- `s3://redshift-downloads/tickit/spectrum/sales/*`
- `s3://redshift-downloads/tickit/spectrum/salesevent/*`

다음과 유사한 Amazon S3 명령을 실행하여 미국 동부 (버지니아 북부)에 있는 샘플 데이터를 AWS 리전에 복사합니다. 명령을 실행하기 전에 Amazon S3 copy 명령과 일치하도록 버킷과 폴더를 버킷에 생성합니다. Amazon S3 copy 명령의 출력은 파일이 원하는 AWS 리전의 *bucket-name*에 복사되었음을 알려줍니다.

```
aws s3 cp s3://redshift-downloads/tickit/spectrum/ s3://bucket-name/tickit/spectrum/ --
copy-props none --recursive
```

AWS CloudFormation을 사용하여 Amazon Redshift Spectrum 시작하기

다음 단계의 대안으로 Redshift Spectrum DataLake AWS CloudFormation 템플릿에 액세스하여 쿼리할 수 있는 Amazon S3 버킷으로 스택을 생성할 수 있습니다. 자세한 내용은 AWS CloudFormation 스택을 시작한 다음 Amazon S3에서 데이터를 쿼리합니다. 단원을 참조하십시오.

단계별로 Amazon Redshift Spectrum 시작하기

Amazon Redshift Spectrum 사용을 시작하려면 다음 단계를 따르세요.

- 1단계. Amazon Redshift에 대한 IAM 역할 생성
- 2단계: IAM 역할을 클러스터와 연결
- 3단계: 외부 스키마와 외부 테이블 생성
- 4단계: Amazon S3에서 데이터 쿼리

1단계. Amazon Redshift에 대한 IAM 역할 생성

클러스터가 AWS Glue 또는 Amazon Athena에 있는 외부 Data Catalog와 Amazon S3에 있는 데이터 파일에 액세스하려면 권한 부여가 필요합니다. 이러한 권한을 부여하려면 클러스터에 연결되어 있는 AWS Identity and Access Management(IAM) 역할을 참조합니다. Amazon Redshift에서의 역할 사용에 대한 자세한 내용은 IAM 역할을 사용하여 COPY 및 UNLOAD 작업 권한 부여 섹션을 참조하세요.

> **Note**
>
> 경우에 따라 Athena Data Catalog를 AWS Glue Data Catalog로 마이그레이션할 수 있습니다. 클러스터가 AWS Glue가 지원되는 AWS 리전에 있고 Athena Data Catalog에 Redshift Spectrum 외부 테이블이 있는 경우 이 작업을 수행할 수 있습니다. AWS Glue 데이터 카탈로그를 Redshift Spectrum과 함께 사용하려면 IAM 정책을 변경해야 할 수 있습니다. 자세한 내용은 Athena User Guide의 Upgrading to the AWS Glue Data Catalog 섹션을 참조하세요.

Amazon Redshift에 대한 역할을 생성할 때 다음 접근 방식 중 하나를 선택합니다.

- Athena Data Catalog 또는 AWS Glue Data Catalog에서 Redshift Spectrum을 사용하는 경우 Amazon Redshift에 대한 IAM 역할을 생성하려면에 약술된 단계를 따릅니다.
- AWS Lake Formation에 대해 사용 설정된 AWS Glue Data Catalog와 함께 Redshift Spectrum을 사용하는 경우 다음 절차에 설명된 단계를 따르세요.
  - AWS Lake Formation에 사용되는 AWS Glue Data Catalog로 Amazon Redshift에 대한 IAM 역할을 생성하려면
  - Lake Formation 데이터베이스에서 쿼리할 테이블에 대한 SELECT 권한을 부여하려면

Amazon Redshift에 대한 IAM 역할을 생성하려면

1. IAM 콘솔을 엽니다.
2. 탐색 창에서 역할을 선택합니다.
3. 역할 생성을 선택합니다.
4. AWS service(AWS 서비스)를 신뢰할 수 있는 엔터티로 선택한 다음 사용 사례로 Redshift를 선택합니다.
5. Use case for other AWS 서비스(다른 AWS 서비스 스의 사용 사례)에서 Redshift - Customizable(Redshift - 사용자 지정)을 선택한 후 Next(다음)를 선택합니다.
6. Add permissions policy(권한 정책 추가) 페이지가 나타납니다. AWS Glue 데이터 카탈로그를 사용하는 경우 `AmazonS3ReadOnlyAccess` 및 `AWSGlueConsoleFullAccess`를 선택합니다. 또는 Athena Data Catalog를 사용하는 경우 `AmazonAthenaFullAccess`를 선택합니다. Next(다음)를 선택합니다.

> **Note**
>
> `AmazonS3ReadOnlyAccess` 정책은 모든 Amazon S3 버킷에 대한 읽기 전용 액세스 권한을 클러스터에 부여합니다. AWS 샘플 데이터 버킷에만 액세스 권한을 부여하려면 새 정책을 생성하고 다음 권한을 추가합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
```

```
                "s3:List*"
            ],
            "Resource": "arn:aws:s3:::redshift-downloads/*"
        }
    ]
}
```

7. Role name(역할 이름)에는 역할 이름을 입력합니다(예: **myspectrum_role**).
8. 정보를 검토한 후 역할 만들기를 선택합니다.
9. 탐색 창에서 역할(Roles)을 선택합니다. 새 역할 이름을 선택하여 요약을 본 다음 역할 ARN을 클립보드에 복사합니다. 이 값은 방금 만든 역할의 Amazon 리소스 이름(ARN)입니다. 외부 테이블을 생성해 Amazon S3에 있는 데이터 파일을 참조할 때는 이 값을 사용합니다.

AWS Lake Formation에 사용되는 AWS Glue Data Catalog로 Amazon Redshift에 대한 IAM 역할을 생성하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 Policies를 선택합니다.

   정책을 처음으로 선택하는 경우 관리형 정책 소개 페이지가 나타납니다. 시작하기(Get Started)를 선택합니다.
3. 정책 생성을 선택합니다.
4. JSON 탭에서 정책을 생성하도록 선택합니다.
5. 다음 JSON 정책 문서를 붙여넣습니다. 이는 Lake Formation에 대한 액세스 권한을 부여하지만 Data Catalog에 대한 관리자 권한은 거부합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RedshiftPolicyForLF",
            "Effect": "Allow",
            "Action": [
                "glue:*",
                "lakeformation:GetDataAccess"
            ],
            "Resource": "*"
        }
```

```
    ]
}
```

6. 작업이 완료되면 검토를 선택하여 정책을 검토합니다. 정책 검사기가 모든 구문 오류를 보고합니다.
7. 정책 검토 페이지의 이름에 **myspectrum_policy**를 입력하여 생성 중인 정책의 이름을 지정합니다. 설명을 입력합니다(선택 사항). 정책 요약을 검토하여 정책이 부여한 권한을 확인합니다. 그런 다음 정책 생성을 선택하여 작업을 저장합니다.

   정책을 생성한 후 사용자에게 연결할 수 있습니다.

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요:

- AWS IAM Identity Center의 사용자 및 그룹:

  권한 세트를 생성합니다. AWS IAM Identity Center 사용 설명서의 권한 세트 생성의 지침을 따릅니다.
- 보안 인증 공급자를 통해 IAM에서 관리되는 사용자:

  ID 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 Create a role for a third-party identity provider (federation)의 지침을 따릅니다.
- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 Create a role for an IAM user의 지침을 따릅니다.
  - (권장되지 않음)정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따르세요.

Lake Formation 데이터베이스에서 쿼리할 테이블에 대한 SELECT 권한을 부여하려면

1. Lake Formation 콘솔(https://console.aws.amazon.com/lakeformation/)을 엽니다.
2. 탐색 창에서 데이터 레이크 권한을 선택한 다음 권한 부여를 선택합니다.
3. AWS Lake Formation 개발자 안내서의 명명된 리소스 방법을 사용하여 테이블 권한 부여의 지침을 따르세요. 다음 정보를 제공합니다.
   - [IAM 역할(IAM role)]에서 생성한 IAM 역할인 `myspectrum_role`을 선택합니다. Amazon Redshift 쿼리 편집기를 실행할 때 데이터에 대한 권한으로 이 IAM 역할을 사용합니다.

> **Note**
>
> Lake Formation 사용 Data Catalog에서 쿼리할 테이블에 대해 SELECT 권한을 부여하려면 다음을 수행합니다.
>
> - Lake Formation에서 데이터의 경로를 등록합니다.
> - Lake Formation에서 사용자에게 해당 경로에 대한 권한을 부여합니다.
> - 생성된 테이블은 Lake Formation에 등록된 경로에서 찾을 수 있습니다.

4. 권한 부여를 선택합니다.

> **Important**
>
> Lake Formation 권한을 통해 기본 Amazon S3 객체에 대한 액세스 권한만 허용하는 것이 좋습니다. 승인되지 않은 액세스를 방지하려면 Lake Formation 외부의 Amazon S3 객체에 대해 부여된 권한을 제거합니다. Lake Formation,을 설정하기 전에 Amazon S3 객체에 액세스한 경우 이전에 설정된 버킷 권한 또는 IAM 정책을 제거합니다. 자세한 내용은 Upgrading AWS Glue Data Permissions to the AWS Lake Formation Model 및 Lake Formation Permissions 섹션을 참조하세요.

2단계: IAM 역할을 클러스터와 연결

이제 Amazon Redshift가 외부 Data Catalog 및 Amazon S3에 액세스할 수 있는 권한을 부여하는 IAM 역할이 있습니다. 이때 해당 역할을 Amazon Redshift 클러스터와 연결해야 합니다.

IAM 역할을 클러스터와 연결하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 클러스터(Clusters)를 선택한 후 업데이트할 클러스터의 이름을 선택합니다.
3. 작업(Actions)에서 IAM 역할 관리(Manage IAM roles)를 선택합니다. IAM 역할 페이지가 나타납니다.
4. Enter ARN(ARN 입력)을 선택한 다음 ARN 또는 IAM 역할을 입력하거나 목록에서 IAM 역할을 선택합니다. 그런 다음 Add IAM role(IAM 역할 추가)을 선택하여 Attached IAM roles(연결된 IAM 역할) 목록에 추가합니다.

5. 완료를 선택하여 IAM 역할을 클러스터에 연결합니다. 변경이 완료되도록 클러스터가 수정되었습니다.

3단계: 외부 스키마와 외부 테이블 생성

외부 스키마에서 외부 테이블을 생성합니다. 외부 스키마는 외부 데이터 카탈로그에 있는 데이터베이스를 참조하며, 사용자를 대신하여 Amazon S3에 액세스하도록 클러스터에 권한을 부여하는 IAM 역할 ARN을 제공합니다. 외부 데이터베이스는 Amazon Athena Data Catalog, AWS Glue Data Catalog 또는 Amazon EMR과 같은 Apache Hive 메타스토어에서 생성할 수 있습니다. 이 예제에서는 외부 스키마 Amazon Redshift를 생성할 때 Amazon Athena Data Catalog에 외부 데이터베이스를 생성합니다. 자세한 내용은 Amazon Redshift Spectrum의 외부 스키마 단원을 참조하십시오.

외부 스키마와 외부 테이블을 생성하려면

1. 외부 스키마를 만들려면 다음 명령에서 IAM 역할 ARN을 1단계에서 만든 역할 ARN으로 대체합니다. 그런 다음 SQL 클라이언트에서 명령을 실행합니다.

```
create external schema myspectrum_schema
from data catalog
database 'myspectrum_db'
iam_role 'arn:aws:iam::123456789012:role/myspectrum_role'
create external database if not exists;
```

2. 외부 테이블을 만들려면 다음 CREATE EXTERNAL TABLE 명령을 실행합니다.

> **Note**
>
> 클러스터와 Amazon S3 버킷이 동일한 AWS 리전에 있어야 합니다. 예로 제시된 CREATE EXTERNAL TABLE 명령의 경우 샘플 데이터가 들어 있는 Amazon S3 버킷이 미국 동부(버지니아 북부) AWS 리전에 있습니다. 소스 데이터를 보려면 sales_ts.000 파일을 다운로드합니다.
> 이 예를 다른 AWS 리전에서 실행하도록 수정할 수 있습니다. 원하는 AWS 리전에 Amazon S3 버킷을 생성합니다. Amazon S3 copy 명령을 사용하여 판매 데이터를 복사합니다. 그런 다음, 예로 제시된 CREATE EXTERNAL TABLE 명령에서 버킷의 위치를 업데이트합니다.
>
> ```
> aws s3 cp s3://redshift-downloads/tickit/spectrum/sales/ s3://bucket-name/
> tickit/spectrum/sales/ --copy-props none --recursive
> ```

Amazon S3 copy 명령의 출력은 파일이 원하는 AWS 리전의 *bucket-name*에 복사되었음을 알려줍니다.

```
copy: s3://redshift-downloads/tickit/spectrum/sales/sales_ts.000 to
 s3://bucket-name/tickit/spectrum/sales/sales_ts.000
```

```
create external table myspectrum_schema.sales(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
row format delimited
fields terminated by '\t'
stored as textfile
location 's3://redshift-downloads/tickit/spectrum/sales/'
table properties ('numRows'='172000');
```

4단계: Amazon S3에서 데이터 쿼리

외부 테이블이 생성된 후에는 다른 Amazon Redshift 테이블을 쿼리할 때와 같은 SELECT 문을 사용하여 외부 테이블을 쿼리할 수 있습니다. 이 SELECT 문 쿼리에는 테이블 조인, 데이터 집계, 조건자 필터링이 포함됩니다.

Amazon S3에서 데이터를 쿼리하려면

1. MYSPECTRUM_SCHEMA.SALES 테이블의 행 수를 가져옵니다.

```
select count(*) from myspectrum_schema.sales;
```

```
count
------
```

```
172462
```

2. 큰 팩트 테이블은 Amazon S3에 두고 작은 차원 테이블은 Amazon Redshift에 두는 것이 모범 사례입니다. 데이터 로드에서 샘플 데이터를 로드한 경우 데이터베이스에 EVENT라는 테이블이 있습니다. 그렇지 않다면 다음 명령을 사용하여 EVENT 테이블을 생성합니다.

```
create table event(
eventid integer not null distkey,
venueid smallint not null,
catid smallint not null,
dateid smallint not null sortkey,
eventname varchar(200),
starttime timestamp);
```

3. 다음 COPY 명령에서 IAM 역할 ARN을 1단계. Amazon Redshift에 대한 IAM 역할 생성에서 만든 역할 ARN으로 대체하여 EVENT 테이블을 로드합니다. 원한다면 AWS 리전 us-east-1의 Amazon S3 버킷에서 allevents_pipe.txt의 소스 데이터를 다운로드하여 볼 수 있습니다.

```
copy event from 's3://redshift-downloads/tickit/allevents_pipe.txt'
iam_role 'arn:aws:iam::123456789012:role/myspectrum_role'
delimiter '|' timeformat 'YYYY-MM-DD HH:MI:SS' region 'us-east-1';
```

다음 예는 상위 10개 이벤트의 총 매출을 찾기 위해 외부 Amazon S3 테이블 MYSPECTRUM_SCHEMA.SALES와 로컬 Amazon Redshift 테이블 EVENT를 조인합니다.

```
select top 10 myspectrum_schema.sales.eventid,
 sum(myspectrum_schema.sales.pricepaid) from myspectrum_schema.sales, event
where myspectrum_schema.sales.eventid = event.eventid
and myspectrum_schema.sales.pricepaid > 30
group by myspectrum_schema.sales.eventid
order by 2 desc;
```

```
eventid | sum
--------+---------
    289 | 51846.00
   7895 | 51049.00
   1602 | 50301.00
    851 | 49956.00
   7315 | 49823.00
   6471 | 47997.00
   2118 | 47863.00
```

```
   984 | 46780.00
  7851 | 46661.00
  5638 | 46280.00
```

4. 이전 쿼리의 쿼리 계획을 확인합니다. Amazon S3의 데이터에 대해 실행된 S3 Seq Scan, S3 HashAggregate 및 S3 Query Scan 단계를 메모합니다.

```
explain
select top 10 myspectrum_schema.sales.eventid,
 sum(myspectrum_schema.sales.pricepaid)
from myspectrum_schema.sales, event
where myspectrum_schema.sales.eventid = event.eventid
and myspectrum_schema.sales.pricepaid > 30
group by myspectrum_schema.sales.eventid
order by 2 desc;
```

```
QUERY PLAN


------------------------------------------------------------------------------
XN Limit  (cost=1001055770628.63..1001055770628.65 rows=10 width=31)


  ->  XN Merge  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)


        Merge Key: sum(sales.derived_col2)


        ->  XN Network  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)


              Send to leader


              ->  XN Sort  (cost=1001055770628.63..1001055770629.13 rows=200
 width=31)


                    Sort Key: sum(sales.derived_col2)
```

```
                    ->  XN HashAggregate  (cost=1055770620.49..1055770620.99
rows=200 width=31)

                          ->  XN Hash Join DS_BCAST_INNER
(cost=3119.97..1055769620.49 rows=200000 width=31)

                                Hash Cond: ("outer".derived_col1 = "inner".eventid)


                                ->  XN S3 Query Scan sales  (cost=3010.00..5010.50
rows=200000 width=31)

                                      ->  S3 HashAggregate  (cost=3010.00..3010.50
rows=200000 width=16)

                                            ->  S3 Seq Scan myspectrum_schema.sales
location:"s3://redshift-downloads/tickit/spectrum/sales" format:TEXT
(cost=0.00..2150.00 rows=172000 width=16)
                                                  Filter: (pricepaid > 30.00)


                                ->  XN Hash  (cost=87.98..87.98 rows=8798 width=4)


                                      ->  XN Seq Scan on event  (cost=0.00..87.98
rows=8798 width=4)
```

AWS CloudFormation 스택을 시작한 다음 Amazon S3에서 데이터를 쿼리합니다.

Amazon Redshift 클러스터를 생성하고 클러스터에 연결한 후 Redshift Spectrum DataLake AWS CloudFormation 템플릿을 설치한 다음 데이터를 쿼리할 수 있습니다.

CloudFormation은 Redshift Spectrum 시작하기 DataLake 템플릿을 설치하고 다음을 포함하는 스택을 생성합니다.

- Redshift 클러스터와 연결된 `myspectrum_role`이라는 역할
- `myspectrum_schema`라는 외부 스키마
- Amazon S3 버킷의 `sales`라는 외부 테이블
- 데이터가 로드된 `event`라는 Redshift 테이블

Redshift Spectrum 시작하기 DataLake CloudFormation 스택을 시작하려면

1. CFN 스택 시작(Launch CFN stack)을 선택합니다. DataLake.yml 템플릿이 선택된 CloudFormation 콘솔이 열립니다.

   또한 Redshift Spectrum 시작하기 DataLake CloudFormation CFN 템플릿을 다운로드하고 사용자 지정한 다음 CloudFormation 콘솔(https://console.aws.amazon.com/cloudformation)을 열고 사용자 지정한 템플릿으로 스택을 생성할 수 있습니다.
2. 다음(Next)을 선택합니다.
3. 파라미터(Parameters)에서 Amazon Redshift 클러스터 이름, 데이터베이스 이름 및 데이터베이스 사용자 이름을 입력합니다.
4. Next(다음)를 선택합니다.

   스택 옵션이 나타납니다.
5. 다음(Next)을 선택하여 기본 설정을 적용합니다.
6. 정보를 검토하고 기능(Capabilities)에서 AWS CloudFormation에서 IAM 리소스를 생성할 수 있음 확인(I acknowledge that might create IAM resources)을 선택합니다.
7. 스택 생성(Create stack)을 선택합니다.

스택이 생성되는 동안 오류가 발생하면 다음 정보를 확인합니다.

- 오류 해결에 도움이 되는 정보는 CloudFormation 이벤트(Events) 탭을 참조합니다.
- 작업을 다시 시도하기 전에 DataLake CloudFormation 스택을 삭제합니다.
- Amazon Redshift 데이터베이스에 연결되어 있는지 확인합니다.
- Amazon Redshift 클러스터 이름, 데이터베이스 이름 및 데이터베이스 사용자 이름을 제대로 입력했는지 확인합니다.

Amazon S3에서 데이터 쿼리

다른 Amazon Redshift 테이블을 쿼리할 때와 같은 SELECT 문을 사용하여 외부 테이블을 쿼리합니다. 이 SELECT 문 쿼리에는 테이블 조인, 데이터 집계, 조건자 필터링이 포함됩니다.

다음 쿼리는 `myspectrum_schema.sales` 외부 테이블의 행 수를 반환합니다.

```
select count(*) from myspectrum_schema.sales;
```

```
count
------
172462
```

로컬 테이블과 외부 테이블 조인

다음 예는 상위 10개 이벤트의 총 매출을 찾기 위해 외부 테이블 myspectrum_schema.sales와 로컬 테이블 event를 조인합니다.

```
select top 10 myspectrum_schema.sales.eventid, sum(myspectrum_schema.sales.pricepaid)
 from myspectrum_schema.sales, event
where myspectrum_schema.sales.eventid = event.eventid
and myspectrum_schema.sales.pricepaid > 30
group by myspectrum_schema.sales.eventid
order by 2 desc;
```

```
eventid | sum
--------+---------
    289 | 51846.00
   7895 | 51049.00
   1602 | 50301.00
    851 | 49956.00
   7315 | 49823.00
   6471 | 47997.00
   2118 | 47863.00
    984 | 46780.00
   7851 | 46661.00
   5638 | 46280.00
```

쿼리 계획 보기

이전 쿼리의 쿼리 계획을 확인합니다. Amazon S3의 데이터에 대해 실행된 S3 Seq Scan, S3 HashAggregate 및 S3 Query Scan 단계를 메모합니다.

```
explain
select top 10 myspectrum_schema.sales.eventid, sum(myspectrum_schema.sales.pricepaid)
from myspectrum_schema.sales, event
where myspectrum_schema.sales.eventid = event.eventid
and myspectrum_schema.sales.pricepaid > 30
group by myspectrum_schema.sales.eventid
```

```
order by 2 desc;
```

```
QUERY PLAN

------------------------------------------------------------------------------
XN Limit  (cost=1001055770628.63..1001055770628.65 rows=10 width=31)

  ->  XN Merge  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

        Merge Key: sum(sales.derived_col2)

        ->  XN Network  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

              Send to leader

              ->  XN Sort  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

                    Sort Key: sum(sales.derived_col2)

                    ->  XN HashAggregate  (cost=1055770620.49..1055770620.99 rows=200
 width=31)

                          ->  XN Hash Join DS_BCAST_INNER  (cost=3119.97..1055769620.49
 rows=200000 width=31)

                                Hash Cond: ("outer".derived_col1 = "inner".eventid)

                                ->  XN S3 Query Scan sales  (cost=3010.00..5010.50
 rows=200000 width=31)

                                      ->  S3 HashAggregate  (cost=3010.00..3010.50
 rows=200000 width=16)
```

```
                                          ->  S3 Seq Scan spectrum.sales
location:"s3://redshift-downloads/tickit/spectrum/sales" format:TEXT
(cost=0.00..2150.00 rows=172000 width=16)
                                                Filter: (pricepaid > 30.00)


                               ->  XN Hash  (cost=87.98..87.98 rows=8798 width=4)


                                     ->  XN Seq Scan on event  (cost=0.00..87.98
rows=8798 width=4)
```

Amazon Redshift Spectrum에 대한 IAM 정책

이 주제에서는 Redshift Spectrum을 사용하는 데 필요한 IAM 권한을 설명합니다.

기본적으로 Amazon Redshift Spectrum은 AWS Glue를 지원하는 AWS 리전의 AWS Glue Data Catalog를 사용합니다. 다른 AWS 리전에서는 Redshift Spectrum이 Athena Data Catalog를 사용합니다. 클러스터가 AWS Glue 또는 Athena에 있는 외부 데이터 카탈로그와 Amazon S3에 있는 데이터 파일에 액세스하려면 권한 부여가 필요합니다. 이러한 권한은 클러스터에 연결되어 있는 IAM(AWS Identity and Access Management) 역할을 참조하면 부여할 수 있습니다. Apache Hive 메타스토어를 사용하여 데이터 카탈로그를 관리하는 경우 Athena에 대한 액세스 권한을 제공할 필요가 없습니다.

역할을 함께 묶어 클러스터가 클러스터에 연결되어 있지 않은 다른 역할을 수임하도록 할 수 있습니다. 자세한 내용은 Amazon Redshift Spectrum에서 IAM 역할 연결 단원을 참조하십시오.

액세스하는 AWS Glue 카탈로그가 보안을 강화하기 위해 암호화될 수 있습니다. AWS Glue 카탈로그가 암호화된 경우 AWS Glue 데이터 카탈로그에 액세스하려면 AWS Glue에 AWS KMS 키가 필요합니다. 자세한 내용은 AWS Glue Developer Guide의 Encrypting Your AWS Glue Data Catalog 섹션을 참조하세요.

주제

- Amazon S3 권한
- 계정 간 Amazon S3 권한
- Redshift Spectrum을 사용하여 액세스 권한을 부여하거나 제한하기 위한 정책
- 최소 권한을 부여하기 위한 정책
- Amazon Redshift Spectrum에서 IAM 역할 연결
- AWS Glue 데이터 카탈로그에 대한 액세스 제어

Amazon S3 권한

클러스터는 적어도 Amazon S3 버킷에 대한 GET 및 LIST 액세스가 필요합니다. 버킷이 클러스터와 동일한 AWS 계정에 속하지 않는 경우에도 마찬가지로 클러스터에게 데이터에 액세스할 수 있는 권한을 부여해야 합니다. 자세한 내용은 Amazon Redshift가 사용자를 대신하여 다른 AWS 서비스에 액세스할 수 있도록 권한 부여 섹션을 참조하세요.

> **Note**
>
> Amazon S3 버킷은 특정 VPC 엔드포인트로부터의 액세스만 제한하는 버킷 정책을 사용할 수 없습니다.

다음 정책은 모든 Amazon S3 버킷에 대한 GET 및 LIST 액세스 권한을 부여합니다. 이 정책은 COPY 작업뿐 아니라 Redshift Spectrum의 Amazon S3 버킷에 대한 액세스를 허용합니다.

```
{
 "Version": "2012-10-17",
 "Statement": [{
  "Effect": "Allow",
  "Action": ["s3:Get*", "s3:List*"],
  "Resource": "*"
 }]
}
```

다음 정책은 `amzn-s3-demo-bucket`이라는 Amazon S3 버킷에 대한 GET 및 LIST 액세스 권한을 부여합니다.

```
{
 "Version": "2012-10-17",
 "Statement": [{
  "Effect": "Allow",
  "Action": ["s3:Get*", "s3:List*"],
  "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
 }]
}
```

계정 간 Amazon S3 권한

Redshift Spectrum에 다른 AWS 계정에 속한 Amazon S3 버킷의 데이터에 액세스할 수 있는 권한을 부여하려면 Amazon S3 버킷에 다음 정책을 추가합니다. 자세한 내용은 교차 계정 버킷 권한 부여를 참조하십시오.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Example permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::redshift-account:role/spectrumrole"
            },
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListMultipartUploadParts",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads"
            ],
            "Resource": [
                "arn:aws:s3:::bucketname",
                "arn:aws:s3:::bucketname/*"
            ]
        }
    ]
}
```

Redshift Spectrum을 사용하여 액세스 권한을 부여하거나 제한하기 위한 정책

Redshift Spectrum을 사용하는 경우에 한해 Amazon S3 버킷에 대한 액세스 권한을 부여하려면 사용자 에이전트 `AWS Redshift/Spectrum`에 대한 액세스를 허용하는 조건을 추가합니다. 다음 정책은 Redshift Spectrum에 한해 Amazon S3 버킷에 대한 액세스를 허용합니다. COPY 작업 같은 다른 액세스는 제외됩니다.

```
{
    "Version": "2012-10-17",
    "Statement": [{
```

```
        "Effect": "Allow",
        "Action": ["s3:Get*", "s3:List*"],
        "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
                "Condition": {"StringEquals": {"aws:UserAgent": "AWS Redshift/
Spectrum"}}
    }]
}
```

마찬가지로 COPY 작업에 대한 액세스는 허용하지만 Redshift Spectrum 액세스는 제외하는 IAM 역할을 생성할 수 있습니다. 이를 위해서는 사용자 에이전트 **`AWS Redshift/Spectrum`**에 대한 액세스를 거부하는 조건을 추가하면 됩니다. 다음 정책은 Redshift Spectrum을 제외하고 Amazon S3 버킷에 대한 액세스를 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": ["s3:Get*", "s3:List*"],
        "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
                "Condition": {"StringNotEquals": {"aws:UserAgent": "AWS Redshift/
Spectrum"}}
    }]
}
```

최소 권한을 부여하기 위한 정책

다음 정책은 Amazon S3, AWS Glue 및 Athena에서 Redshift Spectrum을 사용하는 데 필요한 최소 권한을 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListMultipartUploadParts",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads"
            ],
            "Resource": [
```

```
                "arn:aws:s3:::bucketname",
                "arn:aws:s3:::bucketname/folder1/folder2/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateDatabase",
                "glue:DeleteDatabase",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:UpdateDatabase",
                "glue:CreateTable",
                "glue:DeleteTable",
                "glue:BatchDeleteTable",
                "glue:UpdateTable",
                "glue:GetTable",
                "glue:GetTables",
                "glue:BatchCreatePartition",
                "glue:CreatePartition",
                "glue:DeletePartition",
                "glue:BatchDeletePartition",
                "glue:UpdatePartition",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

데이터 카탈로그에 AWS Glue 대신 Athena를 사용할 경우 정책에 모든 Athena 액세스 권한이 필요합니다. 다음 정책은 Athena 리소스에 대한 액세스 권한을 부여합니다. 외부 데이터베이스가 Hive 메타스토어에 있는 경우 Athena 액세스 권한이 필요 없습니다.

```
{
 "Version": "2012-10-17",
 "Statement": [{
  "Effect": "Allow",
  "Action": ["athena:*"],
```

```
  "Resource": ["*"]
 }]
}
```

Amazon Redshift Spectrum에서 IAM 역할 연결

역할을 클러스터에 연결하면 클러스터가 Amazon S3, Athena 및 AWS Glue에 액세스할 수 있는 역할을 맡을 수 있습니다. 클러스터에 연결된 역할에 필요한 리소스에 대한 액세스 권한이 없는 경우 다른 계정에 속한 다른 역할을 함께 묶을 수 있습니다. 그러면 클러스터는 함께 묶은 역할을 일시적으로 수임하여 데이터에 액세스합니다. 역할을 함께 묶어 교차 계정 액세스를 부여할 수도 있습니다. 최대 10개의 역할을 함께 묶을 수 있습니다. 체인의 각 역할은 클러스터가 체인의 끝에 있는 역할을 수임할 때까지 체인의 다음 역할을 수임합니다.

역할을 함께 묶으려면 역할 간에 신뢰 관계를 구성해야 합니다. 다른 역할을 수임하는 역할에는 지정된 역할을 수임하도록 허용하는 권한 정책이 있어야 합니다. 결국 권한을 전달하는 역할에는 해당 권한을 다른 역할에 전달하도록 허용하는 신뢰 정책이 있어야 합니다. 자세한 내용은 Amazon Redshift에서 IAM 역할 연결 섹션을 참조하세요.

CREATE EXTERNAL SCHEMA 명령을 실행하여 쉼표로 구분된 역할 ARN 목록을 포함해 역할을 함께 묶을 수 있습니다.

> **Note**
>
> 함께 묶은 역할 목록에 공백을 포함하면 안 됩니다.

다음 예에서 `MyRedshiftRole`은 클러스터에 연결됩니다. `MyRedshiftRole`은 `AcmeData` 역할이 `111122223333` 계정에 속한 것으로 간주합니다.

```
create external schema acme from data catalog
database 'acmedb' region 'us-west-2'
iam_role 'arn:aws:iam::123456789012:role/MyRedshiftRole,arn:aws:iam::111122223333:role/
AcmeData';
```

AWS Glue 데이터 카탈로그에 대한 액세스 제어

데이터 카탈로그에 AWS Glue를 사용하는 경우 IAM 정책을 사용해 AWS Glue 데이터 카탈로그에 세분화된 액세스 제어를 적용할 수 있습니다. 예를 들어 특정 IAM 역할에 데이터베이스 및 테이블 몇 개만 보이도록 하려고 할 수 있습니다.

다음 단원에서는 AWS Glue 데이터 카탈로그에 저장된 데이터에 대한 다양한 액세스 수준에 관한 IAM 정책을 설명합니다.

주제

데이터베이스 작업에 대한 정책

사용자에게 데이터베이스 보기 및 생성 권한을 부여하려는 경우 사용자에게는 데이터베이스 및 AWS Glue 데이터 카탈로그 둘 다에 대한 액세스 권한이 필요합니다.

다음 예 쿼리는 데이터베이스를 생성합니다.

```
CREATE EXTERNAL SCHEMA example_db
FROM DATA CATALOG DATABASE 'example_db' region 'us-west-2'
IAM_ROLE 'arn:aws:iam::redshift-account:role/spectrumrole'
CREATE EXTERNAL DATABASE IF NOT EXISTS
```

다음 IAM 정책은 데이터베이스 생성에 필요한 최소 권한을 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetDatabase",
                "glue:CreateDatabase"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:catalog"
            ]
        }
    ]
```

```
}
```

다음 예 쿼리는 현재 데이터베이스를 나열합니다.

```
SELECT * FROM SVV_EXTERNAL_DATABASES WHERE
databasename = 'example_db1' or databasename = 'example_db2';
```

다음 IAM 정책은 현재 데이터베이스를 나열하는 데 필요한 최소 권한을 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetDatabases"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:database/example_db1",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db2",
                "arn:aws:glue:us-west-2:redshift-account:catalog"

            ]
        }
    ]
}
```

테이블 작업에 대한 정책

사용자에게 테이블에 대한 보기, 생성, 삭제, 변경 또는 기타 작업을 수행할 수 있는 권한을 부여하려면 몇 가지 유형의 액세스가 필요합니다. 테이블 자체, 테이블이 속한 데이터베이스 및 카탈로그에 대한 액세스가 필요합니다.

다음 예 쿼리는 외부 테이블을 생성합니다.

```
CREATE EXTERNAL TABLE example_db.example_tbl0(
    col0 INT,
    col1 VARCHAR(255)
) PARTITIONED BY (part INT) STORED AS TEXTFILE
LOCATION 's3://test/s3/location/';
```

다음 IAM 정책은 외부 테이블 생성에 필요한 최소 권한을 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateTable"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
}
```

다음 각 예 쿼리는 현재 외부 테이블을 나열합니다.

```
SELECT * FROM svv_external_tables
WHERE tablename = 'example_tbl0' OR
tablename = 'example_tbl1';
```

```
SELECT * FROM svv_external_columns
WHERE tablename = 'example_tbl0' OR
tablename = 'example_tbl1';
```

```
SELECT parameters FROM svv_external_tables
WHERE tablename = 'example_tbl0' OR
tablename = 'example_tbl1';
```

다음 IAM 정책은 현재 외부 테이블을 나열하는 데 필요한 최소 권한을 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetTables"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/
example_tbl0",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl1"

            ]
        }
    ]
}
```

다음 예 쿼리는 기존 테이블을 변경합니다.

```
ALTER TABLE example_db.example_tbl0
SET TABLE PROPERTIES ('numRows' = '100');
```

다음 IAM 정책은 기존 테이블 변경에 필요한 최소 권한을 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetTable",
                "glue:UpdateTable"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"

            ]
        }
    ]
}
```

다음 예 쿼리는 기존 테이블을 삭제합니다.

```
DROP TABLE example_db.example_tbl0;
```

다음 IAM 정책은 기존 테이블 삭제에 필요한 최소 권한을 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:DeleteTable"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
```

```
}
```

파티션 작업에 대한 정책

사용자에게 파티션 수준 작업(보기, 생성, 삭제, 변경 등)을 수행할 수 있는 권한을 부여하려면 파티션이 속한 테이블과 데이터베이스 및 데이터 카탈로그에 대한 사용 권한이 필요합니다. 또한 관련 데이터베이스 및 AWS Glue 데이터 카탈로그에 대한 권한도 필요합니다.

다음 예 쿼리는 파티션을 생성합니다.

```
ALTER TABLE example_db.example_tbl0
ADD PARTITION (part=0) LOCATION 's3://test/s3/location/part=0/';
ALTER TABLE example_db.example_t
ADD PARTITION (part=1) LOCATION 's3://test/s3/location/part=1/';
```

다음 IAM 정책은 파티션 생성에 필요한 최소 권한을 부여합니다.

```
 {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetTable",
                "glue:BatchCreatePartition"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
}
```

다음 예 쿼리는 현재 파티션을 나열합니다.

```
SELECT * FROM svv_external_partitions
WHERE schemname = 'example_db' AND
tablename = 'example_tbl0'
```

다음 IAM 정책은 현재 파티션을 나열하는 데 필요한 최소 권한을 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetPartitions",
                "glue:GetTables",
                "glue:GetTable"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
}
```

다음 예 쿼리는 기존 파티션을 변경합니다.

```
ALTER TABLE example_db.example_tbl0 PARTITION(part='0')
SET LOCATION 's3://test/s3/new/location/part=0/';
```

다음 IAM 정책은 기존 파티션 변경에 필요한 최소 권한을 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetPartition",
                "glue:UpdatePartition"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
}
```

다음 예 쿼리는 기존 파티션을 삭제합니다.

```
ALTER TABLE example_db.example_tbl0 DROP PARTITION(part='0');
```

다음 IAM 정책은 기존 파티션 삭제에 필요한 최소 권한을 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:DeletePartition"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
```

```
        }
    ]
}
```

Redshift Spectrum 및 AWS Lake Formation

이 주제에서는 Lake Formation과 함께 Redshift Spectrum을 사용하는 방법에 대해 설명합니다. Lake Formation은 분석 데이터를 공유하는 서비스입니다.

AWS Lake Formation을 사용하여 중앙에서 데이터베이스, 테이블 및 열 수준 액세스 정책을 정의하고 Amazon S3에 저장된 데이터에 적용할 수 있습니다. Lake Formation에서 사용할 수 있는 AWS Glue Data Catalog에 데이터를 등록한 후 Redshift Spectrum을 포함한 여러 서비스로 쿼리할 수 있습니다.

Lake Formation은 Data Catalog의 보안 및 거버넌스를 제공합니다. Lake Formation 내에서 데이터베이스, 테이블, 열, 기본 Amazon S3 스토리지 등 Data Catalog 객체에 대한 권한을 부여하고 취소할 수 있습니다.

> **Important**
>
> Lake Formation을 사용할 수 있는 AWS 리전에서는 Lake Formation 사용 Data Catalog와 함께 Redshift Spectrum만 사용할 수 있습니다. 사용할 수 있는 리전 목록은 AWS 일반 참조의 AWS Lake Formation 엔드포인트 및 할당량을 참조하세요.

Lake Formation과 함께 Redshift Spectrum을 사용하여 다음을 수행할 수 있습니다.

- Lake Formation을 데이터 레이크의 모든 데이터에 대한 권한 및 액세스 제어 정책을 부여하고 취소하는 중앙 위치로 사용합니다. Lake Formation은 Data Catalog의 데이터베이스 및 테이블에 대한 액세스를 제어하기 위한 권한 계층을 제공합니다. 자세한 내용은 AWS Lake Formation 개발자 안내서의 Lake Formation 권한 개요를 참조하십시오.
- 외부 테이블을 만들고 데이터 레이크의 데이터에 대한 쿼리를 실행합니다. 계정의 사용자가 쿼리를 실행할 수 있으려면 먼저 데이터 레이크 계정 관리자가 원본 데이터가 포함된 기존 Amazon S3 경로를 Lake Formation에 등록해야 합니다. 또한 관리자는 테이블을 만들고 사용자에게 권한을 부여합니다. 데이터베이스, 테이블 또는 열에 대한 액세스 권한을 부여할 수 있습니다. 관리자는 Lake Formation Formation의 데이터 필터를 사용하여 Amazon S3 저장된 민감한 데이터에 대한 액세스 제어를 세분화하여 부여할 수 있습니다. 자세한 내용은 행 수준 및 셀 수준 보안을 위한 데이터 필터 사용 단원을 참조하십시오.

데이터가 Data Catalog에 등록되면 사용자가 쿼리를 실행하려고 할 때마다 Lake Formation이 해당 특정 보안 주체의 테이블 액세스를 확인합니다. Lake Formation이 Redshift Spectrum에 임시 자격 증명을 전송하고 쿼리가 실행됩니다.

- `GetCredentials` 또는 `GetClusterCredentials`를 사용하여 획득한 IAM 보안 인증 정보를 사용하여 자동 마운트된 AWS Glue Data Catalog에 대해 Redshift Spectrum 쿼리를 실행하고 데이터베이스 사용자(IAMR:username 또는 IAM:username)별로 Lake Formation 권한을 관리합니다.

Lake Formation에 대해 활성화된 데이터 카탈로그와 함께 Redshift Spectrum을 사용하는 경우 다음 중 하나가 있어야 합니다.

- 데이터 카탈로그에 대한 권한이 있는 클러스터와 연결된 IAM 역할입니다.
- 외부 리소스에 대한 액세스를 관리하도록 구성된 페더레이션형 IAM 자격 증명입니다. 자세한 내용은 페더레이션형 ID를 사용하여 로컬 리소스 및 Amazon Redshift 외부 테이블에 대한 Amazon Redshift 액세스 관리를 참조하세요.

> **Important**
>
> Lake Formation에 대해 사용 설정된 Data Catalog와 함께 Redshift Spectrum을 사용하는 경우 IAM 역할을 연결할 수 없습니다.

Redshift Spectrum과 함께 사용하기 위해 AWS Lake Formation을 설정하는 데 필요한 단계에 대한 자세한 내용은 AWS Lake Formation 개발자 안내서의 튜토리얼: Lake Formation의 JDBC 소스에서 데이터 레이크 생성을 참조하세요. 특히 Redshift Spectrum과의 통합에 대한 자세한 내용은 Amazon Redshift Spectrum을 사용한 데이터 레이크 내 데이터 쿼리를 참조하십시오. 이 주제에서 사용되는 데이터 및 AWS 리소스는 튜토리얼에 설명된 이전 단계에 따라 달라집니다.

행 수준 및 셀 수준 보안을 위한 데이터 필터 사용

AWS Lake Formation에서 데이터 필터를 정의하면 데이터 카탈로그에 정의된 데이터에 대한 Redshift Spectrum 쿼리의 행 수준 및 셀 수준 액세스를 제어할 수 있습니다. 이를 설정하려면 다음 작업을 수행합니다.

- Lake Formation에서 다음 정보를 사용하여 데이터 필터를 생성합니다.
  - 쿼리 결과에 포함하거나 제외할 열 목록이 포함된 열 사양.
  - 쿼리 결과에 포함할 행을 지정하는 행 필터 표현식.

데이터 필터를 생성하는 방법에 대한 자세한 내용은 AWS Lake Formation 개발자 안내서의 Lake Formation 내 데이터 필터를 참조하십시오.

- Amazon Redshift 내에, Lake Formation이 활성화된 데이터 카탈로그 내 테이블을 참조하는 외부 테이블을 생성합니다. Redshift Spectrum을 사용하여 Lake Formation 테이블을 쿼리하는 방법에 대한 자세한 내용은 AWS Lake Formation 개발자 안내서의 Amazon Redshift Spectrum을 사용하여 데이터 레이크 내 데이터 쿼리를 참조하십시오.

Amazon Redshift에서 테이블을 정의한 후에는 Lake Formation 테이블을 쿼리하여 데이터 필터에서 허용하는 행과 열에만 액세스할 수 있습니다.

Lake Formation에서 행 수준 및 셀 수준 보안을 설정한 다음 Redshift Spectrum을 사용하여 쿼리하는 방법에 대한 자세한 내용은 Amazon Redshift Spectrum을 AWS Lake Formation에 정의된 행 수준 및 셀 수준 보안 정책과 함께 사용을 참조하십시오.

Amazon Redshift Spectrum의 쿼리용 데이터 파일

이 섹션에서는 Redshift Spectrum이 지원하는 형식으로 Amazon S3의 데이터 파일을 생성하는 방법에 대해 설명합니다.

Amazon Redshift Spectrum에서 쿼리에 사용하는 데이터 파일은 일반적으로 다른 애플리케이션에 사용하는 것과 파일 형식이 동일합니다. 예를 들어 Amazon Athena, Amazon EMR 및 Amazon QuickSight에서 동일한 형식의 파일이 사용됩니다. Amazon S3에서 직접 원본 형식으로 데이터를 쿼리할 수 있습니다. 이렇게 하려면 데이터 파일이 Redshift Spectrum에서 지원하는 형식이어야 하고 클러스터가 액세스할 수 있는 Amazon S3 버킷에 있어야 합니다.

데이터 파일이 있는 Amazon S3 버킷과 Amazon Redshift 클러스터가 동일한 AWS 리전에 있어야 합니다. 지원되는 AWS 리전에 대한 자세한 내용은 Amazon Redshift Spectrum 리전 섹션을 참조하세요.

Redshift Spectrum의 데이터 형식

Redshift Spectrum은 다음과 같은 정형 및 비정형 데이터 형식을 지원합니다.

파일 형식	열 기반	병렬 읽기 지원	분할 단위
PARQUET	예	예	행 그룹
ORC	예	예	Stripe

파일 형식	열 기반	병렬 읽기 지원	분할 단위
RCFile	예	예	행 그룹
TextFile	No	예	열
SequenceFile	No	예	행 또는 블록
RegexSerde	No	예	열
OpenCSV	No	예	열
AVRO	No	예	차단
Ion	No	No	N/A
JSON	No	No	N/A

위 표에서 머리글은 다음을 나타냅니다.

- 열 기반 – 파일 형식이 행 지향 구조가 아닌 열 지향 구조에 데이터를 물리적으로 저장하는지 여부입니다.
- 병렬 읽기 지원 – 파일 형식이 파일 내의 개별 블록 읽기를 지원하는지 여부입니다. 개별 블록을 읽으면 단일 요청으로 전체 파일을 읽을 필요 없이 여러 독립적인 Redshift Spectrum 요청에 걸쳐 파일을 배포하여 처리할 수 있습니다.
- 분할 단위 – 병렬로 읽을 수 있는 파일 형식의 경우 분할 단위는 단일 Redshift Spectrum 요청에서 처리할 수 있는 가장 작은 데이터 청크입니다.

> **Note**
>
> 타임스탬프 값 `yyyy-MM-dd HH:mm:ss.SSSSSS`에서 볼 수 있듯이, 텍스트 파일의 타임스탬프 값은 `2017-05-01 11:30:59.000000` 형식이어야 합니다.

Apache Parquet 같은 열 기반 스토리지 파일 형식을 사용하는 것이 좋습니다. 열 기반 스토리지 파일 형식의 경우 필요한 열만 선택하여 Amazon S3로부터의 데이터 전송을 최소화할 수 있습니다.

Redshift Spectrum의 압축 형식

스토리지 공간을 줄이고 성능을 높이며 비용을 최소화하려면 데이터 파일을 압축하는 것이 좋습니다. Redshift Spectrum은 파일 확장자를 토대로 파일 압축 형식을 인식합니다.

Redshift Spectrum이 지원하는 압축 형식과 확장자는 다음과 같습니다.

압축 알고리즘	파일 확장명	병렬 읽기 지원
Gzip	.gz	No
Bzip2	.bz2	예
Snappy	.snappy	No

여러 레벨에서 압축을 적용할 수 있습니다. 일반적으로 전체 파일을 압축하거나 파일 내의 개별 블록을 압축합니다. 파일 수준에서 열 형식을 압축해도 성능상의 이점이 없습니다.

Redshift Spectrum에서 파일을 병렬로 읽을 수 있으려면 다음을 충족해야 합니다.

- 파일 형식이 병렬 읽기를 지원합니다.
- 파일 레벨 압축(있는 경우)이 병렬 읽기를 지원합니다.

각 분할 단위가 단일 Redshift Spectrum 요청으로 처리되기 때문에 파일 내의 개별 분할 단위가 병렬로 읽을 수 있는 압축 알고리즘을 사용하여 압축되는지 여부는 중요하지 않습니다. 이 예로는 Snappy 압축 Parquet 파일이 있습니다. Parquet 파일 내의 개별 행 그룹은 Snappy를 사용하여 압축되지만 파일의 최상위 구조는 압축되지 않은 상태로 유지됩니다. 이 경우 각 Redshift Spectrum 요청이 Amazon S3에서 개별 행 그룹을 읽고 처리할 수 있기 때문에 파일을 병렬로 읽을 수 있습니다.

Redshift Spectrum의 암호화

Redshift Spectrum은 다음 암호화 옵션을 사용하여 암호화된 데이터 파일을 투명하게 해독합니다.

- Amazon S3가 관리하는 AES-256 암호화 키를 사용하는 서버 측 암호화(SSE-S3).
- AWS Key Management Service(SSE-KMS)에서 관리하는 키로 서버 측 암호화.

Redshift Spectrum은 Amazon S3 클라이언트 측 암호화는 지원하지 않습니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 서버 측 암호화를 사용하여 데이터 보호를 참조하세요.

Amazon Redshift는 대량 병렬 처리(MPP)를 사용하여 대량의 데이터에서 작동하는 복잡한 쿼리를 빠르게 실행할 수 있습니다. Redshift Spectrum은 같은 원칙을 외부 데이터 쿼리에도 확장하여 파일을 스캔하는 데 필요한 경우, 여러 개의 Redshift Spectrum 인스턴스를 사용합니다. 각 테이블별로 별도의 폴더에 파일을 넣습니다.

다음을 수행하여 데이터를 병렬 처리에 적합하게 최적화할 수 있습니다.

- 파일 형식이나 압축이 병렬로 읽기를 지원하지 않으면 큰 파일을 여러 개의 작은 파일로 나눕니다. 64MB~1GB의 파일 크기를 사용하는 것이 좋습니다.
- 모든 파일의 크기를 거의 비슷하게 유지합니다. 다른 파일보다 훨씬 큰 파일이 있는 경우, Redshift Spectrum이 워크로드를 고르게 분산시킬 수 없습니다.

Amazon Redshift Spectrum의 외부 스키마

이 주제에서는 Redshift Spectrum에서 외부 스키마를 만들고 사용하는 방법을 설명합니다. 외부 스키마는 Amazon Redshift 클러스터 외부의 데이터에 액세스하기 위한 참조로 사용하는 테이블 모음입니다. 이 테이블에는 Redshift Spectrum이 읽는 외부 데이터에 대한 메타데이터가 포함되어 있습니다.

모든 외부 테이블은 CREATE EXTERNAL SCHEMA 문을 사용하여 생성하는 외부 스키마에서 만들어야 합니다.

> **Note**
>
> 몇몇 애플리케이션은 데이터베이스라는 용어와 스키마라는 용어를 구분하지 않고 사용합니다. Amazon Redshift에서는 스키마라는 용어를 사용합니다.

Amazon Redshift 외부 스키마는 외부 데이터 카탈로그에 있는 외부 데이터베이스를 참조합니다. 외부 데이터베이스는 Amazon Redshift, Amazon Athena, AWS Glue Data Catalog 또는 Amazon EMR 같은 Apache Hive 메타스토어에서 생성할 수 있습니다. Amazon Redshift에서 생성되는 외부 데이터베이스는 Athena Data Catalog에 상주합니다. Hive 메타스토어에서 데이터베이스를 만들려면 Hive 애플리케이션에서 데이터베이스를 생성해야 합니다.

Amazon Redshift가 사용자를 대신하여 Athena에 있는 Data Catalog와 Amazon S3에 있는 데이터 파일에 액세스하려면 권한 부여가 필요합니다. 이러한 권한을 부여하려면 먼저 IAM(AWS Identity and Access Management) 역할을 생성해야 합니다. 그런 다음 역할을 클러스터에 연결하고 Amazon Redshift `CREATE EXTERNAL SCHEMA` 문에서 역할의 Amazon 리소스 이름(ARN)을 제공해야 합니다. 권한 부여에 대한 자세한 내용은 Amazon Redshift Spectrum에 대한 IAM 정책 섹션을 참조하세요.

> **Note**
>
> 현재 Athena Data Catalog에 Redshift Spectrum 외부 테이블이 있다면 Athena Data Catalog를 AWS Glue Data Catalog로 마이그레이션할 수 있습니다. AWS Glue 데이터 카탈로그를 Redshift Spectrum과 함께 사용하려면 IAM 정책을 변경해야 할 수 있습니다. 자세한 내용은 Amazon Athena User Guide의 Upgrading to the AWS Glue Data Catalog 섹션을 참조하세요.

외부 스키마를 생성하는 동시에 외부 데이터베이스를 생성하려면 `FROM DATA CATALOG`를 지정하고 `CREATE EXTERNAL DATABASE` 문에 `CREATE EXTERNAL SCHEMA` 절을 포함시킵니다.

다음 예에서는 외부 데이터베이스 `spectrum_schema`를 사용하여 `spectrum_db`라는 이름의 외부 스키마를 생성합니다.

```
create external schema spectrum_schema from data catalog
database 'spectrum_db'
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole'
create external database if not exists;
```

Athena를 사용하여 Data Catalog를 관리하는 경우 Athena 데이터베이스 이름 및 Athena Data Catalog가 위치하는 AWS 리전을 지정합니다.

다음 예에서는 Athena Data Catalog에서 기본 `sampledb` 데이터베이스를 사용해 외부 스키마를 생성합니다.

```
create external schema athena_schema from data catalog
database 'sampledb'
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole'
region 'us-east-2';
```

> **Note**
>
> `region` 파라미터는 Amazon S3의 데이터 파일 위치가 아니라 Athena Data Catalog가 위치한 AWS 리전을 참조합니다.

Amazon EMR과 같은 Hive 메타스토어를 사용하여 데이터 카탈로그를 관리하는 경우 보안 그룹이 클러스터 사이의 트래픽을 허용하도록 구성되어야 합니다.

CREATE EXTERNAL SCHEMA 문에서 FROM HIVE METASTORE를 지정하고 메타스토어의 URI와 포트 번호를 추가합니다. 다음 예에서는 hive_db로 명명된 Hive 메타스토어 데이터베이스를 사용하여 외부 스키마를 생성합니다.

```
create external schema hive_schema
from hive metastore
database 'hive_db'
uri '172.10.10.10' port 99
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole'
```

클러스터의 외부 스키마를 보려면 PG_EXTERNAL_SCHEMA 카탈로그 테이블 또는 SVV_EXTERNAL_SCHEMAS 뷰를 쿼리하십시오. 다음 예는 PG_EXTERNAL_SCHEMA와 PG_NAMESPACE를 조인하는 SVV_EXTERNAL_SCHEMAS를 쿼리합니다.

```
select * from svv_external_schemas
```

전체 명령 구문과 예는 CREATE EXTERNAL SCHEMA 섹션을 참조하세요.

Amazon Redshift Spectrum의 외부 카탈로그 작업

Amazon Redshift Spectrum 외부 데이터베이스 및 외부 테이블을 위한 메타데이터는 외부 데이터 카탈로그에 저장됩니다. 기본적으로 Redshift Spectrum 메타데이터는 Athena Data Catalog에 저장됩니다. Athena 콘솔에서 Redshift Spectrum 데이터베이스와 테이블을 보고 관리할 수 있습니다.

Athena 또는 Amazon EMR 등의 Hive 메타스토어를 통해 Hive DDL(데이터 정의 언어) 명령을 사용하여 외부 데이터베이스와 외부 테이블을 생성하고 관리할 수도 있습니다.

> **Note**
>
> Amazon Redshift를 사용하여 Redshift Spectrum에서 외부 데이터베이스와 외부 테이블을 생성하고 관리하는 것이 좋습니다.

Athena와 AWS Glue에서 Redshift Spectrum 데이터베이스 보기

CREATE EXTERNAL SCHEMA 문에 CREATE EXTERNAL DATABASE IF NOT EXISTS 절을 포함하여 외부 데이터베이스를 생성할 수 있습니다. 이 경우 외부 데이터베이스 메타데이터가 데이터 카탈로그에 저장됩니다. 외부 스키마에 의해 정규화되어 생성한 외부 테이블의 메타데이터도 데이터 카탈로그에 저장됩니다.

Athena와 AWS Glue는 지원되는 각각의 AWS 리전마다 데이터 카탈로그를 유지합니다. 테이블 메타데이터를 보려면 Athena 또는 AWS Glue 콘솔에 로그인합니다. Athena에서 Data sources(데이터 소스), 즉 사용자의 AWS Glue를 선택한 후 데이터베이스의 세부 정보를 봅니다. AWS Glue에서 Databases(데이터베이스), 즉 사용자의 외부 데이터베이스를 선택한 후 데이터베이스의 세부 정보를 봅니다.

Athena를 사용하여 외부 테이블을 생성하고 관리하는 경우 CREATE EXTERNAL SCHEMA를 사용하여 데이터베이스를 등록합니다. 예를 들어 다음 명령은 `sampledb`라는 Athena 데이터베이스를 등록합니다.

```
create external schema athena_sample
from data catalog
database 'sampledb'
iam_role 'arn:aws:iam::123456789012:role/mySpectrumRole'
region 'us-east-1';
```

SVV_EXTERNAL_TABLES 시스템 뷰를 쿼리하면 Athena `sampledb` 데이터베이스에 있는 테이블뿐 아니라 Amazon Redshift에서 생성한 테이블도 볼 수 있습니다.

```
select * from svv_external_tables;
```

```
schemaname    | tablename        | location

--------------+------------------
+-----------------------------------------------------
athena_sample | elb_logs         | s3://athena-examples/elb/plaintext
athena_sample | lineitem_1t_csv  | s3://myspectrum/tpch/1000/lineitem_csv

athena_sample | lineitem_1t_part | s3://myspectrum/tpch/1000/lineitem_partition

spectrum      | sales            | s3://redshift-downloads/tickit/spectrum/sales

spectrum      | sales_part       | s3://redshift-downloads/tickit/spectrum/sales_part
```

Apache Hive 메타스토어 데이터베이스 등록

Apache Hive 메타스토어에서 외부 테이블을 생성하는 경우, CREATE EXTERNAL SCHEMA를 사용하여 이 테이블을 Redshift Spectrum에 등록할 수 있습니다.

CREATE EXTERNAL SCHEMA 문에서 FROM HIVE METASTORE 절을 지정하고 Hive 메타스토어 URI와 포트 번호를 제공합니다. IAM 역할에 Amazon S3에 대한 액세스 권한이 포함되어야 하지만 Athena 권한은 필요하지 않습니다. 다음 예는 Hive 메타스토어를 등록합니다.

```
create external schema if not exists hive_schema
from hive metastore
database 'hive_database'
uri 'ip-10-0-111-111.us-west-2.compute.internal' port 9083
iam_role 'arn:aws:iam::123456789012:role/mySpectrumRole';
```

Amazon Redshift 클러스터에 Amazon EMR 클러스터에 대한 액세스 권한 부여

Hive 메타스토어가 Amazon EMR에 있는 경우 Amazon EMR 클러스터에 Amazon Redshift 클러스터에 대한 액세스 권한을 부여해야 합니다. 이렇게 하려면 Amazon EC2 보안 그룹을 생성합니다. 그런 다음 Amazon Redshift 클러스터 보안 그룹과 Amazon EMR 클러스터 보안 그룹에서 EC2 보안 그룹으로의 모든 인바운드 트래픽을 허용합니다. 그런 다음 Amazon Redshift 클러스터와 Amazon EMR 클러스터에 EC2 보안을 추가합니다.

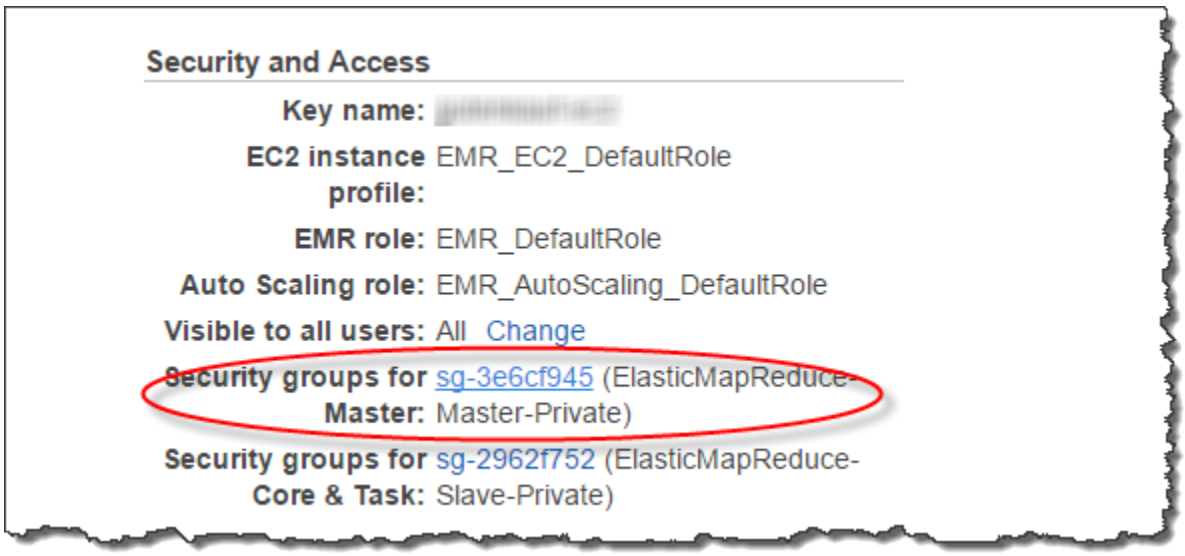
Amazon Redshift 클러스터의 보안 그룹 이름 보기

보안 그룹을 표시하려면 다음을 수행합니다.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 클러스터(Clusters)를 선택한 후 목록에서 클러스터를 선택하여 세부 정보를 엽니다.
3. 속성(Properties)을 선택하고 네트워크 및 보안 설정(Network and security settings) 섹션을 봅니다.
4. VPC 보안 그룹(VPC security group)에서 보안 그룹을 찾아 기록해 둡니다.

Amazon EMR 마스터 노드 보안 그룹 이름 보기

1. Amazon EMR 클러스터를 엽니다. 자세한 내용은 Amazon EMR 관리 가이드의 보안 구성을 사용하여 클러스터 보안 설정을 참조하세요.
2. 보안 및 액세스(Security and access)에서 Amazon EMR 마스터 노드 보안 그룹 이름을 기록해 둡니다.



Amazon Redshift와 Amazon EMR 간의 연결을 허용하도록 Amazon EC2 보안 그룹을 생성하거나 수정하려면

1. Amazon EC2 대시보드에서 보안 그룹(Security groups)을 선택합니다. 자세한 내용은 Amazon EC2 사용 설명서의 보안 그룹 규칙을 참조하세요.
2. 보안 그룹 생성(Create security group)을 선택합니다.
3. VPC를 사용하는 경우 Amazon Redshift 클러스터와 Amazon EMR 클러스터가 있는 VPC를 선택합니다.
4. 인바운드 규칙을 추가합니다.
   1. 유형에 대해 사용자 지정 TCP를 선택합니다.
   2. 소스(Source)에서 사용자 지정(Custom)을 선택합니다.
   3. Amazon Redshift 보안 그룹의 이름을 입력합니다.
5. 다른 인바운드 규칙을 추가합니다.
   1. 유형에서 TCP를 선택합니다.
   2. 포트 범위에 9083을 입력합니다.

> **Note**
>
> EMR HMS의 기본 포트는 9083입니다. HMS가 다른 포트를 사용하는 경우에는 인바운드 규칙과 외부 스키마 정의에 따라 해당 포트를 지정하십시오.

3. 소스(Source)에서 사용자 지정(Custom)을 선택합니다.

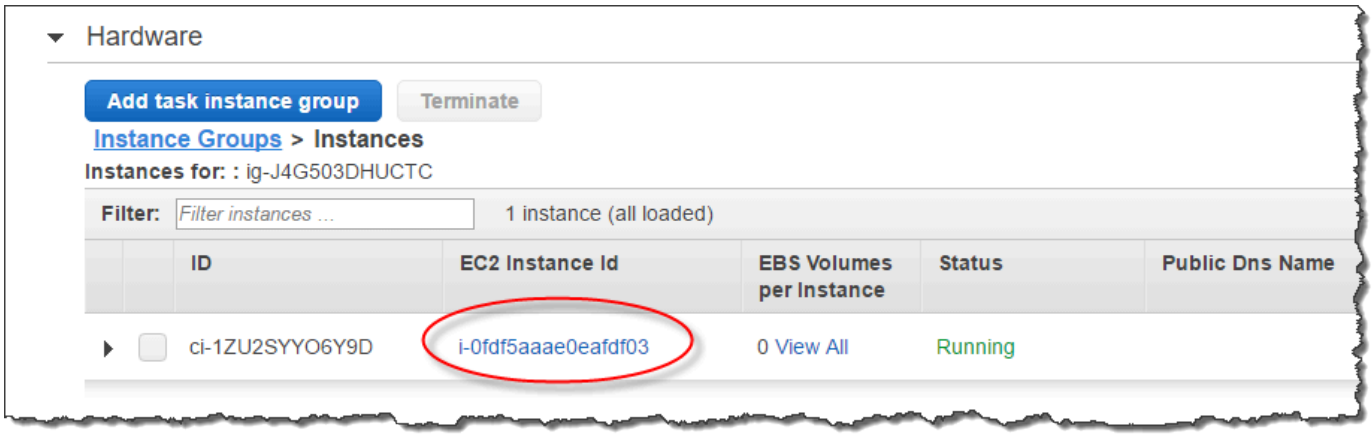
6. 보안 그룹 이름과 설명을 입력합니다.
7. 보안 그룹 생성을 선택합니다.

이전 절차에서 생성한 Amazon EC2 보안 그룹을 Amazon Redshift 클러스터에 추가하려면

1. Amazon Redshift에서 클러스터를 선택합니다.
2. 속성을 선택합니다.
3. 네트워크 및 보안 설정(Network and security settings)을 보고 편집(Edit)을 선택합니다.
4. VPC 보안 그룹(VPC security group)에서 새 보안 그룹 이름을 선택합니다.
5. Save changes(변경 사항 저장)를 선택합니다.

Amazon EMR 클러스터에 Amazon EC2 보안 그룹을 추가하려면

1. Amazon EMR에서 클러스터를 선택합니다. 자세한 내용은 Amazon EMR 관리 가이드의 보안 구성을 사용하여 클러스터 보안 설정을 참조하세요.
2. 하드웨어에서 마스터 노드의 링크를 선택합니다.
3. EC2 인스턴스 ID(EC2 instance ID) 열에서 링크를 선택합니다.



4. 작업(Actions)에서 보안(Security), 보안 그룹 변경(Change security groups)을 선택합니다.
5. 연결된 보안 그룹(Associated sercurity groups)에서 새 보안 그룹을 선택하고 보안 그룹 추가(Add security group)를 선택합니다.
6. Save(저장)를 선택합니다.

Redshift Spectrum용 외부 테이블

이 주제에서는 Redshift Spectrum에서 외부 테이블을 생성하고 사용하는 방법을 설명합니다. 외부 테이블은 Amazon Redshift 클러스터 외부의 데이터에 액세스하는 데 참조로 사용하는 테이블입니다. 이 테이블에는 Redshift Spectrum이 읽는 외부 데이터에 대한 메타데이터가 포함되어 있습니다.

외부 스키마에서 외부 테이블을 생성합니다. 외부 테이블을 생성하려면 외부 스키마의 소유자이거나 수퍼유저여야 합니다. 외부 스키마의 소유권을 이전하려면 ALTER SCHEMA를 사용해 소유자를 변경합니다. 다음 예에서는spectrum_schema 스키마의 소유자를 newowner로 바꿉니다.

```
alter schema spectrum_schema owner to newowner;
```

Redshift Spectrum 쿼리를 실행하는 데 필요한 권한은 다음과 같습니다.

- 스키마에 대한 사용 권한
- 현재 데이터베이스에서 임시 테이블을 생성할 수 있는 권한

다음 예에서는 spectrum_schema 스키마에 대한 사용 권한을 spectrumusers 사용자 그룹에 부여합니다.

```
grant usage on schema spectrum_schema to group spectrumusers;
```

다음 예에서는 spectrumdb 데이터베이스에 대한 임시 권한을 spectrumusers 사용자 그룹에 부여합니다.

```
grant temp on database spectrumdb to group spectrumusers;
```

외부 테이블은 Amazon Redshift, AWS Glue, Amazon Athena 또는 Apache Hive 메타스토어에서 생성할 수 있습니다. 자세한 내용은 AWS Glue Developer Guide의 Getting Started Using AWS Glue, Amazon Athena User Guide의 Getting Started 또는 Amazon EMR Developer Guide의 Apache Hive 섹션을 참조하세요.

외부 테이블이 AWS Glue, Athena 또는 Hive 메타스토어에서 정의된 경우 먼저 외부 데이터베이스를 참조하는 외부 스키마를 생성합니다. 그러면 Amazon Redshift에서 테이블을 생성할 필요 없이 테이블 이름에 스키마 이름을 접두사로 사용하여 SELECT 문에서 외부 테이블을 참조할 수 있습니다. 자세한 내용은 Amazon Redshift Spectrum의 외부 스키마 단원을 참조하십시오.

Amazon Redshift가 AWS Glue Data Catalog의 테이블을 볼 수 있도록 하려면 Amazon Redshift IAM 역할에 glue:GetTable을 추가합니다. 그렇지 않은 경우 다음과 같은 오류가 발생할 수 있습니다.

```
RedshiftIamRoleSession is not authorized to perform: glue:GetTable on resource: *;
```

예를 들어 Athena 외부 카탈로그에서 정의된 lineitem_athena라는 외부 테이블이 있다고 가정해 봅시다. 이 경우, 이름이 athena_schema인 외부 테이블을 정의한 후 다음 SELECT 문을 사용해 테이블을 쿼리할 수 있습니다.

```
select count(*) from athena_schema.lineitem_athena;
```

Amazon Redshift에서 외부 테이블을 정의하려면 CREATE EXTERNAL TABLE 명령을 사용합니다. 외부 테이블 문은 테이블 열, 데이터 파일 형식, Amazon S3에서의 데이터 위치를 정의합니다. Redshift Spectrum은 지정된 폴더에 있는 파일과 하위 폴더를 모두 스캔합니다. Redshift Spectrum은 숨겨진 파일과 마침표, 밑줄 또는 해시 표시(. , _, 또는 #)로 시작하거나 물결표(~)로 끝나는 파일은 무시합니다.

다음 예에서는 spectrum이라는 Amazon Redshift 외부 스키마에서 SALES로 명명된 테이블을 생성합니다. 데이터는 탭으로 구분된 텍스트 파일입니다.

```
create external table spectrum.sales(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
row format delimited
fields terminated by '\t'
stored as textfile
location 's3://redshift-downloads/tickit/spectrum/sales/'
table properties ('numRows'='172000');
```

외부 테이블을 보려면 SVV_EXTERNAL_TABLES 시스템 뷰를 쿼리하십시오.

가상 열

기본적으로 Amazon Redshift에서는 가상 열 $path, $size 및 $spectrum_oid가 있는 외부 테이블을 생성합니다. $path 열을 선택하면 Amazon S3에 있는 데이터 파일 경로를 확인하고 $size 열을 선택하여 쿼리에서 반환한 각 행에 대한 데이터 파일의 크기를 확인할 수 있습니다. $spectrum_oid 열은 Redshift Spectrum으로 상관 쿼리를 수행하는 기능을 제공합니다. 예시는 예: Redshift Spectrum에서 상관 하위 쿼리 수행을 확인하세요. $path, $size 및 $spectrum_oid 열 이름은 큰따옴표로 구분해야 합니다. SELECT * 절은 가상 열을 반환하지 않습니다. 다음 예와 같이 쿼리에 $path, $size 및 $spectrum_oid 열 이름을 명시적으로 포함해야 합니다.

```
select "$path", "$size", "$spectrum_oid"
from spectrum.sales_part where saledate = '2008-12-01';
```

spectrum_enable_pseudo_columns 구성 파라미터를 false로 설정하여 세션에 대해 가상 열 생성을 비활성화할 수 있습니다. 자세한 내용은 spectrum_enable_pseudo_columns 단원을 참조하십시오. enable_spectrum_oid를 false로 설정하여 $spectrum_oid 가상 열만 비활성화할 수도 있습니다. 자세한 내용은 enable_spectrum_oid 단원을 참조하십시오. 그러나 $spectrum_oid 가상 열을 비활성화하면 Redshift Spectrum과의 상관 쿼리에 대한 지원도 비활성화됩니다.

> **Important**
>
> $size, $path 또는 $spectrum_oid를 선택하면 Redshift Spectrum이 Amazon S3의 데이터 파일을 스캔해 결과 집합의 크기를 결정하기 때문에 요금이 발생합니다. 자세한 내용은 Amazon Redshift 요금 섹션을 참조하세요.

가상 열 예제

다음은 외부 테이블의 관련 데이터 파일의 총 크기를 반환하는 예입니다.

```
select distinct "$path", "$size"
from spectrum.sales_part;

 $path                                                                      | $size
----------------------------------------------------------------------------+-------
s3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-01/ |  1616
s3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-02/ |  1444
s3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-03/ |  1644
```

Redshift Spectrum 외부 테이블 파티셔닝

데이터를 파티셔닝할 때 파티션 키를 기준으로 필터링하여 Redshift가 스캔하는 데이터 양을 제한할 수 있습니다. 어떤 키를 기준으로도 데이터를 분할할 수 있습니다.

일반적인 관행은 시간을 기준으로 데이터를 파티셔닝하는 것입니다. 예를 들어 연, 월, 일, 시를 기준으로 파티셔닝할 수 있습니다. 데이터가 여러 원본에서 오는 경우, 데이터 원본 식별자와 날짜를 기준으로 파티셔닝할 수 있습니다.

다음 절차에서는 데이터를 파티셔닝하는 방법에 대해 설명합니다.

데이터를 파티셔닝하려면

1. 파티션 키에 따라 Amazon S3의 폴더에 데이터를 저장합니다.

   파티션 값마다 폴더를 하나씩 만들고 파티션 키와 값을 사용하여 폴더 이름을 지정합니다. 예를 들어 날짜를 기준으로 파티셔닝하는 경우, 폴더의 이름은 `saledate=2017-04-01`, `saledate=2017-04-02` 등이 될 수 있습니다. Redshift Spectrum은 파티션 폴더에 있는 파일과 하위 폴더를 모두 스캔합니다. Redshift Spectrum은 숨겨진 파일과 마침표, 밑줄 또는 해시 표시(. , _, 또는 #)로 시작하거나 물결표(~)로 끝나는 파일은 무시합니다.
2. 외부 테이블을 만들고 PARTITIONED BY 절에서 파티션 키를 지정합니다.

   파티션 키가 테이블 열의 이름이어서는 안 됩니다. 데이터 형식은 SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE 또는 TIMESTAMP 데이터 형식일 수 있습니다.
3. 파티션을 추가합니다.

   ALTER TABLE … ADD PARTITION을 사용하여 각 파티션을 추가하고 파티션 열 및 키 값과 Amazon S3에서의 파티션 폴더 위치를 지정합니다. 단일 ALTER TABLE … ADD 문을 사용하여 여러 파티션을 추가할 수 있습니다. 다음 예는 `'2008-01'` 및 `'2008-03'`에 대한 파티션을 추가합니다.

   ```
   alter table spectrum.sales_part add
   partition(saledate='2008-01-01')
   location 's3://redshift-downloads/tickit/spectrum/sales_partition/
   saledate=2008-01/'
   partition(saledate='2008-03-01')
   location 's3://redshift-downloads/tickit/spectrum/sales_partition/
   saledate=2008-03/';
   ```

> **Note**
>
> AWS Glue 카탈로그를 사용하는 경우 단일 ALTER TABLE 문을 사용하여 파티션을 최대 100개까지 추가할 수 있습니다.

데이터 파티셔닝 예제

이 예에서는 하나의 파티션 키로 분할된 외부 테이블 하나와 두 개의 파티션 키로 분할된 외부 테이블 하나를 생성합니다.

이 예의 샘플 데이터는 인증된 모든 AWS 사용자에게 읽기 권한을 부여하는 Amazon S3 버킷에 위치합니다. 클러스터와 외부 데이터 파일은 같은 AWS 리전에 있어야 합니다. 샘플 데이터 버킷은 미국 동부(버지니아 북부) 리전(us-east-1)에 있습니다. Redshift Spectrum을 사용하여 데이터에 액세스하려면 클러스터도 us-east-1에 있어야 합니다. Amazon S3에서 폴더를 나열하려면 다음 명령을 실행합니다.

```
aws s3 ls s3://redshift-downloads/tickit/spectrum/sales_partition/
```

```
PRE saledate=2008-01/
PRE saledate=2008-03/
PRE saledate=2008-04/
PRE saledate=2008-05/
PRE saledate=2008-06/
PRE saledate=2008-12/
```

아직 외부 스키마가 없는 경우에는 다음 명령을 실행합니다. AWS Identity and Access Management(IAM) 역할의 Amazon 리소스 이름(ARN)을 대체합니다.

```
create external schema spectrum
from data catalog
database 'spectrumdb'
iam_role 'arn:aws:iam::123456789012:role/myspectrumrole'
create external database if not exists;
```

예 1: 단일 파티션 키로 분할

이 다음 예에서는 월 기준으로 분할된 외부 테이블을 만듭니다.

월 기준으로 분할된 외부 테이블을 만들려면 다음 명령을 실행합니다.

```
create external table spectrum.sales_part(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
partitioned by (saledate char(10))
row format delimited
fields terminated by '|'
stored as textfile
location 's3://redshift-downloads/tickit/spectrum/sales_partition/'
table properties ('numRows'='172000');
```

파티션을 추가하려면 다음 ALTER TABLE 명령을 실행합니다.

```
alter table spectrum.sales_part add
partition(saledate='2008-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-01/'

partition(saledate='2008-03')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-03/'

partition(saledate='2008-04')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-04/';
```

파티셔닝된 테이블에서 데이터를 선택하려면 다음 쿼리를 실행합니다.

```
select top 5 spectrum.sales_part.eventid, sum(spectrum.sales_part.pricepaid)
from spectrum.sales_part, event
where spectrum.sales_part.eventid = event.eventid
  and spectrum.sales_part.pricepaid > 30
  and saledate = '2008-01'
group by spectrum.sales_part.eventid
order by 2 desc;
```

```
eventid | sum
```

```
--------+---------
   4124 | 21179.00
   1924 | 20569.00
   2294 | 18830.00
   2260 | 17669.00
   6032 | 17265.00
```

외부 테이블 파티션을 보려면 SVV_EXTERNAL_PARTITIONS 시스템 뷰를 쿼리하십시오.

```
select schemaname, tablename, values, location from svv_external_partitions
where tablename = 'sales_part';
```

```
schemaname | tablename  | values      | location

-----------+------------+-------------
+-------------------------------------------------------------------------
spectrum   | sales_part | ["2008-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-01
spectrum   | sales_part | ["2008-03"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-03
spectrum   | sales_part | ["2008-04"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-04
```

예 2: 여러 파티션 키로 분할

`date` 및 `eventid`로 분할된 외부 테이블을 만들려면 다음 명령을 실행합니다.

```
create external table spectrum.sales_event(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
partitioned by (salesmonth char(10), event integer)
row format delimited
fields terminated by '|'
stored as textfile
```

```
location 's3://redshift-downloads/tickit/spectrum/salesevent/'
table properties ('numRows'='172000');
```

파티션을 추가하려면 다음 ALTER TABLE 명령을 실행합니다.

```
alter table spectrum.sales_event add
partition(salesmonth='2008-01', event='101')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-01/
event=101/'

partition(salesmonth='2008-01', event='102')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-01/
event=102/'

partition(salesmonth='2008-01', event='103')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-01/
event=103/'

partition(salesmonth='2008-02', event='101')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-02/
event=101/'

partition(salesmonth='2008-02', event='102')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-02/
event=102/'

partition(salesmonth='2008-02', event='103')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-02/
event=103/'

partition(salesmonth='2008-03', event='101')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-03/
event=101/'

partition(salesmonth='2008-03', event='102')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-03/
event=102/'

partition(salesmonth='2008-03', event='103')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-03/
event=103/';
```

다음 쿼리를 실행하여 파티셔닝된 테이블에서 데이터를 선택합니다.

```
select spectrum.sales_event.salesmonth, event.eventname,
 sum(spectrum.sales_event.pricepaid)
from spectrum.sales_event, event
where spectrum.sales_event.eventid = event.eventid
  and salesmonth = '2008-02'
 and (event = '101'
 or event = '102'
 or event = '103')
group by event.eventname, spectrum.sales_event.salesmonth
order by 3 desc;
```

```
salesmonth | eventname       | sum
-----------+-----------------+--------
2008-02    | The Magic Flute | 5062.00
2008-02    | La Sonnambula   | 3498.00
2008-02    | Die Walkure     |  534.00
```

외부 테이블 열을 ORC 열에 매핑

Amazon Redshift Spectrum은 외부 테이블을 사용하여 ORC 형식의 파일에서 데이터를 쿼리합니다. ORC(Optimized Row Columnar) 형식은 중첩된 데이터 구조를 지원하는 컬럼 방식 스토리지 파일 형식입니다. 중첩 데이터 쿼리에 대한 자세한 내용은 Amazon Redshift Spectrum을 사용한 중첩 데이터에 대한 쿼리 섹션을 참조하세요.

ORC 파일의 데이터를 참조하는 외부 테이블을 만드는 경우 외부 테이블의 각 열을 ORC 데이터의 열에 매핑합니다. 이렇게 하려면 다음 방법 중 하나를 사용하십시오.

- 위치별 매핑
- 열 이름별 매핑

기본값은 열 이름별 매핑입니다.

위치별 매핑

위치별 매핑을 사용하면 외부 테이블에 정의된 첫 번째 열이 ORC 데이터 파일의 첫 번째 열에 매핑되고 두 번째 열은 두 번째 열에 매핑됩니다. 위치별 매핑을 사용하려면 외부 테이블과 ORC 파일의 열 순서가 일치해야 합니다. 열의 순서가 일치하지 않으면 열을 이름별로 매핑할 수 있습니다.

> **Important**
>
> 이전 릴리스에서는 기본적으로 위치별 매핑을 사용했습니다. 기존 테이블에 대해 위치별 매핑을 계속 사용해야 하는 경우, 다음 예와 같이 테이블 속성 `orc.schema.resolution`을 `position`으로 설정합니다.
>
> ```
> alter table spectrum.orc_example
> set table properties('orc.schema.resolution'='position');
> ```

예를 들어 `SPECTRUM.ORC_EXAMPLE` 테이블은 다음과 같이 정의됩니다.

```
create external table spectrum.orc_example(
int_col int,
float_col float,
nested_col struct<
  "int_col" : int,
  "map_col" : map<int, array<float >>
   >
) stored as orc
location 's3://example/orc/files/';
```

이 테이블 구조는 다음과 같이 추상화될 수 있습니다.

```
• 'int_col' : int
• 'float_col' : float
• 'nested_col' : struct
   o 'int_col' : int
   o 'map_col' : map
      - key : int
      - value : array
         - value : float
```

기본 ORC 파일의 파일 구조는 다음과 같습니다.

```
• ORC file root(id = 0)
   o 'int_col' : int (id = 1)
   o 'float_col' : float (id = 2)
   o 'nested_col' : struct (id = 3)
      - 'int_col' : int (id = 4)
      - 'map_col' : map (id = 5)
```

```
         - key : int (id = 6)
         - value : array (id = 7)
            - value : float (id = 8)
```

이 예에서는 외부 테이블의 각 열을 위치별로 ORC 파일의 열에 정확하게 매핑할 수 있습니다. 다음은 매핑을 보여줍니다.

외부 테이블 열 이름	ORC 열 ID	ORC 열 이름
int_col	1	int_col
float_col	2	float_col
nested_col	3	nested_col
nested_col.int_col	4	int_col
nested_col.map_col	5	map_col
nested_col.map_col.key	6	NA
nested_col.map_col.value	7	NA
nested_col.map_col.value.item	8	NA

열 이름별 매핑

이름 매핑을 사용하면 동일한 레벨에 있는 ORC 파일에서 같은 이름으로 명명된 열에 외부 테이블의 열을 매핑할 수 있습니다.

예를 들어, 앞의 예인 SPECTRUM.ORC_EXAMPLE의 테이블을 다음 파일 구조를 사용하는 ORC 파일과 매핑하려고 한다고 가정합니다.

```
• ORC file root(id = 0)
   o 'nested_col' : struct (id = 1)
      - 'map_col' : map (id = 2)
         - key : int (id = 3)
         - value : array (id = 4)
            - value : float (id = 5)
      - 'int_col' : int (id = 6)
   o 'int_col' : int (id = 7)
```

```
    o 'float_col' : float (id = 8)
```

Redshift Spectrum은 위치 매핑을 사용하여 다음과 같은 매핑을 시도합니다.

외부 테이블 열 이름	ORC 열 ID	ORC 열 이름
int_col	1	struct
float_col	7	int_col
nested_col	8	float_col

앞의 위치 매핑을 사용하여 테이블을 쿼리할 경우 구조가 다르기 때문에 유형 검증 시 SELECT 명령이 실패합니다.

열 이름 매핑을 사용하여 이전 예에 표시된 두 파일 구조에 동일한 외부 테이블을 매핑할 수 있습니다. 테이블 열 `int_col`, `float_col` 및 `nested_col`은 열 이름별 매핑을 통해 ORC 파일에서 같은 이름을 가진 열에 매핑됩니다. 외부 테이블의 `nested_col` 열은 `map_col` 및 `int_col` 하위 열이 있는 `struct` 열입니다. 또한 하위 열은 열 이름별로 ORC 파일의 해당 열과 올바르게 매핑됩니다.

Apache Hudi에서 관리되는 데이터에 대한 외부 테이블 생성

Apache Hudi CoW(쓸 때 복사) 형식으로 데이터를 쿼리하려면 Amazon Redshift Spectrum 외부 테이블을 사용합니다. Hudi Copy On Write 테이블은 Amazon S3에 저장된 Apache Parquet 파일의 모음입니다. 삽입, 삭제, upsert 쓰기 작업으로 생성 및 수정된 Apache Hudi 버전 0.5.2, 0.6.0, 0.7.0, 0.8.0, 0.9.0, 0.10.0, 0.10.1, 0.11.0, 0.11.1에서 쓸 때 복사(CoW) 테이블을 읽을 수 있습니다. 예를 들어 부트스트랩 테이블은 지원되지 않습니다. 자세한 내용은 오픈 소스 Apache Hudi 설명서의 Copy On Write Table 섹션을 참조하세요.

Hudi CoW 형식의 데이터를 참조하는 외부 테이블을 생성하는 경우 외부 테이블의 각 열을 Hudi 데이터의 열에 매핑합니다. 매핑은 열별로 수행됩니다.

분할 및 분할되지 않은 Hudi 테이블에 대한 데이터 정의 언어(DDL) 문은 다른 Apache Parquet 파일 형식에 대한 문과 유사합니다. Hudi 테이블의 경우 `INPUTFORMAT`을 `org.apache.hudi.hadoop.HoodieParquetInputFormat`으로 정의합니다. `LOCATION` 파라미터는 Hudi 커밋 타임라인을 설정하는 데 필요한 `.hoodie` 폴더가 포함된 Hudi 테이블 기본 폴더를 가리켜야 합니다. 경우에 따라 Hudi 테이블에 대한 SELECT 작업이 실패하고 [유효한 Hudi 커밋 타임라인 없음(No valid Hudi commit timeline found)] 메시지가 표시될 수 있습니다. 이러한 경우 `.hoodie` 폴더가 올바른 위치에 있고 유효한 Hudi 커밋 타임라인이 포함되어 있는지 확인합니다.

> **Note**
>
> Apache Hudi 형식은 AWS Glue Data Catalog를 사용할 때만 지원됩니다. Apache Hive 메타스토어를 외부 카탈로그로 사용할 때는 지원되지 않습니다.

분할되지 않은 테이블을 정의하는 DDL의 형식은 다음과 같습니다.

```
CREATE EXTERNAL TABLE tbl_name (columns)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS
INPUTFORMAT 'org.apache.hudi.hadoop.HoodieParquetInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://s3-bucket/prefix'
```

분할된 테이블을 정의하는 DDL의 형식은 다음과 같습니다.

```
CREATE EXTERNAL TABLE tbl_name (columns)
PARTITIONED BY(pcolumn1 pcolumn1-type[,...])
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS
INPUTFORMAT 'org.apache.hudi.hadoop.HoodieParquetInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://s3-bucket/prefix'
```

분할된 Hudi 테이블에 파티션을 추가하려면 ALTER TABLE ADD PARTITION 명령을 실행합니다. 여기서 LOCATION 파라미터는 파티션에 속한 파일이 있는 Amazon S3 하위 폴더를 가리킵니다.

파티션을 추가하는 DDL의 형식은 다음과 같습니다.

```
ALTER TABLE tbl_name
ADD IF NOT EXISTS PARTITION(pcolumn1=pvalue1[,...])
LOCATION 's3://s3-bucket/prefix/partition-path'
```

Delta Lake에서 관리되는 데이터에 대한 외부 테이블 생성

Delta Lake 테이블의 데이터를 쿼리하려면 Amazon Redshift Spectrum 외부 테이블을 사용합니다.

Redshift Spectrum에서 Delta Lake 테이블에 액세스하려면 쿼리 전에 매니페스트를 생성합니다. Delta Lake 매니페스트에는 Delta Lake 테이블의 일관된 스냅샷을 구성하는 파일 목록이 들어 있습니다. 분

할된 테이블에는 파티션당 하나의 매니페스트가 있습니다. Delta Lake 테이블은 Amazon S3에 저장된 Apache Parquet 파일의 모음입니다. 자세한 내용은 오픈 소스 Delta Lake 설명서의 Delta Lake 섹션을 참조하세요.

Delta Lake 테이블의 데이터를 참조하는 외부 테이블을 생성하는 경우 외부 테이블의 각 열을 Delta Lake 테이블의 열에 매핑합니다. 매핑은 열 이름별로 수행됩니다.

분할 및 분할되지 않은 Delta Lake 테이블의 DDL은 다른 Apache Parquet 파일 형식의 DDL과 유사합니다. Delta Lake 테이블에 대해 `INPUTFORMAT`을 `org.apache.hadoop.hive.ql.io.SymlinkTextInputFormat`으로, `OUTPUTFORMAT`을 `org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat`으로 정의합니다. `LOCATION` 파라미터는 테이블 기본 폴더의 매니페스트 폴더를 가리켜야 합니다. Delta Lake 테이블에서 SELECT 작업이 실패하는 경우 가능한 이유는 Delta Lake 테이블에 대한 제한 사항 및 문제 해결 섹션을 참조하세요.

분할되지 않은 테이블을 정의하는 DDL의 형식은 다음과 같습니다.

```
CREATE EXTERNAL TABLE tbl_name (columns)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS
INPUTFORMAT 'org.apache.hadoop.hive.ql.io.SymlinkTextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://s3-bucket/prefix/_symlink_format_manifest'
```

분할된 테이블을 정의하는 DDL의 형식은 다음과 같습니다.

```
CREATE EXTERNAL TABLE tbl_name (columns)
PARTITIONED BY(pcolumn1 pcolumn1-type[,...])
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS
INPUTFORMAT 'org.apache.hadoop.hive.ql.io.SymlinkTextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://s3-bucket>/prefix/_symlink_format_manifest'
```

분할된 Delta Lake 테이블에 파티션을 추가하려면 ALTER TABLE ADD PARTITION 명령을 실행합니다. 여기서 `LOCATION` 파라미터는 파티션의 매니페스트가 들어 있는 Amazon S3 하위 폴더를 가리킵니다.

파티션을 추가하는 DDL의 형식은 다음과 같습니다.

```
ALTER TABLE tbl_name
```

```
ADD IF NOT EXISTS PARTITION(pcolumn1=pvalue1[,...])
LOCATION
's3://s3-bucket/prefix/_symlink_format_manifest/partition-path'
```

또는 Delta Lake 매니페스트 파일을 직접 가리키는 DDL을 실행합니다.

```
ALTER TABLE tbl_name
ADD IF NOT EXISTS PARTITION(pcolumn1=pvalue1[,...])
LOCATION
's3://s3-bucket/prefix/_symlink_format_manifest/partition-path/manifest'
```

Delta Lake 테이블에 대한 제한 사항 및 문제 해결

Redshift Spectrum에서 Delta Lake 테이블을 쿼리할 때 다음 사항을 고려하세요.

- 매니페스트가 더 이상 존재하지 않는 스냅샷이나 파티션을 가리키는 경우 유효한 새 매니페스트가 생성될 때까지 쿼리가 실패합니다. 예를 들어 이는 기본 테이블에 대한 VACUUM 작업의 결과일 수 있습니다.
- Delta Lake 매니페스트는 파티션 수준의 일관성만 제공합니다.

다음 표에서는 Delta Lake 테이블을 쿼리할 때 특정 오류에 대한 몇 가지 잠재적인 원인을 설명합니다.

오류 메시지	가능한 원인
버킷 s3-bucket-1의 Delta Lake 매니페스트는 버킷 s3-bucket-2의 항목을 포함할 수 없습니다.	매니페스트 항목은 지정된 버킷과 다른 Amazon S3 버킷에 있는 파일을 가리킵니다.
Delta Lake 파일은 동일한 폴더에 있어야 합니다.	매니페스트 항목은 지정된 접두사와 다른 Amazon S3 접두사를 가진 파일을 가리킵니다.
Delta Lake 매니페스트 manifest-path에 나열된 파일 filename을 찾을 수 없습니다.	매니페스트에 나열된 파일을 Amazon S3에서 찾을 수 없습니다.
Delta Lake 매니페스트를 가져오는 중 오류가 발생했습니다.	Amazon S3에서 매니페스트를 찾을 수 없습니다.
유효하지 않은 S3 경로입니다.	매니페스트 파일의 항목이 유효한 Amazon S3 경로가 아니거나 매니페스트 파일이 손상되었습니다.

Amazon Redshift와 함께 Apache Iceberg 테이블 사용

이 주제에서는 Redshift Spectrum 또는 Redshift Serverless에서 Apache Iceberg 형식의 테이블을 사용하는 방법을 설명합니다. Apache Iceberg는 대규모 분석 테이블을 위한 고성능 형식입니다.

Redshift Spectrum 또는 Redshift Serverless를 사용하여 AWS Glue Data Catalog에 카탈로그화된 Apache Iceberg 테이블을 쿼리할 수 있습니다. Apache Iceberg는 데이터 레이크를 위한 오픈 소스 테이블 형식입니다. 자세한 내용은 공식 Apache Iceberg 설명서의 Apache Iceberg를 참조하세요.

Amazon Redshift는 Apache Iceberg 테이블 쿼리를 위한 트랜잭션 일관성을 제공합니다. Amazon Redshift를 사용하여 쿼리를 실행하는 동안 Amazon Athena 및 Amazon EMR과 같은 ACID(원자성, 일관성, 격리, 내구성) 호환 서비스를 사용하여 테이블의 데이터를 조작할 수 있습니다. Amazon Redshift는 Apache Iceberg 메타데이터에 저장된 테이블 통계를 사용하여 쿼리 계획을 최적화하고 쿼리 처리 중에 파일 스캔을 줄일 수 있습니다. Amazon Redshift SQL을 사용하면 Redshift 테이블을 데이터 레이크 테이블과 조인할 수 있습니다.

Amazon Redshift와 함께 Iceberg 테이블 사용을 시작하려면 다음을 수행하세요.

1. Amazon Athena 또는 Amazon EMR과 같은 호환되는 서비스를 사용하여 AWS Glue Data Catalog 데이터베이스에 Apache Iceberg 테이블을 만듭니다. Athena를 사용하여 Iceberg 테이블을 생성하려면 Amazon Athena 사용 설명서에서 Apache Iceberg 테이블 사용을 참조하세요.
2. 데이터 레이크에 대한 액세스를 허용하는 연결된 IAM 역할이 있는 Amazon Redshift 클러스터 또는 Redshift Serverless 작업 그룹을 만듭니다. 클러스터 또는 작업 그룹을 만드는 방법에 대한 자세한 내용은 Amazon Redshift 시작 안내서에서 Amazon Redshift 프로비저닝된 데이터 웨어하우스 시작하기 및 Amazon Redshift Serverless를 참조하세요.
3. 쿼리 에디터 v2 또는 타사 SQL 클라이언트를 사용하여 클러스터 또는 작업 그룹에 연결합니다. 쿼리 에디터 v2를 사용하여 연결하는 방법에 대한 자세한 내용은 Amazon Redshift 관리 안내서에서 SQL 클라이언트 도구를 사용하여 Amazon Redshift 데이터 웨어하우스에 연결 단원을 참조하세요.
4. Iceberg 테이블을 포함하는 특정 데이터 카탈로그 데이터베이스에 대한 외부 스키마를 Amazon Redshift 데이터베이스에 생성합니다. 외부 스키마 생성에 대한 자세한 내용은 Amazon Redshift Spectrum의 외부 스키마를 참조하세요.
5. SQL 쿼리를 실행하여 생성한 외부 스키마의 Iceberg 테이블에 액세스합니다.

Amazon Redshift와 함께 Apache Iceberg 테이블을 사용할 때 고려 사항

Iceberg 테이블과 함께 Amazon Redshift를 사용할 때는 다음 사항을 고려하세요.

- Iceberg 버전 지원 - Amazon Redshift는 다음 버전의 Iceberg 테이블에 대한 쿼리 실행을 지원합니다.
  - 버전 1은 변경 불가능한 데이터 파일을 사용하여 대규모 분석 테이블을 관리하는 방법을 정의합니다.
  - 버전 2는 기존 데이터 파일을 변경하지 않은 채로 행 수준 업데이트 및 삭제를 지원하고 삭제 파일을 사용하여 테이블 데이터 변경을 처리하는 기능을 추가합니다.

  버전 1과 버전 2 테이블의 차이점에 대해서는 Apache Iceberg 설명서에서 형식 버전 변경을 참조하세요.
- 쿼리만 지원 - Amazon Redshift는 Apache Iceberg 테이블에 대한 읽기 전용 액세스를 지원합니다. 트랜잭션의 일관된 선택 쿼리를 지원합니다. Amazon Athena와 같은 서비스를 사용하여 AWS Glue Data Catalog에서 Iceberg 테이블의 스키마를 정의하고 업데이트할 수 있습니다.
- 파티션 추가 - Apache Iceberg 테이블에 대한 파티션을 수동으로 추가할 필요가 없습니다. Apache Iceberg 테이블의 새 파티션은 Amazon Redshift에 의해 자동으로 감지되며 테이블 정의에서 파티션을 업데이트하기 위해 수동 작업이 필요하지 않습니다. 파티션 사양의 모든 변경 사항도 사용자 개입 없이 쿼리에 자동으로 적용됩니다.
- Iceberg 데이터를 Amazon Redshift로 수집 - INSERT INTO 또는 CREATE TABLE AS 명령을 사용하여 Iceberg 테이블의 데이터를 로컬 Amazon Redshift 테이블로 가져올 수 있습니다. 현재 COPY 명령을 사용하여 Apache Iceberg 테이블의 콘텐츠를 로컬 Amazon Redshift 테이블로 수집할 수 없습니다.
- 구체화된 뷰 - Amazon Redshift의 다른 외부 테이블과 마찬가지로 Apache Iceberg 테이블에서 구체화된 뷰를 만들 수 있습니다. 다른 데이터 레이크 테이블 형식에 대한 동일한 고려 사항이 Apache Iceberg 테이블에도 적용됩니다. 데이터 레이크 테이블의 증분 업데이트, 자동 새로 고침, 자동 쿼리 재작성 및 자동 MV는 현재 지원되지 않습니다.
- AWS Lake Formation의 세분화된 액세스 제어 - Amazon Redshift는 Apache Iceberg 테이블에서 AWS Lake Formation의 세분화된 액세스 제어를 지원합니다.
- 사용자 정의 데이터 처리 파라미터 - Amazon Redshift는 Apache Iceberg 테이블에서 사용자 정의 데이터 처리 파라미터를 지원합니다. 기존 파일에 사용자 정의 데이터 처리 파라미터를 사용하여 외부 테이블에서 쿼리되는 데이터를 조정하여 스캔 오류를 방지할 수 있습니다. 이러한 파라미터는 테이블 스키마와 파일의 실제 데이터 간의 불일치를 처리하는 기능을 제공합니다. 사용자 정의 데이터 처리 파라미터를 Apache Iceberg 테이블에서도 사용할 수 있습니다.
- 시간 이동 쿼리- 시간 이동 쿼리는 현재 Apache Iceberg 테이블에서 지원되지 않습니다.
- 요금 - 클러스터에서 Iceberg 테이블에 액세스하면 Redshift Spectrum 요금이 청구됩니다. 작업 그룹에서 Iceberg 테이블에 액세스하는 경우 Redshift Serverless 요금이 청구됩니다. Redshift Spectrum 및 Redshift Serverless 요금에 대한 자세한 내용은 Amazon Redshift 요금을 참조하세요.

Apache Iceberg 테이블에서 지원되는 데이터 유형

이 주제에서는 Redshift Spectrum이 Apache Iceberg 형식의 테이블에서 읽을 수 있는 지원되는 데이터 유형에 대해 설명합니다.

Amazon Redshift는 다음 데이터 형식이 포함된 Iceberg 테이블을 쿼리할 수 있습니다.

```
binary
boolean
date
decimal
double
float
int
list
long
map
string
struct
timestamp without time zone
```

Iceberg 데이터 형식에 대한 자세한 내용은 Apache Iceberg 설명서에서 Iceberg용 스키마를 참조하세요.

다음 테이블에서는 Amazon Redshift 데이터 형식과 Iceberg 테이블 데이터 형식 간의 관계를 보여줍니다.

Iceberg 형식	Amazon Redshift 형식	참고
boolean	boolean	
-	tinyint	Amazon Redshift의 Iceberg 테이블에는 지원되지 않습니다.
-	smallint	Amazon Redshift의 Iceberg 테이블에는 지원되지 않습니다.
int	int	Amazon Redshift SQL 문에서 이 형식은 INTEGER입니다.
long	bigint	

Iceberg 형식	Amazon Redshift 형식	참고
double	double	
float	float	
decimal(P, S)	decimal(P, S)	P는 정밀도이며, S는 스케일입니다.
-	char	Redshift Spectrum의 Iceberg 테이블에는 지원되지 않습니다.
string	string	Amazon Redshift SQL 문에서 이 형식은 VARCHAR입니다.
binary	binary	
date	date	
time	-	
timestamp	timestamp	
timestamptz	-	
list<E>	array	
map<K,V>	map	
struct<...>	struct	
fixed(L)	-	fixed(L) 형식은 현재 Redshift Spectrum에서 지원되지 않습니다.

Amazon Redshift의 데이터 형식에 대한 자세한 내용은 데이터 타입을 참조하세요.

Amazon Redshift Spectrum 쿼리 성능

이 주제에서는 Redshift Spectrum 쿼리 성능을 개선하는 방법에 대해 설명합니다.

쿼리 계획을 보고 Amazon Redshift Spectrum 계층으로 어떤 단계들이 푸시되었는지 확인합니다.

Redshift Spectrum 쿼리와 관련된 단계들은 다음과 같습니다.

- S3 Seq Scan
- S3 HashAggregate
- S3 Query Scan
- Seq Scan PartitionInfo
- Partition Loop

다음 예는 외부 테이블을 로컬 테이블에 조인하는 쿼리의 쿼리 계획을 보여 줍니다. Amazon S3에 있는 데이터에 대해 S3 Seq Scan 단계와 S3 HashAggregate 단계가 실행되었음에 유의합니다.

```
explain
select top 10 spectrum.sales.eventid, sum(spectrum.sales.pricepaid)
from spectrum.sales, event
where spectrum.sales.eventid = event.eventid
and spectrum.sales.pricepaid > 30
group by spectrum.sales.eventid
order by 2 desc;
```

```
QUERY PLAN


------------------------------------------------------------------------------
XN Limit  (cost=1001055770628.63..1001055770628.65 rows=10 width=31)


  ->  XN Merge  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)


        Merge Key: sum(sales.derived_col2)
```

```
        ->  XN Network  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

              Send to leader

              ->  XN Sort  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

                    Sort Key: sum(sales.derived_col2)

                    ->  XN HashAggregate  (cost=1055770620.49..1055770620.99 rows=200
width=31)

                          ->  XN Hash Join DS_BCAST_INNER  (cost=3119.97..1055769620.49
rows=200000 width=31)

                                Hash Cond: ("outer".derived_col1 = "inner".eventid)

                                ->  XN S3 Query Scan sales  (cost=3010.00..5010.50
rows=200000 width=31)

                                      ->  S3 HashAggregate  (cost=3010.00..3010.50
rows=200000 width=16)

                                            ->  S3 Seq Scan spectrum.sales
location:"s3://redshift-downloads/tickit/spectrum/sales" format:TEXT
(cost=0.00..2150.00 rows=172000 width=16)
                                                  Filter: (pricepaid > 30.00)

                                ->  XN Hash  (cost=87.98..87.98 rows=8798 width=4)

                                      ->  XN Seq Scan on event  (cost=0.00..87.98
rows=8798 width=4)
```

쿼리 계획에서 다음 요소들에 유의하십시오.

- S3 Seq Scan 노드는 필터 pricepaid > 30.00가 Redshift Spectrum 계층에서 처리되었음을 보여 줍니다.

`XN S3 Query Scan` 노드 아래의 필터 노드는 Redshift Spectrum 계층에서 반환되는 데이터에 기반한 Amazon Redshift에서의 조건자 처리를 나타냅니다.

- `S3 HashAggregate` 노드는 절(`group by spectrum.sales.eventid`)에 의한 Redshift Spectrum 계층에서의 집계를 나타냅니다.

Redshift Spectrum 성능을 개선할 수 있는 방법은 다음과 같습니다.

- Apache Parquet 형식의 데이터 파일을 사용합니다. Parquet은 데이터를 컬럼 형식으로 저장하므로 Redshift Spectrum이 스캔에서 불필요한 열을 제거할 수 있습니다. 데이터가 텍스트 파일 형식인 경우, Redshift Spectrum은 전체 파일을 스캔해야 합니다.
- 병렬 처리에 최적화되도록 여러 파일을 사용합니다. 파일 크기를 64MB 이상으로 유지하십시오. 파일 크기를 비슷하게 유지해 데이터 크기 스큐를 피하십시오. Apache Parquet 파일 및 구성 권장 사항에 대한 자세한 내용은 Apache Parquet 설명서의 파일 형식: 구성을 참조하세요.
- 쿼리에서 사용하는 열을 최소화하십시오.
- 큰 팩트 테이블은 Amazon S3에 넣고 자주 사용하는 작은 차원 테이블은 로컬 Amazon Redshift 데이터베이스에 둡니다.
- TABLE PROPERTIES numRows 속성을 설정하여 외부 테이블 통계를 업데이트합니다. CREATE EXTERNAL TABLE 또는 ALTER TABLE을 사용하여 테이블의 행 수를 반영하도록 TABLE PROPERTIES numRows 파라미터를 설정합니다. Amazon Redshift는 쿼리 옵티마이저가 쿼리 계획을 생성하는 데 사용하는 테이블 통계를 생성하기 위해 외부 테이블을 분석하지 않습니다. 테이블 통계가 외부 테이블에 대해 설정되지 않은 경우 Amazon Redshift는 쿼리 실행 계획을 실행합니다. Amazon Redshift는 외부 테이블이 더 큰 테이블이고 로컬 테이블이 더 작은 테이블이라는 가정을 기반으로 이 계획을 생성합니다.
- Amazon Redshift 쿼리 플래너는 가능하면 항상 조건자 및 집계를 Redshift Spectrum 쿼리 계층으로 푸시합니다. Amazon S3에서 대량의 데이터가 반환되면 클러스터의 리소스에 의해 처리가 제한됩니다. Redshift Spectrum은 대량의 요청을 처리하기 위해 자동으로 조정됩니다. 따라서 처리를 Redshift Spectrum 계층으로 푸시할 수 있을 때마다 전반적인 성능이 향상됩니다.
- Redshift Spectrum 계층으로 푸시할 수 있는 필터와 집계를 사용하도록 쿼리를 작성하십시오.

  다음은 Redshift Spectrum 계층으로 푸시할 수 있는 몇 가지 연산의 예입니다.

  - GROUP BY 절
  - 비교 조건 및 패턴 일치 조건(예: LIKE).
  - COUNT, SUM, AVG, MIN, MAX 등 집계 함수.
  - 문자열 함수.

Redshift Spectrum 계층으로 푸시할 수 없는 연산으로는 DISTINCT와 ORDER BY가 있습니다.

- 파티션을 사용하여 스캔되는 데이터를 제한하십시오. 가장 일반적인 쿼리 조건자를 기준으로 데이터를 파티셔닝한 다음 파티션 열을 기준으로 파티션을 정리합니다. 자세한 내용은 Redshift Spectrum 외부 테이블 파티셔닝 단원을 참조하십시오.

  SVL_S3PARTITION을 쿼리하여 총 파티션과 적격 파티션을 확인합니다.
- AWS Glue의 통계 생성기를 사용하여 AWS Glue Data Catalog 테이블의 열 수준 통계를 계산합니다. AWS Glue가 데이터 카탈로그의 테이블에 대한 통계를 생성하고 나면 Amazon Redshift Spectrum은 자동으로 해당 통계를 사용하여 쿼리 계획을 최적화합니다. AWS Glue를 사용하여 열 수준 통계를 계산하는 방법에 대한 자세한 내용은 AWS Glue 개발자 안내서의 열 통계 작업을 참조하세요.

데이터 처리 옵션

이 주제에서는 Redshift Spectrum이 예기치 않은 형식으로 데이터를 처리하는 방식을 구성하는 방법을 설명합니다.

외부 테이블을 만들 때 테이블 파라미터를 설정하여 외부 테이블에서 쿼리되는 데이터를 조정할 수 있습니다. 그러지 않으면 스캔 오류가 발생할 수 있습니다. 자세한 내용은 CREATE EXTERNAL TABLE에서 테이블 속성을 참조하세요. 예시는 데이터 처리 예 섹션을 참조하세요. 오류 목록은 SVL_SPECTRUM_SCAN_ERROR 단원을 참조하세요.

외부 테이블을 만들 때 다음 테이블 속성을 설정하여 외부 테이블에서 쿼리되는 데이터의 입력 처리를 지정할 수 있습니다.

- `column_count_mismatch_handling`은 파일에 포함된 행 값이 외부 테이블 정의에 지정된 열 수보다 적거나 많은지 식별합니다.
- `invalid_char_handling` - VARCHAR, CHAR 및 문자열 데이터를 포함하는 열에서 잘못된 문자의 입력 처리를 지정합니다. `invalid_char_handling`에 REPLACE를 지정하는 경우 사용할 대체 문자를 지정할 수 있습니다.
- `numeric_overflow_handling` - 정수 및 십진수 데이터가 포함된 열에서 캐스트 오버플로우 처리를 지정합니다.
- VARBYTE 데이터를 포함하는 열의 잉여 바이트에 대한 입력 처리를 지정하는 `surplus_bytes_handling`.
- `surplus_char_handling` - VARCHAR, CHAR 및 문자열 데이터를 포함하는 열에서 잉여 문자의 입력 처리를 지정합니다.

최대 오류 수를 초과하는 쿼리를 취소하도록 구성 옵션을 설정할 수 있습니다. 자세한 내용은 spectrum_query_maxerror 단원을 참조하십시오.

예: Redshift Spectrum에서 상관 하위 쿼리 수행

이 주제에서는 Redshift Spectrum에서 상관 하위 쿼리를 수행하는 방법을 설명합니다. 상관 하위 쿼리는 외부 쿼리의 값을 사용하는 쿼리입니다.

Redshift Spectrum에서 상관 하위 쿼리를 수행할 수 있습니다. `$spectrum_oid` 가상 열은 Redshift Spectrum으로 상관 쿼리를 수행하는 기능을 제공합니다. 상호 연관된 하위 쿼리를 수행하려면 가상 열 `$spectrum_oid`가 활성화되어야 하지만 SQL 문에는 나타나지 않습니다. 자세한 내용은 가상 열 단원을 참조하십시오.

이 예제의 외부 스키마 및 외부 테이블을 생성하려면 Amazon Redshift Spectrum 시작하기 섹션을 참조하세요.

다음은 Redshift Spectrum의 상관 하위 쿼리의 예입니다.

```
select *
from myspectrum_schema.sales s
where exists
( select *
from myspectrum_schema.listing l
where l.listid = s.listid )
order by salesid
limit 5;
```

```
salesid  listid   sellerid   buyerid   eventid   dateid  qtysold  pricepaid
 commission   saletime
1        1        36861      21191     7872      1875    4        728         109.2
    2008-02-18 02:36:48
2        4        8117       11498     4337      1983    2        76          11.4
    2008-06-06 05:00:16
3        5        1616       17433     8647      1983    2        350         52.5
    2008-06-06 08:26:17
4        5        1616       19715     8647      1986    1        175         26.25
    2008-06-09 08:38:52
5        6        47402      14115     8240      2069    2        154         23.1
    2008-08-31 09:17:02
```

Amazon Redshift Spectrum의 지표

이 주제에서는 Redshift Spectrum 쿼리를 모니터링하는 데 사용할 수 있는 시스템 뷰에 대해 설명합니다.

다음 시스템 뷰를 사용하여 Amazon Redshift Spectrum 쿼리를 모니터링할 수 있습니다.

- SVL_S3QUERY

  세그먼트 및 노드 조각 수준에서 Redshift Spectrum 쿼리(S3 쿼리)에 대한 세부 정보를 가져오려면 SVL_S3QUERY 뷰를 사용합니다.
- SVL_S3QUERY_SUMMARY

  시스템에서 실행된 모든 Amazon Redshift Spectrum 쿼리(S3 쿼리)의 요약을 가져오려면 SVL_S3QUERY_SUMMARY 뷰를 사용합니다.

다음은 SVL_S3QUERY_SUMMARY에서 찾아봐야 할 몇 가지입니다.

- Redshift Spectrum 쿼리에 의해 처리된 파일 수.
- Amazon S3에서 스캔한 바이트 수. Redshift Spectrum 쿼리 비용은 Amazon S3에서 스캔된 데이터 양에 반영됩니다.
- Redshift Spectrum 계층에서 클러스터로 반환된 바이트의 수. 반환되는 데이터의 양이 많으면 시스템 성능에 영향을 줄 수 있습니다.
- Redshift Spectrum 요청의 최대 지속 시간 및 평균 지속 시간. 요청이 오랫동안 실행되고 있다면 병목 현상이 발생한 것일 수 있습니다.

Amazon Redshift Spectrum의 쿼리 문제 해결

이 주제는 Amazon Redshift Spectrum 쿼리에서 발생할 수 있는 일반적인 문제에 대한 참조입니다.

Redshift Spectrum 쿼리에서 생성된 오류를 보려면 SVL_S3LOG 시스템 테이블에 대해 쿼리를 실행하십시오.

주제

- 재시도 횟수 초과
- 액세스 조절됨

- 리소스 제한 초과됨
- 파티셔닝된 테이블에 대해 반환된 행 없음
- 권한 없음 오류
- 호환되지 않는 데이터 형식
- Amazon Redshift에서 Hive DDL을 사용할 때 구문 오류
- 임시 테이블을 생성할 수 있는 권한
- 잘못된 범위
- 잘못된 Parquet 버전 번호

재시도 횟수 초과

Amazon Redshift Spectrum 요청이 시간 초과되면 요청이 취소되고 다시 제출됩니다. 재시도가 5회 실패하면 다음 오류가 표시되면서 쿼리가 실패합니다.

```
error:  Spectrum Scan Error: Retries exceeded
```

가능한 값은 다음을 포함합니다.

- 큰 파일 사이즈(1GB 이상). Amazon S3의 파일 크기를 확인하고 큰 파일과 파일 크기 스큐를 찾습니다. 큰 파일은 100MB~1GB의 작은 파일로 나눕니다. 파일의 크기를 거의 비슷하게 만드십시오.
- 느린 네트워크 처리량. 쿼리를 나중에 시도하십시오.

액세스 조절됨

Amazon Redshift Spectrum은 다른 AWS 서비스의 서비스 할당량에 따라 달라질 수 있습니다. 사용량이 많으면 Redshift Spectrum 요청 속도가 느려져서 다음과 같은 오류가 발생할 수 있습니다.

```
error:  Spectrum Scan Error: Access throttled
```

두 가지 유형의 조절이 발생할 수 있습니다.

- Amazon S3에 의해 액세스가 제한됩니다.
- AWS KMS에 의해 액세스 조절됨

오류 컨텍스트는 조절 유형에 대한 세부 정보를 제공합니다. 이를 통해 해당 조절에 대한 원인과 가능한 해결 방법을 찾을 수 있습니다.

Amazon S3에 의해 액세스가 제한됨

Amazon S3는 접두사에 대한 읽기 요청 비율이 너무 높은 경우 Redshift Spectrum 요청을 제한할 수 있습니다. Amazon S3에서 달성할 수 있는 GET/HEAD 요청 속도에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 Amazon S3 성능 최적화를 참조하세요. Amazon S3 GET/HEAD 요청 빈도는 접두사의 모든 GET/HEAD 요청을 고려하므로 동일한 접두사에 액세스하는 다른 애플리케이션이 총 요청 빈도를 공유합니다.

Redshift Spectrum 요청이 Amazon S3에 의해 자주 제한되는 경우 Redshift Spectrum이 Amazon S3에 하는 Amazon S3 GET/HEAD 요청 수를 줄입니다. 이렇게 하려면 작은 파일을 큰 파일로 병합합니다. 64MB보다 큰 파일 크기를 사용하는 것이 좋습니다.

또한 초기 필터링의 이점을 활용하고 Amazon S3에서 액세스되는 파일 수를 줄이기 위해 Redshift Spectrum 테이블 분할을 고려합니다. 자세한 내용은 Redshift Spectrum 외부 테이블 파티셔닝 단원을 참조하십시오.

AWS KMS에서 액세스 조절됨

서버 측 암호화(SSE-S3 또는 SSE-KMS)를 사용하여 Amazon S3에 데이터를 저장하는 경우 Amazon S3에서는 Redshift Spectrum이 액세스하는 각 파일에 대해 AWS KMS에 API 작업을 호출합니다. 이러한 요청은 암호화 작업 할당량에 포함됩니다. 자세한 내용은 AWS KMS 요청 할당량을 참조하십시오. SSE-S3 및 SSE-KMS에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 서버 측 암호화를 사용하여 데이터 보호와 AWS KMS에 저장된 키를 사용하는 서버 측 암호화로 데이터 보호을 참조하세요.

Redshift Spectrum이 AWS KMS에 하는 요청 수를 줄이는 첫 번째 단계는 액세스되는 파일 수를 줄이는 것입니다. 이렇게 하려면 작은 파일을 큰 파일로 병합합니다. 64MB보다 큰 파일 크기를 사용하는 것이 좋습니다.

Redshift Spectrum 요청이 AWS KMS에 의해 자주 제한되는 경우 암호화 작업에 대한 AWS KMS 요청 빈도의 할당량 증가 요청을 고려합니다. 할당량 증가를 요청하려면 Amazon Web Services 일반 참조의 AWS 서비스 제한을 참조하세요.

리소스 제한 초과됨

Redshift Spectrum은 요청이 사용할 수 있는 메모리 양에 상한을 적용합니다. 더 많은 메모리가 필요한 Redshift Spectrum 요청은 실패하고 결과적으로 다음 오류가 발생합니다.

```
error:  Spectrum Scan Error: Resource limit exceeded
```

Redshift Spectrum 요청이 허용되는 메모리를 초과하여 실행할 수 있는 두 가지 일반적인 이유가 있습니다.

- Redshift Spectrum은 더 작은 청크로 분할할 수 없는 큰 데이터 청크를 처리합니다.
- 대규모 집계 단계가 Redshift Spectrum에 의해 처리됩니다.

분할 크기가 128MB 이하인 병렬 읽기를 지원하는 파일 형식을 사용하는 것이 좋습니다. 지원되는 파일 형식 및 데이터 파일 생성에 대한 일반 지침은 Amazon Redshift Spectrum의 쿼리용 데이터 파일 섹션을 참조하세요. 병렬 읽기를 지원하지 않는 파일 형식이나 압축 알고리즘을 사용하는 경우 파일 크기를 64MB에서 128MB 사이로 유지하는 것이 좋습니다.

파티셔닝된 테이블에 대해 반환된 행 없음

파티셔닝된 외부 테이블에서 쿼리가 반환하는 행이 없다면 이 외부 테이블에 대해 파티션이 추가되었는지 확인합니다. Redshift Spectrum은 `ALTER TABLE … ADD PARTITION`을 사용하여 명시적으로 추가된, Amazon S3 위치에 있는 파일만을 스캔합니다. SVV_EXTERNAL_PARTITIONS 뷰를 쿼리하여 기존 파티션을 찾습니다. 누락된 각 파티션에 대해 `ALTER TABLE … ADD PARTITION`을 실행합니다.

권한 없음 오류

클러스터의 IAM 역할이 Amazon S3 파일 객체에 대한 액세스를 허용하는지 확인합니다. 외부 데이터베이스가 Amazon Athena에 있는 경우 IAM 역할이 Athena 리소스에 대한 액세스를 허용하는지 확인합니다. 자세한 내용은 Amazon Redshift Spectrum에 대한 IAM 정책 단원을 참조하십시오.

호환되지 않는 데이터 형식

Apache Parquet 같은 컬럼 파일 형식의 경우 데이터에 열 유형이 포함되어 있습니다. CREATE EXTERNAL TABLE 정의의 열 유형이 데이터 파일의 열 유형과 일치해야 합니다. 일치하지 않으면 다음과 유사한 오류 메시지가 표시됩니다.

```
File 'https://s3bucket/location/file has an incompatible Parquet schema
for column ‘s3://s3bucket/location.col1'. Column type: VARCHAR, Par
```

위의 오류 메시지는 메시지 길이 제한으로 인해 잘릴 수도 있습니다. 이때 열 이름 및 형식을 포함하여 전체 오류 메시지를 가져오려면 SVL_S3LOG 시스템 뷰에 대해 쿼리를 실행하면 됩니다.

다음 예는 완료된 마지막 쿼리에 대해 SVL_S3QUERY를 쿼리합니다.

```
select message
from svl_s3log
where query = pg_last_query_id()
order by query,segment,slice;
```

다음은 결과가 전체 오류 메시지를 표시하는 예입니다.

```
                            message
-----------------------------------------------------------------------------------------------
Spectrum Scan Error. File 'https://s3bucket/location/file has an incompatible
Parquet schema for column ' s3bucket/location.col1'.
Column type: VARCHAR, Parquet schema:\noptional int64 l_orderkey [i:0 d:1 r:0]\n
```

오류를 수정하려면 Parquet 파일의 열 유형과 일치하도록 외부 테이블을 변경합니다.

Amazon Redshift에서 Hive DDL을 사용할 때 구문 오류

Amazon Redshift는 Hive DDL과 비슷한 CREATE EXTERNAL TABLE용 DDL(데이터 정의 언어)을 지원합니다. 하지만 이 두 가지 DDL 유형이 항상 똑같은 것은 아닙니다. Hive DDL을 복사해 Amazon Redshift 외부 테이블을 생성하거나 변경하는 경우 구문 오류가 발생할 수 있습니다. 다음은 Amazon Redshift와 Hive DDL의 차이점의 예입니다.

- Amazon Redshift에는 작은따옴표(')가 필요하지만 Hive DDL은 큰따옴표(")를 지원합니다.
- Amazon Redshift는 STRING 데이터 형식을 지원하지 않습니다. 대신 VARCHAR를 사용하십시오.

임시 테이블을 생성할 수 있는 권한

Redshift Spectrum 쿼리를 실행하려면 데이터베이스 사용자가 데이터베이스에서 임시 테이블을 생성할 수 있는 권한을 보유해야 합니다. 다음 예에서는 `spectrumdb` 데이터베이스에 대한 임시 권한을 `spectrumusers` 사용자 그룹에 부여합니다.

```
grant temp on database spectrumdb  to group spectrumusers;
```

자세한 내용은 GRANT 단원을 참조하십시오.

잘못된 범위

Redshift Spectrum은 쿼리 중에 외부 테이블에 속한 Amazon S3 파일을 덮어쓰지 않을 것으로 예상합니다. 이 경우 다음 오류가 발생할 수 있습니다.

```
Error: HTTP response error code: 416 Message: InvalidRange The requested range is not
 satisfiable
```

이 오류를 방지하려면 Redshift Spectrum으로 쿼리되는 동안 Amazon S3 파일을 덮어쓰지 않아야 합니다.

잘못된 Parquet 버전 번호

Redshift Spectrum럼은 액세스하는 각 Apache Parquet 파일의 메타데이터를 확인합니다. 확인이 실패하면 다음과 비슷한 오류가 발생할 수 있습니다.

```
File 'https://s3.region.amazonaws.com/s3bucket/location/file has an invalid version
 number
```

확인이 실패하게 되는 두 가지 일반적인 이유가 있습니다.

- 쿼리 중에 Parquet 파일을 덮어씁니다 (잘못된 범위 참조).
- Parquet 파일이 손상되었습니다.

튜토리얼: Amazon Redshift Spectrum을 사용한 중첩 데이터 쿼리

이 자습서는 Redshift Spectrum을 사용하여 중첩된 데이터를 쿼리하는 방법을 보여줍니다. 중첩 데이터는 중첩된 필드를 포함하는 데이터입니다. 중첩 필드는 배열, 구조체 또는 객체와 같은 단일 엔터티로 함께 조인되는 필드입니다.

주제

개요

Amazon Redshift Spectrum은 Parquet, ORC, JSON 및 Ion 파일 형식에서 중첩 데이터에 대한 쿼리를 지원합니다. Redshift Spectrum은 외부 테이블을 사용하여 데이터에 액세스합니다. 복합 데이터 형식인 `struct`, `array` 및 `map`을 사용하는 외부 테이블을 생성할 수 있습니다.

예를 들어 `customers`라는 폴더에 다음과 같은 Amazon S3 데이터가 저장된 데이터 파일이 있다고 가정하겠습니다. 단일 루트 요소는 없지만 이 샘플 데이터의 각 JSON 객체는 테이블의 행을 나타냅니다.

```
{"id": 1,
 "name": {"given": "John", "family": "Smith"},
 "phones": ["123-457789"],
 "orders": [{"shipdate": "2018-03-01T11:59:59.000Z", "price": 100.50},
            {"shipdate": "2018-03-01T09:10:00.000Z", "price": 99.12}]
}
{"id": 2,
 "name": {"given": "Jenny", "family": "Doe"},
 "phones": ["858-8675309", "415-9876543"],
 "orders": []
}
{"id": 3,
 "name": {"given": "Andy", "family": "Jones"},
 "phones": [],
 "orders": [{"shipdate": "2018-03-02T08:02:15.000Z", "price": 13.50}]
}
```

Amazon Redshift Spectrum을 사용하여 파일의 중첩된 데이터를 쿼리할 수 있습니다. 다음은 이 방법으로 Apache Parquet 데이터를 쿼리하는 방법을 나타내는 자습서입니다.

사전 조건

Redshift Spectrum을 아직 사용하지 않고 있다면 계속 진행하기 전에 Amazon Redshift Spectrum 시작하기에서 다음 단계를 따르세요.

외부 스키마를 만들려면 다음 명령에서 IAM 역할 ARN을 IAM 역할 생성에서 생성한 역할 ARN으로 대체합니다. 그런 다음 SQL 클라이언트에서 명령을 실행합니다.

```
create external schema spectrum
from data catalog
database 'myspectrum_db'
iam_role 'arn:aws:iam::123456789012:role/myspectrum_role'
create external database if not exists;
```

1단계: 중첩 데이터가 포함된 외부 테이블 만들기

Amazon S3에서 소스 데이터를 다운로드하여 볼 수 있습니다.

이번 자습서에서 사용할 외부 테이블을 만들려면 다음과 같이 명령을 실행합니다.

```
CREATE EXTERNAL TABLE spectrum.customers (
  id     int,
  name   struct<given:varchar(20), family:varchar(20)>,
  phones array<varchar(20)>,
  orders array<struct<shipdate:timestamp, price:double precision>>
)
STORED AS PARQUET
LOCATION 's3://redshift-downloads/tickit/spectrum/customers/';
```

위의 예에서 외부 테이블 `spectrum.customers`는 데이터 형식으로 `struct`와 `array`를 사용하여 중첩 데이터가 포함된 열을 정의합니다. Amazon Redshift Spectrum은 Parquet, ORC, JSON 및 Ion 파일 형식에서 중첩 데이터에 대한 쿼리를 지원합니다. `STORED AS` 파라미터는 Apache Parquet 파일의 경우 `PARQUET`입니다. `LOCATION` 파라미터는 중첩 데이터 또는 파일이 들어 있는 Amazon S3 폴더를 참조해야 합니다. 자세한 내용은 CREATE EXTERNAL TABLE 단원을 참조하십시오.

`array`와 `struct` 형식은 모든 수준에서 중첩이 가능합니다. 예를 들어 다음 예와 같이 `toparray`라는 이름으로 열을 정의할 수 있습니다.

```
toparray array<struct<nestedarray:
         array<struct<morenestedarray:
         array<string>>>>>
```

또한 다음 예의 `struct` 열처럼 `x` 형식을 중첩시킬 수도 있습니다.

```
x struct<a: string,
         b: struct<c: integer,
                   d: struct<e: string>
                  >
        >
```

2단계: SQL 확장을 통한 Amazon S3의 중첩 데이터 쿼리

Redshift Spectrum은 Amazon Redshift SQL 구문 확장을 통해 복합 형식인 `array`, `map` 및 `struct`에 대한 쿼리를 지원합니다.

확장 1: 구조체 열에 대한 액세스

필드 이름을 경로로 연결하는 점 표기법을 사용해 struct 열에서 데이터를 추출할 수 있습니다. 예를 들어 다음은 고객의 성과 이름을 반환하는 쿼리입니다. 이름은 긴 경로인 c.name.given을 통해 액세스하고, 성은 긴 경로인 c.name.family을 통해 액세스합니다.

```
SELECT c.id, c.name.given, c.name.family
FROM   spectrum.customers c;
```

위의 쿼리는 다음과 같은 데이터를 반환합니다.

```
id | given | family
---|-------|-------
1  | John  | Smith
2  | Jenny | Doe
3  | Andy  | Jones
(3 rows)
```

struct는 어떤 수준에서든지 다른 struct의 열이 될 수 있고, 이 열은 또다시 다른 struct의 열이 될 수 있습니다. 이렇게 깊게 중첩되는 struct의 열에 액세스하는 경로는 계속해서 길어질 수 있습니다. 예를 들어 다음 예에서 x 열에 대한 정의를 보십시오.

```
x struct<a: string,
         b: struct<c: integer,
                   d: struct<e: string>
                  >
        >
```

e로 x.b.d.e의 데이터에 액세스할 수 있습니다.

확장 2: FROM 절의 포괄적 배열

array 절에서 테이블 이름이 아닌 map 열을 지정하여 array 열(확장 시 FROM 열까지도)의 데이터를 추출할 수 있습니다. 이러한 확장은 기본 쿼리의 FROM 절은 물론이고 하위 쿼리의 FROM 절에도 적용됩니다.

array 요소는 c.orders[0]와 같이 위치로 참조할 수 있습니다(미리 보기).

포괄적 arrays를 조인과 함께 사용하면 다음 사용 사례에서 설명하겠지만 다양한 유형의 중첩 해제가 가능합니다.

내부 조인을 사용한 중첩 해제

다음은 고객 ID와 주문한 고객의 주문 발송일을 선택하는 쿼리입니다. FROM 절의 SQL 확장인 c.orders o는 별칭인 c에 따라 달라집니다.

```
SELECT c.id, o.shipdate
FROM   spectrum.customers c, c.orders o
```

주문한 고객 c마다 FROM 절이 고객 o의 각 주문 c에 대해 행을 하나씩 반환합니다. 각 행은 고객 행 c와 주문 행 o가 함께 표시됩니다. 그러면 SELECT 절에는 c.id와 o.shipdate만 유지됩니다. 결과는 다음과 같습니다.

```
id|      shipdate
--|---------------------
1 |2018-03-01  11:59:59
1 |2018-03-01  09:10:00
3 |2018-03-02  08:02:15
(3 rows)
```

별칭 c는 고객 필드에 대한 액세스를, 그리고 별칭 o는 주문 필드에 대한 액세스를 제공합니다.

시맨틱은 표준 SQL과 비슷합니다. FROM 절을 다음 중첩 루프를 실행하는 것으로 생각할 수 있습니다. 이후에는 출력할 필드를 선택하는 SELECT 절이 나옵니다.

```
for each customer c in spectrum.customers
  for each order o in c.orders
     output c.id and o.shipdate
```

따라서 주문하지 않은 고객은 결과에서도 표시되지 않습니다.

또한 이것을 FROM 테이블과 JOIN 배열을 사용해 customers을 실행하는 orders 절이라고 생각할 수도 있습니다. 실제로 다음 예와 같이 쿼리를 작성하는 것도 가능합니다.

```
SELECT c.id, o.shipdate
FROM   spectrum.customers c INNER JOIN c.orders o ON true
```

> **Note**
>
> 이름이 c인 테이블에 orders라는 스키마가 존재한다면 c.orders는 orders의 배열 열이 아닌 customers 테이블을 참조합니다.

왼쪽 조인을 사용한 중첩 해제

다음은 모든 고객 이름과 고객의 주문을 출력하는 쿼리입니다. 따라서 주문하지 않은 고객의 이름까지 모두 반환됩니다. 하지만 이 경우에는 아래 Jenny Doe의 예와 같이 주문 열이 NULL 값을 갖습니다.

```
SELECT c.id, c.name.given, c.name.family, o.shipdate, o.price
FROM   spectrum.customers c LEFT JOIN c.orders o ON true
```

위의 쿼리는 다음과 같은 데이터를 반환합니다.

```
id  |  given  | family  |    shipdate          | price
----|---------|---------|----------------------|--------
 1  |  John   | Smith   | 2018-03-01  11:59:59 | 100.5
 1  |  John   | Smith   | 2018-03-01  09:10:00 |  99.12
 2  |  Jenny  | Doe     |                      |
 3  |  Andy   | Jones   | 2018-03-02  08:02:15 |  13.5
 (4 rows)
```

확장 3: 별칭을 사용해 스칼라 배열에 직접 액세스

FROM 절의 별칭 p가 스칼라 배열을 포괄하는 경우에는 쿼리가 p 값을 p로 참조합니다. 예를 들어 다음은 고객 이름과 전화 번호를 쌍으로 반환하는 쿼리입니다.

```
SELECT c.name.given, c.name.family, p AS phone
FROM   spectrum.customers c LEFT JOIN c.phones p ON true
```

위의 쿼리는 다음과 같은 데이터를 반환합니다.

```
given  |  family  |  phone
-------|----------|-----------
John   | Smith    | 123-4577891
Jenny  | Doe      | 858-8675309
Jenny  | Doe      | 415-9876543
Andy   | Jones    |
(4 rows)
```

확장 4: 맵 요소에 대한 액세스

Redshift Spectrum은 map 데이터 형식을 array 열과 struct 열로 구성된 key 형식이 포함된 value 형식으로 처리합니다. key는 scalar가 되어야 하고, 값은 어떤 데이터 형식이든 될 수 있습니다.

예를 들어 다음은 전화 번호를 저장할 목적으로 map이 포함된 외부 테이블을 만드는 코드입니다.

```
CREATE EXTERNAL TABLE spectrum.customers2 (
  id     int,
  name   struct<given:varchar(20), family:varchar(20)>,
  phones map<varchar(20), varchar(20)>,
  orders array<struct<shipdate:timestamp, price:double precision>>
)
STORED AS PARQUET
LOCATION 's3://redshift-downloads/tickit/spectrum/customers/';
```

map 형식은 array 열과 key 열로 구성된 value 형식처럼 처리되기 때문에 앞에 나오는 스키마를 마치 뒤에 나오는 스키마라고 생각할 수 있습니다.

```
CREATE EXTERNAL TABLE spectrum.customers3 (
  id     int,
  name   struct<given:varchar(20), family:varchar(20)>,
  phones array<struct<key:varchar(20), value:varchar(20)>>,
  orders array<struct<shipdate:timestamp, price:double precision>>
)
STORED AS PARQUET
LOCATION 's3://redshift-downloads/tickit/spectrum/customers/';
```

다음은 고객 이름과 휴대 전화 번호를 함께 반환한 후 각 이름의 번호를 반환하는 쿼리입니다. 맵 쿼리는 array 형식의 중첩 struct를 쿼리하는 것과 동일하게 처리됩니다. 다음은 앞에서 설명한 외부 테이블을 만든 경우에만 데이터를 반환하는 쿼리입니다.

```
SELECT c.name.given, c.name.family, p.value
FROM   spectrum.customers c, c.phones p
WHERE  p.key = 'mobile';
```

> **Note**
>
> key에 대한 map는 Ion 및 JSON 파일 형식에 대한 string입니다.

중첩 데이터 사용 사례

이 주제에서는 중첩된 데이터의 사용 사례를 설명합니다. 중첩 데이터는 중첩된 필드를 포함하는 데이터입니다. 중첩 필드는 배열, 구조체 또는 객체와 같은 단일 엔터티로 함께 조인되는 필드입니다.

앞에서 설명한 확장을 일반적인 SQL 기능과 결합할 수 있습니다. 다음 사용 사례는 일반적인 몇 가지 결합에 대한 설명입니다. 각 사례는 중첩 데이터의 사용 방법을 이해하는 데 도움이 될 것입니다. 자습서에는 포함되어 있지 않습니다.

주제

중첩 데이터 수집

CREATE TABLE AS 문을 사용하여 복합 데이터 형식이 포함된 외부 테이블에서 데이터를 수집할 수 있습니다. 다음은 LEFT JOIN을 사용해 외부 테이블의 고객과 고객 전화 번호를 모두 추출한 후 Amazon Redshift 테이블인 CustomerPhones에 저장하는 쿼리입니다.

```
CREATE TABLE CustomerPhones AS
SELECT  c.name.given, c.name.family, p AS phone
FROM    spectrum.customers c LEFT JOIN c.phones p ON true;
```

하위 쿼리를 사용한 중첩 데이터 집계

하위 쿼리를 사용해 중첩 데이터를 집계할 수 있습니다. 다음은 이러한 방법을 설명하는 예입니다.

```
SELECT c.name.given, c.name.family, (SELECT COUNT(*) FROM c.orders o) AS ordercount
FROM   spectrum.customers c;
```

다음 데이터가 반환됩니다.

```
given    |  family   |  ordercount
---------|-----------|--------------
 Jenny   |  Doe      |       0
 John    |  Smith    |       2
 Andy    |  Jones    |       1
 (3 rows)
```

> **Note**
>
> 상위 행을 기준으로 그룹화하여 중첩 데이터를 집계할 때는 이전 예와 같은 방법이 가장 효율적입니다. 위의 예에서 중첩 행인 c.orders는 상위 행인 c를 기준으로 그룹화됩니다. 또는

id가 각 customer마다 고유하고, o.shipdate는 절대 NULL 값이 아니라는 사실을 알고 있다면 다음 예와 같이 집계하는 방법도 있습니다. 하지만 이 방법은 일반적으로 이전 예만큼 효율적이지 않습니다.

```
SELECT    c.name.given, c.name.family, COUNT(o.shipdate) AS ordercount
FROM      spectrum.customers c LEFT JOIN c.orders o ON true
GROUP BY  c.id, c.name.given, c.name.family;
```

또는 FROM 절에서 최상위 쿼리의 별칭(c)을 참조하여 배열 데이터를 추출하는 하위 쿼리를 사용하여 쿼리를 작성할 수도 있습니다. 다음 예에서는 이 방법을 보여 줍니다.

```
SELECT c.name.given, c.name.family, s.count AS ordercount
FROM   spectrum.customers c, (SELECT count(*) AS count FROM c.orders o) s;
```

Amazon Redshift 및 중첩 데이터 조인

Amazon Redshift 데이터를 외부 테이블의 중첩 데이터와 조인시킬 수 있습니다. 예를 들어 Amazon S3에 다음과 같은 중첩 데이터가 있다고 가정하겠습니다.

```
CREATE EXTERNAL TABLE spectrum.customers2 (
  id      int,
  name    struct<given:varchar(20), family:varchar(20)>,
  phones  array<varchar(20)>,
  orders  array<struct<shipdate:timestamp, item:int>>
);
```

또한 Amazon Redshift에 다음과 같은 테이블이 있다고 가정하겠습니다.

```
CREATE TABLE prices (
  id int,
  price double precision
);
```

다음은 앞의 예를 근거로 각 고객이 구매한 총 상품 수와 양을 요청하는 쿼리입니다. 다음은 이해를 돕기 위한 예입니다. 앞에서 설명한 테이블을 만든 경우에만 데이터를 반환합니다.

```
SELECT   c.name.given, c.name.family, COUNT(o.date) AS ordercount, SUM(p.price) AS
 ordersum
```

```
FROM     spectrum.customers2 c, c.orders o, prices p ON o.item = p.id
GROUP BY c.id, c.name.given, c.name.family;
```

중첩된 데이터 제한(미리 보기)

이 주제에서는 Redshift Spectrum을 사용하여 중첩된 데이터를 읽을 때의 제한 사항에 대해 설명합니다. 중첩 데이터는 중첩된 필드를 포함하는 데이터입니다. 중첩 필드는 배열, 구조체 또는 객체와 같은 단일 엔터티로 함께 조인되는 필드입니다.

> **Note**
>
> 다음 목록에서 (미리 보기)로 표시된 제한 사항은 다음 리전에서 생성된 미리 보기 클러스터 및 미리 보기 작업 그룹에만 적용됩니다.
>
> - 미국 동부(오하이오)(us-east-2)
> - 미국 동부(버지니아 북부)(us-east-1)
> - 미국 서부(캘리포니아 북부)(us-west-1)
> - 아시아 태평양(도쿄)(ap-northeast-1)
> - 유럽(아일랜드)(eu-west-1)
> - 유럽(스톡홀름)(eu-north-1)
>
> 미리 보기 클러스터 설정에 대한 내용은 Amazon Redshift 클러스터 관리 안내서의 미리 보기 클러스터 생성을 참조하세요. 미리 보기 작업 그룹을 설정하는 방법에 대한 내용은 Amazon Redshift 관리 안내서의 미리 보기 작업 그룹 만들기를 참조하세요.

중첩 데이터에는 다음과 같은 제약이 적용됩니다.

- `array` 또는 `map` 유형에는 쿼리가 중첩된 `arrays`에 있거나 `maps`가 `scalar` 값을 반환하지 않는 한 다른 `array` 또는 `map` 유형이 포함될 수 있습니다. (미리 보기)
- Amazon Redshift Spectrum은 외부 테이블의 형태로만 복합 데이터 유형을 지원합니다.
- 하위 쿼리 결과 열은 최상위 수준이어야 합니다. (미리 보기)
- `OUTER JOIN` 표현식이 중첩 테이블을 참조하는 경우에는 해당 테이블과 테이블의 중첩 배열(및 맵)만 참조할 수 있습니다. `OUTER JOIN` 표현식이 중첩 테이블을 참조하지 않는 경우에는 비중첩 테이블을 무제한 참조할 수 있습니다.
- 하위 쿼리의 `FROM` 절이 중첩 테이블을 참조하는 경우에는 다른 테이블을 참조할 수 없습니다.

- 하위 쿼리가 상위 테이블을 참조하는 중첩 테이블에 따라 달라지는 경우, 하위 쿼리는 FROM 절에서 상위 테이블만 사용할 수 있습니다. SELECT 또는 WHERE 절 같은 다른 절에서는 상위 쿼리를 사용할 수 없습니다. 예를 들어 다음 쿼리는 하위 쿼리의 SELECT 절이 상위 테이블 c를 참조하므로 실행되지 않습니다.

```
SELECT c.name.given
FROM   spectrum.customers c
WHERE (SELECT COUNT(c.id) FROM c.phones p WHERE p LIKE '858%') > 1;
```

  다음 쿼리는 상위 쿼리의 c가 하위 쿼리의 FROM 절에서만 사용되고 있기 때문에 유효합니다.

```
SELECT c.name.given
FROM   spectrum.customers c
WHERE (SELECT COUNT(*) FROM c.phones p WHERE p LIKE '858%') > 1;
```

- FROM 절이 아닌 다른 곳의 중첩 데이터에 액세스하는 하위 쿼리는 단일 값을 반환해야 합니다. WHERE 절의 (NOT) EXISTS 연산자만 예외입니다.
- (NOT) IN는 지원되지 않습니다.
- 모든 중첩 형식의 최대 중첩 깊이는 100입니다. 이러한 제약은 모든 파일 형식(Parquet, ORC, Ion, JSON)에 적용됩니다.
- 중첩 데이터에 액세스하는 집계 하위 쿼리는 FROM의 arrays 및 maps만 참조할 수 있고 외부 테이블은 참조할 수 없습니다.
- Redshift 스펙트럼 테이블에서 중첩된 데이터의 가성 열을 쿼리하는 것은 지원되지 않습니다. 자세한 내용은 가상 열 단원을 참조하십시오.
- 배열이나 맵 열을 FROM 절에 지정하여 배열이나 맵에서 데이터를 추출하는 경우 값이 scalar인 경우에만 해당 열에서 값을 선택할 수 있습니다. 예를 들어, 다음 쿼리는 모두 배열 내부의 요소에서 SELECT 작업 수행을 시도합니다. arr.a가 scalar 값이기 때문에 arr.a를 선택하는 쿼리가 작동합니다. 두 번째 쿼리는 array가 FROM 절의 s3.nested table에서 추출된 배열이기 때문에 작동하지 않습니다. (미리 보기)

```
SELECT array_column FROM s3.nested_table;

array_column
-----------------
[{"a":1},{"b":2}]

SELECT arr.a FROM s3.nested_table t, t.array_column arr;
```

```
arr.a
-----
1

--This query fails to run.
SELECT array FROM s3.nested_table tab, tab.array_column array;
```

다른 배열이나 맵에서 가져온 FROM 절에 있는 배열이나 맵은 사용할 수 없습니다. 다른 배열 내에 중첩된 배열 또는 기타 복합 구조를 선택하려면 SELECT 문에 인덱스를 사용하는 것이 좋습니다.

복잡한 중첩 JSON 직렬화

이 주제에서는 중첩된 데이터를 JSON 형식으로 직렬화하는 방법을 보여줍니다. 중첩 데이터는 중첩된 필드를 포함하는 데이터입니다. 중첩 필드는 배열, 구조체 또는 객체와 같은 단일 엔터티로 함께 조인되는 필드입니다.

이 튜토리얼에서 설명하는 방법의 대안으로 최상위 중첩 컬렉션 열을 직렬화된 JSON으로 쿼리할 수 있습니다. 직렬화를 사용하여 Redshift Spectrum에서 중첩 데이터를 JSON으로 검사, 변환 및 수집할 수 있습니다. 이 방법은 ORC, JSON, Ion 및 Parquet 형식에 대해 지원됩니다. 세션 구성 파라미터 `json_serialization_enable`을 사용하여 직렬화 동작을 구성합니다. 설정하면 복잡한 JSON 데이터 형식이 VARCHAR(65535)로 직렬화됩니다. 중첩 JSON은 JSON 함수로 액세스할 수 있습니다. 자세한 내용은 json_serialization_enable 단원을 참조하십시오.

예를 들어 `json_serialization_enable`을 설정하지 않으면 중첩 열에 직접 액세스하는 다음 쿼리가 실패합니다.

```
SELECT * FROM spectrum.customers LIMIT 1;

=> ERROR:  Nested tables do not support '*' in the SELECT clause.

SELECT name FROM spectrum.customers LIMIT 1;

=> ERROR:  column "name" does not exist in customers
```

`json_serialization_enable`을 설정하면 최상위 컬렉션을 직접 쿼리할 수 있습니다.

```
SET json_serialization_enable TO true;

SELECT * FROM spectrum.customers order by id LIMIT 1;
```

```
id | name                                    | phones         | orders
---+-----------------------------------------+----------------
+-----------------------------------------------------------------------------------------------------------------
1  | {"given": "John", "family": "Smith"} | ["123-457789"] | [{"shipdate":
 "2018-03-01T11:59:59.000Z", "price": 100.50}, {"shipdate": "2018-03-01T09:10:00.000Z",
 "price": 99.12}]

SELECT name FROM spectrum.customers order by id LIMIT 1;

name
---------
{"given": "John", "family": "Smith"}
```

중첩 JSON을 직렬화할 때 다음 항목을 고려합니다.

- 컬렉션 열이 VARCHAR(65535)로 직렬화되면 쿼리 구문의 일부로(예: 필터 절에서) 중첩 하위 필드에 직접 액세스할 수 없습니다. 그러나 JSON 함수를 사용하여 중첩 JSON에 액세스할 수 있습니다.
- 다음과 같은 특수 표현은 지원되지 않습니다.
  - ORC 조합
  - 복합 형식 키가 있는 ORC 맵
  - Ion 데이터그램
  - Ion SEXP
- 타임스탬프는 ISO 직렬화 문자열로 반환됩니다.
- 기본 맵 키는 문자열로 승격됩니다(예: 1에서 "1"로).
- 최상위 null 값은 NULL로 직렬화됩니다.
- 직렬화가 최대 VARCHAR 크기인 65535를 넘으면 셀이 NULL로 설정됩니다.

JSON 문자열을 포함하는 복합 형식 직렬화

기본적으로 중첩 컬렉션에 포함된 문자열 값은 이스케이프 처리된 JSON 문자열로 직렬화됩니다. 문자열이 유효한 JSON인 경우 이스케이프가 바람직하지 않을 수 있습니다. 대신 JSON으로 직접 VARCHAR인 중첩 하위 요소 또는 필드를 작성할 수 있습니다. `json_serialization_parse_nested_strings` 세션 수준 구성으로 이 동작을 사용합니다. `json_serialization_enable`과 `json_serialization_parse_nested_strings`가 모두 설정되면 유효한 JSON 값이 이스케이프 문자 없이 인라인으로 직렬화됩니다. 값이 유효한 JSON이 아닌 경우 `json_serialization_parse_nested_strings` 구성 값이 설정되지 않은 것처럼 이스케이프 처리됩니다. 자세한 내용은 json_serialization_parse_nested_strings 단원을 참조하십시오.

예를 들어 이전 예의 데이터에서 name VARCHAR(20) 필드에 structs 복합 형식으로 JSON이 포함되어 있다고 가정합니다.

```
name
---------
{"given": "{\"first\":\"John\",\"middle\":\"James\"}", "family": "Smith"}
```

json_serialization_parse_nested_strings가 설정되면 name 열은 다음과 같이 직렬화됩니다.

```
SET json_serialization_enable TO true;
SET json_serialization_parse_nested_strings TO true;
SELECT name FROM spectrum.customers order by id LIMIT 1;

name
---------
{"given": {"first":"John","middle":"James"}, "family": "Smith"}
```

다음과 같이 이스케이프 처리되는 대신

```
SET json_serialization_enable TO true;
SELECT name FROM spectrum.customers order by id LIMIT 1;

name
---------
{"given": "{\"first\":\"John\",\"middle\":\"James\"}", "family": "Smith"}
```

HyperLogLog 스케치

이 주제에서는 Amazon Redshift에서 HyperLogLog 스케치를 사용하는 방법을 설명합니다. HyperLogLog는 데이터세트의 고유한 요소 수에 근접하는 카운트 구분 문제를 위한 알고리즘입니다. HyperLogLog 스케치는 데이터세트에 대한 고유 데이터의 배열입니다.

HyperLogLog는 다중 집합의 카디널리티를 추정하는 데 사용되는 알고리즘입니다. 카디널리티는 다중 집합에서 고유한 값의 수를 나타냅니다. 예를 들어 {4,3,6,2,2,6,4,3,6,2,2,3} 집합에서 카디널리티는 4, 3, 6, 2의 고유한 값을 갖는 4입니다.

HyperLogLog 알고리즘의 정밀도(m 값이라고도 함)는 예상 카디널리티의 정확도에 영향을 줄 수 있습니다. 카디널리티 추정 중 Amazon Redshift는 기본 정밀도 값 15를 사용합니다. 이 값은 더 작은 데이터 집합의 경우 최대 26일 수 있습니다. 따라서 평균 상대 오차 범위는 0.01~0.6%입니다.

다중 집합의 카디널리티를 계산할 때 HyperLogLog 알고리즘은 HLL 스케치라는 구문을 생성합니다. HLL 스케치는 다중 집합의 고유 값에 대한 정보를 캡슐화합니다. Amazon Redshift 데이터 형식 HLLSKETCH는 이러한 스케치 값을 나타냅니다. 이 데이터 형식은 Amazon Redshift 테이블에 스케치를 저장하는 데 사용할 수 있습니다. 또한 Amazon Redshift는 HLLSKETCH 값에 집계 및 스칼라 함수로 적용할 수 있는 작업을 지원합니다. 이러한 함수를 사용하여 HLLSKETCH의 카디널리티를 추출하고 여러 HLLSKETCH 값을 결합할 수 있습니다.

HLLSKETCH 데이터 형식은 큰 데이터 집합에서 카디널리티를 추출할 때 상당한 쿼리 성능 이점을 제공합니다. HLLSKETCH 값을 사용하여 이러한 데이터 집합을 사전 집계하고 테이블에 저장할 수 있습니다. Amazon Redshift는 기본 데이터 집합에 액세스하지 않고 저장된 HLLSKETCH 값에서 직접 카디널리티를 추출할 수 있습니다.

HLL 스케치를 처리할 때 Amazon Redshift는 스케치의 메모리 공간을 최소화하고 추출된 카디널리티의 정밀도를 최대화하는 최적화를 수행합니다. Amazon Redshift는 HLL 스케치에 대해 희소 및 밀집의 두 가지 표현을 사용합니다. HLLSKETCH는 희소 형식으로 시작합니다. 새 값이 삽입되면 크기가 증가합니다. 크기가 밀집 표현의 크기에 도달하면 Amazon Redshift가 자동으로 스케치를 희소에서 밀집으로 변환합니다.

Amazon Redshift는 스케치가 희소 형식일 때 HLLSKETCH를 JSON으로 가져오고 내보내고 인쇄합니다. Amazon Redshift는 스케치가 밀집 형식일 때 HLLSKETCH를 Base64 문자열로 가져오고 내보내고 인쇄합니다. UNLOAD에 대한 자세한 내용은 HLLSKETCH 데이터 형식 언로드 섹션을 참조하세요. 텍스트 또는 쉼표로 구분된 값(CSV) 데이터를 Amazon Redshift로 가져오려면 COPY 명령을 사용합니다. 자세한 내용은 HLLSKETCH 데이터 형식 로드 단원을 참조하십시오.

HyperLogLog와 함께 사용되는 함수에 대한 자세한 내용은 HyperLogLog 함수 섹션을 참조하세요.

주제

고려 사항

이 주제에서는 Amazon Redshift에서 HyperLogLog의 사용 세부 정보를 설명합니다.

다음은 Amazon Redshift에서 HyperLogLog를 사용하기 위한 고려 사항입니다.

- 다음 비 HyperLogLog 함수는 HLLSKETCH 형식의 입력 또는 HLLSKETCH 형식의 열을 사용할 수 있습니다.
  - 집계 함수 COUNT
  - 조건부 표현식 COALESCE 및 NVL
  - CASE 표현식
- 지원되는 인코딩은 RAW입니다.
- HLLSKETCH 열이 있는 테이블에서 텍스트 또는 CSV로 UNLOAD 작업을 수행할 수 있습니다. UNLOAD HLLSKETCH 열을 사용하여 HLLSKETCH 데이터를 쓸 수 있습니다. Amazon Redshift는 희소 표현의 경우 JSON 형식 또는 밀집 표현의 경우 Base64 형식으로 데이터를 표시합니다. UNLOAD에 대한 자세한 내용은 HLLSKETCH 데이터 형식 언로드 섹션을 참조하세요.

  다음은 JSON 형식으로 표현된 희소 HyperLogLog 스케치에 사용되는 형식을 보여줍니다.

  ```
  {"version":1,"logm":15,"sparse":{"indices":
  [15099259,33107846,37891580,50065963],"values":[2,3,2,1]}}
  ```
- COPY 명령을 사용하여 텍스트 또는 CSV 데이터를 Amazon Redshift로 가져올 수 있습니다. 자세한 내용은 HLLSKETCH 데이터 형식 로드 단원을 참조하십시오.
- HLLSKETCH의 기본 인코딩은 RAW입니다. 자세한 내용은 압축 인코딩 단원을 참조하십시오.

제한 사항

이 주제에서는 Amazon Redshift에서 HyperLogLog의 제한 사항에 대해 설명합니다.

다음은 Amazon Redshift에서 HyperLogLog를 사용하기 위한 제한 사항입니다.

- Amazon Redshift 테이블은 HLLSKETCH 열을 Amazon Redshift 테이블의 정렬 키 또는 배포 키로 지원하지 않습니다.
- Amazon Redshift는 ORDER BY, GROUP BY 또는 DISTINCT 절에서 HLLSKETCH 열을 지원하지 않습니다.
- HLLSKETCH 열을 텍스트 또는 CSV 형식으로만 UNLOAD할 수 있습니다. 그런 다음 Amazon Redshift는 HLLSKETCH 데이터를 JSON 형식 또는 Base64 형식으로 씁니다. UNLOAD에 대한 자세한 내용은 UNLOAD 섹션을 참조하세요.
- Amazon Redshift는 정밀도(logm 값)가 15인 HyperLogLog 스케치만 지원합니다.
- JDBC 및 ODBC 드라이버는 HLLSKETCH 데이터 형식을 지원하지 않습니다. 따라서 결과 집합에서는 VARCHAR를 사용하여 HLLSKETCH 값을 나타냅니다.
- Amazon Redshift Spectrum은 기본적으로 HLLSKETCH 데이터를 지원하지 않습니다. 따라서 HLLSKETCH 열이 있는 외부 테이블을 생성하거나 변경할 수 없습니다.
- Python 사용자 정의 함수(UDF)의 데이터 형식은 HLLSKETCH 데이터 형식을 지원하지 않습니다. Python UDF에 대한 자세한 내용은 Scalar Python UDF 섹션을 참조하세요.

예시

이 섹션에는 Amazon Redshift에서 HyperLogLog를 사용하는 예제가 포함되어 있습니다.

주제

- 예: 하위 쿼리에서 카디널리티 반환
- 예: 하위 쿼리의 결합된 스케치에서 HLLSKETCH 형식 반환
- 예: 여러 스케치를 결합하여 HyperLogLog 스케치 반환
- 예: 외부 테이블을 사용하여 S3 데이터에 대한 HyperLogLog 스케치 생성

예: 하위 쿼리에서 카디널리티 반환

다음 예에서는 Sales라는 테이블에 대한 하위 쿼리의 각 스케치에 대한 카디널리티를 반환합니다.

```
CREATE TABLE Sales (customer VARCHAR, country VARCHAR, amount BIGINT);
INSERT INTO Sales VALUES ('David Joe', 'Greece', 14.5),  ('David Joe', 'Greece',
 19.95), ('John Doe', 'USA', 29.95), ('John Doe', 'USA', 19.95), ('George Spanos',
 'Greece', 9.95), ('George Spanos', 'Greece', 2.95);
```

다음 쿼리는 각 국가의 고객에 대한 HLL 스케치를 생성하고 카디널리티를 추출합니다. 이는 각 국가의 고유 고객을 보여줍니다.

```
SELECT hll_cardinality(sketch), country
FROM (SELECT hll_create_sketch(customer) AS sketch, country
        FROM Sales
        GROUP BY country) AS hll_subquery;

hll_cardinality | country
----------------+---------
            1   | USA
            2   | Greece
 ...
```

예: 하위 쿼리의 결합된 스케치에서 HLLSKETCH 형식 반환

다음 예에서는 하위 쿼리의 개별 스케치 조합을 나타내는 단일 HLLSKETCH 형식을 반환합니다. 스케치는 HLL_COMBINE 집계 함수를 사용하여 결합됩니다.

```
SELECT hll_combine(sketch)
FROM (SELECT hll_create_sketch(customer) AS sketch
        FROM Sales
        GROUP BY country) AS hll_subquery

                                        hll_combine
-----------------------------------------------------------------------------------------------
 {"version":1,"logm":15,"sparse":{"indices":[29808639,35021072,47612452],"values":
[1,1,1]}}
(1 row)
```

예: 여러 스케치를 결합하여 HyperLogLog 스케치 반환

다음 예의 경우 테이블 page-users에 사용자가 지정된 웹 사이트에서 방문한 각 페이지에 대해 사전 집계된 스케치가 저장되어 있다고 가정합니다. 이 테이블의 각 행에는 방문한 페이지를 표시하는 모든 사용자 ID를 나타내는 HyperLogLog 스케치가 포함되어 있습니다.

```
page_users
-- +----------------+-------------+--------------+
-- | _PARTITIONTIME | page         | sketch |
-- +----------------+-------------+--------------+
```

```
-- | 2019-07-28     | homepage     | CHAQkAQYA... |
-- | 2019-07-28     | Product A    | CHAQxPnYB... |
-- +----------------+--------------+--------------+
```

다음 예에서는 미리 집계된 여러 스케치를 결합하고 단일 스케치를 생성합니다. 이 스케치는 각 스케치가 캡슐화하는 집합적 카디널리티를 캡슐화합니다.

```
SELECT hll_combine(sketch) as sketch
FROM page_users
```

출력 결과는 다음과 비슷합니다.

```
-- +----------------------------------------+
-- | sketch |
-- +----------------------------------------+
-- | CHAQ3sGoCxgCIAuCB4iAIBgTIBgqgIAgAwY.... |
-- +----------------------------------------+
```

새 스케치가 생성되면 다음과 같이 HLL_CARDINALITY 함수를 사용하여 집합적 고유 값을 얻을 수 있습니다.

```
SELECT hll_cardinality(sketch)
FROM (
  SELECT
  hll_combine(sketch) as sketch
  FROM page_users
) AS hll_subquery
```

출력 결과는 다음과 비슷합니다.

```
-- +-------+
-- | count |
-- +-------+
-- | 54356 |
-- +-------+
```

예: 외부 테이블을 사용하여 S3 데이터에 대한 HyperLogLog 스케치 생성

다음 예에서는 카디널리티 추정을 위해 Amazon S3 직접 액세스하지 않도록 HyperLogLog 스케치를 캐시합니다.

Amazon S3 데이터를 보유하도록 정의된 외부 테이블에서 HyperLogLog 스케치를 사전 집계하고 캐시할 수 있습니다. 이렇게 하면 기본 데이터에 액세스하지 않고도 카디널리티 추정을 추출할 수 있습니다.

예를 들어 탭으로 구분된 텍스트 파일 집합을 Amazon S3로 언로드했다고 가정합니다. 다음 쿼리를 실행하여 `spectrum`이라는 Amazon Redshift 외부 스키마에서 `sales`라는 외부 테이블을 정의합니다. 이 예에서 Amazon S3 버킷은 미국 동부(버지니아 북부) AWS 리전에 있습니다.

```
create external table spectrum.sales(
salesid integer,
listid integer,
sellerid smallint,
buyerid smallint,
eventid integer,
dateid integer,
qtysold integer,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
row format delimited
fields terminated by '\t' stored as textfile
location 's3://redshift-downloads/tickit/spectrum/sales/';
```

임의의 날짜에 항목을 구매한 개별 구매자를 계산한다고 가정합니다. 이를 위해 다음 예에서는 연중 매일 구매자 ID에 대한 스케치를 생성하고 Amazon Redshift 테이블 `hll_sales`에 결과를 저장합니다.

```
CREATE TABLE hll_sales AS
SELECT saletime, hll_create_sketch(buyerid) AS sketch
FROM spectrum.sales
GROUP BY saletime;

SELECT TOP 5 * FROM hll_sales;
```

출력 결과는 다음과 비슷합니다.

```
-- hll_sales

-- | saletime        | sketch
    |
-- +-----------------
+-----------------------------------------------------------------------+
```

```
-- | 7/22/2008 8:30  | {"version":1,"logm":15,"sparse":{"indices":[9281416],"values":
[1]}}
-- | 2/19/2008 0:38  | {"version":1,"logm":15,"sparse":{"indices":[48735497],"values":
[3]}}
-- | 11/5/2008 4:49  | {"version":1,"logm":15,"sparse":{"indices":[27858661],"values":
[1]}}
-- | 10/27/2008 4:08 | {"version":1,"logm":15,"sparse":{"indices":[65295430],"values":
[2]}}
-- | 2/16/2008 9:37  | {"version":1,"logm":15,"sparse":{"indices":[56869618],"values":
[2]}}
-- +----------------
 +-------------------------------------------------------------------------+
```

다음 쿼리는 2008년 추수 감사절 후 금요일에 항목을 구매한 예상 개별 구매자 수를 보여줍니다.

```
SELECT hll_cardinality(hll_combine(sketch)) as distinct_buyers
FROM hll_sales
WHERE trunc(saletime) = '2008-11-28';
```

출력 결과는 다음과 비슷합니다.

```
distinct_buyers
---------------
386
```

특정 날짜 범위에 항목을 구매한 고유한 사용자 수를 원한다고 가정합니다. 추수 감사절 후 금요일부터 다음 월요일까지를 예로 들 수 있습니다. 이를 얻기 위해 다음 쿼리는 `hll_combine` 집계 함수를 사용합니다. 이 함수를 사용하면 선택한 범위의 하루 이상 항목을 구매한 구매자가 이중으로 계산되는 것을 방지할 수 있습니다.

```
SELECT hll_cardinality(hll_combine(sketch)) as distinct_buyers
FROM hll_sales
WHERE saletime BETWEEN '2008-11-28' AND '2008-12-01';
```

출력 결과는 다음과 비슷합니다.

```
distinct_buyers
---------------
1166
```

hll_sales 테이블을 최신 상태로 유지하려면 하루가 끝날 때 다음 쿼리를 실행합니다. 이렇게 하면 오늘 항목을 구매한 구매자의 ID를 기반으로 HyperLogLog 스케치가 생성되어 hll_sales 테이블에 추가됩니다.

```
INSERT INTO hll_sales
SELECT saletime, hll_create_sketch(buyerid)
FROM spectrum.sales
WHERE TRUNC(saletime) = to_char(GETDATE(), 'YYYY-MM-DD')
GROUP BY saletime;
```

데이터베이스 간 쿼리

이 주제에서는 단일 Amazon Redshift 클러스터 내의 여러 Amazon Redshift 데이터베이스에서 작동하는 쿼리인 데이터베이스 간 쿼리에 대해 설명합니다.

Amazon Redshift에서 데이터베이스 간 쿼리를 사용하여 Amazon Redshift 클러스터의 데이터베이스 간에 쿼리할 수 있습니다. 데이터베이스 간 쿼리를 사용하면 연결된 데이터베이스와 관계없이 Amazon Redshift 클러스터의 모든 데이터베이스에서 데이터를 쿼리하고 쓸 수 있습니다. 데이터베이스 간 쿼리는 데이터 복사본을 제거하고 데이터 조직을 단순화하여 동일한 데이터 웨어하우스의 여러 비즈니스 그룹을 지원합니다.

데이터베이스 간 쿼리를 사용하여 다음을 수행할 수 있습니다.

- Amazon Redshift 클러스터의 데이터베이스 간 데이터 쿼리.

  연결된 데이터베이스에서 쿼리할 수 있을 뿐만 아니라 권한이 있는 다른 데이터베이스에서도 읽을 수 있습니다.

  연결되지 않은 다른 데이터베이스에서 데이터베이스 객체를 쿼리할 때 해당 데이터베이스 객체에 대한 읽기 권한만 있습니다. 특정 데이터베이스에 연결할 필요 없이 데이터베이스 간 쿼리를 사용하여 Amazon Redshift 클러스터에 있는 모든 데이터베이스의 데이터에 액세스할 수 있습니다. 이렇게 하면 Amazon Redshift 클러스터의 여러 데이터베이스에 분산된 데이터를 빠르고 쉽게 쿼리하고 조인할 수 있습니다.

  또한 단일 쿼리로 여러 데이터베이스의 데이터 집합을 조인하고 비즈니스 인텔리전스(BI) 또는 분석 도구를 사용하여 데이터를 분석할 수 있습니다. 표준 Amazon Redshift SQL 명령으로 사용자에 대한 세분화된 테이블 수준 읽기 액세스 제어를 계속 설정할 수 있습니다. 이를 통해 사용자에게 권한이 있는 관련 데이터 하위 집합만 표시되게 할 수 있습니다.
- Amazon Redshift 클러스터의 데이터베이스 간에 데이터를 씁니다.

  연결된 데이터베이스에서 쓸 수 있고 권한이 있는 다른 데이터베이스에서도 쓸 수 있습니다.

  연결되지 않은 다른 데이터베이스의 데이터베이스 객체에 대한 쓰기 권한이 있는 경우, 데이터베이스 간 쿼리를 사용하여 특정 데이터베이스에 연결할 필요 없이 Amazon Redshift 클러스터의 원하는 데이터베이스에서 데이터를 쓸 수 있습니다. 이렇게 하면 Amazon Redshift 클러스터의 여러 데이터베이스에 분산된 데이터를 빠르고 쉽게 조인하면서 복잡한 쓰기 작업을 수행하는 데 도움이 될 수 있습니다.

또한 단일 쿼리로 여러 데이터베이스의 데이터세트를 조인하고 다양한 추출-전환-적재(ETL) 또는 분석 도구를 사용하여 데이터를 쓸 수 있습니다. 표준 Amazon Redshift SQL 명령으로 사용자에 대한 세분화된 테이블 수준 쓰기 액세스 제어를 계속 설정할 수 있습니다. 이렇게 하면 사용자는 권한이 있는 관련 데이터 하위 집합만 볼 수 있습니다.

- 객체 쿼리.

  세 부분으로 구성된 표기법으로 표현된 완전한 객체 이름을 사용하여 다른 데이터베이스 객체를 쿼리할 수 있습니다. 데이터베이스 객체의 전체 경로는 데이터베이스 이름, 스키마 및 객체 이름의 세 가지 구성 요소로 이루어집니다. 전체 경로 표기법 *database_name*.*schema_name*.*object_name*을 사용하여 다른 데이터베이스의 객체에 액세스할 수 있습니다. 특정 열에 액세스하려면 *database_name*.*schema_name*.*object_name*.*column_name*을 사용합니다.

  외부 스키마 표기법을 사용하여 다른 데이터베이스의 스키마에 대한 별칭을 생성할 수도 있습니다. 이 외부 스키마는 다른 데이터베이스와 스키마 페어를 참조합니다. 쿼리는 외부 스키마 표기법 *external_schema_name*.*object_name*을 사용하여 다른 데이터베이스 객체에 액세스할 수 있습니다.

  동일한 읽기 전용 쿼리에서 사용자 테이블, 일반 뷰, 구체화된 뷰 및 다른 데이터베이스의 후기 바인딩 뷰와 같은 다양한 데이터베이스 객체를 쿼리할 수 있습니다.

- 권한 관리.

  Amazon Redshift 클러스터의 모든 데이터베이스에 있는 객체에 대한 액세스 권한이 있는 사용자는 해당 객체를 쿼리하고 해당 객체에 데이터를 쓸 수 있습니다. GRANT 명령을 사용하여 사용자와 사용자 그룹에 권한을 부여합니다. 사용자가 특정 데이터베이스 객체에 더 이상 액세스할 필요가 없을 때 REVOKE 명령을 사용하여 권한을 취소할 수도 있습니다.

- 메타데이터와 BI 도구 사용.

  동일한 Amazon Redshift 클러스터 내 다른 Amazon Redshift 데이터베이스의 스키마를 참조하기 위해 외부 스키마를 생성할 수 있습니다. 자세한 내용은 CREATE EXTERNAL SCHEMA 명령을 참조하세요.

  외부 스키마 참조가 생성된 후 Amazon Redshift는 도구가 메타데이터를 탐색할 수 있도록 다른 데이터베이스의 스키마 아래에 있는 테이블을 SVV_EXTERNAL_TABLES 및 SVV_EXTERNAL_COLUMNS에 표시합니다.

데이터베이스 간 쿼리를 BI 도구와 통합하기 위해 다음 시스템 뷰를 사용할 수 있습니다. 이를 통해 Amazon Redshift 클러스터의 연결된 데이터베이스와 기타 데이터베이스에 있는 객체의 메타데이터에 대한 정보를 볼 수 있습니다.

다음은 Amazon Redshift 클러스터에 있는 모든 데이터베이스의 모든 Amazon Redshift 객체 및 외부 객체를 표시하는 시스템 뷰입니다.

- SVV_ALL_COLUMNS
- SVV_ALL_SCHEMAS
- SVV_ALL_TABLES

다음은 Amazon Redshift 클러스터에 있는 모든 데이터베이스의 모든 Amazon Redshift 객체를 표시하는 시스템 뷰입니다.

- SVV_REDSHIFT_COLUMNS
- SVV_REDSHIFT_DATABASES
- SVV_REDSHIFT_FUNCTIONS
- SVV_REDSHIFT_SCHEMAS
- SVV_REDSHIFT_TABLES

주제

고려 사항

이 주제에서는 Amazon Redshift에서 데이터베이스 간 쿼리에 대한 사용 세부 정보를 설명합니다.

Amazon Redshift에서 데이터베이스 간 쿼리 기능을 사용할 때 다음 사항을 고려합니다.

- Amazon Redshift는 모든 ra3 노드 유형 및 서버리스 네임스페이스에서 데이터베이스 간 쿼리를 지원합니다.
- Amazon Redshift는 동일한 Amazon Redshift 클러스터에 있는 하나 이상의 데이터베이스에서 테이블 또는 뷰의 데이터 조인을 지원합니다.

- 연결된 데이터베이스의 트랜잭션에 있는 모든 쿼리는 트랜잭션이 시작될 때의 데이터와 동일한 상태의 다른 데이터베이스에서 데이터를 읽습니다. 이 접근 방식은 데이터베이스 간에 쿼리 트랜잭션 일관성을 제공하는 데 도움이 됩니다. Amazon Redshift는 데이터베이스 간 쿼리에 대한 트랜잭션 일관성을 지원합니다.
- 데이터베이스 전체에서 메타데이터를 가져오려면 SVV_ALL* 및 SVV_REDSHIFT* 메타데이터 보기를 사용합니다. information_schema 및 pg_catalog에서 데이터베이스 간 메타데이터 테이블 또는 보기를 쿼리하기 위해 세 부분으로 된 표기법 또는 외부 스키마를 사용할 수 없습니다.

제한 사항

이 주제에서는 Amazon Redshift에서 데이터베이스 간 쿼리에 대한 제한 사항을 설명합니다.

Amazon Redshift에서 데이터베이스 간 쿼리 기능을 사용하는 경우 다음 제한 사항에 유의합니다.

- 다른 데이터베이스의 객체를 참조하는 다른 데이터베이스에서 생성된 뷰는 쿼리할 수 없습니다.
- 연결되지 않은 다른 데이터베이스의 데이터베이스 객체에는 외부 스키마를 쓸 수 없습니다.
- 클러스터에 있는 다른 데이터베이스의 객체에 대한 후기 바인딩과 구체화된 뷰만 생성할 수 있습니다. 클러스터에 있는 다른 데이터베이스의 객체에 대한 일반 뷰를 생성할 수 없습니다.
- Amazon Redshift는 데이터베이스 간 쿼리에 대해 열 수준 권한이 있는 테이블을 지원하지 않습니다.
- 인터리브 정렬 키가 있는 테이블에서는 데이터베이스 간 쿼리를 실행할 수 없습니다.

데이터베이스 간 쿼리 예제

이 주제에는 데이터베이스 간 쿼리 사용 방법에 관한 예제가 포함되어 있습니다. 데이터베이스 간 쿼리는 단일 Amazon Redshift 클러스터 내의 여러 데이터베이스에서 작동하는 쿼리입니다.

다음 예를 사용하여 Amazon Redshift 데이터베이스를 참조하는 데이터베이스 간 쿼리를 설정하는 방법에 대해 알아봅니다.

시작하려면 Amazon Redshift 클러스터에서 데이터베이스 db1 및 db2와 사용자 user1 및 user2를 생성합니다. 자세한 내용은 데이터베이스 생성 및 사용자 생성 섹션을 참조하세요.

```
--As user1 on db1
CREATE DATABASE db1;

CREATE DATABASE db2;
```

```
CREATE USER user1 PASSWORD 'Redshift01';

CREATE USER user2 PASSWORD 'Redshift01';
```

db1의 user1과 같이 테이블을 생성하고 user2에게 액세스 권한을 부여하고 table1에 값을 삽입합니다. 자세한 내용은 GRANT 및 INSERT 섹션을 참조하세요.

```
--As user1 on db1
CREATE TABLE table1 (c1 int, c2 int, c3 int);

GRANT SELECT ON table1 TO user2;

INSERT INTO table1 VALUES (1,2,3),(4,5,6),(7,8,9);
```

db2의 user2와 같이 세 부분으로 구성된 표기법을 사용하여 db2에서 데이터베이스 간 쿼리를 실행합니다.

```
--As user2 on db2
SELECT * from db1.public.table1 ORDER BY c1;
c1 | c2  | c3
---+-----+----
1  |  2  | 3
4  |  5  | 6
7  |  8  | 9
(3 rows)
```

db2의 user2와 같이 외부 스키마를 생성하고 외부 스키마 표기법을 사용하여 db2에서 데이터베이스 간 쿼리를 실행합니다.

```
--As user2 on db2
CREATE EXTERNAL SCHEMA db1_public_sch
FROM REDSHIFT DATABASE 'db1' SCHEMA 'public';

SELECT * FROM db1_public_sch.table1 ORDER BY c1;

c1  | c2 | c3
----+----+----
1   | 2  | 3
4   | 5  | 6
7   | 8  | 9
```

```
(3 rows)
```

다른 뷰를 생성하고 해당 뷰에 권한을 부여하려면 db1의 user1과 같이 다음을 수행합니다.

```
--As user1 on db1
CREATE VIEW regular_view AS SELECT c1 FROM table1;

GRANT SELECT ON regular_view TO user2;


CREATE MATERIALIZED VIEW mat_view AS SELECT c2 FROM table1;

GRANT SELECT ON mat_view TO user2;


CREATE VIEW late_bind_view AS SELECT c3 FROM public.table1 WITH NO SCHEMA BINDING;

GRANT SELECT ON late_bind_view TO user2;
```

db2의 user2와 같이 세 부분으로 구성된 표기법으로 다음 데이터베이스 간 쿼리를 실행하여 특정 뷰를 봅니다.

```
--As user2 on db2
SELECT * FROM db1.public.regular_view;
c1
----
1
4
7
(3 rows)

SELECT * FROM db1.public.mat_view;
c2
----
8
5
2
(3 rows)

SELECT * FROM db1.public.late_bind_view;
c3
----
3
```

```
6
9
(3 rows)
```

db2의 user2와 같이 외부 스키마 표기법으로 다음 데이터베이스 간 쿼리를 실행하여 후기 바인딩 뷰를 쿼리합니다.

```
--As user2 on db2
SELECT * FROM db1_public_sch.late_bind_view;
c3
----
3
6
9
(3 rows)
```

db2의 user2와 같이 단일 쿼리에서 연결된 테이블을 사용하여 다음 명령을 실행합니다.

```
--As user2 on db2
CREATE TABLE table1 (a int, b int, c int);

INSERT INTO table1 VALUES (1,2,3), (4,5,6), (7,8,9);

SELECT a AS col_1, (db1.public.table1.c2 + b) AS sum_col2, (db1.public.table1.c3 + c)
 AS sum_col3 FROM db1.public.table1, table1 WHERE db1.public.table1.c1 = a;
col_1 | sum_col2 | sum_col3
------+----------+----------
1     | 4        | 6
4     | 10       | 12
7     | 16       | 18
(3 rows)
```

다음 예에서는 클러스터에 있는 모든 데이터베이스를 나열합니다.

```
select database_name, database_owner, database_type
from svv_redshift_databases
where database_name in ('db1', 'db2');

 database_name | database_owner | database_type
---------------+----------------+---------------
 db1           |            100 | local
 db2           |            100 | local
```

```
(2 rows)
```

다음 예에서는 클러스터에 있는 모든 데이터베이스의 모든 Amazon Redshift 스키마를 나열합니다.

```
select database_name, schema_name, schema_owner, schema_type
from svv_redshift_schemas
where database_name in ('db1', 'db2');

 database_name |    schema_name     | schema_owner | schema_type
---------------+--------------------+--------------+-------------
 db1           | pg_catalog         |            1 | local
 db1           | public             |            1 | local
 db1           | information_schema |            1 | local
 db2           | pg_catalog         |            1 | local
 db2           | public             |            1 | local
 db2           | information_schema |            1 | local
(6 rows)
```

다음 예에서는 클러스터에 있는 모든 데이터베이스의 모든 Amazon Redshift 테이블 또는 뷰를 나열합니다.

```
select database_name, schema_name, table_name, table_type
from svv_redshift_tables
where database_name in ('db1', 'db2') and schema_name in ('public');

 database_name | schema_name |     table_name      | table_type
---------------+-------------+---------------------+------------
 db1           | public      | late_bind_view      | VIEW
 db1           | public      | mat_view            | VIEW
 db1           | public      | mv_tbl__mat_view__0 | TABLE
 db1           | public      | regular_view        | VIEW
 db1           | public      | table1              | TABLE
 db2           | public      | table2              | TABLE
(6 rows)
```

다음 예에서는 클러스터에 있는 모든 데이터베이스의 모든 Amazon Redshift 및 외부 스키마를 나열합니다.

```
select database_name, schema_name, schema_owner, schema_type
from svv_all_schemas where database_name in ('db1', 'db2') ;
```

```
 database_name |    schema_name     | schema_owner | schema_type
---------------+--------------------+--------------+-------------
 db1           | pg_catalog         |            1 | local
 db1           | public             |            1 | local
 db1           | information_schema |            1 | local
 db2           | pg_catalog         |            1 | local
 db2           | public             |            1 | local
 db2           | information_schema |            1 | local
 db2           | db1_public_sch     |            1 | external
(7 rows)
```

다음 예에서는 클러스터에 있는 모든 데이터베이스의 모든 Amazon Redshift 및 외부 테이블을 나열합니다.

```
select database_name, schema_name, table_name, table_type
from svv_all_tables
where database_name in ('db1', 'db2') and schema_name in ('public');

 database_name | schema_name |     table_name      | table_type
---------------+-------------+---------------------+------------
 db1           | public      | regular_view        | VIEW
 db1           | public      | mv_tbl__mat_view__0 | TABLE
 db1           | public      | mat_view            | VIEW
 db1           | public      | late_bind_view      | VIEW
 db1           | public      | table1              | TABLE
 db2           | public      | table2              | TABLE
(6 rows)
```

쿼리 에디터에서 데이터베이스 간 쿼리 사용

이 주제에서는 Query Editor에서의 데이터베이스 간 쿼리 사용 방법을 설명합니다. 데이터베이스 간 쿼리는 단일 Amazon Redshift 클러스터 내의 여러 데이터베이스에서 작동하는 쿼리입니다.

특정 데이터베이스에 연결할 필요 없이 데이터베이스 간 쿼리를 사용하여 Amazon Redshift 클러스터에 있는 모든 데이터베이스의 데이터에 액세스할 수 있습니다. 연결되지 않은 다른 데이터베이스에서 데이터베이스 간 쿼리를 실행할 때 해당 데이터베이스 객체에 대해서만 읽기 및 쓰기이 있습니다.

세 부분으로 구성된 표기법으로 표현된 완전한 객체 이름을 사용하여 다른 데이터베이스 객체를 쿼리하고 쓸 수 있습니다. 데이터베이스 객체의 전체 경로는 데이터베이스 이름, 스키마 및 객체 이름의 세 가지 구성 요소로 이루어집니다. 예를 들면, *database_name.schema_name.object_name*입니다.

쿼리 에디터 v2에서 데이터베이스 간 쿼리를 사용하는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. Amazon Redshift 쿼리 에디터 v2에서 데이터베이스 간 쿼리를 사용할 클러스터를 생성합니다. 자세한 내용은 Amazon Redshift 관리 가이드의 클러스터 생성 섹션을 참조하세요.
3. 적절한 권한으로 쿼리 에디터에 대한 액세스를 사용하도록 설정합니다. 자세한 정보는 Amazon Redshift 관리 안내서의 Querying a database using the query editor v2를 참조하세요.
4. 탐색 메뉴에서 쿼리 에디터 v2를 선택한 다음 클러스터의 데이터베이스에 연결합니다.

   쿼리 에디터 v2에 처음 연결할 때 Amazon Redshift는 기본적으로 연결된 데이터베이스의 리소스를 표시합니다.
5. 이러한 다른 데이터베이스에 대한 데이터베이스 객체를 볼 수 있는 액세스 권한이 있는 다른 데이터베이스를 선택합니다. 객체를 보려면 적절한 권한이 있어야 합니다. 데이터베이스를 선택하면 Amazon Redshift가 데이터베이스의 스키마 목록을 표시합니다.

   스키마를 선택하여 해당 스키마 내의 데이터베이스 객체 목록을 봅니다.

   > **Note**
   >
   > Amazon Redshift는 AWS Glue 또는 연합 데이터베이스의 일부인 쿼리 카탈로그 객체를 직접 지원하지 않습니다. 이를 쿼리하려면 먼저 각 데이터베이스의 외부 데이터 원본을 참조하는 외부 스키마를 생성합니다.
   > 세 부분으로 구성된 표기법을 사용하는 Amazon Redshift 데이터베이스 간 쿼리는 `information_schema` 및 `pg_catalog` 스키마 아래의 메타데이터 테이블을 지원하지 않습니다. 이러한 메타데이터 뷰는 데이터베이스에 고유하기 때문입니다.
6. (옵션) 선택한 스키마의 테이블 또는 뷰 목록을 필터링합니다.

Amazon Redshift에 대한 Apache Iceberg 호환성

전체 Amazon Redshift 프로비저닝된 클러스터 또는 서버리스 네임스페이스를 AWS Glue Data Catalog에 등록하여 AWS 계정 간에 라이브 데이터를 안전하게 공유하는 카탈로그를 만들 수 있습니다. Apache Iceberg REST API를 지원하는 SQL 쿼리 엔진에서 이러한 카탈로그에 액세스할 수 있습니다. AWS Lake Formation은 카탈로그에 대한 권한을 관리하므로 사용자는 구체화된 뷰 및 제로 ETL 통합과 같은 Amazon Redshift 기능을 활용하면서 단일 권한 집합으로 단일 데이터 복사본을 관리할 수 있습니다.

AWS Glue Data Catalog의 등록된 Amazon Redshift 프로비저닝된 클러스터 및 서버리스 네임스페이스에서 만들어진 모든 카탈로그는 동일한 계정에서 동일한 AWS 리전의 모든 프로비저닝된 클러스터 및 서버리스 작업 그룹에 외부 데이터베이스로 자동으로 탑재됩니다. Redshift Managed Storage(RMS)에 데이터를 저장하기 위해 AWS Glue Data Catalog에서 만들어진 카탈로그는 유사하게 외부 데이터베이스로 탑재됩니다. 탑재된 후에는 이러한 데이터베이스에 직접 연결하고 세 부분으로 구성된 표기법(`database@namespace-catalog.schema.table`)을 사용하여 객체를 쿼리할 수 있습니다.

Apache Iceberg 호환성을 사용할 수 있는 리전

Amazon Redshift와 Apache Iceberg의 호환성은 다음 AWS 리전에서 사용 가능합니다.

- 미국 동부(버지니아 북부)
- 미국 동부(오하이오)
- 미국 서부(캘리포니아 북부)
- 아시아 태평양(홍콩)
- 아시아 태평양(서울)
- 아시아 태평양(싱가포르)
- 아시아 태평양(시드니)
- 아시아 태평양(도쿄)
- 캐나다(중부)
- 유럽(프랑크푸르트)
- 유럽(아일랜드)
- Europe (London)

- Europe (Stockholm)
- 남아메리카(상파울루)

AWS Glue Data Catalog에서 Amazon Redshift 카탈로그를 사용할 때의 고려 사항 및 제한 사항

AWS Glue Data Catalog에서 Amazon Redshift 카탈로그를 사용할 때 다음 사항을 고려하세요.

- AWS Glue Data Catalog에 등록된 데이터 웨어하우스는 테이블에 액세스하기 위해 세 부분으로 구성된 구문(`database@namespace-catalog.schema.table`)을 따릅니다. 예를 들어, d라는 테이블이 있는 c라는 스키마가 포함된 b라는 데이터베이스로 채워진 a라는 Amazon Redshift 네임스페이스를 등록한 경우 다음 문을 사용하여 d에서 선택합니다.

  ```
  SELECT * FROM b@a.c.d;
  ```

  구문에서 `database@namespace-catalog` 부분 전체 길이는 127자 이하여야 합니다.
- 클러스터 또는 네임스페이스를 AWS Glue Data Catalog에 등록하면 Amazon Redshift는 해당 클러스터 또는 네임스페이스의 모든 데이터베이스와 관계를 등록합니다.
- 여러 Redshift 클러스터와 네임스페이스를 AWS Glue Data Catalog에 등록할 수 있습니다.
- 클러스터 또는 네임스페이스를 등록하면 해당 클러스터 또는 네임스페이스에 있는 내부 스키마 및 관계만 등록됩니다. 다음은 등록되지 않습니다.
  - 외부 스키마
  - 외부 테이블. 외부 테이블에서 만들어진 지연 바인딩 뷰가 등록된다는 것을 참고하세요.
  - 사용자가 만든 함수
  - 프로시저
  - 행 수준 보안 또는 동적 데이터 마스킹 정책이 연결된 테이블
  - 이름이 대문자이거나 대소문자가 혼합된 데이터베이스 객체. 대문자이거나 대소문자가 혼합된 테이블은 등록되지 않습니다. 이는 enable_case_sensitive_identifier가 비활성화된 경우에도 적용됩니다.
- 역할 기반 액세스 제어에서 부여한 역할과 같은 Amazon Redshift 데이터베이스 권한은 AWS Glue Data Catalog의 카탈로그로 전송되지 않습니다. AWS Lake Formation을 사용하여 AWS Glue Data Catalog에 대한 권한을 구성합니다. 권한 구성을 위한 Lake Formation 사용에 대한 자세한 내용은 AWS Lake Formation 개발자 안내서의 Amazon Redshift 데이터 공유에 대한 권한 설정을 참조하세요.

- 등록된 클러스터 또는 서버리스 네임스페이스에서 카탈로그를 생성할 때 AWS Glue Data Catalog는 Amazon Redshift 컴퓨팅 리소스를 사용하여 해당 카탈로그를 쿼리할 때 컴퓨팅 요구 사항을 처리하는 Amazon Redshift 관리형 작업 그룹을 만듭니다. Amazon Redshift Serverless 콘솔에서 관리형 작업 그룹을 보고 AWS Glue에서 관리할 수 있습니다.
- 일시 중지된 클러스터를 등록하면 클러스터가 재개될 때까지 AWS Glue Data Catalog가 해당 클러스터를 카탈로그로 탑재하지 않습니다.
- 적극적으로 사용되지 않는 서버리스 네임스페이스를 등록하면 네임스페이스가 다시 사용될 때까지는 AWS Glue Data Catalog가 해당 네임스페이스를 카탈로그로 탑재하지 않습니다.
- 관리형 작업 그룹을 만들려면 계정에 기본 VPC가 있어야 합니다.

Data Catalog에 등록된 클러스터 및 네임스페이스에 액세스하기 위한 IAM 정책 요구 사항

이 주제에서는 프로비저닝된 클러스터 및 서버리스 네임스페이스를 Data Catalog에 등록하고 Amazon Redshift를 사용하여 액세스하는 데 필요한 IAM 권한을 설명합니다.

프로비저닝된 클러스터 또는 서버리스 네임스페이스를 AWS Glue Data Catalog에 등록한 후에는 이후에 만들어진 카탈로그의 생성 및 변경 사항을 검색하려면 다음 권한이 필요합니다.

- `glue:GetCatalog`
- `glue:GetCatalogs`

이러한 권한은 서비스 연결 역할 `AmazonRedshiftServiceLinkedRolePolicy`에 포함됩니다. 이 역할에 대한 자세한 내용은 Amazon Redshift 관리 안내서의 Amazon Redshift에 대한 서비스 연결 역할 사용을 참조하세요.

Amazon Redshift 클러스터 및 네임스페이스를 AWS Glue Data Catalog에 등록

Amazon Redshift 프로비저닝된 클러스터와 서버리스 네임스페이스를 AWS Glue Data Catalog에 추가하여 Apache Iceberg REST API를 사용하여 액세스할 수 있습니다. 이렇게 하려면 Amazon Redshift 콘솔 또는 AWS CLI를 사용하여 Amazon Redshift 데이터 웨어하우스를 AWS Glue Data Catalog에 등록한 다음 AWS Lake Formation을 사용하여 웨어하우스에 대한 Amazon Redshift 페더레이션 카탈로그를 만듭니다.

Data Catalog에 등록된 Amazon Redshift 데이터 웨어하우스는 생산자 데이터 공유 역할을 합니다. 카탈로그에서 클러스터 또는 서버리스 네임스페이스를 변경하면 Redshift의 클러스터 또는 네임스페이스에 반영되고 그 반대의 경우도 마찬가지입니다.

Registering using the Amazon Redshift console

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 등록할 서버리스 네임스페이스 또는 프로비저닝된 클러스터로 이동하여 선택합니다.
3. 네임스페이스 또는 클러스터의 세부 정보 페이지의 작업 드롭다운 메뉴에서 AWS Glue Data Catalog에 등록을 선택합니다.
4. 네임스페이스 또는 클러스터를 등록할 대상 계정 ID를 입력하고 등록을 선택합니다.
5. 계정에서 AWS Glue의 동일한 계정으로 등록하는 경우 프로세스를 바로 완료하도록 AWS Lake Formation 콘솔로 이동합니다. 다른 계정에 등록하는 경우 Lake Formation으로 연결되는 링크가 나타납니다.

Registering using the AWS CLI

AWS CLI를 사용하여 클러스터 또는 네임스페이스를 AWS Glue Data Catalog에 등록하려면 다음 옵션과 함께 `register-namespace` 명령을 사용합니다.

- `namespace-identifier`: 등록하려는 클러스터 또는 네임스페이스의 고유 식별자가 있는 객체입니다. 이 객체는 프로비저닝된 클러스터를 등록하는지 서버리스 네임스페이스를 등록하는지에 따라 다릅니다. 다음을 고려하세요.
  - 프로비저닝된 클러스터의 경우 등록하려는 클러스터의 고유 식별자가 있는 `ClusterIdentifier` 객체가 포함된 `ProvisionedIdentifier` 객체를 제공합니다.
  - 서버리스 네임스페이스의 경우 등록하려는 네임스페이스의 고유 식별자가 있는 `NamespaceIdentifier` 객체와 해당 네임스페이스와 연결된 작업 그룹의 고유 식별자가 있는 `WorkgroupIdentifier` 객체가 포함된 `ServerlessIdentifier` 객체를 제공합니다.
- `consumer-identifiers`: 클러스터 또는 네임스페이스를 등록하려는 계정의 고유 식별자가 포함된 단일 요소가 있는 배열입니다.

다음 예시에서는 `mySampleNamespace` 서버리스 네임스페이스를 계정 ID `012345678910`에 등록합니다.

```
aws redshift register-namespace /
```

```
--namespace-identifier {ServerlessIdentifier: {NamespaceIdentifer:
 mySampleNamespace, WorkgroupIdentifier: mySampleWorkgroup}} /
--consumer-identifiers [012345678910]
```

AWS Glue Data Catalog에서 Amazon Redshift 클러스터 및 네임스페이스 등록 취소

Amazon Redshift 콘솔 또는 AWS CLI를 사용하여 AWS Glue Data Catalog에서 프로비저닝된 클러스터 또는 서버리스 네임스페이스의 등록을 취소할 수 있습니다. AWS Glue Data Catalog에서 등록 취소한 데이터 웨어하우스는 AWS Glue Data Catalog에서만 제거됩니다. Amazon Redshift 계정에는 웨어하우스가 남아 있습니다. 또한 카탈로그는 AWS Glue에 남아 있으므로 수동으로 제거해야 합니다. AWS Glue에서 카탈로그를 제거하면 카탈로그와 연결된 관리형 작업 그룹도 제거됩니다.

Deregistering using the Amazon Redshift console

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 등록 취소하려는 서버리스 네임스페이스 또는 프로비저닝된 클러스터로 이동하여 선택합니다.
3. 네임스페이스 또는 클러스터의 세부 정보 페이지의 작업 드롭다운 메뉴에서 AWS Glue Data Catalog에서 등록 취소를 선택합니다. 이 옵션은 AWS Glue Data Catalog에 이미 등록된 데이터 웨어하우스를 선택한 경우에만 나타납니다.
4. 클러스터 또는 네임스페이스가 있는 카탈로그를 소유한 계정 ID를 입력하고 등록 취소를 선택합니다.

Deregistering using the AWS CLI

AWS CLI를 사용하여 AWS Glue Data Catalog에서 클러스터 또는 네임스페이스의 등록을 취소하려면 다음 옵션과 함께 `deregister-namespace` 명령을 사용합니다.

- `namespace-identifier`: 등록 취소하려는 클러스터 또는 네임스페이스의 고유 식별자가 있는 객체입니다. 이 객체는 프로비저닝된 클러스터를 등록 취소하는지 서버리스 네임스페이스를 등록 취소하는지에 따라 다릅니다. 다음을 고려하세요.
  - 프로비저닝된 클러스터의 경우 등록 취소하려는 클러스터의 고유 식별자가 있는 `ClusterIdentifier` 객체가 포함된 `ProvisionedIdentifier` 객체를 제공합니다.

- 서버리스 네임스페이스의 경우 등록 취소하려는 네임스페이스의 고유 식별자가 있는 `NamespaceIdentifier` 객체와 해당 네임스페이스와 연결된 작업 그룹의 고유 식별자가 있는 `WorkgroupIdentifier` 객체가 포함된 `ServerlessIdentifier` 객체를 제공합니다.
- `consumer-identifiers`: 클러스터 또는 네임스페이스를 등록 취소하려는 계정의 고유 식별자가 포함된 단일 요소가 있는 배열입니다.

다음 예시에서는 `mySampleNamespace` 서버리스 네임스페이스를 계정 ID `012345678910`에서 등록 취소합니다.

```
aws redshift deregister-namespace /
--namespace-identifier {ServerlessIdentifier: {NamespaceIdentifer:
 mySampleNamespace, WorkgroupIdentifier: mySampleWorkgroup}} /
--consumer-identifiers [012345678910]
```

관리형 작업 그룹

프로비저닝된 클러스터 또는 서버리스 네임스페이스를 AWS Glue Data Catalog에 등록하고 여기에서 카탈로그를 만들면 AWS Glue는 해당 카탈로그에 액세스하는 SQL 쿼리 엔진에 대한 컴퓨팅 리소스를 제공하는 관리형 작업 그룹을 만듭니다. 예를 들어 카탈로그에서 테이블을 쿼리하는 Amazon Athena 사용자는 관리형 작업 그룹의 Amazon Redshift 컴퓨팅 리소스로 컴퓨팅 요구 사항을 충족합니다. 관리형 작업 그룹은 Amazon Redshift 컴퓨팅 리소스를 사용하지만 AWS Glue에서 관리됩니다. 관리형 작업 그룹을 만들려면 계정에 기본 VPC가 있어야 합니다.

관리형 작업 그룹은 기본적으로 25인 Amazon Redshift Serverless 작업 그룹 할당량에 포함됩니다. 자세한 내용은 Amazon Redshift 관리 가이드의 Amazon Redshift의 할당량 및 제한 섹션을 참조하세요.

Amazon Redshift Serverless 콘솔에서 관리형 작업 그룹에 대한 일반 정보, 성능 지표 및 쿼리 정보를 볼 수 있습니다. 자세한 내용은 Amazon Redshift 관리 안내서의 작업 그룹 속성 보기를 참조하세요.

AWS Glue Data Catalog에 등록된 카탈로그 쿼리

Amazon Redshift 데이터 웨어하우스를 AWS Glue Data Catalog에 등록하고 AWS Lake Formation에서 결과 카탈로그에 대한 권한을 설정하면 동일한 계정 및 AWS 리전의 소스 데이터 웨어하우스에 액세스할 수 있는 모든 Amazon Redshift 인스턴스에 카탈로그가 자동으로 탑재됩니다. 그런 다음 로컬 클러스터 또는 작업 그룹과 마찬가지로 해당 카탈로그를 쿼리할 수 있습니다. Apache Iceberg REST Open API를 지원하는 SQL 엔진을 사용하여 AWS Glue Data Catalog에 등록된 카탈로그를 쿼리할 수

도 있습니다. Apache Iceberg REST API를 사용하여 AWS Glue Data Catalog에서 카탈로그를 쿼리하는 방법에 대한 자세한 내용은 AWS Glue 개발자 안내서의 데이터 카탈로그 액세스를 참조하세요. Apache Iceberg REST API에 대한 자세한 내용은 Apache Iceberg REST Open API specification을 참조하세요.

카탈로그를 쿼리하려면 먼저 AWS Lake Formation을 사용하여 카탈로그에 대한 권한을 설정해야 합니다. AWS Lake Formation에서 카탈로그에 대한 권한 설정에 대한 자세한 내용은 AWS Lake Formation 개발자 안내서의 Amazon Redshift 데이터 공유에 대한 권한 설정을 참조하세요. `AmazonRedshiftServiceLinkedRolePolicy` 관리형 정책이 연결된 IAM 역할도 필요합니다. 서비스 연결 역할에 대한 자세한 내용은 Amazon Redshift 관리 안내서의 Amazon Redshift에 대한 서비스 연결 역할 사용을 참조하세요.

카탈로그에 대한 쿼리는 테이블에 액세스하려면 다음 세 부분으로 구성된 구문을 따라야 합니다.

```
database@namespace.schema.table
```

Amazon Redshift 데이터 웨어하우스 쿼리에 대한 일반적인 정보는 Amazon Redshift 관리 안내서의 데이터베이스 쿼리를 참조하세요.

Querying using the query editor v2

관리형 작업 그룹에 액세스하기 위해 계정에 대한 권한을 설정하면 해당 관리형 작업 그룹이 서버리스 데이터베이스의 외부 데이터베이스 섹션 아래 트리 뷰 패널에 나타납니다. 내부 Amazon Redshift 프로비저닝된 클러스터 또는 서버리스 작업 그룹을 쿼리하는 것과 동일한 방식으로 세 부분으로 구성된 구문 형식(`database@namespace/cluster.schema.table`)을 사용하여 관리형 작업 그룹을 쿼리할 수 있습니다. 다음 샘플 문을 참고하세요.

```
SELECT price FROM sales_db@mynamespace.sales_schema.inventory_table
```

Querying using the Data API

카탈로그의 Amazon 리소스 이름(ARN)을 관련 `database` 속성에 전달하여 내부 Amazon Redshift 프로비저닝된 클러스터 또는 서버리스 작업 그룹을 쿼리하는 것과 동일한 방식으로 Amazon Redshift Data API를 사용하여 관리형 작업 그룹을 쿼리할 수 있습니다. 카탈로그에 테이블을 만드는 다음 예시를 참고하세요.

```
aws redshift-data execute-statement –sql 'CREATE TABLE IF NOT EXISTS "dev@test-
rms-catalog".public.t1 (c1 INT, c2 VARCHAR(10));' –database arn:aws:glue:us-
east-1:550022730026:catalog/test-rms-catalog
```

Amazon Redshift에서 데이터 공유

Amazon Redshift를 사용하면 Amazon Redshift 클러스터 또는 다른 AWS 서비스 전반에서 데이터를 안전하게 공유할 수 있습니다. 데이터 공유를 통해 복사본을 만들거나 이전하지 않고도 실시간 데이터 공유가 가능합니다. 데이터베이스 관리자와 데이터 엔지니어는 데이터 공유를 사용하여 데이터에 대한 제어를 유지 관리하면서 분석 목적으로 데이터에 대해 안전한 읽기 전용 액세스를 제공할 수 있습니다. 데이터 분석가, 비즈니스 인텔리전스 전문가 및 데이터 과학자는 공유 데이터를 활용하여 데이터를 복제하거나 이전하지 않고도 인사이트를 확보합니다. 일반적인 사용 사례에는 파트너와 데이터 공유, 부서 간 분석 사용, 조직 내 데이터 민주화 촉진이 포함됩니다. 다음 섹션에서는 Amazon Redshift에서 데이터 공유를 구성하고 관리하는 방법에 대한 세부 정보를 다룹니다.

Amazon Redshift 데이터 공유를 사용하면 데이터를 수동으로 이동하거나 복사하지 않고도 Amazon Redshift 클러스터, 작업 그룹, AWS 계정 및 AWS 리전 전체에서 라이브 데이터에 대한 액세스를 안전하게 공유할 수 있습니다. 데이터가 라이브 상태이므로 모든 사용자는 업데이트되는 즉시 Amazon Redshift에서 가장 최신의 일관된 정보를 볼 수 있습니다.

프로비저닝된 클러스터, 서버리스 작업 그룹, 가용 영역, AWS 계정, AWS 리전 등에서 데이터를 공유할 수 있습니다. 클러스터 유형 간뿐만 아니라 프로비저닝된 클러스터와 서버리스 간에 공유할 수 있습니다.

서로 다른 Amazon Redshift 클러스터 또는 Amazon Redshift Serverless 작업 그룹 내에서, 한 AWS 계정에서, 또는 한 AWS 계정에서 다른 계정 간에 읽기 및 쓰기 모두에 대한 데이터베이스 객체를 공유할 수 있습니다. 서로 다른 리전 간에도 데이터를 쓸 수 있습니다. 서로 다른 테이블에 대해 SELECT, INSERT, UPDATE 등의 권한을 부여하고 서로 다른 스키마에 대해 USAGE 및 CREATE와 같은 권한을 부여할 수 있습니다. 데이터는 쓰기 트랜잭션이 커밋되는 즉시 라이브 상태로 모든 웨어하우스에서 사용할 수 있습니다.

Amazon Redshift의 데이터 공유 사용 사례

Amazon Redshift datashare는 다음과 같은 사용 사례에 특히 유용합니다.

- 다양한 종류의 비즈니스 크리티컬 워크로드 지원 – 여러 비즈니스 인텔리전스(BI) 또는 분석 클러스터와 데이터를 공유하는 중앙 추출, 변환 및 로드(ETL) 클러스터를 사용합니다. 이 접근 방식은 개별 워크로드에 대한 읽기 워크로드 격리 및 차지백을 제공합니다. 가격 및 성능에 대한 워크로드별 요구 사항에 따라 개별 워크로드 컴퓨팅의 크기를 조정하고 확장할 수 있습니다.
- 그룹 간 협업 사용 - 광범위한 분석, 데이터 과학 및 제품 간 영향 분석을 위해 팀과 비즈니스 그룹 간의 원활한 협업을 지원합니다.

- 데이터를 서비스로 제공 – 조직 전체에서 데이터를 서비스로 공유합니다.
- 환경 간 datashare – 개발, 테스트 및 프로덕션 환경 간에 데이터를 공유합니다. 다양한 세분화된 수준에서 데이터를 공유하여 팀 민첩성을 향상시킬 수 있습니다.
- Amazon Redshift의 데이터에 대한 액세스 라이선스 부여 – 고객이 몇 분 안에 찾고, 구독하고, 쿼리할 수 있는 Amazon Redshift 데이터 집합을 AWS Data Exchange 카탈로그에 나열합니다.

데이터 공유 쓰기 액세스 사용 사례

쓰기 데이터 공유에는 몇 가지 중요한 사용 사례가 있습니다.

- 생산자에 대한 비즈니스 소스 데이터 업데이트 - 조직 전체에서 데이터를 서비스로 공유할 수 있지만, 이렇게 하면 소비자도 소스 데이터에 대한 작업을 수행할 수 있습니다. 예를 들어 소비자가 최신 값을 전달하거나 데이터 수신을 승인할 수 있습니다. 이는 비즈니스 사용 사례 중 몇 가지에 불과합니다.
- 생산자에 추가 레코드 삽입 - 소비자가 원래 소스 데이터에 레코드를 추가할 수 있습니다. 필요한 경우 소비자가 추가한 것으로 표시할 수 있습니다.

데이터 공유에서 쓰기 작업을 수행하는 방법에 대한 구체적인 내용은 Sharing write access to data를 참조하세요.

Amazon Redshift에서 다양한 수준의 데이터 공유

Amazon Redshift를 사용하면 여러 수준에서 데이터를 공유할 수 있습니다. 이러한 수준에는 데이터베이스, 스키마, 테이블, 뷰(일반, 후기 바인딩 및 구체화된 뷰 포함) 및 SQL 사용자 정의 함수(UDF)가 포함됩니다. 지정된 데이터베이스에 대해 여러 datashare를 생성할 수 있습니다. datashare에는 공유가 생성되는 데이터베이스에 있는 여러 스키마의 객체가 포함될 수 있습니다.

datashare에 있어 이러한 유연성을 갖추면 세분화된 액세스 제어가 가능합니다. Amazon Redshift 데이터에 액세스해야 하는 다양한 사용자와 비즈니스에 맞게 이 제어를 조정할 수 있습니다.

Amazon Redshift에서 데이터 공유를 위한 일관성 관리

Amazon Redshift는 모든 생산자 및 소비자 클러스터에서 트랜잭션 일관성을 제공하고 모든 소비자와 데이터의 일관된 최신 뷰를 공유합니다.

생산자 클러스터에서 데이터를 지속적으로 업데이트할 수 있습니다. 트랜잭션 내의 소비자 클러스터에 대한 모든 쿼리는 공유 데이터의 동일한 상태를 읽습니다. Amazon Redshift는 소비자 클러스터에서

트랜잭션이 시작된 후 커밋된 생산자 클러스터의 다른 트랜잭션에 의해 변경된 데이터를 고려하지 않습니다. 생산자 클러스터에서 데이터 변경이 커밋된 후 소비자 클러스터의 새 트랜잭션은 업데이트된 데이터를 즉시 쿼리할 수 있습니다.

일관성이 뛰어나 데이터 공유 중 잘못된 결과를 포함할 수도 있는 충실도가 낮은 비즈니스 보고서가 작성될 위험이 없습니다. 이 요소는 재무 분석이나 기계 학습 모델을 훈련하는 데 사용되는 데이터 집합을 준비하는 데 결과를 사용할 수 있는 경우에 특히 중요합니다.

Amazon Redshift의 데이터 공유 고려 사항

다음은 Amazon Redshift 데이터 공유 작업 시 고려 사항입니다.

- 리전 간 데이터 공유에는 리전 간 데이터 전송 요금이 추가로 부과됩니다. 이러한 데이터 전송 요금은 동일한 리전 내에서는 적용되지 않고 리전 간에만 적용됩니다. 자세한 내용은 리전 간 데이터 공유를 위한 비용 관리 단원을 참조하십시오.
- 데이터 공유에서 데이터를 읽을 때는 로컬 클러스터 데이터베이스에 연결된 상태를 유지합니다. 데이터 공유에서 생성된 데이터베이스 설정 및 읽기에 대한 자세한 내용은 데이터 공유 객체 쿼리 및 Amazon Redshift Spectrum의 외부 데이터 레이크 테이블에 대한 구체화된 뷰를 참조하세요.
- 생산자의 데이터를 쿼리하는 데 필요한 모든 컴퓨팅 및 리전 간 데이터 전송 요금은 소비자에게 부과됩니다. 프로비저닝된 클러스터 또는 서버리스 네임스페이스의 기본 데이터 스토리지에 대해서는 생산자에게 요금이 부과됩니다.
- 공유 데이터에 대한 쿼리의 성능은 소비자 클러스터의 컴퓨팅 용량에 따라 다릅니다.

데이터 공유를 위한 클러스터 암호화 관리

AWS 계정에서 데이터를 공유하려면 생산자 클러스터와 소비자 클러스터를 모두 암호화해야 합니다.

Amazon Redshift에서는 클러스터의 데이터베이스 암호화를 통해 저장된 데이터를 보호할 수 있습니다. 클러스터에서 암호화를 활성화하면 해당 클러스터와 스냅샷의 데이터 블록 및 시스템 메타데이터가 암호화됩니다. 클러스터를 시작할 때 암호화를 활성화하거나, AWS Key Management Service(AWS KMS) 암호화를 사용하도록 암호화되지 않은 클러스터를 수정할 수 있습니다.

Amazon Redshift 데이터베이스 암호화에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 Amazon Redshift 데이터베이스 암호화 섹션을 참조하세요.

전송 중인 데이터 보호를 위해 생산자 클러스터의 암호화 스키마를 통해 전송 중인 모든 데이터를 암호화합니다. 소비자 클러스터는 데이터가 로드될 때 이 암호화 스키마를 채택합니다. 그런 다음 소비자 클러스터는 일반적인 암호화된 클러스터로 작동합니다. 생산자와 소비자 간의 통신도 공유 키 스키

마를 사용하여 암호화됩니다. 전송 중 암호화에 대한 자세한 내용은 전송 중 데이터 암호화를 참조하세요.

클러스터 내 및 클러스터 간 데이터 공유

서로 다른 Amazon Redshift 프로비저닝 클러스터 또는 서버리스 작업 그룹 간에 데이터를 공유할 때만 데이터 공유가 필요합니다. 다른 데이터베이스의 객체에 대해 필요한 권한이 있는 한, 동일한 클러스터 내에서 간단한 세 부분으로 구성된 표기법 `database.schema.table`을 사용하여 다른 데이터베이스를 쿼리할 수 있습니다.

읽기 전용 및 다중 웨어하우스 쓰기 데이터 공유 비교

이전에는 데이터 공유의 객체가 모든 상황에서 읽기 전용이었습니다. 데이터 공유의 객체에 쓰는 기능은 새로운 기능입니다. 생산자가 데이터 공유의 객체에 INSERT 또는 CREATE와 같은 쓰기 권한을 특별히 부여한 경우에만 데이터 공유의 객체에 쓰기가 활성화됩니다. 또한 계정 간 공유의 경우 생산자는 쓰기에 대한 데이터 공유를 승인하고 소비자는 쓰기를 위해 특정 클러스터와 작업 그룹을 연결해야 합니다.

데이터 공유 읽기에 대한 제한 사항

다음은 Amazon Redshift에서 데이터 공유 읽기 작업 시 제한 사항입니다.

- 데이터 공유는 모든 프로비저닝된 RA3 클러스터 유형 및 Amazon Redshift Serverless에서 지원됩니다. 다른 클러스터 유형에서는 지원되지 않습니다.
- 생산자와 소비자 클러스터 및 서버리스 네임스페이스가 모두 동일한 계정에 있는 경우 동일한 암호화 유형(암호화되지 않은 유형 또는 암호화된 유형 중 하나)을 보유해야 합니다. Lake Formation 관리형 데이터 공유를 포함한 다른 모든 경우에는 소비자와 생산자를 모두 암호화해야 합니다. 이는 보안을 위한 것입니다. 하지만 동일한 암호화 키를 공유할 필요는 없습니다.
- 데이터 공유를 통해서만 SQL UDF를 공유할 수 있습니다. Python 및 Lambda UDF는 지원되지 않습니다.
- 생산자 데이터베이스에 특정 데이터 정렬이 있는 경우 소비자 데이터베이스에 동일한 데이터 정렬 설정을 사용합니다.
- Amazon Redshift는 생산자 클러스터에서 중첩된 SQL 사용자 정의 함수를 지원하지 않습니다.
- Amazon Redshift는 인터리브 정렬 키가 있는 테이블과 이러한 테이블을 참조하는 뷰 공유를 지원하지 않습니다.
- 소비자는 데이터 공유 객체를 다른 데이터 공유에 추가할 수 없습니다. 또한 소비자는 데이터 공유 객체를 참조하는 뷰를 다른 데이터 공유에 추가할 수 없습니다.

- Amazon Redshift는 액세스 준비와 실행 간에 동시 DDL이 발생한 데이터 공유 객체에 대한 액세스를 지원하지 않습니다.
- Amazon Redshift는 데이터 공유를 통한 저장 프로시저 공유를 지원하지 않습니다.
- Amazon Redshift는 메타데이터 시스템 뷰 및 시스템 테이블 공유를 지원하지 않습니다.

다중 웨어하우스 쓰기에 대한 제한 사항

> **Note**
>
> 데이터 공유를 사용하는 Amazon Redshift 다중 웨어하우스 쓰기는 현재 트랙 버전 1.0.78881 이상의 프로비저닝된 클러스터에 대한 Amazon Redshift 패치 186과 버전 1.0.78890 이상의 Amazon Redshift Serverless 작업 그룹에 대해서만 지원됩니다.

다음은 Amazon Redshift에서 데이터 공유 쓰기 작업 시 제한 사항입니다.

- 연결 - 데이터 공유에 쓰려면 데이터 공유 데이터베이스에 직접 연결하거나 USE 명령을 실행해야 합니다. 세 부분으로 구성된 표기법을 사용할 수도 있습니다. USE 명령은 외부 테이블에서 지원되지 않습니다.
- 컴퓨팅 유형 - 이 기능을 사용하려면 Serverless 작업 그룹, ra3.xlplus 클러스터, ra3.4xl 클러스터 또는 ra3.16xl 클러스터를 사용해야 합니다.
- 메타데이터 검색 - Redshift JDBC, ODBC 또는 Python 드라이버를 통해 데이터 공유 데이터베이스에 직접 연결된 소비자인 경우 다음과 같은 방법으로 카탈로그 데이터를 볼 수 있습니다.
  - SQL SHOW 명령
  - information_schema 테이블 및 뷰 쿼리
  - SVV 메타데이터 뷰 쿼리
- 권한 가시성 - 소비자는 SHOW GRANTS SQL 명령을 통해 데이터 공유에 부여된 권한을 볼 수 있습니다.
- 암호화 - 계정 간 데이터 공유의 경우 생산자 및 소비자 클러스터를 모두 암호화해야 합니다.
- 격리 수준 - 다른 Serverless 작업 그룹과 프로비저닝된 클러스터가 데이터베이스에 쓸 수 있으려면 데이터베이스의 격리 수준이 스냅샷 격리여야 합니다.
- 자동 작업 - 소비자가 데이터 공유 객체에 쓰는 경우 자동 분석 작업이 트리거되지 않습니다. 따라서 테이블 통계를 업데이트하려면 생산자가 테이블에 데이터를 삽입한 후 분석을 수동으로 실행해야 합니다. 이렇게 하지 않으면 쿼리 계획이 최적화되지 않을 수 있습니다.

- 다중 문 쿼리 및 트랜잭션 - 트랜잭션 블록 외부의 다중 문 쿼리는 현재 지원되지 않습니다. 따라서 dbeaver와 같은 쿼리 에디터를 사용하고 쓰기 쿼리가 여러 개 있는 경우 쿼리를 명시적인 BEGIN...END 트랜잭션 문으로 래핑해야 합니다.

  다중 명령문이 트랜잭션 외부에서 사용되는 경우 첫 번째 명령이 생산자 데이터베이스에 대한 쓰기인 경우 문의 후속 쓰기 명령은 해당 생산자 데이터베이스에만 허용됩니다. 첫 번째 명령이 읽기인 경우 후속 쓰기 명령은 사용된 데이터베이스가 설정된 경우 사용된 데이터베이스에만 허용되고, 설정되지 않은 경우 로컬 데이터베이스에만 허용됩니다. 트랜잭션의 쓰기는 단일 데이터베이스에만 지원됩니다.
- 소비자 크기 조정 - 데이터 공유를 사용하여 쓰기를 수행하려면 소비자 클러스터에 64개 이상의 조각이 있어야 합니다.
- 뷰 및 구체화된 뷰 - 데이터 공유 데이터베이스에서 뷰 또는 구체화된 뷰를 만들거나 업데이트하거나 변경할 수 없습니다.
- 보안 - 열 수준(CLS), 행 수준(RLS) 및 동적 데이터 마스킹(DDM)과 같은 보안 정책을 데이터 공유 객체에 연결하거나 제거할 수 없습니다.
- 관리 가능성 - 소비자 웨어하우스는 데이터 공유 객체 또는 데이터 공유 객체를 참조하는 뷰를 다른 데이터 공유에 추가할 수 없습니다. 또한 소비자는 기존 데이터 공유를 수정하거나 삭제할 수 없습니다.

데이터 레이크 테이블에 대한 제한 사항

다음은 Amazon Redshift에서 데이터 레이크 테이블 작업 시 제한 사항입니다.

- 데이터 레이크 테이블의 데이터 공유는 Amazon S3 버킷 암호화를 위한 고객 관리형 AWS KMS keys를 지원하지 않습니다. 암호화에 AWS 관리형 키를 사용할 수 있습니다. 자세한 내용은 Amazon S3 관리형 키를 사용한 서버 측 암호화(SSE-S3) 사용을 참조하세요.
- 암호화된 AWS Glue 카탈로그에서 데이터 레이크 테이블을 데이터 공유하려면 데이터 카탈로그 암호화의 지침에 따라 AWS KMS 작업을 IAM 역할에 위임해야 합니다.

데이터 공유에 부여할 수 있는 권한

데이터 공유 맥락에서 다양한 객체 유형과 다양한 권한을 부여할 수 있습니다.

스키마:

- 객체

- CREATE

테이블:

- SELECT
- INSERT
- UPDATE
- DELETE
- TRUNCATE
- DROP
- REFERENCES

함수:

- EXECUTE

데이터베이스

- CREATE

지원되는 SQL 문

다음 문은 쓰기를 포함한 데이터 공유에서 지원됩니다.

- ALTER TABLE RENAME { TO | COLUMN TO }
- ALTER SCHEMA RENAME
- BEGIN | START TRANSACTION
- END | COMMIT | ROLLBACK
- COPY(COMPUPDATE 제외)
- { CREATE | DROP } SCHEMA
- { CREATE | DROP | SHOW } TABLE
- CREATE TABLE table_name AS
- DELETE

- { GRANT | REVOKE } privilege_name ON OBJECT_TYPE object_name TO consumer_user
- INSERT
- SELECT
- INSERT INTO SELECT
- SHOW GRANTS. 자세한 내용은 SHOW GRANTS 단원을 참조하십시오.
- TRUNCATE
- UPDATE
- Super 데이터 유형 열

지원되지 않는 문 유형 - 다음은 지원되지 않습니다.

- 생산자에 쓸 때 소비자 웨어하우스에 대한 다중 문 쿼리
- 이전 명령이 읽기 문인 경우 다른 데이터베이스의 소비자 웨어하우스에 대한 다중 문 쿼리
- 공유 데이터베이스에 연결되지 않은 경우 one.dot 또는 two.dot 표기법과 같은 점 3개 표기법 이외의 객체 참조
- 소비자가 생산자에게 쓰는 동시성 조정 쿼리
- 소비자가 생산자에게 쓰는 자동 복사 작업 쓰기
- 소비자가 생산자에게 쓰는 스트리밍 작업
- 생산자 클러스터에서 소비자가 제로 ETL 통합 테이블 생성 제로 ETL 통합에 대한 자세한 내용은 제로 ETL 통합 작업을 참조하세요.
- 데이터 공유를 통한 인터리브 정렬 키를 사용한 테이블에 쓰기
- 데이터 공유를 통해 저장된 프로시저에 쓰기
- 데이터 공유를 통해 SQL 사용자 정의 함수(UDF)에 쓰기. 여기에는 중첩, Python 및 Lambda UDF가 포함됩니다.
- 생산자보다 컴퓨팅 조각이 많은 소비자 웨어하우스에 대한 ID 열의 UPDATE, INSERT 또는 COPY 문
- 생산자에게 쓸 때 소비자 웨어하우스에 비RMS 외부 테이블의 MERGE 문
- 다음이 포함된 CREATE TABLE 문:
  - 데이터 유형 VARBYTE로 설정한 DEFAULT 표현식. VARBYTE 데이터 유형은 암시적으로 다른 데이터 유형으로 변환될 수 없습니다. 자세한 내용은 CAST 함수 단원을 참조하십시오.
  - 생산자에게 쓸 때 소비자 웨어하우스에 대한 NULL 파라미터가 있는 AS OF SELECT 문
  - 생산자에게 쓸 때 소비자 웨어하우스에 대한 LIKE 파라미터

데이터 공유가 가능한 AWS 리전

다음 테이블에는 데이터 공유 읽기 및 쓰기 기능의 사용 가능 여부가 나와 있습니다.

리전	동일 리전 데이터 공유	리전 간 데이터 공유	AWS Lake Formation에서 통제하는 데이터 공유
미국 동부(버지니아 북부)(us-east-1)	예	예	예
미국 동부(오하이오)(us-east-2)	예	예	예
미국 서부(캘리포니아 북부)(us-west-1)	예	예	예
미국 서부(오레곤)(us-west-2)	예	예	예
아시아 태평양(홍콩)(ap-east-1)	예	아니요	No
아시아 태평양(뭄바이)(ap-south-1)	예	예	예
아시아 태평양(하이데라바드)(ap-south-2)	예	아니요	No
아시아 태평양(도쿄)(ap-northeast-1)	예	예	예
아시아 태평양(싱가포르)(ap-southeast-1)	예	예	예
아시아 태평양(시드니)(ap-southeast-2)	예	예	예

리전	동일 리전 데이터 공유	리전 간 데이터 공유	AWS Lake Formation에서 통제하는 데이터 공유
아시아 태평양(자카르타)(ap-southeast-3)	예	아니요	No
아시아 태평양(멜버른)(ap-southeast-4)	예	아니요	No
아시아 태평양(서울)(ap-northeast-2)	예	예	예
아시아 태평양(오사카)(ap-northeast-3)	예	아니요	No
중국(베이징) (cn-north-1)	예	아니요	No
중국(닝샤) (cn-northwest-1)	예	아니요	No
아프리카(케이프타운)(af-south-1)	예	아니요	No
캐나다 서부(캘거리)(ca-west-1)	예	아니요	No
캐나다(중부)(ca-central-1)	예	예	예
유럽(프랑크푸르트)(eu-central-1)	예	예	예
유럽(취리히)(eu-central-2)	예	아니요	No
유럽(아일랜드)(eu-west-1)	예	예	예

리전	동일 리전 데이터 공유	리전 간 데이터 공유	AWS Lake Formation에서 통제하는 데이터 공유
유럽(런던)(eu-west-2)	예	예	예
유럽(파리)(eu-west-3)	예	예	예
유럽(밀라노) (eu-south-1)	예	아니요	No
유럽(스페인)(eu-south-2)	예	아니요	No
유럽(스톡홀름)(eu-north-1)	예	예	예
중동(UAE)(me-central-1)	예	아니요	No
중동(바레인)(me-south-1)	예	아니요	No
이스라엘(텔아비브)(il-central-1)	예	아니요	No
남아메리카(상파울루)(sa-east-1)	예	예	예
AWS GovCloud(미국 동부)(us-gov-east-1)	예	아니요	예
AWS GovCloud(미국 서부)(us-gov-west-1)	예	아니요	예

Amazon Redshift에서 데이터 공유 시작하기

이 섹션의 다음 가이드 중 하나에 따라 데이터 공유를 시작할 수 있습니다.

주제

콘솔에서 읽기 전용 데이터 공유 시작하기

Amazon Redshift를 사용하면 콘솔을 사용하여 쓰기로 데이터 공유를 관리하여 Amazon Redshift 클러스터 및 AWS 계정 전반의 데이터에 대한 액세스를 제어하고 데이터를 거버넌스할 수 있습니다. 다음 섹션에서는 Amazon Redshift 콘솔을 사용하여 쓰기로 데이터 공유를 구성하고 관리하는 방법에 대한 단계별 설명을 제공합니다.

Amazon Redshift 콘솔을 사용하여 계정에서 생성되거나 다른 계정에서 공유된 datashare를 관리합니다.

데이터 공유를 생성, 편집 또는 삭제하려면 권한이 필요합니다. 자세한 내용은 Amazon Redshift에서 데이터 공유에 대한 권한 관리을 참조하세요.

- 생산자 관리자는 클러스터 탭에서 데이터 공유 만들기, 데이터 소비자 추가, 데이터 공유 객체 추가, 데이터 공유에서 데이터베이스 만들기, 데이터 공유 편집 또는 데이터 공유 삭제를 수행할 수 있습니다.

  탐색 메뉴에서 [클러스터(Clusters)] 탭으로 이동하고 클러스터 목록에서 클러스터를 선택합니다. 그런 다음, 다음 중 하나를 수행하세요.

  - Datashare(데이터 공유) 탭을 선택하고 Datashares created in my namespace(내 네임스페이스에서 생성된 datashare) 섹션을 선택합니다. 그런 다음, 다음 중 하나를 수행하세요.
    - 데이터 공유 만들기

      datashare가 생성되면 datashare 객체 또는 데이터 소비자를 추가할 수 있습니다. 자세한 내용은 데이터 공유에 데이터 공유 객체 추가 및 데이터 공유에 데이터 소비자 추가 섹션을 참조하세요.
    - 계정에 생성된 datashare 편집
    - 계정에 만들어진 데이터 공유 삭제
  - [Datashare(Datashares)]를 선택하고 [다른 클러스터의 datashare(Datashares from other clusters)]에서 datashare를 선택합니다. 그런 다음, 다음 중 하나를 수행하세요.

  - 데이터 공유 만들기
  - datashare에서 데이터베이스 생성
- [데이터베이스(Databases)]를 선택하고 [데이터베이스(Databases)] 섹션에서 데이터베이스를 선택합니다. 그런 다음 [datashare 생성(Create datashare)]을 선택합니다. 자세한 내용은 datashare에서 데이터베이스 생성 단원을 참조하십시오.

> **Note**
>
> 데이터베이스 및 데이터베이스 내의 객체를 보거나 클러스터의 datashare를 보려면 데이터베이스에 연결합니다. 자세한 내용은 데이터베이스로 연결 단원을 참조하십시오.

데이터베이스로 연결

Amazon Redshift를 사용하면 데이터 웨어하우스 클러스터에 대한 연결을 설정하고 SQL 쿼리를 실행하거나 데이터를 로드하거나 관리 작업을 수행할 수 있습니다. 데이터베이스에 연결하는 것은 클라이언트 애플리케이션 또는 도구와 Amazon Redshift 클러스터 간에 보안 채널을 만드는 과정입니다. 다음 섹션에서는 Amazon Redshift 데이터베이스에 연결하는 방법에 대한 단계별 설명을 제공합니다.

데이터베이스에 연결하여 이 클러스터의 데이터베이스 내에서 데이터베이스와 객체를 보거나 datashare를 봅니다.

모든 datashare를 보려면 지정된 데이터베이스에 연결하는 데 사용되는 사용자 자격 증명에 필요한 권한이 있어야 합니다.

로컬 연결이 없는 경우 다음 중 하나를 수행합니다.

- 클러스터 세부 정보 페이지의 데이터베이스(Databases) 탭에 있는 데이터베이스(Databases) 또는 Datashare 객체(Datashare objects) 섹션에서 데이터베이스에 연결(Connect to database)을 선택하면 클러스터의 데이터베이스 객체를 볼 수 있습니다.
- 클러스터 세부 정보 페이지의 [Datashare(Datashares)] 탭에서 다음 작업 중 하나를 수행합니다.
  - [다른 클러스터의 datashare(Datashares from other clusters)] 섹션에서 [데이터베이스에 연결(Connect to database)]을 선택하여 다른 클러스터의 datashare를 봅니다.
  - [내 클러스터에 생성된 datashare(Datashares created in my cluster)] 섹션에서 [데이터베이스에 연결(Connect to database)]을 선택하여 클러스터의 datashare를 봅니다.
- [데이터베이스에 연결(Connect to database)] 창에서 다음 중 하나를 수행합니다.

- [새 연결 생성(Create a new connection)]을 선택하는 경우 [AWS Secrets Manager]를 선택하여 저장된 보안 암호로 연결에 대한 액세스를 인증합니다.

  또는 [임시 자격 증명(Temporary credentials)]을 선택하여 데이터베이스 자격 증명으로 연결에 대한 액세스를 인증합니다. [데이터베이스 이름(Database name)] 및 [데이터베이스 사용자(Database user)] 값을 지정합니다.

  연결을 선택합니다.
- [최근 연결 사용(Use a recent connection)]을 선택하여 필요한 권한이 있는 다른 데이터베이스에 연결합니다.

  Amazon Redshift에서 자동으로 연결을 설정합니다.

데이터베이스 연결이 설정되면 데이터 공유 만들기, 데이터 공유 쿼리 또는 데이터 공유에서 데이터베이스 만들기를 시작할 수 있습니다.

datashare 생성

Amazon Redshift가 있으면 데이터 공유를 사용하여 Amazon Redshift 클러스터 또는 AWS 계정 전반에서 실시간 데이터를 공유할 수 있습니다. 데이터 공유는 Amazon Redshift 클러스터의 실시간 데이터를 다른 클러스터 또는 AWS 계정과 공유할 수 있는 소비자-생산자 객체입니다. 데이터 공유를 만들어 액세스에 대한 제어를 유지 관리하고 데이터를 최신 상태로 유지하면서 안전한 데이터 공유가 가능합니다. 다음 섹션에서는 데이터 공유를 만들고 스키마, 테이블 및 뷰와 같은 데이터베이스 객체를 추가하여 실시간 데이터를 안전하게 공유하는 방법에 대한 세부 정보를 제공합니다.

데이터 공유 만들기

데이터 공유는 데이터베이스 객체, 권한 및 소비자가 포함된 논리적 컨테이너입니다. 소비자는 사용자 계정 및 기타 AWS 계정의 Amazon Redshift 프로비저닝된 클러스터 또는 Amazon Redshift Serverless 네임스페이스입니다. 각 데이터 공유는 해당 데이터 공유가 생성된 데이터베이스와 연결되며 해당 데이터베이스의 객체만 추가할 수 있습니다. 생산자 관리자는 아래 절차 중 하나를 수행하여 콘솔과 SQL에서 데이터 공유를 만들 수 있습니다.

Console

콘솔에서 클러스터 또는 네임스페이스 세부 정보 페이지의 데이터 공유 탭에서 데이터 공유를 만들 수 있습니다. 데이터 공유가 만들어진 후 소비자 관리자로서 소비자의 데이터 공유에서 데이터베이스를 만들 수 있습니다.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 클러스터(Clusters)를 선택한 다음 클러스터를 선택합니다. 클러스터 세부 정보 페이지가 나타납니다.
3. 데이터베이스 연결이 없는 경우 클러스터 또는 네임스페이스 세부 정보 페이지에서 데이터 공유 탭의 데이터 공유 섹션에서 데이터베이스에 연결합니다. 내 계정에서 생성된 데이터 공유 섹션에서 데이터 공유 생성을 선택합니다. 데이터 공유 생성 페이지가 나타납니다.
4. [datashare 생성(Create datashare)]을 선택합니다. 로컬 데이터베이스에서만 datashare를 생성할 수 있습니다. 데이터베이스에 연결하지 않은 경우 데이터베이스에 연결 페이지가 나타납니다. 데이터베이스로 연결의 절차에 따라 데이터베이스에 연결합니다. 최근 연결이 있는 경우 데이터 공유 생성 페이지가 나타납니다.
5. Datashare 정보(Datashare information) 섹션에서 다음 중 하나를 선택합니다.
   - 데이터 공유를 선택하여 각기 다른 Amazon Redshift 데이터 웨어하우스(프로비저닝된 클러스터 또는 Serverless 엔드포인트)에서 또는 동일한 AWS 계정이나 다른 AWS 계정에서 읽기 또는 쓰기 목적으로 데이터를 공유할 데이터 공유를 만듭니다.
   - AWS Data Exchange datashare를 선택하여 AWS Data Exchange를 통해 데이터에 라이선스를 부여할 datashare를 생성합니다.
6. Datashare 이름(Datashare name), 데이터베이스 이름(Database name) 및 공개적으로 액세스 가능(Publicly accessible) 값을 지정합니다. 데이터베이스 이름을 변경할 때 새 데이터베이스 연결을 만듭니다.
7. 범위가 지정된 권한 또는 직접 권한 섹션을 사용하여 데이터 공유에 객체를 추가합니다. 데이터 공유에 객체를 추가하려면 Amazon Redshift에서 데이터 공유 만들기 섹션을 참조하세요.
8. 데이터 소비자 섹션에서 Amazon Redshift에 게시하거나 Lake Formation과 데이터 공유 프로세스를 시작하는 AWS Glue Data Catalog에 게시하도록 선택할 수 있습니다. 데이터 공유를 Amazon Redshift에 게시한다는 것은 다른 네임스페이스 또는 소비자 역할을 하는 Amazon Redshift 계정과 데이터를 공유한다는 의미입니다.

   > **Note**
   >
   > 데이터 공유가 생성되면 구성을 편집하여 다른 옵션에 게시할 수 없습니다.

9. [datashare 생성(Create datashare)]을 선택합니다.

SQL

다음 명령을 실행하여 데이터 공유를 생성합니다.

```
CREATE DATASHARE salesshare;
```

데이터 공유를 만들 때 각 데이터 공유는 데이터베이스와 연결됩니다. 해당 데이터베이스의 객체만 해당 datashare에서 공유할 수 있습니다. 동일하거나 다른 세부 수준의 객체를 사용하여 동일한 데이터베이스에 여러 datashare를 생성할 수 있습니다. 클러스터가 생성할 수 있는 datashare 수에는 제한이 없습니다. Amazon Redshift 콘솔을 사용하여 datashare를 생성할 수도 있습니다. 자세한 내용은 CREATE DATASHARE 단원을 참조하십시오.

만드는 중에 데이터 공유에 대한 보안 제한을 제어할 수도 있습니다. 다음 예에서는 퍼블릭 IP 액세스 권한이 있는 소비자가 데이터 공유를 읽을 수 있음을 보여줍니다.

```
CREATE DATASHARE my_datashare [PUBLICACCESSIBLE = TRUE];
```

PUBLICACCESSIBLE = TRUE로 설정하면 소비자가 공개적으로 액세스할 수 있는 클러스터 및 프로비저닝된 작업 그룹에서 데이터 공유를 쿼리할 수 있습니다. 허용하지 않으려면 이 옵션을 생략하거나 명시적으로 false로 설정하세요.

데이터 공유를 만든 후 소비자 유형에 대한 속성을 수정할 수 있습니다. 예를 들어 지정된 datashare의 데이터를 소비하려는 클러스터에 공개적으로 액세스할 수 없도록 정의할 수 있습니다. datashare에 지정된 보안 제한을 충족하지 않는 소비자 클러스터의 쿼리는 쿼리 런타임에 거부됩니다. 자세한 내용은 ALTER DATASHARE 단원을 참조하십시오.

데이터 공유에 데이터 공유 객체 추가

다음 절차 중 하나를 수행하여 콘솔 및 SQL에서 다양한 유형의 데이터베이스 객체를 추가할 수 있습니다.

Console

범위가 지정된 권한 또는 직접 권한 섹션을 사용하여 데이터 공유에 객체를 추가할 수 있습니다. 객체를 추가할 범위가 지정된 권한 부여 또는 직접 권한 부여를 선택합니다. 추가 버튼을 선택하여 객체를 추가합니다. 대화 상자가 나타납니다. 다음 단계를 수행합니다.

1. 범위가 지정된 권한 부여를 선택하면 데이터베이스 또는 스키마 수준에서 범위가 지정된 권한을 부여할 수 있는 범위가 지정된 권한 부여 페이지가 나타납니다. 범위가 지정된 권한이 있는

데이터 공유는 데이터베이스 또는 스키마 내의 모든 현재 및 미래 객체에 대해 지정된 권한을 갖습니다. 자세한 내용은 범위가 지정된 권한 항목을 참조하세요.

1. 그런 다음 데이터베이스 범위가 지정된 권한을 선택하여 데이터베이스 수준에서 범위가 지정된 권한을 부여합니다. 범위가 지정된 권한을 부여하면 데이터 공유를 만드는 동안 현재 데이터베이스에 적용됩니다. 이러한 권한은 개별 객체에 부여할 수 없으며 기존 객체와 새 객체(스키마, 테이블, 뷰, UDF) 모두에 적용됩니다.
2. 스키마, 테이블 또는 뷰 또는 사용자 정의 함수에 범위가 지정된 권한을 하나 이상 선택합니다. 이렇게 하면 데이터베이스의 모든 객체가 소비자에게 부여된 선택된 권한을 갖게 됩니다. 데이터베이스 범위가 지정된 권한 부여를 완료하려면 부여를 선택합니다.
3. 그런 다음 스키마 범위가 지정된 권한을 선택하여 스키마 수준에서 범위가 지정된 권한을 부여합니다. 스키마 범위가 지정된 권한을 부여하면 스키마에 추가된 모든 객체가 지정된 데이터 공유 권한을 갖게 됩니다.
4. 드롭다운에서 데이터 공유에 추가할 스키마를 선택합니다. 한 번에 하나의 스키마만 선택할 수 있습니다. 그런 다음 선택한 스키마에 부여하려는 직접 권한을 하나 이상 선택합니다.
5. 테이블, 뷰 및 사용자 정의 함수와 같은 스키마 객체에 범위가 지정된 권한을 하나 이상 선택합니다. 스키마의 매칭되는 모든 객체에 권한이 부여됩니다. 이러한 객체는 기존 객체이거나 향후 추가될 객체일 수 있습니다. 적용되면 범위가 지정된 권한을 취소하지 않고 객체에서 권한을 제거할 수 없습니다.
6. 스키마 범위가 지정된 권한 부여를 완료하려면 부여를 선택합니다.

2. 직접 권한 부여를 선택하면 직접 권한 부여 페이지가 나타나 스키마, 테이블, 뷰 또는 사용자 정의 함수와 같은 각 객체 수준에서 직접 권한을 부여할 수 있습니다. 직접 권한을 부여하려면 먼저 데이터 공유에 관련 스키마를 추가해야 합니다.

   1. 그런 다음 스키마에 직접 권한 부여를 선택하여 특정 스키마에 직접 권한을 적용합니다. 그런 다음 테이블, 뷰 및 사용자 정의 함수와 같은 스키마 객체의 스키마 권한을 하나 이상 선택하고 데이터 공유에 추가할 스키마를 선택합니다. 부여를 선택하여 데이터 공유에 스키마 추가를 완료합니다.
   2. 데이터 공유에 스키마를 추가한 후 스키마 객체에 대한 직접 권한 추가를 진행할 수 있습니다. 직접 권한 부여를 다시 선택합니다. 직접 권한 부여 페이지가 나타납니다. 그런 다음 스키마 객체에 대한 직접 권한 탭으로 이동합니다.
   3. 테이블 및 뷰에 직접 권한 부여를 선택하여 이러한 객체에 객체 수준 직접 권한을 부여합니다. 목록에서 필요한 직접 권한 하나 이상과 필요한 객체를 선택합니다. 검색 필드를 사용하여 데이터 공유 객체를 찾습니다. 부여를 선택하여 데이터 공유에 테이블 및 뷰 추가를 완료합니다.

4. 사용자 정의 함수에 직접 권한 부여를 선택하여 사용자 정의 함수에 객체 수준 직접 권한을 부여합니다. 목록에서 필요한 직접 권한 하나 이상과 필요한 객체를 선택합니다. 검색 필드를 사용하여 데이터 공유 객체를 찾습니다. 부여를 선택하여 데이터 공유에 사용자 정의 함수 추가를 완료합니다.

3. 미래 객체를 추가할지도 선택할 수 있습니다. 스키마에 추가된 데이터 공유 객체를 포함하도록 선택하면 스키마에 추가된 객체가 데이터 공유에 자동으로 추가됩니다.
4. 추가를 선택하여 섹션을 완료하고 객체를 추가합니다. 추가된 객체는 데이터 공유 객체에 나열됩니다.
5. 객체를 추가한 후 개별 객체를 선택하고 권한을 편집할 수 있습니다. 스키마를 선택하면 범위가 지정된 권한을 추가할지 묻는 대화 상자가 나타납니다. 이렇게 하면 스키마의 기존 객체 또는 추가된 각 객체가 객체 유형에 적합한 미리 선택된 권한 세트를 갖게 됩니다. 예를 들어 관리자는 추가된 모든 테이블에 SELECT 및 UPDATE 권한을 갖도록 설정할 수 있습니다.
6. 모든 데이터 공유 객체는 범위가 지정된 권한 또는 직접 권한 섹션에 나열됩니다.
7. 데이터 소비자 섹션에서 네임스페이스를 추가하거나 AWS 계정을 데이터 공유의 소비자로 추가할 수 있습니다.
8. 데이터 공유 생성을 선택하여 변경 사항을 저장합니다.

데이터 공유를 만들면 내 네임스페이스에서 생성된 데이터 공유 아래 목록에 해당 데이터 공유가 나타납니다. 목록에서 데이터 공유를 선택하면 해당 소비자, 객체 및 기타 속성을 볼 수 있습니다.

SQL

SQL을 사용하면 데이터 공유 소유자는 데이터 공유에 추가할 스키마에 대해 USAGE 권한을 부여해야 합니다. GRANT는 스키마에서 CREATE 및 USAGE를 포함하여 다양한 작업을 허용하는 데 사용됩니다. 스키마에는 공유 객체가 들어 있습니다.

```
CREATE SCHEMA myshared_schema1;
CREATE SCHEMA myshared_schema2;

GRANT USAGE ON SCHEMA myshared_schema1 TO DATASHARE my_datashare;
GRANT CREATE, USAGE ON SCHEMA myshared_schema2 TO DATASHARE my_datashare;
```

또는 관리자가 ALTER 명령을 계속 실행하여 데이터 공유에 스키마를 추가할 수도 있습니다. 이러한 방식으로 스키마를 추가할 경우 USAGE 권한만 부여됩니다.

```
ALTER DATASHARE my_datashare ADD SCHEMA myshared_schema1;
```

관리자는 스키마를 추가한 후 스키마의 객체에 대해 데이터 공유 권한을 부여할 수 있습니다. 이 권한은 읽기 및 쓰기 권한일 수 있습니다. GRANT ALL 샘플은 모든 권한을 부여하는 방법을 보여줍니다.

```
GRANT SELECT, INSERT ON TABLE myshared_schema1.table1, myshared_schema1.table2,
 myshared_schema2.table1
TO DATASHARE my_datashare;

GRANT ALL ON TABLE myshared_schema1.table4 TO DATASHARE my_datashare;
```

계속해서 ALTER DATASHARE와 같은 명령을 실행하여 테이블을 추가할 수 있습니다. 이렇게 하면 추가된 객체에 대해 SELECT 권한만 부여됩니다.

```
ALTER DATASHARE my_datashare ADD TABLE myshared_schema1.table1,
 myshared_schema1.table2, myshared_schema2.table1;
```

데이터베이스 또는 스키마 내 특정 유형의 모든 객체에 대해 범위가 지정된 권한을 데이터 공유에 부여할 수 있습니다. 범위가 지정된 권한이 있는 데이터 공유는 데이터베이스 또는 스키마 내의 모든 현재 및 미래 객체에 대해 지정된 권한을 갖습니다.

SVV_DATABASE_PRIVILEGES에서 데이터베이스 수준 범위 지정 권한의 범위를 볼 수 있습니다. SVV_SCHEMA_PRIVILEGES에서 스키마 수준 범위 지정 권한의 범위를 볼 수 있습니다.

다음은 데이터 공유에 범위가 지정된 권한을 부여할 때 사용하는 구문입니다. 범위가 지정된 권한에 대한 자세한 내용은 범위가 지정된 권한 섹션을 참조하세요.

```
GRANT { CREATE | USAGE | ALTER | DROP } [,...] | ALL [ PRIVILEGES ] }FOR SCHEMAS IN
DATABASE db_name
TO DATASHARE { datashare_name}

GRANT { { SELECT | INSERT | UPDATE | DELETE | DROP | ALTER | TRUNCATE | REFERENCES }
 [, ...] } | ALL [PRIVILEGES] } }FOR TABLES IN
{SCHEMA schema_name [DATABASE db_name ] | DATABASE db_name }
TO DATASHARE { datashare_name}

GRANT { EXECUTE | ALL [ PRIVILEGES ] }FOR FUNCTIONS IN
{SCHEMA schema_name [DATABASE db_name ] | DATABASE db_name }
TO DATASHARE { datashare_name}
```

데이터 공유에 데이터 소비자 추가

datashare기에 데이터 소비자를 하나 이상 추가할 수 있습니다. 데이터 소비자는 Amazon Redshift 클러스터 또는 AWS 계정을 고유하게 식별한 네임스페이스일 수 있습니다.

퍼블릭 액세스 권한이 있는 클러스터에서 datashare를 해제하거나 설정하도록 명시적으로 선택해야 합니다.

- 데이터 공유에 네임스페이스 추가를 선택합니다. 네임스페이스는 Amazon Redshift 클러스터에 대한 GUID(전역 고유 식별자)입니다.
- datashare에 AWS 계정 추가(Add)를 선택합니다. 지정된 AWS 계정에 datashare에 대한 액세스 권한이 있어야 합니다.

datashare에서 권한 부여 또는 제거

생산자 관리자는 데이터 공유에 대한 액세스 권한을 부여하거나 제거할 데이터 소비자를 선택합니다. 권한이 부여된 데이터 소비자는 datashare에 대한 작업을 수행하라는 알림을 받습니다. 네임스페이스를 데이터 소비자로 추가하는 경우 권한 부여를 수행할 필요가 없습니다.

사전 조건: datashare에 대한 권한을 부여하거나 제거하려면 datashare에 추가된 데이터 소비자가 하나 이상 있어야 합니다.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 Datashares를 선택합니다. datashare 목록 페이지가 나타납니다.
3. [내 계정에서(In my account)]를 선택합니다.
4. [내 계정의 datashare(Datashares in my account)] 섹션에서 다음 중 하나를 수행합니다.
   - 승인하려는 하나 이상의 소비자 클러스터를 선택합니다. 데이터 소비자 권한 부여 페이지가 나타납니다. 그런 다음 권한 부여(Authorize)를 선택합니다.

     데이터 공유를 생성할 때 Publish to AWS Glue Data Catalog(Glue 데이터 카탈로그에 게시)를 선택한 경우 데이터 공유 권한을 Lake Formation 계정에만 부여할 수 있습니다.

     AWS Data Exchange datashare의 경우 한 번에 하나의 datashare에만 권한을 부여할 수 있습니다.

     AWS Data Exchange datashare에 권한을 부여하면 AWS Data Exchange 서비스와 datashare를 공유하고 AWS Data Exchange에서 사용자를 대신하여 datashare에 대한 액세스를 관리하도록 허용합니다. AWS Data Exchange에서는 소비자가 제품을 구독할 때 AWS Data Exchange

datashare에 소비자 계정을 데이터 소비자로 추가하여 소비자에 대한 액세스를 허용합니다. AWS Data Exchange에는 datashare에 대한 읽기 권한이 없습니다.

- 승인을 제거하려는 하나 이상의 소비자 클러스터를 선택합니다. 그러면 [권한 제거(Remove authorization)]를 선택합니다.

권한을 부여 받은 데이터 소비자는 datashare 객체에 액세스하고 소비자 데이터베이스를 생성하여 데이터를 쿼리할 수 있습니다.

권한이 제거되면 데이터 소비자는 datashare에 대한 액세스 권한을 즉시 상실합니다.

소비자로서 다른 계정의 데이터 공유 관리

데이터 소비자에게서 datashare의 연결 제거

소비자 관리자는 데이터 소비자에게서 데이터 공유 연결을 제거할 수 있습니다.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 Datashares를 선택합니다. datashare 목록 페이지가 나타납니다.
3. [다른 계정에서(From other accounts)]를 선택합니다.
4. [다른 계정의 datashare(Datashares from other accounts)] 섹션에서 데이터 소비자에게서 연결을 제거할 datashare를 선택합니다.
5. [데이터 소비자(Data consumers)] 섹션에서 연결을 제거할 데이터 소비자를 하나 이상 선택합니다. 그런 다음 [연결 제거(Remove association)]를 선택합니다.
6. 연결 제거 페이지가 나타나면 [연결 제거(Remove association)]를 선택합니다.

연결이 제거되면 데이터 소비자는 datashare에 대한 액세스 권한을 상실합니다. 언제든지 데이터 소비자 연결을 변경할 수 있습니다.

datashare 거부

소비자 관리자는 사용 가능 또는 활성 상태인 데이터 공유를 거부할 수 있습니다. 데이터 공유를 거부하면 소비자 클러스터 사용자는 데이터 공유에 대한 액세스 권한을 상실합니다. Amazon Redshift는 `DescribeDataSharesForConsumer` API 작업을 호출하는 경우 거부된 데이터 공유를 반환하지 않습니다. 생산자 관리자가 `DescribeDataSharesForProducer` API 작업을 실행하면 데이터 공유가 거부된 것을 확인할 수 있습니다. 데이터 공유가 거부되면 생산자 관리자는 소비자 클러스터에 대한 데이터 공유를 다시 승인할 수 있으며, 소비자 관리자는 자신의 AWS 계정을 데이터 공유와 연결하거나 이를 거부하도록 선택할 수 있습니다.

AWS 계정이 데이터 공유와 연결되어 있고 Lake Formation에서 관리하는 데이터 공유에 대한 보류 중인 연결이 있는 경우. Lake Formation에서 관리하는 데이터 공유 연결을 거부하면 원래 데이터 공유도 거부됩니다. 특정 연결을 거부하려면 생산자 관리자가 지정된 데이터 공유에서 권한 부여를 제거할 수 있습니다. 이 작업은 다른 데이터 공유에 영향을 주지 않습니다.

데이터 공유를 거부하려면 AWS 콘솔, API 작업 `RejectDataShare` 또는 AWS CLI의 `reject-datashare`를 사용하세요.

AWS 콘솔을 사용하여 데이터 공유 생성

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 데이터 공유를 선택합니다.
3. [다른 계정에서(From other accounts)]를 선택합니다.
4. [다른 계정의 datashare(Datashares from other accounts)] 섹션에서 거부할 datashare를 선택합니다. datashare 거부(Decline datashare) 페이지가 나타나면 거부(Decline)를 선택합니다.

datashare를 거부한 후에는 변경 사항을 되돌릴 수 없습니다. Amazon Redshift가 목록에서 데이터 공유를 제거합니다. 데이터 공유를 다시 보려면 생산자 관리자가 데이터 공유를 다시 승인해야 합니다.

기존 데이터 공유 관리

Amazon Redshift를 사용하면 기존 데이터 공유를 관리하여 Amazon Redshift 클러스터의 데이터에 대한 액세스를 제어할 수 있습니다. 다음 섹션에서는 Amazon Redshift 환경에서 데이터 공유를 관리하는 방법에 대한 단계별 가이드를 제공합니다.

datashare 보기

[datashare(DATASHARES)] 또는 [클러스터(CLUSTERS)] 탭에서 datashare를 봅니다.

- [datashare(DATASHARES)] 탭을 사용하여 자신의 계정이나 다른 계정의 datashare를 나열합니다.
  - 계정에서 생성된 datashare를 보려면 [내 계정에서(In my account)]를 선택하고 보려는 datashare를 선택합니다.
  - 다른 계정에서 공유되는 datashare를 보려면 [다른 계정에서(From other accounts)]를 선택하고 보려는 datashare를 선택합니다.
- [클러스터(CLUSTERS)] 탭을 사용하여 자신의 클러스터나 다른 클러스터의 datashare를 나열합니다.

데이터베이스에 연결합니다. 자세한 내용은 데이터베이스로 연결 단원을 참조하십시오.

그런 다음 [다른 클러스터의 datashare(Datashares from other clusters)] 또는 [내 클러스터에 생성된 datashare(Datashares created in my cluster)] 섹션에서 datashare를 선택하여 해당 세부 정보를 봅니다.

datashare에서 datashare 객체 제거

다음 절차를 사용하여 datashare에서 객체를 하나 이상 제거할 수 있습니다.

datashare에서 객체를 하나 이상 제거하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 클러스터(Clusters)를 선택한 다음 클러스터를 선택합니다. 클러스터 세부 정보 페이지가 나타납니다.
3. [Datashare(Datashares)]를 선택합니다.
4. [내 계정에 생성된 datashare(Datashares created in my account)] 섹션에서 [데이터베이스에 연결(Connect to database)]을 선택합니다. 자세한 내용은 데이터베이스로 연결 단원을 참조하십시오.
5. 편집할 datashare를 선택한 다음 편집(Edit)을 선택합니다. datashare 세부 정보 페이지가 나타납니다.
6. datashare 객체를 하나 이상 제거하려면 다음 중 하나를 수행합니다.

   - datashare에서 스키마를 제거하려면 스키마를 하나 이상 선택합니다. 그런 다음 [제거(Remove)]를 선택합니다. Amazon Redshift는 datashare에서 지정된 스키마와 해당 스키마의 모든 객체를 제거합니다.
   - datashare에서 테이블과 뷰를 제거하려면 테이블과 뷰를 하나 이상 선택합니다. 그런 다음 [제거(Remove)]를 선택합니다. 또는 [스키마별 제거(Remove by schema)]를 선택하여 지정된 스키마의 모든 테이블과 뷰를 제거합니다.
   - datashare에서 사용자 정의 함수를 제거하려면 사용자 정의 함수를 하나 이상 선택합니다. 그런 다음 [제거(Remove)]를 선택합니다. 또는 [스키마별 제거(Remove by schema)]를 선택하여 지정된 스키마의 모든 사용자 정의 함수를 제거합니다.

datashare에서 데이터 소비자 제거

datashare에서 데이터 소비자를 하나 이상 제거할 수 있습니다. 데이터 소비자는 Amazon Redshift 클러스터 또는 AWS 계정을 고유하게 식별한 네임스페이스일 수 있습니다.

네임스페이스 ID 또는 AWS 계정에서 데이터 소비자를 하나 이상 선택한 다음 제거를 선택합니다.

Amazon Redshift는 datashare에서 지정된 데이터 소비자를 제거합니다. 제거된 소비자는 datashare에 대한 액세스 권한을 즉시 상실합니다.

계정에 생성된 datashare 편집

콘솔을 사용하여 계정에 생성된 datashare를 편집합니다. 먼저 데이터베이스에 연결하여 계정에 생성된 datashare 목록을 봅니다.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 클러스터(Clusters)를 선택한 다음 클러스터를 선택합니다. 클러스터 세부 정보 페이지가 나타납니다.
3. [Datashare(Datashares)]를 선택합니다.
4. [내 계정에 생성된 datashare(Datashares created in my account)] 섹션에서 [데이터베이스에 연결(Connect to database)]을 선택합니다. 자세한 내용은 데이터베이스로 연결 단원을 참조하십시오.
5. 편집할 datashare를 선택한 다음 편집(Edit)을 선택합니다. datashare 세부 정보 페이지가 나타납니다.
6. [datashare 객체(Datashare objects)] 또는 [데이터 소비자(Data consumers)] 섹션에서 변경합니다.

> **Note**
>
> 데이터 공유를 AWS Glue Data Catalog에 게시하도록 선택한 경우 데이터 공유를 다른 Amazon Redshift 계정에 게시하도록 구성을 편집할 수 없습니다.

7. Save changes(변경 사항 저장)를 선택합니다.

변경 사항으로 datashare가 업데이트됩니다.

계정에 만들어진 데이터 공유 삭제

콘솔을 사용하여 계정에 생성된 datashare를 삭제합니다. 먼저 데이터베이스에 연결하여 계정에 생성된 datashare 목록을 봅니다.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 클러스터(Clusters)를 선택한 다음 클러스터를 선택합니다. 클러스터 세부 정보 페이지가 나타납니다.
3. [Datashare(Datashares)]를 선택합니다. datashare 목록이 나타납니다.
4. [내 계정에 생성된 datashare(Datashares created in my account)] 섹션에서 [데이터베이스에 연결(Connect to database)]을 선택합니다. 자세한 내용은 데이터베이스로 연결 단원을 참조하십시오.
5. 삭제하려는 datashare를 하나 이상 선택하고 [삭제(Delete)]를 선택합니다. datashare 삭제 페이지가 나타납니다.

   Lake Formation과 공유된 데이터 공유를 삭제해도 Lake Formation에서 연결된 권한이 자동으로 제거되지 않습니다. 이를 제거하려면 Lake Formation 콘솔로 이동하세요.
6. [삭제(Delete)]를 입력하여 지정된 datashare 삭제를 확인합니다.
7. Delete(삭제)를 선택합니다.

datashare가 삭제되면 데이터 소비자는 datashare에 대한 액세스 권한을 상실합니다.

데이터 공유 쿼리

Amazon Redshift를 사용하면 생산자 클러스터에서 데이터 공유 전반에 걸쳐 데이터를 쿼리하여 실시간 데이터를 복사하거나 전송하지 않고도 안전하게 액세스할 수 있습니다. 다음 섹션에서는 Amazon Redshift 환경에서 데이터 공유 설정 및 쿼리에 대한 세부 정보를 다룹니다.

datashare에서 데이터베이스 생성

datashare에서 데이터 쿼리를 시작하기 위해 datashare에서 데이터베이스를 생성합니다. 지정된 datashare에서 데이터베이스를 하나만 생성할 수 있습니다.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 클러스터(Clusters)를 선택한 다음 클러스터를 선택합니다. 클러스터 세부 정보 페이지가 나타납니다.
3. [Datashare(Datashares)]를 선택합니다. datashare 목록이 나타납니다.
4. [다른 클러스터의 datashare(Datashares from other clusters)] 섹션에서 [데이터베이스에 연결(Connect to database)]을 선택합니다. 자세한 내용은 데이터베이스로 연결 단원을 참조하십시오.

5. 데이터베이스를 생성하려는 datashare를 선택하고 [datashare에서 데이터베이스 생성(Create database from datashare)]을 선택합니다. datashare에서 데이터베이스 생성 페이지가 나타납니다.
6. [데이터베이스 이름(Database name)]에서 데이터베이스 이름을 지정합니다. 데이터베이스 이름은 1~64자의 영숫자(소문자만)여야 하며 예약어일 수 없습니다.
7. 생성(Create)을 선택합니다.

datashare가 생성된 후 데이터베이스에서 데이터를 쿼리할 수 있습니다.

AWS Data Exchange 데이터 공유 쿼리

Amazon Redshift를 사용하면 데이터 추출 또는 파이프라인을 생성하고 관리할 필요 없이 AWS Data Exchange에서 실시간 데이터를 안전하게 공유 및 수신할 수 있습니다. AWS Data Exchange 데이터 공유를 관리하면 서드 파티 데이터 제품을 구독하고 실시간 데이터 스트림을 Amazon Redshift 데이터 웨어하우스에 직접 통합할 수 있습니다. 다음 섹션에서는 Amazon Redshift 클러스터 내에서 AWS Data Exchange 데이터 공유를 관리하는 방법을 보여 줍니다.

AWS Data Exchange에서 데이터 집합 생성

AWS Data Exchange에 데이터 집합을 생성합니다.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 클러스터(Clusters)를 선택한 다음 클러스터를 선택합니다. 클러스터 세부 정보 페이지가 나타납니다.
3. Datashare를 선택합니다.
4. 내 계정에 생성된 datashare(Datashares created in my account) 섹션에서 AWS Data Exchange datashare를 선택합니다.
5. Create data set on AWS Data Exchange(ADE에서 데이터 집합 생성)를 선택합니다. 자세한 내용은 새 제품 게시를 참조하세요.

AWS Data Exchange datashare 편집

콘솔을 사용하여 AWS Data Exchange datashare를 편집합니다. 먼저 데이터베이스에 연결하여 계정에 생성된 datashare 목록을 봅니다.

AWS Data Exchange datashare의 경우 데이터 소비자를 변경할 수 없습니다.

AWS Data Exchange datashare에 대해 공개적으로 액세스할 수 있는 설정을 편집하려면 쿼리 편집기 v2를 사용합니다. Amazon Redshift는 임의의 일회성 값을 생성하여이 설정을 해제할 수 있도록 세션 변수를 설정합니다. 자세한 내용은 ALTER DATASHARE 사용 참고 사항 단원을 참조하십시오.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 클러스터(Clusters)를 선택한 다음 클러스터를 선택합니다. 클러스터 세부 정보 페이지가 나타납니다.
3. 탐색기 메뉴에서 편집기(Editor), 쿼리 편집기 v2(Query editor v2)를 차례로 선택합니다.
4. 쿼리 편집기 v2를 처음 사용하는 경우 AWS 계정을 구성합니다. 기본적으로 AWS 소유의 키가 리소스를 암호화하는 데 사용됩니다. AWS 계정 구성에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 AWS 계정 구성 섹션을 참조하세요.
5. AWS Data Exchange datashare가 있는 클러스터에 연결하려면 트리 보기 패널에서 데이터베이스(Database)와 클러스터 이름을 선택합니다. 메시지가 나타나면 연결 파라미터를 입력합니다.
6. 다음 SQL 문을 복사합니다. 다음 예에서는 공개적으로 액세스할 수 있는 salesshare datashare 설정을 변경합니다.

```
ALTER DATASHARE salesshare SET PUBLICACCESSIBLE FALSE;
```

7. 복사된 SQL 문을 실행하려면 쿼리(Queries)를 선택하고 복사된 SQL 문을 쿼리 영역에 붙여 넣습니다. 그런 다음 실행(Run)을 선택합니다.

   다음과 같은 오류가 나타납니다.

```
ALTER DATASHARE salesshare SET PUBLICACCESSIBLE FALSE;
ERROR:  Alter of ADX-managed datashare salesshare requires session variable
 datashare_break_glass_session_var to be set to value 'c670ba4db22f4b'
```

   값 'c670ba4db22f4b'는 권장되지 않는 작업이 발생할 때 Amazon Redshift가 생성하는 임의의 일회성 값입니다.

8. 다음 샘플 문을 복사하여 쿼리 영역에 붙여 넣습니다. 그런 다음 명령을 실행합니다. SET datashare_break_glass_session_var 명령은 AWS Data Exchange datashare에 대해 권장되지 않는 작업을 허용하는 권한을 적용합니다.

```
SET datashare_break_glass_session_var to 'c670ba4db22f4b';
```

9. ALTER DATASHARE 문을 다시 실행합니다.

```
ALTER DATASHARE salesshare;
```

Amazon Redshift는 변경 사항으로 datashare를 업데이트합니다.

계정에 생성된 AWS Data Exchange datashare 삭제

콘솔을 사용하여 계정에 생성된 AWS Data Exchange datashare를 삭제합니다. 먼저 데이터베이스에 연결하여 계정에 생성된 datashare 목록을 봅니다.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 클러스터(Clusters)를 선택한 다음 클러스터를 선택합니다. 클러스터 세부 정보 페이지가 나타납니다.
3. 탐색기 메뉴에서 편집기(Editor), 쿼리 편집기 v2(Query editor v2)를 차례로 선택합니다.
4. 쿼리 편집기 v2를 처음 사용하는 경우 AWS 계정을 구성합니다. 기본적으로 AWS 소유의 키가 리소스를 암호화하는 데 사용됩니다. AWS 계정 구성에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 AWS 계정 구성 섹션을 참조하세요.
5. AWS Data Exchange datashare가 있는 클러스터에 연결하려면 트리 보기 패널에서 데이터베이스(Database)와 클러스터 이름을 선택합니다. 메시지가 나타나면 연결 파라미터를 입력합니다.
6. 다음 SQL 문을 복사합니다. 다음 예에서는 salesshare datashare를 삭제합니다.

   ```
   DROP DATASHARE salesshare
   ```

7. 복사된 SQL 문을 실행하려면 쿼리(Queries)를 선택하고 복사된 SQL 문을 쿼리 영역에 붙여 넣습니다. 그런 다음 실행(Run)을 선택합니다.

   다음과 같은 오류가 나타납니다.

   ```
   ERROR:  Drop of ADX-managed datashare salesshare requires session variable
    datashare_break_glass_session_var to be set to value '620c871f890c49'
   ```

   값 '620c871f890c49'는 권장되지 않은 작업이 발생할 때 Amazon Redshift가 생성하는 임의의 일회성 값입니다.
8. 다음 샘플 문을 복사하여 쿼리 영역에 붙여 넣습니다. 그런 다음 명령을 실행합니다. SET datashare_break_glass_session_var 명령은 AWS Data Exchange datashare에 대해 권장되지 않는 작업을 허용하는 권한을 적용합니다.

```
SET datashare_break_glass_session_var to '620c871f890c49';
```

9. DROP DATASHARE 문을 다시 실행합니다.

```
DROP DATASHARE salesshare;
```

datashare가 삭제되면 datashare 소비자는 datashare에 대한 액세스 권한을 상실합니다.

공유 AWS Data Exchange datashare를 삭제하면 AWS Data Exchange의 데이터 제품 약관을 위반할 수 있습니다.

SQL 인터페이스를 사용한 읽기 전용 데이터 공유 시작하기

Amazon Redshift를 사용하면 Amazon Redshift 클러스터 전반에서 데이터를 안전하게 공유할 수 있으므로 데이터 소비자가 데이터를 복사하거나 복제하지 않고도 실시간 데이터를 쿼리하고 액세스할 수 있습니다. 데이터 공유를 통해 공유하려는 데이터베이스 객체를 참조하는 생산자 측 객체인 데이터 공유를 생성 및 구성합니다.

AWS 계정 또는 AWS 리전 내 또는 계정 간에 여러 Amazon Redshift 클러스터에서 읽기용으로 데이터를 공유할 수 있습니다.

주제

- AWS 계정 내 데이터에 대한 읽기 액세스 공유
- Amazon Redshift 데이터 공유에서 보기 작업
- 데이터 공유에 데이터 레이크 테이블 추가
- AWS 계정에서 데이터 공유
- AWS 리전에서 데이터 공유
- AWS Data Exchange에서 라이선스가 부여된 Amazon Redshift 데이터 공유
- AWS Lake Formation 관리형 데이터 공유 시작하기

AWS 계정 내 데이터에 대한 읽기 액세스 공유

Amazon Redshift를 사용하면 동일한 AWS 계정 내의 여러 데이터베이스 사용자 또는 그룹 전반에서 데이터에 대한 읽기 액세스를 공유할 수 있습니다. 이 기능을 사용하면 세분화된 수준에서 데이터 액세스 권한을 제어할 수 있으므로 권한이 부여된 사용자 또는 그룹만 특정 데이터세트를 읽게 됩니다.

생산자 관리자 또는 데이터베이스 소유자로서 읽기 목적으로 데이터 공유

1. 클러스터에서 datashare를 생성합니다. 자세한 내용은 CREATE DATASHARE 단원을 참조하십시오.

   ```
   CREATE DATASHARE salesshare;
   ```

   클러스터 슈퍼 사용자와 데이터베이스 소유자는 datashare를 생성할 수 있습니다. 각 datashare는 생성하는 동안 데이터베이스와 연결됩니다. 해당 데이터베이스의 객체만 해당 datashare에서 공유할 수 있습니다. 동일하거나 다른 세부 수준의 객체를 사용하여 동일한 데이터베이스에 여러 datashare를 생성할 수 있습니다. 클러스터가 생성할 수 있는 datashare 수에는 제한이 없습니다.

   Amazon Redshift 콘솔을 사용하여 datashare를 생성할 수도 있습니다. 자세한 내용은 데이터 공유 만들기 단원을 참조하십시오.

2. datashare에 대한 작업 권한을 위임합니다. 자세한 내용은 GRANT 또는 REVOKE을 참조하세요.

   다음 예에서는 `salesshare`의 `dbuser`에 권한을 부여합니다.

   ```
   GRANT ALTER, SHARE ON DATASHARE salesshare TO dbuser;
   ```

   클러스터 슈퍼 사용자와 datashare 소유자는 datashare에 대한 수정 권한을 추가 사용자에게 부여하거나 취소할 수 있습니다.

3. datashare에 객체를 추가하거나 제거합니다. datashare에 객체를 추가하려면 객체를 추가하기 전에 스키마를 추가합니다. 스키마를 추가할 때 Amazon Redshift는 스키마 아래에 모든 객체를 추가하지 않습니다. 이러한 항목을 명시적으로 추가해야 합니다. 자세한 내용은 ALTER DATASHARE 단원을 참조하십시오.

   ```
   ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
   ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;
   ALTER DATASHARE salesshare ADD ALL TABLES IN SCHEMA PUBLIC;
   ```

   datashare에 뷰를 추가할 수도 있습니다.

   ```
   CREATE VIEW public.sales_data_summary_view AS SELECT * FROM
    public.tickit_sales_redshift;
   ALTER DATASHARE salesshare ADD TABLE public.sales_data_summary_view;
   ```

ALTER DATASHARE를 사용하여 지정된 스키마에서 스키마, 테이블, 뷰 및 함수를 공유합니다. 슈퍼 사용자, datashare 소유자 또는 datashare에 대한 ALTER 또는 ALL 권한이 있는 사용자는 datashare를 변경하여 객체를 추가하거나 제거할 수 있습니다. 사용자는 datashare에서 객체를 추가하거나 제거할 수 있는 권한이 있어야 합니다. 사용자는 객체 소유자이거나 객체에 대한 SELECT, USAGE 또는 ALL 권한도 있어야 합니다.

GRANT를 사용하여 데이터 공유에 객체를 추가할 수도 있습니다. 방법은 아래 예시와 같습니다.

```
GRANT SELECT ON TABLE public.tickit_sales_redshift TO DATASHARE salesshare;
```

이 구문은 기능면에서 `ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;`와 동일합니다.

INCLUDENEW 절을 사용하여 지정된 스키마에서 생성된 새 테이블, 뷰 또는 SQL 사용자 정의 함수(UDF)를 datashare에 추가합니다. 슈퍼 사용자만 각 datashare-스키마 페어에 대해 이 속성을 변경할 수 있습니다.

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare SET INCLUDENEW = TRUE FOR SCHEMA PUBLIC;
```

Amazon Redshift 콘솔을 사용하여 datashare에서 객체를 추가하거나 제거할 수도 있습니다. 자세한 내용은 데이터 공유에 데이터 공유 객체 추가, datashare에서 datashare 객체 제거, 계정에 생성된 datashare 편집 섹션을 참조하세요.

4. datashare에서 소비자를 추가하거나 제거합니다. 다음 예에서는 소비자 네임스페이스를 `salesshare`에 추가합니다. 네임스페이스는 계정에 있는 소비자 클러스터의 네임스페이스 전역 고유 식별자(GUID)입니다. 자세한 내용은 GRANT 또는 REVOKE을 참조하세요.

```
GRANT USAGE ON DATASHARE salesshare TO NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

GRANT 문에서 한 명의 datashare 소비자에게만 권한을 부여할 수 있습니다.

클러스터 슈퍼 사용자와 데이터 공유 객체의 소유자 또는 데이터 공유에 대한 SHARE 권한이 있는 사용자는 데이터 공유에서 소비자를 추가하거나 제거할 수 있습니다. 이를 위해 GRANT USAGE 또는 REVOKE USAGE를 사용합니다.

현재 보고 있는 클러스터의 네임스페이스를 찾으려면 SELECT CURRENT_NAMESPACE 명령을 사용합니다. 동일한 AWS 계정 내에서 다른 클러스터의 네임스페이스를 찾으려면 Amazon Redshift 콘솔 클러스터 세부 정보 페이지로 이동합니다. 해당 페이지에서 새로 추가된 네임스페이스 필드를 찾습니다.

Amazon Redshift 콘솔을 사용하여 datashare에 대한 데이터 소비자를 추가하거나 제거할 수도 있습니다. 자세한 내용은 데이터 공유에 데이터 소비자 추가 및 datashare에서 데이터 소비자 제거 섹션을 참조하세요.

5. (옵션) datashare에 보안 제한을 추가합니다. 다음 예에서는 퍼블릭 IP 액세스 권한이 있는 소비자 클러스터가 datashare를 읽을 수 있음을 보여줍니다. 자세한 내용은 ALTER DATASHARE 단원을 참조하십시오.

```
ALTER DATASHARE salesshare SET PUBLICACCESSIBLE = TRUE;
```

datashare 생성 후 소비자 유형에 대한 속성을 수정할 수 있습니다. 예를 들어 지정된 datashare의 데이터를 소비하려는 클러스터에 공개적으로 액세스할 수 없도록 정의할 수 있습니다. datashare에 지정된 보안 제한을 충족하지 않는 소비자 클러스터의 쿼리는 쿼리 런타임에 거부됩니다.

Amazon Redshift 콘솔을 사용하여 datashare를 편집할 수도 있습니다. 자세한 내용은 계정에 생성된 datashare 편집 단원을 참조하십시오.

6. 클러스터에서 생성된 datashare를 나열하고 datashare의 내용을 살펴봅니다.

다음 예에서는 `salesshare`라는 datashare의 정보를 보여줍니다.

```
DESC DATASHARE salesshare;

 producer_account  |          producer_namespace          | share_type | share_name
 | object_type |           object_name          |   include_new
-------------------+--------------------------------------+------------
+------------+-------------+----------------------------------+-------------------
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_users_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_venue_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_category_redshift|
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_date_redshift    |
```

```
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_event_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_listing_redshift |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_sales_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | schema      | public                         |  t
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | view        | public.sales_data_summary_view |
```

다음 예에서는 생산자 클러스터의 아웃바운드 datashare를 보여줍니다.

```
SHOW DATASHARES LIKE 'sales%';
```

출력 결과는 다음과 비슷합니다.

```
share_name | share_owner  | source_database | consumer_database | share_type |
 createdate      | is_publicaccessible  | share_acl | producer_account |
 producer_namespace
-----------+--------------+-----------------+-------------------+------------
+---------------------+----------------------+-----------+-----------------
+--------------------------------------
salesshare |    100       | dev             |                   |  OUTBOUND
 | 2020-12-09 02:27:08 |          True        |           |   123456789012   |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d
```

자세한 내용은 DESC DATASHARE 및 SHOW DATASHARES 섹션을 참조하세요.

SVV_DATASHARES, SVV_DATASHARE_CONSUMERS, 및 SVV_DATASHARE_OBJECTS를 사용하여 datashare, datashare 내의 객체 및 datashare 소비자를 볼 수도 있습니다.

7. datashare를 삭제합니다. 자세한 내용은 DROP DATASHARE 단원을 참조하십시오.

   DROP DATASHARE를 사용하여 언제든지 datashare 객체를 삭제할 수 있습니다. 클러스터 슈퍼 사용자와 datashare 소유자는 datashare를 삭제할 수 있습니다.

   다음 예에서는 `salesshare`라는 datashare를 삭제합니다.

   ```
   DROP DATASHARE salesshare;
   ```

Amazon Redshift 콘솔을 사용하여 datashare를 삭제할 수도 있습니다. 자세한 내용은 계정에 만들어진 데이터 공유 삭제 단원을 참조하십시오.

8. ALTER DATASHARE를 사용하여 datashare에서 언제든지 객체를 제거합니다. REVOKE USAGE ON을 사용하여 특정 소비자에 대한 datashare 권한을 취소합니다. datashare 내의 객체에 대한 USAGE 권한을 취소하고 모든 소비자 클러스터에 대한 액세스를 즉시 중지합니다. 데이터베이스 및 테이블 나열 등의 datashare 나열과 메타데이터 쿼리는 액세스가 취소된 후 공유 객체를 반환하지 않습니다.

```
ALTER DATASHARE salesshare REMOVE TABLE public.tickit_sales_redshift;
```

Amazon Redshift 콘솔을 사용하여 datashare를 편집할 수도 있습니다. 자세한 내용은 계정에 생성된 datashare 편집 단원을 참조하십시오.

9. 더 이상 소비자와 데이터를 공유하지 않으려면 네임스페이스에서 datashare에 대한 액세스를 취소합니다.

```
REVOKE USAGE ON DATASHARE salesshare FROM NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

Amazon Redshift 콘솔을 사용하여 datashare를 편집할 수도 있습니다. 자세한 내용은 계정에 생성된 datashare 편집 단원을 참조하십시오.

소비자 관리자로서 읽기 목적으로 데이터 공유

1. 사용할 수 있는 datashare를 나열하고 datashare의 내용을 봅니다. 자세한 내용은 DESC DATASHARE 및 SHOW DATASHARES 섹션을 참조하세요.

   다음 예에서는 지정된 생산자 네임스페이스의 인바운드 datashare 정보를 보여줍니다. 소비자 관리자로 DESC DATASHARE를 실행할 때 인바운드 데이터 공유를 보려면 NAMESPACE 옵션을 지정해야 합니다.

```
DESC DATASHARE salesshare OF NAMESPACE '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';


 producer_account  |          producer_namespace          | share_type | share_name
 | object_type |           object_name           |   include_new
-------------------+--------------------------------------+------------
+------------+-------------+----------------------------------+------------------
```

```
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_users_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_venue_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_category_redshift |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_date_redshift     |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_event_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_listing_redshift  |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_sales_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | schema      | public                          |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | view        | public.sales_data_summary_view  |
```

클러스터 슈퍼 사용자만 이 작업을 수행할 수 있습니다. SVV_DATASHARES를 사용하여 datashare를 보고 SVV_DATASHARE_OBJECTS를 사용하여 datashare 내의 객체를 볼 수도 있습니다.

다음 예에서는 소비자 클러스터의 인바운드 datashare를 보여줍니다.

```
SHOW DATASHARES LIKE 'sales%';


 share_name | share_owner | source_database | consumer_database | share_type
 | createdate | is_publicaccessible | share_acl | producer_account |
 producer_namespace
------------+-------------+-----------------+-------------------+------------
+------------+---------------------+-----------+------------------
+--------------------------------------
 salesshare |             |                 |                   | INBOUND
    |            |         t           |           |   123456789012   |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d
```

2. Datashare 슈퍼 사용자는 datashare를 참조하는 로컬 데이터베이스를 생성할 수 있습니다. 자세한 내용은 데이터베이스 생성 단원을 참조하십시오.

```
CREATE DATABASE sales_db FROM DATASHARE salesshare OF NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

로컬 데이터베이스의 객체에 대한 액세스를 보다 세밀하게 제어하려면 데이터베이스를 만들 때 WITH PERMISSIONS 절을 사용하세요. 이렇게 하면 4단계에서 데이터베이스의 객체에 객체 수준 권한을 부여할 수 있습니다.

```
CREATE DATABASE sales_db WITH PERMISSIONS FROM DATASHARE salesshare OF NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

SVV_REDSHIFT_DATABASES 뷰를 쿼리하여 datashare에서 생성한 데이터베이스를 볼 수 있습니다. 그러나 소비자 클러스터의 로컬 데이터베이스에 연결하고 데이터베이스 간 쿼리를 수행하여 데이터 공유에서 만들어진 데이터베이스의 데이터를 쿼리할 수 있습니다. 기존 datashare에서 생성된 데이터베이스 객체 위에 datashare를 생성할 수 없습니다. 그러나 데이터를 소비자 클러스터의 별도 테이블에 복사하고 필요한 처리를 수행한 다음 생성된 새 객체를 공유할 수 있습니다.

Amazon Redshift 콘솔을 사용하여 datashare에서 데이터베이스를 생성할 수도 있습니다. 자세한 내용은 datashare에서 데이터베이스 생성 단원을 참조하십시오.

3. (옵션) 소비자 클러스터에서 가져온 소비자 데이터베이스의 특정 스키마를 참조하고 세분화된 권한을 할당할 외부 스키마를 생성합니다. 자세한 내용은 CREATE EXTERNAL SCHEMA 단원을 참조하십시오.

```
CREATE EXTERNAL SCHEMA sales_schema FROM REDSHIFT DATABASE 'sales_db' SCHEMA
 'public';
```

4. 필요에 따라 소비자 클러스터의 사용자 및 역할에 데이터 공유에서 생성된 데이터베이스 및 스키마 참조에 대한 권한을 부여합니다. 자세한 내용은 GRANT 또는 REVOKE을 참조하세요.

```
GRANT USAGE ON DATABASE sales_db TO Bob;
```

```
GRANT USAGE ON SCHEMA sales_schema TO ROLE Analyst_role;
```

WITH PERMISSIONS를 사용하지 않고 데이터베이스를 생성한 경우 데이터 공유에서 생성된 전체 데이터베이스에 대한 권한만 사용자와 역할에 할당할 수 있습니다. 경우에 따라 datashare에서 생성된 데이터베이스 객체의 하위 집합에 대한 세분화된 제어가 필요합니다. 이러한 경우 이전 단

계에서 설명한 대로 datashare의 특정 스키마를 가리키는 외부 스키마 참조를 생성하고 스키마 수준에서 세분화된 권한을 제공할 수 있습니다.

또한 공유 객체 위에 후기 바인딩 뷰를 생성하고 이를 사용하여 세분화된 권한을 할당할 수 있습니다. 생산자 클러스터가 필요한 세분성으로 추가 datashare를 생성하도록 할 수도 있습니다.

2단계에서 WITH PERMISSIONS를 사용하여 데이터베이스를 만든 경우 공유 데이터베이스의 객체에 객체 수준 권한을 할당해야 합니다. USAGE 권한만 있는 사용자는 추가 객체 수준 권한을 부여받을 때까지 WITH PERMISSION을 사용하여 만든 데이터베이스의 객체에 액세스할 수 없습니다.

```
GRANT SELECT ON sales_db.public.tickit_sales_redshift to Bob;
```

5. datashare의 공유 객체에서 데이터를 쿼리합니다.

   소비자 데이터베이스 및 소비자 클러스터의 스키마에 대한 권한이 있는 사용자와 역할은 모든 공유 객체의 메타데이터를 탐색할 수 있습니다. 또한 소비자 클러스터에서 로컬 객체를 탐색할 수 있습니다. 이를 위해 JDBC 또는 ODBC 드라이버 또는 SVV_ALL 및 SVV_REDSHIFT 뷰를 사용합니다.

   생산자 클러스터의 경우 각 스키마 내의 데이터베이스, 테이블 및 뷰에 많은 스키마가 있을 수 있습니다. 소비자 측의 사용자는 datashare를 통해 사용할 수 있는 객체의 하위 집합만 볼 수 있습니다. 이러한 사용자는 생산자 클러스터에서 전체 메타데이터를 볼 수 없습니다. 이 접근 방식은 datashare로 세분화된 메타데이터 보안 제어를 제공하는 데 도움이 됩니다.

   로컬 클러스터 데이터베이스에 계속 연결합니다. 그러나 이제 세 부분으로 구성된 database.schema.table 표기법을 사용하여 datashare에서 생성된 데이터베이스와 스키마에서도 읽을 수도 있습니다. 사용자에게 표시되는 모든 데이터베이스에 걸쳐 쿼리를 수행할 수 있습니다. 이러한 데이터베이스는 클러스터의 로컬 데이터베이스이거나 datashare에서 생성된 데이터베이스일 수 있습니다. 소비자 클러스터는 datashare에서 생성된 데이터베이스에 연결할 수 없습니다.

   전체 자격을 사용하여 데이터에 액세스할 수 있습니다. 자세한 내용은 데이터베이스 간 쿼리 예제 단원을 참조하십시오.

```
SELECT * FROM sales_db.public.tickit_sales_redshift ORDER BY 1,2 LIMIT 5;

 salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
 commission |      saletime
---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------
```

```
     1 |      1 |    36861 |   21191 |    7872 |   1875 |       4 |    728.00 |
 109.20 | 2008-02-18 02:36:48
     2 |      4 |     8117 |   11498 |    4337 |   1983 |       2 |     76.00 |
  11.40 | 2008-06-06 05:00:16
     3 |      5 |     1616 |   17433 |    8647 |   1983 |       2 |    350.00 |
  52.50 | 2008-06-06 08:26:17
     4 |      5 |     1616 |   19715 |    8647 |   1986 |       1 |    175.00 |
  26.25 | 2008-06-09 08:38:52
     5 |      6 |    47402 |   14115 |    8240 |   2069 |       2 |    154.00 |
  23.10 | 2008-08-31 09:17:02
```

공유 객체에는 SELECT 문만 사용할 수 있습니다. 그러나 다른 로컬 데이터베이스에 있는 공유 객체의 데이터를 쿼리하여 소비자 클러스터에 테이블을 생성할 수 있습니다.

쿼리 외에도 소비자는 공유 객체에 대한 뷰를 생성할 수 있습니다. 후기 바인딩 뷰나 구체화된 뷰만 지원됩니다. Amazon Redshift는 공유 데이터에 대한 일반 뷰를 지원하지 않습니다. 소비자가 생성하는 뷰는 여러 로컬 데이터베이스 또는 datashare에서 생성된 데이터베이스에 걸쳐 있을 수 있습니다. 자세한 내용은 CREATE VIEW 단원을 참조하십시오.

```
// Connect to a local cluster database

// Create a view on shared objects and access it.
CREATE VIEW sales_data
AS SELECT *
FROM sales_db.public.tickit_sales_redshift
WITH NO SCHEMA BINDING;

SELECT * FROM sales_data;
```

Amazon Redshift 데이터 공유에서 보기 작업

생산자 클러스터는 일반, 후기 바인딩 및 구체화된 뷰를 공유할 수 있습니다. 일반 뷰 또는 후기 바인딩 뷰를 공유할 때 기본 테이블을 공유할 필요가 없습니다. 다음 표에서는 datashare에서 뷰가 지원되는 방식을 보여줍니다.

뷰 이름	datashare에 이 뷰를 추가할 수 있습니까?	소비자가 클러스터 전체의 datashare 객체에 대해 이 뷰를 생성할 수 있습니까?
일반 뷰	예	No
후기 바인딩 보기	예	예
구체화된 뷰	예	예, 그러나 전체 새로 고침을 수행하는 경우에만 가능합니다.

다음 쿼리는 datashare와 함께 지원되는 일반 뷰의 출력을 보여줍니다. 일반 뷰 정의에 대한 자세한 내용은 CREATE VIEW 섹션을 참조하세요.

```
SELECT * FROM tickit_db.public.myevent_regular_vw
ORDER BY eventid LIMIT 5;

   eventid  |  eventname
  ----------+-------------
     3835   | LeAnn Rimes
     3967   | LeAnn Rimes
     4856   | LeAnn Rimes
     4948   | LeAnn Rimes
     5131   | LeAnn Rimes
```

다음 쿼리는 데이터 공유와 함께 지원되는 후기 바인딩 보기의 출력을 보여줍니다. 후기 바인딩 보기 정의에 대한 자세한 내용은 CREATE VIEW 섹션을 참조하세요.

```
SELECT * FROM tickit_db.public.event_lbv
ORDER BY eventid LIMIT 5;

 eventid | venueid | catid | dateid |          eventname           |      starttime
 --------+---------+-------+--------+------------------------------
+---------------------
     1   |   305   |   8   |  1851  |        Gotterdammerung       | 2008-01-25
 14:30:00
     2   |   306   |   8   |  2114  |           Boris Godunov      | 2008-10-15
 20:00:00
```

```
    3   |   302   |   8   |  1935  |              Salome          | 2008-04-19
14:30:00
    4   |   309   |   8   |  2090  |  La Cenerentola (Cinderella) | 2008-09-21
14:30:00
    5   |   302   |   8   |  1982  |          Il Trovatore        | 2008-06-05
19:00:00
```

다음 쿼리는 datashare와 함께 지원되는 구체화된 뷰의 출력을 보여줍니다. 구체화된 뷰 정의에 대한 자세한 내용은 CREATE MATERIALIZED VIEW 섹션을 참조하세요.

```
SELECT * FROM tickit_db.public.tickets_mv;

 catgroup | qtysold
----------+---------
 Concerts | 195444
   Shows  | 149905
```

생산자 클러스터의 모든 테넌트에서 공통 테이블을 유지 관리할 수 있습니다. 또한 `tenant_id`(`account_id` 또는 `namespace_id`)와 같은 차원 열로 필터링된 데이터의 하위 집합을 소비자 클러스터와 공유할 수 있습니다. 이를 위해 이러한 ID 열(예: `current_aws_account = tenant_id`)에 대한 필터를 사용하여 기본 테이블에 대한 뷰를 정의할 수 있습니다. 소비자 측에서 뷰를 쿼리하면 계정에 적합한 행만 표시됩니다. 이를 위해 Amazon Redshift 컨텍스트 함수 `current_aws_account` 및 `current_namespace`를 사용할 수 있습니다.

다음 쿼리는 현재 Amazon Redshift 클러스터가 있는 계정 ID를 반환합니다. Amazon Redshift에 연결된 경우 이 쿼리를 실행할 수 있습니다.

```
select current_user, current_aws_account;

current_user | current_aws_account
-------------+--------------------
dwuser       |    111111111111
(1row)
```

다음 쿼리는 현재 Amazon Redshift 클러스터의 네임스페이스를 반환합니다. 데이터베이스에 연결되어 있으면 이 쿼리를 실행할 수 있습니다.

```
select current_user, current_namespace;

current_user | current_namespace
```

```
-------------+--------------------------------------
dwuser       | 86b5169f-01dc-4a6f-9fbb-e2e24359e9a8
(1 row)
```

데이터 공유의 구체화된 뷰에 대한 증분 새로 고침

Amazon Redshift는 기본 테이블이 공유될 때 소비자 데이터 공유의 구체화된 뷰에 대한 증분 새로 고침을 지원합니다. 증분 새로 고침은 Amazon Redshift가 이전 새로 고침 이후에 발생한 기본 테이블의 변경 사항을 식별하고 구체화된 뷰의 해당 레코드만 업데이트하는 작업입니다. 이 동작에 대한 자세한 내용은 구체화된 뷰 생성을 참조하세요.

데이터 공유에 데이터 레이크 테이블 추가

데이터 공유를 사용하면 데이터 생산자는 스키마 및 테이블 등의 세분화된 데이터베이스 객체를 동일한 AWS 계정 또는 다른 계정의 소비자와 안전하게 공유할 수 있습니다. 또한 생산자가 리전 전반에서 객체를 공유할 수 있습니다. 이 주제에서는 데이터 레이크, 특히 AWS Glue 데이터 카탈로그에서 데이터 공유로 객체를 추가하는 방법을 설명합니다. 여기에는 2가지 사용 사례가 포함됩니다.

- 데이터 레이크의 테이블을 참조하는 데이터 공유에 지연 바인딩 뷰 추가 - 이는 Lake Formation과 같은 외부 소스 데이터에 대한 권한 정의 등의 예비 구성이 이미 완료되었을 가능성이 높으므로 소비자에게 편리합니다. 또 다른 이점은 데이터 공유에 추가된 뷰가 Redshift 네이티브 테이블을 사용하여 데이터 레이크의 테이블을 조인할 수 있다는 것입니다.
- 외부 스키마의 테이블을 데이터 공유에 직접 추가 - 이렇게 하면 추가 계층이나 로직 없이 데이터 레이크의 객체를 소비자가 사용할 수 있습니다. 소비자는 테이블을 쿼리하거나 소비자의 테이블과 조인할 수 있습니다.

이러한 사례는 CREATE EXTERNAL SCHEMA를 사용하여 Redshift의 AWS 데이터 카탈로그에서 테이블을 참조한 후에 적용됩니다. AWS 데이터 카탈로그의 모든 테이블이 소스일 수 있습니다.

> **Note**
>
> 데이터 공유에 추가하는 데이터 레이크 테이블에는 Lake Formation에 등록된 테이블과 AWS Glue 데이터 카탈로그 테이블이 포함될 수 있습니다.

외부 스키마와 외부 테이블 생성

외부 스키마와 외부 테이블을 생성하여 다음 섹션의 데이터 공유에 추가합니다. 이 과정은 예비 단계입니다. 이미 완료했으면 이 섹션을 건너뜁니다.

1. 생산자의 경우 Amazon S3에 저장된 데이터 레이크 데이터를 참조하는 외부 스키마를 생성합니다. 외부 스키마는 AWS Glue Data Catalog를 참조합니다. 샘플의 역할 및 리전은 다음과 같습니다.

```
CREATE EXTERNAL SCHEMA external_schema_name FROM DATA CATALOG
DATABASE 'glue_database_name'
IAM_ROLE 'arn:aws:iam::123456789012:role/sample-role'
REGION 'us-east-1';
```

2. 외부 스키마에서 데이터 레이크 테이블을 생성합니다.

```
CREATE EXTERNAL TABLE external_schema_name.sales(
salesid INTEGER,
sellerid INTEGER,
buyerid INTEGER,
saledate DATE,
pricepaid DECIMAL(8,2))
ROW FORMAT delimited
FIELDS TERMINATED BY '\t'
STORED AS textfile
LOCATION 's3://redshift-downloads/tickit/spectrum/sales/';
```

   샘플 코드에는 LOCATION이 포함됩니다. 폴더가 지정된 `s3://{bucket_name}/{folder}/` 형식이어야 합니다. 폴더의 길이는 최소 문자 한 개 이상으로 구성되어야 합니다. 필요에 따라 하위 폴더를 포함할 수 있습니다. 데이터 레이크에서 테이블을 생성하는 추가 예제를 보려면 CREATE EXTERNAL TABLE 예제를 참조하시기 바랍니다.

> **Note**
>
> 공유는 생산자의 IAM 역할이 테이블에 대해 SELECT 액세스 권한을 갖춘 테이블에만 지원됩니다.

데이터 레이크 테이블을 데이터 공유에 참조하는 지연 바인딩 뷰 추가

AWS 데이터 카탈로그에서 외부 스키마를 기반으로 테이블을 생성하고 데이터 공유에 테이블을 추가하려는 경우 가장 일반적인 방법은 데이터 레이크의 데이터를 포함하여 생성한 테이블을 참조하는 Redshift 지연 바인딩 뷰를 추가하는 것입니다. 절차에 포함되는 단계는 다음과 같습니다.

1. 이전에 생성한 외부 테이블을 참조하는 지연 바인딩 뷰를 생성합니다.

```
CREATE VIEW lbv AS
select * from external_schema_name.sales, other_schema.t1
WITH NO SCHEMA BINDING;
```

2. 데이터 공유에 뷰 스키마를 추가합니다. 이는 지연 바인딩 뷰가 포함된 로컬 스키마입니다.

```
ALTER DATASHARE dsx_datashare ADD SCHEMA public;
```

3. 지연 바인딩 뷰에서 참조하는 테이블이 포함된 스키마를 데이터 공유에 추가합니다. 스키마에 로컬 데이터베이스 객체 또는 데이터 레이크의 객체가 포함되어 있는지 여부에 관계없이 데이터 공유에 추가된 뷰에서 참조된 기본 테이블에 스키마를 추가해야 합니다. 지연 바인딩 뷰를 추가하기 전에 먼저 이 스키마를 추가해야 합니다.

```
ALTER DATASHARE dsx_datashare ADD SCHEMA external_schema_name;
ALTER DATASHARE dsx_datashare ADD SCHEMA other_schema;
```

4. SQL 명령을 사용하여 데이터 공유에 뷰를 추가합니다. 테이블 이름에 스키마 접두사가 포함된 것을 알 수 있습니다.

```
ALTER DATASHARE my_datashare ADD TABLE public.lbv;
```

5. 뷰 및 스키마가 데이터 공유에 성공적으로 추가되었는지 확인합니다.

```
SELECT * FROM svv_datashare_objects WHERE share_name = 'my_datashare';
```

6. 소비자 관리자는 데이터 공유에서 데이터베이스를 생성한 다음, 소비자 사용자에게 사용량을 부여합니다.

단계를 완료한 후 데이터 공유 뷰에 액세스할 수 있는 데이터베이스 소비자 사용자가 데이터를 쿼리할 수 있습니다.

데이터 레이크 테이블을 데이터 공유에 직접 추가

외부 스키마의 테이블을 데이터 공유에 추가하는 작업은 뷰를 추가하는 과정과 유사합니다. 이는 소비자가 초기 상태에서 데이터 레이크 테이블을 쿼리하려는 경우 또는 소비자가 소비자 데이터 웨어하우스의 테이블에 조인하려는 경우에 적합합니다. 다음 단계에서는 SQL을 사용하여 데이터 공유에 데이터 레이크 테이블을 추가하는 방법을 보여 줍니다.

1. 이 주제의 첫 번째 섹션에 설명된 대로 외부 스키마와 외부 테이블을 생성합니다.

2. 외부 스키마에서 기존 테이블을 검색하여 생성한 테이블을 사용할 수 있는지 확인합니다.

```
SELECT * FROM svv_external_tables WHERE schemaname = 'external_schema_name';
```

3. 외부 스키마를 다음과 같이 데이터 공유에 추가합니다.

```
ALTER DATASHARE my_datashare ADD SCHEMA external_schema_name;
```

4. 외부 테이블을 데이터 공유에 추가합니다. 테이블 이름에 스키마 접두사가 포함된 것을 알 수 있습니다.

```
ALTER DATASHARE my_datashare ADD TABLE external_schema_name.sales;
```

5. 테이블이 데이터 공유에 성공적으로 추가되었는지 확인합니다.

```
SELECT * FROM svv_datashare_objects WHERE share_name = 'my_datashare';
```

   더 자세한 설명은 AWS 계정 내 데이터에 대한 읽기 액세스 권한 공유를 참조하시기 바랍니다.

6. 공유 데이터를 수신하는 데이터베이스인 소비자에서 관리자는 데이터 공유를 연결하여 사용자가 쿼리할 수 있는 공유 테이블을 제공합니다. 이 단계를 수행하는 방법에 관한 자세한 내용은 소비자가 다른 계정의 데이터 공유 관리를 참조하시기 바랍니다.

관리자가 단계를 완료하면 소비자의 데이터베이스 사용자는 쿼리를 작성하여 공유 테이블에서 데이터를 검색하고 소비자의 다른 테이블과 이를 조인할 수 있습니다.

데이터 공유에 데이터 레이크 객체를 추가하기 위한 사용 정보

데이터 공유의 데이터 레이크에서 테이블과 뷰 사용 시 알아야 할 몇 가지 사항이 있습니다.

- AWS CloudTrail과 로깅 - 데이터 공유를 통해 공유된 데이터 레이크 테이블에 액세스할 때 데이터 생산자 계정은 AWS CloudTrail 로그를 사용하여 감사할 수 있습니다.
  - 로그 데이터를 사용하여 데이터 액세스 제어 - CloudTrail 로그는 Redshift 데이터 공유 생산자와 소비자를 비롯해 공유 테이블에 액세스하는 사용자에 대한 세부 정보를 기록합니다. 식별자는 `AssumeRole` CloudTrail 로그 아래의 `ExternalId` 필드에서 사용할 수 있습니다. 데이터 소유자는 작업을 통해 IAM 정책의 데이터 액세스에 대한 추가 제한 사항을 구성할 수 있습니다. 정책을 통한 데이터 액세스 정의에 관한 자세한 내용은 서드 파티가 소유한 AWS 계정 액세스를 참조하시기 바랍니다.

- 보안 및 소비자 권한 - Lake Formation 등록 테이블의 경우 Amazon S3 리소스는 Lake Formation에서 보호되며 Lake Formation에서 제공하는 자격 증명 인증을 사용하여 이용 가능하도록 제공됩니다.

데이터 레이크 객체를 데이터 공유에 추가하기 위한 결제 고려 사항

다음은 데이터 공유에서 데이터 레이크 객체를 저장하고 스캔하는 데 드는 비용을 계산하는 방법을 자세히 설명합니다.

- 소비자가 데이터 레이크에서 공유 객체를 쿼리하면 스캔 비용이 소비자에게 부과됩니다.
  - 소비자가 프로비저닝 클러스터이면 Redshift는 Redshift Spectrum을 사용하여 Amazon S3 데이터를 스캔합니다. 따라서 Spectrum 비용이 소비자 계정으로 청구됩니다.
  - 소비자가 Amazon Redshift Serverless 작업 그룹이면 Spectrum에 따로 요금이 부과되지 않습니다.
- 버킷 나열 등의 스토리지 및 작업에 대한 Amazon S3 비용은 각 Amazon S3 버킷을 소유한 계정으로 청구됩니다.

Amazon Redshift Serverless 결제에 관한 자세한 내용은 Amazon Redshift Serverless 결제를 참고하시기 바랍니다. Amazon Redshift 요금에서 더 많은 결제 및 요금 정보를 확인할 수 있습니다.

AWS 계정에서 데이터 공유

AWS 계정 간에 읽기용으로 데이터를 공유할 수 있습니다. AWS 계정 간 데이터 공유는 한 계정 내 데이터 공유와 유사합니다. 차이점은 AWS 계정 간 데이터 공유에는 양방향 핸드셰이크가 필요하다는 것입니다. 생산자 계정 관리자는 소비자 계정에 datashare 액세스 권한을 부여하거나 어떠한 액세스 권한도 부여하지 않도록 선택할 수 있습니다. 권한이 부여된 datashare를 사용하기 위해 소비자 계정 관리자는 datashare를 연결할 수 있습니다. 관리자는 데이터 공ㅇ를 전체 AWS 계정 계정 또는 소비자 계정의 특정 클러스터와 연결하거나 데이터 공유를 거부할 수 있습니다. 계정 내 datashare에 대한 자세한 내용은 AWS 계정 내 데이터에 대한 읽기 액세스 공유 섹션을 참조하세요.

데이터 공유에는 동일한 계정 또는 다른 AWS 계정의 네임스페이스인 데이터 소비자가 있을 수 있습니다. 계정 내 공유와 계정 간 공유를 위해 별도의 datashare를 생성할 필요가 없습니다.

계정 간 datashare의 경우 생산자 및 소비자 클러스터를 모두 암호화해야 합니다.

AWS 계정과 데이터를 공유할 때 생산자 관리자는 AWS 계정과 엔터티로 공유합니다. 소비자 관리자는 소비자 계정의 어떤 네임스페이스가 데이터 공유에 액세스할 수 있는지 결정할 수 있습니다.

주제

- 생산자 관리자 작업
- 소비자 계정 관리자 작업
- 소비자 관리자 작업

생산자 관리자 작업

Amazon Redshift로 생산자 클러스터에서 관리 작업을 수행하여 데이터 수집 및 로드 처리를 관리합니다.

생산자 관리자 또는 데이터베이스 소유자인 경우 - 다음 단계를 따릅니다.

1. 클러스터에 datashare를 생성하고 datashare에 datashare 객체를 추가합니다. datashare를 생성하고 datashare 객체를 datashare에 추가하는 방법에 대한 자세한 단계는 AWS 계정 내 데이터에 대한 읽기 액세스 공유 섹션을 참조하세요. CREATE DATASHARE 및 ALTER DATASHARE에 대한 자세한 내용은 CREATE DATASHARE 및 ALTER DATASHARE 섹션을 참조하세요.

   다음 예에서는 datashare `salesshare`에 다른 datashare 객체를 추가합니다.

   ```
   -- Add schema to datashare
   ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;

   -- Add table under schema to datashare
   ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;

   -- Add view to datashare
   ALTER DATASHARE salesshare ADD TABLE public.sales_data_summary_view;

   -- Add all existing tables and views under schema to datashare (does not include
    future table)
   ALTER DATASHARE salesshare ADD ALL TABLES in schema public;
   ```

   Amazon Redshift 콘솔을 사용하여 datashare를 생성하거나 편집할 수도 있습니다. 자세한 내용은 데이터 공유 만들기 및 계정에 생성된 datashare 편집 섹션을 참조하세요.
2. datashare에 대한 작업 권한을 위임합니다. 자세한 내용은 GRANT 또는 REVOKE을 참조하세요.

   다음 예에서는 `salesshare`의 `dbuser`에 권한을 부여합니다.

   ```
   GRANT ALTER, SHARE ON DATASHARE salesshare TO dbuser;
   ```

클러스터 슈퍼 사용자와 datashare 소유자는 datashare에 대한 수정 권한을 추가 사용자에게 부여하거나 취소할 수 있습니다.

3. datashare에서 소비자를 추가하거나 제거합니다. 다음 예에서는 `salesshare`에 AWS 계정 ID를 추가합니다. 자세한 내용은 GRANT 또는 REVOKE을 참조하세요.

```
GRANT USAGE ON DATASHARE salesshare TO ACCOUNT '123456789012';
```

GRANT 문에서 한 명의 datashare 소비자에게만 권한을 부여할 수 있습니다.

클러스터 슈퍼 사용자와 datashare 객체의 소유자 또는 datashare에 대한 SHARE 권한이 있는 사용자는 datashare에서 소비자를 추가하거나 제거할 수 있습니다. 이를 위해 GRANT USAGE 또는 REVOKE USAGE를 사용합니다.

Amazon Redshift 콘솔을 사용하여 datashare에 대한 데이터 소비자를 추가하거나 제거할 수도 있습니다. 자세한 내용은 데이터 공유에 데이터 소비자 추가 및 datashare에서 데이터 소비자 제거 단원을 참조하세요.

4. (옵션) 더 이상 소비자와 데이터를 공유하지 않으려면 AWS 계정에서 datashare에 대한 액세스를 취소합니다.

```
REVOKE USAGE ON DATASHARE salesshare FROM ACCOUNT '123456789012';
```

생산자 계정 관리자인 경우 다음 절차를 따릅니다.

AWS 계정에 사용 권한을 부여한 후 datashare 상태는 `pending_authorization`입니다. 생산자 계정 관리자는 Amazon Redshift 콘솔을 사용하여 datashare 권한을 부여하고 데이터 소비자를 선택해야 합니다.

https://console.aws.amazon.com/redshiftv2/에 로그인합니다. 그런 다음 datashare에 대한 액세스 권한을 부여하거나 제거할 데이터 소비자를 선택합니다. 권한이 부여된 데이터 소비자는 datashare에 대한 작업을 수행하라는 알림을 받습니다. 네임스페이스를 데이터 소비자로 추가하는 경우 권한 부여를 수행할 필요가 없습니다. 권한을 부여 받은 데이터 소비자는 datashare 객체에 액세스하고 소비자 데이터베이스를 생성하여 데이터를 쿼리할 수 있습니다. 자세한 내용은 datashare에서 권한 부여 또는 제거 단원을 참조하십시오.

데이터에 대한 계정 간 쓰기 권한 공유

Amazon Redshift를 사용하면 AWS 계정 전반에서 데이터를 공유하고 쓰기 권한을 부여하여 팀 또는 조직 간의 협업 및 데이터 공유를 지원할 수 있습니다. 계정 간 데이터 공유를 통해 데이터베이스, 스키마 및 테이블을 만들고 관리하는 데이터 공급자 계정을 설정할 수 있으며, 이 계정에서는 데이터 소비자 계정과 안전하게 공유할 수 있습니다. 다음 섹션에서는 계정 간 데이터 공유를 구성하고 Amazon Redshift에서 쓰기 액세스 권한을 부여하는 프로세스를 보여 줍니다.

소비자 계정 관리자 작업

Amazon Redshift로 소비자 계정을 관리하고 데이터 웨어하우징 리소스에 대한 액세스를 제어합니다.

소비자 계정 관리자인 경우 다음 절차를 따릅니다.

다른 계정에서 공유되는 하나 이상의 데이터 공유를 전체 AWS 계정 또는 계정의 특정 네임스페이스와 연결하려면 Amazon Redshift 콘솔을 사용합니다.

https://console.aws.amazon.com/redshiftv2/에 로그인합니다. 그런 다음 다른 계정에서 공유되는 하나 이상의 데이터 공유를 전체 AWS 계정 계정 또는 계정별 네임스페이스와 연결합니다. 자세한 내용은 Amazon Redshift에서 다른 AWS 계정의 데이터 공유 연결 단원을 참조하십시오.

AWS 계정 또는 특정 네임스페이스가 연결된 후 데이터 공유를 사용할 수 있게 됩니다. datashare 연결을 언제든지 변경할 수도 있습니다. 개별 네임스페이스에서 AWS 계정으로 연결을 변경하면 Amazon Redshift가 네임스페이스를 AWS 계정 정보로 덮어씁니다. AWS 계정에서 특정 네임스페이스로 연결을 변경할 때 Amazon Redshift는 AWS 계정 정보를 네임스페이스 정보로 덮어씁니다. 계정의 모든 네임스페이스는 데이터에 액세스할 수 있습니다.

소비자 관리자 작업

Amazon Redshift로 소비자 클러스터에서 관리 작업을 수행하여 데이터 수집 및 로드 처리를 관리합니다.

소비자 관리자인 경우 - 다음 단계를 따릅니다.

1. 사용할 수 있는 datashare를 나열하고 datashare의 내용을 봅니다. 데이터 공유의 내용은 생산자 관리자가 데이터 공유 권한을 부여받고 소비자 관리자가 데이터 공유를 수락하고 연결한 경우에만 사용할 수 있습니다. 자세한 내용은 DESC DATASHARE 및 SHOW DATASHARES 단원을 참조하세요.

   다음 예에서는 지정된 생산자 네임스페이스의 인바운드 datashare 정보를 보여줍니다. 소비자 관리자로 DESC DATAHSARE를 실행할 때 인바운드 데이터 공유를 보려면 NAMESPACE와 계정 ID를 지정해야 합니다. 아웃바운드 datashare의 경우 datashare 이름을 지정합니다.

```
SHOW DATASHARES LIKE 'sales%';

share_name | share_owner | source_database | consumer_database | share_type
 | createdate | is_publicaccessible | share_acl | producer_account |
 producer_namespace
-----------+-------------+-----------------+-------------------+------------
+------------+---------------------+-----------+-----------------
+--------------------------------------
salesshare |             |                 |                   | INBOUND    |
            |        t            |           | 123456789012    | 'dd8772e1-
d792-4fa4-996b-1870577efc0d'
```

```
DESC DATASHARE salesshare OF ACCOUNT '123456789012' NAMESPACE 'dd8772e1-
d792-4fa4-996b-1870577efc0d';


 producer_account |          producer_namespace          | share_type | share_name |
 object_type |           object_name
------------------+--------------------------------------+------------+------------
+-------------+---------------------------------
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_users_redshift
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_venue_redshift
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_category_redshift
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_date_redshift
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_event_redshift
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_listing_redshift
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_sales_redshift
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 schema      | public
(8 rows)
```

클러스터 슈퍼 사용자만 이 작업을 수행할 수 있습니다. SVV_DATASHARES를 사용하여 datashare를 보고 SVV_DATASHARE_OBJECTS를 사용하여 datashare 내의 객체를 볼 수도 있습니다.

다음 예에서는 소비자 클러스터의 인바운드 datashare를 보여줍니다.

```
SELECT * FROM SVV_DATASHARES WHERE share_name LIKE 'sales%';

share_name | share_owner | source_database | consumer_database | share_type
 | createdate | is_publicaccessible | share_acl | producer_account |
 producer_namespace
-----------+-------------+-----------------+-------------------+-----------
+------------+---------------------+-----------+------------------
+--------------------------------------
salesshare |             |                 |                   | INBOUND    |
            |        t            |           | 123456789012      | 'dd8772e1-
d792-4fa4-996b-1870577efc0d'
```

```
SELECT * FROM SVV_DATASHARE_OBJECTS WHERE share_name LIKE 'sales%';
 share_type | share_name | object_type |           object_name           |
 producer_account |          producer_namespace
------------+------------+-------------+---------------------------------
+------------------+--------------------------------------
 INBOUND    | salesshare | table       | public.tickit_users_redshift    |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
 INBOUND    | salesshare | table       | public.tickit_venue_redshift    |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
 INBOUND    | salesshare | table       | public.tickit_category_redshift |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
 INBOUND    | salesshare | table       | public.tickit_date_redshift     |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
 INBOUND    | salesshare | table       | public.tickit_event_redshift    |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
 INBOUND    | salesshare | table       | public.tickit_listing_redshift  |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
 INBOUND    | salesshare | table       | public.tickit_sales_redshift    |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
 INBOUND    | salesshare | schema      | public                          |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
(8 rows)
```

2. datashare를 참조하는 로컬 데이터베이스를 생성합니다. datashare에서 데이터베이스를 생성할 때 NAMESPACE와 계정 ID를 지정합니다. 자세한 내용은 데이터베이스 생성 단원을 참조하십시오.

```
CREATE DATABASE sales_db FROM DATASHARE salesshare OF ACCOUNT '123456789012'
 NAMESPACE 'dd8772e1-d792-4fa4-996b-1870577efc0d';
```

로컬 데이터베이스의 객체에 대한 액세스를 보다 세밀하게 제어하려면 데이터베이스를 만들 때 WITH PERMISSIONS 절을 사용하세요. 이렇게 하면 4단계에서 데이터베이스의 객체에 객체 수준 권한을 부여할 수 있습니다.

```
CREATE DATABASE sales_db WITH PERMISSIONS FROM DATASHARE salesshare OF ACCOUNT
 '123456789012' NAMESPACE 'dd8772e1-d792-4fa4-996b-1870577efc0d';
```

SVV_REDSHIFT_DATABASES 뷰를 쿼리하여 datashare에서 생성한 데이터베이스를 볼 수 있습니다. 그러나 소비자 클러스터의 로컬 데이터베이스에 연결하고 데이터베이스 간 쿼리를 수행하여 데이터 공유에서 만들어진 데이터베이스의 데이터를 쿼리할 수 있습니다. 기존 datashare에서 생성된 데이터베이스 객체 위에 datashare를 생성할 수 없습니다. 그러나 데이터를 소비자 클러스터의 별도 테이블에 복사하고 필요한 처리를 수행한 다음 생성된 새 객체를 공유할 수 있습니다.

3. (옵션) 소비자 클러스터에서 가져온 소비자 데이터베이스의 특정 스키마를 참조하고 세분화된 권한을 할당할 외부 스키마를 생성합니다. 자세한 내용은 CREATE EXTERNAL SCHEMA 단원을 참조하십시오.

```
CREATE EXTERNAL SCHEMA sales_schema FROM REDSHIFT DATABASE 'sales_db' SCHEMA
 'public';
```

4. 필요에 따라 소비자 클러스터의 사용자 또는 역할에 데이터 공유에서 생성된 데이터베이스 및 스키마 참조에 대한 권한을 부여합니다. 자세한 내용은 GRANT 또는 REVOKE을 참조하세요.

```
GRANT USAGE ON DATABASE sales_db TO Bob;
```

```
GRANT USAGE ON SCHEMA sales_schema TO ROLE Analyst_role;
```

WITH PERMISSIONS를 사용하지 않고 데이터베이스를 생성한 경우 데이터 공유에서 생성된 전체 데이터베이스에 대한 권한만 사용자 또는 역할에 할당할 수 있습니다. 경우에 따라 datashare에서 생성된 데이터베이스 객체의 하위 집합에 대한 세분화된 제어가 필요합니다. 이러한 경우 이전 단계에서 설명한 대로 datashare의 특정 스키마를 가리키는 외부 스키마 참조를 생성할 수 있습니다. 그런 다음 스키마 수준에서 세분화된 권한을 제공할 수 있습니다. 또한 공유 객체 위에 후기 바인딩 뷰를 생성하고 이를 사용하여 세분화된 권한을 할당할 수 있습니다. 생산자 클러스터가 필요한 세분성

으로 추가 datashare를 생성하도록 할 수도 있습니다. 데이터 공유에서 생성된 데이터베이스에 대해 필요에 따라 원하는 만큼 스키마 참조를 생성할 수 있습니다.

2단계에서 WITH PERMISSIONS를 사용하여 데이터베이스를 만든 경우 공유 데이터베이스의 객체에 객체 수준 권한을 할당해야 합니다. USAGE 권한만 있는 사용자는 추가 객체 수준 권한을 부여받을 때까지 WITH PERMISSION을 사용하여 만든 데이터베이스의 객체에 액세스할 수 없습니다.

```
GRANT SELECT ON sales_db.public.tickit_sales_redshift to Bob;
```

5. datashare의 공유 객체에서 데이터를 쿼리합니다.

   소비자 데이터베이스 및 소비자 클러스터의 스키마에 대한 권한이 있는 사용자와 역할은 모든 공유 객체의 메타데이터를 탐색할 수 있습니다. 또한 소비자 클러스터에서 로컬 객체를 탐색할 수 있습니다. 이를 위해 JDBC 또는 ODBC 드라이버 또는 SVV_ALL 및 SVV_REDSHIFT 뷰를 사용합니다.

   생산자 클러스터의 경우 각 스키마 내의 데이터베이스, 테이블 및 뷰에 많은 스키마가 있을 수 있습니다. 소비자 측의 사용자는 datashare를 통해 사용할 수 있는 객체의 하위 집합만 볼 수 있습니다. 이러한 사용자는 생산자 클러스터에서 모든 메타데이터를 볼 수 없습니다. 이 접근 방식은 datashare로 세분화된 메타데이터 보안 제어를 제공하는 데 도움이 됩니다.

   로컬 클러스터 데이터베이스에 계속 연결합니다. 그러나 이제 세 부분으로 구성된 database.schema.table 표기법을 사용하여 datashare에서 생성된 데이터베이스와 스키마에서도 읽을 수도 있습니다. 사용자에게 표시되는 모든 데이터베이스에 걸쳐 쿼리를 수행할 수 있습니다. 이러한 데이터베이스는 클러스터의 로컬 데이터베이스이거나 datashare에서 생성된 데이터베이스일 수 있습니다. 소비자 클러스터는 datashare에서 생성된 데이터베이스에 연결할 수 없습니다.

   전체 자격을 사용하여 데이터에 액세스할 수 있습니다. 자세한 내용은 데이터베이스 간 쿼리 예제 단원을 참조하십시오.

   ```
   SELECT * FROM sales_db.public.tickit_sales_redshift;
   ```

   공유 객체에는 SELECT 문만 사용할 수 있습니다. 그러나 다른 로컬 데이터베이스에 있는 공유 객체의 데이터를 쿼리하여 소비자 클러스터에 테이블을 생성할 수 있습니다.

   쿼리를 수행하는 것 외에도 소비자는 공유 객체에 대한 보기를 생성할 수 있습니다. 후기 바인딩 보기와 구체화된 보기만 지원됩니다. Amazon Redshift는 공유 데이터에 대한 일반 뷰를 지원하지 않습니다. 소비자가 생성하는 뷰는 여러 로컬 데이터베이스 또는 datashare에서 생성된 데이터베이스에 걸쳐 있을 수 있습니다. 자세한 내용은 CREATE VIEW 단원을 참조하십시오.

```
// Connect to a local cluster database

// Create a view on shared objects and access it.
CREATE VIEW sales_data
AS SELECT *
FROM sales_db.public.tickit_sales_redshift
WITH NO SCHEMA BINDING;

SELECT * FROM sales_data;
```

AWS 리전에서 데이터 공유

AWS 리전의 Amazon Redshift 클러스터에서 읽기용으로 데이터를 공유할 수 있습니다. 리전 간 데이터 공유를 사용하면 데이터를 수동으로 복사할 필요 없이 AWS 리전 간에 데이터를 공유할 수 있습니다. 데이터를 Amazon S3로 언로드하고 데이터를 새 Amazon Redshift 클러스터로 복사하거나 리전 간 스냅샷 복사를 수행할 필요가 없습니다.

리전 간 데이터 공유를 사용하면 클러스터가 다른 리전에 있는 경우에도 동일한 AWS 계정 또는 다른 AWS 계정의 클러스터 간에 데이터를 공유할 수 있습니다. AWS 계정은 같지만 AWS 리전은 다른 Amazon Redshift 클러스터와 데이터를 공유하는 경우 AWS 계정 내에서 데이터를 공유하는 것과 동일한 워크플로를 따릅니다. 자세한 내용은 AWS 계정 내 데이터에 대한 읽기 액세스 공유 단원을 참조하십시오.

데이터를 공유하는 클러스터가 다른 AWS 계정과 AWS 리전에 있는 경우 AWS 계정 간에 데이터를 공유하는 것과 동일한 워크플로를 따르고 소비자 클러스터에 리전 수준 연결을 포함할 수 있습니다. 리전 간 데이터 공유는 전체 AWS 계정, 전체 AWS 리전 또는 AWS 리전 내의 특정 네임스페이스와의 데이터 공유 연결을 지원합니다. AWS 계정 간 데이터 공유에 대한 자세한 내용은 AWS 계정에서 데이터 공유 섹션을 참조하세요.

다른 리전의 데이터를 사용하는 경우 소비자는 생산자 리전에서 소비자 리전으로의 리전 간 데이터 전송 요금을 지불합니다.

datashare를 사용하기 위해 소비자 계정 관리자는 다음 세 가지 방법 중 하나로 datashare를 연결할 수 있습니다.

- 모든 AWS 리전에 걸쳐 있는 전체 AWS 계정과의 연결
- AWS 계정의 특정 AWS 리전과 연결
- AWS 리전 내의 특정 네임스페이스와의 연결

관리자가 전체 AWS 계정을 선택하면 계정의 다른 AWS 리전에 있는 모든 기존 및 미래 네임스페이스가 데이터 공유에 액세스할 수 있습니다. 소비자 계정 관리자는 리전 내에서 특정 AWS 리전 또는 네임스페이스를 선택하여 데이터 공유에 대한 액세스 권한을 부여할 수도 있습니다.

생산자 관리자 또는 데이터베이스 소유자인 경우 데이터 공유를 만들고, 데이터베이스 객체와 데이터 소비자를 데이터 공유에 추가하고, 데이터 소비자에게 권한을 부여합니다. 자세한 내용은 생산자 관리자 작업 단원을 참조하십시오.

생산자 계정 관리자인 경우 AWS Command Line Interface(AWS CLI) 또는 Amazon Redshift 콘솔을 사용하여 datashare 권한을 부여하고 데이터 소비자를 선택합니다.

소비자 계정 관리자인 경우 다음 절차를 따릅니다.

다른 계정에서 공유되는 하나 이상의 데이터 공유를 전체 AWS 계정, 특정 AWS 리전 또는 AWS 리전 내의 네임스페이스에 연결하려면 Amazon Redshift 콘솔을 사용합니다.

리전 간 데이터 공유에서는 AWS Command Line Interface(AWS CLI) 또는 Amazon Redshift 콘솔을 사용하여 특정 AWS 리전의 클러스터를 추가할 수 있습니다.

하나 이상의 AWS 리전을 지정하려면 선택 사항 `consumer-region` 옵션과 함께 `associate-data-share-consumer` CLI 명령을 사용할 수 있습니다.

CLI를 사용할 경우, 다음 예는 `associate-entire-account` 옵션을 사용하여 `Salesshare`을 전체 AWS 계정과 연결합니다. 한 번에 한 리전만 연결할 수 있습니다.

```
aws redshift associate-data-share-consumer
--region {PRODUCER_REGION}
--data-share-arn arn:aws:redshift:{PRODUCER_REGION}:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/Salesshare
--associate-entire-account
```

다음 예에서는`Salesshare`을 미국 동부(오하이오) 리전(`us-east-2`)과 연결합니다.

```
aws redshift associate-data-share-consumer
--region {PRODUCER_REGION}
--data-share-arn arn:aws:redshift:{PRODUCER_REGION}:0123456789012:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/Salesshare
--consumer-region 'us-east-2'
```

다음 예에서는 `Salesshare`을 아시아 태평양(시드니) 리전(`ap-southeast-2`)의 다른 AWS 계정에 있는 특정 소비자 네임스페이스와 연결합니다.

```
aws redshift associate-data-share-consumer
--data-share-arn arn:aws:redshift:{PRODUCER_REGION}:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/Salesshare
--consumer-arn 'arn:aws:redshift:ap-southeast-2:{CONSUMER_ACCOUNT}:namespace:
{ConsumerImmutableClusterId}'
```

Amazon Redshift 콘솔을 사용하여 데이터 공유를 전체 AWS 계정 또는 AWS 리전 내의 특정 AWS 리전 또는 네임스페이스와 연결할 수 있습니다. 이렇게 하려면 https://console.aws.amazon.com/redshiftv2/에 로그인합니다. 그런 다음 다른 계정에서 공유되는 하나 이상의 데이터 공유를 전체 AWS 계정, 전체 AWS 리전 또는 AWS 리전 내의 특정 네임스페이스와 연결합니다. 자세한 내용은 Amazon Redshift에서 다른 AWS 계정의 데이터 공유 연결 단원을 참조하십시오.

AWS 계정 또는 특정 네임스페이스가 연결된 후 데이터 공유를 사용할 수 있게 됩니다. datashare 연결을 언제든지 변경할 수도 있습니다. 개별 네임스페이스에서 AWS 계정으로 연결을 변경하면 Amazon Redshift가 네임스페이스를 AWS 계정 정보로 덮어씁니다. AWS 계정에서 특정 네임스페이스로 연결을 변경할 때 Amazon Redshift는 AWS 계정 정보를 네임스페이스 정보로 덮어씁니다. 전체 AWS 계정에서 특정 AWS 리전 및 네임스페이스로 연결을 변경할 때 Amazon Redshift는 AWS 계정 정보를 특정 리전 및 네임스페이스 정보로 덮어씁니다.

소비자 관리자인 경우 데이터 공유를 참조하는 로컬 데이터베이스를 만들고 필요에 따라 데이터 공유에서 만들어진 데이터베이스에 대한 권한을 소비자 클러스터의 사용자 또는 역할에 부여할 수 있습니다. 공유 객체에 대한 보기를 생성하고, 소비자 클러스터에서 가져온 소비자 데이터베이스의 특정 스키마를 참조하고 세분화된 권한을 할당할 외부 스키마를 생성할 수도 있습니다. 자세한 내용은 소비자 관리자 작업 단원을 참조하십시오.

리전 간 데이터 공유를 위한 비용 관리

Amazon Redshift를 사용하면 AWS 리전 간에 전송되는 데이터의 양을 제한하도록 데이터 공유를 구성하여 리전 간 데이터 공유에 대한 비용 제어를 관리할 수 있습니다. 리전 간 데이터 공유에 대한 비용 제어 관리를 통해 데이터 전송 한도를 설정하고, 데이터 전송 사용량을 모니터링하고, 이러한 한도에 근접하거나 초과할 때 알림을 받게 됩니다.

다른 리전의 데이터를 사용하는 경우 소비자는 생산자 리전에서 소비자 리전으로의 리전 간 데이터 전송 요금을 지불합니다. 데이터 전송 가격은 리전마다 다릅니다. 요금은 모든 성공적인 쿼리 실행에 대해 스캔된 데이터 바이트를 기준으로 합니다. Amazon Redshift 요금에 대한 자세한 내용은 Amazon Redshift 요금을 참조하세요.

요금은 바이트 단위로 부과되며 바이트는 메가바이트로 반올림됩니다. 쿼리당 최소값은 10MB입니다. 쿼리 사용량에 대한 비용 관리를 설정하고 클러스터에서 쿼리당 전송되는 데이터의 양을 볼 수 있습니다.

리전 간 데이터 공유를 사용한 사용 및 관련 비용을 모니터링하고 제어하기 위해 매일, 매주, 매달 사용 한도를 생성하고 이 한도에 도달하면 Amazon Redshift에서 자동으로 수행할 작업을 정의하여 예산을 예측 가능하게 유지할 수 있습니다. Amazon Redshift의 사용 한도에 대한 자세한 내용은 Amazon Redshift에서 사용 한도 관리를 참조하세요.

사용자가 설정한 사용 한도에 따라 Amazon Redshift가 수행하는 작업을 이벤트를 시스템 테이블에 기록하거나 CloudWatch 경보를 전송한 후 Amazon SNS 통해 관리자에게 알리거나 추가 사용을 위해 리전 간 데이터 공유를 해제합니다. 작업에 대한 자세한 내용은 Amazon Redshift에서 사용 한도 관리를 참조하세요.

Amazon Redshift 콘솔에서 사용 한도를 정의하려면 클러스터에 대한 작업(Actions)에서 사용량 한도 구성을 선택합니다. 클러스터 성능(Cluster performance) 또는 모니터링(Monitoring) 탭에서 자동으로 생성된 CloudWatch 메트릭과 함께 사용량 추세를 모니터링하고 정의된 한도를 초과하는 사용량 발생 시 알림을 받을 수 있습니다. 또한 AWS CLI 또는 Amazon Redshift API 동작을 이용하여 프로그램 방식으로 사용 한도를 생성, 수정 및 삭제할 수도 있습니다. 자세한 내용은 Amazon Redshift에서 사용 한도 관리를 참조하세요.

AWS Data Exchange에서 라이선스가 부여된 Amazon Redshift 데이터 공유

공급자는 AWS Data Exchange datashare를 생성하고 AWS Data Exchange 제품에 추가할 때 소비자에게 활성 AWS Data Exchange 구독이 있으면 Amazon Redshift에서 최신 데이터를 검색, 구독 및 쿼리할 수 있는 Amazon Redshift의 데이터에 라이선스를 부여할 수 있습니다.

AWS Data Exchange datashare가 AWS Data Exchange 제품에 추가되면 소비자는 구독이 시작될 때 제품의 datashare에 자동으로 액세스할 수 있으며 구독이 활성화되어 있는 한 액세스를 유지합니다.

주제

- 생산자로서 AWS Data Exchange datashare 작업
- 소비자로서 AWS Data Exchange datashare 작업

생산자로서 AWS Data Exchange datashare 작업

Amazon Redshift로 데이터 공유를 생성 및 관리하여 생산자로서 AWS Data Exchange를 사용해 실시간으로 데이터 제품을 공유합니다.

생산자 관리자인 경우 다음 단계에 따라 Amazon Redshift 콘솔에서 AWS Data Exchange 데이터 공유를 관리합니다.

1. 클러스터에 datashare를 생성하여 AWS Data Exchange에서 데이터를 공유하고 datashare에 AWS Data Exchange에 대한 액세스 권한을 부여합니다.

클러스터 슈퍼 사용자와 데이터베이스 소유자는 datashare를 생성할 수 있습니다. 각 datashare는 생성하는 동안 데이터베이스와 연결됩니다. 해당 데이터베이스의 객체만 해당 datashare에서 공유할 수 있습니다. 동일하거나 다른 세부 수준의 객체를 사용하여 동일한 데이터베이스에 여러 datashare를 생성할 수 있습니다. 클러스터에서 생성할 수 있는 datashare 수에는 제한이 없습니다.

Amazon Redshift 콘솔을 사용하여 datashare를 생성할 수도 있습니다. 자세한 내용은 데이터 공유 만들기 단원을 참조하십시오.

CREATE DATASHARE 문을 실행할 때 MANAGEDBY ADX 옵션을 사용하여 암시적으로 AWS Data Exchange에 대한 datashare 액세스 권한을 부여합니다. 이는 AWS Data Exchange에서 이 datashare를 관리함을 나타냅니다. 새 datashare를 생성할 때만 MANAGEDBY ADX 옵션을 사용할 수 있습니다. ALTER DATASHARE 문으로 기존 datashare를 수정하여 MANAGEDBY ADX 옵션을 추가할 수 없습니다. MANAGEDBY ADX 옵션으로 datashare가 생성되면 AWS Data Exchange만 datashare에 액세스하고 관리할 수 있습니다.

```
CREATE DATASHARE salesshare
[[SET] MANAGEDBY [=] {ADX} ];
```

2. datashare에 객체를 추가합니다. 생산자 관리자는 AWS Data Exchange datashare에서 사용할 수 있는 datashare 객체를 계속 관리합니다.

   datashare에 객체를 추가하려면 객체를 추가하기 전에 스키마를 추가합니다. 스키마를 추가할 때 Amazon Redshift는 스키마 아래에 모든 객체를 추가하지 않습니다. 명시적으로 스키마를 추가해야 합니다. 자세한 내용은 ALTER DATASHARE 단원을 참조하십시오.

   ```
   ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
   ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;
   ALTER DATASHARE salesshare ADD ALL TABLES IN SCHEMA PUBLIC;
   ```

   datashare에 뷰를 추가할 수도 있습니다.

   ```
   CREATE VIEW public.sales_data_summary_view AS SELECT * FROM
    public.tickit_sales_redshift;
   ALTER DATASHARE salesshare ADD TABLE public.sales_data_summary_view;
   ```

   ALTER DATASHARE를 사용하여 지정된 스키마에서 스키마, 테이블, 뷰 및 함수를 공유합니다. 슈퍼 사용자, datashare 소유자 또는 datashare에 대한 ALTER 또는 ALL 권한이 있는 사용자

는 datashare를 변경하여 객체를 추가하거나 제거할 수 있습니다. 사용자는 datashare에서 객체를 추가하거나 제거할 수 있는 권한이 있어야 합니다. 사용자는 객체 소유자이거나 객체에 대한 SELECT, USAGE 또는 ALL 권한도 있어야 합니다.

INCLUDENEW 절을 사용하여 지정된 스키마에서 생성된 새 테이블, 뷰 또는 SQL 사용자 정의 함수(UDF)를 datashare에 추가합니다. 슈퍼 사용자만 각 datashare-스키마 페어에 대해 이 속성을 변경할 수 있습니다.

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare SET INCLUDENEW = TRUE FOR SCHEMA PUBLIC;
```

Amazon Redshift 콘솔을 사용하여 datashare에서 객체를 추가하거나 제거할 수도 있습니다. 자세한 내용은 데이터 공유에 데이터 공유 객체 추가, datashare에서 datashare 객체 제거, AWS Data Exchange datashare 편집 섹션을 참조하세요.

3. AWS Data Exchange에 대해 datashare에 대한 액세스 권한을 부여하려면 다음 중 하나를 수행합니다.

   - `aws redshift authorize-data-share` API에서 `ADX` 키워드를 사용하여 AWS Data Exchange에 대해 datashare에 대한 액세스 권한을 명시적으로 부여합니다. 이를 통해 AWS Data Exchange는 서비스 계정의 datashare를 인식하고 소비자를 datashare에 연결하는 것을 관리할 수 있습니다.

     ```
     aws redshift authorize-data-share
     --data-share-arn arn:aws:redshift:us-east-1:{PRODUCER_ACCOUNT}:datashare:
     {PRODUCER_CLUSTER_NAMESPACE}/salesshare
     --consumer-identifier ADX
     ```

     `AuthorizeDataShare` 및 `DeauthorizeDataShare` API에 조건부 키 `ConsumerIdentifier`를 사용하여 AWS Data Exchange가 IAM 정책에서 두 API를 호출하는 것을 명시적으로 허용하거나 거부할 수 있습니다.

     ```
     {
         "Version": "2012-10-17",
         "Statement": [
             {
                 "Sid": "VisualEditor0",
                 "Effect": "Deny",
                 "Action": [
                     "redshift:AuthorizeDataShare",
     ```

```
                "redshift:DeauthorizeDataShare"
            ],
            "Resource": "*",
            "Condition": {
                "StringEqualsIgnoreCase": {
                    "redshift:ConsumerIdentifier": "ADX"
                }
            }
        }
    ]
}
```

- Amazon Redshift 콘솔을 사용하여 AWS Data Exchange datashare 권한을 부여하거나 제거합니다. 자세한 내용은 datashare에서 권한 부여 또는 제거 단원을 참조하십시오.
- 선택적으로 AWS Data Exchange datashare를 AWS Data Exchange 데이터 집합으로 가져올 때 datashare에 대한 액세스 권한을 암시적으로 부여할 수 있습니다.

AWS Data Exchange datashares에 대한 액세스 권한 부여를 제거하려면 `aws redshift deauthorize-data-share` API 작업에서 `ADX` 키워드를 사용합니다. 이렇게 하면 AWS Data Exchange에서 서비스 계정의 datashare를 인식하고 datashare에서 연결 제거를 관리할 수 있습니다.

```
aws redshift deauthorize-data-share
--data-share-arn arn:aws:redshift:us-east-1:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/salesshare
--consumer-identifier ADX
```

4. 클러스터에서 생성된 datashare를 나열하고 datashare의 내용을 살펴봅니다.

   다음 예에서는 salesshare라는 datashare의 정보를 보여줍니다. 자세한 내용은 DESC DATASHARE 및 SHOW DATASHARES 섹션을 참조하세요.

```
DESC DATASHARE salesshare;

 producer_account  |          producer_namespace          | share_type | share_name
 | object_type |           object_name          |   include_new
-------------------+--------------------------------------+------------
+------------+-------------+----------------------------------+-------------------
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_users_redshift   |
```

```
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_venue_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_category_redshift|
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_date_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_event_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_listing_redshift |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_sales_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | schema      | public                         |  t
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | view        | public.sales_data_summary_view |
```

다음 예에서는 생산자 클러스터의 아웃바운드 datashare를 보여줍니다.

```
SHOW DATASHARES LIKE 'sales%';
```

출력 결과는 다음과 비슷합니다.

```
share_name | share_owner  | source_database | consumer_database | share_type |
 createdate      | is_publicaccessible  | share_acl | producer_account |
 producer_namespace
-----------+-------------+-----------------+-------------------+------------
+---------------------+----------------------+-----------+-----------------
+--------------------------------------
salesshare |    100       | dev             |                   |  OUTBOUND
 | 2020-12-09 02:27:08 |          True        |           |   123456789012   |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d
```

자세한 내용은 DESC DATASHARE 및 SHOW DATASHARES 섹션을 참조하세요.

SVV_DATASHARES, SVV_DATASHARE_CONSUMERS, 및 SVV_DATASHARE_OBJECTS를 사용하여 datashare, datashare 내의 객체 및 datashare 소비자를 볼 수도 있습니다.

5. datashare를 삭제합니다. DROP DATASHARE 문을 사용하여 다른 AWS 계정과 공유되는 AWS Data Exchange datashare를 삭제하지 않는 것이 좋습니다. 이러한 계정은 dathare에 대한 액세스 권한을 상실합니다. 이 작업은 되돌릴 수 없습니다. 이는 AWS Data Exchange의 데이터 제품

및 서비스 약관을 위반할 수 있습니다. AWS Data Exchange datashare를 삭제하려는 경우 DROP DATASHARE 사용 참고 사항 섹션을 참조하세요.

다음 예에서는 salesshare라는 datashare를 삭제합니다.

```
DROP DATASHARE salesshare;
ERROR:  Drop of ADX-managed datashare salesshare requires session variable
 datashare_break_glass_session_var to be set to value '620c871f890c49'
```

AWS Data Exchange datashare 삭제를 허용하려면 datashare_break_glass_session_var 변수를 설정하고 DROP DATASHARE 문을 다시 실행합니다. AWS Data Exchange datashare를 삭제하려는 경우 DROP DATASHARE 사용 참고 사항 섹션을 참조하세요.

Amazon Redshift 콘솔을 사용하여 datashare를 삭제할 수도 있습니다. 자세한 내용은 계정에 생성된 AWS Data Exchange datashare 삭제 단원을 참조하십시오.

6. ALTER DATASHARE를 사용하여 datashare에서 언제든지 객체를 제거합니다. REVOKE USAGE ON을 사용하여 특정 소비자에 대한 datashare 권한을 취소합니다. datashare 내의 객체에 대한 USAGE 권한을 취소하고 모든 소비자 클러스터에 대한 액세스를 즉시 중지합니다. 데이터베이스 및 테이블 나열 등의 datashare 나열과 메타데이터 쿼리는 액세스가 취소된 후 공유 객체를 반환하지 않습니다.

```
ALTER DATASHARE salesshare REMOVE TABLE public.tickit_sales_redshift;
```

Amazon Redshift 콘솔을 사용하여 datashare를 편집할 수도 있습니다. 자세한 내용은 AWS Data Exchange datashare 편집 단원을 참조하십시오.

7. AWS Data Exchange datashare에서 GRANT USAGE를 부여하거나 취소합니다. AWS Data Exchange datashare에 대한 GRANT USAGE를 부여하거나 취소할 수 없습니다. 다음 예에서는 AWS Data Exchange에서 관리하는 datashare에 대해 GRANT USAGE 권한이 AWS 계정에 부여된 경우 오류를 보여줍니다.

```
CREATE DATASHARE salesshare MANAGEDBY ADX;
```

```
GRANT USAGE ON DATASHARE salesshare TO ACCOUNT '012345678910';
ERROR:  Permission denied to add/remove consumer to/from datashare salesshare.
 Datashare consumers are managed by ADX.
```

자세한 내용은 GRANT 또는 REVOKE을 참조하세요.

생산자 관리자인 경우 다음 단계에 따라 AWS Data Exchange 콘솔에서 데이터 공유 제품을 만들고 게시합니다.

- AWS Data Exchange datashare가 생성되면 생산자는 새 데이터 집합을 생성하고, 자산을 가져오고, 개정을 생성하고, 새 제품을 생성 및 게시합니다.

  Amazon Redshift 콘솔을 사용하여 데이터 집합을 생성합니다. 자세한 내용은 AWS Data Exchange에서 데이터 집합 생성 단원을 참조하십시오.

  자세한 내용은 AWS Data Exchange에서 데이터 제품 제공을 참조하세요.

소비자로서 AWS Data Exchange datashare 작업

Amazon Redshift로 데이터 사본을 저장하거나 관리할 필요 없이 AWS Data Exchange에서 데이터세트에 액세스하고 분석합니다.

소비자인 경우 다음 단계에 따라 AWS Data Exchange datashare가 포함된 데이터 제품을 검색하고 Amazon Redshift 데이터를 쿼리합니다.

1. AWS Data Exchange 콘솔에서 AWS Data Exchange datashare가 포함된 데이터 제품을 검색하고 구독합니다.

   구독이 시작되면 AWS Data Exchange datashare가 포함된 데이터 집합에 자산으로 가져온 라이선스가 부여된 Amazon Redshift 데이터에 액세스할 수 있습니다.

   AWS Data Exchange datashare가 포함된 데이터 제품 사용을 시작하는 방법에 대한 자세한 내용은 AWS Data Exchange에서 데이터 제품 구독을 참조하세요.
2. 필요한 경우 Amazon Redshift 콘솔에서 Amazon Redshift 클러스터를 생성합니다.

   클러스터 생성 방법에 대한 자세한 내용은 클러스터 생성을 참조하세요.
3. 사용할 수 있는 datashare를 나열하고 datashare의 내용을 봅니다. 자세한 내용은 DESC DATASHARE 및 SHOW DATASHARES 섹션을 참조하세요.

   다음 예에서는 지정된 생산자 네임스페이스의 인바운드 datashare 정보를 보여줍니다. 소비자 관리자로 DESC DATASHARE를 실행할 때 인바운드 데이터 공유를 보려면 ACCOUNT 및 NAMESPACE 옵션을 지정해야 합니다.

```
DESC DATASHARE salesshare of ACCOUNT '123456789012' NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

```
 producer_account  |          producer_namespace          | share_type | share_name
 | object_type |           object_name           |   include_new
-------------------+--------------------------------------+------------
+------------+-------------+----------------------------------+-----------------
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_users_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_venue_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_category_redshift |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_date_redshift     |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_event_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_listing_redshift  |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_sales_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | schema      | public                          |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | view        | public.sales_data_summary_view  |
```

클러스터 슈퍼 사용자만 이 작업을 수행할 수 있습니다. SVV_DATASHARES를 사용하여 datashare를 보고 SVV_DATASHARE_OBJECTS를 사용하여 datashare 내의 객체를 볼 수도 있습니다.

다음 예에서는 소비자 클러스터의 인바운드 datashare를 보여줍니다.

```
SHOW DATASHARES LIKE 'sales%';


 share_name | share_owner | source_database | consumer_database | share_type
 | createdate | is_publicaccessible | share_acl | producer_account |
 producer_namespace
------------+-------------+-----------------+-------------------+------------
+------------+---------------------+-----------+------------------
+--------------------------------------
 salesshare |             |                 |                   | INBOUND
    |            |         t           |           |   123456789012   |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d
```

4. datashare를 참조하는 로컬 데이터베이스를 생성합니다. AWS Data Exchange datashare를 위한 로컬 데이터베이스를 생성하려면 ACCOUNT 및 NAMESPACE 옵션을 지정해야 합니다. 자세한 내용은 데이터베이스 생성 단원을 참조하십시오.

```
CREATE DATABASE sales_db FROM DATASHARE salesshare OF ACCOUNT '123456789012'
 NAMESPACE '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

로컬 데이터베이스의 객체에 대한 액세스를 보다 세밀하게 제어하려면 데이터베이스를 만들 때 WITH PERMISSIONS 절을 사용하세요. 이렇게 하면 6단계에서 데이터베이스의 객체에 객체 수준 권한을 부여할 수 있습니다.

```
CREATE DATABASE sales_db WITH PERMISSIONS FROM DATASHARE salesshare OF ACCOUNT
 '123456789012' NAMESPACE '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

SVV_REDSHIFT_DATABASES 뷰를 쿼리하여 datashare에서 생성한 데이터베이스를 볼 수 있습니다. 그러나 소비자 클러스터의 로컬 데이터베이스에 연결하고 데이터베이스 간 쿼리를 수행하여 데이터 공유에서 만들어진 데이터베이스의 데이터를 쿼리할 수 있습니다. 기존 datashare에서 생성된 데이터베이스 객체 위에 datashare를 생성할 수 없습니다. 그러나 데이터를 소비자 클러스터의 별도 테이블에 복사하고 필요한 처리를 수행한 다음 생성된 새 객체를 공유할 수 있습니다.

Amazon Redshift 콘솔을 사용하여 datashare에서 데이터베이스를 생성할 수도 있습니다. 자세한 내용은 datashare에서 데이터베이스 생성 단원을 참조하십시오.

5. (옵션) 소비자 클러스터에서 가져온 소비자 데이터베이스의 특정 스키마를 참조하고 세분화된 권한을 할당할 외부 스키마를 생성합니다. 자세한 내용은 CREATE EXTERNAL SCHEMA 단원을 참조하십시오.

```
CREATE EXTERNAL SCHEMA sales_schema FROM REDSHIFT DATABASE 'sales_db' SCHEMA
 'public';
```

6. 필요에 따라 소비자 클러스터의 사용자 또는 역할에 데이터 공유에서 생성된 데이터베이스 및 스키마 참조에 대한 권한을 부여합니다. 자세한 내용은 GRANT 또는 REVOKE을 참조하세요.

```
GRANT USAGE ON DATABASE sales_db TO Bob;
```

```
GRANT USAGE ON SCHEMA sales_schema TO ROLE Analyst_role;
```

WITH PERMISSIONS를 사용하지 않고 데이터베이스를 생성한 경우 데이터 공유에서 생성된 전체 데이터베이스에 대한 권한만 사용자와 역할에 할당할 수 있습니다. 경우에 따라 datashare에서 생성된 데이터베이스 객체의 하위 집합에 대한 세분화된 제어가 필요합니다. 이러한 경우 이전 단계에서 설명한 대로 datashare의 특정 스키마를 가리키는 외부 스키마 참조를 생성하고 스키마 수준에서 세분화된 권한을 제공할 수 있습니다.

또한 공유 객체 위에 후기 바인딩 뷰를 생성하고 이를 사용하여 세분화된 권한을 할당할 수 있습니다. 생산자 클러스터가 필요한 세분성으로 추가 datashare를 생성하도록 할 수도 있습니다. 데이터 공유에서 생성된 데이터베이스에 대해 필요에 따라 원하는 만큼 스키마 참조를 생성할 수 있습니다.

4단계에서 WITH PERMISSIONS를 사용하여 데이터베이스를 만든 경우 공유 데이터베이스의 객체에 객체 수준 권한을 할당해야 합니다. USAGE 권한만 있는 사용자는 추가 객체 수준 권한을 부여받을 때까지 WITH PERMISSION을 사용하여 만든 데이터베이스의 객체에 액세스할 수 없습니다.

```
GRANT SELECT ON sales_db.public.tickit_sales_redshift to Bob;
```

7. datashare의 공유 객체에서 데이터를 쿼리합니다.

   소비자 데이터베이스 및 소비자 클러스터의 스키마에 대한 권한이 있는 사용자와 역할은 모든 공유 객체의 메타데이터를 탐색할 수 있습니다. 또한 소비자 클러스터에서 로컬 객체를 탐색할 수 있습니다. 이를 위해 JDBC 또는 ODBC 드라이버, SHOW 명령, SVV_ALL 및 SVV_REDSHIFT 뷰를 사용합니다.

   생산자 클러스터의 경우 각 스키마 내의 데이터베이스, 테이블 및 뷰에 많은 스키마가 있을 수 있습니다. 소비자 측의 사용자는 datashare를 통해 사용할 수 있는 객체의 하위 집합만 볼 수 있습니다. 이러한 사용자는 생산자 클러스터에서 전체 메타데이터를 볼 수 없습니다. 이 접근 방식은 datashare로 세분화된 메타데이터 보안 제어를 제공하는 데 도움이 됩니다.

   로컬 클러스터 데이터베이스에 계속 연결합니다. 그러나 이제 세 부분으로 구성된 database.schema.table 표기법을 사용하여 datashare에서 생성된 데이터베이스와 스키마에서도 읽을 수도 있습니다. 사용자에게 표시되는 모든 데이터베이스에 걸쳐 쿼리를 수행할 수 있습니다. 이러한 데이터베이스는 클러스터의 로컬 데이터베이스이거나 datashare에서 생성된 데이터베이스일 수 있습니다. 또는 이러한 소비자 데이터베이스에 직접 연결하고 부분 표기법으로 공유 객체에 대해 쿼리를 실행할 수 있습니다.

전체 자격을 사용하여 데이터에 액세스할 수 있습니다. 자세한 내용은 데이터베이스 간 쿼리 예제 단원을 참조하십시오.

```
SELECT * FROM sales_db.public.tickit_sales_redshift ORDER BY 1,2 LIMIT 5;

 salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
 commission |      saletime
---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------
       1 |      1 |    36861 |   21191 |    7872 |   1875 |       4 |    728.00 |
   109.20 | 2008-02-18 02:36:48
       2 |      4 |     8117 |   11498 |    4337 |   1983 |       2 |     76.00 |
    11.40 | 2008-06-06 05:00:16
       3 |      5 |     1616 |   17433 |    8647 |   1983 |       2 |    350.00 |
    52.50 | 2008-06-06 08:26:17
       4 |      5 |     1616 |   19715 |    8647 |   1986 |       1 |    175.00 |
    26.25 | 2008-06-09 08:38:52
       5 |      6 |    47402 |   14115 |    8240 |   2069 |       2 |    154.00 |
    23.10 | 2008-08-31 09:17:02
```

공유 객체에는 SELECT 문만 사용할 수 있습니다. 그러나 다른 로컬 데이터베이스에 있는 공유 객체의 데이터를 쿼리하여 소비자 클러스터에 테이블을 생성할 수 있습니다.

쿼리 외에도 소비자는 공유 객체에 대한 뷰를 생성할 수 있습니다. 후기 바인딩 뷰나 구체화된 뷰만 지원됩니다. Amazon Redshift는 공유 데이터에 대한 일반 뷰를 지원하지 않습니다. 소비자가 생성하는 뷰는 여러 로컬 데이터베이스 또는 datashare에서 생성된 데이터베이스에 걸쳐 있을 수 있습니다. 자세한 내용은 CREATE VIEW 단원을 참조하십시오.

```
// Connect to a local cluster database

// Create a view on shared objects and access it.
CREATE VIEW sales_data
AS SELECT *
FROM sales_db.public.tickit_sales_redshift
WITH NO SCHEMA BINDING;

SELECT * FROM sales_data;
```

AWS Lake Formation 관리형 데이터 공유 시작하기

Amazon Redshift를 사용하면 AWS Lake Formation 관리형 데이터 공유를 통해 AWS 계정 및 Amazon Redshift 클러스터 전반에서 실시간 데이터에 액세스하고 공유할 수 있습니다. AWS Lake Formation 데이터 공유에서는 데이터 공급자가 다른 AWS 계정 및 Amazon Redshift 클러스터를 포함한 모든 소비자와 Amazon S3 데이터 레이크의 실시간 데이터를 안전하게 공유하도록 지원합니다.

생산자로서 Lake Formation에서 관리하는 데이터 공유 사용

Amazon Redshift로 AWS Lake Formation 데이터 공유를 통해 공유된 데이터에 액세스하고 분석합니다. AWS Lake Formation 데이터 공유를 사용하면 기본 데이터를 복사하거나 이전할 필요 없이 AWS 계정과 Amazon Redshift 클러스터 간에 안전한 데이터 공유가 가능합니다.

AWS Lake Formation에 데이터를 공유하면 Amazon Redshift 데이터 공유의 AWS Lake Formation 권한을 중앙에서 정의하고 데이터 공유 내의 객체에 대한 사용자 액세스를 제한할 수 있습니다.

Amazon Redshift가 있으면 AWS Lake Formation 관리형 데이터 공유를 생산자로 사용하여 AWS 계정 및 Amazon Redshift 클러스터 전반에서 실시간 데이터를 안전하게 공유할 수 있습니다. Lake Formation 관리형 데이터 공유는 Amazon Redshift 클러스터의 실시간 데이터를 다른 AWS 계정 및 서비스와 공유할 수 있는 객체입니다.

생산자 클러스터 또는 작업 그룹 관리자는 다음 단계에 따라 Lake Formation Formation에 데이터 공유를 공유하세요.

1. 클러스터에서 데이터 공유를 생성하고 데이터 공유에 액세스할 수 있도록 AWS Lake Formation에 권한을 부여합니다.

   클러스터 슈퍼 사용자와 데이터베이스 소유자만 datashare를 생성할 수 있습니다. 각 datashare는 생성하는 동안 데이터베이스와 연결됩니다. 해당 데이터베이스의 객체만 해당 datashare에서 공유할 수 있습니다. 동일하거나 다른 세부 수준의 객체를 사용하여 동일한 데이터베이스에 여러 datashare를 생성할 수 있습니다. 클러스터에서 생성할 수 있는 datashare 수에는 제한이 없습니다.

   ```
   CREATE DATASHARE salesshare;
   ```

2. 데이터 공유에 객체를 추가합니다. 생산자 클러스터 또는 작업 그룹 관리자는 사용 가능한 데이터 공유 객체를 계속 관리합니다. datashare에 객체를 추가하려면 객체를 추가하기 전에 스키마를 추가합니다. 스키마를 추가할 때 Amazon Redshift는 스키마 아래에 모든 객체를 추가하지 않습니다. 명시적으로 스키마를 추가해야 합니다. 자세한 내용은 ALTER DATASHARE를 참조하세요.

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;
ALTER DATASHARE salesshare ADD ALL TABLES IN SCHEMA PUBLIC;
```

datashare에 뷰를 추가할 수도 있습니다. 지원되는 는 표준 뷰, 후기 바인딩 뷰 및 구체화된 뷰입니다.

```
CREATE VIEW public.sales_data_summary_view AS SELECT * FROM
 public.tickit_sales_redshift;
ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;
```

ALTER DATASHARE를 사용하여 지정된 스키마에서 스키마, 테이블 및 뷰를 공유합니다. 슈퍼 사용자, datashare 소유자 또는 datashare에 대한 ALTER 또는 ALL 권한이 있는 사용자는 datashare를 변경하여 객체를 추가하거나 제거할 수 있습니다. 데이터베이스 사용자는 객체의 소유자이거나 객체에 대한 SELECT, USAGE 또는 ALL 권한이 있어야 합니다.

INCLUDENEW 절을 사용하여 지정된 스키마에서 생성된 새 테이블과 뷰를 데이터 공유에 추가합니다. 슈퍼 사용자만 각 datashare-스키마 페어에 대해 이 속성을 변경할 수 있습니다.

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare SET INCLUDENEW = TRUE FOR SCHEMA PUBLIC;
```

3. 수퍼유저만 각 데이터 공유-스키마 쌍에 대해 이 속성을 변경할 수 있습니다.

```
GRANT USAGE ON DATASHARE salesshare TO ACCOUNT '012345678910' VIA DATA CATALOG;
```

사용을 취소하려면 다음 명령을 사용하세요.

```
REVOKE USAGE ON DATASHARE salesshare FROM ACCOUNT '012345678910' VIA DATA CATALOG;
```

4. `aws redshift authorize-data-share` API 작업을 사용하여 Lake Formation의 데이터 공유에 대한 액세스 권한을 부여합니다. 이를 통해 Lake Formation은 서비스 계정에서 데이터 공유를 인식하고 데이터 공유에 대한 소비자 연결을 관리할 수 있습니다.

```
aws redshift authorize-data-share
--data-share-arn arn:aws:redshift:us-east-1:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/salesshare
--consumer-identifier {"DataCatalog/<consumer-account-id>"}
```

Lake Formation에서 관리하는 데이터 공유에서 권한을 제거하려면 `aws redshift deauthorize-data-share` API 작업을 사용하세요. 이렇게 하면 AWS Lake Formation이 서비스 계정에서 데이터 공유를 인식하고 권한 부여를 제거할 수 있습니다.

```
aws redshift deauthorize-data-share
--data-share-arn arn:aws:redshift:us-east-1:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/salesshare
--consumer-identifier {"DataCatalog/<consumer-account-id>"}
```

언제든지 생산자 클러스터 또는 작업 그룹 관리자가 더 이상 소비자 클러스터 또는 작업 그룹과 데이터를 공유할 필요가 없다고 판단하면 DROP DATASHARE를 사용하여 데이터 공유를 삭제하거나, 데이터 공유의 인증을 해제하거나, 데이터 공유 권한을 취소할 수 있습니다. Lake Formation의 연결된 권한 및 객체는 자동으로 삭제되지 않습니다.

```
DROP DATASHARE salesshare;
```

데이터 공유를 관리하도록 Lake Formation 계정에 권한을 부여한 후 Lake Formation 관리자는 공유 데이터 공유를 검색하고 데이터 카탈로그 ARN과 데이터 공유를 연결하고, 데이터 공유에 연결되는 데이터베이스를 AWS Glue Data Catalog 카탈로그에서 생성할 수 있습니다. AWS CLI를 사용하여 데이터 공유를 연결하려면 associate-data-share-consumer 명령을 사용합니다. AWS 리전 간에 데이터 공유를 공유하려면 `associate-data-share-consumer` 명령에 `--region` 파라미터를 지정하거나 AWS 콘솔을 사용하여 데이터 소비자를 선택합니다. 다음 예제는 Lake Formation에서 관리하는 데이터 공유를 여러 리전에서 공유하는 방법을 보여줍니다.

```
aws redshift associate-data-share-consumer --region <region-1>
--data-share-arn 'arn:aws:redshift:us-
east-1:12345678912:datashare:035c45ea-61ce-86f0-8b75-19ac6102c3b7/sample_share'
--consumer-arn 'arn:aws:glue:<region-1>:111912345678:catalog'
```

Lake Formation 관리자는 데이터 공유 내의 객체가 Lake Formation 내의 객체에 매핑되는 방법을 정의하는 로컬 리소스도 생성해야 합니다. 데이터 공유 검색 및 로컬 리소스 생성에 대한 자세한 내용은 Amazon Redshift 데이터 공유의 데이터에 대한 권한 관리를 참조하세요.

소비자로서 Lake Formation에서 관리하는 데이터 공유 사용

Amazon Redshift를 사용하면 AWS Lake Formation 데이터 공유를 통해 공유된 데이터에 액세스하고 분석할 수 있습니다. 데이터 공유는 다양한 데이터 소스의 테이블 또는 데이터베이스 등 데이터 객체 모음이 포함된 데이터 제품입니다.

AWS Lake Formation 관리자가 데이터 공유 초대를 검색하고 데이터 공유에 연결되는 AWS Glue Data Catalog에 데이터베이스를 생성한 후, 소비자 클러스터 또는 작업 그룹 관리자는 클러스터를 AWS Glue Data Catalog의 데이터 공유 및 데이터베이스와 연결하고 소비자 클러스터 또는 작업 그룹에 로컬 데이터베이스를 생성하며 Amazon Redshift 소비자 클러스터 또는 작업 그룹의 사용자 및 역할이 쿼리를 시작하도록 액세스 권한을 부여할 수 있습니다. 쿼리 권한을 설정하려면 다음 단계를 따르세요.

1. 필요한 경우 Amazon Redshift 콘솔에서 소비자 클러스터 또는 작업 그룹 역할을 할 Redshift 클러스터를 생성합니다. 클러스터 생성 방법에 대한 자세한 내용은 클러스터 생성을 참조하세요.
2. AWS Glue Data Catalog 소비자 클러스터 또는 작업 그룹 사용자가 액세스할 수 있는 데이터베이스를 나열하려면 SHOW DATABASES 명령을 실행합니다.

   ```
   SHOW DATABASES FROM DATA CATALOG [ACCOUNT <account-id>,<account-id2>] [LIKE
    <expression>]
   ```

   이렇게 하면 AWS Glue 데이터베이스의 ARN, 데이터베이스 이름 및 데이터 공유에 대한 정보와 같이 Data Catalog에서 사용할 수 있는 리소스가 나열됩니다.
3. SHOW DATABASES의 AWS Glue 데이터베이스 ARN을 사용하여 소비자 클러스터 또는 작업 그룹에 로컬 데이터베이스를 생성합니다. 자세한 내용은 Creating a database(데이터베이스 생성)를 참조하세요.

   ```
   CREATE DATABASE lf_db FROM ARN <lake-formation-database-ARN> WITH [NO] DATA CATALOG
    SCHEMA [<schema>];
   ```
4. 필요에 따라 소비자 클러스터 또는 작업 그룹의 사용자 및 역할에 데이터 공유에서 생성된 데이터베이스 및 스키마 참조에 대한 액세스 권한을 부여합니다. 자세한 내용은 GRANT 또는 REVOKE를 참조하세요. CREATE USER 명령으로 생성된 사용자는 Lake Formation에 공유된 데이터 공유의 객체에 액세스할 수 없습니다. Redshift와 Lake Formation 모두에 대한 액세스 권한이 있는 사용자만 Lake Formation과 공유된 데이터 공유에 액세스할 수 있습니다.

   ```
   GRANT USAGE ON DATABASE sales_db TO IAM:Bob;
   ```

소비자 클러스터 또는 작업 그룹 관리자는 데이터 공유에서 생성된 전체 데이터베이스에 대한 권한만 사용자와 역할에 할당할 수 있습니다. 경우에 따라 datashare에서 생성된 데이터베이스 객체의 하위 집합에 대한 세분화된 제어가 필요합니다.

또한 공유 객체 위에 후기 바인딩 뷰를 생성하고 이를 사용하여 세분화된 권한을 할당할 수 있습니다. 생산자 클러스터 또는 작업 그룹이 필요한 세분성으로 추가 데이터 공유를 생성하도록 할 수도 있습니다. datashare에서 생성된 데이터베이스에 대한 많은 스키마 참조를 생성할 수 있습니다.

5. 데이터베이스 사용자는 SVV_EXTERNAL_TABLES 및 SVV_EXTERNAL_COLUMNS 뷰를 사용하여 AWS Glue 데이터베이스 내의 모든 공유 테이블 또는 열을 찾을 수 있습니다.

```
SELECT * from svv_external_tables WHERE redshift_database_name = 'lf_db';

SELECT * from svv_external_columns WHERE redshift_database_name = 'lf_db';
```

6. datashare의 공유 객체에서 데이터를 쿼리합니다.

소비자 데이터베이스 및 소비자 클러스터 또는 작업 그룹의 스키마에 대한 권한이 있는 사용자와 역할은 모든 공유 객체의 메타데이터를 탐색할 수 있습니다. 또한 소비자 클러스터 또는 작업 그룹에서 로컬 객체를 탐색할 수 있습니다. 이를 위해 JDBC 또는 ODBC 드라이버나 SVV_ALL 및 SVV_EXTERNAL 뷰를 사용할 수 있습니다.

```
SELECT * FROM lf_db.schema.table;
```

공유 객체에는 SELECT 문만 사용할 수 있습니다. 그러나 다른 로컬 데이터베이스에 있는 공유 객체의 데이터를 쿼리하여 소비자 클러스터에 테이블을 생성할 수 있습니다.

```
// Connect to a local cluster database

// Create a view on shared objects and access it.

CREATE VIEW sales_data
AS SELECT *
FROM sales_db.public.tickit_sales_redshift
WITH NO SCHEMA BINDING;

SELECT * FROM sales_data;
```

Amazon Redshift에서 데이터 공유를 사용하여 다중 웨어하우스 쓰기 시작하기

동일한 AWS 계정에 있는 서로 다른 Amazon Redshift 클러스터 또는 Amazon Redshift Serverless 작업 그룹 내에서 계정과 리전 간에 읽기 및 쓰기 모두에 대해 데이터베이스 객체를 공유할 수 있습니다. 이 주제에 나온 절차는 쓰기 권한이 포함된 데이터 공유 설정 방법을 보여줍니다. 서로 다른 테이블에 대해 SELECT, INSERT, UPDATE 등의 권한을 부여하고 스키마에 대해 USAGE 및 CREATE와 같은 권한을 부여할 수 있습니다.

데이터는 쓰기 트랜잭션을 커밋하자마자 모든 웨어하우스에서 라이브 상태가 되며 사용 가능하게 됩니다. 생산자 계정 관리자는 특정 네임스페이스나 리전에 읽기 전용, 읽기 및 쓰기 또는 데이터에 대한 모든 액세스 권한을 부여할지를 결정할 수 있습니다. 이 절차에서는 프로비저닝된 클러스터 또는 Amazon Redshift Serverless 작업 그룹의 데이터베이스에서 작업하는 것으로 가정합니다.

Amazon Redshift를 사용하면 콘솔 또는 SQL 인터페이스를 사용하여 쓰기로 데이터 공유를 관리하여 Amazon Redshift 클러스터 및 AWS 계정 전반의 데이터에 대한 액세스를 제어하고 데이터를 통제할 수 있습니다. 다음 섹션에서는 Amazon Redshift를 사용하여 쓰기로 데이터 공유를 구성하고 관리하는 방법에 대한 단계별 지침을 제공합니다.

데이터 공유를 사용할 수 있는 리전 목록은 데이터 공유가 가능한 AWS 리전 섹션을 참조하세요. 쓰기 관련 고려 사항 및 제한 사항은 Amazon Redshift의 데이터 공유 고려 사항 섹션을 참조하세요.

> **Note**
>
> 데이터 공유를 사용하는 Amazon Redshift 다중 웨어하우스 쓰기는 현재 트랙 버전 1.0.78881 이상의 프로비저닝된 클러스터에 대한 Amazon Redshift 패치 186과 버전 1.0.78890 이상의 Amazon Redshift Serverless 작업 그룹에 대해서만 지원됩니다.

주제

Amazon Redshift에서 데이터베이스에 연결

Amazon Redshift를 사용하면 데이터 웨어하우스 클러스터에 대한 연결을 설정하고 SQL 쿼리를 실행하거나 데이터를 로드하거나 관리 작업을 수행할 수 있습니다. 데이터베이스에 연결하는 것은 클라이언트 애플리케이션 또는 도구와 Amazon Redshift 클러스터 간에 보안 채널을 만드는 과정입니다. 다음 섹션에서는 Amazon Redshift 데이터베이스에 연결하는 방법에 대한 단계별 지침을 제공합니다.

Console

데이터베이스에 연결하여 데이터베이스 내에서 데이터베이스와 객체를 보거나 Amazon Redshift 데이터 웨어하우스에서 데이터 공유를 봅니다. 모든 datashare를 보려면 지정된 데이터베이스에 연결하는 데 사용되는 사용자 자격 증명에 필요한 권한이 있어야 합니다.

로컬 연결이 없는 경우 다음 중 하나를 수행합니다.

- 생산자 관리자인 경우 프로비저닝된 클러스터의 클러스터 탭 또는 Serverless 엔드포인트의 네임스페이스 구성 탭으로 이동합니다. 목록에서 해당 클러스터 또는 네임스페이스를 선택합니다.
- 클러스터 또는 네임스페이스 세부 정보 페이지의 데이터 공유 탭에서 데이터베이스에 연결을 선택하고 다음 작업 중 하나를 수행합니다.
  - 다른 네임스페이스 및 AWS 계정의 데이터 공유 섹션에서 다른 클러스터, 네임스페이스 또는 계정의 데이터 공유를 봅니다.
  - 내 클러스터에서 생성된 데이터 공유 섹션에서 클러스터의 데이터 공유를 봅니다.
- [데이터베이스에 연결(Connect to database)] 창에서 다음 중 하나를 수행합니다.
  - [새 연결 생성(Create a new connection)]을 선택하는 경우 [AWS Secrets Manager]를 선택하여 저장된 보안 암호로 연결에 대한 액세스를 인증합니다.

    또는 [임시 자격 증명(Temporary credentials)]을 선택하여 데이터베이스 자격 증명으로 연결에 대한 액세스를 인증합니다. [데이터베이스 이름(Database name)] 및 [데이터베이스 사용자(Database user)] 값을 지정합니다.

    연결을 선택합니다.
  - [최근 연결 사용(Use a recent connection)]을 선택하여 필요한 권한이 있는 다른 데이터베이스에 연결합니다.

    Amazon Redshift에서 자동으로 연결을 설정합니다.

데이터베이스 연결이 설정되면 데이터 공유 만들기, 데이터 공유 쿼리 또는 데이터 공유에서 데이터베이스 만들기를 시작할 수 있습니다.

Amazon Redshift에서 새 데이터 공유에 대한 생산자 작업

Amazon Redshift가 있으면 데이터 공유를 사용하여 Amazon Redshift 클러스터 또는 AWS 계정 전반에서 실시간 데이터를 공유할 수 있습니다. 데이터 공유는 Amazon Redshift 클러스터의 실시간 데이터를 다른 클러스터 또는 AWS 계정과 공유할 수 있는 소비자-생산자 객체입니다. 데이터 공유를 만들어 액세스에 대한 제어를 유지 관리하고 데이터를 최신 상태로 유지하면서 안전한 데이터 공유가 가능합니다. 다음 섹션에서는 데이터 공유를 만들고 스키마, 테이블 및 뷰와 같은 데이터베이스 객체를 추가하여 실시간 데이터를 안전하게 공유하는 방법에 대한 세부 정보를 제공합니다.

주제

- Amazon Redshift에서 데이터 공유 만들기
- Amazon Redshift에서 데이터 공유에 객체 추가
- Amazon Redshift에서 데이터 공유에 데이터 소비자 추가
- Amazon Redshift에서 데이터 공유 권한 부여

Amazon Redshift에서 데이터 공유 만들기

데이터 공유는 데이터베이스 객체, 권한 및 소비자가 포함된 논리적 컨테이너입니다. 소비자는 사용자 계정 및 기타 AWS 계정의 Amazon Redshift 프로비저닝된 클러스터 또는 Amazon Redshift Serverless 네임스페이스입니다. 각 데이터 공유는 해당 데이터 공유가 생성된 데이터베이스와 연결되며 해당 데이터베이스의 객체만 추가할 수 있습니다. 생산자 관리자는 아래 절차 중 하나를 수행하여 콘솔과 SQL에서 데이터 공유를 만들 수 있습니다.

Console

콘솔에서 클러스터 또는 네임스페이스 세부 정보 페이지의 데이터 공유 탭에서 데이터 공유를 만들 수 있습니다. 데이터 공유가 만들어진 후 소비자 관리자로서 소비자의 데이터 공유에서 데이터베이스를 만들 수 있습니다.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 클러스터(Clusters)를 선택한 다음 클러스터를 선택합니다. 클러스터 세부 정보 페이지가 나타납니다.
3. 데이터베이스 연결이 없는 경우 클러스터 또는 네임스페이스 세부 정보 페이지에서 데이터 공유 탭의 데이터 공유 섹션에서 데이터베이스에 연결합니다. 내 계정에서 생성된 데이터 공유 섹션에서 데이터 공유 생성을 선택합니다. 데이터 공유 생성 페이지가 나타납니다.

4. [datashare 생성(Create datashare)]을 선택합니다. 로컬 데이터베이스에서만 datashare를 생성할 수 있습니다. 데이터베이스에 연결하지 않은 경우 데이터베이스에 연결 페이지가 나타납니다. 데이터베이스로 연결의 절차에 따라 데이터베이스에 연결합니다. 최근 연결이 있는 경우 데이터 공유 생성 페이지가 나타납니다.
5. Datashare 정보(Datashare information) 섹션에서 다음 중 하나를 선택합니다.
   - 데이터 공유를 선택하여 각기 다른 Amazon Redshift 데이터 웨어하우스(프로비저닝된 클러스터 또는 Serverless 엔드포인트)에서 또는 동일한 AWS 계정이나 다른 AWS 계정에서 읽기 또는 쓰기 목적으로 데이터를 공유할 데이터 공유를 만듭니다.
   - AWS Data Exchange datashare를 선택하여 AWS Data Exchange를 통해 데이터에 라이선스를 부여할 datashare를 생성합니다.
6. Datashare 이름(Datashare name), 데이터베이스 이름(Database name) 및 공개적으로 액세스 가능(Publicly accessible) 값을 지정합니다. 데이터베이스 이름을 변경할 때 새 데이터베이스 연결을 만듭니다.
7. 범위가 지정된 권한 또는 직접 권한 섹션을 사용하여 데이터 공유에 객체를 추가합니다. 데이터 공유에 객체를 추가하려면 Amazon Redshift에서 데이터 공유 만들기 섹션을 참조하세요.
8. 데이터 소비자 섹션에서 Amazon Redshift에 게시하거나 Lake Formation과 데이터 공유 프로세스를 시작하는 AWS Glue Data Catalog에 게시하도록 선택할 수 있습니다. 데이터 공유를 Amazon Redshift에 게시한다는 것은 다른 네임스페이스 또는 소비자 역할을 하는 Amazon Redshift 계정과 데이터를 공유한다는 의미입니다.

   > **Note**
   >
   > 데이터 공유가 생성되면 구성을 편집하여 다른 옵션에 게시할 수 없습니다.

9. [datashare 생성(Create datashare)]을 선택합니다.

SQL

다음 명령을 실행하여 데이터 공유를 생성합니다.

```
CREATE DATASHARE salesshare;
```

데이터 공유를 만들 때 각 데이터 공유는 데이터베이스와 연결됩니다. 해당 데이터베이스의 객체만 해당 datashare에서 공유할 수 있습니다. 동일하거나 다른 세부 수준의 객체를 사용하여 동일한 데이터베이스에 여러 datashare를 생성할 수 있습니다. 클러스터가 생성할 수 있는 datashare 수에는

제한이 없습니다. Amazon Redshift 콘솔을 사용하여 datashare를 생성할 수도 있습니다. 자세한 내용은 CREATE DATASHARE 단원을 참조하십시오.

만드는 중에 데이터 공유에 대한 보안 제한을 제어할 수도 있습니다. 다음 예에서는 퍼블릭 IP 액세스 권한이 있는 소비자가 데이터 공유를 읽을 수 있음을 보여줍니다.

```
CREATE DATASHARE my_datashare [PUBLICACCESSIBLE = TRUE];
```

PUBLICACCESSIBLE = TRUE로 설정하면 소비자가 공개적으로 액세스할 수 있는 클러스터 및 프로비저닝된 작업 그룹에서 데이터 공유를 쿼리할 수 있습니다. 허용하지 않으려면 이 옵션을 생략하거나 명시적으로 false로 설정하세요.

데이터 공유를 만든 후 소비자 유형에 대한 속성을 수정할 수 있습니다. 예를 들어 지정된 datashare의 데이터를 소비하려는 클러스터에 공개적으로 액세스할 수 없도록 정의할 수 있습니다. datashare에 지정된 보안 제한을 충족하지 않는 소비자 클러스터의 쿼리는 쿼리 런타임에 거부됩니다. 자세한 내용은 ALTER DATASHARE 단원을 참조하십시오.

Amazon Redshift에서 데이터 공유에 객체 추가

다음 절차 중 하나를 수행하여 콘솔 및 SQL에서 다양한 유형의 데이터베이스 객체를 추가할 수 있습니다.

Console

범위가 지정된 권한 또는 직접 권한 섹션을 사용하여 데이터 공유에 객체를 추가할 수 있습니다. 객체를 추가할 범위가 지정된 권한 부여 또는 직접 권한 부여를 선택합니다. 추가 버튼을 선택하여 객체를 추가합니다. 대화 상자가 나타납니다. 다음 단계를 수행합니다.

1. 범위가 지정된 권한 부여를 선택하면 데이터베이스 또는 스키마 수준에서 범위가 지정된 권한을 부여할 수 있는 범위가 지정된 권한 부여 페이지가 나타납니다. 범위가 지정된 권한이 있는 데이터 공유는 데이터베이스 또는 스키마 내의 모든 현재 및 미래 객체에 대해 지정된 권한을 갖습니다. 자세한 내용은 범위가 지정된 권한 항목을 참조하세요.

    1. 그런 다음 데이터베이스 범위가 지정된 권한을 선택하여 데이터베이스 수준에서 범위가 지정된 권한을 부여합니다. 범위가 지정된 권한을 부여하면 데이터 공유를 만드는 동안 현재 데이터베이스에 적용됩니다. 이러한 권한은 개별 객체에 부여할 수 없으며 기존 객체와 새 객체(스키마, 테이블, 뷰, UDF) 모두에 적용됩니다.

2. 스키마, 테이블 또는 뷰 또는 사용자 정의 함수에 범위가 지정된 권한을 하나 이상 선택합니다. 이렇게 하면 데이터베이스의 모든 객체가 소비자에게 부여된 선택된 권한을 갖게 됩니다. 데이터베이스 범위가 지정된 권한 부여를 완료하려면 부여를 선택합니다.
3. 그런 다음 스키마 범위가 지정된 권한을 선택하여 스키마 수준에서 범위가 지정된 권한을 부여합니다. 스키마 범위가 지정된 권한을 부여하면 스키마에 추가된 모든 객체가 지정된 데이터 공유 권한을 갖게 됩니다.
4. 드롭다운에서 데이터 공유에 추가할 스키마를 선택합니다. 한 번에 하나의 스키마만 선택할 수 있습니다. 그런 다음 선택한 스키마에 부여하려는 직접 권한을 하나 이상 선택합니다.
5. 테이블, 뷰 및 사용자 정의 함수와 같은 스키마 객체에 범위가 지정된 권한을 하나 이상 선택합니다. 스키마의 매칭되는 모든 객체에 권한이 부여됩니다. 이러한 객체는 기존 객체이거나 향후 추가될 객체일 수 있습니다. 적용되면 범위가 지정된 권한을 취소하지 않고 객체에서 권한을 제거할 수 없습니다.
6. 스키마 범위가 지정된 권한 부여를 완료하려면 부여를 선택합니다.

2. 직접 권한 부여를 선택하면 직접 권한 부여 페이지가 나타나 스키마, 테이블, 뷰 또는 사용자 정의 함수와 같은 각 객체 수준에서 직접 권한을 부여할 수 있습니다. 직접 권한을 부여하려면 먼저 데이터 공유에 관련 스키마를 추가해야 합니다.

   1. 그런 다음 스키마에 직접 권한 부여를 선택하여 특정 스키마에 직접 권한을 적용합니다. 그런 다음 테이블, 뷰 및 사용자 정의 함수와 같은 스키마 객체의 스키마 권한을 하나 이상 선택하고 데이터 공유에 추가할 스키마를 선택합니다. 부여를 선택하여 데이터 공유에 스키마 추가를 완료합니다.
   2. 데이터 공유에 스키마를 추가한 후 스키마 객체에 대한 직접 권한 추가를 진행할 수 있습니다. 직접 권한 부여를 다시 선택합니다. 직접 권한 부여 페이지가 나타납니다. 그런 다음 스키마 객체에 대한 직접 권한 탭으로 이동합니다.
   3. 테이블 및 뷰에 직접 권한 부여를 선택하여 이러한 객체에 객체 수준 직접 권한을 부여합니다. 목록에서 필요한 직접 권한 하나 이상과 필요한 객체를 선택합니다. 검색 필드를 사용하여 데이터 공유 객체를 찾습니다. 부여를 선택하여 데이터 공유에 테이블 및 뷰 추가를 완료합니다.
   4. 사용자 정의 함수에 직접 권한 부여를 선택하여 사용자 정의 함수에 객체 수준 직접 권한을 부여합니다. 목록에서 필요한 직접 권한 하나 이상과 필요한 객체를 선택합니다. 검색 필드를 사용하여 데이터 공유 객체를 찾습니다. 부여를 선택하여 데이터 공유에 사용자 정의 함수 추가를 완료합니다.

3. 미래 객체를 추가할지도 선택할 수 있습니다. 스키마에 추가된 데이터 공유 객체를 포함하도록 선택하면 스키마에 추가된 객체가 데이터 공유에 자동으로 추가됩니다.

4. 추가를 선택하여 섹션을 완료하고 객체를 추가합니다. 추가된 객체는 데이터 공유 객체에 나열됩니다.
5. 객체를 추가한 후 개별 객체를 선택하고 권한을 편집할 수 있습니다. 스키마를 선택하면 범위가 지정된 권한을 추가할지 묻는 대화 상자가 나타납니다. 이렇게 하면 스키마의 기존 객체 또는 추가된 각 객체가 객체 유형에 적합한 미리 선택된 권한 세트를 갖게 됩니다. 예를 들어 관리자는 추가된 모든 테이블에 SELECT 및 UPDATE 권한을 갖도록 설정할 수 있습니다.
6. 모든 데이터 공유 객체는 범위가 지정된 권한 또는 직접 권한 섹션에 나열됩니다.
7. 데이터 소비자 섹션에서 네임스페이스를 추가하거나 AWS 계정을 데이터 공유의 소비자로 추가할 수 있습니다.
8. 데이터 공유 생성을 선택하여 변경 사항을 저장합니다.

데이터 공유를 만들면 내 네임스페이스에서 생성된 데이터 공유 아래 목록에 해당 데이터 공유가 나타납니다. 목록에서 데이터 공유를 선택하면 해당 소비자, 객체 및 기타 속성을 볼 수 있습니다.

SQL

SQL을 사용하면 데이터 공유 소유자는 데이터 공유에 추가할 스키마에 대해 USAGE 권한을 부여해야 합니다. GRANT는 스키마에서 CREATE 및 USAGE를 포함하여 다양한 작업을 허용하는 데 사용됩니다. 스키마에는 공유 객체가 들어 있습니다.

```
CREATE SCHEMA myshared_schema1;
CREATE SCHEMA myshared_schema2;

GRANT USAGE ON SCHEMA myshared_schema1 TO DATASHARE my_datashare;
GRANT CREATE, USAGE ON SCHEMA myshared_schema2 TO DATASHARE my_datashare;
```

또는 관리자가 ALTER 명령을 계속 실행하여 데이터 공유에 스키마를 추가할 수도 있습니다. 이러한 방식으로 스키마를 추가할 경우 USAGE 권한만 부여됩니다.

```
ALTER DATASHARE my_datashare ADD SCHEMA myshared_schema1;
```

관리자는 스키마를 추가한 후 스키마의 객체에 대해 데이터 공유 권한을 부여할 수 있습니다. 이 권한은 읽기 및 쓰기 권한일 수 있습니다. GRANT ALL 샘플은 모든 권한을 부여하는 방법을 보여줍니다.

```
GRANT SELECT, INSERT ON TABLE myshared_schema1.table1, myshared_schema1.table2,
 myshared_schema2.table1
TO DATASHARE my_datashare;
```

```
GRANT ALL ON TABLE myshared_schema1.table4 TO DATASHARE my_datashare;
```

계속해서 ALTER DATASHARE와 같은 명령을 실행하여 테이블을 추가할 수 있습니다. 이렇게 하면 추가된 객체에 대해 SELECT 권한만 부여됩니다.

```
ALTER DATASHARE my_datashare ADD TABLE myshared_schema1.table1,
 myshared_schema1.table2, myshared_schema2.table1;
```

데이터베이스 또는 스키마 내 특정 유형의 모든 객체에 대해 범위가 지정된 권한을 데이터 공유에 부여할 수 있습니다. 범위가 지정된 권한이 있는 데이터 공유는 데이터베이스 또는 스키마 내의 모든 현재 및 미래 객체에 대해 지정된 권한을 갖습니다.

SVV_DATABASE_PRIVILEGES에서 데이터베이스 수준 범위 지정 권한의 범위를 볼 수 있습니다. SVV_SCHEMA_PRIVILEGES에서 스키마 수준 범위 지정 권한의 범위를 볼 수 있습니다.

다음은 데이터 공유에 범위가 지정된 권한을 부여할 때 사용하는 구문입니다. 범위가 지정된 권한에 대한 자세한 내용은 범위가 지정된 권한 섹션을 참조하세요.

```
GRANT { CREATE | USAGE | ALTER | DROP } [,...] | ALL [ PRIVILEGES ] }FOR SCHEMAS IN
DATABASE db_name
TO DATASHARE { datashare_name}

GRANT { { SELECT | INSERT | UPDATE | DELETE | DROP | ALTER | TRUNCATE | REFERENCES }
 [, ...] } | ALL [PRIVILEGES] } }FOR TABLES IN
{SCHEMA schema_name [DATABASE db_name ] | DATABASE db_name }
TO DATASHARE { datashare_name}

GRANT { EXECUTE | ALL [ PRIVILEGES ] }FOR FUNCTIONS IN
{SCHEMA schema_name [DATABASE db_name ] | DATABASE db_name }
TO DATASHARE { datashare_name}
```

Amazon Redshift에서 데이터 공유에 데이터 소비자 추가

콘솔 또는 SQL을 사용하여 데이터 공유에 데이터 소비자를 하나 이상 추가할 수 있습니다. 데이터 소비자는 Amazon Redshift 클러스터 또는 AWS 계정을 고유하게 식별한 네임스페이스일 수 있습니다.

Console

퍼블릭 액세스 권한이 있는 클러스터에서 datashare를 해제하거나 설정하도록 명시적으로 선택해야 합니다.

- 데이터 공유에 네임스페이스 추가를 선택합니다. 네임스페이스는 Amazon Redshift 클러스터에 대한 GUID(전역 고유 식별자)입니다.
- datashare에 AWS 계정 추가(Add)를 선택합니다. 지정된 AWS 계정에 datashare에 대한 액세스 권한이 있어야 합니다.

SQL

SQL을 사용하여 관리자는 계정의 특정 네임스페이스에 데이터 공유 사용 권한을 부여합니다. ARN의 일부인 네임스페이스 ID는 클러스터 세부 정보 페이지, Amazon Redshift Serverless 네임스페이스 세부 정보 페이지 또는 `SELECT current_namespace;` 명령을 실행하여 찾을 수 있습니다. 자세한 내용은 CURRENT_NAMESPACE를 참조하세요.

```
GRANT USAGE ON DATASHARE my_datashare TO NAMESPACE '86b5169f-012a-234b-9fbb-
e2e24359e9a8';
```

다음은 AWS 계정에 데이터 공유 사용 권한을 부여하는 방법의 예입니다.

```
GRANT USAGE ON DATASHARE salesshare TO ACCOUNT '123456789012';
```

다음은 Lake Formation 계정에 데이터 공유 사용 권한을 부여하는 방법의 예입니다.

```
GRANT USAGE ON DATASHARE salesshare TO ACCOUNT '123456789012' VIA DATA CATALOG;
```

Amazon Redshift에서 데이터 공유 권한 부여

Amazon Redshift를 사용하면 지정된 소비자에 대한 권한을 부여하여 데이터 공유에 대한 액세스를 제어할 수 있습니다. 데이터 공유를 통해 동일하거나 다른 AWS 계정의 Amazon Redshift 클러스터 간에 실시간 데이터를 공유할 수 있으므로 원활한 방법으로 분석 데이터를 배포하고 사용합니다. 이 섹션에서는 Amazon Redshift에서 데이터 공유에 대한 소비자 액세스 권한을 부여하고 취소하기 위한 단계별 지침을 제공합니다.

> **Note**
>
> 네임스페이스를 데이터 소비자로 추가하는 경우 권한 부여를 수행할 필요가 없습니다. 데이터 공유에 대한 권한을 부여하려면 데이터 공유에 데이터 소비자가 하나 이상 추가되어 있어야 합니다.

Console

콘솔에서 생산자 관리자는 데이터 공유에 대한 액세스 권한을 부여하거나 제거할 데이터 소비자를 선택할 수 있습니다. 권한이 부여된 데이터 소비자는 datashare에 대한 작업을 수행하라는 알림을 받습니다. 네임스페이스를 데이터 소비자로 추가하는 경우 권한 부여를 수행할 필요가 없습니다.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 Datashares를 선택합니다. 여기에서 데이터 공유 소비자라는 목록을 볼 수 있습니다. 승인하려는 하나 이상의 소비자 클러스터를 선택합니다. 그런 다음 권한 부여(Authorize)를 선택합니다.
3. 계정 권한 부여 대화 상자가 나타납니다. 몇 가지 권한 부여 유형 중에서 선택할 수 있습니다.
   - [클러스터 이름 또는 작업 그룹 이름]에 대해 읽기 전용 - 이 옵션은 데이터 공유 생성자가 쓰기 권한을 부여했더라도 소비자는 쓰기 권한을 사용할 수 없다는 뜻입니다.
   - [클러스터 이름 또는 작업 그룹 이름]에 대해 읽기 및 쓰기 - 이 옵션은 쓰기 권한을 포함하여 생산자가 부여한 모든 권한을 소비자가 사용할 수 있다는 뜻입니다.
4. Save(저장)를 선택합니다.

소비자로서 AWS Data Exchange 권한을 부여할 수도 있습니다.

1. 데이터 공유를 생성할 때 Publish to AWS Glue Data Catalog(Glue 데이터 카탈로그에 게시)를 선택한 경우 데이터 공유 권한을 Lake Formation 계정에만 부여할 수 있습니다.

   AWS Data Exchange datashare의 경우 한 번에 하나의 datashare에만 권한을 부여할 수 있습니다.

   AWS Data Exchange datashare에 권한을 부여하면 AWS Data Exchange 서비스와 datashare를 공유하고 AWS Data Exchange에서 사용자를 대신하여 datashare에 대한 액세스를 관리하도록 허용합니다. AWS Data Exchange에서는 소비자가 제품을 구독할 때 AWS Data Exchange datashare에 소비자 계정을 데이터 소비자로 추가하여 소비자에 대한 액세스를 허용합니다. AWS Data Exchange에는 datashare에 대한 읽기 권한이 없습니다.
2. Save(저장)를 선택합니다.

권한을 부여 받은 데이터 소비자는 datashare 객체에 액세스하고 소비자 데이터베이스를 생성하여 데이터를 쿼리할 수 있습니다.

API

생산자 보안 관리자는 다음을 결정합니다.

- 다른 계정이 데이터 공유에 액세스할 수 있는지 여부
- 계정에 데이터 공유에 대한 액세스 권한이 있는 경우, 해당 계정에 쓰기 권한이 있는지 여부

데이터 공유 권한을 부여하려면 다음과 같은 IAM 권한이 필요합니다.

redshift:AuthorizeDataShare

CLI 호출 또는 API를 사용하여 사용 및 쓰기 권한을 부여할 수 있습니다.

```
authorize-data-share
--data-share-arn <value>
--consumer-identifier <value>
[--allow-writes | --no-allow-writes]
```

명령에 대한 자세한 내용은 authorize-data-share를 참조하세요.

소비자 식별자는 다음 중 하나일 수 있습니다.

- 12자리 AWS 계정 ID
- 네임스페이스 식별자 ARN

> **Note**
>
> 권한 부여 단계에서는 쓰기 권한이 부여되지 않습니다. 데이터 공유에 쓰기 권한을 부여하면 데이터 공유 관리자가 부여한 쓰기 권한만 계정에 허용됩니다. 관리자가 쓰기를 허용하지 않는 경우 해당 소비자가 사용할 수 있는 권한은 SELECT, USAGE, EXECUTE뿐입니다.

다른 값으로 `authorize-data-share`를 다시 호출하여 데이터 공유 소비자의 권한 부여를 변경할 수 있습니다. 새로운 권한 부여가 이전의 권한 부여를 덮어씁니다. 즉, 처음에 쓰기 권한을 부여하고 허용했지만 다시 권한을 부여하고 `no-allow-writes`를 지정하거나 아니면 단순히 값을 지정하지 않는 경우 소비자의 쓰기 권한이 취소됩니다.

Amazon Redshift에서 새 데이터 공유에 대한 소비자 작업

Amazon Redshift를 사용하면 다른 AWS 계정의 데이터 공유를 사용하여 계정 간 데이터 공유 및 협업을 사용할 수 있습니다. 데이터 공유는 다른 AWS 계정에 있더라도 Amazon Redshift 클러스터 간에 실시간 데이터를 공유하는 안전한 방법입니다. 다음 섹션에서는 액세스를 구성하고, 데이터 공유에서 데이터베이스를 만들고, 객체 수준 권한을 부여하고, 공유 데이터를 쿼리하는 자세한 단계를 제공합니다.

주제

Amazon Redshift에서 다른 AWS 계정의 데이터 공유 연결

Amazon Redshift를 사용하면 다른 AWS 계정에서 공유한 데이터 공유를 연결할 수 있으므로 조직 경계 전반에서 원활하고 안전한 데이터 공유가 가능합니다. 데이터 공유는 하나 이상의 Amazon Redshift 데이터베이스에서 데이터를 캡슐화하는 공유 가능한 데이터베이스 객체입니다. 다음 섹션에서는 데이터 공유를 연결하는 프로세스를 보여줍니다.

Console

소비자 관리자는 다른 계정에서 공유되는 하나 이상의 데이터 공유를 전체 AWS 계정 또는 계정의 특정 네임스페이스에 연결할 수 있습니다.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 Datashares를 선택합니다. datashare 목록 페이지가 나타납니다. 다른 계정에서(From other accounts)를 선택합니다.
3. 다른 계정의 datashare(Datashares from other accounts) 섹션에서 연결할 datashare를 선택하고 연결(Associate)을 선택합니다. 데이터 공유 연결 페이지가 나타나면 다음 연결 유형 중 하나를 선택합니다.

   - 전체 AWS 계정을 선택하여 AWS 계정의 여러 AWS 리전에 있는 기존 및 미래 클러스터 네임스페이스를 모두 데이터 공유와 연결합니다.
   - 데이터 공유가 AWS Glue Data Catalog 데이터 카탈로그에 게시된 경우 데이터 공유를 전체 AWS 계정에만 연결할 수 있습니다.

4. 여기에서 허용된 권한을 선택할 수 있습니다. 선택할 수 있는 항목은 다음과 같습니다.

   - 읽기 전용 - 읽기 전용을 선택하면 생산자 측에서 UPDATE 또는 INSERT와 같은 쓰기 권한을 부여했더라도 소비자가 해당 권한을 사용할 수 없습니다.
   - 읽기 및 쓰기 - 소비자 데이터 공유 사용자는 생산자가 부여하고 승인한 모든 읽기 및 쓰기 권한을 갖게 됩니다.
5. 특정 AWS 리전 및 클러스터 네임스페이스를 선택하여 하나 이상의 AWS 리전 및 특정 네임스페이스를 데이터 공유와 연결합니다. 리전 추가를 선택하여 데이터 공유에 특정 AWS 리전 및 네임스페이스를 추가합니다. Add AWS Region (AWS 리전 추가) 페이지가 나타납니다.
6. AWS Region.리전을 선택합니다.
7. 다음 중 하나를 수행합니다.

   - 모든 네임스페이스 추가를 선택하여 이 리전의 모든 기존 및 향후 네임스페이스를 데이터 공유에 추가합니다.
   - 특정 네임스페이스 추가를 선택하여 데이터 공유에 이 리전의 특정 네임스페이스를 하나 이상 추가합니다.
   - 네임스페이스를 하나 이상 선택하고 AWS 리전 추가를 선택합니다.
8. 연결을 선택합니다.

생산자가 돌아가서 권한 부여에 대한 설정을 변경할 수 있으며, 이는 소비자의 연결 설정에 영향을 미칠 수 있습니다.

데이터 공유를 Lake Formation 계정과 연결하는 경우 Lake Formation 콘솔로 이동하여 데이터베이스를 만든 다음 데이터베이스에 대한 권한을 정의합니다. 자세한 내용은 AWS Lake Formation 개발자 안내서의 Amazon Redshift 데이터 공유에 대한 권한 설정을 참조하세요. AWS Glue 데이터베이스 또는 페더레이션 데이터베이스를 생성하면 소비자 클러스터와 함께 쿼리 편집기 v2 또는 선호하는 SQL 클라이언트를 사용하여 데이터를 쿼리할 수 있습니다.

datashare가 연결되면 datashare를 사용할 수 있게 됩니다.

> **Note**
>
> datashare 연결을 언제든지 변경할 수도 있습니다. 특정 AWS 리전 및 네임스페이스에서 전체 AWS 계정으로 연결을 변경할 때 Amazon Redshift는 특정 리전 및 네임스페이스 정보를 AWS 계정 정보로 덮어씁니다. 그러면 AWS 계정의 모든 AWS 리전 및 네임스페이스가 데이터 공유에 액세스할 수 있습니다.

API

> **Note**
>
> 이 섹션의 단계는 생산자 관리자가 공유 데이터베이스 객체에 대한 특정 작업을 허용하고, 데이터 공유를 다른 계정과 공유하는 경우 생산자 보안 관리자가 액세스를 권한을 부여한 후에 수행됩니다.

소비자 보안 관리자는 다음을 결정합니다.

- 계정의 모든 네임스페이스인지, 계정 내 특정 리전의 네임스페이스인지 또는 특정 네임스페이스가 데이터 공유에 액세스할 수 있는지 여부
- 네임스페이스가 데이터 공유에 액세스할 수 있는 경우 해당 네임스페이스에 쓰기 권한이 있는지 여부

소비자 보안 관리자는 데이터 공유를 다음 명령과 연결할 수 있습니다.

```
associate-data-share-consumer
--data-share-arn <value>
--consumer-identifier <value>
[--allow-writes | --no-allow-writes]
```

명령에 대한 자세한 내용은 associate-data-share-consumer를 참조하세요.

소비자 보안 관리자는 데이터 공유를 네임스페이스와 연결할 때 INSERT 및 UPDATE 명령을 사용할 수 있도록 `allow-writes`를 명시적으로 true로 설정해야 합니다. 이렇게 하지 않으면 사용자는 SELECT, USAGE 또는 EXECUTE 권한과 같은 읽기 작업만 수행할 수 있습니다.

다른 값으로 `associate-data-share-consumer`를 다시 호출하여 데이터 공유의 네임스페이스 연결을 변경할 수 있습니다. 새 연결이 이전 연결을 덮어쓰므로 처음에 `allow-writes`를 연결하고 설정했더라도 `no-allow-writes`를 연결하고 지정하거나 아니면 단순히 값을 지정하지 않는 경우 소비자의 쓰기 권한이 취소됩니다.

Amazon Redshift의 데이터 공유에서 데이터베이스 만들기

Amazon Redshift를 사용하면 데이터 공유를 사용하여 데이터베이스를 만든 후 생산자 클러스터에서 데이터 공유 전반에 걸쳐 데이터를 쿼리하여 실시간 데이터를 복사하거나 전송하지 않고도 안전하게

액세스할 수 있습니다. 다음 단계에서는 Amazon Redshift 환경에서 데이터베이스 설정에 대한 세부 정보를 다룹니다.

Console

데이터 공유에서 데이터 쿼리를 시작하기 전에 데이터 공유에서 데이터베이스를 만들어야 합니다. 지정된 datashare에서 데이터베이스를 하나만 생성할 수 있습니다.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 클러스터(Clusters)를 선택한 다음 클러스터를 선택합니다. 클러스터 세부 정보 페이지가 나타납니다.
3. [Datashare(Datashares)]를 선택합니다. datashare 목록이 나타납니다.
4. [다른 클러스터의 datashare(Datashares from other clusters)] 섹션에서 [데이터베이스에 연결(Connect to database)]을 선택합니다. 자세한 내용은 데이터베이스로 연결 단원을 참조하십시오.
5. 데이터베이스를 생성하려는 datashare를 선택하고 [datashare에서 데이터베이스 생성(Create database from datashare)]을 선택합니다. datashare에서 데이터베이스 생성 페이지가 나타납니다.
6. [데이터베이스 이름(Database name)]에서 데이터베이스 이름을 지정합니다. 데이터베이스 이름은 1~64자의 영숫자(소문자만)여야 하며 예약어일 수 없습니다.
7. 생성(Create)을 선택합니다.

데이터베이스를 만든 후 소비자 관리자가 권한을 부여, 승인 및 연결한 경우 데이터베이스의 데이터를 쿼리하거나 쓰기 작업을 수행할 수 있습니다.

API

소비자 관리자로서 읽기 목적으로 데이터를 공유하려면 다음 단계를 수행합니다.

1. 사용할 수 있는 datashare를 나열하고 datashare의 내용을 봅니다. 자세한 내용은 DESC DATASHARE 및 SHOW DATASHARES 섹션을 참조하세요.

   다음 예에서는 지정된 생산자 네임스페이스의 인바운드 datashare 정보를 보여줍니다. 소비자 관리자로 DESC DATASHARE를 실행할 때 인바운드 데이터 공유를 보려면 NAMESPACE 옵션을 지정해야 합니다.

```
DESC DATASHARE salesshare OF NAMESPACE '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

```
 producer_account  |          producer_namespace          | share_type |
 share_name | object_type |           object_name           |   include_new
-------------------+-------------------------------------+------------
+------------+-------------+---------------------------------+------------------
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    |
 salesshare | table       | public.tickit_users_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    |
 salesshare | table       | public.tickit_venue_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    |
 salesshare | table       | public.tickit_category_redshift |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    |
 salesshare | table       | public.tickit_date_redshift     |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    |
 salesshare | table       | public.tickit_event_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    |
 salesshare | table       | public.tickit_listing_redshift  |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    |
 salesshare | table       | public.tickit_sales_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    |
 salesshare | schema      | public                          |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    |
 salesshare | view        | public.sales_data_summary_view  |
```

클러스터 슈퍼 사용자만 이 작업을 수행할 수 있습니다. SVV_DATASHARES를 사용하여 datashare를 보고 SVV_DATASHARE_OBJECTS를 사용하여 datashare 내의 객체를 볼 수도 있습니다.

다음 예에서는 소비자 클러스터의 인바운드 datashare를 보여줍니다.

```
SHOW DATASHARES LIKE 'sales%';


 share_name | share_owner | source_database | consumer_database | share_type
 | createdate | is_publicaccessible | share_acl | producer_account |
 producer_namespace
------------+-------------+-----------------+-------------------+------------
+------------+---------------------+-----------+------------------
+-------------------------------------
```

```
 salesshare |             |                 |                  | INBOUND
    |           |         t          |          |   123456789012   |
13b8833d-17c6-4f16-8fe4-1a018f5ed00d
```

2. Datashare 슈퍼 사용자는 datashare를 참조하는 로컬 데이터베이스를 생성할 수 있습니다. 자세한 내용은 데이터베이스 생성 단원을 참조하십시오.

```
CREATE DATABASE sales_db FROM DATASHARE salesshare OF NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

로컬 데이터베이스의 객체에 대한 액세스를 보다 세밀하게 제어하려면 데이터베이스를 만들 때 WITH PERMISSIONS 절을 사용하세요. 이렇게 하면 4단계에서 데이터베이스의 객체에 객체 수준 권한을 부여할 수 있습니다.

```
CREATE DATABASE sales_db WITH PERMISSIONS FROM DATASHARE salesshare OF NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

SVV_REDSHIFT_DATABASES 뷰를 쿼리하여 datashare에서 생성한 데이터베이스를 볼 수 있습니다. 그러나 소비자 클러스터의 로컬 데이터베이스에 연결하고 데이터베이스 간 쿼리를 수행하여 데이터 공유에서 만들어진 데이터베이스의 데이터를 쿼리할 수 있습니다.

> **Note**
>
> 기존 datashare에서 생성된 데이터베이스 객체 위에 datashare를 생성할 수 없습니다. 그러나 데이터를 소비자 클러스터의 별도 테이블에 복사하고 필요한 처리를 수행한 다음 생성된 새 객체를 공유할 수 있습니다.

Amazon Redshift 콘솔을 사용하여 datashare에서 데이터베이스를 생성할 수도 있습니다. 자세한 내용은 datashare에서 데이터베이스 생성 단원을 참조하십시오.

Amazon Redshift에서 소비자 사용자 및 역할에 객체 수준 권한 부여

소비자 관리자는 다음 단계를 완료하여 객체 수준에서 소비자 사용자 및 역할에 권한을 부여할 수 있습니다.

SQL

WITH PERMISSIONS를 사용하지 않고 데이터베이스를 생성한 경우 데이터 공유에서 생성된 전체 데이터베이스에 대한 권한만 사용자와 역할에 할당할 수 있습니다.

```
GRANT USAGE ON DATABASE sales_db TO Bob;
```

```
GRANT USAGE ON SCHEMA sales_schema TO ROLE Analyst_role;
```

또한 공유 객체 위에 후기 바인딩 뷰를 생성하고 이를 사용하여 세분화된 권한을 할당할 수 있습니다. 생산자 클러스터가 필요한 세분성으로 추가 datashare를 생성하도록 할 수도 있습니다.

WITH PERMISSIONS을 사용하여 데이터베이스를 만든 경우 공유 데이터베이스의 객체에 객체 수준 권한을 할당해야 합니다. USAGE 권한만 있는 사용자는 추가 객체 수준 권한을 부여받을 때까지 WITH PERMISSION을 사용하여 만든 데이터베이스의 객체에 액세스할 수 없습니다.

```
GRANT SELECT ON sales_db.public.tickit_sales_redshift to Bob;
```

다중 웨어하우스 쓰기 권한 부여에 대한 자세한 내용은 Amazon Redshift에서 데이터 공유에 대한 권한 관리 섹션을 참조하세요.

Amazon Redshift의 데이터 공유에서 데이터 쿼리

Amazon Redshift를 사용하면 생산자 클러스터에서 데이터 공유 전반에 걸쳐 데이터를 쿼리하여 실시간 데이터를 복사하거나 전송하지 않고도 안전하게 액세스할 수 있습니다. 다음 섹션에서는 Amazon Redshift 환경에서 데이터 공유 쿼리에 대한 세부 정보를 다룹니다.

소비자 데이터베이스 및 소비자 클러스터의 스키마에 대한 권한이 있는 사용자와 역할은 모든 공유 객체의 메타데이터를 탐색할 수 있습니다. 또한 소비자 클러스터에서 로컬 객체를 탐색할 수 있습니다. 이를 위해 JDBC 또는 ODBC 드라이버 또는 SVV_ALL 및 SVV_REDSHIFT 뷰를 사용합니다.

생산자 클러스터의 경우 각 스키마 내의 데이터베이스, 테이블 및 뷰에 많은 스키마가 있을 수 있습니다. 소비자 측의 사용자는 datashare를 통해 사용할 수 있는 객체의 하위 집합만 볼 수 있습니다. 이러한 사용자는 생산자 클러스터에서 전체 메타데이터를 볼 수 없습니다. 이 접근 방식은 datashare로 세분화된 메타데이터 보안 제어를 제공하는 데 도움이 됩니다.

로컬 클러스터 데이터베이스에 계속 연결합니다. 그러나 이제 세 부분으로 구성된 database.schema.table 표기법을 사용하여 datashare에서 생성된 데이터베이스와 스키마에서도 읽을

수도 있습니다. 사용자에게 표시되는 모든 데이터베이스에 걸쳐 쿼리를 수행할 수 있습니다. 이러한 데이터베이스는 클러스터의 로컬 데이터베이스이거나 datashare에서 생성된 데이터베이스일 수 있습니다. 소비자 클러스터는 datashare에서 생성된 데이터베이스에 연결할 수 없습니다.

전체 자격을 사용하여 데이터에 액세스할 수 있습니다. 자세한 내용은 데이터베이스 간 쿼리 예제 단원을 참조하십시오.

SQL

```
SELECT * FROM sales_db.public.tickit_sales_redshift ORDER BY 1,2 LIMIT 5;

 salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
 commission |      saletime
---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------
       1 |      1 |    36861 |   21191 |    7872 |   1875 |       4 |    728.00 |
  109.20 | 2008-02-18 02:36:48
       2 |      4 |     8117 |   11498 |    4337 |   1983 |       2 |     76.00 |
   11.40 | 2008-06-06 05:00:16
       3 |      5 |     1616 |   17433 |    8647 |   1983 |       2 |    350.00 |
   52.50 | 2008-06-06 08:26:17
       4 |      5 |     1616 |   19715 |    8647 |   1986 |       1 |    175.00 |
   26.25 | 2008-06-09 08:38:52
       5 |      6 |    47402 |   14115 |    8240 |   2069 |       2 |    154.00 |
   23.10 | 2008-08-31 09:17:02
```

공유 객체에는 SELECT 문만 사용할 수 있습니다. 그러나 다른 로컬 데이터베이스에 있는 공유 객체의 데이터를 쿼리하여 소비자 클러스터에 테이블을 생성할 수 있습니다.

Amazon Redshift에서 기존 데이터 공유에 대한 생산자 작업

Amazon Redshift를 사용하면 기존 데이터 공유를 관리하여 Amazon Redshift 클러스터의 데이터에 대한 액세스를 제어할 수 있습니다. 다음 섹션에서는 데이터 공유 객체 수정, 데이터 공유 권한 관리, 데이터 공유 속성 업데이트에 대한 단계별 가이드를 제공하여 Amazon Redshift 환경에서 데이터 액세스를 효과적으로 제어하고 감사할 수 있습니다.

주제

- Amazon Redshift에서 데이터 공유 보기
- Amazon Redshift에서 자신의 계정에서 만들어진 데이터 공유 편집
- Amazon Redshift의 데이터 공유에서 권한 부여 제거

- Amazon Redshift의 데이터 공유에서 데이터 공유 객체 제거
- Amazon Redshift의 데이터 공유에서 데이터 소비자 제거
- Amazon Redshift에서 계정에서 만들어진 데이터 공유 삭제

Amazon Redshift에서 데이터 공유 보기

콘솔 또는 SQL을 사용하여 데이터 공유를 볼 수 있습니다.

Console

데이터 공유 또는 클러스터 탭에서 데이터 공유를 볼 수 있습니다.

- 데이터 공유 탭을 사용하여 자신의 계정이나 다른 계정의 데이터 공유를 나열합니다.
  - 계정에서 생성된 datashare를 보려면 [내 계정에서(In my account)]를 선택하고 보려는 datashare를 선택합니다.
  - 다른 계정에서 공유되는 datashare를 보려면 [다른 계정에서(From other accounts)]를 선택하고 보려는 datashare를 선택합니다.
- 클러스터 탭을 사용하여 자신의 클러스터나 다른 클러스터의 데이터 공유를 나열합니다.

  먼저 데이터베이스에 연결합니다. 그런 다음 다른 클러스터의 데이터 공유 또는 내 클러스터에서 생성된 데이터 공유 섹션에서 데이터 공유를 선택하여 세부 정보를 봅니다.

SQL

클러스터에서 만들어진 데이터 공유를 나열하고 데이터 공유의 내용을 살펴볼 수 있습니다.

다음 예에서는 `salesshare`라는 datashare의 정보를 보여줍니다.

```
DESC DATASHARE salesshare;

 producer_account  |          producer_namespace          | share_type | share_name
 | object_type |           object_name          |   include_new
------------------+--------------------------------------+------------+------------
+-------------+--------------------------------+------------------
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_users_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_venue_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_category_redshift|
```

```
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_date_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_event_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_listing_redshift |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_sales_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | schema      | public                         |  t
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | view        | public.sales_data_summary_view |
```

다음 예에서는 생산자 클러스터의 아웃바운드 datashare를 보여줍니다.

```
SHOW DATASHARES LIKE 'sales%';
```

출력 결과는 다음과 비슷합니다.

```
share_name | share_owner  | source_database | consumer_database | share_type |
  createdate      | is_publicaccessible  | share_acl | producer_account |
 producer_namespace
-----------+--------------+-----------------+-------------------+------------
+---------------------+----------------------+-----------+------------------
+--------------------------------------
salesshare |    100       | dev             |                   |  OUTBOUND
 | 2020-12-09 02:27:08 |          True        |           |   123456789012   |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d
```

자세한 내용은 DESC DATASHARE 및 SHOW DATASHARES 섹션을 참조하세요.

SVV_DATASHARES, SVV_DATASHARE_CONSUMERS, 및 SVV_DATASHARE_OBJECTS를 사용하여 datashare, datashare 내의 객체 및 datashare 소비자를 볼 수도 있습니다.

Amazon Redshift에서 자신의 계정에서 만들어진 데이터 공유 편집

콘솔과 SQL을 사용하여 자신의 계정에서 만들어진 데이터 공유를 편집할 수 있습니다.

Console

콘솔에서 다음 단계를 따라 먼저 데이터베이스에 연결하여 계정에서 만들어진 데이터 공유 목록을 봅니다.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 클러스터(Clusters)를 선택한 다음 클러스터를 선택합니다. 클러스터 세부 정보 페이지가 나타납니다.
3. [Datashare(Datashares)]를 선택합니다.
4. [내 계정에 생성된 datashare(Datashares created in my account)] 섹션에서 [데이터베이스에 연결(Connect to database)]을 선택합니다.
5. 편집할 datashare를 선택한 다음 편집(Edit)을 선택합니다. datashare 세부 정보 페이지가 나타납니다.
6. [datashare 객체(Datashare objects)] 또는 [데이터 소비자(Data consumers)] 섹션에서 변경합니다.
7. Save changes(변경 사항 저장)를 선택합니다. 변경 사항으로 datashare가 업데이트됩니다.

> **Note**
>
> 데이터 공유를 AWS Glue Data Catalog에 게시하도록 선택한 경우 데이터 공유를 다른 Amazon Redshift 계정에 게시하도록 구성을 편집할 수 없습니다.

SQL

ALTER DATASHARE를 사용하여 datashare에서 언제든지 객체를 제거합니다. 스키마를 제거하려면 다음 명령을 사용합니다.

```
ALTER DATASHARE salesshare REMOVE SCHEMA PUBLIC;
```

테이블을 제거하려면 다음 명령을 사용합니다.

```
ALTER DATASHARE salesshare REMOVE TABLE public.tickit_sales_redshift;
```

REVOKE USAGE ON을 사용하여 특정 소비자에 대한 datashare 권한을 취소합니다. datashare 내의 객체에 대한 USAGE 권한을 취소하고 모든 소비자 클러스터에 대한 액세스를 즉시 중지합니다. 데이터베이스 및 테이블 나열 등의 datashare 나열과 메타데이터 쿼리는 액세스가 취소된 후 공유 객체를 반환하지 않습니다. 더 이상 소비자와 데이터를 공유하지 않으려면 네임스페이스에서 datashare에 대한 액세스를 취소합니다.

```
REVOKE USAGE ON DATASHARE salesshare FROM NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

더 이상 소비자와 데이터를 공유하지 않으려면 AWS 계정에서 데이터 공유에 대한 액세스를 취소합니다.

```
REVOKE USAGE ON DATASHARE salesshare FROM ACCOUNT '123456789012';
```

Amazon Redshift의 데이터 공유에서 권한 부여 제거

Amazon Redshift를 사용하면 지정된 소비자에 대한 권한을 취소하여 데이터 공유에 대한 액세스를 제어할 수 있습니다. 이 섹션에서는 Amazon Redshift에서 데이터 공유에 대한 소비자 액세스 권한을 취소하기 위한 지침을 제공합니다.

> **Note**
>
> 데이터 공유에 대한 권한을 제거하려면 데이터 공유에 데이터 소비자가 하나 이상 추가되어 있어야 합니다.

Console

승인을 제거하려는 하나 이상의 소비자 클러스터를 선택합니다. 그런 다음 권한 제거를 선택합니다.

권한이 제거되면 데이터 소비자는 datashare에 대한 액세스 권한을 즉시 상실합니다.

API

생산자 보안 관리자는 다음을 결정합니다.

- 다른 계정이 데이터 공유에 액세스할 수 있는지 여부
- 계정에 데이터 공유에 대한 액세스 권한이 있는 경우, 해당 계정에 쓰기 권한이 있는지 여부

데이터 공유 권한을 제거하려면 다음과 같은 IAM 권한이 필요합니다.

redshift:DeauthorizeDataShare

CLI 직접 호출 또는 API를 사용하여 사용 및 쓰기 권한을 제거할 수 있습니다.

```
deauthorize-data-share
--data-share-arn <value>
--consumer-identifier <value>
```

명령에 대한 자세한 내용은 deauthorize-data-share를 참조하세요.

Amazon Redshift의 데이터 공유에서 데이터 공유 객체 제거

다음 절차를 사용하여 datashare에서 객체를 하나 이상 제거할 수 있습니다.

Console

콘솔을 사용하여 데이터 공유에서 하나 이상의 객체를 제거하려면 다음 단계를 따르세요.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 클러스터(Clusters)를 선택한 다음 클러스터를 선택합니다. 클러스터 세부 정보 페이지가 나타납니다.
3. [Datashare(Datashares)]를 선택합니다.
4. [내 계정에 생성된 datashare(Datashares created in my account)] 섹션에서 [데이터베이스에 연결(Connect to database)]을 선택합니다. 자세한 내용은 데이터베이스로 연결 단원을 참조하십시오.
5. 편집할 datashare를 선택한 다음 편집(Edit)을 선택합니다. datashare 세부 정보 페이지가 나타납니다.
6. datashare 객체를 하나 이상 제거하려면 다음 중 하나를 수행합니다.
   - datashare에서 스키마를 제거하려면 스키마를 하나 이상 선택합니다. 그런 다음 [제거(Remove)]를 선택합니다. Amazon Redshift는 datashare에서 지정된 스키마와 해당 스키마의 모든 객체를 제거합니다.
   - datashare에서 테이블과 뷰를 제거하려면 테이블과 뷰를 하나 이상 선택합니다. 그런 다음 [제거(Remove)]를 선택합니다. 또는 [스키마별 제거(Remove by schema)]를 선택하여 지정된 스키마의 모든 테이블과 뷰를 제거합니다.
   - datashare에서 사용자 정의 함수를 제거하려면 사용자 정의 함수를 하나 이상 선택합니다. 그런 다음 [제거(Remove)]를 선택합니다. 또는 [스키마별 제거(Remove by schema)]를 선택하여 지정된 스키마의 모든 사용자 정의 함수를 제거합니다.

SQL

ALTER DATASHARE를 사용하여 datashare에서 언제든지 객체를 제거합니다. 스키마를 제거하려면 다음 명령을 사용합니다.

```
ALTER DATASHARE salesshare REMOVE SCHEMA PUBLIC;
```

테이블을 제거하려면 다음 명령을 사용합니다.

```
ALTER DATASHARE salesshare REMOVE TABLE public.tickit_sales_redshift;
```

Amazon Redshift의 데이터 공유에서 데이터 소비자 제거

datashare에서 데이터 소비자를 하나 이상 제거할 수 있습니다. 데이터 소비자는 Amazon Redshift 클러스터 또는 AWS 계정을 고유하게 식별한 네임스페이스일 수 있습니다.

Console

콘솔을 사용하여 데이터 공유에서 하나 이상의 데이터 소비자를 제거하려면 네임스페이스 ID 또는 AWS 계정에서 하나 이상의 데이터 소비자를 선택합니다. 그런 다음 제거를 선택합니다.

Amazon Redshift는 datashare에서 지정된 데이터 소비자를 제거합니다. 제거된 소비자는 datashare에 대한 액세스 권한을 즉시 상실합니다.

SQL

REVOKE USAGE ON을 사용하여 특정 소비자에 대한 datashare 권한을 취소합니다. datashare 내의 객체에 대한 USAGE 권한을 취소하고 모든 소비자 클러스터에 대한 액세스를 즉시 중지합니다. 데이터베이스 및 테이블 나열 등의 datashare 나열과 메타데이터 쿼리는 액세스가 취소된 후 공유 객체를 반환하지 않습니다. 더 이상 소비자와 데이터를 공유하지 않으려면 네임스페이스에서 datashare에 대한 액세스를 취소합니다.

```
REVOKE USAGE ON DATASHARE salesshare FROM NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

Amazon Redshift에서 계정에서 만들어진 데이터 공유 삭제

콘솔 또는 SQL을 사용하여 자신의 계정에서 만들어진 데이터 공유를 삭제할 수 있습니다.

Console

콘솔을 사용하여 계정에서 만들어진 데이터 공유를 삭제하려면 먼저 데이터베이스에 연결하여 계정에서 만들어진 데이터 공유 목록을 확인합니다.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 클러스터(Clusters)를 선택한 다음 클러스터를 선택합니다. 클러스터 세부 정보 페이지가 나타납니다.
3. [Datashare(Datashares)]를 선택합니다. datashare 목록이 나타납니다.
4. [내 계정에 생성된 datashare(Datashares created in my account)] 섹션에서 [데이터베이스에 연결(Connect to database)]을 선택합니다.
5. 삭제하려는 datashare를 하나 이상 선택하고 [삭제(Delete)]를 선택합니다. datashare 삭제 페이지가 나타납니다.

   Lake Formation과 공유된 데이터 공유를 삭제해도 Lake Formation에서 연결된 권한이 자동으로 제거되지 않습니다. 이를 제거하려면 Lake Formation 콘솔로 이동하세요.
6. [삭제(Delete)]를 입력하여 지정된 datashare 삭제를 확인합니다.
7. Delete(삭제)를 선택합니다.

datashare가 삭제되면 데이터 소비자는 datashare에 대한 액세스 권한을 상실합니다.

SQL

DROP DATASHARE를 사용하여 언제든지 SQL을 사용해 데이터 공유 객체를 삭제할 수 있습니다. 클러스터 슈퍼 사용자와 datashare 소유자는 datashare를 삭제할 수 있습니다.

다음 예에서는 `salesshare`라는 datashare를 삭제합니다.

```
DROP DATASHARE salesshare;
```

Amazon Redshift에서 기존 데이터 공유에 대한 소비자 작업

Amazon Redshift를 사용하면 기존 데이터 공유를 관리하여 Amazon Redshift 클러스터의 데이터에 대한 액세스를 제어할 수 있습니다. 다음 섹션에서는 Amazon Redshift 환경에서 소비자 관리자로서 데이터 공유를 관리하는 방법에 대한 단계별 가이드를 제공합니다.

주제

- Amazon Redshift에서 데이터 공유에 대한 권한 관리
- Amazon Redshift의 다른 AWS 계정에 있는 데이터 소비자에게서 데이터 공유 연결 제거
- Amazon Redshift에서 다른 AWS 계정의 데이터 공유 거부

Amazon Redshift에서 데이터 공유에 대한 권한 관리

생산자 관리자는 공유 중인 데이터세트에 대한 제어 권한을 보유합니다. 따라서 datashare에서 새 객체를 추가하거나 제거할 수 있습니다. 소비자 클러스터 및 AWS 계정 또는 AWS 리전에 대해 전체적으로 datashare에 대한 액세스 권한을 부여하거나 취소할 수도 있습니다. 권한이 취소되면 소비자 클러스터는 즉시 공유 객체에 대한 액세스 권한을 상실하고 SVV_DATASHARES의 INBOUND datashare 목록에서 해당 객체를 볼 수 없습니다.

다음 예제에서는 `salesshare` 데이터 공유를 만들고, `public` 스키마를 추가하고, `public.tickit_sales_redshift` 테이블을 `salesshare`에 추가합니다. 또한 특정 네임스페이스에 `salesshare`에 대한 사용 권한을 부여합니다.

```
CREATE DATASHARE salesshare;

ALTER DATASHARE salesshare ADD SCHEMA public;

ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;

GRANT USAGE ON DATASHARE salesshare TO NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

CREATE DATASHARE의 경우 슈퍼 사용자와 데이터베이스 소유자는 datashare를 생성할 수 있습니다. 자세한 내용은 CREATE DATASHARE 단원을 참조하십시오. ALTER DATASHARE의 경우 추가하거나 제거할 datashare 객체에 대한 필수 권한이 있는 datashare 소유자는 datashare를 변경할 수 있습니다. 자세한 내용은 ALTER DATASHARE을 참조하세요.

생산자 관리자가 삭제한 datashare는 소비자 클러스터에 더 이상 나열되지 않습니다. 삭제된 datashare에서 소비자 클러스터에 생성된 데이터베이스 및 스키마 참조는 객체 없이 계속 존재합니다. 소비자 관리자는 수동으로 이러한 데이터베이스를 삭제해야 합니다.

소비자 측에서 소비자 관리자는 데이터 공유에서 데이터베이스를 만들어 공유 데이터에 액세스해야 하는 사용자와 역할을 결정할 수 있습니다. 데이터베이스를 생성할 때 선택한 옵션에 따라 다음과 같이 데이터베이스 액세스를 제어할 수 있습니다. 데이터 공유에서 데이터베이스 생성에 대한 자세한 내용은 데이터베이스 생성 섹션을 참조하세요.

데이터 공유 설정 및 소비자의 데이터 읽기에 대한 자세한 내용은 AWS 계정 내 데이터에 대한 읽기 액세스 공유를 참조하세요.

WITH PERMISSIONS 절을 사용하지 않고 데이터베이스 생성

관리자는 데이터베이스 또는 스키마 수준에서 액세스를 제어할 수 있습니다. 스키마 수준에서 액세스를 제어하려면 관리자는 데이터 공유에서 생성된 Amazon Redshift 데이터베이스에서 외부 스키마를 생성해야 합니다.

다음 예에서는 데이터베이스 수준 및 스키마 수준에서 공유 테이블에 액세스할 수 있는 권한을 부여합니다.

```
GRANT USAGE ON DATABASE sales_db TO Bob;

CREATE EXTERNAL SCHEMA sales_schema FROM REDSHIFT DATABASE sales_db SCHEMA 'public';

GRANT USAGE ON SCHEMA sales_schema TO ROLE Analyst_role;
```

액세스를 추가로 제한하려면 공유 객체 위에 뷰를 생성하여 필요한 데이터만 표시할 수 있습니다. 그런 다음 이러한 뷰를 사용하여 사용자와 역할에 대한 액세스 권한을 부여할 수 있습니다.

사용자에게 데이터베이스 또는 스키마에 대한 액세스 권한이 부여된 후 해당 데이터베이스 또는 스키마의 모든 공유 객체에 액세스할 수 있습니다.

WITH PERMISSIONS 절을 사용하여 데이터베이스 생성

데이터베이스 또는 스키마에 대한 사용 권한을 부여한 후 관리자는 로컬 데이터베이스 또는 스키마에서와 동일한 권한 부여 프로세스를 사용하여 액세스를 추가로 제어할 수 있습니다. 개별 객체 권한이 없으면 사용자는 USAGE 권한을 부여받은 후에도 데이터 공유 데이터베이스 또는 스키마의 객체에 액세스할 수 없습니다.

다음 예에서는 데이터베이스 수준에서 공유 테이블에 액세스할 수 있는 권한을 부여합니다.

```
GRANT USAGE ON DATABASE sales_db TO Bob;
GRANT USAGE FOR SCHEMAS IN DATABASE sales_db TO Bob;
GRANT SELECT ON sales_db.public.tickit_sales_redshift TO Bob;
```

데이터베이스 또는 스키마에 대한 액세스 권한을 부여받은 후에도 사용자는 액세스하려는 데이터베이스 또는 스키마의 모든 객체에 대한 관련 권한을 부여받아야 합니다.

WITH PERMISSIONS를 사용한 세분화된 공유

WITH PERMISSIONS을 사용하여 세분화된 공유를 사용해 클러스터 또는 Serverless 작업 그룹이 데이터 공유를 쿼리하도록 할 수 있습니다. 이 프로세스에서는 데이터 공유가 사용자 계정의 다른 클러스터 또는 Amazon Redshift Serverless 네임스페이스에서 시작되거나 다른 계정에서 가져왔고 사용 중인 네임스페이스와 연결되어 있다고 가정합니다.

1. 소비자 데이터베이스 관리자는 데이터 공유에서 데이터베이스를 생성할 수 있습니다.

   ```
   CREATE DATABASE my_ds_db [WITH PERMISSIONS] FROM DATASHARE my_datashare OF
    NAMESPACE 'abc123def';
   ```

   WITH PERMISSIONS을 사용하여 데이터베이스를 만드는 경우 데이터 공유 객체에 대한 세분화된 권한을 다양한 사용자 및 역할에 부여할 수 있습니다. 이렇게 하지 않으면 데이터 공유 데이터베이스에 대한 USAGE 권한이 부여된 모든 사용자 및 역할에 데이터 공유 데이터베이스 내의 모든 객체에 대한 모든 권한이 부여됩니다.
2. 다음은 Redshift 데이터베이스 사용자 또는 역할에 권한을 부여하는 방법을 보여줍니다. 이 문을 실행하려면 로컬 데이터베이스에 연결되어 있어야 합니다. 권한 부여 문을 실행하기 전에 데이터 공유 데이터베이스에서 USE 명령을 실행하면 이러한 문을 실행할 수 없습니다.

   ```
   GRANT USAGE ON DATABASE my_ds_db TO ROLE data_eng;
   GRANT CREATE, USAGE ON SCHEMA my_ds_db.my_shared_schema TO ROLE data_eng;
   GRANT ALL ON ALL TABLES IN SCHEMA my_ds_db.my_shared_schema TO ROLE data_eng;

   GRANT USAGE ON DATABASE my_ds_db TO bi_user;
   GRANT USAGE ON SCHEMA my_ds_db.my_shared_schema TO bi_user;
   GRANT SELECT ON my_ds_db.my_shared_schema.table1 TO bi_user;
   ```

Amazon Redshift의 다른 AWS 계정에 있는 데이터 소비자에게서 데이터 공유 연결 제거

Amazon Redshift를 사용하면 다른 AWS 계정에서 공유하는 데이터 공유에서 연결을 제거할 수 있습니다. 데이터 공유는 하나 이상의 Redshift 데이터베이스에서 데이터를 캡슐화하는 공유 가능한 데이터베이스 객체입니다. 다음 섹션에서는 Redshift 환경 내에서 데이터 공유의 연결을 해제하는 프로세스를 보여줍니다.

Console

콘솔에서 소비자 관리자는 데이터 소비자에게서 데이터 공유 연결을 제거할 수 있습니다.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.

2. 탐색 메뉴에서 Datashares를 선택합니다. datashare 목록 페이지가 나타납니다.
3. [다른 계정에서(From other accounts)]를 선택합니다.
4. [다른 계정의 datashare(Datashares from other accounts)] 섹션에서 데이터 소비자에게서 연결을 제거할 datashare를 선택합니다.
5. [데이터 소비자(Data consumers)] 섹션에서 연결을 제거할 데이터 소비자를 하나 이상 선택합니다. 그런 다음 [연결 제거(Remove association)]를 선택합니다.
6. 연결 제거 페이지가 나타나면 [연결 제거(Remove association)]를 선택합니다.

연결이 제거되면 데이터 소비자는 datashare에 대한 액세스 권한을 상실합니다. 언제든지 데이터 소비자 연결을 변경할 수 있습니다.

SQL

> **Note**
>
> 이 섹션의 단계는 생산자 관리자가 공유 데이터베이스 객체에 대한 특정 작업을 허용하고, 데이터 공유를 다른 계정과 공유하는 경우 생산자 보안 관리자가 액세스를 권한을 부여한 후에 수행됩니다.

소비자 보안 관리자는 데이터 공유를 다음 명령과 연결 해제할 수 있습니다.

```
disassociate-data-share-consumer
--data-share-arn <value>
```

명령에 대한 자세한 내용은 disassociate-data-share-consumer를 참조하세요.

Amazon Redshift에서 다른 AWS 계정의 데이터 공유 거부

Amazon Redshift를 사용하면 다른 AWS 계정에서 공유한 데이터 공유를 거부할 수 있으므로 조직 경계 전반에서 원활하고 안전한 데이터 공유가 가능합니다. 데이터 공유는 하나 이상의 Amazon Redshift 데이터베이스에서 데이터를 캡슐화하는 공유 가능한 데이터베이스 객체입니다.

소비자 관리자는 사용 가능 또는 활성 상태인 데이터 공유를 거부할 수 있습니다. 데이터 공유를 거부하면 소비자 클러스터 사용자는 데이터 공유에 대한 액세스 권한을 상실합니다. Amazon Redshift는 `DescribeDataSharesForConsumer` API 작업을 호출하는 경우 거부된 데이터 공유를 반환하지 않습니다. 생산자 관리자가 `DescribeDataSharesForProducer` API 작업을 실행하면 데이터 공유가 거부된 것을 확인할 수 있습니다. 데이터 공유가 거부되면 생산자 관리자는 소비자 클러스터에 대한 데

이터 공유를 다시 승인할 수 있으며, 소비자 관리자는 자신의 AWS 계정을 데이터 공유와 연결하거나 이를 거부하도록 선택할 수 있습니다.

AWS 계정이 데이터 공유와 연결되어 있고 Lake Formation에서 관리하는 데이터 공유에 대한 보류 중인 연결이 있는 경우. Lake Formation에서 관리하는 데이터 공유 연결을 거부하면 원래 데이터 공유도 거부됩니다. 특정 연결을 거부하려면 생산자 관리자가 지정된 데이터 공유에서 권한 부여를 제거할 수 있습니다. 이 작업은 다른 데이터 공유에 영향을 주지 않습니다.

데이터 공유를 거부하려면 AWS 콘솔, API 작업 `RejectDataShare` 또는 AWS CLI의 `reject-datashare`를 사용하세요.

Console

AWS Console을 사용하여 데이터 공유를 거부하려면 다음 단계를 수행합니다.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 데이터 공유를 선택합니다.
3. [다른 계정에서(From other accounts)]를 선택합니다.
4. [다른 계정의 datashare(Datashares from other accounts)] 섹션에서 거부할 datashare를 선택합니다. datashare 거부(Decline datashare) 페이지가 나타나면 거부(Decline)를 선택합니다.

datashare를 거부한 후에는 변경 사항을 되돌릴 수 없습니다. Amazon Redshift가 목록에서 데이터 공유를 제거합니다. 데이터 공유를 다시 보려면 생산자 관리자가 데이터 공유를 다시 승인해야 합니다.

CLI

데이터 공유를 거부하려면 관리자가 다음 명령을 사용합니다.

```
reject-data-share
--data-share-arn <value>
```

명령에 대한 자세한 내용은 reject-data-share를 참조하세요.

Amazon Redshift에서 AWS CloudFormation과 데이터 공유 시작하기

AWS 리소스를 프로비저닝하는 ㅂAWS CloudFormation스택을 사용하여 데이터 공유 설정을 자동화할 수 있습니다. 다음에 설명하는 CloudFormation 스택은 동일한 AWS 계정에 있는 두 Amazon

Redshift 클러스터 간의 데이터 공유를 설정합니다. 따라서 리소스 프로비저닝을 위해 SQL 문을 실행하지 않고도 데이터 공유를 시작할 수 있습니다.

스택은 사용자가 지정하는 클러스터에 datashare를 생성합니다. datashare에는 테이블과 샘플 읽기 전용 데이터가 포함됩니다. 이 데이터는 다른 Amazon Redshift 클러스터에서 읽을 수 있습니다.

CloudFormation을 사용하지 않고 SQL 문을 실행하여 datashare를 설정하고 권한을 부여하여 AWS 계정에서 데이터 공유를 시작하려는 경우 AWS 계정 내 데이터에 대한 읽기 액세스 공유 섹션을 참조하세요.

데이터 공유 CloudFormation 스택을 실행하기 전에 IAM 역할과 Lambda 함수를 생성할 권한이 있는 사용자로 로그인해야 합니다. 또한 동일한 계정에 2개의 Amazon Redshift 클러스터가 필요합니다. 하나는 샘플 데이터를 공유하는 데 사용하는 생산자이고, 다른 하나는 샘플 데이터를 읽는 데 사용하는 소비자입니다. 이러한 클러스터의 기본 요구 사항은 각각 RA3 노드를 사용한다는 것입니다. 추가적인 필수 사항은 Amazon Redshift의 데이터 공유 고려 사항 섹션을 참조하세요.

Amazon Redshift 클러스터 설정 시작하기에 대한 자세한 내용은 Amazon Redshift 프로비저닝된 데이터 웨어하우스 시작하기를 참조하세요. CloudFormation으로 설정 자동화에 대한 자세한 내용은 AWS CloudFormation이란 무엇인가요?를 참조하세요.

> **Important**
>
> CloudFormation 스택을 시작하기 전에 동일한 계정에 두 개의 Amazon Redshift 클러스터가 있고 클러스터가 RA3 노드를 사용하는지 확인합니다. 각 클러스터에 데이터베이스와 슈퍼유저가 있는지 확인합니다. 자세한 내용은 데이터베이스 생성 및 superuser 단원을 참조하세요.

Amazon Redshift 데이터 공유를 위한 CloudFormation 스택을 시작하려면

1. CFN 스택 시작(Launch CFN stack)을 클릭하여 AWS Management Console에서 CloudFormation 서비스로 이동합니다.

   메시지가 나타나면 로그인합니다.

   Amazon S3에 저장된 CloudFormation 템플릿 파일을 참조하여 스택 생성 프로세스가 시작됩니다. CloudFormation 템플릿은 스택을 구성하는 AWS 리소스를 선언하는 JSON 형식의 텍스트 파일입니다. CloudFormation 템플릿에 대한 자세한 내용은 템플릿 기본 사항 알아보기를 참조하세요.
2. 다음(Next)을 선택하여 스택 세부 정보를 입력합니다.
3. 파라미터(Parameters)에서 각 클러스터에 대해 다음을 입력합니다.

- Amazon Redshift 클러스터 이름(예: **ra3-consumer-cluster**)
- 데이터베이스 이름(예: **dev**)
- 데이터베이스 사용자의 이름(예: **consumeruser**)

스택은 여러 데이터베이스 객체를 생성하므로 테스트 클러스터를 사용하는 것이 좋습니다.

Next(다음)를 선택합니다.

4. 스택 옵션이 나타납니다.

   다음(Next)을 선택하여 기본 설정을 적용합니다.

5. 기능(Capabilities)에서 AWS CloudFormation 에서 IAM 리소스를 생성할 수 있음을 승인합니다(I acknowledge that AWS CloudFormation might create IAM resources)를 선택합니다.
6. 스택 생성을 선택합니다.

CloudFormation에서 템플릿을 사용하여 Amazon Redshift 스택을 구축하고 `myproducer_share`라는 datashare를 생성하는 데 10분 정도 걸립니다. 스택은 스택 세부 정보에 지정된 데이터베이스에 datashare를 생성합니다. 해당 데이터베이스의 객체만 공유할 수 있습니다.

스택이 생성되는 동안 오류가 발생하면 다음을 수행합니다.

- 각 Redshift 클러스터에 대해 올바른 클러스터 이름, 데이터베이스 이름 및 데이터베이스 사용자 이름을 입력했는지 확인합니다.
- 클러스터에 RA3 노드가 있는지 확인합니다.
- IAM 역할 및 Lambda 함수를 만들 수 있는 권한이 있는 사용자로 로그인했는지 확인하세요. IAM 역할 생성에 대한 자세한 내용은 IAM 역할 생성을 참조하세요. Λ 함수 생성 정책에 대한 자세한 내용은 함수 개발을 참조하세요.

생성한 datashare 쿼리

다음 절차를 사용하려면 설명된 각 클러스터에서 쿼리를 실행하는 데 필요한 권한이 있는지 확인해야 합니다.

datashare를 쿼리하려면

1. Amazon Redshift 쿼리 에디터 v2와 같은 클라이언트 도구를 사용하여 CloudFormation 스택이 생성될 때 입력한 데이터베이스의 생산자 클러스터에 연결합니다.

2. datashare를 쿼리합니다.

```
SHOW DATASHARES;

+-----------------+-------------+-----------------+------------------
+-----------+-----------+--------------------+----------+-----------------
+-------------------------------------+
|    share_name    | share_owner | source_database | consumer_database | share_type
 | createdate | is_publicaccessible | share_acl | producer_account |
 producer_namespace          |
+-----------------+-------------+-----------------+------------------
+-----------+-----------+--------------------+----------+-----------------
+-------------------------------------+
| myproducer_share | 100         | sample_data_dev | myconsumer_db     | INBOUND
  | NULL       | true               | NULL      |  producer-acct   |        your-
producer-namespace       |
+-----------------+-------------+-----------------+------------------
+-----------+-----------+--------------------+----------+-----------------
+-------------------------------------+
```

앞의 명령은 스택에 의해 생성된 datashare의 이름인 myproducer_share를 반환합니다. 또한 datashare와 연결된 데이터베이스의 이름인 myconsumer_db를 반환합니다.

이후 단계에서 사용할 생산자 네임스페이스 식별자를 복사합니다.

3. datashare의 객체를 설명합니다.

```
DESC DATASHARE myproducer_share;

+-----------------+-------------------------------------+-----------
+-----------------+-------------+-------------------------------------
+-------------+
| producer_account |          producer_namespace          | share_type |
 share_name    | object_type |             object_name             | include_new |
+-----------------+-------------------------------------+-----------
+-----------------+-------------+-------------------------------------
+-------------+
|   producer-acct  |        your-producer-namespace       | OUTBOUND   |
 myproducer_share | schema      | myproducer_schema                   | true
 |
|   producer-acct  |        your-producer-namespace       | OUTBOUND   |
 myproducer_share | table       | myproducer_schema.tickit_sales      | NULL
 |
```

```
|   producer-acct   |        your-producer-namespace        | OUTBOUND    |
 myproducer_share | view        | myproducer_schema.ticket_sales_view | NULL
 |
+------------------+---------------------------------------+-----------
+------------------+-------------+-------------------------------------
+-------------+
```

datashare를 설명하면 테이블 및 보기에 대한 속성이 반환됩니다. 스택은 샘플 데이터가 있는 테이블과 보기를 생산자 데이터베이스에 추가합니다(예: `tickit_sales` 및 `tickit_sales_view`). TICKIT 샘플 데이터베이스에 대한 자세한 내용은 샘플 데이터베이스를 참조하세요.

쿼리를 실행하기 위해 datashare에 대한 권한을 위임할 필요가 없습니다. 스택은 필요한 권한을 부여합니다.

4. 클라이언트 도구를 사용하여 소비자 클러스터에 연결합니다. 생산자의 네임스페이스를 지정하여 datashare를 설명합니다.

```
DESC DATASHARE myproducer_share OF NAMESPACE '<namespace id>'; --specify the unique
 identifier for the producer namespace

+------------------+---------------------------------------+-----------
+------------------+-------------+-------------------------------------
+-------------+
| producer_account |          producer_namespace           | share_type |
 share_name    | object_type |             object_name             | include_new |
+------------------+---------------------------------------+-----------
+------------------+-------------+-------------------------------------
+-------------+
|   producer-acct   |        your-producer-namespace        | INBOUND     |
 myproducer_share | schema      | myproducer_schema                    | NULL
 |
|   producer-acct   |        your-producer-namespace        | INBOUND     |
 myproducer_share | table       | myproducer_schema.tickit_sales       | NULL
 |
|   producer-acct   |        your-producer-namespace        | INBOUND     |
 myproducer_share | view        | myproducer_schema.ticket_sales_view | NULL
 |
+------------------+---------------------------------------+-----------
+------------------+-------------+-------------------------------------
+-------------+
```

5. datashare의 데이터베이스와 스키마를 지정하여 datashare의 테이블을 쿼리할 수 있습니다. 자세한 내용은 데이터베이스 간 쿼리 예제 단원을 참조하십시오. 다음 쿼리는 TICKIT 샘플 데이터베이스의 SALES 테이블에서 판매 및 판매자 데이터를 반환합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

```
SELECT * FROM myconsumer_db.myproducer_schema.tickit_sales_view;

+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
| salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
 commission |      saletime       |
+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
|       1 |      1 |    36861 |   21191 |    7872 |   1875 |       4 |       728 |
     109.2 | 2008-02-18 02:36:48 |
|       2 |      4 |     8117 |   11498 |    4337 |   1983 |       2 |        76 |
      11.4 | 2008-06-06 05:00:16 |
|       3 |      5 |     1616 |   17433 |    8647 |   1983 |       2 |       350 |
      52.5 | 2008-06-06 08:26:17 |
|       4 |      5 |     1616 |   19715 |    8647 |   1986 |       1 |       175 |
     26.25 | 2008-06-09 08:38:52 |
|       5 |      6 |    47402 |   14115 |    8240 |   2069 |       2 |       154 |
      23.1 | 2008-08-31 09:17:02 |
+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
```

> **Note**
>
> 쿼리는 공유 스키마의 보기에 대해 실행됩니다. datashare에서 생성된 데이터베이스에는 직접 연결할 수 없습니다. 읽기 전용입니다.

6. 집계를 포함하는 쿼리를 실행하려면 다음 예제를 사용합니다.

```
SELECT * FROM myconsumer_db.myproducer_schema.tickit_sales ORDER BY 1,2 LIMIT 5;

+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
| salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
 commission |      saletime       |
+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
```

```
|       1 |      1 |    36861 |   21191 |    7872 |   1875 |       4 |       728 |
     109.2 | 2008-02-18 02:36:48 |
|       2 |      4 |     8117 |   11498 |    4337 |   1983 |       2 |        76 |
      11.4 | 2008-06-06 05:00:16 |
|       3 |      5 |     1616 |   17433 |    8647 |   1983 |       2 |       350 |
      52.5 | 2008-06-06 08:26:17 |
|       4 |      5 |     1616 |   19715 |    8647 |   1986 |       1 |       175 |
     26.25 | 2008-06-09 08:38:52 |
|       5 |      6 |    47402 |   14115 |    8240 |   2069 |       2 |       154 |
      23.1 | 2008-08-31 09:17:02 |
+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
```

쿼리는 샘플 TICKIT 데이터에서 매출 및 판매자 데이터를 반환합니다.

datashare 쿼리의 추가 예는 AWS 계정 내 데이터에 대한 읽기 액세스 공유 섹션을 참조하세요.

Amazon Redshift의 데이터 공유 유형

datashare는 Amazon Redshift에서 데이터를 공유하는 단위입니다. 데이터 공유를 사용해 동일한 AWS 계정 또는 다른 AWS 계정에서 데이터를 공유합니다. 또한 여러 Amazon Redshift 클러스터에서 읽기용으로 데이터를 공유합니다.

각 datashare는 Amazon Redshift 클러스터의 특정 데이터베이스와 연결됩니다.

생산자 관리자는 데이터 공유를 만들고 데이터 공유 객체를 추가하여 다른 클러스터와 데이터를 공유할 수 있으며, 이를 아웃바운드 공유라고 합니다. 소비자 관리자는 인바운드 공유라고 하는 다른 클러스터로부터 데이터 공유를 받을 수 있습니다. 생산자와 소비자에 대한 자세한 내용은 데이터 공유 생산자와 소비자를 참조하세요.

데이터 공유 객체는 생산자 관리자가 데이터 소비자와 공유할 데이터 공유에 추가할 수 있는 클러스터에 있는 특정 데이터베이스의 객체입니다. datashare 객체는 데이터 소비자에게 읽기 전용입니다. datashare 객체의 예로 테이블, 뷰 및 사용자 정의 함수가 있습니다. datashare를 생성하거나 편집하는 동안 언제든지 datashare에 datashare 객체를 추가할 수 있습니다.

클러스터의 크기를 조정하거나 생산자 클러스터가 일시 중지된 경우에도 데이터 공유는 계속 작동합니다.

데이터 공유에는 표준 데이터 공유, AWS Data Exchange 데이터 공유, AWS Lake Formation 관리형 데이터 공유 등 다양한 유형이 있습니다. 다음 페이지에서는 이러한 각 유형에 대한 개요를 제공합니다.

표준 데이터공유

표준 데이터 공유를 사용하면 프로비저닝된 클러스터, 서버리스 작업 그룹, 가용 영역, AWS 계정 및 AWS 리전 간에 데이터를 공유할 수 있습니다. 클러스터 형식 간은 물론 프로비저닝된 클러스터와 Amazon Redshift 서버리스 간에도 공유할 수 있습니다.

데이터를 공유하려면 다음과 같은 프로비저닝된 클러스터, 서버리스 네임스페이스 및 AWS 계정 계정 식별자에 유의하세요

- 프로비저닝된 네임스페이스는 Amazon Redshift 프로비저닝된 클러스터를 식별하는 식별자입니다. 프로비저닝된 클러스터 생성 중 네임스페이스 전역 고유 식별자(GUID)가 자동으로 생성되어 클러스터에 연결됩니다. 네임스페이스 Amazon 리소스 이름(ARN)은 arn: {partition}:redshift: {region}: {account-id}: namespace:{namespace-guid} 형식으로 되어 있습니다. Amazon Redshift 콘솔의 클러스터 세부 정보 페이지에서 프로비저닝된 클러스터의 네임스페이스를 볼 수 있습니다.

  데이터 공유 워크플로에서 네임스페이스 GUID 값과 네임스페이스 ARN은 AWS 계정의 클러스터와 데이터를 공유하는 데 사용됩니다. `current_namespace` 함수를 사용하여 현재 클러스터의 네임스페이스를 찾을 수도 있습니다.
- 서버리스 네임스페이스는 Amazon Redshift Serverless를 식별하는 식별자입니다. Amazon Redshift Serverless 생성 중 네임스페이스 전역 고유 식별자(GUID)가 자동으로 생성되어 인스턴스에 연결됩니다. 서버리스 네임스페이스 ARN은 arn:{partition}:redshift-serverless:{region}:{account-id}:namespace/{namespace-guid} 형식으로 되어 있습니다.
- AWS 계정은 데이터 공유의 소비자가 될 수 있으며, 각각 12자리 AWS 계정 ID로 표시됩니다.

표준 데이터 공유의 경우 다음을 고려하세요.

- 생산자 클러스터가 삭제될 때 Amazon Redshift는 생산자 클러스터에서 생성한 datashare를 삭제합니다. 생산자 클러스터가 백업 및 복원될 때 생성된 datashare는 복원된 클러스터에서 계속 유지됩니다. 그러나 다른 클러스터에 부여된 datashare 권한은 복원된 클러스터에서 더 이상 유효하지 않습니다. 원하는 소비자 클러스터에 datashare 사용 권한을 다시 부여합니다. 소비자 클러스터의 소비자 데이터베이스는 스냅샷이 생성된 원래 클러스터의 datashare를 가리킵니다. 복원된 클러스터에서 공유 데이터를 쿼리하기 위해 소비자 관리자는 다른 데이터베이스를 만듭니다. 또는 관리자가 새로 복원된 클러스터의 datashare를 사용하기 위해 기존 소비자 데이터베이스를 삭제하고 다시 생성할 수 있습니다.
- 소비자 클러스터가 삭제되고 스냅샷에서 복원되면 이 클러스터에 공유된 이전 액세스가 더 이상 유효하지 않고 표시되지 않습니다. 복원된 소비자 클러스터에서 데이터 공유에 대한 액세스가 여전히 필요한 경우 생산자 관리자는 복원된 소비자 클러스터에 데이터 공유 사용 권한을 다시 부여해야 합

니다. 소비자 관리자는 비활성 데이터 공유에서 만들어졌으며 오래되어 쓸모없어진 소비자 데이터베이스를 삭제해야 합니다. 그런 다음 생산자가 권한을 다시 부여한 후 관리자는 datashare에서 소비자 데이터베이스를 다시 생성해야 합니다. 네임스페이스 GUID는 복원된 클러스터에서 원래 클러스터와 다르기 때문에 소비자 또는 생산자 클러스터가 백업에서 복원될 때 데이터 공유 권한을 다시 부여합니다.

AWS Data Exchange 데이터 공유

AWS Data Exchange 데이터 공유를 사용하여 Amazon Redshift 데이터 공유에 대한 결제를 관리할 수 있습니다.

AWS Data Exchange 데이터 공유는 AWS Data Exchange를 통해 데이터를 공유하기 위한 라이선스 단위입니다. AWS는 AWS Data Exchange 구독 및 Amazon Redshift 데이터 공유 사용과 관련된 모든 청구 및 결제를 관리합니다. 승인된 데이터 제공자는 AWS Data Exchange 제품에 AWS Data Exchange datashare를 추가할 수 있습니다. 고객이 AWS Data Exchange datashare가 있는 제품을 구독하면 제품의 datashare에 액세스할 수 있습니다.

Amazon Redshift용 AWS Data Exchange를 사용하면 AWS Data Exchange를 통해 Amazon Redshift 데이터에 대한 액세스에 편리하게 라이선스를 부여할 수 있습니다. 고객이 AWS Data Exchange datashare가 있는 제품에 가입하면 AWS Data Exchange는 해당 제품에 포함된 모든 AWS Data Exchange datashare에 고객을 데이터 소비자로 자동 추가합니다. 인보이스가 자동으로 생성되고 지불이 중앙에서 수집되고 AWS Marketplace Entitlement Service을 통해 자동으로 지출됩니다.

공급자는 스키마, 테이블, 보기 및 사용자 정의 함수와 같은 세분화된 수준에서 Amazon Redshift의 데이터에 라이선스를 부여할 수 있습니다. 여러 AWS Data Exchange 제품에서 동일한 AWS Data Exchange datashare를 사용할 수 있습니다. AWS Data Exchange datashare에 추가된 모든 객체는 소비자가 사용할 수 있습니다. 생산자는 Amazon Redshift API 작업, SQL 명령 및 Amazon Redshift 콘솔을 통해 생산자를 대신하여 AWS Data Exchange에서 관리하는 모든 AWS Data Exchange datashare를 볼 수 있습니다. 제품 AWS Data Exchange datashare를 구독하는 고객은 datashare의 객체에 대한 읽기 전용 액세스 권한을 가집니다.

서드 파티 생산자 데이터를 사용하려는 고객은 AWS Data Exchange 카탈로그를 검색하여 Amazon Redshift에서 데이터 집합을 검색하고 구독할 수 있습니다. AWS Data Exchange 구독이 활성화되면 클러스터의 datashare에서 데이터베이스를 생성하고 Amazon Redshift에서 데이터를 쿼리할 수 있습니다.

AWS Data Exchange datashare 작동 방식

생산자 관리자로 AWS Data Exchange datashare 관리

데이터 생산자(AWS Data Exchange에서는 공급자라고도 함)는 Amazon Redshift 데이터베이스에 연결하는 AWS Data Exchange datashare를 생성할 수 있습니다. AWS Data Exchange의 제품에 AWS Data Exchange datashare를 추가하려면 등록된 AWS Data Exchange 공급자여야 합니다.

AWS Data Exchange datashare를 시작하는 방법에 대한 자세한 내용은 AWS Data Exchange에서 라이선스가 부여된 Amazon Redshift 데이터 공유 섹션을 참조하세요.

활성 AWS Data Exchange 구독이 있는 소비자로 AWS Data Exchange datashare 사용

활성 AWS Data Exchange 구독(AWS Data Exchange에서는 구독자라고도 함)이 있는 소비자는 AWS Data Exchange 콘솔에서 AWS Data Exchange 카탈로그를 찾아보고 AWS Data Exchange datashare가 포함된 제품을 검색할 수 있습니다.

AWS Data Exchange datashare가 포함된 제품을 구독한 후 클러스터 내의 datashare에서 데이터베이스를 생성합니다. 그런 다음 데이터를 추출, 변환 및 로드하지 않고 Amazon Redshift에서 직접 데이터를 쿼리할 수 있습니다.

AWS Data Exchange datashare를 시작하는 방법에 대한 자세한 내용은 AWS Data Exchange에서 라이선스가 부여된 Amazon Redshift 데이터 공유 섹션을 참조하세요.

AWS Data Exchange 데이터 교환 데이터 공유의 경우 다음을 고려하세요.

- 생산자 클러스터가 삭제될 때 Amazon Redshift는 생산자 클러스터에서 생성한 datashare를 삭제합니다. 생산자 클러스터가 백업 및 복원될 때 생성된 datashare는 복원된 클러스터에서 계속 유지됩니다. 데이터 구독자가 계속해서 데이터에 액세스할 수 있도록 하려면 AWS Data Exchange datashare를 다시 생성하고 제품의 데이터 집합에 게시합니다. 소비자 클러스터의 소비자 데이터베이스는 스냅샷이 생성된 원래 클러스터의 datashare를 가리킵니다. 복원된 클러스터에서 공유 데이터를 쿼리하기 위해 소비자 관리자는 다른 데이터베이스를 만들거나 기존 소비자 데이터베이스를 삭제하고 다시 만들어 새로 복원된 클러스터에서 새로 만들어진 AWS Data Exchange 데이터 공유를 사용합니다.
- 소비자 클러스터가 삭제되고 스냅샷에서 복원되면 이 클러스터에 공유된 이전 액세스 권한은 유효하고 표시됩니다. 소비자 관리자는 비활성 데이터 공유에서 만들어졌고 오래되어 쓸모없어진 소비자 데이터베이스를 삭제하고 생산자가 권한을 다시 부여한 후 데이터 공유에서 소비자 데이터베이스를 다시 만들어야 합니다. 네임스페이스 GUID는 복원된 클러스터에서 원래 클러스터와 다르기 때문에 생산자 클러스터가 백업에서 복원될 때 데이터 공유 권한을 다시 부여합니다.

- AWS Data Exchange datashare가 있는 경우 클러스터를 삭제하지 않는 것이 좋습니다. 이러한 유형의 변경을 수행하면 AWS Data Exchange의 데이터 제품 조건을 위반할 수 있습니다.

데이터 공유 생산자와 소비자

데이터 생산자(데이터 공유 생산자 또는 데이터 공유 생산자라고도 함)는 공유하려는 데이터가 있는 클러스터입니다. 생산자 관리자와 데이터베이스 소유자는 CREATE DATASHARE 명령을 사용하여 데이터 공유를 만들 수 있습니다. 생산자 클러스터가 소비자 클러스터와 공유할 데이터베이스에서 스키마, 테이블, 보기 및 SQL 사용자 정의 함수(UDF)와 같은 객체를 추가할 수 있습니다.

AWS Data Exchange datashare에 대한 데이터 생산자(AWS Data Exchange에서는 공급자라고도 함)는 AWS Data Exchange를 통해 데이터에 라이선스를 부여할 수 있습니다. 승인된 제공자는 AWS Data Exchange 제품에 AWS Data Exchange datashare를 추가할 수 있습니다.

고객이 AWS Data Exchange datashare가 있는 제품을 구독하면 AWS Data Exchange는 제품에 포함된 모든 AWS Data Exchange datashare에 고객을 데이터 소비자로 자동 추가합니다. AWS Data Exchange는 또한 구독이 종료되면 AWS Data Exchange datashare에서 모든 고객을 제거합니다. AWS Data Exchange는 또한 AWS Data Exchange datashare를 통해 유료 제품에 대한 청구, 인보이스 발행, 결제 수집 및 결제 배포를 자동으로 관리합니다. 자세한 내용은 AWS Data Exchange 데이터 공유 단원을 참조하십시오. AWS Data Exchange 데이터 공급자로 등록하려면 공급자로 시작하기를 참조하세요.

데이터 소비자(datashare 소비자라고도 함)는 생산자 클러스터에서 datashare를 수신하는 클러스터입니다.

데이터를 공유하는 Amazon Redshift 클러스터는 동일하거나 다른 AWS 계정 또는 다른 AWS 리전에 있을 수 있으므로 조직 간에 데이터를 공유하고 다른 당사자와 협업할 수 있습니다. 소비자 관리자는 사용 권한을 부여받은 데이터 공유를 받고 각 데이터 공유의 내용을 검토합니다. 공유 데이터를 사용하기 위해 소비자 관리자는 데이터 공유에서 Amazon Redshift 데이터베이스를 만듭니다. 그런 다음 관리자는 데이터베이스에 대한 권한을 소비자 클러스터의 사용자 및 역할에 할당합니다. 권한이 부여되면 사용자 및 역할은 소비자 클러스터의 로컬 데이터와 함께 표준 메타데이터 쿼리의 일부로 공유 객체를 나열할 수 있습니다. 또한 즉시 쿼리를 시작할 수 있습니다.

활성 AWS Data Exchange 구독이 있는 소비자(AWS Data Exchange에서는 구독자라고도 함)인 경우 Amazon Redshift에서 데이터를 추출, 변환 및 로드할 필요 없이 세분화된 최신 데이터를 찾고 구독하고 쿼리할 수 있습니다. 자세한 내용은 AWS Data Exchange 데이터 공유 단원을 참조하십시오.

Amazon Redshift용 AWS Data Exchange 사용 시 고려 사항

Amazon Redshift용 AWS Data Exchange 사용 시 다음 사항을 고려하세요.

- 생산자와 소비자 모두 Amazon Redshift datashare를 사용하려면 RA3 인스턴스 유형을 사용해야 합니다. 생산자는 최신 Amazon Redshift 클러스터 버전과 함께 RA3 인스턴스 유형을 사용해야 합니다.
- 생산자 클러스터와 소비자 클러스터를 모두 암호화해야 합니다.
- AWS Data Exchange datashare가 포함된 제품을 비롯하여 AWS Data Exchange에 제품을 나열하려면 AWS Data Exchange 공급자로 등록해야 합니다. 자세한 내용은 공급자로 시작하기를 참조하세요.
- AWS Data Exchange를 통해 Amazon Redshift 데이터를 찾고, 구독하고, 쿼리하기 위해 등록된 AWS Data Exchange 공급자일 필요는 없습니다.
- 데이터에 대한 액세스를 제어하려면 공개적으로 액세스할 수 있는 설정이 지정된 상태에서 AWS Data Exchange datashare를 생성합니다. 공개적으로 액세스할 수 있는 설정을 해제하도록 AWS Data Exchange datashare를 변경하려면 ALTER DATASHARE SET PUBLICACCESSIBLE FALSE를 허용하도록 세션 변수를 설정합니다. 자세한 내용은 ALTER DATASHARE 사용 참고 사항 단원을 참조하십시오.
- AWS Data Exchange datashare에 대한 액세스 권한은 datashare가 포함된 AWS Data Exchange 제품에 대한 활성 구독을 기반으로 부여되기 때문에 생산자는 AWS Data Exchange datashare에서 소비자를 수동으로 추가하거나 제거할 수 없습니다.
- 생산자는 소비자가 실행하는 SQL 쿼리를 볼 수 없습니다. 생산자만 액세스할 수 있는 Amazon Redshift 테이블을 통해서만 쿼리 수 또는 소비자가 쿼리하는 객체와 같은 메타데이터를 볼 수 있습니다. 자세한 내용은 Amazon Redshift의 데이터 공유 모니터링 및 감사 단원을 참조하십시오.
- datashare를 공개적으로 액세스할 수 있게 하는 것이 좋습니다. 그렇지 않으면 공개적으로 액세스할 수 있는 소비자 클러스터가 있는 AWS Data Exchange의 구독자가 datashare를 사용할 수 없습니다.
- DROP DATASHARE 문을 사용하여 다른 AWS 계정과 공유되는 AWS Data Exchange datashare를 삭제하지 않는 것이 좋습니다. 그렇게 하면 datashare에 대한 액세스 권한이 있는 AWS 계정가 액세스 권한을 상실합니다. 이 작업은 되돌릴 수 없습니다. 이러한 유형의 변경을 수행하면 AWS Data Exchange의 데이터 제품 조건을 위반할 수 있습니다. AWS Data Exchange datashare를 삭제하려는 경우 DROP DATASHARE 사용 참고 사항 섹션을 참조하세요.
- 리전 간 데이터 공유의 경우 AWS Data Exchange datashare를 생성하여 라이선스 데이터를 공유할 수 있습니다.
- 다른 리전의 데이터를 사용하는 경우 소비자는 생산자 리전에서 소비자 리전으로의 리전 간 데이터 전송 요금을 지불합니다.

AWS Lake Formation-관리형 데이터 공유

Amazon Redshift를 사용하면 AWS Lake Formation 관리형 데이터 공유를 통해 AWS 계정 및 Amazon Redshift 클러스터 전반에서 실시간 데이터에 액세스하고 공유할 수 있습니다. AWS Lake Formation 데이터 공유에서는 데이터 공급자가 다른 AWS 계정 및 Amazon Redshift 클러스터를 포함한 모든 소비자와 Amazon S3 데이터 레이크의 실시간 데이터를 안전하게 공유하도록 지원합니다.

AWS Lake Formation을 사용하면 Amazon Redshift 데이터 공유의 데이터베이스, 테이블, 열 및 행 수준 액세스 권한을 중앙에서 정의 및 적용하고 데이터 공유 내의 객체에 대한 사용자 액세스를 제한할 수 있습니다. Lake Formation을 통해 데이터를 공유하면 Lake Formation에서 권한을 정의하고 해당 권한을 모든 데이터 공유 및 해당 객체에 적용할 수 있습니다. 예를 들어 직원 정보가 포함된 테이블이 있는 경우 Lake Formation의 열 수준 필터를 사용하여 HR 부서에서 일하지 않는 직원이 주민등록번호와 같은 개인 식별 정보(PII)를 보지 못하도록 할 수 있습니다. 데이터 필터에 대한 자세한 내용은 AWS Lake Formation 개발자 안내서의 Lake Formation에서 데이터 필터링 및 셀 수준 보안을 참조하세요.

Lake Formation의 태그를 사용하여 Lake Formation 리소스에 대한 권한을 구성할 수도 있습니다. 자세한 내용은 Lake Formation 태그 기반 액세스 제어를 참조하세요.

Amazon Redshift는 현재 동일한 계정 내에서 또는 계정 간에 공유할 때 Lake Formation을 통한 데이터 공유를 지원합니다. 현재 교차 리전 공유는 지원되지 않습니다.

다음은 Lake Formation을 사용하여 데이터 공유 권한을 제어하는 방법에 대한 높은 수준의 개요입니다.

1. Amazon Redshift에서 생산자 클러스터 또는 작업 그룹 관리자는 생산자 클러스터 또는 작업 그룹에 데이터 공유를 생성하고 Lake Formation 계정에 사용 권한을 부여합니다.
2. 생산자 클러스터 또는 작업 그룹 관리자는 데이터 공유에 액세스할 수 있도록 Lake Formation 계정에 권한을 부여합니다.
3. Lake Formation 관리자는 데이터 공유를 검색하고 등록합니다. 또한 액세스 권한이 있는 AWS Glue ARN을 검색하고 데이터 공유를 AWS Glue Data Catalog ARN과 연결해야 합니다. AWS CLI를 사용하는 경우 Redshift CLI 작업 `describe-data-shares` 및 `associate-data-share-consumer`를 통해 데이터 공유를 검색 및 수락할 수 있습니다. 데이터 공유를 등록하려면 Lake Formation CLI 작업 `register-resource`를 사용하세요.
4. Lake Formation 관리자는 AWS Glue Data Catalog에 페더레이션 데이터베이스를 생성하고 데이터 공유 내의 객체에 대한 사용자 액세스를 제어하도록 Lake Formation 권한을 구성합니다. AWS Glue의 페더레이션 데이터베이스에 대한 자세한 내용은 Amazon Redshift 데이터 공유의 데이터에 대한 권한 관리를 참조하세요.

5. Lake Formation 관리자는 액세스 권한이 있는 AWS Glue 데이터베이스를 검색하고 데이터 공유를 AWS Glue Data Catalog ARN과 연결합니다.
6. Redshift 관리자는 액세스 권한이 있는 AWS Glue 데이터베이스 ARN을 검색하고, AWS Glue 데이터베이스 ARN을 사용하여 Amazon Redshift 소비자 클러스터에 외부 데이터베이스를 생성하며, IAM 보안 인증 정보로 인증된 데이터베이스 사용자에게 Amazon Redshift 데이터베이스 쿼리를 시작할 수 있는 사용 권한을 부여합니다.
7. 데이터베이스 사용자는 SVV_EXTERNAL_TABLES 및 SVV_EXTERNAL_COLUMNS 뷰를 사용하여 액세스 권한이 있는 AWS Glue 데이터베이스 내의 모든 테이블 또는 열을 찾은 다음 AWS Glue 데이터베이스의 테이블을 쿼리할 수 있습니다.
8. 생산자 클러스터 또는 작업 그룹 관리자가 더 이상 소비자 클러스터와 데이터를 공유하지 않기로 결정하면 생산자 관리자는 Redshift로부터 데이터 공유의 사용 권한을 취소하거나, 데이터 공유의 권한을 제거하거나, 데이터 공유를 삭제할 수 있습니다. Lake Formation의 연결된 권한 및 객체는 자동으로 삭제되지 않습니다.

생산자 클러스터 또는 작업 그룹 관리자로서 AWS Lake Formation과 데이터 공유를 공유하는 방법에 대한 자세한 내용은 생산자로서 Lake Formation에서 관리하는 데이터 공유 사용 섹션을 참조하세요. 생산자 클러스터 또는 작업 그룹의 공유 데이터를 사용하려면 소비자로서 Lake Formation에서 관리하는 데이터 공유 사용 섹션을 참조하세요.

Amazon Redshift와 함께 AWS Lake Formation을 사용할 때의 고려 사항 및 제한 사항

다음은 Lake Formation을 통해 Amazon Redshift 데이터를 공유할 때 고려해야 할 사항과 제한 사항입니다. 데이터 공유 고려 사항 및 제한 사항에 대한 자세한 내용은 Amazon Redshift에서 데이터 공유를 사용할 때의 고려 사항을 참조하세요. Lake Formation 제한에 대한 자세한 내용은 Lake Formation에서의 Amazon Redshift 데이터 공유 작업에 대한 참고 사항을 참조하세요.

- 리전 간 Lake Formation에 데이터 공유를 공유하는 것은 현재 지원되지 않습니다.
- 공유 관계에서 사용자에 대해 열 수준 필터가 정의된 경우 `SELECT *` 작업을 수행하면 사용자가 액세스할 수 있는 열만 반환됩니다.
- Lake Formation의 셀 수준 필터는 지원되지 않습니다.
- 뷰 및 해당 테이블을 생성하고 Lake Formation에 공유한 경우 테이블 액세스를 관리하도록 필터를 구성할 수 있으며, 소비자 클러스터 사용자가 공유 객체에 액세스하는 경우 Amazon Redshift가 Lake Formation에 정의된 정책을 적용합니다. 사용자가 Lake Formation과 공유된 뷰에 액세스하면 Redshift는 뷰에 정의된 Lake Formation 정책만 적용하고 뷰에 포함된 테이블에 정의된 정책은 적용하지 않습니다. 그러나 사용자가 테이블에 직접 액세스하면 Redshift는 정의된 Lake Formation 정책을 테이블에 적용합니다.

- 공유 테이블에 Lake Formation 필터가 구성된 경우 공유 테이블을 기반으로 소비자에 구체화된 뷰를 생성할 수 없습니다.
- Lake Formation 관리자에게는 데이터 레이크 관리자 권한과 데이터 공유를 수락하는 데 필요한 권한이 있어야 합니다.
- 생산자 소비자 클러스터가 Lake Formation을 통해 데이터 공유를 공유하려면 최신 Amazon Redshift 클러스터 버전 또는 서버리스 작업 그룹이 포함된 RA3 클러스터여야 합니다.
- 생산자 클러스터와 소비자 클러스터를 모두 암호화해야 합니다.
- 생산자 클러스터 또는 작업 그룹에 구현된 Redshift 행 수준 및 열 수준 액세스 제어 정책은 데이터 공유가 Lake Formation에 공유될 때 무시됩니다. Lake Formation 관리자는 Lake Formation에서 이러한 정책을 구성해야 합니다. 생산자 클러스터 또는 작업 그룹 관리자는 ALTER TABLE 명령을 사용하여 테이블에 대한 RLS를 끌 수 있습니다.
- Lake Formation을 통한 데이터 공유는 Redshift와 Lake Formation에 모두 액세스할 수 있는 사용자만 사용할 수 있습니다.

Amazon Redshift의 데이터 공유 상태 값

Amazon Redshift를 사용하면 데이터를 복사하거나 전송할 필요 없이 Amazon Redshift 클러스터 전반에서 실시간 데이터를 안전하게 공유할 수 있습니다. Amazon Redshift용 데이터 공유로 소스 데이터에 대한 업데이트를 포함하여 실시간 쿼리 결과를 동일하거나 다른 AWS 계정 및 AWS 리전의 모든 Amazon Redshift 클러스터와 공유합니다. 이 주제에서는 데이터 공유를 통해 Amazon Redshift에서 나타날 수 있는 상태를 설명합니다.

계정 간 datashare의 경우 작업이 필요한 여러 가지 datashare 상태가 있습니다. 데이터 공유에는 활성, 작업 필요 또는 비활성 상태가 포함될 수 있습니다.

다음은 각 datashare 상태와 필요한 작업에 대한 설명입니다.

- 생산자 관리자가 데이터 공유를 만들 때 생산자 클러스터의 데이터 공유 상태는 권한 부여 보류 중입니다. 생산자 관리자는 데이터 소비자가 데이터 공유에 액세스하도록 권한을 부여할 수 있습니다. 소비자 관리자의 작업은 없습니다.
- 생산자 관리자가 데이터 공유 권한을 부여하면 생산자 클러스터에서 데이터 공유 상태가 권한 부여됨이 됩니다. 생산자 관리자의 작업은 없습니다. datashare에 대한 데이터 소비자와의 연결이 하나 이상 있는 경우 datashare 상태가 [권한 부여됨(Authorized)]에서 [활성(Active)]으로 바뀝니다.

  그러면 소비자 클러스터에서 datashare 상태가 [사용 가능(Amazon Redshift 콘솔에서 작업 필요)]이 됩니다. 소비자 관리자는 데이터 공유를 데이터 소비자와 연결하거나 데이터 공유를 거부할 수 있습

니다. 소비자 관리자는 AWS CLI 명령 `describeDatashareforConsumer`를 사용하여 데이터 공유 상태를 볼 수도 있습니다. 또는 관리자는 CLI 명령 `describeDatashare`를 사용하고 datashare Amazon 리소스 이름(ARN)을 제공하여 datashare 상태를 볼 수 있습니다.

- 소비자 관리자가 데이터 공유를 데이터 소비자와 연결하면 생산자 클러스터에서 데이터 공유 상태가 활성이 됩니다. datashare에 대한 데이터 소비자와의 연결이 하나 이상 있는 경우 datashare 상태가 [권한 부여됨(Authorized)]에서 [활성(Active)]으로 바뀝니다. 생산자 관리자가 해야 할 작업은 없습니다.

  소비자 클러스터에서 datashare 상태가 [활성(Active)]이 됩니다. 소비자 관리자가 해야 할 작업은 없습니다.
- 소비자 관리자가 데이터 공유에서 소비자 연결을 제거하면 데이터 공유 상태가 활성 또는 권한 부여됨이 됩니다. 다른 데이터 소비자와의 datashare에 대해 연결이 하나 이상 있으면 [활성(Active)]이 됩니다. 생산자 클러스터의 datashare와의 소비자 연결이 없으면 [권한 부여됨(Authorized)]이 됩니다. 생산자 관리자의 작업은 없습니다.

  모든 연결이 제거되면 datashare 상태는 소비자 클러스터에서 [작업 필요(Action required)]가 됩니다. 소비자 관리자는 소비자가 데이터 공유를 사용할 수 있을 때 데이터 공유를 데이터 소비자와 다시 연결할 수 있습니다.
- 소비자 관리자가 데이터 공유를 거부하면 생산자 클러스터의 데이터 공유 상태가 소비자 클러스터에서 작업 필요 및 거부됨이 됩니다. 생산자 관리자는 데이터 공유 권한을 다시 부여할 수 있습니다. 소비자 관리자의 작업은 없습니다.
- 생산자 관리자가 데이터 공유에서 권한 부여를 제거하면 생산자 클러스터에서 데이터 공유 상태가 작업 필요가 됩니다. 생산자 관리자는 필요한 경우 데이터 공유 권한을 다시 부여할 수 있습니다. 소비자 관리자가 해야 할 작업은 없습니다.

IAM 정책으로 데이터 공유 API 작업에 대한 액세스 관리

데이터 공유 API 작업에 대한 액세스를 제어하려면 IAM 작업 기반 정책을 사용합니다. IAM 정책을 관리하는 방법에 대한 자세한 내용은 IAM User Guide의 Managing IAM policies 섹션을 참조하세요.

데이터 공유 API 작업을 사용하는 데 필요한 권한에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 데이터 공유 API 작업을 사용하는 데 필요한 권한 섹션을 참조하세요.

계정 간 데이터 공유를 보다 안전하게 하기 위해 `AuthorizeDataShare` 및 `DeauthorizeDataShare` API 작업에 조건부 키 `ConsumerIdentifier`를 사용할 수 있습니다. 이렇게 하면 두 API 작업을 호출할 수 있는 AWS 계정를 명시적으로 제어할 수 있습니다.

자신의 계정이 아닌 소비자에 대한 데이터 공유 권한 부여 또는 권한 부여 취소를 거부할 수 있습니다. 이렇게 하려면 IAM 정책에서 AWS 계정 번호를 지정합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Deny",
            "Action": [
                "redshift:AuthorizeDataShare",
                "redshift:DeauthorizeDataShare"
            ],
            "Resource": "*",
            "Condition": {
                "StringNotEquals": {
                    "redshift:ConsumerIdentifier": "555555555555"
                }
            }
        }
    ]
}
```

IAM 정책에서 DataShareArn **`testshare2`**가 있는 생산자가 AWS 계정이 111122223333인 소비자와 명시적으로 공유하도록 허용할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "redshift:AuthorizeDataShare",
                "redshift:DeauthorizeDataShare"
            ],
            "Resource": "arn:aws:redshift:us-
east-1:666666666666:datashare:af06285e-8a45-4ee9-b598-648c218c8ff1/testshare2",
            "Condition": {
                "StringEquals": {
                    "redshift:ConsumerIdentifier": "111122223333"
                }
            }
```

```
        }
    ]
}
```

Amazon Redshift의 소비자 데이터베이스에 연결

데이터 공유 데이터베이스에 직접 연결하면 다른 유형의 Amazon Redshift 데이터베이스에 연결하는 것과 동일한 방식으로 데이터 공유에서 만들어진 데이터베이스에 직접 연결할 수 있습니다. 예를 들어 JDBC 또는 ODBC 드라이버, Amazon Redshift Query Editor V2 또는 Amazon Redshift 데이터베이스에 연결할 수 있는 기타 도구를 사용하여 데이터 공유에서 만들어진 데이터베이스에 연결하면 됩니다. 자세한 내용은 SQL 클라이언트 도구를 사용하여 Amazon Redshift 데이터 웨어하우스에 연결을 참조하시기 바랍니다.

공유 데이터에 액세스

데이터 공유에서 만들어진 데이터베이스에 연결하면 2부분 표기법(*schema_name*.*table_name*)을 사용하여 공유 객체를 쿼리할 수 있습니다. 소비자 데이터베이스 검색 경로에서 테이블을 찾을 수 있으면 1부분 표기법(*table_name*)을 사용하는 것도 가능합니다.

데이터베이스 간 쿼리를 수행하려면 3부분 표기법(*consumer_database_name*.*schema_name*.*table_name*)을 사용합니다. 이러한 쿼리는 클러스터의 다른 소비자 데이터베이스에서 공유된 객체 또는 로컬 데이터베이스의 로컬 객체를 참조합니다. 동일한 쿼리 내에서 로컬 데이터와 다른 클러스터에서 공유된 데이터를 모두 쿼리할 수 있습니다.

> **Note**
>
> 데이터 공유에서 만들어진 데이터베이스에는 로컬 카탈로그가 없습니다. 따라서 `pg_class` 등의 로컬 카탈로그 테이블에 액세스하는 모든 쿼리에는 빈 결과가 반환됩니다.

공유 객체의 메타데이터에 액세스

클러스터 관리자가 소비자 데이터베이스에서 공유 객체를 검색하도록 지원하기 위해 Amazon Redshift에서는 이러한 객체의 메타데이터를 나열하는 메타데이터 뷰 및 SHOW 명령 모음을 제공합니다. 소비자 데이터베이스에 연결하면 이러한 메타데이터 뷰 및 명령에서 데이터베이스 간 메타데이터 검색이 지원되지 않습니다. 연결된 데이터베이스와 관련 있는 데이터 공유의 공유 객체에 대한 메타데이터만 반환됩니다.

연결된 데이터베이스와 관련 있는 데이터 공유에서 공유 스키마 목록을 보려면 SHOW SCHEMAS를 사용합니다. 자세한 내용은 SHOW SCHEMAS 단원을 참조하십시오.

연결된 데이터베이스와 관련 있는 데이터 공유에서 공유 스키마의 테이블 목록을 보려면 SHOW TABLES를 사용합니다. 자세한 내용은 SHOW TABLES 단원을 참조하십시오.

연결된 데이터베이스와 관련 있는 데이터 공유에서 공유 테이블의 열 목록을 보려면 SHOW COLUMNS를 사용합니다. 자세한 내용은 SHOW COLUMNS 단원을 참조하십시오.

연결된 데이터베이스와 관련 있는 데이터 공유에서 공유 스키마 목록을 보려면 SVV_ALL_SCHEMAS를 사용합니다. 자세한 내용은 SVV_ALL_SCHEMAS 단원을 참조하십시오.

연결된 데이터베이스와 관련 있는 데이터 공유에서 공유 테이블 목록을 보려면 SVV_ALL_TABLES를 사용합니다. 자세한 내용은 SVV_ALL_TABLES 단원을 참조하십시오.

연결된 데이터베이스와 관련 있는 데이터 공유에서 공유 열의 목록을 보려면 SVV_ALL_COLUMNS를 사용합니다. 자세한 내용은 SVV_ALL_COLUMNS 단원을 참조하십시오.

비즈니스 인텔리전스 도구와 Amazon Redshift 데이터 공유 통합

datashare를 비즈니스 인텔리전스(BI) 도구와 통합하려면 Amazon Redshift JDBC 또는 ODBC 드라이버를 사용하는 것이 좋습니다. Amazon Redshift JDBC 및 ODBC 드라이버는 드라이버의 `GetCatalogs` API 작업을 지원합니다. 이 작업에서는 데이터 공유에서 만들어진 데이터베이스를 포함하여 모든 데이터베이스 목록이 반환됩니다.

드라이버는 `GetCatalogs`가 반환하는 모든 데이터베이스의 데이터를 반환하는 `GetSchemas`, GetTables 등의 다운스트림 작업도 지원합니다. 드라이버는 카탈로그가 직접 호출에 명시적으로 지정되지 않은 경우에도 이 지원을 제공합니다. JDBC 또는 ODBC 드라이버에 관한 자세한 내용은 Amazon Redshift 관리 가이드의 연결 구성 을 참조하시기 바랍니다.

Amazon Redshift Query Editor V2의 해당 연결 전환 인터페이스에는 소비자 데이터베이스가 포함되어 있습니다. 그러나 대부분의 도구에는 이러한 데이터베이스가 제외되어 있으며, 연결 가능한 데이터베이스로 로컬 클러스터 데이터베이스만 포함되어 있습니다.

> **Note**
>
> 내부 유지 관리를 위해 `sys:internal`이라는 새 시스템 데이터베이스가 추가되었습니다. 일부 도구에는 이 시스템 데이터베이스가 연결 가능한 데이터베이스로 포함됩니다. 그러나 객체에 연결하거나 객체에 대해 쿼리를 실행할 수 없습니다.

Amazon Redshift의 데이터 공유 모니터링 및 감사

Amazon Redshift를 사용하면 데이터 공유 활동을 모니터링하고 감사하여 규정 준수 및 보안을 보장할 수 있습니다.

생산자는 datashare를 감사하여 datashare 변화를 추적할 수 있습니다. 예를 들어 감사는 datashare가 생성되고, 객체가 추가 또는 제거되고, Amazon Redshift 클러스터 및 AWS 계정 또는 AWS 리전에 권한이 부여 또는 취소되는 시점을 추적하는 데 도움이 됩니다.

감사 외에도 생산자와 소비자는 계정, 클러스터 및 객체 수준과 같은 다양한 세부 수준에서 datashare 사용량을 추적합니다. 사용량 추적 및 감사 뷰에 대한 자세한 내용은 SVL_DATASHARE_CHANGE_LOG 및 SVL_DATASHARE_USAGE_PRODUCER 섹션을 참조하세요.

시스템 뷰를 쿼리하여 데이터 공유를 모니터링할 수 있습니다.

1. 데이터를 공유하려는 생산자 관리자는 Amazon Redshift 데이터 공유를 만듭니다. 그런 다음 생산자 관리자가 필요한 데이터베이스 객체를 추가합니다. 이는 datashare에 대한 스키마, 테이블 및 보기일 수 있으며 객체를 공유할 소비자 목록을 지정합니다.

   다음 시스템 뷰를 사용하여 생산자 및/또는 소비자 클러스터에서 데이터 공유의 변경 사항 및 사용량을 추적할 수 있는 통합 뷰를 확인하세요:

   - SYS_DATASHARE_CHANGE_LOG
   - SYS_DATASHARE_USAGE_CONSUMER
   - SYS_DATASHARE_USAGE_PRODUCER

   아웃바운드 datashare에 대한 datashare 객체 및 데이터 소비자 정보를 보려면 다음 시스템 뷰를 사용합니다.

   - SVV_DATASHARES
   - SVV_DATASHARE_CONSUMERS
   - SVV_DATASHARE_OBJECTS
2. 소비자 관리자는 사용 권한이 부여된 데이터 공유를 보고 SVV_DATASHARES를 사용해 인바운드 데이터 공유를 확인하여 각 데이터 공유의 내용을 검토합니다.

   공유 데이터를 사용하기 위해 각 소비자 관리자는 데이터 공유에서 Amazon Redshift 데이터베이스를 만듭니다. 그런 다음 관리자는 소비자 클러스터의 적절한 사용자 및 역할에 권한을 할당합니다. 사용자와 역할은 다음 메타데이터 시스템 뷰를 보고 표준 메타데이터 쿼리의 일부로 공유 객체를 나열할 수 있으며 데이터 쿼리를 즉시 시작할 수 있습니다.

   - SVV_REDSHIFT_COLUMNS

- SVV_REDSHIFT_DATABASES
- SVV_REDSHIFT_FUNCTIONS
- SVV_REDSHIFT_SCHEMAS
- SVV_REDSHIFT_TABLES

Amazon Redshift 로컬 및 공유 스키마와 외부 스키마 모두의 객체를 보려면 다음 메타데이터 시스템 뷰를 사용하여 쿼리합니다.

- SVV_ALL_COLUMNS
- SVV_ALL_SCHEMAS
- SVV_ALL_TABLES

소비자 데이터베이스에 연결하면 데이터베이스 간 검색이 사용 해제됩니다. 메타데이터 시스템 뷰는 연결된 데이터베이스와 관련 있는 데이터 공유의 공유 객체에 대한 메타데이터만 반환합니다.

AWS CloudTrail과 Amazon Redshift 데이터 공유 통합

데이터 공유는 AWS CloudTrail과 통합됩니다. CloudTrail은 Amazon Redshift에서 사용자, 역할 또는 AWS 서비스가 수행한 작업의 레코드를 제공하는 서비스입니다. CloudTrail은 데이터 공유에 대한 모든 API 호출을 이벤트로 캡처합니다. 캡처된 호출에는 AWS CloudTrail 콘솔의 호출과 데이터 공유 작업에 대한 코드 호출이 포함됩니다. Amazon Redshift와 AWS CloudTrail 통합에 대한 자세한 내용은 CloudTrail을 사용한 로깅을 참조하십시오.

CloudTrail에 대한 자세한 내용은 CloudTrail 작동 방식을 참조하십시오.

Amazon Redshift의 반정형 데이터

Amazon Redshift의 비정형 데이터 지원을 사용하여 Amazon Redshift 데이터 웨어하우스에서 비정형 데이터를 수집하고 저장할 수 있습니다. Amazon Redshift는 SUPER 데이터 형식과 PartiQL 언어로 데이터 웨어하우스 기능을 확장하여 SQL 및 NoSQL 데이터 원본과 통합합니다. 이러한 방식으로 Amazon Redshift는 JSON과 같은 관계형 및 비정형 저장 데이터에 대한 효율적인 분석을 지원합니다.

Amazon Redshift는 SUPER 데이터 형식과 Amazon Redshift Spectrum의 두 가지 형태의 비정형 데이터 지원을 제공합니다.

짧은 대기 시간으로 작은 JSON 데이터 배치를 삽입하거나 업데이트해야 하는 경우 SUPER 데이터 형식을 사용합니다. 또한 쿼리에 강력한 일관성, 예측 가능한 쿼리 성능, 복잡한 쿼리 지원, 진화하는 스키마 및 스키마 없는 데이터의 사용 편의성이 필요한 경우 SUPER를 사용합니다.

이와 반대로 데이터 쿼리에 다른 AWS 서비스 및 주로 보관 목적으로 Amazon S3에 저장된 데이터와의 통합이 필요한 경우 오픈 파일 형식과 함께 Amazon Redshift Spectrum을 사용합니다.

SUPER 데이터 형식의 사용 사례

Amazon Redshift에서 SUPER 데이터 형식을 사용하는 비정형 데이터 지원은 뛰어난 성능, 유연성 및 사용 편의성을 제공합니다. 다음 사용 사례는 SUPER와 함께 비정형 데이터 지원을 사용하는 방법을 보여줍니다.

빠르고 유연한 JSON 데이터 삽입 - Amazon Redshift는 JSON을 구문 분석하고 SUPER 값으로 저장할 수 있는 빠른 트랜잭션을 지원합니다. 삽입 트랜잭션은 SUPER의 속성을 기존 열로 나눈 테이블에 동일한 삽입을 수행하는 것보다 최대 5배 빠르게 작동할 수 있습니다. 예를 들어 수신 JSON이 {“a”:.., “b”:.., “c”“..., ...} 형식이라고 가정합니다. 수신 JSON을 'a', 'b', "c' 열이 있는 기존 테이블 TR에 저장하는 대신 단일 SUPER 열 S가 있는 테이블 TJ에 저장하여 삽입 성능을 몇 배 가속화할 수 있습니다. JSON에 수백 개의 속성이 있는 경우 SUPER 데이터 형식의 성능 이점은 상당합니다.

또한 SUPER 데이터 형식에는 일반 스키마가 필요하지 않습니다. 수신 JSON을 저장하기 전에 내부 검사를 수행하고 정리할 필요가 없습니다. 예를 들어 수신 JSON에 문자열 "c" 속성이 있고 다른 JSON에는 SUPER 데이터 형식 없이 정수 "c" 속성이 있다고 가정합니다. 이 경우 c_string과 c_int 열을 분리하거나 데이터를 정리해야 합니다. 반대로 SUPER 데이터 형식을 사용하면 모든 JSON 데이터가 수집 중 정보 손실 없이 저장됩니다. 나중에 SQL의 PartiQL 확장을 사용하여 정보를 분석할 수 있습니다.

검색을 위한 유연한 쿼리 - JSON과 같은 비정형 데이터를 SUPER 데이터 값에 저장한 후 스키마를 적용하지 않고 쿼리할 수 있습니다. 쿼리 전에 스키마를 적용할 필요 없이 PartiQL 동적 형식 지정 및 lax 의미 체계를 사용하여 쿼리를 실행하고 필요한 깊게 중첩 데이터를 검색할 수 있습니다.

기존의 구체화된 뷰에 대한 추출, 로드, 변환(ETL) 작업을 위한 유연한 쿼리 – 스키마 없는 비정형 데이터를 SUPER에 저장한 후 PartiQL 구체화된 뷰를 사용하여 데이터를 내부 검사하고 구체화된 뷰로 나눌 수 있습니다.

나뉜 데이터가 들어 있는 구체화된 뷰는 기존 분석 사례에 대한 성능 및 유용성 이점을 보여주는 좋은 예입니다. 나뉜 데이터에 대한 분석을 수행할 때 Amazon Redshift 구체화된 뷰의 열 형식 구조가 더 나은 성능을 제공합니다. 또한 수집된 데이터에 대한 기존 스키마가 필요한 사용자 및 BI(비즈니스 인텔리전스) 도구는 데이터의 기존 스키마 표현으로 뷰(구체화된 뷰 또는 가상 뷰)를 사용할 수 있습니다.

PartiQL 구체화된 뷰가 JSON 또는 SUPER에 있는 데이터를 기존의 열 형식의 구체화된 뷰로 추출한 후 구체화된 뷰를 쿼리할 수 있습니다. SUPER 데이터 형식이 구체화된 뷰에서 작동하는 방식에 대한 자세한 내용은 SUPER 데이터 형식과 구체화된 뷰 섹션을 참조하세요.

SUPER 유형 열의 경로에 있는 `scalar` 값에 동적 데이터 마스킹 정책을 적용할 수 있습니다. 동적 데이터 마스킹에 대한 자세한 내용은 동적 데이터 마스킹 섹션을 참조하세요. SUPER 데이터 유형에 동적 데이터 마스킹을 사용하는 방법에 대한 자세한 내용은 SUPER 데이터 유형 경로와 함께 동적 데이터 마스킹 사용 섹션을 참조하세요(미리 보기).

SUPER 데이터 형식에 대한 내용은 SUPER 형식 섹션을 참조하세요.

SUPER 데이터 형식 사용 예는 SUPER 샘플 데이터 집합으로 시작하는 이 항목의 하위 섹션을 참조하세요.

SUPER 데이터 형식 사용에 대한 개념

다음에서 몇 가지 Amazon Redshift SUPER 데이터 형식 개념을 찾아볼 수 있습니다.

Amazon Redshift에서 SUPER 데이터 형식이 무엇인지 이해 – SUPER 데이터 형식은 Amazon Redshift 스칼라와 중첩 배열 및 구조를 포함하는 스키마 없는 배열 및 구조를 저장할 수 있는 Amazon Redshift 데이터 형식입니다. SUPER 데이터 형식은 기본적으로 JSON 또는 문서 지향 소스에서 가져온 데이터를 비롯한 다양한 형식의 비정형 데이터를 저장할 수 있습니다. 새 SUPER 열을 추가하여 비정형 데이터를 저장하고 일반적인 스칼라 열과 함께 SUPER 열에 액세스하는 쿼리를 작성할 수 있습니다. SUPER 데이터 형식에 대한 자세한 내용은 SUPER 형식 섹션을 참조하세요.

스키마 없는 JSON을 SUPER로 수집 - Amazon Redshift는 유연한 비정형 SUPER 데이터 형식으로 스키마 없는 JSON을 수신하고 SUPER 값으로 수집할 수 있습니다. 예를 들어 Amazon Redshift는 JSON 값 [10.5, "first"]를 SUPER 값 [10.5, 'first'], 즉 Amazon Redshift 십진수 10.5와 varchar 'first'를 포함하는 배열로 수집할 수 있습니다. Amazon Redshift는 COPY 명령 또는 json_parse('[10.5,

"first"]')와 같은 JSON 구문 분석 기능을 사용하여 JSON을 SUPER 값으로 수집할 수 있습니다. COPY와 `json_parse`는 기본적으로 strict 구문 분석 의미 체계를 사용하여 JSON을 수집합니다. 데이터베이스 데이터 자체를 사용하여 배열 및 구조를 포함한 SUPER 값을 구성할 수도 있습니다.

스키마 없는 JSON의 불규칙한 구조를 수집하는 동안 SUPER 열의 스키마 수정이 필요 없습니다. 예를 들어 클릭 스트림을 분석하는 동안 처음에는 "IP" 및 "time" 속성이 있는 "click" 구조를 SUPER 열에 저장합니다. 이러한 변경 사항을 수집하기 위해 스키마를 변경하지 않고 "customer ID" 속성을 추가할 수 있습니다.

SUPER 데이터 형식에 사용되는 기본 형식은 텍스트 형식의 JSON 값보다 작은 공간이 필요한 이진 형식입니다. 이를 통해 쿼리 시 SUPER 값의 빠른 수집 및 런타임 처리가 가능합니다.

PartiQL을 사용하여 SUPER 데이터 쿼리 – PartiQL은 현재 많은 AWS 서비스에서 사용하는 SQL-92의 이전 버전과 호환되는 확장입니다. PartiQL을 사용하면 친숙한 SQL 구문이 SUPER의 기존 테이블 형식 SQL 데이터와 비정형 데이터 모두에 대한 액세스를 원활하게 결합합니다. 객체 및 배열 탐색을 수행하고 배열을 중첩 해제할 수 있습니다. PartiQL은 표준 SQL 언어를 확장하여 중첩 및 다중값 데이터를 선언적으로 표현하고 처리합니다.

PartiQL은 SUPER 열의 중첩 및 스키마 없는 데이터가 일급 시민인 SQL의 확장입니다. PartiQL은 쿼리 컴파일 시간 동안 모든 쿼리 표현식의 형식 검사를 요구하지 않습니다. 이 접근 방식을 사용하면 SUPER 열 내부의 실제 데이터 형식에 액세스할 때 쿼리 실행 중에 SUPER 데이터 형식을 포함하는 쿼리 표현식을 동적으로 입력할 수 있습니다. 또한 PartiQL은 형식 불일치로 인해 오류가 발생하지 않고 null을 반환하는 lax 모드에서 작동합니다. 스키마 없는 쿼리 처리와 lax 쿼리 처리가 결합된 PartiQL은 SQL 쿼리가 SUPER 열에서 수집되는 JSON 데이터를 평가하는 추출, 로드, 전송(ELT) 애플리케이션에 이상적입니다.

Redshift Spectrum과 통합 - Amazon Redshift는 JSON, Parquet 및 중첩 데이터가 있는 기타 형식을 통해 Redshift Spectrum 쿼리를 실행할 때 PartiQL의 여러 측면을 지원합니다. Redshift Spectrum은 스키마가 있는 중첩 데이터만 지원합니다. 예를 들어 Redshift Spectrum을 사용하면 JSON 데이터의 스키마 ARRAY<STRUCT<a:INTEGER, b:DECIMAL(5,2)>>에 속성 nested_schemaful_example이 있다고 선언할 수 있습니다. 이 속성의 스키마는 데이터가 항상 정수 a와 소수 b가 있는 구조를 포함하는 배열을 포함한다고 결정합니다. 더 많은 속성을 포함하도록 데이터가 변경되면 형식도 변경됩니다. 반대로 SUPER 데이터 형식에는 스키마가 필요하지 않습니다. 속성이나 형식이 다른 구조 요소가 있는 배열을 저장할 수 있습니다. 또한 일부 값은 배열 외부에 저장할 수 있습니다.

SUPER 데이터 형식을 지원하는 함수에 대한 자세한 내용은 다음을 참조하세요.

- ABS 함수
- CEILING(또는 CEIL) 함수

- FLOOR 함수
- ROUND 함수
- SIGN 함수
- TRUNC 함수

SUPER 데이터에 대한 고려 사항

SUPER 데이터 작업 시 다음 사항을 고려합니다.

- JDBC 드라이버 버전 1.2.50, ODBC 드라이버 버전 1.4.17 이상 및 Amazon Redshift Python 드라이버 버전 2.0.872 이상을 사용합니다.

  JDBC 드라이버에 대한 자세한 내용은 JDBC 연결 구성을 참조하세요.

  ODBC 드라이버에 대한 자세한 내용은 ODBC 연결 구성을 참조하세요.
- SUPER 샘플 데이터 집합에서 다음 주제에 사용된 스키마 예를 찾아보세요.
- 다음 주제에 사용된 모든 SQL 코드 예제에는 다운로드를 위해 동일한 S3 접두사가 포함되어 있습니다. 여기에는 DDL(데이터 정의 언어)과 COPY 문 및 SUPER와 함께 작동하는 특정 TPC-H 수정 쿼리가 포함됩니다.

  SQL 파일을 보거나 다운로드하려면 다음 중 하나를 수행합니다.
  - SUPER 튜토리얼 SQL 파일과 TPC-H 파일을 다운로드합니다.
  - Amazon S3 CLI를 사용하여 다음 명령을 실행합니다. 고유의 대상 경로를 사용할 수 있습니다.

```
aws s3 cp s3://redshift-downloads/semistructured/tutorialscripts/semistructured-
tutorial.sql /target/path
aws s3 cp s3://redshift-downloads/semistructured/tutorialscripts/
super_tpch_queries.sql /target/path
```

SUPER 구성에 대한 자세한 내용은 SUPER 구성 섹션을 참조하세요.

SUPER 샘플 데이터 집합

다양한 범주, 리전 및 기간에 걸친 가상 제품 판매와 관련된 데이터가 포함된 SUPER 샘플 데이터세트를 탐색하고 분석할 수 있습니다. 다음 섹션에서는 Amazon Redshift 클러스터 내에서 SUPER 샘플 데이터세트에 액세스, 쿼리 및 조작하는 방법에 대한 세부 정보를 제공합니다.

수집 및 쿼리 예제에 사용되는 테이블 스키마와 데이터 모델은 다음과 같이 정의됩니다.

```
/*customer-orders-lineitem*/
CREATE TABLE customer_orders_lineitem
(c_custkey bigint
,c_name varchar
,c_address varchar
,c_nationkey smallint
,c_phone varchar
,c_acctbal decimal(12,2)
,c_mktsegment varchar
,c_comment varchar
,c_orders super
);

/* Datamodel of documents to be stored in c_orders Super column would be as follows*/
ARRAY < STRUCT < o_orderkey:bigint
                         ,o_orderstatus:string
                         ,o_totalprice:double
                         ,o_orderdate:string
                         ,o_orderpriority:string
                         ,o_clerk:string
                         ,o_shippriority:int
                         ,o_comment:string
                         ,o_lineitems:ARRAY < STRUCT < l_partkey:bigint
                                                      ,l_suppkey:bigint
                                                      ,l_linenumber:int
                                                      ,l_quantity:double
                                                      ,l_extendedprice:double
                                                      ,l_discount:double
                                                      ,l_tax:double
                                                      ,l_returnflag:string
                                                      ,l_linestatus:string
                                                      ,l_shipdate:string
                                                      ,l_commitdate:string
                                                      ,l_receiptdate:string
                                                      ,l_shipinstruct:string
                                                      ,l_shipmode:string
                                                      ,l_comment:string
                                                      > >
                          > >

/*part*/
CREATE TABLE part
```

```
(
 p_partkey bigint
 ,p_name varchar
 ,p_mfgr varchar
 ,p_brand varchar
 ,p_type varchar
 ,p_size int
 ,p_container varchar
 ,p_retailprice decimal(12,2)
 ,p_comment varchar
 );

/*region-nations*/
CREATE TABLE region_nations
(
 r_regionkey smallint
 ,r_name varchar
 ,r_comment varchar
 ,r_nations super
 );

 /* Datamodel of documents to be stored in r_nations Super column would be as follows*/
 ARRAY < STRUCT < n_nationkey:int,n_name:string,n_comment:string > >

/*supplier-partsupp*/
CREATE TABLE supplier_partsupp
(
 s_suppkey bigint
 ,s_name varchar
 ,s_address varchar
 ,s_nationkey smallint
 ,s_phone varchar
 ,s_acctbal double precision
 ,s_comment varchar
 ,s_partsupps super
 );

 /* Datamodel of documents to be stored in s_partsupps Super column would be as
 follows*/
 ARRAY < STRUCT <
 ps_partkey:bigint,ps_availqty:int,ps_supplycost:double,ps_comment:string > >
```

Amazon Redshift로 비정형 데이터 로드

SUPER 데이터 형식을 사용하여 Amazon Redshift에서 계층 및 일반 데이터를 유지하고 쿼리합니다. Amazon Redshift는 `json_parse` 함수를 도입하여 JSON 형식의 데이터를 구문 분석하고 이를 SUPER 표현으로 변환합니다. Amazon Redshift는 COPY 명령을 사용한 SUPER 열 로드도 지원합니다. 지원되는 파일 형식은 JSON, Avro, 텍스트, CSV(쉼표로 구분된 값) 형식, Parquet 및 ORC입니다.

다음 예에 사용된 테이블에 대한 자세한 내용은 SUPER 샘플 데이터 집합 섹션을 참조하세요.

`json_parse` 함수에 대한 내용은 JSON_PARSE 함수 섹션을 참조하세요.

SUPER 데이터 형식의 기본 인코딩은 ZSTD입니다.

SUPER 열로 JSON 문서 구문 분석

`json_parse` 함수를 사용하여 SUPER 열에 JSON 데이터를 삽입하거나 업데이트할 수 있습니다. 이 함수는 JSON 형식의 데이터를 구문 분석하고 INSERT 또는 UPDATE 문에서 사용할 수 있는 SUPER 데이터 형식으로 변환합니다.

다음 예에서는 JSON 데이터를 SUPER 열에 삽입합니다. 쿼리에서 `json_parse` 함수가 누락된 경우 Amazon Redshift는 구문 분석해야 하는 JSON 형식 문자열 대신 단일 문자열로 값을 처리합니다.

SUPER 데이터 열을 업데이트하는 경우 Amazon Redshift는 전체 문서를 열 값으로 전달해야 합니다. Amazon Redshift는 부분 업데이트를 지원하지 않습니다.

```
INSERT INTO region_nations VALUES(0,
   'lar deposits. blithely final packages cajole. regular waters are final requests.
 regular accounts are according to',
   'AFRICA',
   JSON_PARSE('{"r_nations":[
      {"n_comment":" haggle. carefully final deposits detect slyly agai",
         "n_nationkey":0,
         "n_name":"ALGERIA"
      },
      {"n_comment":"ven packages wake quickly. regu",
         "n_nationkey":5,
         "n_name":"ETHIOPIA"
      },
      {"n_comment":" pending excuses haggle furiously deposits. pending, express pinto
 beans wake fluffily past t",
         "n_nationkey":14,
         "n_name":"KENYA"
```

```
      },
      {"n_comment":"rns. blithely bold courts among the closely regular packages use
 furiously bold platelets?",
         "n_nationkey":15,
         "n_name":"MOROCCO"
      },
      {"n_comment":"s. ironic, unusual asymptotes wake blithely r",
         "n_nationkey":16,
         "n_name":"MOZAMBIQUE"
      }
   ]
}'));
```

COPY를 사용하여 Amazon Redshift에서 SUPER 열 로드

다음 섹션에서는 COPY 명령을 사용하여 JSON 데이터를 Amazon Redshift로 로드하는 다양한 방법에 대해 알아볼 수 있습니다.

JSON 및 Avro에서 데이터 복사

Amazon Redshift에서 비정형 데이터 지원을 사용하면 JSON 구조의 속성을 여러 열로 나누지 않고도 JSON 문서를 로드할 수 있습니다.

Amazon Redshift는 JSON 구조가 완전히 또는 부분적으로 알려지지 않은 경우에도 COPY를 사용하여 JSON 문서를 수집하는 두 가지 방법을 제공합니다.

1. `noshred` 옵션을 사용하여 JSON 문서에서 파생된 데이터를 단일 SUPER 데이터 열에 저장합니다. 이 방법은 스키마를 알 수 없거나 변경될 것으로 예상되는 경우에 유용합니다. 따라서 이 방법을 사용하면 단일 SUPER 열에 전체 튜플을 더 쉽게 저장할 수 있습니다.
2. `auto` 또는 `jsonpaths` 옵션을 사용하여 JSON 문서를 여러 Amazon Redshift 열로 나눕니다. 속성은 Amazon Redshift 스칼라 또는 SUPER 값일 수 있습니다.

JSON 또는 Avro 형식과 함께 이러한 옵션을 사용할 수 있습니다.

나누기 전 JSON 객체의 최대 크기는 4MB입니다.

단일 SUPER 데이터 열에 JSON 문서 복사

단일 SUPER 데이터 열에 JSON 문서를 복사하려면 단일 SUPER 데이터 열이 있는 테이블을 생성합니다.

```
CREATE TABLE region_nations_noshred (rdata SUPER);
```

Amazon S3의 데이터를 단일 SUPER 데이터 열에 복사합니다. JSON 원본 데이터를 단일 SUPER 데이터 열로 수집하려면 FORMAT JSON 절에 noshred 옵션을 지정합니다.

```
COPY region_nations_noshred FROM 's3://redshift-downloads/semistructured/tpch-nested/
data/json/region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT JSON 'noshred';
```

COPY가 JSON을 성공적으로 수집한 후 테이블에는 전체 JSON 객체의 데이터가 포함된 rdata SUPER 데이터 열이 있습니다. 수집된 데이터는 JSON 계층 구조의 모든 속성을 유지합니다. 그러나 리프는 효율적인 쿼리 처리를 위해 Amazon Redshift 스칼라 형식으로 변환됩니다.

다음 쿼리를 사용하여 원래 JSON 문자열을 검색합니다.

```
SELECT rdata FROM region_nations_noshred;
```

Amazon Redshift가 SUPER 데이터 열을 생성하면 JSON 직렬화를 통해 JDBC를 문자열로 사용하여 액세스할 수 있습니다. 자세한 내용은 복잡한 중첩 JSON 직렬화 섹션을 참조하세요.

여러 SUPER 데이터 열에 JSON 문서 복사

JSON 문서를 SUPER 데이터 열 또는 Amazon Redshift 스칼라 형식일 수 있는 여러 열로 나눌 수 있습니다. Amazon Redshift는 JSON 객체의 서로 다른 부분을 여러 열로 분산합니다.

```
CREATE TABLE region_nations
(
 r_regionkey smallint
 ,r_name varchar
 ,r_comment varchar
 ,r_nations super
 );
```

이전 예의 데이터를 테이블에 복사하려면 FORMAT JSON 절에 AUTO 옵션을 지정하여 JSON 값을 여러 열로 분할합니다. COPY는 최상위 JSON 속성을 열 이름과 일치시키고 중첩 값을 JSON 배열 및 객체와 같은 SUPER 값으로 수집할 수 있도록 합니다.

```
COPY region_nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/json/
region_nation'
```

```
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT JSON 'auto';
```

JSON 속성 이름이 대문자와 소문자가 혼합된 경우 FORMAT JSON 절에 `auto ignorecase` 옵션을 지정합니다. COPY 명령 사용에 대한 자세한 내용은 'auto ignorecase' 옵션을 사용하여 JSON 데이터에서 로드 섹션을 참조하세요.

경우에 따라 열 이름과 JSON 속성이 일치하지 않거나 로드할 속성이 한 수준 이상으로 중첩됩니다. 그렇다면 `jsonpaths` 파일을 사용하여 JSON 속성을 Amazon Redshift 열에 수동으로 매핑합니다.

```
CREATE TABLE nations
(
 regionkey smallint
 ,name varchar
 ,comment super
 ,nations super
 );
```

열 이름이 JSON 속성과 일치하지 않는 테이블에 데이터를 로드하려고 한다고 가정합니다. 다음 예에서 `nations` 테이블이 그와 같은 테이블입니다. `jsonpaths` 배열에서 속성의 위치에 따라 테이블 열에 속성의 경로를 매핑하는 `jsonpaths` 파일을 생성할 수 있습니다.

```
{"jsonpaths": [
       "$.r_regionkey",
       "$.r_name",
       "$.r_comment",
       "$.r_nations
    ]
}
```

`jsonpaths` 파일의 위치는 FORMAT JSON에 대한 인수로 사용됩니다.

```
COPY nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/json/
region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT JSON 's3://redshift-downloads/semistructured/tpch-nested/data/jsonpaths/
nations_jsonpaths.json';
```

다음 쿼리를 사용하여 여러 열에 분산된 데이터를 보여주는 테이블에 액세스합니다. SUPER 데이터 열은 JSON 형식을 사용하여 인쇄됩니다.

```
SELECT r_regionkey,r_name,r_comment,r_nations[0].n_nationkey FROM region_nations ORDER
 BY 1,2,3 LIMIT 1;
```

Jsonpaths 파일은 JSON 문서의 필드를 테이블 열에 매핑합니다. 전체 문서를 SUPER 열로 로드하면서 배포 및 정렬 키와 같은 추가 열을 추출할 수 있습니다. 다음 쿼리는 전체 문서를 nations 열에 로드합니다. name 열은 정렬 키이고 regionkey 열은 배포 키입니다.

```
CREATE TABLE nations_sorted (
    regionkey smallint,
    name varchar,
    nations super
) DISTKEY(regionkey) SORTKEY(name);
```

루트 jsonpath "$"는 다음과 같이 문서의 루트에 매핑됩니다.

```
{"jsonpaths": [
       "$.r_regionkey",
       "$.r_name",
       "$"
    ]
}
```

jsonpaths 파일의 위치는 FORMAT JSON에 대한 인수로 사용됩니다.

```
COPY nations_sorted FROM 's3://redshift-downloads/semistructured/tpch-nested/data/json/
region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT JSON 's3://redshift-downloads/semistructured/tpch-nested/data/jsonpaths/
nations_sorted_jsonpaths.json';
```

텍스트 및 CSV에서 데이터 복사

Amazon Redshift는 텍스트 및 CSV 형식의 SUPER 열을 직렬화된 JSON으로 표현합니다. SUPER 열이 올바른 유형 정보와 함께 로드되려면 유효한 JSON 형식이 필요합니다. 객체, 배열, 숫자, 부울 및 null 값에서 따옴표를 제거합니다. 문자열 값을 큰따옴표로 묶습니다. SUPER 열은 텍스트 및 CSV 형식에 대한 표준 이스케이프 규칙을 사용합니다. CSV의 경우 구분 기호는 CSV 표준에 따라 이스케이프됩니다. 텍스트의 경우 선택한 구분 기호가 SUPER 필드에도 나타날 수 있는 경우 COPY 및 UNLOAD 중에 ESCAPE 옵션을 사용합니다.

```
COPY region_nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/csv/
region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT CSV;
```

```
COPY region_nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/text/
region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
DELIMITER ','
ESCAPE;
```

열 형식 Parquet 및 ORC에서 데이터 복사

비정형 데이터나 중첩 데이터가 이미 Apache Parquet 또는 Apache ORC 형식으로 사용 가능한 경우 COPY 명령을 사용하여 데이터를 Amazon Redshift로 수집할 수 있습니다.

Amazon Redshift 테이블 구조는 Parquet 또는 ORC 파일의 열 수 및 열 데이터 형식과 일치해야 합니다. COPY 명령에 SERIALIZETOJSON을 지정하면 테이블의 SUPER 열에 맞는 파일의 모든 열 형식을 SUPER로 로드할 수 있습니다. 여기에는 구조 및 배열 형식이 포함됩니다.

```
COPY region_nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/
parquet/region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT PARQUET SERIALIZETOJSON;
```

다음 예에서는 ORC 형식을 사용합니다.

```
COPY region_nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/orc/
region_nation'
IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT ORC SERIALIZETOJSON;
```

날짜 또는 시간 데이터 형식의 속성이 ORC에 있는 경우 Amazon Redshift는 SUPER로 인코딩할 때 이를 varchar로 변환합니다.

비정형 데이터 언로드

Amazon Redshift를 사용하면 텍스트, Apache Parquet, Apache ORC, Avro 등 다양한 형식으로 Amazon Redshift 클러스터에서 Amazon S3로 반정형 데이터를 내보낼 수 있습니다. 다음 섹션에서는

Amazon Redshift에서 반정형 데이터에 대한 언로드 작업을 구성하고 실행하는 프로세스를 설명합니다.

CSV or text formats

SUPER 데이터 열이 있는 테이블을 쉼표로 구분된 값(CSV) 또는 텍스트 형식으로 Amazon S3로 언로드할 수 있습니다. Amazon Redshift는 탐색 절과 중첩 해제 절을 조합해서 사용하여 SUPER 데이터 형식의 계층적 데이터를 CSV 또는 텍스트 형식으로 Amazon S3로 언로드합니다. 그런 다음 언로드된 데이터에 대해 외부 테이블을 생성하고 Redshift Spectrum을 사용하여 쿼리할 수 있습니다. UNLOAD 및 필요한 IAM 권한 사용에 대한 자세한 내용은 UNLOAD 섹션을 참조하세요.

다음 예시를 실행하기 전에 Amazon Redshift로 비정형 데이터 로드의 프로세스를 사용하여 region_nations 테이블을 채우세요. 다음 예에 사용된 테이블에 대한 자세한 내용은 SUPER 샘플 데이터 집합 섹션을 참조하세요.

다음 예에서는 데이터를 Amazon S3로 언로드합니다.

```
UNLOAD ('SELECT * FROM region_nations')
TO 's3://xxxxxx/'
IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3-Write'
DELIMITER AS '|'
GZIP
ALLOWOVERWRITE;
```

사용자 정의 문자열이 null 값을 나타내는 다른 데이터 형식과 달리 Amazon Redshift는 JSON 형식을 사용하여 SUPER 데이터 열을 내보내고 JSON 형식에 따라 null로 나타냅니다. 결과적으로 SUPER 데이터 열은 UNLOAD 명령에 사용된 NULL [AS] 옵션을 무시합니다.

Parquet format

SUPER 데이터 열이 있는 테이블을 Parquet 형식으로 Amazon S3로 언로드할 수 있습니다. Amazon Redshift는 Parquet의 SUPER 열을 JSON 데이터 유형으로 나타냅니다. 이를 통해 반정형 데이터를 Parquet로 표시할 수 있습니다. Redshift Spectrum을 사용하여 이러한 열을 쿼리하거나 COPY 명령을 사용하여 Amazon Redshift로 다시 수집할 수 있습니다. UNLOAD 및 필요한 IAM 권한 사용에 대한 자세한 내용은 UNLOAD 섹션을 참조하세요.

다음 예에서는 데이터를 Parquet 형식으로 Amazon S3로 언로드합니다.

```
UNLOAD ('SELECT * FROM region_nations')
TO 's3://xxxxxx/'
IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3-Write'
```

```
FORMAT PARQUET;
```

비정형 데이터 쿼리

Amazon Redshift를 사용하면 정형화된 데이터와 함께 JSON, Avro 또는 Ion 등의 반정형 데이터를 쿼리 및 분석할 수 있습니다. 반정형 데이터는 유연한 스키마가 있는 데이터를 나타내며 계층 구조 또는 중첩 구조를 허용합니다. 다음 섹션에서는 Amazon Redshift의 개방형 데이터 형식 지원을 사용하여 반정형 데이터를 쿼리하여 복잡한 데이터 구조에서 중요한 정보를 활용할 수 있도록 설명합니다.

Amazon Redshift는 PartiQL 언어를 사용하여 관계형, 비정형 및 중첩 데이터에 대한 SQL 호환 액세스를 제공합니다.

PartiQL은 동적 형식으로 작업합니다. 이를 통해 정형, 반정형 및 중첩 데이터 집합의 조합에 대한 직관적 필터링, 조인 및 집계가 가능합니다. PartiQL 구문은 중첩 데이터에 액세스할 때 경로 탐색을 위해 점 표기법과 배열 첨자를 사용합니다. 또한 FROM 절 항목이 배열을 반복하고 중첩 해제 작업에 사용할 수 있습니다. 다음에서 SUPER 데이터 유형 사용을 경로 및 배열 탐색, 중첩 해제, 피벗 해제 및 조인과 결합하는 다양한 쿼리 패턴에 대한 설명을 찾을 수 있습니다.

다음 예에 사용된 테이블에 대한 자세한 내용은 SUPER 샘플 데이터 집합 섹션을 참조하세요.

주제

- 탐색
- 쿼리 중첩 해제
- 객체 피벗 해제
- 동적 형식 지정
- Lax 의미 체계
- 내부 검사 유형
- 순서 기준

탐색

Amazon Redshift는 PartiQL을 사용하여 각각 [...] 대괄호와 점 표기법을 사용하여 배열과 구조를 탐색할 수 있도록 합니다. 또한 점 표기법을 사용하는 구조와 대괄호 표기법을 사용하는 배열을 혼합하여 탐색할 수 있습니다. 예를 들어 다음 예에서는 `c_orders` SUPER 데이터 열이 구조가 있는 배열이고 속성 이름이 `o_orderkey`라고 가정합니다.

customer_orders_lineitem 테이블의 데이터를 수집하려면 다음 명령을 실행합니다. IAM 역할을 사용자의 자격 증명으로 바꿉니다.

```
COPY customer_orders_lineitem FROM 's3://redshift-downloads/semistructured/tpch-nested/
data/json/customer_orders_lineitem'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT JSON 'auto';

SELECT c_orders[0].o_orderkey FROM customer_orders_lineitem;
```

Amazon Redshift는 또한 테이블 별칭을 표기법에 대한 접두사로 사용합니다. 다음 예는 이전 예와 동일한 쿼리입니다.

```
SELECT cust.c_orders[0].o_orderkey FROM customer_orders_lineitem AS cust;
```

필터링, 조인 및 집계와 같은 모든 형식의 쿼리에 점 및 대괄호 표기법을 사용할 수 있습니다. 일반적으로 열 참조가 있는 쿼리에서 이러한 표기법을 사용할 수 있습니다. 다음 예에서는 결과를 필터링하는 SELECT 문을 사용합니다.

```
SELECT count(*) FROM customer_orders_lineitem WHERE c_orders[0]. o_orderkey IS NOT
 NULL;
```

다음 예에서는 GROUP BY 및 ORDER BY 절에 괄호와 점 탐색을 사용합니다.

```
SELECT c_orders[0].o_orderdate,
       c_orders[0].o_orderstatus,
       count(*)
FROM customer_orders_lineitem
WHERE c_orders[0].o_orderkey IS NOT NULL
GROUP BY c_orders[0].o_orderstatus,
         c_orders[0].o_orderdate
ORDER BY c_orders[0].o_orderdate;
```

쿼리 중첩 해제

쿼리 중첩을 해제하기 위해 Amazon Redshift는 PartiQL 구문을 사용하여 SUPER 배열을 반복합니다. 쿼리의 FROM 절로 배열을 탐색하여 이를 수행합니다. 다음 예에서는 이전 예를 사용하여 c_orders에 대한 속성 값을 반복합니다.

```
SELECT c.*, o FROM customer_orders_lineitem c, c.c_orders o;
```

중첩 해제 구문은 FROM 절의 확장입니다. 표준 SQL에서 FROM 절 x （AS） y는 y가 x 관계에 있는 각 튜플을 반복함을 의미합니다. 이 경우 x는 관계를 나타내고, y는 관계 x에 대한 별칭을 나타냅니다. 마찬가지로, FROM 절 항목 x （AS） y를 사용하여 중첩 해제하는 PartiQL 구문은 y가 (SUPER) 배열 표현식 x의 각 (SUPER) 값을 반복함을 의미합니다. 이 경우 x는 SUPER 표현식이고, y는 x에 대한 별칭입니다.

왼쪽 피연산자는 일반 탐색을 위해 점 및 대괄호 표기법을 사용할 수도 있습니다. 앞의 예에서 `customer_orders_lineitem c`는 `customer_order_lineitem` 기본 테이블에 대한 반복이고 `c.c_orders o`는 `c.c_orders` 배열에 대한 반복입니다. 배열 내의 배열인 `o_lineitems` 속성을 반복하려면 여러 절을 추가해야 합니다.

```
SELECT c.*, o, l FROM customer_orders_lineitem c, c.c_orders o, o.o_lineitems l;
```

Amazon Redshift는 AT 키워드를 사용하여 배열을 반복할 때 배열 인덱스도 지원합니다. 절 x AS y AT z는 배열 x를 반복하고 배열 인덱스인 필드 z,를 생성합니다. 다음 예에서는 배열 인덱스의 작동 방식을 보여줍니다.

```
SELECT c_name,
       orders.o_orderkey AS orderkey,
       index AS orderkey_index
FROM customer_orders_lineitem c, c.c_orders AS orders AT index
ORDER BY orderkey_index;

c_name             | orderkey | orderkey_index
-------------------+----------+----------------
Customer#000008251 | 3020007  |        0
Customer#000009452 | 4043971  |        0
  (2 rows)
```

다음 예에서는 스칼라 배열을 반복합니다.

```
CREATE TABLE bar AS SELECT json_parse('{"scalar_array": [1, 2.3, 45000000]}') AS data;

SELECT index, element FROM bar AS b, b.data.scalar_array AS element AT index;

 index | element
-------+----------
     0 | 1
     1 | 2.3
     2 | 45000000
```

```
(3 rows)
```

다음 예에서는 여러 수준의 배열을 반복합니다. 이 예제에서는 여러 unnest 절을 사용하여 가장 안쪽 배열로 반복합니다. `f.multi_level_array` AS 배열은 `multi_level_array`를 반복합니다. 배열 AS 요소는 `multi_level_array` 내의 배열에 대한 반복입니다.

```
CREATE TABLE foo AS SELECT json_parse('[[1.1, 1.2], [2.1, 2.2], [3.1, 3.2]]') AS
 multi_level_array;

SELECT array, element FROM foo AS f, f.multi_level_array AS array, array AS element;

   array   | element
-----------+---------
 [1.1,1.2] | 1.1
 [1.1,1.2] | 1.2
 [2.1,2.2] | 2.1
 [2.1,2.2] | 2.2
 [3.1,3.2] | 3.1
 [3.1,3.2] | 3.2
(6 rows)
```

FROM 절에 대한 자세한 내용은 FROM 절 섹션을 참조하세요.

객체 피벗 해제

객체 피벗 해제를 수행하기 위해 Amazon Redshift는 PartiQL 구문을 사용하여 SUPER 객체를 반복합니다. UNPIVOT 키워드가 있는 쿼리의 FROM 절을 사용하여 이 작업을 수행합니다. 이 경우 표현식은 `c.c_orders[0]` 객체입니다. 예제 쿼리는 객체가 반환한 각 속성을 반복합니다.

```
SELECT attr as attribute_name, json_typeof(val) as value_type
FROM customer_orders_lineitem c, UNPIVOT c.c_orders[0] AS val AT attr
WHERE c_custkey = 9451;

 attribute_name  | value_type
-----------------+------------
 o_orderstatus   | string
 o_clerk         | string
 o_lineitems     | array
 o_orderdate     | string
 o_shippriority  | number
 o_totalprice    | number
 o_orderkey      | number
```

```
 o_comment       | string
 o_orderpriority | string
(9 rows)
```

중첩 해제와 마찬가지로 피벗 해제 구문도 FROM 절의 확장입니다. 차이점은 피벗 해제 구문이 UNPIVOT 키워드를 사용하여 배열 대신 객체를 반복하고 있음을 나타냅니다. 객체 내부의 모든 값에 대한 반복에는 AS `value_alias`를 사용하고 모든 속성에 대한 반복에는 AT `attribute_alias`를 사용합니다. 다음 구문 조각을 고려합니다.

```
UNPIVOT expression AS value_alias [ AT attribute_alias ]
```

Amazon Redshift는 다음과 같이 단일 FROM 절에서 객체 unpivoting 및 array unnesting 사용을 지원합니다.

```
SELECT attr as attribute_name, val as object_value
FROM customer_orders_lineitem c, c.c_orders AS o, UNPIVOT o AS val AT attr
WHERE c_custkey = 9451;
```

객체 피벗 해제를 사용하는 경우 Amazon Redshift는 상관 관계가 있는 피벗 해제를 지원하지 않습니다. 특히, 서로 다른 쿼리 수준에서 피벗 해제 예가 여러 개 있고 내부 피벗 해제가 외부 쿼리 수준을 참조하는 경우가 있다고 가정합니다. Amazon Redshift는 이러한 유형의 다중 피벗 해제를 지원하지 않습니다.

FROM 절에 대한 자세한 내용은 FROM 절 섹션을 참조하세요. PIVOT 및 UNPIVOT을 사용하여 구조화된 데이터를 쿼리하는 방법을 보여주는 예는 PIVOT 및 UNPIVOT 예 섹션을 참조하세요.

동적 형식 지정

동적 형식 지정에는 점 및 대괄호 경로에서 추출된 데이터의 명시적 캐스팅이 필요하지 않습니다. Amazon Redshift는 동적 형식 지정을 사용하여 쿼리에 사용하기 전에 데이터 형식을 선언할 필요 없이 스키마 없는 SUPER 데이터를 처리합니다. 동적 형식 지정은 Amazon Redshift 형식으로 명시적으로 캐스팅하지 않고도 SUPER 데이터 열로 이동한 결과를 사용합니다. 동적 형식 지정은 조인과 GROUP BY 절에서 가장 유용합니다. 다음 예에서는 일반적인 Amazon Redshift 형식으로 점 및 대괄호 표현식을 명시적으로 캐스팅할 필요가 없는 SELECT 문을 사용합니다. 형식 호환성 및 변환에 대한 자세한 내용은 형식 호환성 및 변환 섹션을 참조하세요.

```
SELECT c_orders[0].o_orderkey
FROM customer_orders_lineitem
```

```
WHERE c_orders[0].o_orderstatus = 'P';
```

이 쿼리의 등호는 c_orders[0].o_orderstatus가 문자열 'P'일 때 `true`로 평가됩니다. 등식의 인수가 다른 형식인 경우를 포함한 다른 모든 경우에는 등호가 `false`로 평가됩니다.

동적 및 정적 형식 지정

동적 형식 지정을 사용하지 않으면 c_orders[0].o_orderstatus가 문자열, 정수 또는 구조인지 확인할 수 없습니다. c_orders[0].o_orderstatus가 Amazon Redshift 스칼라, 배열 또는 구조가 될 수 있는 SUPER 데이터 형식인지만 결정할 수 있습니다. c_orders[0].o_orderstatus의 정적 형식은 SUPER 데이터 형식입니다. 일반적으로 형식은 SQL에서 암시적으로 정적 형식입니다.

Amazon Redshift는 스키마 없는 데이터 처리에 동적 형식 지정을 사용합니다. 쿼리가 데이터를 평가할 때 c_orders[0].o_orderstatus가 특정 형식으로 판명됩니다. 예를 들어 customer_orders_lineitem의 첫 번째 레코드에서 c_orders[0].o_orderstatus를 평가하면 정수가 될 수 있습니다. 두 번째 레코드를 평가하면 문자열이 될 수 있습니다. 다음은 표현식의 동적 형식입니다.

동적 형식이 있는 점 및 대괄호 표현식과 함께 SQL 연산자 또는 함수를 사용하는 경우 Amazon Redshift는 각 정적 형식과 함께 표준 SQL 연산자 또는 함수를 사용하는 것과 유사한 결과를 생성합니다. 이 예에서 경로 표현식의 동적 형식이 문자열인 경우 문자열 'P'와의 비교는 의미가 있습니다. c_orders[0].o_orderstatus의 동적 형식이 문자열이 아닌 다른 데이터 형식일 때마다 등식은 false를 반환합니다. 잘못된 형식의 인수가 사용되면 다른 함수는 null을 반환합니다.

다음 예에서는 정적 형식 지정을 사용하여 이전 쿼리를 작성합니다.

```
SELECT c_custkey
FROM customer_orders_lineitem
WHERE CASE WHEN JSON_TYPEOF(c_orders[0].o_orderstatus) = 'string'
           THEN c_orders[0].o_orderstatus::VARCHAR = 'P'
           ELSE FALSE END;
```

동등 술어와 비교 술어의 다음과 같은 차이에 유의합니다. 이전 예에서 동등 술어를 작거나 같음 술어로 바꾸면 의미 체계가 false 대신 null을 생성합니다.

```
SELECT c_orders[0]. o_orderkey
FROM customer_orders_lineitem
WHERE c_orders[0].o_orderstatus <= 'P';
```

이 예에서 c_orders[0].o_orderstatus가 문자열이고 알파벳순으로 'P'보다 작거나 같으면 Amazon Redshift는 true를 반환합니다. 알파벳순으로 'P'보다 크면 Amazon Redshift는 false를 반환합니다. 그

러나 c_orders[0].o_orderstatus가 문자열이 아닌 경우 Amazon Redshift는 다음 쿼리와 같이 다른 형식의 값을 비교할 수 없으므로 null을 반환합니다.

```
SELECT c_custkey
FROM customer_orders_lineitem
WHERE CASE WHEN JSON_TYPEOF(c_orders[0].o_orderstatus) = 'string'
           THEN c_orders[0].o_orderstatus::VARCHAR <= 'P'
           ELSE NULL END;
```

동적 형식 지정은 최소한으로 비교 가능한 형식의 비교에서 제외되지 않습니다. 예를 들어 CHAR 및 VARCHAR Amazon Redshift 스칼라 형식을 모두 SUPER로 변환할 수 있습니다. 이들은 Amazon Redshift CHAR 및 VARCHAR 형식과 유사한 후행 공백 문자를 무시하는 것을 포함하여 문자열과 유사합니다. 마찬가지로 정수, 소수 및 부동 소수점 값은 SUPER 값으로 비교 가능합니다. 특히 소수 열의 경우 값마다 소수 자릿수가 다를 수도 있습니다. Amazon Redshift는 여전히 이들을 동적 형식으로 간주합니다.

Amazon Redshift는 또한 객체 또는 배열을 심층적으로 평가하고 모든 속성을 비교하는 것과 같이 깊은 동등(deep equal)으로 평가되는 객체 및 배열에 대한 동등성을 지원합니다. 깊은 동등(deep equal)은 수행하는 데 시간이 많이 걸릴 수 있으므로 주의해서 사용합니다.

조인에 동적 형식 지정 사용

조인의 경우 동적 형식 지정은 나타날 수 있는 데이터 형식을 찾기 위해 긴 CASE WHEN 분석을 수행하지 않고 값을 다른 동적 형식과 일치시킵니다. 예를 들어 시간이 지남에 따라 조직에서 부분 키에 사용하던 형식을 변경했다고 가정합니다.

발행된 초기 정수 부분 키는 'A55'와 같은 문자열 부분 키로 대체되고 나중에 문자열과 숫자를 결합하는 ['X', 10]과 같은 배열 부분 키로 대체됩니다. Amazon Redshift는 부분 키에 대해 긴 사례 분석을 수행할 필요가 없으며 다음 예와 같이 조인을 사용할 수 있습니다.

```
SELECT c.c_name
    ,l.l_extendedprice
    ,l.l_discount
FROM customer_orders_lineitem c
    ,c.c_orders o
    ,o.o_lineitems l
    ,supplier_partsupp s
    ,s.s_partsupps ps
WHERE l.l_partkey = ps.ps_partkey
AND c.c_nationkey = s.s_nationkey
ORDER BY c.c_name;
```

다음 예에서는 동적 형식 지정을 사용하지 않으면 동일한 쿼리가 얼마나 복잡하고 비효율적일 수 있는지 보여줍니다.

```
SELECT c.c_name
    ,l.l_extendedprice
    ,l.l_discount
FROM customer_orders_lineitem c
    ,c.c_orders o
    ,o.o_lineitems l
    ,supplier_partsupp s
    ,s.s_partsupps ps
WHERE CASE WHEN IS_INTEGER(l.l_partkey) AND IS_INTEGER(ps.ps_partkey)
           THEN l.l_partkey::integer = ps.ps_partkey::integer
           WHEN IS_VARCHAR(l.l_partkey) AND IS_VARCHAR(ps.ps_partkey)
           THEN l.l_partkey::varchar = ps.ps_partkey::varchar
           WHEN IS_ARRAY(l.l_partkey) AND IS_ARRAY(ps.ps_partkey)
                AND IS_VARCHAR(l.l_partkey[0]) AND IS_VARCHAR(ps.ps_partkey[0])
                AND IS_INTEGER(l.l_partkey[1]) AND IS_INTEGER(ps.ps_partkey[1])
           THEN l.l_partkey[0]::varchar = ps.ps_partkey[0]::varchar
                AND l.l_partkey[1]::integer = ps.ps_partkey[1]::integer
           ELSE FALSE END
AND c.c_nationkey = s.s_nationkey
ORDER BY c.c_name;
```

Lax 의미 체계

기본적으로 SUPER 값에 대한 탐색 작업은 탐색이 유효하지 않을 때 오류를 반환하는 대신 null을 반환합니다. SUPER 값이 객체가 아니거나 SUPER 값이 객체이지만 쿼리에 사용된 속성 이름을 포함하지 않는 경우 객체 탐색이 유효하지 않습니다. 예를 들어 다음 쿼리는 SUPER 데이터 열 cdata에서 잘못된 속성 이름에 액세스합니다.

```
SELECT c.c_orders.something FROM customer_orders_lineitem c;
```

SUPER 값이 배열이 아니거나 배열 인덱스가 범위를 벗어난 경우 배열 탐색은 null을 반환합니다. 다음 쿼리는 c_orders[1][1]이 범위를 벗어났기 때문에 null을 반환합니다.

```
SELECT c.c_orders[1][1] FROM customer_orders_lineitem c;
```

Lax 의미 체계는 동적 형식 지정을 사용하여 SUPER 값을 캐스팅할 때 특히 유용합니다. SUPER 값을 잘못된 형식으로 캐스팅하면 캐스트가 유효하지 않은 경우 오류 대신 null이 반환됩니다. 예를 들어 다음 쿼리는 객체 속성 o_orderstatus의 문자열 값 'Good'을 INTEGER로 캐스팅할 수 없기 때문에 null을

반환합니다. Amazon Redshift는 VARCHAR에서 INTEGER로 캐스트에 대해서는 오류를 반환하지만 SUPER 캐스트에 대해서는 오류를 반환하지 않습니다.

```
SELECT c.c_orders.o_orderstatus::integer FROM customer_orders_lineitem c;
```

내부 검사 유형

SUPER 데이터 열은 SUPER 값에 대한 동적 형식과 기타 형식 정보를 반환하는 내부 검사 함수를 지원합니다. 가장 일반적인 예는 SUPER 값의 동적 형식에 따라 부울, 숫자, 문자열, 객체, 배열 또는 null 값이 있는 VARCHAR를 반환하는 JSON_TYPEOF 스칼라 함수입니다. Amazon Redshift는 SUPER 데이터 열에 대해 다음과 같은 부울 함수를 지원합니다.

- DECIMAL_PRECISION
- DECIMAL_SCALE
- IS_ARRAY
- IS_BIGINT
- IS_CHAR
- IS_DECIMAL
- IS_FLOAT
- IS_INTEGER
- IS_OBJECT
- IS_SCALAR
- IS_SMALLINT
- IS_VARCHAR
- JSON_TYPEOF

입력 값이 null인 경우 이러한 함수는 모두 false를 반환합니다. IS_SCALAR, IS_OBJECT 및 IS_ARRAY는 상호 배타적이며 null을 제외한 모든 가능한 값을 포함합니다.

데이터에 해당하는 형식을 추론하기 위해 Amazon Redshift는 다음 예와 같이 SUPER 값의 형식(최상위 수준)을 반환하는 JSON_TYPEOF 함수를 사용합니다.

```
SELECT JSON_TYPEOF(r_nations) FROM region_nations;
 json_typeof
 -------------
```

```
 array
(1 row)
```

```
SELECT JSON_TYPEOF(r_nations[0].n_nationkey) FROM region_nations;
 json_typeof
 -------------
 number
```

Amazon Redshift는 이를 SUPER 대신 VARCHAR 열에 삽입하는 것과 유사한 단일 긴 문자열로 간주합니다. 열이 SUPER이므로 단일 문자열은 여전히 유효한 SUPER 값이고 차이점은 JSON_TYPEOF에 표시됩니다.

```
SELECT IS_VARCHAR(r_nations[0].n_name) FROM region_nations;
 is_varchar
-------------
 true
(1 row)
```

```
SELECT r_nations[4].n_name FROM region_nations
WHERE CASE WHEN IS_INTEGER(r_nations[4].n_nationkey)
           THEN r_nations[4].n_nationkey::INTEGER = 15
           ELSE false END;
```

순서 기준

Amazon Redshift는 동적 형식이 다른 값 간의 SUPER 비교를 정의하지 않습니다. 문자열인 SUPER 값은 숫자인 SUPER 값보다 작지도 크지도 않습니다. SUPER 열과 함께 ORDER BY 절을 사용하기 위해 Amazon Redshift는 ORDER BY 절을 사용하여 SUPER 값의 순위를 지정할 때 관찰할 다양한 형식 간의 총 순서를 정의합니다. 동적 형식 중 순서는 부울, 숫자, 문자열, 배열, 객체입니다. 다음 예에서는 서로 다른 형식의 순서를 보여줍니다.

```
INSERT INTO region_nations VALUES
(100,'name1','comment1','AWS'),
(200,'name2','comment2',1),
(300,'name3','comment3',ARRAY(1, 'abc', null)),
(400,'name4','comment4',-2.5),
(500,'name5','comment5','Amazon');

SELECT r_nations FROM region_nations order by r_nations;
```

```
r_nations
---------------
 -2.5
 1
 "Amazon"
 "AWS"
 [1,"abc",null]
(5 rows)
```

ORDER BY 절에 대한 자세한 내용은 ORDER BY 절 섹션을 참조하세요.

연산자 및 함수

Amazon Redshift를 사용하면 연산자와 함수를 사용하여 SUPER 데이터로 대규모 데이터세트에 대한 고급 분석을 수행할 수 있습니다. SUPER 데이터의 연산자 및 함수는 Amazon Redshift 테이블에 저장된 반정형 데이터의 복잡한 분석 및 조작을 지원하는 SQL 구문입니다.

다음 섹션에서는 Amazon Redshift의 SUPER 데이터에 연산자와 함수를 사용하여 반정형 데이터의 잠재력을 최대한 활용하기 위한 구문, 예제 및 모범 사례를 다룹니다.

산술 연산자

SUPER 값은 동적 형식 지정을 사용하여 모든 기본 산술 연산자 +, -, *, /, %를 지원합니다. 작업의 결과 형식은 SUPER로 유지됩니다. 이항 연산자 +를 제외한 모든 연산자의 경우 입력 피연산자는 숫자여야 합니다. 그렇지 않으면 Amazon Redshift가 null을 반환합니다. 소수 자릿수와 부동 소수점 값의 차이는 Amazon Redshift에서 이러한 연산자를 실행하고 동적 형식이 변경되지 않는 경우에도 유지됩니다. 그러나 곱셈과 나눗셈을 사용하면 소수 자릿수가 변경됩니다. 산술 오버플로는 여전히 쿼리 오류를 발생시키며 null로 변경되지 않습니다. 이항 연산자 +는 입력이 숫자이면 덧셈을 수행하고 입력이 문자열이면 연결을 수행합니다. 한 피연산자가 문자열이고 다른 피연산자가 숫자이면 결과는 null입니다. 단항 접두사 연산자 + 및 -는 다음 예와 같이 SUPER 값이 숫자가 아닌 경우 null을 반환합니다.

```
SELECT (c_orders[0]. o_orderkey + 0.5) * c_orders[0]. o_orderkey / 10 AS math FROM
 customer_orders_lineitem;
            math
----------------------------
 1757958232200.1500
(1 row)
```

동적 형식 지정을 사용하면 SUPER의 소수 값이 다른 소수 자릿수를 가질 수 있습니다. Amazon Redshift는 소수 값을 서로 다른 정적 형식인 것처럼 취급하고 모든 수학 연산을 허용합니다. Amazon

Redshift는 피연산자의 소수 자릿수를 기준으로 결과 소수 자릿수를 동적으로 계산합니다. 피연산자 중 하나가 부동 소수점 숫자인 경우 Amazon Redshift는 다른 피연산자를 부동 소수점 숫자로 승격시키고 결과를 부동 소수점 숫자로 생성합니다.

산술 함수

Amazon Redshift는 SUPER 열에 대해 다음과 같은 산술 함수를 지원합니다. 입력이 숫자가 아닌 경우 null이 반환됩니다.

- FLOOR. 자세한 내용은 FLOOR 함수 섹션을 참조하세요.
- CEIL 및 CEILING. 자세한 내용은 CEILING(또는 CEIL) 함수 섹션을 참조하세요.
- ROUND. 자세한 내용은 ROUND 함수 섹션을 참조하세요.
- TRUNC. 자세한 내용은 TRUNC 함수 섹션을 참조하세요.
- ABS. 자세한 내용은 ABS 함수 섹션을 참조하세요.

다음 예에서는 산술 함수를 사용하여 데이터를 쿼리합니다.

```
SELECT x, FLOOR(x), CEIL(x), ROUND(x)
FROM (
    SELECT (c_orders[0]. o_orderkey + 0.5) * c_orders[0].o_orderkey / 10 AS x
    FROM customer_orders_lineitem
    );

         x          |     floor     |     ceil      |     round
--------------------+---------------+---------------+---------------
 1389636795898.0500  | 1389636795898  | 1389636795899  | 1389636795898
```

ABS 함수는 FLOOR, CEIL 동안 입력 소수의 소수 자릿수를 유지합니다. ROUND는 입력 소수의 소수 자릿수를 제거합니다.

배열 함수

Amazon Redshift는 다음과 같은 배열 구성과 유틸리티 함수 array, array_concat, subarray, array_flatten, get_array_length 및 split_to_array를 지원합니다.

ARRAY 함수로 다른 SUPER 값을 포함한 Amazon Redshift 데이터 형식의 값에서 SUPER 배열을 구성할 수 있습니다. 다음 예에서는 가변 함수 ARRAY를 사용합니다.

```
SELECT ARRAY(1, c.c_custkey, NULL, c.c_name, 'abc') FROM customer_orders_lineitem c;
```

```
                                array
 ------------------------------------------------------
[1,8401,null,""Customer#000008401"",""abc""]
[1,9452,null,""Customer#000009452"",""abc""]
[1,9451,null,""Customer#000009451"",""abc""]
[1,8251,null,""Customer#000008251"",""abc""]
[1,5851,null,""Customer#000005851"",""abc""]
(5 rows)
```

다음 예에서는 ARRAY_CONCAT 함수와 함께 배열 연결을 사용합니다.

```
SELECT ARRAY_CONCAT(JSON_PARSE('[10001,10002]'),JSON_PARSE('[10003,10004]'));

             array_concat
-------------------------------------
 [10001,10002,10003,10004]
(1 row)
```

다음 예에서는 입력 배열의 하위 집합을 반환하는 SUBARRAY 함수와 함께 배열 조작을 사용합니다.

```
SELECT SUBARRAY(ARRAY('a', 'b', 'c', 'd', 'e', 'f'), 2, 3);

   subarray
---------------
 ["c","d","e"]
(1 row))
```

다음 예에서는 ARRAY_FLATTEN을 사용하여 여러 수준의 배열을 단일 배열로 병합합니다.

```
SELECT x, ARRAY_FLATTEN(x) FROM (SELECT ARRAY(1, ARRAY(2, ARRAY(3, ARRAY())))  AS x);

     x           | array_flatten
 ----------------+---------------
 [1,[2,[3,[]]]]  | [1,2,3]
(1 row)
```

배열 함수 ARRAY_CONCAT 및 ARRAY_FLATTEN은 동적 형식 지정 규칙을 사용합니다. 이들은 입력이 배열이 아닌 경우 오류 대신 null을 반환합니다. GET_ARRAY_LENGTH 함수는 객체 또는 배열 경로가 지정된 SUPER 배열의 길이를 반환합니다.

```
SELECT c_name
```

```
FROM customer_orders_lineitem
WHERE GET_ARRAY_LENGTH(c_orders) = (
    SELECT MAX(GET_ARRAY_LENGTH(c_orders))
    FROM customer_orders_lineitem
    );
```

다음 예에서는 SPLIT_TO_ARRAY를 사용하여 문자열을 문자열 배열로 분할합니다. 이 함수는 구분 기호를 옵션 파라미터로 사용합니다. 구분 기호가 없으면 기본값은 쉼표입니다.

```
SELECT SPLIT_TO_ARRAY('12|345|6789', '|');

   split_to_array
---------------------
 ["12","345","6789"]
(1 row)
```

SUPER 구성

특정 시나리오에 맞게 SUPER 데이터를 구성할 수 있습니다. 다음 섹션에서는 데이터 형식 요구 사항에 따라 적절한 SUPER 구성을 선택하고 적용하는 방법에 대한 세부 정보를 제공합니다.

주제

- SUPER의 lax 및 strict 모드
- 대문자 및 대소문자 혼합 필드 이름 또는 속성이 포함된 JSON 필드에 액세스
- SUPER에 대한 구문 분석 옵션

SUPER의 lax 및 strict 모드

SUPER 데이터를 쿼리할 때 경로 표현식이 실제 SUPER 데이터 구조와 일치하지 않을 수 있습니다. 존재하지 않는 객체 멤버 또는 배열 요소에 액세스하려고 할 때 쿼리가 기본 lax 모드에서 실행되는 경우 Amazon Redshift가 NULL 값을 반환합니다. strict 모드에서 쿼리를 실행하면 Amazon Redshift가 오류를 반환합니다. 다음 세션 파라미터를 설정하여 lax 모드를 설정하거나 해제할 수 있습니다.

다음 예에서는 세션 파라미터를 사용하여 lax 모드를 사용합니다.

```
SET navigate_super_null_on_error=ON;  --default lax mode for navigation

SET cast_super_null_on_error=ON;  --default lax mode for casting
```

```
SET parse_super_null_on_error=OFF;  --default strict mode for ingestion
```

대문자 및 대소문자 혼합 필드 이름 또는 속성이 포함된 JSON 필드에 액세스

JSON 속성 이름이 대문자 또는 대소문자 혼합으로 되어 있는 경우 SUPER 유형 구조를 대소문자를 구분하는 방식으로 탐색할 수 있어야 합니다. 이렇게 하려면 enable_case_sensitive_identifier를 TRUE로 구성하고 대문자 및 대소문자 혼합 속성 이름을 큰따옴표로 묶습니다. enable_case_sensitive_super_attribute를 TRUE로 구성할 수도 있습니다. 이 경우 쿼리에 대문자 및 대소문자 혼합 속성 이름을 큰따옴표로 묶지 않고 사용할 수 있습니다.

다음 예에서는 데이터를 쿼리하도록 enable_case_sensitive_identifier를 설정하는 방법을 설명합니다.

```
SET enable_case_sensitive_identifier to TRUE;

-- Accessing JSON attribute names with uppercase and mixedcase names
SELECT json_table.data."ITEMS"."Name",
       json_table.data."price"
FROM
  (SELECT json_parse('{"ITEMS":{"Name":"TV"}, "price": 345}') AS data) AS json_table;

 Name | price
------+-------
 "TV" | 345
(1 row)

RESET enable_case_sensitive_identifier;

-- After resetting the above configuration, the following query accessing JSON
 attribute names with uppercase and mixedcase names should return null (if in lax
 mode).
SELECT json_table.data."ITEMS"."Name",
       json_table.data."price"
FROM
  (SELECT json_parse('{"ITEMS":{"Name":"TV"}, "price": 345}') AS data) AS json_table;

 name | price
------+-------
      | 345
(1 row)
```

다음 예에서는 데이터를 쿼리하도록 enable_case_sensitive_super_attribute를 설정하는 방법을 설명합니다.

```
SET enable_case_sensitive_super_attribute to TRUE;
 -- Accessing JSON attribute names with uppercase and mixedcase names

SELECT json_table.data.ITEMS.Name,
       json_table.data.price
FROM
  (SELECT json_parse('{"ITEMS":{"Name":"TV"}, "price": 345}') AS data) AS json_table;

 name | price
 -----+-------
 "TV" | 345
(1 row)

 RESET enable_case_sensitive_super_attribute;

 -- After resetting enable_case_sensitive_super_attribute, the query now returns NULL
 for ITEMS.Name (if in lax mode).

SELECT json_table.data.ITEMS.Name,
       json_table.data.price
FROM
  (SELECT json_parse('{"ITEMS":{"Name":"TV"}, "price": 345}') AS data) AS json_table;

 name | price
 -----+-------
      | 345
(1 row)
```

SUPER에 대한 구문 분석 옵션

JSON_PARSE 함수를 사용하여 JSON 문자열을 SUPER 값으로 구문 분석하는 경우 특정 제한이 적용됩니다.

- 동일한 속성 이름이 동일한 객체에 나타날 수는 없지만 중첩된 객체에는 나타날 수 있습니다. json_parse_dedup_attributes 구성 옵션을 사용하면 JSON_PARSE가 오류를 반환하는 대신 마지막으로 발생한 중복 속성만 유지할 수 있습니다.
- 문자열 값은 시스템 최대 varchar 크기인 65535바이트를 초과할 수 없습니다. json_parse_truncate_strings 구성 옵션을 사용하면 JSON_PARSE()가 오류를 반환하지 않

고 이 제한보다 긴 문자열을 자동으로 자를 수 있습니다. 이 동작은 문자열 값에만 영향을 주며 속성 이름에는 영향을 주지 않습니다.

JSON_PARSE 함수에 대한 자세한 내용은 JSON_PARSE 함수 섹션을 참조하세요.

다음 예에서는 json_parse_dedup_attributes 구성 옵션을 중복 속성에 대한 오류를 반환하는 기본 동작으로 설정하는 방법을 보여줍니다.

```
SET json_parse_dedup_attributes=OFF;  --default behavior of returning error instead of
 de-duplicating attributes
```

다음 예에서는 json_parse_truncate_strings 구성 옵션을 이 제한보다 긴 문자열에 대해 오류를 반환하는 기본 동작으로 설정하는 방법을 보여줍니다.

```
SET json_parse_truncate_strings=OFF;  --default behavior of returning error instead of
 truncating strings
```

제한 사항

Amazon Redshift를 사용하면 SUPER 데이터 유형을 사용하여 JSON, Avro 또는 Ion 등의 반정형 데이터를 저장 및 쿼리할 수 있습니다. SUPER 데이터 유형 제한 사항에서는 Amazon Redshift에서 이 데이터 유형을 사용할 때의 제약 조건과 한계를 알려줍니다. 다음 섹션에서는 반정형 데이터 내에서 지원되는 최대 크기, 중첩 수준 및 데이터 유형 등 SUPER 데이터 유형의 특정 제한 사항에 대한 세부 정보를 제공합니다.

- SUPER 열을 배포 또는 정렬 키로 정의할 수 없습니다.
- 개별 SUPER 객체는 최대 16MB의 데이터를 저장할 수 있습니다.
- SUPER 객체 내의 개별 값은 해당 Amazon Redshift 유형의 최대 길이로 제한됩니다. 예를 들어 SUPER에 로드되는 단일 문자열 값은 최대 VARCHAR 길이 65,535바이트로 제한됩니다.
- SUPER 열에서는 부분 업데이트 또는 변환 작업을 수행할 수 없습니다.
- 오른쪽 조인 또는 전체 외부 조인에서는 SUPER 데이터 형식과 별칭을 사용할 수 없습니다.
- SUPER 데이터 형식은 XML을 인바운드 또는 아웃바운드 직렬화 형식으로 지원하지 않습니다.
- 중첩 해제를 위해 테이블 변수를 참조하는 하위 쿼리(상관 여부에 관계 없음)의 FROM 절에서 쿼리는 상위 테이블만 참조할 수 있고 다른 테이블은 참조할 수 없습니다.
- 캐스팅 제한

SUPER 값은 다음 예외를 제외하고 다른 데이터 형식으로 또는 다른 데이터 형식에서 캐스팅할 수 있습니다.

- Amazon Redshift는 정수와 소수 자릿수가 0인 소수를 구분하지 않습니다.
- 소수 자릿수가 0이 아닌 경우 SUPER 데이터 형식은 다음 예와 같이 Amazon Redshift가 SUPER 관련 오류를 null로 변환한다는 점을 제외하고 다른 Amazon Redshift 데이터 형식과 동일하게 동작합니다.

```
SELECT 5::bool;
 bool
-------
 True
(1 row)

SELECT 5::decimal::bool;
ERROR:  cannot cast type numeric to boolean

SELECT 5::super::bool;
 bool
-------
 True
(1 row)

SELECT 5.0::bool;
ERROR:  cannot cast type numeric to boolean

SELECT 5.0::super::bool;
 bool
-------
(1 row)
```

- Amazon Redshift는 날짜 및 시간 형식을 SUPER 데이터 형식으로 캐스팅하지 않습니다. Amazon Redshift는 다음 예와 같이 SUPER 데이터 형식에서 날짜 및 시간 데이터 형식만 캐스팅할 수 있습니다.

```
SELECT o.o_orderdate FROM customer_orders_lineitem c,c.c_orders o;
  order_date
------------------
 "2001-09-08"
(1 row)
```

```
SELECT JSON_TYPEOF(o.o_orderdate) FROM customer_orders_lineitem c,c.c_orders o;
 json_typeof
-----------------
 string
(1 row)


SELECT o.o_orderdate::date FROM customer_orders_lineitem c,c.c_orders o;
 order_date
----------------
 2001-09-08
(1 row)


--date/time cannot be cast to super
SELECT '2019-09-09'::date::super;
ERROR:  cannot cast type date to super
```

- 스칼라가 아닌 값(객체 및 배열)에서 문자열로 캐스팅하면 NULL이 반환됩니다. 이러한 스칼라가 아닌 값을 올바르게 직렬화하려면 캐스팅하지 않습니다. 대신 `json_serialize`를 사용하여 스칼라가 아닌 값을 캐스팅합니다. `json_serialize` 함수는 varchar를 반환합니다. 일반적으로 다음 첫 번째 예와 같이 Amazon Redshift가 암시적으로 직렬화하므로 스칼라가 아닌 값을 varchar로 캐스팅할 필요가 없습니다.

```
SELECT r_nations FROM region_nations WHERE r_regionkey=300;
   r_nations
----------------
 [1,"abc",null]
(1 row)

SELECT r_nations::varchar FROM region_nations WHERE r_regionkey=300;
 r_nations
-------------
(1 row)

SELECT JSON_SERIALIZE(r_nations) FROM region_nations WHERE r_regionkey=300;
 json_serialize
-----------------
 [1,"abc",null]
(1 row)
```

- 대/소문자를 구분하지 않는 데이터베이스의 경우 Amazon Redshift는 SUPER 데이터 형식을 지원하지 않습니다. 대/소문자를 구분하지 않는 열의 경우 Amazon Redshift는 열을 SUPER 형식으로

캐스팅하지 않습니다. 따라서 Amazon Redshift는 캐스팅을 트리거하는 대/소문자를 구분하지 않는 열과 상호 작용하는 SUPER 열을 지원하지 않습니다.

- Amazon Redshift는 하위 쿼리로 IN 함수의 외부 테이블 또는 왼쪽(LHS)을 중첩 해제하는 이러한 하위 쿼리에서 RANDOM( ) 또는 TIMEOFDAY( )와 같은 휘발성 함수를 지원하지 않습니다.

SUPER 데이터 형식과 구체화된 뷰

Amazon Redshift를 사용하면 SUPER 데이터 유형을 사용하여 구체화된 뷰의 성능과 유연성을 향상시킬 수 있습니다. SUPER 데이터 유형으로 기본 테이블의 열 슈퍼 세트를 구체화된 뷰에 저장할 수 있으므로 기본 테이블을 조인하지 않고도 구체화된 뷰를 직접 쿼리하는 것이 가능합니다. 다음 섹션에서는 Amazon Redshift에서 SUPER 데이터 유형을 사용하여 구체화된 뷰를 생성하고 사용하는 방법을 보여줍니다.

Amazon Redshift는 구체화된 뷰에서 SUPER 데이터 형식 및 PartiQL과 함께 작동하도록 구체화된 뷰의 기능을 확장합니다. SQL 및 PartiQL 쿼리는 증분 구체화된 뷰를 사용하여 미리 계산할 수 있습니다. 구체화된 뷰에 대한 자세한 내용은 Amazon Redshift의 구체화된 뷰 섹션을 참조하세요.

스키마 없는 비정형 데이터를 SUPER에 저장한 후에는 PartiQL 구체화된 뷰를 사용하여 데이터를 내부 검사하고 구체화된 뷰로 나눌 수 있습니다.

PartiQL 쿼리 가속화

구체화된 뷰를 사용하여 SUPER 열에서 계층적 데이터를 탐색 및/또는 중첩 해제하는 PartiQL 쿼리를 가속화할 수 있습니다. 하나 이상의 구체화된 뷰를 생성하여 SUPER 값을 여러 열로 나누고 Amazon Redshift 분석 쿼리의 열 형식 구조를 활용합니다. 따라서 쿼리는 구체화된 뷰를 사용합니다.

구체화된 뷰는 본질적으로 중첩 데이터를 추출하고 정규화합니다. 정규화 수준은 SUPER 데이터를 기존의 열 형식 데이터로 변환하는 데 얼마나 많은 노력을 기울이는지에 따라 다릅니다.

구체화된 뷰가 있는 SUPER 열로 나누기

Amazon Redshift가 있으면 구체화된 뷰를 사용하여 SUPER 열로 데이터를 파쇄하여 쿼리 성능을 개선하고 스토리지 요구 사항을 줄일 수 있습니다. 파쇄는 반정형 JSON 또는 XML 등 복잡한 데이터 유형을 더 작고 평평한 열로 세분화하는 프로세스를 말합니다. SUPER 열은 파쇄된 데이터를 빠르게 스캔하는 데 최적화된 특수한 형태의 열 스토리지입니다.

다음 섹션에서는 Amazon Redshift에서 구체화된 뷰를 사용하여 SUPER 열로 데이터를 파쇄하는 단계와 고려 사항을 설명합니다.

다음 예에서는 결과 열이 여전히 SUPER 데이터 형식인 중첩 데이터를 나누는 구체화된 뷰를 보여줍니다.

```
SELECT c.c_name, o.o_orderstatus
FROM customer_orders_lineitem c, c.c_orders o;
```

다음 예에서는 나뉜 데이터에서 기존의 Amazon Redshift 스칼라 열을 생성하는 구체화된 뷰를 보여줍니다.

```
SELECT c.c_name, c.c_orders[0].o_totalprice
FROM customer_orders_lineitem c;
```

단일 구체화된 뷰 super_mv를 생성하여 두 쿼리를 모두 가속화할 수 있습니다.

첫 번째 쿼리에 응답하려면 속성 o_orderstatus를 구체화해야 합니다. 중첩 탐색이나 중첩 해제를 포함하지 않기 때문에 속성 c_name을 생략할 수 있습니다. 구체화된 뷰에 customer_orders_lineitem의 속성 c_custkey를 포함해야 구체화된 뷰와 기본 테이블을 조인할 수 있습니다.

두 번째 쿼리에 응답하려면 속성 o_totalprice 및 c_orders의 배열 인덱스 o_idx도 구체화해야 합니다. 따라서 c_orders의 인덱스 0에 액세스할 수 있습니다.

```
CREATE MATERIALIZED VIEW super_mv distkey(c_custkey) sortkey(c_custkey) AS (
  SELECT c_custkey, o.o_orderstatus, o.o_totalprice, o_idx
  FROM customer_orders_lineitem c, c.c_orders o AT o_idx
);
```

구체화된 뷰 super_mv의 속성 o_orderstatus 및 o_totalprice는 SUPER입니다.

구체화된 뷰 super_mv는 기본 테이블 customer_orders_lineitem이 변경되면 점진적으로 새로 고쳐집니다.

```
REFRESH MATERIALIZED VIEW super_mv;
INFO: Materialized view super_mv was incrementally updated successfully.
```

첫 번째 PartiQL 쿼리를 일반 SQL 쿼리로 다시 작성하려면 다음과 같이 customer_orders_lineitem을 super_mv와 조인합니다.

```
SELECT c.c_name, v.o_orderstatus
FROM customer_orders_lineitem c
JOIN super_mv v ON c.c_custkey = v.c_custkey;
```

마찬가지로 두 번째 PartiQL 쿼리를 다시 작성할 수 있습니다. 다음 예에서는 o_idx = 0에 대한 필터를 사용합니다.

```
SELECT c.c_name, v.o_totalprice
FROM customer_orders_lineitem c
JOIN super_mv v ON c.c_custkey = v.c_custkey
WHERE v.o_idx = 0;
```

CREATE MATERIALIZED VIEW 명령에서 c_custkey를 배포 키로 지정하고 super_mv에 대한 정렬 키로 지정합니다. Amazon Redshift는 c_custkey가 customer_orders_lineitem의 배포 키이자 정렬 키라고 가정하고 효율적인 병합 조인을 수행합니다. 그렇지 않은 경우 다음과 같이 c_custkey를 customer_orders_lineitem의 정렬 키와 배포 키로 지정할 수 있습니다.

```
ALTER TABLE customer_orders_lineitem
ALTER DISTKEY c_custkey, ALTER SORTKEY (c_custkey);
```

EXPLAIN 문을 사용하여 Amazon Redshift가 재작성된 쿼리에 대해 병합 조인을 수행하는지 확인합니다.

```
EXPLAIN
      SELECT c.c_name, v.o_orderstatus
      FROM customer_orders_lineitem c JOIN super_mv v ON c.c_custkey = v.c_custkey;

      QUERY PLAN

 ------------------------------------------------------------------------------------------------------
      XN Merge Join DS_DIST_NONE  (cost=0.00..34701.82 rows=1470776 width=27)
      Merge Cond: ("outer".c_custkey = "inner".c_custkey)
      ->  XN Seq Scan on mv_tbl__super_mv__0 derived_table2  (cost=0.00..14999.86
 rows=1499986 width=13)
      ->  XN Seq Scan on customer_orders_lineitem c  (cost=0.00..999.96 rows=99996
 width=30)
      (4 rows)
```

나뉜 데이터에서 Amazon Redshift 스칼라 열 생성

SUPER에 저장된 스키마 없는 데이터는 Amazon Redshift의 성능에 영향을 미칠 수 있습니다. 예를 들어 범위 제한 스캔은 영역 맵을 효과적으로 사용할 수 없으므로 술어를 필터링하거나 조건을 조인합니다. 사용자와 BI 도구는 구체화된 뷰를 데이터의 기존 표현으로 사용하고 분석 쿼리의 성능을 높일 수 있습니다.

다음 쿼리는 구체화된 뷰 super_mv를 스캔하고 o_orderstatus를 필터링합니다.

```
SELECT c.c_name, v.o_totalprice
FROM customer_orders_lineitem c
JOIN super_mv v ON c.c_custkey = v.c_custkey
WHERE v.o_orderstatus = 'F';
```

stl_scan을 검사하여 Amazon Redshift가 o_orderstatus에 대한 범위 제한 스캔에서 영역 맵을 효과적으로 사용할 수 없는지 확인합니다.

```
SELECT slice, is_rrscan FROM stl_scan
WHERE query = pg_last_query_id() AND perm_table_name LIKE '%super_mv%';

 slice | is_rrscan
-------+-----------
     0 | f
     1 | f
     5 | f
     4 | f
     2 | f
     3 | f
(6 rows)
```

다음 예에서는 구체화된 뷰 super_mv를 조정하여 나뉜 데이터에서 스칼라 열을 생성합니다. 이 경우 Amazon Redshift는 SUPER에서 VARCHAR로 o_orderstatus를 캐스팅합니다. 또한 super_mv의 정렬 키로 o_orderstatus를 지정합니다.

```
CREATE MATERIALIZED VIEW super_mv distkey(c_custkey) sortkey(c_custkey, o_orderstatus)
 AS (
  SELECT c_custkey, o.o_orderstatus::VARCHAR AS o_orderstatus, o.o_totalprice, o_idx
  FROM customer_orders_lineitem c, c.c_orders o AT o_idx
);
```

쿼리를 다시 실행한 후 Amazon Redshift가 이제 영역 맵을 사용할 수 있는지 확인합니다.

```
SELECT v.o_totalprice
FROM super_mv v
WHERE v.o_orderstatus = 'F';
```

이제 범위 제한 스캔이 다음과 같이 영역 맵을 사용하는지 확인할 수 있습니다.

```
SELECT slice, is_rrscan FROM stl_scan
WHERE query = pg_last_query_id() AND perm_table_name LIKE '%super_mv%';

 slice | is_rrscan
-------+-----------
     0 | t
     1 | t
     2 | t
     3 | t
     4 | t
     5 | t
(6 rows)
```

구체화된 뷰에서 SUPER 데이터 형식 사용에 대한 제한 사항

Amazon Redshift를 사용하면 SUPER 데이터 유형을 사용하여 구체화된 뷰를 만들어 복잡한 쿼리를 미리 컴퓨팅하고 쿼리 성능을 개선할 수 있습니다. SUPER 데이터 유형으로 쿼리의 결과 세트를 플랫 데이터 구조로 저장하여 액세스 및 처리를 더 빠르게 지원할 수 있습니다.

다음 섹션에서는 Amazon Redshift에서 구체화된 뷰로 SUPER 데이터 유형을 사용하기 위한 제한 사항 및 모범 사례에 대한 세부 정보를 제공합니다.

Amazon Redshift의 구체화된 뷰에는 PartiQL 또는 SUPER와 관련된 특정 제한 사항이 없습니다.

구체화된 뷰 생성 시 일반적인 SQL 제한 사항에 대한 자세한 내용은 제한 사항 섹션을 참조하세요.

구체화된 뷰의 증분 새로 고침에 대한 일반적인 SQL 제한 사항에 대한 자세한 내용은 증분 새로 고침에 대한 제한 사항 섹션을 참조하세요.

기계 학습

Amazon Redshift 기계 학습은 모든 기술 수준의 분석가와 데이터 사이언티스트가 기계 학습 기술을 더 쉽게 사용할 수 있도록 하는 강력한 클라우드 기반 서비스입니다. Amazon Redshift ML은 모델을 사용하여 결과를 만듭니다. 모델 사용 방법은 다음과 같습니다.

- 모델을 훈련하려는 데이터와 데이터 입력과 연결된 메타데이터를 Amazon Redshift에 제공합니다. 그런 다음 Amazon Redshift ML은 입력 데이터의 패턴을 캡처하는 Amazon SageMaker AI의 모델을 생성합니다. 모델에 자체 데이터를 사용하여 Amazon Redshift ML로 이탈 예측, 고객 생애 주기 값 또는 수익 예측 등의 데이터 추세를 식별할 수 있습니다. 이러한 모델을 사용하면 추가 비용 없이 새 입력 데이터에 대한 예측을 생성하는 것이 가능합니다.
- Claude 또는 Amazon Titan과 같이 Amazon Bedrock에서 제공하는 파운데이션 모델(FM) 중 하나를 사용할 수 있습니다. Amazon Bedrock을 사용하면 몇 가지 단계로 대규모 언어 모델(LLM)의 힘을 Amazon Redshift의 분석 데이터와 통합할 수 있습니다. 외부 대규모 언어 모델(LLM)을 사용하여 Amazon Redshift로 데이터에 대한 자연어 처리(NLP)를 수행하는 것이 가능합니다. 텍스트 작성, 감정 분석 또는 번역 등의 애플리케이션에 NLP를 사용할 수 있습니다. Amazon Redshift에서 Amazon Bedrock 사용에 관한 자세한 내용은 Amazon Redshift ML과 Amazon Bedrock의 통합 섹션을 참조하시기 바랍니다.

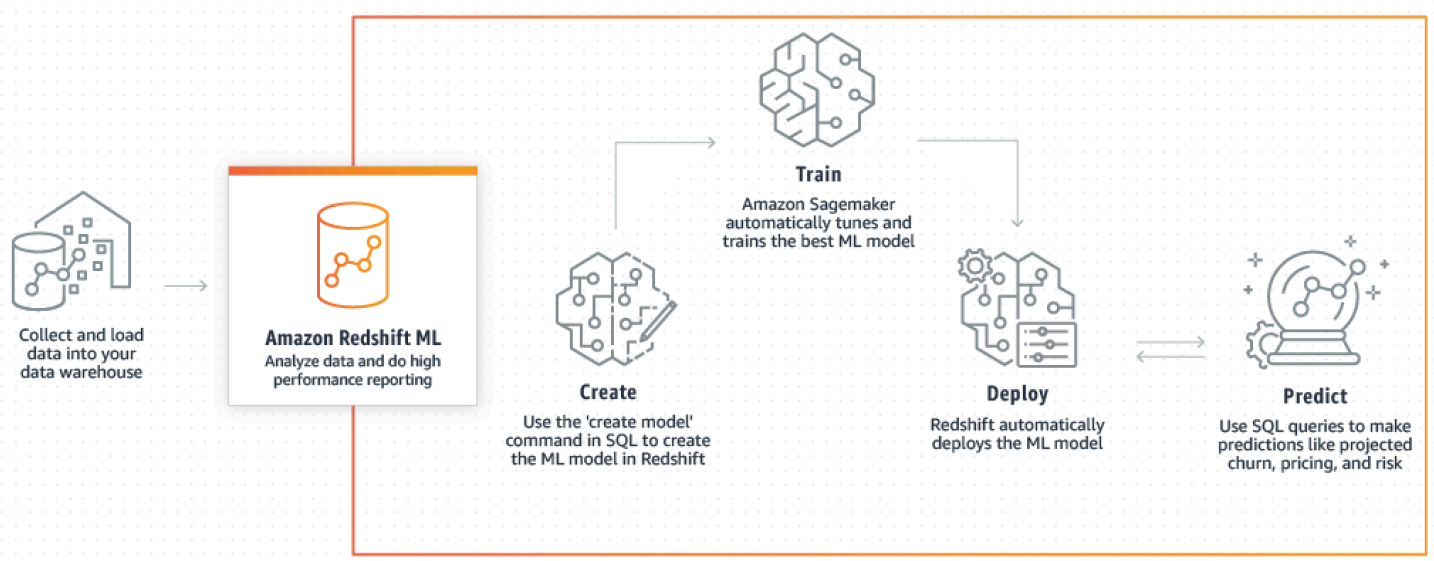
> **Note**
>
> 서비스 개선을 위한 데이터 사용 선택 해제
> Amazon Bedrock 모델을 사용하고 있으며 서비스 개선을 목적으로 AWS가 데이터를 처리하지 않도록 하려면 Amazon Bedrock의 선택 해제 정책을 사용 설정해야 합니다.

> **Note**
>
> LLM은 부정확한 정보나 불완전한 정보를 생성할 수 있습니다. LLM이 생성하는 정보를 확인하여 정확하고 완전한 정보인지 확인하는 것이 좋습니다.

Amazon Redshift 기계 학습이 Amazon SageMaker AI와 작동하는 방식

Amazon Redshift는 Amazon SageMaker AI Autopilot과 함께 작동하여 최상의 모델을 자동으로 얻고 Amazon Redshift에서 예측 함수를 사용할 수 있도록 합니다.

다음 다이어그램은 Amazon Redshift 기계 학습의 작동 방식을 보여줍니다.



일반적인 워크플로는 다음과 같습니다.

1. Amazon Redshift는 훈련 데이터를 Amazon S3로 내보냅니다.
2. Amazon SageMaker AI Autopilot은 훈련 데이터를 사전 처리합니다. 사전 처리는 누락 값 대치와 같은 중요한 기능을 수행합니다. 특정 열이 우편 번호와 같은 범주형임을 인식하고 훈련을 위해 적절한 형식을 지정하고 기타 여러 태스크를 수행합니다. 훈련 데이터세트에 적용할 최고의 전처리기를 선택하는 것 자체가 문제이며 Amazon SageMaker AI Autopilot은 솔루션을 자동화합니다.
3. Amazon SageMaker AI Autopilot은 가장 정확한 예측으로 모델을 제공하는 알고리즘 및 알고리즘 하이퍼파라미터를 찾습니다.
4. Amazon Redshift는 예측 함수를 Amazon Redshift 클러스터에 SQL 함수로 등록합니다.
5. CREATE MODEL 문을 실행할 때 Amazon Redshift는 훈련에 Amazon SageMaker AI를 사용합니다. 따라서 모델 훈련을 위한 관련 비용이 있습니다. 이는 AWS 청구서에서 Amazon SageMaker AI에 대한 별도의 품목입니다. 또한 훈련 데이터를 저장하기 위해 Amazon S3에서 사용하는 스토리지에 대한 비용을 지불합니다. Redshift 클러스터에서 컴파일하고 실행할 수 있는 CREATE MODEL로 생성된 모델을 사용한 추론에는 요금이 부과되지 않습니다. Amazon Redshift 기계 학습 사용에 따른 추가 Amazon Redshift 요금은 없습니다.

주제

- 기계 학습 개요
- 초보자 및 전문가를 위한 기계 학습
- Amazon Redshift 기계 학습 사용 비용

- Amazon Redshift ML 시작하기
- Amazon Redshift ML 튜토리얼
- Amazon Redshift ML과 Amazon Bedrock의 통합

기계 학습 개요

Amazon Redshift를 사용하면 기계 학습 기능을 활용하여 데이터에서 중요한 인사이트를 확보할 수 있습니다. 이 기계 학습(ML) 개요에서는 ML 모델 훈련 및 배포용 데이터를 탐색, 시각화 및 준비하는 방법을 보여 줍니다. 다음 섹션에서는 Amazon Redshift ML을 사용하여 기계 학습으로 데이터의 잠재력을 활용하는 프로세스를 설명합니다.

Amazon Redshift 기계 학습을 사용하면 SQL 문으로 기계 학습 모델을 훈련하고 예측을 위해 SQL 쿼리에서 해당 기계 학습 모델을 호출할 수 있습니다.

Amazon Redshift 기계 학습 사용 방법에 대해 알아보려면 Amazon Redshift 기계 학습 동영상을 시청하세요.

Redshift 클러스터 또는 서버리스 작업 그룹 설정을 위한 전제 조건, 권한 및 Amazon Redshift ML 사용에 대한 소유권에 대한 자세한 내용은 다음 섹션을 참조하시기 바랍니다. 이 섹션에서는 Amazon Redshift 기계 학습에서 간단한 훈련 및 예측이 작동하는 방식도 설명합니다.

기계 학습으로 문제를 해결하는 방법

기계 학습 모델은 훈련 데이터에서 패턴을 찾아 새 데이터에 적용하여 예측을 생성합니다. 기계 학습에서는 데이터를 가장 잘 설명하는 패턴을 학습하여 이러한 모델을 훈련합니다. 그런 다음 모델을 사용하여 새 데이터에 대한 예측(추론이라고도 함)을 수행합니다. 기계 학습은 일반적으로 파라미터를 변경하고 훈련 데이터를 개선하여 예측의 정확도를 계속해서 개선할 수 있는 반복적인 프로세스입니다. 데이터가 변경되면 새 데이터 집합으로 새 모델을 다시 훈련합니다.

다양한 비즈니스 목표를 달성하기 위한 여러 가지 기본 기계 학습 접근 방식이 있습니다.

Amazon Redshift 기계 학습의 지도 학습

Amazon Redshift는 고급 엔터프라이즈 분석에 대한 가장 일반적인 접근 방식인 지도 학습을 지원합니다. 지도 학습은 설정된 데이터 집합이 있고 특정 입력 데이터가 다양한 비즈니스 결과를 예측하는 방법을 이해하고 있는 경우 선호되는 기계 학습 방식입니다. 이러한 결과는 레이블이라고도 합니다. 특히 데이터 집합은 특성(입력)과 타겟(출력)으로 구성된 속성이 있는 테이블입니다. 예를 들어 과거 및 현재

고객의 연령 및 우편 번호를 제공하는 테이블이 있다고 가정합니다. 현재 고객에 대해 true이고 멤버십을 일시 중단한 고객에 대해 false인 "활성" 필드도 있다고 가정합니다. 지도 기계 학습의 목표는 대상이 "False"인 고객으로 대표되는 고객 이탈로 이어지는 연령 및 우편 번호 패턴을 파악하는 것입니다. 이 모델을 사용하여 멤버십 일시 중단과 같이 이탈할 가능성이 있는 고객을 예측하고 잠재적으로 유지 인센티브를 제공할 수 있습니다.

Amazon Redshift는 회귀, 이진 분류 및 다중 클래스 분류를 포함하는 지도 학습을 지원합니다. 회귀는 고객의 총 지출과 같은 연속 값을 예측하는 문제를 나타냅니다. 이진 분류는 고객 이탈 여부를 예측하는 것과 같이 두 가지 결과 중 하나를 예측하는 문제를 나타냅니다. 다중 클래스 분류는 고객이 관심을 가질 만한 항목을 예측하는 것과 같이 많은 결과 중 하나를 예측하는 문제를 나타냅니다. 데이터 분석가와 데이터 사이언티스트는 이를 사용하여 예측, 개인화 또는 고객 이탈 예측에 이르는 문제를 해결할 수 있는 지도 학습을 수행할 수 있습니다. 또한 매출 종료 예측, 수익 예측, 사기 탐지 및 고객 평생 가치 예측과 같은 문제에서 지도 학습을 사용할 수 있습니다.

Amazon Redshift 기계 학습의 비지도 학습

비지도 학습은 기계 학습 알고리즘을 사용하여 레이블이 지정되지 않은 훈련 데이터를 분석하고 그룹화합니다. 알고리즘은 숨겨진 패턴 또는 그룹화를 검색합니다. 목표는 데이터의 기본 구조 또는 분포를 모델링하여 데이터에 대해 자세히 알아보는 것입니다.

비지도 학습 문제를 해결하기 위해 Amazon Redshift는 K-Means 클러스터링 알고리즘을 지원합니다. 이 알고리즘은 데이터에서 그룹을 검색하려는 클러스터링 문제를 해결합니다. K-Means 알고리즘은 데이터 내에서 개별 그룹 찾기를 시도합니다. 분류되지 않은 데이터는 유사점과 차이점에 따라 그룹화되고 분할됩니다. 그룹화를 통해 K-Means 알고리즘은 최상의 중심을 반복적으로 결정하고 각 구성원을 가장 가까운 중심에 할당합니다. 같은 중심에 가장 가까운 멤버는 같은 그룹에 속합니다. 그룹의 멤버는 같은 그룹의 다른 멤버와 최대한 유사하고 다른 그룹의 멤버와 최대한 다릅니다. 예를 들어, K-Means 클러스터링 알고리즘은 전염병의 영향을 받는 도시를 분류하거나 소비재의 인기를 기반으로 도시를 분류하는 데 사용할 수 있습니다.

K-Means 알고리즘을 사용할 때 데이터에서 찾을 클러스터 수를 지정하는 입력 k를 지정합니다. 이 알고리즘의 출력은 k 중심 집합입니다. 각 데이터 포인트는 가장 가까운 k 클러스터 중 하나에 속합니다. 각 클러스터는 중심으로 설명됩니다. 클러스터의 다차원 평균으로 중심을 생각할 수 있습니다. K-Means 알고리즘은 거리를 비교하여 클러스터가 서로 얼마나 다른지 확인합니다. 거리가 멀수록 일반적으로 클러스터 간의 차이가 커집니다.

데이터 사전 처리는 모델의 특성이 동일한 규모로 유지되고 신뢰할 수 있는 결과를 생성하도록 보장하기 때문에 K-Means에 중요합니다. Amazon Redshift는 StandardScaler, MinMax 및 NumericPassthrough와 같은 CREATE MODEL 문에 대한 일부 K-Means 프로프로세서를 지원합니다. K-means에 대한 사전 처리를 적용하지 않으려면 변환기로 명시적으로 NumericPassthrough를 선택합

니다. K-Means 파라미터에 대한 자세한 내용은 K-MEANS 파라미터를 사용한 CREATE MODEL 섹션을 참조하세요.

K-Means 클러스터링을 사용하여 비지도 학습을 수행하는 방법에 대해 알아보려면 K-Means 클러스터링을 사용한 비지도 학습 동영상을 시청하세요.

Amazon Redshift 기계 학습 용어 및 개념

다음 용어는 몇 가지 Amazon Redshift 기계 학습 개념을 설명하는 데 사용됩니다.

- Amazon Redshift의 기계 학습은 하나의 SQL 명령으로 모델을 훈련합니다. Amazon Redshift 기계 학습과 Amazon SageMaker AI는 모든 데이터 변환, 권한, 리소스 사용 및 적절한 모델 검색을 관리합니다.
- 훈련은 Amazon Redshift가 지정된 데이터 하위 집합을 모델로 실행하여 기계 학습 모델을 생성하는 단계입니다. Amazon Redshift는 자동으로 Amazon SageMaker AI에서 훈련 작업을 시작하고 모델을 생성합니다.
- 추론이라고도 하는 예측은 Amazon Redshift SQL 쿼리에서 모델을 사용하여 결과를 예측하는 것입니다. 추론 시 Amazon Redshift는 모델 기반 예측 함수를 더 큰 쿼리의 일부로 사용하여 예측을 생성합니다. 예측은 Redshift 클러스터에서 로컬로 계산되므로 처리량이 높고 대기 시간이 짧으며 추가 비용이 없습니다.
- 기존 보유 모델 사용(BYOM)으로 Amazon Redshift에서 로컬로 데이터베이스 내 추론을 위해 Amazon SageMaker AI와 함께 Amazon Redshift 외부에서 훈련된 모델을 사용할 수 있습니다. Amazon Redshift 기계 학습은 로컬 추론에서 BYOM 사용을 지원합니다.
- 로컬 추론은 모델이 Amazon SageMaker AI에서 사전 훈련되고 Amazon SageMaker AI Neo에서 컴파일되고 Amazon Redshift 기계 학습에서 현지화될 때 사용됩니다. 로컬 추론이 지원되는 모델을 Amazon Redshift로 가져오려면 CREATE MODEL 명령을 사용합니다. Amazon Redshift는 Amazon SageMaker AI Neo를 직접 호출하여 사전 훈련된 SageMaker AI 모델을 가져옵니다. 여기서 모델을 컴파일하고 컴파일된 모델을 Amazon Redshift로 가져옵니다. 더 빠른 속도와 더 낮은 비용을 위해 로컬 추론을 사용합니다.
- 원격 추론은 Amazon Redshift가 SageMaker AI에 배포된 모델 엔드포인트를 간접 호출할 때 사용됩니다. 원격 추론은 Amazon SageMaker AI에서 구축 및 배포한 TensorFlow 모델과 같은 모든 유형의 사용자 지정 정의 및 딥 러닝 모델을 간접 호출할 수 있는 유연성을 제공합니다.

다음과 같은 사항도 중요합니다.

- Amazon SageMaker AI는 완전관리형 기계 학습 서비스입니다. Amazon SageMaker AI를 통해 데이터 과학자와 개발자는 모델을 쉽게 구축 및 훈련하고 프로덕션 지원 호스팅 환경에 직접 배포할 수

있습니다. Amazon SageMaker AI에 대한 자세한 내용은 Amazon SageMaker AI Developer Guide의 What is Amazon SageMaker AI 섹션을 참조하세요.

- Amazon SageMaker AI Autopilot는 데이터를 기반으로 분류 또는 회귀에 가장 적합한 기계 학습 모델을 자동으로 훈련하고 조정하는 기능 집합입니다. 완전한 제어와 가시성을 유지합니다. Amazon SageMaker AI Autopilot은 테이블형 입력 데이터를 지원합니다. Amazon SageMaker AI Autopilot은 자동 데이터 정리 및 사전 처리, 선형 회귀를 위한 자동 알고리즘 선택, 바이너리 분류 및 멀티클래스 분류를 제공합니다. 또한 자동 하이퍼파라미터 최적화(HPO), 분산 교육, 자동 인스턴스 및 클러스터 크기 선택을 지원합니다. Amazon SageMaker AI Autopilot에 대한 자세한 내용은 Amazon SageMaker AI Developer Guide의 Automate model development with Amazon SageMaker AI Autopilot 섹션을 참조하세요.
- Amazon Bedrock은 완전관리형 서비스로, 생성형 AI 애플리케이션을 구축하는 데 필요한 광범위한 여러 기능과 함께 단일 API를 통해 AI21 Labs, Anthropic, Cohere, Meta, Mistral AI, Stability AI 및 Amazon과 같은 선도적인 AI 회사의 고성능 파운데이션 모델(FM)을 제공합니다.

초보자 및 전문가를 위한 기계 학습

Amazon Redshift를 사용하면 기계 학습(ML) 기능을 활용하여 ML 초보자든 전문가든 관계없이 데이터에서 인사이트를 얻을 수 있습니다. 기계 학습은 광범위한 ML 전문 지식이나 복잡한 데이터 엔지니어링 없이 SQL 명령을 사용하여 ML 모델을 생성, 훈련 및 배포할 수 있는 Amazon Redshift 기능입니다.

다음 섹션을 통해 기계 학습을 활용하는 프로세스를 설명하여 Amazon Redshift를 사용하여 데이터의 잠재력을 최대한 활용할 수 있도록 지원합니다.

Amazon Redshift ML을 사용하면 단일 SQL CREATE MODEL 명령으로 모델을 훈련할 수 있습니다. CREATE MODEL 명령은 Amazon Redshift에서 익숙한 SQL 구문으로 모델 기반 예측을 생성하는 데 사용하는 모델을 생성합니다.

Amazon Redshift 기계 학습은 기계 학습, 도구, 언어, 알고리즘 및 API에 대한 전문 지식이 없는 경우 특히 유용합니다. Amazon Redshift 기계 학습을 사용하면 외부 기계 학습 서비스와 통합하는 데 필요한 획일적이고 과중한 업무를 수행할 필요가 없습니다. Amazon Redshift를 사용하면 데이터 형식 지정 및 이동, 권한 제어 관리, 사용자 정의 통합, 워크플로 및 스크립트 구축에 소요되는 시간을 절약할 수 있습니다. 인기 있는 기계 학습 알고리즘을 쉽게 사용하고 훈련에서 예측까지 자주 반복해야 하는 훈련 요구 사항을 단순화할 수 있습니다. Amazon Redshift는 자동으로 최상의 알고리즘을 검색하고 문제에 가장 적합한 모델을 조정합니다. Amazon Redshift 외부로 데이터를 이동하거나 다른 서비스와 인터페이스하고 비용을 지불할 필요 없이 Amazon Redshift 클러스터 내에서 예측할 수 있습니다.

Amazon Redshift 기계 학습은 기계 학습을 사용하는 데이터 분석가와 데이터 사이언티스트를 지원합니다. 또한 이를 통해 기계 학습 전문가는 지식을 사용하여 CREATE MODEL 문이 지정한 측면만 사용

하도록 안내할 수 있습니다. 이렇게 하면 CREATE MODEL이 최상의 후보를 찾는 데 필요한 시간을 단축하거나 모델의 정확도를 높이거나 둘 다 가능합니다.

CREATE MODEL 문은 훈련 작업에 파라미터를 지정하는 방법에 유연성을 제공합니다. 이러한 유연성을 통해 기계 학습 초보자 또는 전문가 모두 선호하는 프로프로세서, 알고리즘, 문제 유형 및 하이퍼파라미터를 선택할 수 있습니다. 예를 들어 고객 이탈에 관심이 있는 사용자는 CREATE MODEL 문에서 문제 유형이 고객 이탈에 잘 작동하는 이진 분류라고 지정할 수 있습니다. 그런 다음 CREATE MODEL 문은 최상의 모델 검색을 이진 분류 모델로 좁힙니다. 사용자가 문제 유형을 선택하더라도 CREATE MODEL 문이 사용할 수 있는 옵션은 여전히 많습니다. 예를 들어 CREATE MODEL은 최상의 사전 처리 변환을 검색 및 적용하고 최상의 하이퍼파라미터 설정을 검색합니다.

Amazon Redshift 기계 학습이 Amazon SageMaker AI Autopilot을 사용하여 최적의 모델을 자동으로 찾아 훈련을 더 쉽게 만듭니다. 백그라운드에서 Amazon SageMaker AI Autopilot은 제공된 데이터를 기반으로 최고의 기계 학습 모델을 자동으로 훈련하고 조정합니다. 그런 다음 Amazon SageMaker AI Neo가 훈련 모델을 컴파일하고 Redshift 클러스터에서 예측에 사용할 수 있도록 합니다. 훈련된 모델을 사용하여 기계 학습 추론 쿼리를 실행할 때 쿼리는 Amazon Redshift의 대규모 병렬 처리 기능을 사용할 수 있습니다. 동시에 쿼리는 기계 학습 기반 예측을 사용할 수 있습니다.

- 기계 학습 초보자로서 프로프로세서, 알고리즘 및 하이퍼파라미터와 같은 기계 학습의 다양한 측면에 대한 일반 지식이 있는 경우 지정한 측면에만 CREATE MODEL 문을 사용합니다. 그런 다음 CREATE MODEL이 최상의 후보를 찾는 데 필요한 시간을 줄이거나 모델의 정확도를 향상시킬 수 있습니다. 문제 유형이나 목표와 같은 추가 도메인 지식을 도입하여 예측의 비즈니스 가치를 높일 수도 있습니다. 예를 들어 고객 이탈 시나리오에서 "고객이 활동하고 있지 않음"이라는 결과가 드물다면 F1 목표가 정확도 목표보다 선호되는 경우가 많습니다. 높은 정확도 모델은 항상 "고객이 활동하고 있음"이라고 예측할 수 있기 때문에 정확도는 높지만 비즈니스 가치는 거의 없습니다. F1 목표에 대한 자세한 내용은 Amazon SageMaker AI API Reference의 AutoMLJobObjective 섹션을 참조하세요.

  CREATE MODEL 문의 기본 옵션에 대한 자세한 내용은 단순 CREATE MODEL 섹션을 참조하세요.
- 기계 학습 고급 실무자는 특정(일부) 기능에 대한 문제 유형과 전처리기를 지정할 수 있습니다. 그런 다음 CREATE MODEL은 지정된 측면에 대한 제안을 따릅니다. 동시에 CREATE MODEL은 여전히 나머지 기능과 최고의 하이퍼파라미터에 대한 최고의 프로프로세서를 검색합니다. 훈련 파이프라인의 하나 이상의 측면을 제한하는 방법에 대한 자세한 내용은 사용자 안내에 따라 CREATE MODEL 섹션을 참조하세요.
- 기계 학습 전문가는 훈련 및 하이퍼파라미터 튜닝을 완벽하게 제어할 수 있습니다. 그러면 CREATE MODEL 문은 사용자가 모든 선택을 하기 때문에 최적의 프로프로세서, 알고리즘 및 하이퍼파라미터를 검색하려고 하지 않습니다. AUTO OFF와 함께 CREATE MODEL을 사용하는 방법에 대한 자세한 내용은 AUTO OFF로 CREATE XGBoost 모델 섹션을 참조하세요.

- 데이터 엔지니어는 로컬 추론을 위해 Amazon SageMaker AI에서 Amazon Redshift로 사전 훈련된 XGBoost 모델을 가져올 수 있습니다. 기존 보유 모델 사용(BYOM)으로 Amazon Redshift에서 로컬로 데이터베이스 내 추론을 위해 Amazon SageMaker AI와 함께 Amazon Redshift 외부에서 훈련된 모델을 사용할 수 있습니다. Amazon Redshift 기계 학습은 로컬 또는 원격 추론에서 BYOM 사용을 지원합니다.

  로컬 또는 원격 추론에 CREATE MODEL 문을 사용하는 방법에 대한 자세한 내용은 기존 보유 모델 사용(BYOM) - 로컬 추론 섹션을 참조하세요.

Amazon Redshift 기계 학습 사용자는 다음 옵션을 선택하여 모델을 훈련하고 배포할 수 있습니다.

- 문제 유형, 사용자 안내에 따라 CREATE MODEL 섹션을 참조하세요.
- 목표, 사용자 안내에 따라 CREATE MODEL 또는 AUTO OFF로 CREATE XGBoost 모델 섹션을 참조하세요.
- 모델 유형, AUTO OFF로 CREATE XGBoost 모델 섹션을 참조하세요.
- 전처리기, 사용자 안내에 따라 CREATE MODEL 섹션을 참조하세요.
- 하이퍼파라미터, AUTO OFF로 CREATE XGBoost 모델 섹션을 참조하세요.
- 기존 보유 모델 사용(BYOM), 기존 보유 모델 사용(BYOM) - 로컬 추론 섹션을 참조하세요

Amazon Redshift 기계 학습 사용 비용

Amazon Redshift를 사용하면 광범위한 데이터 엔지니어링 또는 기계 학습 전문 지식 없이 기계 학습 기능을 활용하여 데이터에서 인사이트를 도출할 수 있습니다. 다음 섹션에서는 Amazon Redshift ML 사용과 관련된 비용을 설명하며, 이는 강력한 기계 학습 통합을 활용하면서 비용을 계획하고 최적화하는 데 도움이 됩니다.

SageMaker AI에서 Amazon Redshift ML 사용 시 비용

SageMaker AI용 Amazon Redshift ML은 예측에 기존 클러스터 리소스를 사용하므로 추가 Amazon Redshift 요금을 피할 수 있습니다. 모델 생성 또는 사용에 대한 추가 Amazon Redshift 요금은 없습니다. Redshift 클러스터에서 로컬로 예측이 이루어지므로 클러스터 크기를 조정해야 하는 경우가 아니면 추가 비용을 지불할 필요가 없습니다. Amazon Redshift 기계 학습은 모델 훈련에 Amazon SageMaker AI를 사용하며, 이 경우 추가 관련 비용이 발생합니다.

Amazon Redshift 클러스터 내에서 실행되는 예측 함수에 대한 추가 요금은 없습니다. CREATE MODEL 문은 Amazon SageMaker AI를 사용하며 추가 비용이 발생합니다. 비용은 훈련 데이터의 셀 수에 따라 증가합니다. 셀 수는 레코드 수(훈련 쿼리 또는 테이블 시간)에 열 수를 곱한 값입니다. 예를

들어 CREATE MODEL 문의 SELECT 쿼리가 10,000개의 레코드와 5개의 열을 생성할 때 생성되는 셀의 수는 50,000개입니다.

경우에 따라 CREATE MODEL의 SELECT 쿼리에 의해 생성된 훈련 데이터가 제공한 MAX_CELLS 제한(제한을 제공하지 않은 경우 기본 100만 개)을 초과합니다. 이러한 경우 CREATE MODEL은 약 MAX_CELLS개(즉, 훈련 데이터 집합의 "열 수" 레코드)를 무작위로 선택합니다. 그런 다음 CREATE MODEL은 무작위로 선택된 튜플을 사용하여 훈련을 수행합니다. 무작위 샘플링은 감소된 훈련 데이터 집합에 편향이 없도록 합니다. 따라서 MAX_CELLS를 설정하여 훈련 비용을 제어할 수 있습니다.

CREATE MODEL 문을 사용할 때 MAX_CELLS 및 MAX_RUNTIME 옵션으로 비용, 시간 및 잠재적인 모델 정확도를 제어할 수 있습니다.

MAX_RUNTIME은 AUTO ON 또는 OFF 옵션이 사용될 때 SageMaker AI에서 훈련에 소요될 수 있는 최대 시간을 지정합니다. 데이터 집합의 크기에 따라 MAX_RUNTIME보다 훈련 작업이 빨리 완료되는 경우가 많습니다. 모델 훈련 후 Amazon Redshift는 백그라운드에서 추가 작업을 수행하여 모델을 컴파일하고 클러스터에 설치합니다. 따라서 CREATE MODEL을 완료하는 데 MAX_RUNTIME보다 더 오래 걸릴 수 있습니다. 그러나 MAX_RUNTIME은 SageMaker AI에서 모델 훈련에 사용되는 계산량과 시간을 제한합니다. SHOW MODEL을 사용하여 언제든지 모델 상태를 확인할 수 있습니다.

AUTO ON으로 CREATE MODEL을 실행하면 Amazon Redshift 기계 학습은 SageMaker AI Autopilot을 사용하여 다양한 모델 또는 후보를 지능적으로 자동 탐색하여 최적의 모델을 찾습니다. MAX_RUNTIME은 소요되는 시간과 계산 시간을 제한합니다. MAX_RUNTIME이 너무 낮게 설정되면 하나의 후보라도 탐색할 시간이 충분하지 않을 수 있습니다. ["Autopilot 후보에 모델이 없습니다.(Autopilot candidate has no models)"] 오류가 표시되면 MAX_RUNTIME 값을 늘리고 CREATE MODEL을 다시 실행합니다. 이 파라미터에 대한 자세한 내용은 Amazon SageMaker AI API Reference의 MaxAutoMLJobRuntimeInSeconds 섹션을 참조하세요.

AUTO OFF로 CREATE MODEL을 실행하면 MAX_RUNTIME은 SageMaker AI에서 훈련 작업이 실행되는 시간에 대한 제한에 해당합니다. 데이터 집합의 크기와 MODEL_TYPE XGBOOST의 num_rounds와 같이 사용된 기타 파라미터에 따라 훈련 작업이 더 빨리 완료되는 경우가 많습니다.

CREATE MODEL을 실행할 때 MAX_CELLS 값을 줄여서 비용을 제어하거나 훈련 시간을 줄일 수도 있습니다. 셀은 데이터베이스의 항목입니다. 각 행은 고정된 너비 또는 다양한 너비의 열 수만큼의 셀에 해당합니다. MAX_CELLS는 셀 수를 제한하므로 모델 훈련에 사용되는 훈련 예제의 수를 제한합니다. 기본적으로 MAX_CELLS는 셀 100만 개로 설정됩니다. MAX_CELLS를 줄이면 Amazon Redshift가 모델 훈련을 위해 내보내고 SageMaker AI로 보내는 CREATE MODEL의 SELECT 쿼리 결과에서 행 수가 줄어듭니다. 따라서 MAX_CELLS를 줄이면 AUTO ON 및 AUTO OFF로 모델을 훈련하는 데 사용되는 데이터 집합의 크기가 줄어듭니다. 이 방법으로 모델 훈련에 드는 비용과 시간을 줄일 수 있습

니다. 특정 훈련 작업의 훈련 및 청구 시간에 대한 정보를 보려면 Amazon SageMaker AI에서 훈련 작업을 선택합니다.

MAX_RUNTIME 및 MAX_CELLS를 늘리면 SageMaker AI가 더 많은 후보를 탐색할 수 있어 모델 품질이 향상되는 경우가 많습니다. 이러한 방식으로 SageMaker AI는 더 많은 시간을 들여 각 후보를 훈련하고 더 많은 데이터를 사용하여 더 나은 모델을 훈련할 수 있습니다. 데이터 집합을 더 빠르게 반복하거나 탐색하려면 MAX_RUNTIME과 MAX_CELLS를 줄입니다. 모델의 정확도를 높이려면 MAX_RUNTIME과 MAX_CELL을 늘립니다.

다양한 셀 번호와 관련된 비용 및 무료 평가판 세부 정보에 대한 자세한 내용은 Amazon Redshift 요금 섹션을 참조하세요.

Amazon Bedrock에서 Amazon Redshift ML 사용 시 비용

Amazon Bedrock에서 Amazon Redshift ML을 사용하면 추가 비용이 발생합니다. 자세한 내용은 Amazon Bedrock 요금을 참조하세요.

Amazon Redshift ML 시작하기

Amazon Redshift 기계 학습을 사용하면 SQL 사용자가 익숙한 SQL 명령으로 기계 학습 모델을 쉽게 생성, 훈련 및 배포할 수 있습니다. Amazon Redshift ML을 사용하면 Redshift 클러스터의 데이터와 Amazon SageMaker AI로 모델을 훈련할 수 있습니다. 그 후 모델이 현지화되고 Amazon Redshift 데이터베이스 내에서 예측이 가능합니다. 현재 Amazon Redshift ML은 기계 학습 알고리즘인 XGBoost(AUTO ON 및 OFF)와 다계층 퍼셉트론(AUTO ON), K-평균(AUTO OFF) 및 선형 학습기를 지원합니다.

주제

- Amazon Redshift 기계 학습 관리를 위한 클러스터 및 구성 설정
- Amazon Redshift 기계 학습에서 모델 설명 사용
- Amazon Redshift ML 확률 지표

Amazon Redshift 기계 학습 관리를 위한 클러스터 및 구성 설정

Amazon Redshift 기계 학습을 사용하기 전에 클러스터 설정을 완료하고 Amazon Redshift 기계 학습 사용 권한을 구성합니다.

Amazon Redshift 기계 학습 사용을 위한 클러스터 설정

Amazon Redshift 기계 학습을 사용하기 전에 다음 사전 조건을 충족합니다.

Amazon Redshift 관리자는 Amazon Redshift 프로비저닝 클러스터를 사용하기 위해 다음의 일회성 설정을 수행합니다. Amazon Redshift Serverless에서 Amazon Redshift ML을 사용하려면 Amazon Redshift Serverless 시작하기를 참조하시기 바랍니다.

Amazon Redshift 기계 학습에 대해 일회성 클러스터 설정을 수행하려면

1. AWS Management Console 또는 AWS Command Line Interface(AWS CLI)를 사용하여 Redshift 클러스터를 생성합니다. 클러스터를 생성하는 동안 AWS Identity and Access Management(IAM) 정책을 연결해야 합니다. Amazon SageMaker AI에서 Amazon Redshift ML을 사용하는 데 필요한 권한에 대한 자세한 내용은 Amazon Redshift 기계 학습(ML)을 사용하는 데 필요한 권한을 참조하시기 바랍니다.
2. 다음 방법 중 하나로 Amazon Redshift 기계 학습을 사용하는 데 필요한 IAM 역할을 생성합니다.

   - Amazon Redshift ML로 SageMaker AI를 사용하려면 `AmazonS3FullAccess` 및 `AmazonSageMakerFullAccess` 정책을 포함해 IAM 역할을 만듭니다. 예측 모델도 만들 계획이라면 `AmazonForecastFullAccess` 정책도 역할에 연결하세요.
   - Amazon Redshift ML로 Amazon Bedrock을 사용하려면 `AmazonS3FullAccess` 및 `AmazonBedrockFullAccess` 정책을 포함해 IAM 역할을 만듭니다.
   - CREATE MODEL과 같은 SQL 명령을 실행할 수 있는 권한을 가진 `AmazonRedshiftAllCommandsFullAccess` 정책이 있는 Amazon Redshift 콘솔을 통해 IAM 역할을 생성하는 것이 좋습니다. Amazon Redshift는 원활한 API 기반 메커니즘을 사용하여 사용자를 대신하여 AWS 계정에서 프로그래밍 방식으로 IAM 역할을 생성합니다. Amazon Redshift는 기존 AWS 관리형 정책을 IAM 역할에 자동으로 연결합니다. 이 접근 방식을 사용하면 Amazon Redshift 콘솔 내에 머물 수 있고 역할 생성을 위해 IAM 콘솔로 전환할 필요가 없습니다. 자세한 내용은 Amazon Redshift의 기본값으로 IAM 역할 생성을 참조하세요.

     IAM 역할이 클러스터의 기본값으로 생성되면 리소스 이름의 일부로 `redshift`를 포함하거나 Redshift 관련 태그를 사용하여 해당 리소스에 태그를 지정합니다.

     클러스터에 Enhanced Amazon VPC Routing이 설정된 경우 Amazon Redshift 콘솔을 통해 생성된 IAM 역할을 사용할 수 있습니다. 이 IAM 역할에는 `AmazonRedshiftAllCommandsFullAccess` 정책이 연결되어 있으며 정책에 다음 권한을 추가합니다. 이러한 추가 권한을 통해 Amazon Redshift는 계정에서 탄력적 네트워크 인터페이스(ENI)를 생성 및 삭제하고 Amazon EC2 또는 Amazon ECS에서 실행되는 컴파일 태스크에 연

결할 수 있습니다. 이렇게 하면 인터넷 액세스가 차단된 Virtual Private Cloud(VPC) 내에서만 Amazon S3 버킷의 객체에 액세스할 수 있습니다.

```
{
    "Effect": "Allow",
    "Action": [
        "ec2:DescribeVpcEndpoints",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeVpcs",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DeleteNetworkInterfacePermission",
        "ec2:DeleteNetworkInterface",
        "ec2:CreateNetworkInterfacePermission",
        "ec2:CreateNetworkInterface",
        "ec2:ModifyNetworkInterfaceAttribute"
    ],
    "Resource": "*"
}
```

Amazon Bedrock 파운데이션 모델을 사용하려면 다음 섹션을 추가합니다.

```
// Required section if you use Bedrock models.
{
   "Effect": "Allow",
   "Action": "bedrock:InvokeModel",
   "Resource": [
       "arn:aws:bedrock:<region>::foundation-model/*"
   ]
}
```

- 보다 제한적인 정책으로 IAM 역할을 생성하려는 경우 다음 정책을 사용합니다. 필요에 맞게 이 정책을 수정할 수도 있습니다.

  Amazon S3 버킷 `redshift-downloads/redshift-ml/`은 다른 단계 및 예제에 사용되는 샘플 데이터가 저장되는 위치입니다. Amazon S3에서 데이터를 로드할 필요가 없으면 이 버킷을 제거할 수 있습니다. 또는 Amazon Redshift로 데이터를 로드하는 데 사용하는 다른 Amazon S3 버킷으로 바꿉니다.

your-account-id, *your-role* 및 amzn-s3-demo-bucket 값은 CREATE MODEL 명령의 일부로 지정합니다.

(옵션) Amazon Redshift 기계 학습을 사용하는 동안 AWS KMS 키를 지정하는 경우 샘플 정책의 AWS KMS 키 섹션을 사용합니다. *your-kms-key* 값은 CREATE MODEL 명령의 일부로 사용하는 키입니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:GetAuthorizationToken",
                "ecr:GetDownloadUrlForLayer",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "sagemaker:*Job*",
                "sagemaker:AddTags",
                "sagemaker:CreateModel",
                "sagemaker:CreateEndpoint",
                "sagemaker:CreateEndpointConfig",
                "sagemaker:DeleteEndpoint",
                "sagemaker:DeleteEndpointConfig,
                "sagemaker:DeleteModel"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole",
                "s3:AbortMultipartUpload",
                "s3:GetObject",
                "s3:DeleteObject",
                "s3:PutObject"
            ],
```

```
            "Resource": [
                "arn:aws:iam::<your-account-id>:role/<your-role>",
                "arn:aws:s3:::amzn-s3-demo-bucket/*",
                "arn:aws:s3:::redshift-downloads/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket,
                "arn:aws:s3:::redshift-downloads"
            ]
        }
        // Optional section needed if you use AWS KMS keys.
        ,{
            "Effect": "Allow",
            "Action": [
                "kms:CreateGrant",
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:Encrypt",
                "kms:GenerateDataKey*"
             ],
             "Resource": [
                "arn:aws:kms:<your-region>:<your-account-id>:key/<your-kms-key>"
             ]
        }
    ]
}
```

3. Amazon Redshift와 SageMaker AI가 다른 서비스와 상호 작용하는 역할을 맡도록 허용하려면 IAM 역할에 다음 신뢰 정책을 추가합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
```

```
        "Service": [
          "redshift.amazonaws.com",
          "sagemaker.amazonaws.com",
          "forecast.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

4. (선택 사항) Amazon S3 버킷과 AWS KMS 키를 생성합니다. 이들은 Amazon Redshift가 Amazon SageMaker AI로 전송된 훈련 데이터를 저장하고 Amazon SageMaker AI에서 훈련된 모델을 수신하는 데 사용됩니다.
5. (옵션) 여러 사용자 그룹에 대한 액세스를 제어하기 위해 IAM 역할과 Amazon S3 버킷의 다양한 조합을 생성합니다.
6. (선택 사항) Redshift 클러스터에 대해 VPC 라우팅을 설정하면 Redshift 클러스터가 있는 VPC에 대해 Amazon S3 엔드포인트와 SageMaker AI 엔드포인트를 생성합니다. 이렇게 하면 CREATE MODEL을 수행하는 동안 서비스 간에 VPC를 통해 트래픽을 실행할 수 있습니다. VPC 라우팅에 대한 자세한 내용은 Amazon Redshift의 Enhanced VPC Routing 섹션을 참조하세요.

   하이퍼파라미터 튜닝 작업에 대해 프라이빗 VPC를 지정하는 데 필요한 권한에 대한 자세한 내용은 Permissions required to use Amazon Redshift ML with Amazon SageMaker AI를 참조하세요.

CREATE MODEL 문을 사용하여 다양한 사용 사례에 대한 모델 생성을 시작하는 방법에 대한 자세한 내용은 CREATE MODEL 섹션을 참조하세요.

권한 및 소유권 관리

테이블 또는 함수 등의 다른 데이터베이스 객체와 마찬가지로 Amazon Redshift는 기계 학습 모델 생성 및 사용을 액세스 제어 메커니즘에 바인딩합니다. 예측 함수를 실행하는 모델을 생성하기 위한 별도의 권한이 있습니다.

다음 예에서는 2개의 사용자 그룹인 `retention_analyst_grp`(모델 생성자)와 `marketing_analyst_grp`(모델 사용자)를 사용하여 Amazon Redshift의 액세스 제어 관리 방법을 보여줍니다. 보존 분석가는 획득한 권한을 통해 다른 사용자 집합이 사용할 수 있는 기계 학습 모델을 생성합니다.

슈퍼 사용자는 다음 문을 사용하여 기계 학습 모델 생성을 위한 GRANT USER 또는 GROUP 권한을 가질 수 있습니다.

```
GRANT CREATE MODEL TO GROUP retention_analyst_grp;
```

이 권한이 있는 사용자 또는 그룹은 사용자에게 SCHEMA에 대한 일반적인 CREATE 권한이 있는 경우 클러스터의 모든 스키마에서 모델을 생성할 수 있습니다. 기계 학습 모델은 테이블, 뷰, 프로시저 및 사용자 정의 함수와 유사한 방식으로 스키마 계층의 일부입니다.

demo_ml 스키마가 이미 있다고 가정하고 두 사용자 그룹에 다음과 같이 스키마에 대한 권한을 부여합니다.

```
GRANT CREATE, USAGE ON SCHEMA demo_ml TO GROUP retention_analyst_grp;
```

```
GRANT USAGE ON SCHEMA demo_ml TO GROUP marketing_analyst_grp;
```

다른 사용자가 기계 학습 추론 함수를 사용하도록 하려면 EXECUTE 권한을 부여합니다. 다음 예에서는 EXECUTE 권한을 사용하여 marketing_analyst_grp GROUP에 모델을 사용할 수 있는 권한을 부여합니다.

```
GRANT EXECUTE ON MODEL demo_ml.customer_churn_auto_model TO GROUP
 marketing_analyst_grp;
```

REVOKE 문을 CREATE MODEL 및 EXECUTE와 함께 사용하여 사용자 또는 그룹에서 해당 권한을 취소합니다. 권한 제어 명령에 대한 자세한 내용은 GRANT 및 REVOKE 섹션을 참조하세요.

Amazon Redshift 기계 학습에서 모델 설명 사용

Amazon Redshift 기계 학습의 모델 설명으로 기능 중요도 값을 사용하여 훈련 데이터의 각 속성이 예측 결과에 어떻게 기여하는지 파악할 수 있습니다.

모델 설명은 모델의 예측을 설명하여 기계 학습모델을 개선하는 데 도움이 됩니다. 모델 설명은 이러한 모델이 특성 속성 접근 방식을 사용하여 예측하는 방법을 설명하는 데 도움이 됩니다.

Amazon Redshift 기계 학습은 모델 설명을 통합하여 Amazon Redshift 기계 학습 사용자에게 모델 설명 기능을 제공합니다. 모델 설명에 대한 자세한 내용은 Amazon SageMaker AI Developer Guide의 What Is Fairness and Model Explainability for Machine Learning Predictions? 섹션을 참조하세요.

모델 설명은 또한 특성 속성 드리프트를 위해 프로덕션에서 모델이 만드는 추론을 모니터링합니다. 또한 위험 및 규정 준수 팀과 외부 규제 기관에 알리는 데 사용할 수 있는 모델 거버넌스 보고서를 생성하는 데 도움이 되는 도구를 제공합니다.

CREATE MODEL 문을 사용할 때 AUTO ON 또는 AUTO OFF 옵션을 지정하면 모델 훈련 작업이 완료된 후 SageMaker AI가 설명 출력을 생성합니다. EXPLAIN_MODEL 함수를 사용하여 설명 보고서를 JSON 형식으로 쿼리할 수 있습니다. 자세한 내용은 기계 학습 함수 단원을 참조하십시오.

Amazon Redshift ML 확률 지표

지도 학습 문제에서 클래스 레이블은 입력 데이터를 사용하는 예측의 결과입니다. 예를 들어 고객이 스트리밍 서비스에 다시 가입할지를 예측하기 위해 모델을 사용하는 경우 가능한 레이블은 가능성이 있고 가능성이 낮습니다. Redshift ML은 가능성을 나타내기 위해 각 레이블에 확률을 할당하는 확률 지표 기능을 제공합니다. 이를 통해 예측된 결과를 기반으로 정보에 입각한 결정을 더 많이 내릴 수 있습니다. Amazon Redshift ML에서는 문제 유형이 이진 분류 또는 다중 클래스 분류인 AUTO ON 모델을 생성할 때 확률 지표를 사용할 수 있습니다. AUTO ON 파라미터를 생략하면 Redshift ML은 모델이 AUTO ON이어야 한다고 가정합니다.

모델 생성

모델을 생성할 때 Amazon Redshift는 모델 유형과 문제 유형을 자동으로 감지합니다. 분류 문제인 경우 Redshift는 각 레이블과 관련된 확률을 출력하는 데 사용할 수 있는 두 번째 추론 함수를 자동으로 생성합니다. 이 두 번째 추론 함수의 이름은 지정된 추론 함수 이름 뒤에 오는 `_probabilities` 문자열입니다. 예를 들어 추론 함수의 이름을 `customer_churn_predict`로 지정하면 두 번째 추론 함수의 이름은 `customer_churn_predict_probabilities`입니다. 그런 다음 이 함수를 쿼리하여 각 레이블의 확률을 가져올 수 있습니다.

```
CREATE MODEL customer_churn_model
FROM customer_activity
    PROBLEM_TYPE BINARY_CLASSIFICATION
TARGET churn
FUNCTION customer_churn_predict
IAM_ROLE {default}
AUTO ON
SETTINGS ( S3_BUCKET 'amzn-s3-demo-bucket'
```

확률 가져오기

확률 함수가 준비되면 명령을 실행하면 반환된 확률 및 관련 레이블의 배열이 포함된 SUPER 형식이 반환됩니다. 예를 들어 `"probabilities" : [0.7, 0.3], "labels" : ["False.", "True."]` 결과는 False 레이블의 확률이 0.7이고 True 레이블의 확률이 0.3임을 의미합니다.

```
SELECT customer_churn_predict_probabilities(Account_length, Area_code,
```

```
            VMail_message, Day_mins, Day_calls, Day_charge,Eve_mins, Eve_calls,
            Eve_charge, Night_mins, Night_calls, Night_charge,Intl_mins, Intl_calls,
            Intl_charge, Cust_serv_calls)
FROM customer_activity;

customer_churn_predict_probabilities
 --------------------
 {"probabilities" : [0.7, 0.3], "labels" : ["False.", "True."]}
 {"probabilities" : [0.8, 0.2], "labels" : ["False.", "True."]}
 {"probabilities" : [0.75, 0.25], "labels" : ["True.", "False"]}
```

확률 및 레이블 배열은 항상 확률에 따라 내림차순으로 정렬됩니다. 확률 함수의 SUPER 반환 결과를 중첩 해제하여 확률이 가장 높은 예측 레이블만 반환하는 쿼리를 작성할 수 있습니다.

```
SELECT prediction.labels[0], prediction.probabilities[0]
            FROM (SELECT customer_churn_predict_probabilities(Account_length,
 Area_code,
            VMail_message, Day_mins, Day_calls, Day_charge,Eve_mins, Eve_calls,
            Eve_charge, Night_mins, Night_calls, Night_charge,Intl_mins, Intl_calls,
            Intl_charge, Cust_serv_calls) AS prediction
FROM customer_activity);

  labels   | probabilities
-----------+--------------
 "False."  | 0.7
 "False."  | 0.8
 "True."   | 0.75
```

쿼리를 더 간단하게 만들기 위해 예측 함수의 결과를 테이블에 저장할 수 있습니다.

```
CREATE TABLE churn_auto_predict_probabilities AS
             (SELECT customer_churn_predict_probabilities(Account_length, Area_code,
             VMail_message, Day_mins, Day_calls, Day_charge,Eve_mins, Eve_calls,
             Eve_charge, Night_mins, Night_calls, Night_charge,Intl_mins,
             Intl_calls, Intl_charge, Cust_serv_calls) AS prediction
FROM customer_activity);
```

결과가 포함된 테이블을 쿼리하여 확률이 0.7보다 높은 예측만 반환할 수 있습니다.

```
SELECT prediction.labels[0], prediction.probabilities[0]
FROM churn_auto_predict_probabilities
WHERE prediction.probabilities[0] > 0.7;
```

```
  labels   | probabilities
-----------+--------------
 "False."  | 0.8
 "True."   | 0.75
```

색인 표기를 사용하면 특정 레이블의 확률을 얻을 수 있습니다. 다음 예제에서는 모든 `True.` 레이블의 확률을 반환합니다.

```
SELECT label, index, p.prediction.probabilities[index]
FROM churn_auto_predict_probabilities p, p.prediction.labels AS label AT index
WHERE label='True.';

  label  | index | probabilities
---------+-------+---------------
 "True." |     0 | 0.3
 "True." |     0 | 0.2
 "True." |     0 | 0.75
```

다음 예에서는 `True`가 있는 모든 행을 반환합니다. 0.7보다 큰 확률로 레이블을 지정하여 고객이 이탈할 가능성이 있음을 나타냅니다.

```
SELECT prediction.labels[0], prediction.probabilities[0]
FROM churn_auto_predict_probabilities
WHERE prediction.probabilities[0] > 0.7 AND prediction.labels[0] = "True.";

labels     | probabilities
-----------+--------------
 "True."   | 0.75
```

Amazon Redshift ML 튜토리얼

Amazon Redshift ML을 사용하면 SQL 스테이트먼트로 기계 학습 모델을 훈련하고 예측을 위해 SQL 쿼리에서 해당 기계 학습 모델을 호출할 수 있습니다. Amazon Redshift의 기계 학습은 하나의 SQL 명령으로 모델을 훈련합니다. Amazon Redshift는 자동으로 Amazon SageMaker AI에서 훈련 작업을 시작하고 모델을 생성합니다. 모델이 생성되면 모델의 예측 함수를 사용하여 Amazon Redshift에서 예측을 수행할 수 있습니다.

이러한 튜토리얼의 단계에 따라 Amazon Redshift ML의 기능을 알아보세요.

- 튜토리얼: 고객 이탈 모델 구축 – 이 튜토리얼에서는 Amazon Redshift ML을 사용하여 CREATE MODEL 명령으로 고객 이탈 모델을 생성하고 사용자 시나리오에 대한 예측 쿼리를 실행합니다. 그런 다음 CREATE MODEL 명령이 생성하는 SQL 함수를 사용하여 쿼리를 구현합니다.
- 튜토리얼: 원격 추론 모델 구축 – 이 튜토리얼에서는 이전에 Amazon Redshift 외부의 Amazon SageMaker AI에서 훈련 및 배포한 Random Cut Forest 모델을 생성하는 방법을 단계별로 살펴봅니다.
- 튜토리얼: K-평균 클러스터링 모델 구축 – 이 튜토리얼에서는 Amazon Redshift ML을 사용하여 K-평균 알고리즘을 바탕으로 기계 학습 모델을 생성, 훈련 및 배포합니다.
- 튜토리얼: 다중 클래스 분류 모델 구축 – 이 튜토리얼에서는 Amazon Redshift ML을 사용하여 다중 클래스 분류 문제를 해결하는 기계 학습 모델을 생성합니다. 다중 클래스 분류 알고리즘은 데이터 포인트를 세 개 이상의 클래스 중 하나로 분류합니다. 그런 다음 CREATE MODEL 명령이 생성하는 SQL 함수를 사용하여 쿼리를 구현합니다.
- 튜토리얼: XGBoost 모델 구축 – 이 튜토리얼에서는 Amazon S3의 데이터로 모델을 생성하고 Amazon Redshift ML을 사용하여 이 모델로 예측 쿼리를 실행합니다. XGBoost 알고리즘은 그래디언트 부스트 트리 알고리즘을 최적화한 것입니다.
- 튜토리얼: 회귀 모델 구축 – 이 튜토리얼에서는 Amazon Redshift ML을 사용하여 기계 학습 회귀 모델을 생성하고 모델에서 예측 쿼리를 실행합니다. 회귀 모델을 사용하면 주택 가격이나 도시의 자전거 대여 서비스를 이용할 사람 수와 같은 수치 결과를 예측할 수 있습니다.
- 튜토리얼: 선형 학습기를 사용하여 회귀 모델 구축 – 이 튜토리얼에서는 Amazon S3의 데이터로 선형 학습기 모델을 생성하고 Amazon Redshift ML을 사용하여 이 모델로 예측 쿼리를 실행합니다. SageMaker AI 선형 학습기 알고리즘은 회귀 또는 멀티클래스 분류 문제를 해결합니다.
- 튜토리얼: 선형 학습기를 사용하여 다중 클래스 분류 모델 구축 – 이 튜토리얼에서는 Amazon S3의 데이터로 선형 학습기 모델을 생성하고 Amazon Redshift ML을 사용하여 이 모델로 예측 쿼리를 실행합니다. SageMaker AI 선형 학습기 알고리즘은 회귀 또는 분류 문제를 해결합니다.

튜토리얼: 고객 이탈 모델 구축

이 튜토리얼에서는 Amazon Redshift ML을 사용하여 CREATE MODEL 명령으로 고객 이탈 모델을 생성하고 사용자 시나리오에 대한 예측 쿼리를 실행합니다. 그런 다음 CREATE MODEL 명령이 생성하는 SQL 함수를 사용하여 쿼리를 구현합니다.

간단한 CREATE MODEL 명령을 사용하여 훈련 데이터를 내보내고, 모델을 훈련하고, 모델을 가져오고, Amazon Redshift 예측 함수를 준비할 수 있습니다. 테이블 또는 SELECT 문으로 훈련 데이터를 지정하려면 CREATE MODEL 문을 사용합니다.

이 예에서는 기록 정보를 사용하여 한 통신사의 고객 이탈에 대한 기계 학습 모델을 구성합니다. 먼저 SageMaker AI가 기계 학습 모델을 훈련시킨 다음 임의의 고객 프로필 정보를 사용하여 모델을 테스트합니다. 모델을 검증한 후 Amazon SageMaker AI가 모델과 예측 함수를 Amazon Redshift에 배포합니다. 예측 함수를 사용하여 고객의 이탈 여부를 예측할 수 있습니다.

사용 사례

Amazon Redshift ML을 사용하여 영업 리드의 마감 여부 예측과 같은 다른 바이너리 분류 문제를 해결할 수 있습니다. 금융 거래가 사기인지 아닌지도 예측할 수 있습니다.

업무

- 사전 조건
- 1단계: Amazon S3에서 Amazon Redshift로 데이터 로드
- 2단계: 기계 학습 모델 생성
- 3단계: 모델을 사용하여 예측 수행

사전 조건

이 튜토리얼을 완료하려면 다음과 같은 사전 조건이 필요합니다.

- Amazon Redshift ML에 대해 Amazon Redshift 클러스터를 설정해야 합니다. 설정을 위해 Amazon Redshift 기계 학습 관리를 위한 클러스터 및 구성 설정 설명서를 참조하세요.
- 모델을 생성하는 데 사용하는 Amazon Redshift 클러스터와 훈련 데이터를 스테이징하고 모델 아티팩트를 준비하는 데 사용하는 Amazon S3 버킷은 동일한 AWS 리전에 있어야 합니다.
- 이 설명서에 사용된 SQL 명령과 샘플 데이터 세트를 다운로드하려면 다음 중 하나를 수행하세요.
  - SQL 명령, 고객 활동 파일 및 Abalone 파일을 다운로드합니다.
  - Amazon S3용 AWS CLI를 사용하여 다음 명령을 실행합니다. 고유의 대상 경로를 사용할 수 있습니다.

```
aws s3 cp s3://redshift-downloads/redshift-ml/tutorial-scripts/redshift-ml-
tutorial.sql </target/path>
aws s3 cp s3://redshift-downloads/redshift-ml/customer_activity/
customer_activity.csv </target/path>
aws s3 cp s3://redshift-downloads/redshift-ml/abalone_xgb/abalone_xgb.csv </target/
path>
```

1단계: Amazon S3에서 Amazon Redshift로 데이터 로드

Amazon Redshift 쿼리 편집기 v2를 사용하여 쿼리를 편집 및 실행하고 결과를 시각화합니다.

다음 쿼리를 실행하면 customer_activity라는 테이블이 생성되며 Amazon S3에서 샘플 데이터 세트를 수집합니다.

```
DROP TABLE IF EXISTS customer_activity;

CREATE TABLE customer_activity (
state varchar(2),
account_length int,
area_code int,
phone varchar(8),
intl_plan varchar(3),
vMail_plan varchar(3),
vMail_message int,
day_mins float,
day_calls int,
day_charge float,
total_charge float,
eve_mins float,
eve_calls int,
eve_charge float,
night_mins float,
night_calls int,
night_charge float,
intl_mins float,
intl_calls int,
intl_charge float,
cust_serv_calls int,
churn varchar(6),
record_date date
);

COPY customer_activity
FROM 's3://redshift-downloads/redshift-ml/customer_activity/'
REGION 'us-east-1' IAM_ROLE default
FORMAT AS CSV IGNOREHEADER 1;
```

2단계: 기계 학습 모델 생성

이탈은 이 모델에서 목표로 하는 입력입니다. 이 모델에 대한 다른 모든 입력은 이탈을 예측하는 함수를 생성하는 데 도움이 되는 속성입니다.

다음 예에서는 고객의 연령, 우편번호, 지출 및 사례와 같은 입력을 사용하여 CREATE MODEL 작업으로 고객의 활동 여부를 예측하는 모델을 제공합니다. 다음 예제에서는 amzn-s3-demo-bucket을 사용자의 자체 Amazon S3 버킷으로 바꿉니다.

```
CREATE MODEL customer_churn_auto_model
FROM
    (
      SELECT state,
             account_length,
             area_code,
             total_charge/account_length AS average_daily_spend,
             cust_serv_calls/account_length AS average_daily_cases,
             churn
      FROM customer_activity
      WHERE  record_date < '2020-01-01'
     )
TARGET churn FUNCTION ml_fn_customer_churn_auto
IAM_ROLE default SETTINGS (
  S3_BUCKET 'amzn-s3-demo-bucket'
);
```

이전 예의 SELECT 쿼리는 훈련 데이터를 생성합니다. TARGET 절은 CREATE MODEL 작업이 예측 방법을 학습하는 데 사용하는 기계 학습 레이블 열을 지정합니다. 대상 열 "이탈"은 고객이 여전히 활성 멤버십을 보유하고 있는지 아니면 멤버십을 일시중단했는지를 나타냅니다. S3_BUCKET 필드는 사용자가 이전에 생성한 Amazon S3 버킷의 이름입니다. 이 Amazon S3 버킷은 Amazon Redshift와 Amazon SageMaker AI 간에 훈련 데이터와 아티팩트를 공유하는 데 사용됩니다. 나머지 열은 예측에 사용되는 특성입니다.

CREATE MODEL 명령의 간단한 사용 사례를 보여주는 구문 및 특성에 대한 요약은 단순 CREATE MODEL 섹션을 참조하세요.

서버 측 암호화에 대한 권한 추가(선택 사항)

Amazon Redshift는 기본적으로 훈련에 Amazon SageMaker AI Autopilot을 사용합니다. 특히 Amazon Redshift는 고객이 지정한 Amazon S3 버킷에 훈련 데이터를 안전하게 내보냅니다. KMS_KEY_ID를 지정하지 않으면 기본적으로 서버 측 암호화(SSE)를 사용하여 데이터가 암호화됩니다.

AWS KMS 관리형 키(SSE-MMS)로 서버 측 암호화를 사용하여 입력을 암호화할 때 다음 권한을 추가합니다.

```
{
    "Effect": "Allow",
    "Action": [
    "kms:Encrypt"
    "kms:Decrypt"
    ]
}
```

Amazon SageMaker AI 역할에 대한 자세한 내용은 Amazon SageMaker AI Developer Guide의 Amazon SageMaker AI roles 섹션을 참조하세요.

모델 훈련 상태 확인(선택 사항)

SHOW MODEL 명령을 사용하여 모델이 준비되었는지 알 수 있습니다.

다음 작업을 사용하여 모델의 상태를 확인합니다.

```
SHOW MODEL customer_churn_auto_model;
```

다음은 이전 작업 출력의 예시입니다.

```
+--------------------------
+-----------------------------------------------------------------------------------------------------------
+
|           Key             |
                 Value
             |
+--------------------------
+-----------------------------------------------------------------------------------------------------------
+
|        Model Name         |
        customer_churn_auto_model
             |
|       Schema Name         |
                 public
             |
|          Owner            |
                 awsuser
             |
```

```
|      Creation Time       |
        Tue, 14.06.2022 17:15:52
             |
|       Model State        |
                TRAINING
             |
|                          |

             |
|      TRAINING DATA:      |

             |
|          Query           | SELECT STATE, ACCOUNT_LENGTH, AREA_CODE, TOTAL_CHARGE /
 ACCOUNT_LENGTH AS AVERAGE_DAILY_SPEND, CUST_SERV_CALLS / ACCOUNT_LENGTH AS
 AVERAGE_DAILY_CASES, CHURN |
|                          |
         FROM CUSTOMER_ACTIVITY
             |
|                          |
    WHERE RECORD_DATE < '2020-01-01'
             |
|      Target Column       |
                  CHURN
             |
|                          |

             |
|       PARAMETERS:        |

             |
|        Model Type        |
                  auto
             |
|       Problem Type       |

             |
|        Objective         |

             |
|     AutoML Job Name      |
     redshiftml-20220614171552640901
             |
```

```
|      Function Name       |
        ml_fn_customer_churn_auto
             |
|   Function Parameters    |                                                  state
 account_length area_code average_daily_spend average_daily_cases
                       |
| Function Parameter Types |
      varchar int4 int4 float8 int4
             |
|         IAM Role         |
          default-aws-iam-role
             |
|        S3 Bucket         |
              amzn-s3-demo-bucket
               |
|       Max Runtime        |
                  5400
             |
+--------------------------
+-----------------------------------------------------------------------------------------------------------
+
```

모델 학습이 완료되면 `model_state` 변수가 `Model is Ready`가 되며, 예측 함수를 사용할 수 있게 됩니다.

3단계: 모델을 사용하여 예측 수행

SQL 스테이트먼트를 사용하여 예측 모델에서 수행한 예측을 볼 수 있습니다. 이 예에서 CREATE MODEL 작업으로 생성된 예측 함수의 이름은 `ml_fn_customer_churn_auto`입니다. 예측 함수에 대한 입력 인수는 `state`의 경우 varchar, `account_length`의 경우 integer와 같이 특성의 유형과 일치합니다. 예측 함수의 출력은 CREATE MODEL 문의 TARGET 열과 동일한 형식입니다.

1. 2020-01-01 이전의 데이터에 대해 모델을 훈련시켰으므로 이제 테스트 세트에 예측 함수를 사용합니다. 다음 쿼리는 2020-01-01 이후에 가입한 고객이 이탈할지를 표시합니다.

```
SELECT
    phone,
    ml_fn_customer_churn_auto(
        state,
        account_length,
        area_code,
        total_charge / account_length,
```

```
        cust_serv_calls / account_length
    ) AS active
FROM
    customer_activity
WHERE
    record_date > '2020-01-01';
```

2. 다음 예에서는 다른 사용 사례에 동일한 예측 함수를 사용합니다. 이 경우 Amazon Redshift는 기록 날짜가 2020-01-01보다 큰 여러 주의 고객 중 이탈자 및 비이탈자 비율을 예측합니다.

```
WITH predicted AS (
    SELECT
        state,
        ml_fn_customer_churn_auto(
            state,
            account_length,
            area_code,
            total_charge / account_length,
            cust_serv_calls / account_length
        ) :: varchar(6) AS active
    FROM
        customer_activity
    WHERE
        record_date > '2020-01-01'
)
SELECT
    state,
    SUM(
        CASE
            WHEN active = 'True.' THEN 1
            ELSE 0
        END
    ) AS churners,
    SUM(
        CASE
            WHEN active = 'False.' THEN 1
            ELSE 0
        END
    ) AS nonchurners,
    COUNT(*) AS total_per_state
FROM
    predicted
GROUP BY
```

```
    state
ORDER BY
    state;
```

3. 다음 예에서는 한 주에서 이탈하는 고객의 비율을 예측하는 사용 사례에 예측 함수를 사용합니다. 이 경우 Amazon Redshift는 기록 날짜가 2020-01-01보다 클 때의 이탈률을 예측합니다.

```
WITH predicted AS (
    SELECT
        state,
        ml_fn_customer_churn_auto(
            state,
            account_length,
            area_code,
            total_charge / account_length,
            cust_serv_calls / account_length
        ) :: varchar(6) AS active
    FROM
        customer_activity
    WHERE
        record_date > '2020-01-01'
)
SELECT
    state,
    CAST((CAST((SUM(
        CASE
            WHEN active = 'True.' THEN 1
            ELSE 0
        END
    )) AS FLOAT) / CAST(COUNT(*) AS FLOAT)) AS DECIMAL (3, 2)) AS pct_churn,
    COUNT(*) AS total_customers_per_state
FROM
    predicted
GROUP BY
    state
ORDER BY
    3 DESC;
```

관련 주제

Amazon Redshift ML에 대한 자세한 내용은 다음 설명서를 참조하세요.

- Amazon Redshift 기계 학습 사용 비용
- CREATE MODEL 명령
- EXPLAIN_MODEL 함수

기계 학습에 대한 자세한 내용은 다음 설명서를 참조하세요.

- 기계 학습 개요
- 초보자 및 전문가를 위한 기계 학습
- What Is Fairness and Model Explainability for Machine Learning Predictions?(기계 학습 예측을 위한 공정성과 모델 설명 가능성이란 무엇입니까?)

튜토리얼: 원격 추론 모델 구축

이 튜토리얼에서는 이전에 Amazon Redshift 외부의 Amazon SageMaker AI에서 훈련 및 배포한 Random Cut Forest 모델을 생성하는 방법을 단계별로 살펴봅니다. Random Cut Forest 알고리즘은 데이터 세트 내에서 이상한 데이터 포인트를 감지합니다. 원격 추론을 사용하여 모델을 생성하면 Random Cut Forest SageMaker AI 모델을 Amazon Redshift로 가져올 수 있습니다. 그런 다음 Amazon Redshift에서 SQL을 사용하여 원격 SageMaker AI 엔드포인트에 대한 예측을 수행합니다.

CREATE MODEL 명령을 사용하여 Amazon SageMaker AI 엔드포인트에서 기계 학습 모델을 가져오고 Amazon Redshift 예측 함수를 준비할 수 있습니다. CREATE MODEL 작업을 사용할 때 SageMaker AI 기계 학습 모델의 엔드포인트 이름을 제공합니다.

이 튜토리얼에서는 SageMaker AI 모델 엔드포인트를 사용하여 Amazon Redshift 기계 학습 모델을 생성합니다. 기계 학습 모델이 준비되면 이를 사용하여 Amazon Redshift에서 예측을 수행할 수 있습니다. 먼저 Amazon SageMaker AI에서 엔드포인트를 훈련하고 생성한 다음 엔드포인트 이름을 얻습니다. 그런 다음 CREATE MODEL 명령을 사용하여 Amazon Redshift ML을 통해 모델을 생성합니다. 마지막으로 CREATE MODEL 명령이 생성하는 예측 함수를 사용하여 모델에 대한 예측을 수행합니다.

사용 사례

Random Cut Forest 모델과 원격 추론을 사용하여 다른 데이터 세트에서 전자 상거래 트랜잭션의 급격한 증가 또는 감소 예측과 같은 이상 징후를 감지할 수 있습니다. 날씨 또는 지진 활동의 중대한 변화도 예측할 수 있습니다.

업무

- 사전 조건
- 1단계: Amazon SageMaker AI 모델 배포
- 2단계: SageMaker AI 모델 엔드포인트 가져오기
- 3단계: Amazon S3에서 Amazon Redshift로 데이터 로드
- 4단계: Amazon Redshift ML을 사용하여 모델 생성
- 5단계: 모델을 사용하여 예측 수행

사전 조건

이 튜토리얼을 완료하려면 다음과 같은 사전 조건이 필요합니다.

- Amazon Redshift ML의 관리 설정을 완료한 상태여야 합니다.
- NYC 택시 데이터 세트를 다운로드하고, Amazon S3 버킷을 생성하고, Amazon S3 버킷에 데이터를 업로드한 상태여야 합니다.
- SageMaker AI 모델 및 엔드포인트를 훈련하고 배포하고 SageMaker AI 엔드포인트의 이름을 가져와야 합니다. 이 AWS CloudFormation 템플릿을 사용하여 AWS 계정에 있는 모든 SageMaker AI 리소스를 자동으로 프로비저닝합니다.

1단계: Amazon SageMaker AI 모델 배포

1. 모델을 배포하려면 Amazon SageMaker AI 콘솔로 이동하여 탐색 창의 노트북에서 노트북 인스턴스를 선택합니다.
2. CloudFormation 템플릿으로 생성된 Jupyter Notebook에서 Open Jupyter(Jupyter 열기)를 선택합니다.
3. `bring-your-own-model-remote-inference.ipynb`를 선택합니다.
4. 다음 줄을 Amazon S3 버킷과 접두사로 대체하여 Amazon S3에 훈련 입력 및 출력을 저장하도록 파라미터를 설정합니다.

```
data_location=f"s3://{bucket}/{prefix}/",
output_path=f"s3://{bucket}/{prefix}/output",
```

5. Fast-forward(빨리 감기) 버튼을 눌러 모든 셀을 실행합니다.

2단계: SageMaker AI 모델 엔드포인트 가져오기

Amazon SageMaker AI 콘솔 탐색창의 추론에서 엔드포인트를 눌러 모델 이름을 찾습니다. Amazon Redshift에서 원격 추론 모델을 생성할 때 모델의 엔드포인트 이름을 복사해야 합니다.

3단계: Amazon S3에서 Amazon Redshift로 데이터 로드

Amazon Redshift 쿼리 편집기 v2를 사용하여 Amazon Redshift에서 다음 SQL 명령을 실행합니다. 이 명령은 `rcf_taxi_data` 테이블이 있는 경우 이 테이블을 삭제하고 같은 이름의 테이블을 생성한 후 샘플 데이터 세트를 테이블로 로드합니다.

```
DROP TABLE IF EXISTS public.rcf_taxi_data CASCADE;

CREATE TABLE public.rcf_taxi_data (ride_timestamp timestamp, nbr_passengers int);

COPY public.rcf_taxi_data
FROM
    's3://sagemaker-sample-files/datasets/tabular/anomaly_benchmark_taxi/
NAB_nyc_taxi.csv'
    IAM_ROLE default
    IGNOREHEADER 1
    FORMAT AS CSV;
```

4단계: Amazon Redshift ML을 사용하여 모델 생성

다음 쿼리를 실행하여 이전 단계에서 얻은 SageMaker AI 모델 엔드포인트를 사용하여 Amazon Redshift ML에 모델을 생성합니다. *randomcutforest-xxxxxxxxx*를 사용자의 SageMaker AI 엔드포인트 이름으로 대체합니다.

```
CREATE MODEL public.remote_random_cut_forest
FUNCTION remote_fn_rcf(int)
RETURNS decimal(10, 6) SAGEMAKER '<randomcutforest-xxxxxxxxx>' IAM_ROLE default;
```

모델 상태 확인(선택 사항)

SHOW MODEL 명령을 사용하여 모델이 준비되었는지 알 수 있습니다.

모델 상태를 확인하려면 다음 SHOW MODEL 작업을 사용합니다.

```
SHOW MODEL public.remote_random_cut_forest
```

출력에 SageMaker AI 엔드포인트와 함수 이름이 표시됩니다.

```
+--------------------------+-----------------------------------------+
|        Model Name        |       remote_random_cut_forest          |
+--------------------------+-----------------------------------------+
|       Schema Name        |                 public                  |
|          Owner           |                 awsuser                 |
|      Creation Time       |       Wed, 15.06.2022 17:58:21          |
|       Model State        |                  READY                  |
|                          |                                         |
|       PARAMETERS:        |                                         |
|         Endpoint         |     <randomcutforest-xxxxxxxxx>         |
|      Function Name       |              remote_fn_rcf              |
|      Inference Type      |                 Remote                  |
| Function Parameter Types |                  int4                   |
|         IAM Role         |          default-aws-iam-role           |
+--------------------------+-----------------------------------------+
```

5단계: 모델을 사용하여 예측 수행

Amazon SageMaker AI Random Cut Forest 알고리즘은 데이터세트 내에서 이상한 데이터 포인트를 감지하도록 설계되었습니다. 이 예에서 모델은 중요한 행사로 인한 택시 이용의 급증을 감지하도록 설계되었습니다. 이 모델을 사용하여 각 데이터 포인트에 대한 이상 점수를 생성하면 이상 이벤트를 예측할 수 있습니다.

다음 쿼리를 사용하여 전체 택시 데이터 세트의 이상 점수를 계산합니다. 이전 단계에서 CREATE MODEL 스테이트먼트에 사용한 함수를 참조한다는 점에 유의하세요.

```
SELECT
    ride_timestamp,
    nbr_passengers,
    public.remote_fn_rcf(nbr_passengers) AS score
FROM
    public.rcf_taxi_data;
```

높음 및 낮음 이상 여부 확인(선택 사항)

다음 쿼리를 실행하여 점수가 평균 점수에서 3표준편차보다 큰 데이터 요소를 찾습니다.

```
WITH score_cutoff AS (
    SELECT
```

```
        STDDEV(public.remote_fn_rcf(nbr_passengers)) AS std,
        AVG(public.remote_fn_rcf(nbr_passengers)) AS mean,
        (mean + 3 * std) AS score_cutoff_value
    FROM
        public.rcf_taxi_data
)
SELECT
    ride_timestamp,
    nbr_passengers,
    public.remote_fn_rcf(nbr_passengers) AS score
FROM
    public.rcf_taxi_data
WHERE
    score > (
        SELECT
            score_cutoff_value
        FROM
            score_cutoff
    )
ORDER BY
    2 DESC;
```

다음 쿼리를 실행하여 점수가 평균 점수에서 3표준편차보다 큰 데이터 요소를 찾습니다.

```
WITH score_cutoff AS (
    SELECT
        STDDEV(public.remote_fn_rcf(nbr_passengers)) AS std,
        AVG(public.remote_fn_rcf(nbr_passengers)) AS mean,
        (mean - 3 * std) AS score_cutoff_value
    FROM
        public.rcf_taxi_data
)
SELECT
    ride_timestamp,
    nbr_passengers,
    public.remote_fn_rcf(nbr_passengers) AS score
FROM
    public.rcf_taxi_data
WHERE
    score < (
        SELECT
            score_cutoff_value
        FROM
```

```
            score_cutoff
    )
ORDER BY
    2 DESC;
```

관련 주제

Amazon Redshift ML에 대한 자세한 내용은 다음 설명서를 참조하세요.

- Amazon Redshift 기계 학습 사용 비용
- CREATE MODEL 작업
- EXPLAIN_MODEL 함수

기계 학습에 대한 자세한 내용은 다음 설명서를 참조하세요.

- 기계 학습 개요
- 초보자 및 전문가를 위한 기계 학습
- What Is Fairness and Model Explainability for Machine Learning Predictions?(기계 학습 예측을 위한 공정성과 모델 설명 가능성이란 무엇입니까?)

튜토리얼: K-평균 클러스터링 모델 구축

이 튜토리얼에서는 Amazon Redshift ML을 사용하여 K-평균 알고리즘을 바탕으로 기계 학습 모델을 생성, 훈련 및 배포합니다. 이 알고리즘은 데이터에서 그룹을 검색하려는 클러스터링 문제를 해결합니다. K-평균은 아직 레이블이 지정되지 않은 데이터를 그룹화하는 데 도움이 됩니다. K-평균 클러스터링에 대한 자세한 내용은 Amazon SageMaker AI Developer Guide의 How K-means Clustering Works 섹션을 참조하세요.

CREATE MODEL 작업을 사용하여 Amazon Redshift 클러스터에서 K-평균 모델을 생성합니다. CREATE MODEL 명령을 사용하여 훈련 데이터를 내보내고, 모델을 훈련하고, 모델을 가져오고, Amazon Redshift 예측 함수를 준비할 수 있습니다. CREATE MODEL 스테이트먼트를 사용하여 테이블 또는 SELECT 작업으로 훈련 데이터를 지정합니다.

이 튜토리얼에서는 전 세계의 세계 뉴스를 모니터링하고 데이터가 매일 매초마다 저장되는 Global Database of Events, Language and Tone(GDELT) 데이터 세트에 K-평균을 사용합니다. K-평균은 비슷한 어조, 행위자 또는 위치를 가진 이벤트를 그룹화합니다. 데이터는 Amazon Simple Storage Service에서 두 개의 서로 다른 폴더에 여러 파일로 저장됩니다. 폴더는 1979년부터 2013년까지의 기

록 폴더와 2013년 이후의 일일 업데이트 폴더입니다. 이 예에서는 기록 형식을 사용하여 1979년 데이터를 가져옵니다.

사용 사례

스트리밍 서비스에서 시청 습관이 비슷한 고객을 그룹화하는 등 Amazon Redshift ML을 사용하여 다른 클러스터링 문제를 해결할 수 있습니다. Redshift ML을 사용하여 배송 서비스를 위한 최적의 물류 센터 수를 예측할 수도 있습니다.

업무

- 사전 조건
- 1단계: Amazon S3에서 Amazon Redshift로 데이터 로드
- 2단계: 기계 학습 모델 생성
- 3단계: 모델을 사용하여 예측 수행

사전 조건

이 튜토리얼을 완료하려면 Amazon Redshift ML의 관리 설정을 완료해야 합니다.

1단계: Amazon S3에서 Amazon Redshift로 데이터 로드

1. Amazon Redshift 쿼리 편집기 v2를 사용하여 다음 쿼리를 실행합니다. 이 쿼리는 `gdelt_data` 테이블이 존재하는 경우 퍼블릭 스키마에서 이 테이블을 삭제하고 퍼블릭 스키마에 같은 이름의 테이블을 생성합니다.

```
DROP TABLE IF EXISTS gdelt_data CASCADE;

CREATE TABLE gdelt_data (
    GlobalEventId bigint,
    SqlDate bigint,
    MonthYear bigint,
    Year bigint,
    FractionDate double precision,
    Actor1Code varchar(256),
    Actor1Name varchar(256),
    Actor1CountryCode varchar(256),
    Actor1KnownGroupCode varchar(256),
    Actor1EthnicCode varchar(256),
    Actor1Religion1Code varchar(256),
    Actor1Religion2Code varchar(256),
```

```
    Actor1Type1Code varchar(256),
    Actor1Type2Code varchar(256),
    Actor1Type3Code varchar(256),
    Actor2Code varchar(256),
    Actor2Name varchar(256),
    Actor2CountryCode varchar(256),
    Actor2KnownGroupCode varchar(256),
    Actor2EthnicCode varchar(256),
    Actor2Religion1Code varchar(256),
    Actor2Religion2Code varchar(256),
    Actor2Type1Code varchar(256),
    Actor2Type2Code varchar(256),
    Actor2Type3Code varchar(256),
    IsRootEvent bigint,
    EventCode bigint,
    EventBaseCode bigint,
    EventRootCode bigint,
    QuadClass bigint,
    GoldsteinScale double precision,
    NumMentions bigint,
    NumSources bigint,
    NumArticles bigint,
    AvgTone double precision,
    Actor1Geo_Type bigint,
    Actor1Geo_FullName varchar(256),
    Actor1Geo_CountryCode varchar(256),
    Actor1Geo_ADM1Code varchar(256),
    Actor1Geo_Lat double precision,
    Actor1Geo_Long double precision,
    Actor1Geo_FeatureID bigint,
    Actor2Geo_Type bigint,
    Actor2Geo_FullName varchar(256),
    Actor2Geo_CountryCode varchar(256),
    Actor2Geo_ADM1Code varchar(256),
    Actor2Geo_Lat double precision,
    Actor2Geo_Long double precision,
    Actor2Geo_FeatureID bigint,
    ActionGeo_Type bigint,
    ActionGeo_FullName varchar(256),
    ActionGeo_CountryCode varchar(256),
    ActionGeo_ADM1Code varchar(256),
    ActionGeo_Lat double precision,
    ActionGeo_Long double precision,
    ActionGeo_FeatureID bigint,
```

```
    DATEADDED bigint
);
```

2. 다음 쿼리는 샘플 데이터를 gdelt_data 테이블에 로드합니다.

```
COPY gdelt_data
FROM 's3://gdelt-open-data/events/1979.csv'
REGION 'us-east-1'
IAM_ROLE default
CSV
DELIMITER '\t';
```

훈련 데이터 검사(선택 사항)

모델을 어떤 데이터로 훈련시킬지 보려면 다음 쿼리를 사용합니다.

```
SELECT
    AvgTone,
    EventCode,
    NumArticles,
    Actor1Geo_Lat,
    Actor1Geo_Long,
    Actor2Geo_Lat,
    Actor2Geo_Long
FROM
    gdelt_data LIMIT 100;
```

2단계: 기계 학습 모델 생성

다음 예에서는 CREATE MODEL 명령을 사용하여 데이터를 7개의 클러스터로 그룹화하는 모델을 생성합니다. K 값은 데이터 포인트가 분류되는 클러스터 수입니다. 이 모델은 데이터 포인트를 서로 비슷한 데이터 포인트끼리 모인 클러스터로 분류합니다. 데이터 포인트를 그룹으로 클러스터링함으로써 K-평균 알고리즘은 최상의 클러스터 센터를 반복적으로 결정합니다. 그런 다음 알고리즘은 각 데이터 포인트를 가장 가까운 클러스터 센터에 할당합니다. 같은 클러스터 센터에 가장 가까이 있는 멤버가 같은 그룹에 속합니다. 그룹의 멤버는 같은 그룹의 다른 멤버와 최대한 유사하고 다른 그룹의 멤버와 최대한 다릅니다. K 값은 주관적이며 데이터 포인트 간의 유사성을 측정하는 방법에 따라 달라집니다. 클러스터가 고르지 않게 분포되어 있는 경우 K 값을 변경하여 클러스터 크기를 균일하게 만들 수 있습니다.

다음 예제에서는 amzn-s3-demo-bucket을 사용자의 자체 Amazon S3 버킷으로 바꿉니다.

```
CREATE MODEL news_data_clusters
FROM
    (
        SELECT
            AvgTone,
            EventCode,
            NumArticles,
            Actor1Geo_Lat,
            Actor1Geo_Long,
            Actor2Geo_Lat,
            Actor2Geo_Long
        FROM
            gdelt_data
    ) FUNCTION news_monitoring_cluster
    IAM_ROLE default
    AUTO OFF
    MODEL_TYPE KMEANS
    PREPROCESSORS 'none'
    HYPERPARAMETERS DEFAULT
    EXCEPT
    (K '7')
    SETTINGS (S3_BUCKET 'amzn-s3-demo-bucket');
```

모델 훈련 상태 확인(선택 사항)

SHOW MODEL 명령을 사용하여 모델이 준비되었는지 알 수 있습니다.

모델 상태를 확인하려면 다음 SHOW MODEL 작업을 사용하고 `Model State`가 `Ready`인지 확인합니다.

```
SHOW MODEL NEWS_DATA_CLUSTERS;
```

모델이 준비되면 이전 작업의 출력에 `Model State`가 `Ready`라고 표시됩니다. 다음은 SHOW MODEL 작업 출력의 예시입니다.

```
+--------------------------
+-----------------------------------------------------------------------------------------------------
+
|        Model Name        |
 news_data_clusters                                        |
```

```
+--------------------------
+-----------------------------------------------------------------------------------------------------------
+
|       Schema Name        |                                                 public
                                          |
|          Owner           |                                                awsuser
                                          |
|      Creation Time       |                                       Fri, 17.06.2022
 16:32:19                                      |
|       Model State        |                                                 READY
                                          |
|        train:msd         |                                              2973.822754
                                          |
|      train:progress      |                                               100.000000
                                          |
|     train:throughput     |                                             237114.875000
                                          |
|      Estimated Cost      |                                                0.004983
                                          |
|                          |
                                          |
|      TRAINING DATA:      |
                                          |
|          Query           | SELECT AVGTONE, EVENTCODE, NUMARTICLES, ACTOR1GEO_LAT,
 ACTOR1GEO_LONG, ACTOR2GEO_LAT, ACTOR2GEO_LONG |
|                          |                                            FROM GDELT_DATA
                                          |
|                          |
                                          |
|       PARAMETERS:        |
                                          |
|        Model Type        |                                                 kmeans
                                          |
|    Training Job Name     |
 redshiftml-20220617163219978978-kmeans                                 |
|      Function Name       |
 news_monitoring_cluster                                      |
|   Function Parameters    |       avgtone eventcode numarticles actor1geo_lat
 actor1geo_long actor2geo_lat actor2geo_long        |
| Function Parameter Types |                              float8 int8 int8 float8 float8
 float8 float8                            |
|         IAM Role         |                                      default-aws-iam-
role                                        |
```

```
|        S3 Bucket          |                                                amzn-s3-demo-
bucket                                    |
|       Max Runtime         |                                                     5400
                                        |
|                           |
                                        |
|     HYPERPARAMETERS:      |
                                        |
|       feature_dim         |                                                      7
                                        |
|            k              |                                                      7
                                        |
+---------------------------
+-----------------------------------------------------------------------------------------------
+
```

3단계: 모델을 사용하여 예측 수행

클러스터 식별

모델이 데이터에서 별개의 그룹, 즉 클러스터를 식별해 낸 것을 볼 수 있습니다. 클러스터는 다른 클러스터 센터보다 해당 클러스터 센터에 더 가까운 데이터 포인트의 집합입니다. K 값은 모델의 클러스터 수를 나타내므로 클러스터 센터의 수를 나타내기도 합니다. 다음 쿼리는 각 globaleventid와 연관된 클러스터를 표시하여 클러스터를 식별합니다.

```
SELECT
    globaleventid,
    news_monitoring_cluster (
        AvgTone,
        EventCode,
        NumArticles,
        Actor1Geo_Lat,
        Actor1Geo_Long,
        Actor2Geo_Lat,
        Actor2Geo_Long
    ) AS cluster
FROM
    gdelt_data;
```

데이터 분포 확인

클러스터 간의 데이터 분포를 확인하여 선택한 K 값으로 인해 데이터가 어느 정도 균등하게 분포되었는지 확인할 수 있습니다. 다음 쿼리를 사용하여 데이터가 클러스터 전체에 고르게 분포되어 있는지 확인합니다.

```
SELECT
    events_cluster,
    COUNT(*) AS nbr_events
FROM
    (
        SELECT
            globaleventid,
            news_monitoring_cluster(
                AvgTone,
                EventCode,
                NumArticles,
                Actor1Geo_Lat,
                Actor1Geo_Long,
                Actor2Geo_Lat,
                Actor2Geo_Long
            ) AS events_cluster
        FROM
            gdelt_data
    )
GROUP BY
    1;
```

클러스터가 고르지 않게 분포되어 있는 경우 K 값을 변경하여 클러스터 크기를 균일하게 만들 수 있습니다.

클러스터 센터 결정

데이터 포인트는 다른 클러스터 센터보다 자신이 속한 클러스터 센터에 더 가깝습니다. 따라서 클러스터 센터를 찾으면 클러스터를 정의하는 데 도움이 됩니다.

다음 쿼리를 실행하여 이벤트 코드별 기사 수를 기반으로 클러스터 센터를 확인합니다.

```
SELECT
    news_monitoring_cluster (
        AvgTone,
        EventCode,
        NumArticles,
```

```
        Actor1Geo_Lat,
        Actor1Geo_Long,
        Actor2Geo_Lat,
        Actor2Geo_Long
    ) AS events_cluster,
    eventcode,
    SUM(numArticles) AS numArticles
FROM
    gdelt_data
GROUP BY
    1,
    2;
```

클러스터의 데이터 포인트에 대한 정보 표시

다음 쿼리를 사용하여 다섯 번째 클러스터에 할당된 포인트에 대한 데이터를 반환합니다. 선택한 기사에는 두 명의 행위자가 있어야 합니다.

```
SELECT
    news_monitoring_cluster (
        AvgTone,
        EventCode,
        NumArticles,
        Actor1Geo_Lat,
        Actor1Geo_Long,
        Actor2Geo_Lat,
        Actor2Geo_Long
    ) AS events_cluster,
    eventcode,
    actor1name,
    actor2name,
    SUM(numarticles) AS totalarticles
FROM
    gdelt_data
WHERE
    events_cluster = 5
    AND actor1name <> ' '
    AND actor2name <> ' '
GROUP BY
    1,
    2,
    3,
    4
```

```
ORDER BY
    5 desc;
```

동일한 민족 코드의 행위자가 있는 이벤트에 대한 데이터 표시

다음 쿼리는 긍정적인 어조로 이벤트에 대해 작성된 기사 수를 계산합니다. 또한 두 행위자의 민족 코드가 같아야 하며 각 이벤트가 할당된 클러스터를 반환합니다.

```
SELECT
    news_monitoring_cluster (
        AvgTone,
        EventCode,
        NumArticles,
        Actor1Geo_Lat,
        Actor1Geo_Long,
        Actor2Geo_Lat,
        Actor2Geo_Long
    ) AS events_cluster,
    SUM(numarticles) AS total_articles,
    eventcode AS event_code,
    Actor1EthnicCode AS ethnic_code
FROM
    gdelt_data
WHERE
    Actor1EthnicCode = Actor2EthnicCode
    AND Actor1EthnicCode <> ' '
    AND Actor2EthnicCode <> ' '
    AND AvgTone > 0
GROUP BY
    1,
    3,
    4
HAVING
    (total_articles) > 4
ORDER BY
    1,
    2 ASC;
```

관련 주제

Amazon Redshift ML에 대한 자세한 내용은 다음 설명서를 참조하세요.

- Amazon Redshift 기계 학습 사용 비용

- CREATE MODEL 작업
- EXPLAIN_MODEL 함수

기계 학습에 대한 자세한 내용은 다음 설명서를 참조하세요.

- 기계 학습 개요
- 초보자 및 전문가를 위한 기계 학습
- What Is Fairness and Model Explainability for Machine Learning Predictions?(기계 학습 예측을 위한 공정성과 모델 설명 가능성이란 무엇입니까?)

튜토리얼: 다중 클래스 분류 모델 구축

이 튜토리얼에서는 Amazon Redshift ML을 사용하여 다중 클래스 분류 문제를 해결하는 기계 학습 모델을 생성합니다. 다중 클래스 분류 알고리즘은 데이터 포인트를 세 개 이상의 클래스 중 하나로 분류합니다. 그런 다음 CREATE MODEL 명령이 생성하는 SQL 함수를 사용하여 쿼리를 구현합니다.

CREATE MODEL 명령을 사용하여 훈련 데이터를 내보내고, 모델을 훈련하고, 모델을 가져오고, Amazon Redshift 예측 함수를 준비할 수 있습니다. CREATE MODEL 스테이트먼트를 사용하여 테이블 또는 SELECT 작업으로 훈련 데이터를 지정합니다.

튜토리얼대로 진행하기 위해 온라인 영국 소매업체의 판매 데이터가 들어 있는 E-Commerce Sales Forecast 퍼블릭 데이터 세트를 사용합니다. 생성하는 모델은 특별 고객 충성도 프로그램을 위해 가장 활발한 고객을 타겟팅합니다. 다중 클래스 분류를 사용하면 모델을 사용하여 13개월 동안 고객의 활동 기간이 몇 개월일지 예측할 수 있습니다. 예측 함수는 프로그램 가입을 위해 7개월 이상 활동할 것으로 예상되는 고객을 지정합니다.

사용 사례

Amazon Redshift ML을 사용하면 제품 라인에서 가장 많이 팔리는 제품을 예측하는 등 다른 다중 클래스 분류 문제를 해결할 수 있습니다. 사과, 배 또는 오렌지를 선택하는 등 이미지에 포함된 과일을 예측할 수도 있습니다.

업무

- 사전 조건
- 1단계: Amazon S3에서 Amazon Redshift로 데이터 로드
- 2단계: 기계 학습 모델 생성

- 3단계: 모델을 사용하여 예측 수행

사전 조건

이 튜토리얼을 완료하려면 Amazon Redshift ML의 관리 설정을 완료해야 합니다.

1단계: Amazon S3에서 Amazon Redshift로 데이터 로드

Amazon Redshift 쿼리 편집기 v2를 사용하여 다음 쿼리를 실행합니다. 이 쿼리는 샘플 데이터를 Amazon Redshift로 로드합니다.

1. 다음 예에서는 ecommerce_sales라는 테이블을 생성합니다.

```
CREATE TABLE IF NOT EXISTS ecommerce_sales (
    invoiceno VARCHAR(30),
    stockcode VARCHAR(30),
    description VARCHAR(60),
    quantity DOUBLE PRECISION,
    invoicedate VARCHAR(30),
    unitprice DOUBLE PRECISION,
    customerid BIGINT,
    country VARCHAR(25)
);
```

2. 다음 쿼리는 E-Commerce Sales Forecast dataset의 샘플 데이터를 ecommerce_sales 테이블로 복사합니다.

```
COPY ecommerce_sales
FROM
    's3://redshift-ml-multiclass/ecommerce_data.txt'
IAM_ROLE default
DELIMITER '\t'
IGNOREHEADER 1
REGION 'us-east-1'
MAXERROR 100;
```

데이터 분할

Amazon Redshift ML에서 모델을 생성하면 SageMaker AI가 자동으로 데이터를 훈련 세트와 테스트 세트로 분할하므로 SageMaker AI가 모델 정확도를 판단할 수 있습니다. 이 단계에서 데이터를 수동으로 분할하면 추가 예측 세트를 할당하여 모델의 정확도를 확인할 수 있습니다.

다음 SQL 스테이트먼트를 사용하여 훈련, 검증, 예측을 위해 데이터를 세 개의 세트로 분할합니다.

```
--creates table with all data
CREATE TABLE ecommerce_sales_data AS (
    SELECT
        t1.stockcode,
        t1.description,
        t1.invoicedate,
        t1.customerid,
        t1.country,
        t1.sales_amt,
        CAST(RANDOM() * 100 AS INT) AS data_group_id
    FROM
        (
            SELECT
                stockcode,
                description,
                invoicedate,
                customerid,
                country,
                SUM(quantity * unitprice) AS sales_amt
            FROM
                ecommerce_sales
            GROUP BY
                1,
                2,
                3,
                4,
                5
        ) t1
);

--creates training set
CREATE TABLE ecommerce_sales_training AS (
    SELECT
        a.customerid,
        a.country,
        a.stockcode,
        a.description,
        a.invoicedate,
        a.sales_amt,
        (b.nbr_months_active) AS nbr_months_active
    FROM
        ecommerce_sales_data a
```

```
        INNER JOIN (
            SELECT
                customerid,
                COUNT(
                    DISTINCT(
                        DATE_PART(y, CAST(invoicedate AS DATE)) || '-' || LPAD(
                            DATE_PART(mon, CAST(invoicedate AS DATE)),
                            2,
                            '00'
                        )
                    )
                ) AS nbr_months_active
            FROM
                ecommerce_sales_data
            GROUP BY
                1
        ) b ON a.customerid = b.customerid
    WHERE
        a.data_group_id < 80
);

--creates validation set
CREATE TABLE ecommerce_sales_validation AS (
    SELECT
        a.customerid,
        a.country,
        a.stockcode,
        a.description,
        a.invoicedate,
        a.sales_amt,
        (b.nbr_months_active) AS nbr_months_active
    FROM
        ecommerce_sales_data a
        INNER JOIN (
            SELECT
                customerid,
                COUNT(
                    DISTINCT(
                        DATE_PART(y, CAST(invoicedate AS DATE)) || '-' || LPAD(
                            DATE_PART(mon, CAST(invoicedate AS DATE)),
                            2,
                            '00'
                        )
                    )
```

```
                ) AS nbr_months_active
            FROM
                ecommerce_sales_data
            GROUP BY
                1
        ) b ON a.customerid = b.customerid
    WHERE
        a.data_group_id BETWEEN 80
        AND 90
);

--creates prediction set
CREATE TABLE ecommerce_sales_prediction AS (
    SELECT
        customerid,
        country,
        stockcode,
        description,
        invoicedate,
        sales_amt
    FROM
        ecommerce_sales_data
    WHERE
        data_group_id > 90);
```

2단계: 기계 학습 모델 생성

이 단계에서는 CREATE MODEL 스테이트먼트를 사용하여 다중 클래스 분류를 사용하여 기계 학습 모델을 생성합니다.

다음 쿼리에서는 CREATE MODEL 작업을 사용하여 훈련 세트로 다중 클래스 분류 모델을 생성합니다. amzn-s3-demo-bucket을 자체 Amazon S3 버킷으로 교체합니다.

```
CREATE MODEL ecommerce_customer_activity
FROM
    (
        SELECT
            customerid,
            country,
            stockcode,
            description,
            invoicedate,
            sales_amt,
```

```
            nbr_months_active
        FROM
            ecommerce_sales_training
    ) TARGET nbr_months_active FUNCTION predict_customer_activity IAM_ROLE default
 PROBLEM_TYPE MULTICLASS_CLASSIFICATION SETTINGS (
        S3_BUCKET 'amzn-s3-demo-bucket',
        S3_GARBAGE_COLLECT OFF
    );
```

이 쿼리에서는 문제 유형을 `Multiclass_Classification`으로 지정합니다. 모델에 대해 예측하는 목표 값은 `nbr_months_active`입니다. SageMaker AI가 모델 훈련을 마치면 `predict_customer_activity` 함수를 만듭니다. 이 함수는 Amazon Redshift에서 예측을 수행하는 데 사용합니다.

모델 훈련 상태 표시(선택 사항)

SHOW MODEL 명령을 사용하여 모델이 준비되었는지 알 수 있습니다.

다음 쿼리를 사용하여 모델 상태 및 정확도를 포함한 모델의 다양한 메트릭을 반환합니다.

```
SHOW MODEL ecommerce_customer_activity;
```

모델이 준비되면 이전 작업의 출력에 `Model State`가 `Ready`라고 표시됩니다. 다음은 SHOW MODEL 작업 출력의 예시입니다.

```
+--------------------------
+-----------------------------------------------------------------------------------------------
+
|        Model Name        |
 ecommerce_customer_activity                                |
+--------------------------
+-----------------------------------------------------------------------------------------------
+
|       Schema Name        |                                                  public
                                  |
|          Owner           |                                                  awsuser
                                  |
|      Creation Time       |                                   Fri, 17.06.2022 19:02:15
                                  |
|       Model State        |                                                   READY
                                  |
|   Training Job Status    |
 MaxAutoMLJobRuntimeReached                                 |
```

```
|   validation:accuracy    |                                                0.991280
                                    |
|      Estimated Cost      |                                                7.897689
                                    |
|                          |
                                    |
|      TRAINING DATA:      |
                                    |
|          Query           | SELECT CUSTOMERID, COUNTRY, STOCKCODE, DESCRIPTION,
 INVOICEDATE, SALES_AMT, NBR_MONTHS_ACTIVE |
|                          |                                      FROM
 ECOMMERCE_SALES_TRAINING                                    |
|      Target Column       |                                           NBR_MONTHS_ACTIVE
                                    |
|                          |
                                    |
|       PARAMETERS:        |
                                    |
|        Model Type        |                                             xgboost
                                    |
|       Problem Type       |                                    MulticlassClassification
                                    |
|        Objective         |                                             Accuracy
                                    |
|     AutoML Job Name      |
 redshiftml-20220617190215268770                                    |
|      Function Name       |
 predict_customer_activity                                     |
|   Function Parameters    |                customerid country stockcode description
 invoicedate sales_amt                |
| Function Parameter Types |                            int8 varchar varchar varchar
 varchar float8                          |
|         IAM Role         |                                       default-aws-iam-role
                                    |
|        S3 Bucket         |                                           amzn-s3-demo-
bucket                                       |
|       Max Runtime        |                                              5400
                                    |
+--------------------------
+-----------------------------------------------------------------------------------------------
+
```

3단계: 모델을 사용하여 예측 수행

다음 쿼리는 고객 충성도 프로그램을 이용할 자격이 있는 고객을 보여줍니다. 고객이 최소 7개월 동안 활동할 것으로 예측되면 모델은 해당 고객을 충성도 프로그램에 선택합니다.

```
SELECT
    customerid,
    predict_customer_activity(
        customerid,
        country,
        stockcode,
        description,
        invoicedate,
        sales_amt
    ) AS predicted_months_active
FROM
    ecommerce_sales_prediction
WHERE
    predicted_months_active >= 7
GROUP BY
    1,
    2
LIMIT
    10;
```

검증 데이터에 대해 예측 쿼리 실행(선택 사항)

검증 데이터에 대해 다음 예측 쿼리를 실행하여 모델의 정확도 수준을 확인합니다.

```
SELECT
    CAST(SUM(t1.match) AS decimal(7, 2)) AS predicted_matches,
    CAST(SUM(t1.nonmatch) AS decimal(7, 2)) AS predicted_non_matches,
    CAST(SUM(t1.match + t1.nonmatch) AS decimal(7, 2)) AS total_predictions,
    predicted_matches / total_predictions AS pct_accuracy
FROM
    (
        SELECT
            customerid,
            country,
            stockcode,
            description,
            invoicedate,
            sales_amt,
```

```
            nbr_months_active,
            predict_customer_activity(
                customerid,
                country,
                stockcode,
                description,
                invoicedate,
                sales_amt
            ) AS predicted_months_active,
            CASE
                WHEN nbr_months_active = predicted_months_active THEN 1
                ELSE 0
            END AS match,
            CASE
                WHEN nbr_months_active <> predicted_months_active THEN 1
                ELSE 0
            END AS nonmatch
        FROM
            ecommerce_sales_validation
    )t1;
```

진입하지 못한 고객 수 예측(선택 사항)

다음 쿼리는 5~6개월 동안만 활동할 것으로 예상되는 고객 수를 비교합니다. 이 모델은 이러한 고객이 충성도 프로그램에 선택되지 않을 것으로 예측합니다. 그런 다음 쿼리는 간발의 차로 선택되지 못하는 고객의 수와 충성도 프로그램에 선택될 자격이 있는 것으로 예측되는 고객의 수를 비교합니다. 이 쿼리는 충성도 프로그램의 기준을 낮출지 결정하는 데 사용될 수 있습니다. 또한 프로그램에 간발의 차로 선택되지 않을 것으로 예상되는 고객이 많은지도 판단할 수 있습니다. 그러면 해당 고객에게 활동을 늘려 충성도 프로그램 멤버십을 얻도록 권장할 수 있습니다.

```
SELECT
    predict_customer_activity(
        customerid,
        country,
        stockcode,
        description,
        invoicedate,
        sales_amt
    ) AS predicted_months_active,
    COUNT(customerid)
FROM
    ecommerce_sales_prediction
WHERE
```

```
    predicted_months_active BETWEEN 5 AND 6
GROUP BY
    1
ORDER BY
    1 ASC
LIMIT
    10)
UNION
(SELECT
      NULL AS predicted_months_active,
    COUNT (customerid)
FROM
    ecommerce_sales_prediction
WHERE
    predict_customer_activity(
        customerid,
        country,
        stockcode,
        description,
        invoicedate,
        sales_amt
    ) >=7);
```

관련 주제

Amazon Redshift ML에 대한 자세한 내용은 다음 설명서를 참조하세요.

- Amazon Redshift 기계 학습 사용 비용
- CREATE MODEL 작업
- EXPLAIN_MODEL 함수

기계 학습에 대한 자세한 내용은 다음 설명서를 참조하세요.

- 기계 학습 개요
- 초보자 및 전문가를 위한 기계 학습
- What Is Fairness and Model Explainability for Machine Learning Predictions?(기계 학습 예측을 위한 공정성과 모델 설명 가능성이란 무엇입니까?)

튜토리얼: XGBoost 모델 구축

이 튜토리얼에서는 Amazon S3의 데이터로 모델을 생성하고 Amazon Redshift ML을 사용하여 이 모델로 예측 쿼리를 실행합니다. XGBoost 알고리즘은 그래디언트 부스트 트리 알고리즘을 최적화한 것입니다. XGBoost는 다른 그래디언트 부스트 트리 알고리즘보다 더 많은 데이터 유형, 관계 및 분포를 처리합니다. 회귀, 바이너리 분류, 다중 클래스 분류, 순위 결정 관련 문제에 XGBoost를 사용할 수 있습니다. XGBoost 알고리즘에 대한 자세한 내용은 Amazon SageMaker AI Developer Guide의 XGBoost algorithm 섹션을 참조하세요.

`AUTO OFF` 옵션을 사용한 Amazon Redshift ML `CREATE MODEL` 작업은 현재 XGBoost를 `MODEL_TYPE`으로 지원합니다. 사용 사례에 따라 `CREATE MODEL` 명령의 일부로 목적 및 하이퍼파라미터와 같은 관련 정보를 제공할 수 있습니다.

이 튜토리얼에서는 주어진 지폐가 원본인지 위조인지 예측하는 바이너리 분류 문제인 banknote authentication 데이터 세트를 사용합니다.

사용 사례

Amazon Redshift ML을 사용하여 환자의 건강 또는 질병 유무 예측과 같은 다른 바이너리리 분류 문제를 해결할 수 있습니다. 이메일이 스팸인지 아닌지도 예측할 수 있습니다.

업무

- 사전 조건
- 1단계: Amazon S3에서 Amazon Redshift로 데이터 로드
- 2단계: 기계 학습 모델 생성
- 3단계: 모델을 사용하여 예측 수행

사전 조건

이 튜토리얼을 완료하려면 Amazon Redshift ML의 관리 설정을 완료해야 합니다.

1단계: Amazon S3에서 Amazon Redshift로 데이터 로드

Amazon Redshift 쿼리 편집기 v2를 사용하여 다음 쿼리를 실행합니다.

다음 쿼리는 두 개의 테이블을 생성하고 Amazon S3에서 데이터를 로드한 다음 데이터를 훈련 세트와 테스트 세트로 분할합니다. 훈련 세트를 사용하여 모델을 훈련시키고 예측 함수를 생성합니다. 그런 다음 테스트 세트에서 예측 함수를 테스트합니다.

```
--create training set table
CREATE TABLE banknoteauthentication_train(
    variance FLOAT,
    skewness FLOAT,
    curtosis FLOAT,
    entropy FLOAT,
    class INT
);

--Load into training table
COPY banknoteauthentication_train
FROM
    's3://redshiftbucket-ml-sagemaker/banknote_authentication/train_data/' IAM_ROLE
 default REGION 'us-west-2' IGNOREHEADER 1 CSV;

--create testing set table
CREATE TABLE banknoteauthentication_test(
    variance FLOAT,
    skewness FLOAT,
    curtosis FLOAT,
    entropy FLOAT,
    class INT
);

--Load data into testing table
COPY banknoteauthentication_test
FROM
    's3://redshiftbucket-ml-sagemaker/banknote_authentication/test_data/'
    IAM_ROLE default
    REGION 'us-west-2'
    IGNOREHEADER 1
    CSV;
```

2단계: 기계 학습 모델 생성

다음 쿼리는 이전 단계에서 생성한 훈련 세트에서 Amazon Redshift ML에 XGBoost 모델을 생성합니다. amzn-s3-demo-bucket을 입력 데이터 세트 및 기타 Redshift ML 아티팩트를 저장할 사용자의 S3_BUCKET으로 대체합니다.

```
CREATE MODEL model_banknoteauthentication_xgboost_binary
FROM
    banknoteauthentication_train
```

```
    TARGET class
    FUNCTION func_model_banknoteauthentication_xgboost_binary
    IAM_ROLE default
    AUTO OFF
    MODEL_TYPE xgboost
    OBJECTIVE 'binary:logistic'
    PREPROCESSORS 'none'
    HYPERPARAMETERS DEFAULT
EXCEPT(NUM_ROUND '100')
SETTINGS(S3_BUCKET 'amzn-s3-demo-bucket');
```

모델 훈련 상태 표시(선택 사항)

SHOW MODEL 명령을 사용하여 모델이 준비되었는지 알 수 있습니다.

다음 쿼리를 사용하여 모델 훈련의 진행 상황을 모니터링합니다.

```
SHOW MODEL model_banknoteauthentication_xgboost_binary;
```

모델이 READY인 경우 SHOW MODEL 작업은 다음 출력 예시에서 보여주는 것처럼 `train:error` 메트릭을 제공합니다. `train:error` 메트릭은 소수점 6자리까지 측정하는 모델의 정확도를 측정한 것입니다. 값이 0이면 가장 정확하고 값이 1이면 가장 부정확합니다.

```
+--------------------------+-------------------------------------------------+
|        Model Name        |   model_banknoteauthentication_xgboost_binary   |
+--------------------------+-------------------------------------------------+
| Schema Name              | public                                          |
| Owner                    | awsuser                                         |
| Creation Time            | Tue, 21.06.2022 19:07:35                        |
| Model State              | READY                                           |
| train:error              |                                        0.000000 |
| Estimated Cost           |                                        0.006197 |
|                          |                                                 |
| TRAINING DATA:           |                                                 |
| Query                    | SELECT *                                        |
|                          | FROM "BANKNOTEAUTHENTICATION_TRAIN"             |
| Target Column            | CLASS                                           |
|                          |                                                 |
| PARAMETERS:              |                                                 |
| Model Type               | xgboost                                         |
| Training Job Name        | redshiftml-20220621190735686935-xgboost         |
| Function Name            | func_model_banknoteauthentication_xgboost_binary |
| Function Parameters      | variance skewness curtosis entropy              |
```

```
| Function Parameter Types | float8 float8 float8 float8                          |
| IAM Role                 | default-aws-iam-role                                 |
| S3 Bucket                | amzn-s3-demo-bucket                                  |
| Max Runtime              |                                                5400  |
|                          |                                                      |
| HYPERPARAMETERS:         |                                                      |
| num_round                |                                                 100  |
| objective                | binary:logistic                                      |
+--------------------------+------------------------------------------------------+
```

3단계: 모델을 사용하여 예측 수행

모델의 정확도 확인

다음 예측 쿼리에서는 이전 단계에서 생성한 예측 함수를 사용하여 모델의 정확도를 확인합니다. 테스트 세트에서 이 쿼리를 실행하여 모델이 훈련 세트와 너무 근접하게 대응되지 않도록 합니다. 이러한 근접 대응을 과적합이라고도 하며, 과적합으로 인해 모델이 신뢰할 수 없는 예측을 수행할 수 있습니다.

```
WITH predict_data AS (
    SELECT
        class AS label,
        func_model_banknoteauthentication_xgboost_binary (variance, skewness, curtosis,
 entropy) AS predicted,
        CASE
            WHEN label IS NULL THEN 0
            ELSE label
        END AS actual,
        CASE
            WHEN actual = predicted THEN 1 :: INT
            ELSE 0 :: INT
        END AS correct
    FROM
        banknoteauthentication_test
),
aggr_data AS (
    SELECT
        SUM(correct) AS num_correct,
        COUNT(*) AS total
    FROM
        predict_data
```

```
)
SELECT
    (num_correct :: FLOAT / total :: FLOAT) AS accuracy
FROM
    aggr_data;
```

원본 및 위조 지폐의 양 예측

다음 예측 쿼리는 테스트 세트에 있는 원본 및 위조 지폐의 예상 양을 반환합니다.

```
WITH predict_data AS (
    SELECT
        func_model_banknoteauthentication_xgboost_binary(variance, skewness, curtosis,
 entropy) AS predicted
    FROM
        banknoteauthentication_test
)
SELECT
    CASE
        WHEN predicted = '0' THEN 'Original banknote'
        WHEN predicted = '1' THEN 'Counterfeit banknote'
        ELSE 'NA'
    END AS banknote_authentication,
    COUNT(1) AS count
FROM
    predict_data
GROUP BY
    1;
```

원본 지폐와 위조 지폐에 대한 평균 관찰 값 찾기

다음 예측 쿼리는 테스트 세트에서 원본 및 위조 지폐로 예측되는 지폐에 대한 각 특성의 평균값을 반환합니다.

```
WITH predict_data AS (
    SELECT
        func_model_banknoteauthentication_xgboost_binary(variance, skewness, curtosis,
 entropy) AS predicted,
          variance,
          skewness,
          curtosis,
          entropy
    FROM
```

```
        banknoteauthentication_test
)
SELECT
    CASE
        WHEN predicted = '0' THEN 'Original banknote'
        WHEN predicted = '1' THEN 'Counterfeit banknote'
        ELSE 'NA'
    END AS banknote_authentication,
    TRUNC(AVG(variance), 2) AS avg_variance,
    TRUNC(AVG(skewness), 2) AS avg_skewness,
    TRUNC(AVG(curtosis), 2) AS avg_curtosis,
    TRUNC(AVG(entropy), 2) AS avg_entropy
FROM
    predict_data
GROUP BY
    1
ORDER BY
    2;
```

관련 주제

Amazon Redshift ML에 대한 자세한 내용은 다음 설명서를 참조하세요.

- Amazon Redshift 기계 학습 사용 비용
- CREATE MODEL 작업
- EXPLAIN_MODEL 함수

기계 학습에 대한 자세한 내용은 다음 설명서를 참조하세요.

- 기계 학습 개요
- 초보자 및 전문가를 위한 기계 학습
- What Is Fairness and Model Explainability for Machine Learning Predictions?

튜토리얼: 회귀 모델 구축

이 튜토리얼에서는 Amazon Redshift ML을 사용하여 기계 학습 회귀 모델을 생성하고 모델에 대해 예측 쿼리를 실행합니다. 회귀 모델을 사용하면 주택 가격이나 도시의 자전거 대여 서비스를 이용할 사람 수와 같은 수치 결과를 예측할 수 있습니다. Amazon Redshift에서 훈련 데이터와 함께 CREATE MODEL 명령을 사용합니다. 그런 다음 Amazon Redshift ML이 모델을 컴파일하고 훈련된 모델을

Redshift로 가져온 다음 SQL 예측 함수를 준비합니다. 예측 함수는 Amazon Redshift의 SQL 쿼리에 사용할 수 있습니다.

이 튜토리얼에서는 Amazon Redshift ML을 사용하여 하루 중 특정 시간에 토론토 시의 자전거 공유 서비스를 사용하는 사람의 수를 예측하는 회귀 모델을 구축합니다. 모델의 입력에는 휴일 및 기상 조건이 포함됩니다. 이 문제에 대한 수치적 결과를 원하기 때문에 회귀 모델을 사용하게 됩니다.

CREATE MODEL 명령을 사용하여 훈련 데이터를 내보내고, 모델을 훈련하고, Amazon Redshift에서 SQL 함수로 모델을 사용 가능하게 할 수 있습니다. CREATE MODEL 스테이트먼트를 사용하여 테이블 또는 SELECT 작업으로 훈련 데이터를 지정합니다.

사용 사례

Amazon Redshift ML을 사용하여 고객의 평생 가치 예측과 같은 기타 회귀 문제를 해결할 수 있습니다. 또한 Redshift ML을 사용하여 제품의 가장 수익성이 높은 가격과 그에 따른 제품 수익을 예측할 수 있습니다.

업무

- 사전 조건
- 1단계: Amazon S3에서 Amazon Redshift로 데이터 로드
- 2단계: 기계 학습 모델 생성
- 3단계: 모델 검증

사전 조건

이 튜토리얼을 완료하려면 Amazon Redshift ML의 관리 설정을 완료해야 합니다.

1단계: Amazon S3에서 Amazon Redshift로 데이터 로드

Amazon Redshift 쿼리 편집기 v2를 사용하여 다음 쿼리를 실행합니다.

1. 세 개의 퍼블릭 데이터 세트를 Amazon Redshift로 로드하려면 테이블 세 개를 생성해야 합니다. 데이터 세트는 Toronto Bike Ridership Data, historical weather data, historical holidays data입니다. Amazon Redshift 쿼리 편집기에서 다음 쿼리를 실행하여 `ridership`, `weather`, `holiday`라는 테이블을 생성합니다.

```
CREATE TABLE IF NOT EXISTS ridership (
    trip_id INT,
    trip_duration_seconds INT,
```

```
    trip_start_time timestamp,
    trip_stop_time timestamp,
    from_station_name VARCHAR(50),
    to_station_name VARCHAR(50),
    from_station_id SMALLINT,
    to_station_id SMALLINT,
    user_type VARCHAR(20)
);

CREATE TABLE IF NOT EXISTS weather (
    longitude_x DECIMAL(5, 2),
    latitude_y DECIMAL(5, 2),
    station_name VARCHAR(20),
    climate_id BIGINT,
    datetime_utc TIMESTAMP,
    weather_year SMALLINT,
    weather_month SMALLINT,
    weather_day SMALLINT,
    time_utc VARCHAR(5),
    temp_c DECIMAL(5, 2),
    temp_flag VARCHAR(1),
    dew_point_temp_c DECIMAL(5, 2),
    dew_point_temp_flag VARCHAR(1),
    rel_hum SMALLINT,
    rel_hum_flag VARCHAR(1),
    precip_amount_mm DECIMAL(5, 2),
    precip_amount_flag VARCHAR(1),
    wind_dir_10s_deg VARCHAR(10),
    wind_dir_flag VARCHAR(1),
    wind_spd_kmh VARCHAR(10),
    wind_spd_flag VARCHAR(1),
    visibility_km VARCHAR(10),
    visibility_flag VARCHAR(1),
    stn_press_kpa DECIMAL(5, 2),
    stn_press_flag VARCHAR(1),
    hmdx SMALLINT,
    hmdx_flag VARCHAR(1),
    wind_chill VARCHAR(10),
    wind_chill_flag VARCHAR(1),
    weather VARCHAR(10)
);

CREATE TABLE IF NOT EXISTS holiday (holiday_date DATE, description VARCHAR(100));
```

2. 다음 쿼리는 샘플 데이터를 이전 단계에서 생성한 테이블로 로드합니다.

```
COPY ridership
FROM
    's3://redshift-ml-bikesharing-data/bike-sharing-data/ridership/'
    IAM_ROLE default
    FORMAT CSV
    IGNOREHEADER 1
    DATEFORMAT 'auto'
    TIMEFORMAT 'auto'
    REGION 'us-west-2'
    gzip;

COPY weather
FROM
    's3://redshift-ml-bikesharing-data/bike-sharing-data/weather/'
    IAM_ROLE default
    FORMAT csv
    IGNOREHEADER 1
    DATEFORMAT 'auto'
    TIMEFORMAT 'auto'
    REGION 'us-west-2'
    gzip;

COPY holiday
FROM
    's3://redshift-ml-bikesharing-data/bike-sharing-data/holiday/'
    IAM_ROLE default
    FORMAT csv
    IGNOREHEADER 1
    DATEFORMAT 'auto'
    TIMEFORMAT 'auto'
    REGION 'us-west-2'
    gzip;
```

3. 다음 쿼리는 `ridership` 및 `weather` 데이터 세트에 변환을 수행하여 편향이나 이상을 제거합니다. 편향과 이상을 제거하면 모델 정확도가 향상됩니다. 이 쿼리는 `ridership_view`와 `weather_view`라는 두 개의 새로운 보기를 생성하여 테이블을 단순화합니다.

```
CREATE
OR REPLACE VIEW ridership_view AS
SELECT
    trip_time,
```

```
    trip_count,
    TO_CHAR(trip_time, 'hh24') :: INT trip_hour,
    TO_CHAR(trip_time, 'dd') :: INT trip_day,
    TO_CHAR(trip_time, 'mm') :: INT trip_month,
    TO_CHAR(trip_time, 'yy') :: INT trip_year,
    TO_CHAR(trip_time, 'q') :: INT trip_quarter,
    TO_CHAR(trip_time, 'w') :: INT trip_month_week,
    TO_CHAR(trip_time, 'd') :: INT trip_week_day
FROM
    (
        SELECT
            CASE
                WHEN TRUNC(r.trip_start_time) < '2017-07-01' :: DATE THEN
 CONVERT_TIMEZONE(
                    'US/Eastern',
                    DATE_TRUNC('hour', r.trip_start_time)
                )
                ELSE DATE_TRUNC('hour', r.trip_start_time)
            END trip_time,
            COUNT(1) trip_count
        FROM
            ridership r
        WHERE
            r.trip_duration_seconds BETWEEN 60
            AND 60 * 60 * 24
        GROUP BY
            1
    );

CREATE
OR REPLACE VIEW weather_view AS
SELECT
    CONVERT_TIMEZONE(
        'US/Eastern',
        DATE_TRUNC('hour', datetime_utc)
    ) daytime,
    ROUND(AVG(temp_c)) temp_c,
    ROUND(AVG(precip_amount_mm)) precip_amount_mm
FROM
    weather
GROUP BY
    1;
```

4. 다음 쿼리는 ridership_view, weather_view에서 모든 관련 입력 속성을 trip_data 테이블에 결합하는 테이블을 생성합니다.

```
CREATE TABLE trip_data AS
SELECT
    r.trip_time,
    r.trip_count,
    r.trip_hour,
    r.trip_day,
    r.trip_month,
    r.trip_year,
    r.trip_quarter,
    r.trip_month_week,
    r.trip_week_day,
    w.temp_c,
    w.precip_amount_mm,CASE
        WHEN h.holiday_date IS NOT NULL THEN 1
        WHEN TO_CHAR(r.trip_time, 'D') :: INT IN (1, 7) THEN 1
        ELSE 0
    END is_holiday,
    ROW_NUMBER() OVER (
        ORDER BY
            RANDOM()
    ) serial_number
FROM
    ridership_view r
    JOIN weather_view w ON (r.trip_time = w.daytime)
    LEFT OUTER JOIN holiday h ON (TRUNC(r.trip_time) = h.holiday_date);
```

샘플 데이터 보기(선택 사항)

다음 쿼리는 테이블의 항목을 보여줍니다. 이 작업을 실행하여 테이블이 올바르게 생성되었는지 확인할 수 있습니다.

```
SELECT *
FROM trip_data
LIMIT 5;
```

다음은 이전 작업 출력의 예시입니다.

```
+---------------------+-----------+----------+----------+-----------+----------
+-------------+----------------+--------------+--------+-----------------
+------------+--------------+
|      trip_time      | trip_count | trip_hour | trip_day | trip_month | trip_year
 | trip_quarter | trip_month_week | trip_week_day | temp_c | precip_amount_mm |
 is_holiday | serial_number |
+---------------------+-----------+----------+----------+-----------+----------
+-------------+----------------+--------------+--------+-----------------
+------------+--------------+
| 2017-03-21 22:00:00 |         47 |        22 |       21 |          3 |        17 |
         1 |               3 |            3 |      1 |               0 |          0 |
             1 |
| 2018-05-04 01:00:00 |         19 |         1 |        4 |          5 |        18 |
         2 |               1 |            6 |     12 |               0 |          0 |
             3 |
| 2018-01-11 10:00:00 |         93 |        10 |       11 |          1 |        18 |
         1 |               2 |            5 |      9 |               0 |          0 |
             5 |
| 2017-10-28 04:00:00 |         20 |         4 |       28 |         10 |        17 |
         4 |               4 |            7 |     11 |               0 |          1 |
             7 |
| 2017-12-31 21:00:00 |         11 |        21 |       31 |         12 |        17 |
         4 |               5 |            1 |    -15 |               0 |          1 |
             9 |
+---------------------+-----------+----------+----------+-----------+----------
+-------------+----------------+--------------+--------+-----------------
+------------+--------------+
```

속성 간의 상관 관계 표시(선택 사항)

상관 관계를 결정하면 속성 간의 연관 강도를 측정하는 데 도움이 됩니다. 연관 수준은 목표 출력에 영향을 미치는 요소를 결정하는 데 도움이 될 수 있습니다. 이 튜토리얼의 목표 출력은 `trip_count`입니다.

다음 쿼리는 `sp_correlation` 절차를 생성하거나 대체합니다. `sp_correlation`라는 저장된 절차를 사용하여 Amazon Redshift의 테이블에 있는 속성과 다른 속성 간의 상관 관계를 표시합니다.

```
CREATE OR REPLACE PROCEDURE sp_correlation(source_schema_name in varchar(255),
 source_table_name in varchar(255), target_column_name in varchar(255),
 output_temp_table_name inout varchar(255)) AS $$
DECLARE
  v_sql varchar(max);
  v_generated_sql varchar(max);
```

```
  v_source_schema_name varchar(255)=lower(source_schema_name);
  v_source_table_name varchar(255)=lower(source_table_name);
  v_target_column_name varchar(255)=lower(target_column_name);
BEGIN
  EXECUTE 'DROP TABLE IF EXISTS ' || output_temp_table_name;
  v_sql = '
SELECT
  ''CREATE temp table '|| output_temp_table_name||' AS SELECT ''|| outer_calculation||
  '' FROM (SELECT COUNT(1) number_of_items, SUM('||v_target_column_name||')
 sum_target, SUM(POW('||v_target_column_name||',2)) sum_square_target, POW(SUM('||
v_target_column_name||'),2) square_sum_target,''||
  inner_calculation||
  '' FROM (SELECT ''||
  column_name||
  '' FROM '||v_source_table_name||''))''
FROM
  (
  SELECT
    DISTINCT
    LISTAGG(outer_calculation,'','') OVER () outer_calculation
    ,LISTAGG(inner_calculation,'','') OVER () inner_calculation
    ,LISTAGG(column_name,'','') OVER () column_name
  FROM
    (
    SELECT
      CASE WHEN atttypid=16 THEN ''DECODE(''||column_name||'',true,1,0)'' ELSE
 column_name END column_name
      ,atttypid
      ,''CAST(DECODE(number_of_items * sum_square_''||rn||'' - square_sum_''||
rn||'',0,null,(number_of_items*sum_target_''||rn||'' - sum_target * sum_''||rn||
        '')/SQRT((number_of_items * sum_square_target - square_sum_target) *
 (number_of_items * sum_square_''||rn||
        '' - square_sum_''||rn||'')) AS numeric(5,2)) ''||column_name
 outer_calculation
      ,''sum(''||column_name||'') sum_''||rn||'',''||
            ''SUM(trip_count*''||column_name||'') sum_target_''||rn||'',''||
            ''SUM(POW(''||column_name||'',2)) sum_square_''||rn||'',''||
            ''POW(SUM(''||column_name||''),2) square_sum_''||rn inner_calculation
    FROM
      (
      SELECT
        row_number() OVER (order by a.attnum) rn
        ,a.attname::VARCHAR column_name
        ,a.atttypid
```

```
      FROM pg_namespace AS n
        INNER JOIN pg_class AS c ON n.oid = c.relnamespace
        INNER JOIN pg_attribute AS a ON c.oid = a.attrelid
      WHERE a.attnum > 0
        AND n.nspname = '''||v_source_schema_name||'''
        AND c.relname = '''||v_source_table_name||'''
        AND a.atttypid IN (16,20,21,23,700,701,1700)
      )
    )
)';
  EXECUTE v_sql INTO v_generated_sql;
  EXECUTE  v_generated_sql;
END;
$$ LANGUAGE plpgsql;
```

다음 쿼리는 대상 열, `trip_count`, 데이터 세트 내의 기타 숫자 속성 간의 상관 관계를 표시합니다.

```
call sp_correlation(
    'public',
    'trip_data',
    'trip_count',
    'tmp_corr_table'
);

SELECT
    *
FROM
    tmp_corr_table;
```

다음은 이전 `sp_correlation` 작업 출력의 예시입니다.

```
+------------+-----------+----------+------------+-----------+--------------
+-----------------+---------------+--------+------------------+------------
+---------------+
| trip_count | trip_hour | trip_day | trip_month | trip_year | trip_quarter
 | trip_month_week | trip_week_day | temp_c | precip_amount_mm | is_holiday |
 serial_number |
+------------+-----------+----------+------------+-----------+--------------
+-----------------+---------------+--------+------------------+------------
+---------------+
|          1 |      0.32 |     0.01 |       0.18 |      0.12 |         0.18 |
     0 |          0.02 |   0.53 |            -0.07 |      -0.13 |             0 |
```

```
+------------+-----------+----------+------------+-----------+--------------
+-----------------+---------------+--------+------------------+------------
+---------------+
```

2단계: 기계 학습 모델 생성

1. 다음 쿼리는 데이터 세트의 80%를 훈련으로 지정하고 20%를 검증으로 지정하여 데이터를 훈련 세트와 검증 세트로 분할합니다. 훈련 세트는 ML 모델에서 가능한 최상의 알고리즘을 식별하기 위한 입력입니다. 모델을 생성한 후 검증 세트를 사용하여 모델 정확도를 검증합니다.

```
CREATE TABLE training_data AS
SELECT
    trip_count,
    trip_hour,
    trip_day,
    trip_month,
    trip_year,
    trip_quarter,
    trip_month_week,
    trip_week_day,
    temp_c,
    precip_amount_mm,
    is_holiday
FROM
    trip_data
WHERE
    serial_number > (
        SELECT
            COUNT(1) * 0.2
        FROM
            trip_data
    );

CREATE TABLE validation_data AS
SELECT
    trip_count,
    trip_hour,
    trip_day,
    trip_month,
    trip_year,
    trip_quarter,
    trip_month_week,
    trip_week_day,
```

```
    temp_c,
    precip_amount_mm,
    is_holiday,
    trip_time
FROM
    trip_data
WHERE
    serial_number <= (
        SELECT
            COUNT(1) * 0.2
        FROM
            trip_data
    );
```

2. 다음 쿼리는 임의의 입력 날짜 및 시간에 대해 trip_count 값을 예측하는 회귀 모델을 생성합니다. 다음 예제에서는 amzn-s3-demo-bucket을 자체 S3 버킷으로 바꿉니다.

```
CREATE MODEL predict_rental_count
FROM
    training_data TARGET trip_count FUNCTION predict_rental_count
    IAM_ROLE default
    PROBLEM_TYPE regression
    OBJECTIVE 'mse'
    SETTINGS (
        s3_bucket 'amzn-s3-demo-bucket',
        s3_garbage_collect off,
        max_runtime 5000
    );
```

3단계: 모델 검증

1. 다음 쿼리를 사용하여 모델의 측면을 출력하고 출력에서 평균 제곱 오차 메트릭을 찾습니다. 평균 제곱 오차는 회귀 문제에서 흔히 사용하는 정확도 메트릭입니다.

```
show model predict_rental_count;
```

2. 검증 데이터에 대해 다음 예측 쿼리를 실행하여 예측된 이동 횟수를 실제 이동 횟수와 비교합니다.

```
SELECT
    trip_time,
    actual_count,
```

```
    predicted_count,
    (actual_count - predicted_count) difference
FROM
    (
        SELECT
            trip_time,
            trip_count AS actual_count,
            PREDICT_RENTAL_COUNT (
                trip_hour,
                trip_day,
                trip_month,
                trip_year,
                trip_quarter,
                trip_month_week,
                trip_week_day,
                temp_c,
                precip_amount_mm,
                is_holiday
            ) predicted_count
        FROM
            validation_data
    )
LIMIT
    5;
```

3. 다음 쿼리는 검증 데이터를 기반으로 평균 제곱 오차 및 평균 제곱근 오차를 계산합니다. 평균 제곱 오차와 평균 제곱근 평균 제곱 오차를 사용하여 예상 수치 목표 값과 실제 수치 답변 간의 거리를 측정합니다. 좋은 모델은 두 메트릭 모두에서 점수가 낮습니다. 다음 쿼리는 두 메트릭의 값을 반환합니다.

```
SELECT
    ROUND(
        AVG(POWER((actual_count - predicted_count), 2)),
        2
    ) mse,
    ROUND(
        SQRT(AVG(POWER((actual_count - predicted_count), 2))),
        2
    ) rmse
FROM
    (
        SELECT
            trip_time,
```

```
            trip_count AS actual_count,
            PREDICT_RENTAL_COUNT (
                trip_hour,
                trip_day,
                trip_month,
                trip_year,
                trip_quarter,
                trip_month_week,
                trip_week_day,
                temp_c,
                precip_amount_mm,
                is_holiday
            ) predicted_count
        FROM
            validation_data
    );
```

4. 다음 쿼리는 2017-01-01에 일어난 각 이동 시간에 대한 이동 횟수의 백분율 오차를 계산합니다. 쿼리는 오류 비율이 가장 낮은 시간부터 오류 비율이 가장 높은 시간 순서로 이동 시간을 정렬합니다.

```
SELECT
    trip_time,
    CAST(ABS(((actual_count - predicted_count) / actual_count)) * 100 AS DECIMAL
 (7,2)) AS pct_error
FROM
    (
        SELECT
            trip_time,
            trip_count AS actual_count,
            PREDICT_RENTAL_COUNT (
                trip_hour,
                trip_day,
                trip_month,
                trip_year,
                trip_quarter,
                trip_month_week,
                trip_week_day,
                temp_c,
                precip_amount_mm,
                is_holiday
            ) predicted_count
        FROM
            validation_data
```

```
    )
WHERE
   trip_time LIKE '2017-01-01 %%:%%:%%'
ORDER BY
   2 ASC;
```

관련 주제

Amazon Redshift ML에 대한 자세한 내용은 다음 설명서를 참조하세요.

- Amazon Redshift 기계 학습 사용 비용
- CREATE MODEL 작업
- EXPLAIN_MODEL 함수

기계 학습에 대한 자세한 내용은 다음 설명서를 참조하세요.

- 기계 학습 개요
- 초보자 및 전문가를 위한 기계 학습
- What Is Fairness and Model Explainability for Machine Learning Predictions?

튜토리얼: 선형 학습기를 사용하여 회귀 모델 구축

이 튜토리얼에서는 Amazon S3의 데이터로 선형 학습기 모델을 생성하고 Amazon Redshift ML을 사용하여 이 모델로 예측 쿼리를 실행합니다. SageMaker AI 선형 학습기 알고리즘은 회귀 또는 멀티클래스 분류 문제를 해결합니다. 회귀 및 멀티클래스 분류 문제에 대한 자세한 내용은 Amazon SageMaker AI Developer Guide의 기본 기계 학습 패러다임의 문제 유형을 참조하세요. 이 튜토리얼에서는 회귀 문제를 해결합니다. 선형 학습기 알고리즘은 여러 모델을 병렬로 훈련시키고 가장 최적화된 모델을 자동으로 결정합니다. Amazon Redshift에서 CREATE MODEL 작업을 사용하면 SageMaker AI를 사용하여 선형 학습기 모델을 생성하고 예측 함수를 Amazon Redshift로 보냅니다. 선형 학습기 알고리즘에 대한 자세한 내용은 Amazon SageMaker AI Developer Guide의 Linear Learner Algorithm 섹션을 참조하세요.

CREATE MODEL 명령을 사용하여 훈련 데이터를 내보내고, 모델을 훈련하고, 모델을 가져오고, Amazon Redshift 예측 함수를 준비할 수 있습니다. CREATE MODEL 스테이트먼트를 사용하여 테이블 또는 SELECT 작업으로 훈련 데이터를 지정합니다.

선형 학습기 모델은 연속형 목표 또는 이산 목표를 최적화합니다. 연속형 목표는 회귀에 사용되고 이산형 변수는 분류에 사용됩니다. 회귀 방법과 같은 일부 방법은 연속형 목표에 대해서만 솔루션을 제공합니다. 선형 학습기 알고리즘은 Naive Bayes 기법과 같은 단순한 하이퍼파라미터 최적화 기법에 비해 속도를 높입니다. 나이브 최적화 기법에서는 각 입력 변수가 독립적이라고 가정합니다. 선형 학습기 알고리즘을 사용하려면 입력의 차원을 나타내는 열과 관측치를 나타내는 행을 제공해야 합니다. 선형 학습기 알고리즘에 대한 자세한 내용은 Amazon SageMaker AI Developer Guide의 Linear Learner Algorithm 섹션을 참조하세요.

이 튜토리얼에서는 전복의 나이를 예측하는 선형 학습기 모델을 구축합니다. Abalone dataset에서 CREATE MODEL 명령을 사용하여 전복의 물리적 측정값 간의 관계를 결정합니다. 그런 다음 모델을 사용하여 전복의 나이를 결정합니다.

사용 사례

선형 학습기 및 Amazon Redshift ML을 사용하여 주택 가격 예측과 같은 다른 회귀 문제를 해결할 수 있습니다. Redshift ML을 사용하여 도시의 자전거 대여 서비스를 이용할 사람의 수도 예측할 수 있습니다.

업무

- 사전 조건
- 1단계: Amazon S3에서 Amazon Redshift로 데이터 로드
- 2단계: 기계 학습 모델 생성
- 3단계: 모델 검증

사전 조건

이 튜토리얼을 완료하려면 Amazon Redshift ML의 관리 설정을 완료해야 합니다.

1단계: Amazon S3에서 Amazon Redshift로 데이터 로드

Amazon Redshift 쿼리 편집기 v2를 사용하여 다음 쿼리를 실행합니다. 이 쿼리는 샘플 데이터를 Redshift로 로드하고 데이터를 훈련 세트와 검증 세트로 나눕니다.

1. 다음 예에서는 `abalone_dataset` 테이블을 생성합니다.

```
CREATE TABLE abalone_dataset (
    id INT IDENTITY(1, 1),
    Sex CHAR(1),
    Length float,
```

```
    Diameter float,
    Height float,
    Whole float,
    Shucked float,
    Viscera float,
    Shell float,
    Rings integer
);
```

2. 다음 쿼리는 이전에 Amazon Redshift에서 생성한 abalone_dataset 테이블에 Amazon S3의 Abalone dataset에 있는 샘플 데이터를 복사합니다.

```
COPY abalone_dataset
FROM
    's3://redshift-ml-multiclass/abalone.csv' REGION 'us-east-1' IAM_ROLE default CSV
 IGNOREHEADER 1 NULL AS 'NULL';
```

3. 데이터를 수동으로 분할하면 추가 예측 세트를 할당하여 모델의 정확도를 확인할 수 있습니다. 다음 쿼리는 데이터를 두 세트로 분할합니다. abalone_training 테이블은 훈련용이고 abalone_validation 테이블은 검증용입니다.

```
CREATE TABLE abalone_training as
SELECT
    *
FROM
    abalone_dataset
WHERE
    mod(id, 10) < 8;

CREATE TABLE abalone_validation as
SELECT
    *
FROM
    abalone_dataset
WHERE
    mod(id, 10) >= 8;
```

2단계: 기계 학습 모델 생성

이 단계에서는 CREATE MODEL 스테이트먼트를 사용하여 선형 학습기 알고리즘으로 머신 러닝 모델을 생성합니다.

다음 쿼리는 S3 버킷을 사용하여 CREATE MODEL 작업으로 선형 학습기 모델을 생성합니다. amzn-s3-demo-bucket을 자체 S3 버킷으로 교체합니다.

```
CREATE MODEL model_abalone_ring_prediction
FROM
    (
        SELECT
            Sex,
            Length,
            Diameter,
            Height,
            Whole,
            Shucked,
            Viscera,
            Shell,
            Rings AS target_label
        FROM
            abalone_training
    ) TARGET target_label FUNCTION f_abalone_ring_prediction IAM_ROLE default
 MODEL_TYPE LINEAR_LEARNER PROBLEM_TYPE REGRESSION OBJECTIVE 'MSE' SETTINGS (
        S3_BUCKET 'amzn-s3-demo-bucket',
        MAX_RUNTIME 15000
    );
```

모델 훈련 상태 표시(선택 사항)

SHOW MODEL 명령을 사용하여 모델이 준비되었는지 알 수 있습니다.

다음 쿼리를 사용하여 모델 훈련의 진행 상황을 모니터링합니다.

```
SHOW MODEL model_abalone_ring_prediction;
```

모델이 준비되면 이전 작업의 출력은 다음 예시와 비슷할 것입니다. 출력은 평균 제곱 오차를 나타내는 `validation:mse` 메트릭을 제공합니다. 다음 단계에서 평균 제곱 오차를 사용하여 모델의 정확도를 확인합니다.

```
+--------------------------
+-----------------------------------------------------------------------------------------------
+
|        Model Name        |
 model_abalone_ring_prediction                                    |
```

```
+--------------------------
+-----------------------------------------------------------------------------------------------------------
+
| Schema Name               | public
                                          |
| Owner                     | awsuser
                                          |
| Creation Time             | Thu, 30.06.2022 18:00:10
                                          |
| Model State               | READY
                                          |
| validation:mse            |
                                  4.168633 |
| Estimated Cost            |
                                  4.291608 |
|                           |
                                          |
| TRAINING DATA:            |
                                          |
| Query                     | SELECT SEX , LENGTH , DIAMETER , HEIGHT , WHOLE ,
 SHUCKED , VISCERA , SHELL, RINGS AS TARGET_LABEL |
|                           | FROM ABALONE_TRAINING
                                          |
| Target Column             | TARGET_LABEL
                                          |
|                           |
                                          |
| PARAMETERS:               |
                                          |
| Model Type                | linear_learner
                                          |
| Problem Type              | Regression
                                          |
| Objective                 | MSE
                                          |
| AutoML Job Name           | redshiftml-20220630180010947843
                                          |
| Function Name             | f_abalone_ring_prediction
                                          |
| Function Parameters       | sex length diameter height whole shucked viscera shell
                                          |
| Function Parameter Types | bpchar float8 float8 float8 float8 float8 float8 float8
                                          |
```

```
| IAM Role                  | default-aws-iam-role
                                        |
| S3 Bucket                 | amzn-s3-demo-bucket
                                        |
| Max Runtime               |
                                 15000 |
+--------------------------
+-----------------------------------------------------------------------------------------------------------
+
```

3단계: 모델 검증

1. 다음 예측 쿼리는 평균 제곱 오차 및 제곱 평균 제곱근 오차를 계산하여 abalone_validation 데이터 세트에서 모델의 정확도를 검증합니다.

```
SELECT
    ROUND(AVG(POWER((tgt_label - predicted), 2)), 2) mse,
    ROUND(SQRT(AVG(POWER((tgt_label - predicted), 2))), 2) rmse
FROM
    (
        SELECT
            Sex,
            Length,
            Diameter,
            Height,
            Whole,
            Shucked,
            Viscera,
            Shell,
            Rings AS tgt_label,
            f_abalone_ring_prediction(
                Sex,
                Length,
                Diameter,
                Height,
                Whole,
                Shucked,
                Viscera,
                Shell
            ) AS predicted,
            CASE
                WHEN tgt_label = predicted then 1
                ELSE 0
```

```
            END AS match,
            CASE
                WHEN tgt_label <> predicted then 1
                ELSE 0
            END AS nonmatch
        FROM
            abalone_validation
    ) t1;
```

이전 쿼리의 출력은 다음 예시와 비슷합니다. 평균 제곱 오차 메트릭의 값은 SHOW MODEL 작업의 출력에 표시되는 `validation:mse` 메트릭과 유사할 것입니다.

```
+-----+--------------------+
| mse |        rmse        |
+-----+--------------------+
| 5.1 | 2.2600000000000002 |
+-----+--------------------+
```

2. 다음 쿼리를 사용하여 예측 함수에 EXPLAIN_MODEL 작업을 실행합니다. 이 작업은 모델 설명 가능성 보고서를 반환합니다. EXPLAIN_MODEL 작업에 대한 자세한 내용은 Amazon Redshift 데이터베이스 개발자 안내서의 EXPLAIN_MODEL 함수를 참조하세요.

```
SELECT
    EXPLAIN_MODEL ('model_abalone_ring_prediction');
```

다음 정보는 이전 EXPLAIN_MODEL 작업에서 생성된 모델 설명 가능성 보고서의 예입니다. 각 입력의 값은 Shapley 값입니다. Shapley 값은 각 입력이 모델 예측에 미치는 영향을 나타내며 값이 높을수록 예측에 더 많은 영향을 미칩니다. 이 예에서 입력 값이 높을수록 전복의 나이를 예측하는 데 더 많은 영향을 미칩니다.

```
{
    "explanations": {
        "kernel_shap": {
            "label0": {
                "expected_value" :10.290688514709473,
                "global_shap_values": {
                    "diameter" :0.6856910187882492,
                    "height" :0.4415323937124035,
                    "length" :0.21507476107609084,
                    "sex" :0.448611774505744,
                    "shell" :1.70426496893776,
```

```
                    "shucked" :2.1181392924386994,
                    "viscera" :0.342220754059912,
                    "whole" :0.6711906974084011
                }
            }
        }
    },
    "version" :"1.0"
};
```

3. 다음 쿼리를 사용하여 모델이 아직 성숙하지 않은 전복에 대해 수행하는 올바른 예측의 백분율을 계산합니다. 미성숙한 전복은 고리가 10개 이하이며 실제 고리 수의 한 고리 내에서까지 정확한 예측이 가능합니다.

```
SELECT
    TRUNC(
        SUM(
            CASE
                WHEN ROUND(
                    f_abalone_ring_prediction(
                        Sex,
                        Length,
                        Diameter,
                        Height,
                        Whole,
                        Shucked,
                        Viscera,
                        Shell
                    ),
                    0
                ) BETWEEN Rings - 1
                AND Rings + 1 THEN 1
                ELSE 0
            END
        ) / CAST(COUNT(SHELL) AS FLOAT),
        4
    ) AS prediction_pct
FROM
    abalone_validation
WHERE
    Rings <= 10;
```

관련 주제

Amazon Redshift ML에 대한 자세한 내용은 다음 설명서를 참조하세요.

- Amazon Redshift 기계 학습 사용 비용
- CREATE MODEL 작업
- EXPLAIN_MODEL 함수

기계 학습에 대한 자세한 내용은 다음 설명서를 참조하세요.

- 기계 학습 개요
- 초보자 및 전문가를 위한 기계 학습
- What Is Fairness and Model Explainability for Machine Learning Predictions?

튜토리얼: 선형 학습기를 사용하여 다중 클래스 분류 모델 구축

이 튜토리얼에서는 Amazon S3의 데이터로 선형 학습기 모델을 생성하고 Amazon Redshift ML을 사용하여 이 모델로 예측 쿼리를 실행합니다. SageMaker AI 선형 학습기 알고리즘은 회귀 또는 분류 문제를 해결합니다. 회귀 및 멀티클래스 분류 문제에 대한 자세한 내용은 Amazon SageMaker AI Developer Guide의 기본 기계 학습 패러다임의 문제 유형을 참조하세요. 이 튜토리얼에서는 다중 클래스 분류 문제를 해결합니다. 선형 학습기 알고리즘은 여러 모델을 병렬로 훈련시키고 가장 최적화된 모델을 자동으로 결정합니다. Amazon Redshift에서 CREATE MODEL 작업을 사용하면 SageMaker AI를 사용하여 선형 학습기 모델을 생성하고 예측 함수를 Amazon Redshift로 보냅니다. 선형 학습기 알고리즘에 대한 자세한 내용은 Amazon SageMaker AI Developer Guide의 Linear Learner Algorithm 섹션을 참조하세요.

CREATE MODEL 명령을 사용하여 훈련 데이터를 내보내고, 모델을 훈련하고, 모델을 가져오고, Amazon Redshift 예측 함수를 준비할 수 있습니다. CREATE MODEL 스테이트먼트를 사용하여 테이블 또는 SELECT 작업으로 훈련 데이터를 지정합니다.

선형 학습기 모델은 연속형 목표 또는 이산 목표를 최적화합니다. 연속형 목표는 회귀에 사용되고 이산형 변수는 분류에 사용됩니다. 회귀 방법과 같은 일부 방법은 연속형 목표에 대해서만 솔루션을 제공합니다. 선형 학습기 알고리즘은 Naive Bayes 기법과 같은 단순한 하이퍼파라미터 최적화 기법에 비해 속도를 높입니다. 나이브 최적화 기법에서는 각 입력 변수가 독립적이라고 가정합니다. 선형 학습기 알고리즘은 여러 모델을 병렬로 훈련시키고 가장 최적화된 모델을 선택합니다. 비슷한 알고리즘은 XGBoost로, 더 간단하고 약한 모델 세트의 추정치를 결합하여 예측을 수행합니다. XGBoost에 대한 자세한 내용은 Amazon SageMaker AI Developer Guide의 XGBoost algorithm 섹션을 참조하세요.

선형 학습기 알고리즘을 사용하려면 입력의 차원을 나타내는 열과 관측치를 나타내는 행을 제공해야 합니다. 선형 학습기 알고리즘에 대한 자세한 내용은 Amazon SageMaker AI Developer Guide의 Linear Learner Algorithm 섹션을 참조하세요.

이 튜토리얼에서는 특정 영역의 식물 유형을 예측하는 선형 학습기 모델을 구축합니다. UCI Machine Learning Repository에 있는 Covertype dataset에서 CREATE MODEL 명령을 사용합니다. 그런 다음 명령으로 생성한 예측 함수를 사용하여 황야 지역의 식물 유형을 결정합니다. 임상 식물 유형은 일반적으로 나무 중 하나입니다. Redshift ML에서 모델을 생성하는 데 사용할 입력에는 토양 유형, 도로까지의 거리 및 황야 지역 지정이 포함됩니다. 이 데이터 세트에 대한 자세한 내용은 UCI Machine Learning Repository에서 Covertype Dataset를 참조하세요.

사용 사례

Amazon Redshift ML을 사용하는 선형 학습기로 이미지에서 식물의 종을 예측하는 등 다른 다중 클래스 분류 문제를 해결할 수 있습니다. 고객이 구매할 제품의 수량도 예측할 수 있습니다.

업무

- 사전 조건
- 1단계: Amazon S3에서 Amazon Redshift로 데이터 로드
- 2단계: 기계 학습 모델 생성
- 3단계: 모델 검증

사전 조건

이 튜토리얼을 완료하려면 Amazon Redshift ML의 관리 설정을 완료해야 합니다.

1단계: Amazon S3에서 Amazon Redshift로 데이터 로드

Amazon Redshift 쿼리 편집기 v2를 사용하여 다음 쿼리를 실행합니다. 이 쿼리는 샘플 데이터를 Redshift로 로드하고 데이터를 훈련 세트와 검증 세트로 나눕니다.

1. 다음 예에서는 `covertype_data` 테이블을 생성합니다.

```
CREATE TABLE public.covertype_data (
    elevation bigint ENCODE az64,
    aspect bigint ENCODE az64,
    slope bigint ENCODE az64,
    horizontal_distance_to_hydrology bigint ENCODE az64,
    vertical_distance_to_hydrology bigint ENCODE az64,
```

```
    horizontal_distance_to_roadways bigint ENCODE az64,
    hillshade_9am bigint ENCODE az64,
    hillshade_noon bigint ENCODE az64,
    hillshade_3pm bigint ENCODE az64,
    horizontal_distance_to_fire_points bigint ENCODE az64,
    wilderness_area1 bigint ENCODE az64,
    wilderness_area2 bigint ENCODE az64,
    wilderness_area3 bigint ENCODE az64,
    wilderness_area4 bigint ENCODE az64,
    soil_type1 bigint ENCODE az64,
    soil_type2 bigint ENCODE az64,
    soil_type3 bigint ENCODE az64,
    soil_type4 bigint ENCODE az64,
    soil_type5 bigint ENCODE az64,
    soil_type6 bigint ENCODE az64,
    soil_type7 bigint ENCODE az64,
    soil_type8 bigint ENCODE az64,
    soil_type9 bigint ENCODE az64,
    soil_type10 bigint ENCODE az64,
    soil_type11 bigint ENCODE az64,
    soil_type12 bigint ENCODE az64,
    soil_type13 bigint ENCODE az64,
    soil_type14 bigint ENCODE az64,
    soil_type15 bigint ENCODE az64,
    soil_type16 bigint ENCODE az64,
    soil_type17 bigint ENCODE az64,
    soil_type18 bigint ENCODE az64,
    soil_type19 bigint ENCODE az64,
    soil_type20 bigint ENCODE az64,
    soil_type21 bigint ENCODE az64,
    soil_type22 bigint ENCODE az64,
    soil_type23 bigint ENCODE az64,
    soil_type24 bigint ENCODE az64,
    soil_type25 bigint ENCODE az64,
    soil_type26 bigint ENCODE az64,
    soil_type27 bigint ENCODE az64,
    soil_type28 bigint ENCODE az64,
    soil_type29 bigint ENCODE az64,
    soil_type30 bigint ENCODE az64,
    soil_type31 bigint ENCODE az64,
    soil_type32 bigint ENCODE az64,
    soil_type33 bigint ENCODE az64,
    soil_type34 bigint ENCODE az64,
    soil_type35 bigint ENCODE az64,
```

```
    soil_type36 bigint ENCODE az64,
    soil_type37 bigint ENCODE az64,
    soil_type38 bigint ENCODE az64,
    soil_type39 bigint ENCODE az64,
    soil_type40 bigint ENCODE az64,
    cover_type bigint ENCODE az64
) DISTSTYLE AUTO;
```

2. 다음 쿼리는 이전에 Amazon Redshift에서 생성한 `covertype_data` 테이블에 Amazon S3의 Covertype dataset에 있는 샘플 데이터를 복사합니다.

```
COPY public.covertype_data
FROM
    's3://redshift-ml-multiclass/covtype.data.gz' IAM_ROLE DEFAULT gzip DELIMITER ','
 REGION 'us-east-1';
```

3. 데이터를 수동으로 분할하면 추가 테스트 세트를 할당하여 모델의 정확도를 확인할 수 있습니다. 다음 쿼리는 데이터를 세 세트로 분할합니다. `covertype_training` 테이블은 훈련용이고 `covertype_validation` 테이블은 검증용, `covertype_test` 테이블은 모델 테스트용입니다. 훈련 세트를 사용하여 모델을 학습시키고 검증 세트를 사용하여 모델 개발을 검증합니다. 그런 다음 테스트 세트를 사용하여 모델의 성능을 테스트하고 모델이 데이터 세트를 과적합으로 처리하는지 또는 과소적합으로 처리하는지 확인합니다.

```
CREATE TABLE public.covertype_data_prep AS
SELECT
    a.*,
    CAST (random() * 100 AS int) AS data_group_id
FROM
    public.covertype_data a;

--training dataset
 CREATE TABLE public.covertype_training as
SELECT
    *
FROM
    public.covertype_data_prep
WHERE
    data_group_id < 80;

--validation dataset
 CREATE TABLE public.covertype_validation AS
SELECT
```

```
    *
FROM
    public.covertype_data_prep
WHERE
    data_group_id BETWEEN 80
    AND 89;

--test dataset
 CREATE TABLE public.covertype_test AS
SELECT
    *
FROM
    public.covertype_data_prep
WHERE
    data_group_id > 89;
```

2단계: 기계 학습 모델 생성

이 단계에서는 CREATE MODEL 스테이트먼트를 사용하여 선형 학습기 알고리즘으로 머신 러닝 모델을 생성합니다.

다음 쿼리는 S3 버킷을 사용하여 CREATE MODEL 작업으로 선형 학습기 모델을 생성합니다. amzn-s3-demo-bucket을 자체 S3 버킷으로 교체합니다.

```
CREATE MODEL forest_cover_type_model
FROM
    (
        SELECT
            Elevation,
            Aspect,
            Slope,
            Horizontal_distance_to_hydrology,
            Vertical_distance_to_hydrology,
            Horizontal_distance_to_roadways,
            HIllshade_9am,
            Hillshade_noon,
            Hillshade_3pm,
            Horizontal_Distance_To_Fire_Points,
            Wilderness_Area1,
            Wilderness_Area2,
            Wilderness_Area3,
            Wilderness_Area4,
```

```
            soil_type1,
            Soil_Type2,
            Soil_Type3,
            Soil_Type4,
            Soil_Type5,
            Soil_Type6,
            Soil_Type7,
            Soil_Type8,
            Soil_Type9,
            Soil_Type10,
            Soil_Type11,
            Soil_Type12,
            Soil_Type13,
            Soil_Type14,
            Soil_Type15,
            Soil_Type16,
            Soil_Type17,
            Soil_Type18,
            Soil_Type19,
            Soil_Type20,
            Soil_Type21,
            Soil_Type22,
            Soil_Type23,
            Soil_Type24,
            Soil_Type25,
            Soil_Type26,
            Soil_Type27,
            Soil_Type28,
            Soil_Type29,
            Soil_Type30,
            Soil_Type31,
            Soil_Type32,
            Soil_Type33,
            Soil_Type34,
            Soil_Type36,
            Soil_Type37,
            Soil_Type38,
            Soil_Type39,
            Soil_Type40,
            Cover_type
        from
            public.covertype_training
    ) TARGET cover_type FUNCTION predict_cover_type IAM_ROLE default MODEL_TYPE
LINEAR_LEARNER PROBLEM_TYPE MULTICLASS_CLASSIFICATION OBJECTIVE 'Accuracy' SETTINGS (
```

```
        S3_BUCKET 'amzn-s3-demo-bucket',
        S3_GARBAGE_COLLECT OFF,
        MAX_RUNTIME 15000
    );
```

모델 훈련 상태 표시(선택 사항)

SHOW MODEL 명령을 사용하여 모델이 준비되었는지 알 수 있습니다.

다음 쿼리를 사용하여 모델 훈련의 진행 상황을 모니터링합니다.

```
SHOW MODEL forest_cover_type_model;
```

모델이 준비되면 이전 작업의 출력은 다음 예시와 비슷할 것입니다. 출력은 다음 예의 오른쪽에서 볼 수 있는 validation:multiclass_accuracy 메트릭을 제공합니다. 다중 클래스 정확도는 모델에 따라 올바르게 분류된 데이터 포인트의 백분율을 측정합니다. 다음 단계에서 다중 클래스 정확도를 사용하여 모델의 정확도를 확인합니다.

```
+---------------------------------
+-------------------------------------------------------------------------------------------------
+
|              Key               |




 Value




                                                          |
+---------------------------------
+-------------------------------------------------------------------------------------------------
+
| Model Name                     | forest_cover_type_model
```

```
                                                    |
| Schema Name                    | public
                                                    |
| Owner                          | awsuser
                                                    |
| Creation Time                  | Tue, 12.07.2022 20:24:32
                                                    |
| Model State                    | READY
                                                    |
| validation:multiclass_accuracy |
```

```
                                        0.724952 |
| Estimated Cost                  |

                                        5.341750 |
|                                 |

                                                 |
| TRAINING DATA:                  |

                                                 |
| Query                           | SELECT ELEVATION, ASPECT, SLOPE,
 HORIZONTAL_DISTANCE_TO_HYDROLOGY, VERTICAL_DISTANCE_TO_HYDROLOGY,
 HORIZONTAL_DISTANCE_TO_ROADWAYS, HILLSHADE_9AM, HILLSHADE_NOON, HILLSHADE_3PM ,
 HORIZONTAL_DISTANCE_TO_FIRE_POINTS, WILDERNESS_AREA1, WILDERNESS_AREA2,
 WILDERNESS_AREA3, WILDERNESS_AREA4, SOIL_TYPE1, SOIL_TYPE2, SOIL_TYPE3, SOIL_TYPE4,
 SOIL_TYPE5, SOIL_TYPE6, SOIL_TYPE7, SOIL_TYPE8, SOIL_TYPE9, SOIL_TYPE10 , SOIL_TYPE11,
```

```
 SOIL_TYPE12 , SOIL_TYPE13 , SOIL_TYPE14, SOIL_TYPE15, SOIL_TYPE16, SOIL_TYPE17,
 SOIL_TYPE18, SOIL_TYPE19, SOIL_TYPE20, SOIL_TYPE21, SOIL_TYPE22, SOIL_TYPE23,
 SOIL_TYPE24, SOIL_TYPE25, SOIL_TYPE26, SOIL_TYPE27, SOIL_TYPE28, SOIL_TYPE29,
 SOIL_TYPE30, SOIL_TYPE31, SOIL_TYPE32, SOIL_TYPE33, SOIL_TYPE34, SOIL_TYPE36,
 SOIL_TYPE37, SOIL_TYPE38, SOIL_TYPE39, SOIL_TYPE40, COVER_TYPE |
|                               | FROM PUBLIC.COVERTYPE_TRAINING

                                                                |
| Target Column                 | COVER_TYPE

                                                                |
|                               |

                                                                |
| PARAMETERS:                   |
```

```
                                                    |
| Model Type                    | linear_learner

                                                    |
| Problem Type                  | MulticlassClassification

                                                    |
| Objective                     | Accuracy

                                                    |
| AutoML Job Name               | redshiftml-20220712202432187659

                                                    |
| Function Name                 | predict_cover_type
```

```
                                                    |
| Function Parameters            | elevation aspect slope
 horizontal_distance_to_hydrology vertical_distance_to_hydrology
 horizontal_distance_to_roadways hillshade_9am hillshade_noon hillshade_3pm
 horizontal_distance_to_fire_points wilderness_area1 wilderness_area2 wilderness_area3
 wilderness_area4 soil_type1 soil_type2 soil_type3 soil_type4 soil_type5 soil_type6
 soil_type7 soil_type8 soil_type9 soil_type10 soil_type11 soil_type12 soil_type13
 soil_type14 soil_type15 soil_type16 soil_type17 soil_type18 soil_type19 soil_type20
 soil_type21 soil_type22 soil_type23 soil_type24 soil_type25 soil_type26 soil_type27
 soil_type28 soil_type29 soil_type30 soil_type31 soil_type32 soil_type33 soil_type34
 soil_type36 soil_type37 soil_type38 soil_type39 soil_type40
                                                   |
| Function Parameter Types       | int8 int8 int8 int8 int8 int8 int8 int8 int8 int8
 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8
 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8
 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8
 int8 int8 int8 int8 int8 int8 int8 int8 int8
```

```
                                                          |
| IAM Role                       | default-aws-iam-role
```

```
                                                    |
| S3 Bucket                      | amzn-s3-demo-bucket
```

```
                                                    |
| Max Runtime                    |

                                                15000 |
+--------------------------------
+-----------------------------------------------------------------------------------------------
+
```

3단계: 모델 검증

1. 다음 예측 쿼리는 다중 클래스 정확도를 계산하여 covertype_validation 데이터 세트에서 모델의 정확도를 검증합니다. 다중 클래스 정확도는 올바른 모델 예측의 백분율입니다.

```
SELECT
    CAST(sum(t1.match) AS decimal(7, 2)) AS predicted_matches,
    CAST(sum(t1.nonmatch) AS decimal(7, 2)) AS predicted_non_matches,
    CAST(sum(t1.match + t1.nonmatch) AS decimal(7, 2)) AS total_predictions,
    predicted_matches / total_predictions AS pct_accuracy
FROM
    (
        SELECT
            Elevation,
            Aspect,
            Slope,
            Horizontal_distance_to_hydrology,
            Vertical_distance_to_hydrology,
            Horizontal_distance_to_roadways,
            HIllshade_9am,
            Hillshade_noon,
            Hillshade_3pm,
            Horizontal_Distance_To_Fire_Points,
            Wilderness_Area1,
```

```
            Wilderness_Area2,
            Wilderness_Area3,
            Wilderness_Area4,
            soil_type1,
            Soil_Type2,
            Soil_Type3,
            Soil_Type4,
            Soil_Type5,
            Soil_Type6,
            Soil_Type7,
            Soil_Type8,
            Soil_Type9,
            Soil_Type10,
            Soil_Type11,
            Soil_Type12,
            Soil_Type13,
            Soil_Type14,
            Soil_Type15,
            Soil_Type16,
            Soil_Type17,
            Soil_Type18,
            Soil_Type19,
            Soil_Type20,
            Soil_Type21,
            Soil_Type22,
            Soil_Type23,
            Soil_Type24,
            Soil_Type25,
            Soil_Type26,
            Soil_Type27,
            Soil_Type28,
            Soil_Type29,
            Soil_Type30,
            Soil_Type31,
            Soil_Type32,
            Soil_Type33,
            Soil_Type34,
            Soil_Type36,
            Soil_Type37,
            Soil_Type38,
            Soil_Type39,
            Soil_Type40,
            Cover_type AS actual_cover_type,
            predict_cover_type(
```

```
                Elevation,
                Aspect,
                Slope,
                Horizontal_distance_to_hydrology,
                Vertical_distance_to_hydrology,
                Horizontal_distance_to_roadways,
                HIllshade_9am,
                Hillshade_noon,
                Hillshade_3pm,
                Horizontal_Distance_To_Fire_Points,
                Wilderness_Area1,
                Wilderness_Area2,
                Wilderness_Area3,
                Wilderness_Area4,
                soil_type1,
                Soil_Type2,
                Soil_Type3,
                Soil_Type4,
                Soil_Type5,
                Soil_Type6,
                Soil_Type7,
                Soil_Type8,
                Soil_Type9,
                Soil_Type10,
                Soil_Type11,
                Soil_Type12,
                Soil_Type13,
                Soil_Type14,
                Soil_Type15,
                Soil_Type16,
                Soil_Type17,
                Soil_Type18,
                Soil_Type19,
                Soil_Type20,
                Soil_Type21,
                Soil_Type22,
                Soil_Type23,
                Soil_Type24,
                Soil_Type25,
                Soil_Type26,
                Soil_Type27,
                Soil_Type28,
                Soil_Type29,
                Soil_Type30,
```

```
                Soil_Type31,
                Soil_Type32,
                Soil_Type33,
                Soil_Type34,
                Soil_Type36,
                Soil_Type37,
                Soil_Type38,
                Soil_Type39,
                Soil_Type40
            ) AS predicted_cover_type,
            CASE
                WHEN actual_cover_type = predicted_cover_type THEN 1
                ELSE 0
            END AS match,
            CASE
                WHEN actual_cover_type <> predicted_cover_type THEN 1
                ELSE 0
            END AS nonmatch
        FROM
            public.covertype_validation
    ) t1;
```

이전 쿼리의 출력은 다음 예시와 비슷합니다. 다중 클래스 정확도 메트릭의 값은 SHOW MODEL 작업의 출력에 표시되는 validation:multiclass_accuracy 메트릭과 유사할 것입니다.

```
+-------------------+-----------------------+-------------------+--------------+
| predicted_matches | predicted_non_matches | total_predictions | pct_accuracy |
+-------------------+-----------------------+-------------------+--------------+
|             41211 |                 16324 |             57535 |   0.71627704 |
+-------------------+-----------------------+-------------------+--------------+
```

2. 다음 쿼리는 wilderness_area2에서 가장 일반적인 식물 유형을 예측합니다. 이 데이터 세트에는 4개의 황야 지역과 7개의 식물 유형이 포함되어 있습니다. 황야 지역에는 여러 식물 유형이 있을 수 있습니다.

```
SELECT t1. predicted_cover_type, COUNT(*)
FROM
(
SELECT
   Elevation,
   Aspect,
   Slope,
```

```
    Horizontal_distance_to_hydrology,
    Vertical_distance_to_hydrology,
    Horizontal_distance_to_roadways,
    HIllshade_9am,
    Hillshade_noon,
    Hillshade_3pm ,
    Horizontal_Distance_To_Fire_Points,
    Wilderness_Area1,
    Wilderness_Area2,
    Wilderness_Area3,
    Wilderness_Area4,
    soil_type1,
    Soil_Type2,
    Soil_Type3,
    Soil_Type4,
    Soil_Type5,
    Soil_Type6,
    Soil_Type7,
    Soil_Type8,
    Soil_Type9,
    Soil_Type10 ,
    Soil_Type11,
    Soil_Type12 ,
    Soil_Type13 ,
    Soil_Type14,
    Soil_Type15,
    Soil_Type16,
    Soil_Type17,
    Soil_Type18,
    Soil_Type19,
    Soil_Type20,
    Soil_Type21,
    Soil_Type22,
    Soil_Type23,
    Soil_Type24,
    Soil_Type25,
    Soil_Type26,
    Soil_Type27,
    Soil_Type28,
    Soil_Type29,
    Soil_Type30,
    Soil_Type31,
    Soil_Type32,
    Soil_Type33,
```

```
    Soil_Type34,
    Soil_Type36,
    Soil_Type37,
    Soil_Type38,
    Soil_Type39,
    Soil_Type40,
    predict_cover_type( Elevation,
    Aspect,
    Slope,
    Horizontal_distance_to_hydrology,
    Vertical_distance_to_hydrology,
    Horizontal_distance_to_roadways,
    HIllshade_9am,
    Hillshade_noon,
    Hillshade_3pm ,
    Horizontal_Distance_To_Fire_Points,
    Wilderness_Area1,
    Wilderness_Area2,
    Wilderness_Area3,
    Wilderness_Area4,
    soil_type1,
    Soil_Type2,
    Soil_Type3,
    Soil_Type4,
    Soil_Type5,
    Soil_Type6,
    Soil_Type7,
    Soil_Type8,
    Soil_Type9,
    Soil_Type10,
    Soil_Type11,
    Soil_Type12,
    Soil_Type13,
    Soil_Type14,
    Soil_Type15,
    Soil_Type16,
    Soil_Type17,
    Soil_Type18,
    Soil_Type19,
    Soil_Type20,
    Soil_Type21,
    Soil_Type22,
    Soil_Type23,
    Soil_Type24,
```

```
   Soil_Type25,
   Soil_Type26,
   Soil_Type27,
   Soil_Type28,
   Soil_Type29,
   Soil_Type30,
   Soil_Type31,
   Soil_Type32,
   Soil_Type33,
   Soil_Type34,
   Soil_Type36,
   Soil_Type37,
   Soil_Type38,
   Soil_Type39,
   Soil_Type40) AS predicted_cover_type

FROM public.covertype_test
WHERE wilderness_area2 = 1)
t1
GROUP BY 1;
```

이전 작업의 출력은 다음 예시와 비슷할 것입니다. 이 출력은 모델이 식물의 대부분이 식물 유형 1이고 식물 유형 2, 7이 일부 있다고 예측했음을 의미합니다.

```
+----------------------+-------+
| predicted_cover_type | count |
+----------------------+-------+
|                    2 |   564 |
|                    7 |    97 |
|                    1 |  2309 |
+----------------------+-------+
```

3. 다음 쿼리는 하나의 황야 지역에서 가장 일반적인 식물 유형을 보여줍니다. 쿼리에는 해당 식물 유형의 양과 해당 식물 유형의 황야 지역이 표시됩니다.

```
SELECT t1. predicted_cover_type, COUNT(*), wilderness_area
FROM
(
SELECT
   Elevation,
   Aspect,
   Slope,
```

```
    Horizontal_distance_to_hydrology,
    Vertical_distance_to_hydrology,
    Horizontal_distance_to_roadways,
    HIllshade_9am,
    Hillshade_noon,
    Hillshade_3pm ,
    Horizontal_Distance_To_Fire_Points,
    Wilderness_Area1,
    Wilderness_Area2,
    Wilderness_Area3,
    Wilderness_Area4,
    soil_type1,
    Soil_Type2,
    Soil_Type3,
    Soil_Type4,
    Soil_Type5,
    Soil_Type6,
    Soil_Type7,
    Soil_Type8,
    Soil_Type9,
    Soil_Type10 ,
    Soil_Type11,
    Soil_Type12 ,
    Soil_Type13 ,
    Soil_Type14,
    Soil_Type15,
    Soil_Type16,
    Soil_Type17,
    Soil_Type18,
    Soil_Type19,
    Soil_Type20,
    Soil_Type21,
    Soil_Type22,
    Soil_Type23,
    Soil_Type24,
    Soil_Type25,
    Soil_Type26,
    Soil_Type27,
    Soil_Type28,
    Soil_Type29,
    Soil_Type30,
    Soil_Type31,
    Soil_Type32,
    Soil_Type33,
```

```
    Soil_Type34,
    Soil_Type36,
    Soil_Type37,
    Soil_Type38,
    Soil_Type39,
    Soil_Type40,
    predict_cover_type( Elevation,
    Aspect,
    Slope,
    Horizontal_distance_to_hydrology,
    Vertical_distance_to_hydrology,
    Horizontal_distance_to_roadways,
    HIllshade_9am,
    Hillshade_noon,
    Hillshade_3pm ,
    Horizontal_Distance_To_Fire_Points,
    Wilderness_Area1,
    Wilderness_Area2,
    Wilderness_Area3,
    Wilderness_Area4,
    soil_type1,
    Soil_Type2,
    Soil_Type3,
    Soil_Type4,
    Soil_Type5,
    Soil_Type6,
    Soil_Type7,
    Soil_Type8,
    Soil_Type9,
    Soil_Type10,
    Soil_Type11,
    Soil_Type12,
    Soil_Type13,
    Soil_Type14,
    Soil_Type15,
    Soil_Type16,
    Soil_Type17,
    Soil_Type18,
    Soil_Type19,
    Soil_Type20,
    Soil_Type21,
    Soil_Type22,
    Soil_Type23,
    Soil_Type24,
```

```
   Soil_Type25,
   Soil_Type26,
   Soil_Type27,
   Soil_Type28,
   Soil_Type29,
   Soil_Type30,
   Soil_Type31,
   Soil_Type32,
   Soil_Type33,
   Soil_Type34,
   Soil_Type36,
   Soil_Type37,
   Soil_Type38,
   Soil_Type39,
   Soil_Type40) AS predicted_cover_type,
   CASE WHEN Wilderness_Area1 = 1 THEN 1
        WHEN Wilderness_Area2 = 1 THEN 2
        WHEN Wilderness_Area3 = 1 THEN 3
        WHEN Wilderness_Area4 = 1 THEN 4
        ELSE 0
   END AS wilderness_area

FROM public.covertype_test)
t1
GROUP BY 1, 3
ORDER BY 2 DESC
LIMIT 1;
```

이전 작업의 출력은 다음 예시와 비슷할 것입니다.

```
+----------------------+-------+-----------------+
| predicted_cover_type | count | wilderness_area |
+----------------------+-------+-----------------+
|                    2 | 15738 |               1 |
+----------------------+-------+-----------------+
```

관련 주제

Amazon Redshift ML에 대한 자세한 내용은 다음 설명서를 참조하세요.

- Amazon Redshift 기계 학습 사용 비용

- CREATE MODEL 작업
- EXPLAIN_MODEL 함수

기계 학습에 대한 자세한 내용은 다음 설명서를 참조하세요.

- 기계 학습 개요
- 초보자 및 전문가를 위한 기계 학습
- What Is Fairness and Model Explainability for Machine Learning Predictions?(기계 학습 예측을 위한 공정성과 모델 설명 가능성이란 무엇입니까?)

Amazon Redshift ML과 Amazon Bedrock의 통합

이 섹션에서는 Amazon Redshift ML과 Amazon Bedrock의 통합을 사용하는 방법에 대해 설명합니다. 이 기능을 사용하면 SQL을 사용하여 Amazon Bedrock 모델을 간접적으로 호출할 수 있으며 Amazon Redshift 데이터 웨어하우스의 데이터를 사용하여 텍스트 작성, 감정 분석 또는 번역과 같은 생성형 AI 애플리케이션을 구축할 수 있습니다.

주제

- Amazon Redshift ML과 Amazon Bedrock의 통합을 위한 IAM 역할 생성 또는 업데이트
- Amazon Redshift ML과 Amazon Bedrock의 통합용 외부 모델 생성
- Amazon Redshift ML과 Amazon Bedrock의 통합용 외부 모델 사용
- Amazon Redshift ML과 Amazon Bedrock의 통합 시 프롬프트 엔지니어링

Amazon Redshift ML과 Amazon Bedrock의 통합을 위한 IAM 역할 생성 또는 업데이트

이 섹션에서는 Amazon Redshift ML과 Amazon Bedrock의 통합에 사용할 IAM 역할을 만드는 방법에 대해 보여 줍니다.

Amazon Redshift ML과 Amazon Bedrock의 통합에 사용하는 IAM 역할에 다음 정책을 추가합니다.

- `AmazonBedrockFullAccess`

Amazon Redshift가 다른 서비스와 상호 작용하는 역할을 맡도록 허용하려면 IAM 역할에 다음의 신뢰 정책을 추가합니다.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "Service": [
               "redshift.amazonaws.com"
            ]
         },
         "Action": "sts:AssumeRole"
      }
   ]
}
```

클러스터 또는 네임스페이스가 VPC에 있는 경우 Amazon Redshift 기계 학습 관리를 위한 클러스터 및 구성 설정에 나와 있는 설명을 따릅니다.

더 제한적인 정책이 필요한 경우 다음 페이지에 명시된 Amazon Bedrock 권한만 포함하는 정책을 생성할 수 있습니다.

- Amazon Redshift 기계 학습 관리를 위한 클러스터 및 구성 설정
- Amazon Redshift 기계 학습(ML)을 사용하는 데 필요한 권한

IAM 역할 생성에 관한 자세한 내용은 AWS Identity and Access Management 사용 설명서에 나와 있는 IAM 역할 생성 섹션을 참조하시기 바랍니다.

Amazon Redshift ML과 Amazon Bedrock의 통합용 외부 모델 생성

이 섹션에서는 Amazon Redshift 데이터 웨어하우스 내에서 Amazon Bedrock의 인터페이스로 사용할 외부 모델을 생성하는 방법을 보여 줍니다.

Amazon Redshift에서 Amazon Bedrock 모델을 간접적으로 호출하려면 먼저 `CREATE EXTERNAL MODEL` 명령을 실행해야 합니다. 이 명령을 사용하면 데이터베이스에 외부 모델 객체와 Amazon Bedrock을 사용하여 텍스트 콘텐츠를 작성하는 데 사용하는 관련 사용자 함수가 만들어집니다.

다음 코드 예제는 기본 `CREATE EXTERNAL MODEL` 명령을 보여 줍니다.

```
CREATE EXTERNAL MODEL llm_claude
FUNCTION llm_claude_func
IAM_ROLE '<IAM role arn>'
```

```
MODEL_TYPE BEDROCK
SETTINGS (
   MODEL_ID 'anthropic.claude-v2:1',
   PROMPT 'Summarize the following text:');
```

CREATE EXTERNAL MODEL 명령은 메시지를 지원하는 모든 파운데이션 모델(FM)에 대해 Amazon Bedrock과 일관된 통합 인터페이스를 제공합니다. CREATE EXTERNAL MODEL 명령을 사용하거나 요청 유형을 UNIFIED로 명시적으로 지정할 경우의 기본 옵션입니다. 자세한 내용은 Amazon Bedrock API 설명서에 나와 있는 Converse API 설명서를 참조하시기 바랍니다.

FM이 메시지를 지원하지 않는 경우 request_type 설정을 RAW로 설정해야 합니다. request_type을 RAW로 설정하면 선택한 FM을 기반으로 추론 함수를 사용할 경우 Amazon Bedrock에 전송된 요청을 구성해야 합니다.

CREATE EXTERNAL MODEL 명령의 PROMPT 파라미터는 정적 프롬프트입니다. 애플리케이션에 동적 프롬프트가 필요하면 추론 함수 사용 시 동적 프롬프트를 지정해야 합니다. 자세한 내용은 다음 Amazon Redshift ML과 Amazon Bedrock의 통합 시 프롬프트 엔지니어링 섹션을 참조하시기 바랍니다.

CREATE EXTERNAL MODEL 스테이트먼트와 해당 파라미터, 설정에 대한 자세한 내용은 CREATE EXTERNAL MODEL 섹션을 참조하시기 바랍니다.

Amazon Redshift ML과 Amazon Bedrock의 통합용 외부 모델 사용

이 섹션에서는 외부 모델을 간접적으로 호출하여 제공된 프롬프트에 대한 응답으로 텍스트를 작성하는 방법을 보여 줍니다. 외부 모델을 간접적으로 호출하려면 CREATE EXTERNAL MODEL로 생성한 추론 함수를 사용합니다.

주제

- UNIFIED 요청 유형 모델로 추론
- RAW 요청 유형 모델로 추론
- 리더 전용 함수로서의 추론 함수
- 추론 함수 사용 정보

UNIFIED 요청 유형 모델로 추론

UNIFIED 요청 유형이 있는 모델의 추론 함수에는 함수에 전달되는 다음 3가지 파라미터가 순서대로 포함됩니다.

- 입력 텍스트(필수): 이 파라미터는 Amazon Redshift가 Amazon Bedrock에 전달하는 입력 텍스트를 지정합니다.
- 추론 구성 및 추가 모델 요청 필드(선택 사항): Amazon Redshift는 이러한 파라미터를 Converse 모델 API의 해당 파라미터에 전달합니다.

다음 코드 예제는 UNIFIED 유형 추론 함수를 사용하는 방법을 보여 줍니다.

```
SELECT llm_claude_func(input_text, object('temperature', 0.7, 'maxtokens', 500))
   FROM some_data;
```

RAW 요청 유형 모델로 추론

RAW 요청 유형이 있는 모델의 추론 함수에는 SUPER 데이터 유형의 파라미터가 하나만 포함됩니다. 이 파라미터의 구문은 사용된 Amazon Bedrock 모델에 따라 달라집니다.

다음 코드 예제는 RAW 유형 추론 함수를 사용하는 방법을 보여 줍니다.

```
SELECT llm_titan_func(
    object(
        "inputText", "Summarize the following text: " | input_text,
        "textGenerationConfig", object("temperature", 0.5, "maxTokenCount", 500)
    )
)
FROM some_data;
```

리더 전용 함수로서의 추론 함수

Amazon Bedrock 모델의 추론 함수는 이를 사용하는 쿼리가 테이블을 참조하지 않을 때 리더 노드 전용 함수로 실행될 수 있습니다. 이는 LLM에 빠르게 질문하려는 경우에 유용할 수 있습니다.

다음 코드 예제는 리더 전용 추론 함수를 사용하는 방법을 보여 줍니다.

```
SELECT general_titan_llm_func('Summarize the benefits of LLM on data analytics in 100
 words');
```

추론 함수 사용 정보

Amazon Redshift ML과 Amazon Bedrock의 통합으로 추론 함수를 사용할 때 다음 사항을 참고하시기 바랍니다.

- 모든 Amazon Bedrock 모델의 파라미터 이름은 대문자와 소문자를 구분합니다. 파라미터가 모델에 필요한 파라미터와 일치하지 않으면 Amazon Bedrock에서는 조용히 파라미터를 무시할 수 있습니다.
- 추론 쿼리의 처리량은 Amazon Bedrock이 제공하는 다양한 모델의 런타임 할당량에 따라 다른 리전에서 제한됩니다. 자세한 내용은 Amazon Bedrock 사용자 가이드에 나와 있는 Amazon Bedrock 할당량을 참조하시기 바랍니다.
- 안정적이고 일관된 처리량이 필요한 경우 Amazon Bedrock에서 필요한 모델의 프로비저닝 처리량을 가져오는 것이 좋습니다. 자세한 내용은 Amazon Bedrock 사용자 가이드에 나와 있는 Increase model invocation capacity with Provisioned Throughput in Amazon Bedrock을 참조하시기 바랍니다.
- 대량 데이터가 포함된 추론 쿼리에서는 스로틀링 예외가 발생할 수도 있습니다. 이는 Amazon Bedrock의 런타임 할당량이 제한되어 있기 때문입니다. Amazon Redshift는 요청을 여러 번 재시도하지만, 프로비저닝되지 않은 모델의 처리량이 가변적일 수 있으므로 쿼리는 여전히 스로틀링될 수 있습니다.
- 소량의 데이터라도 `Too many requests, please wait before trying again.`과 같이 Amazon Bedrock에서 발생하는 스로틀링 예외가 있는 경우 Amazon Bedrock 계정의 Service Quotas에서 할당량을 확인합니다. 적용된 계정 수준 할당량이 사용 중인 모델의 InvokeModel 요청에 대한 AWS 기본 할당량 값과 적어도 동일한지 확인합니다.

Amazon Redshift ML과 Amazon Bedrock의 통합 시 프롬프트 엔지니어링

이 섹션에서는 외부 모델에서 정적 프롬프트를 사용하는 방법을 보여 줍니다.

외부 모델에 정적 접두사 및 접미사 프롬프트를 사용하려면 `CREATE EXTERNAL MODEL` 스테이트먼트에서 `PROMPT` 및 `SUFFIX` 파라미터를 사용하여 제공합니다. 이러한 프롬프트는 외부 모델을 사용하여 모든 쿼리에 추가됩니다.

다음 예제에서는 외부 모델에 접두사 및 접미사 프롬프트를 추가하는 방법을 보여 줍니다.

```
CREATE EXTERNAL MODEL llm_claude
FUNCTION llm_claude_func
IAM_ROLE '<IAM role arn>'
MODEL_TYPE BEDROCK
SETTINGS (
   MODEL_ID 'anthropic.claude-v2:1',
   PROMPT 'Summarize the following text:',
   SUFFIX 'Respond in an analytic tone');
```

동적 프롬프트를 사용하려면 추론 함수 사용 시 함수 입력에 이를 연결하여 제공할 수 있습니다. 다음 예제에서는 추론 함수와 함께 동적 프롬프트를 사용하는 방법을 보여 줍니다.

```
SELECT llm_claude_func('Summarize the following review:' | input_text | 'The review
 should have formal tone.')
FROM some_data
```

쿼리 성능 튜닝

Amazon Redshift는 SQL 기반 쿼리를 사용하여 시스템 데이터 및 객체와 상호작용합니다. 데이터 조작 언어(DML)는 데이터를 보거나, 추가하거나, 변경하거나, 삭제하는 데 사용되는 SQL의 하위 집합입니다. 데이터 정의 언어(DDL)는 테이블 및 뷰 같은 데이터베이스 객체를 추가하거나, 변경하거나, 삭제하는 데 사용되는 SQL의 하위 집합입니다.

일단 시스템 설정을 마치면 특히 데이터를 가져오거나 확인하는 SELECT 명령에서 DML을 가장 많이 사용하게 됩니다. Amazon Redshift에서 데이터 가져오기 쿼리를 효과적으로 작성하려면 먼저 SELECT 문에 대해 이해한 후 Amazon Redshift 테이블 설계 모범 사례에서 간략하게 소개한 팁을 적용하여 쿼리 효율성을 극대화할 수 있습니다.

Amazon Redshift의 쿼리 처리 방식에 대해 이해하려면 쿼리 처리 및 쿼리 분석 및 개선 사항 섹션을 참조하세요. 그런 다음 진단 도구와 함께 이 정보를 적용하면 쿼리 성능 문제를 찾아내 제거할 수 있습니다.

Amazon Redshift 쿼리에서 발생하는 가장 공통적인 문제와 가장 심각한 문제를 찾아내 해결하는 방법은 쿼리 문제 해결 섹션에서 확인할 수 있습니다.

주제

- 쿼리 처리
- 쿼리 분석 및 개선 사항
- 쿼리 문제 해결

쿼리 처리

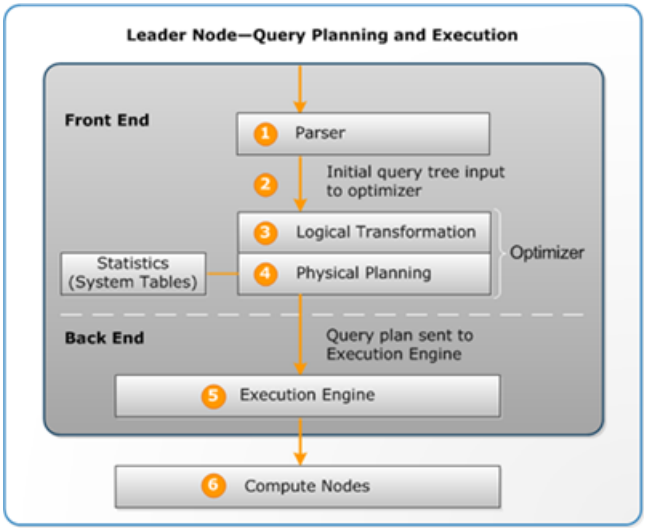
Amazon Redshift는 제출된 SQL 쿼리를 구문 분석기와 옵티마이저를 통해 라우팅하여 쿼리 계획을 작성합니다. 그러면 실행 엔진이 쿼리 계획을 코드로 변환한 후 해당 코드를 실행할 수 있도록 컴퓨팅 노드로 전송합니다.

주제

- 쿼리 계획 및 실행 워크플로우
- 쿼리 계획 생성 및 해석
- 쿼리 계획 단계 검토
- 쿼리 성능에 영향을 미치는 요인

쿼리 계획 및 실행 워크플로우

다음 그림은 쿼리 계획 및 실행 워크플로우를 종합적으로 도식한 것입니다.



쿼리 계획 및 실행 워크플로우는 다음과 같은 단계로 구성됩니다.

1. 리더 노드가 쿼리를 수신하고 SQL 구문을 분석합니다.
2. 구문 분석기가 원래 쿼리에 대한 논리적 표현인 초기 쿼리 트리를 생성합니다. 그런 다음 Amazon Redshift는 이 쿼리 트리를 쿼리 옵티마이저에 입력합니다.
3. 옵티마이저가 평가를 거쳐 필요하다고 판단되면 쿼리를 재작성하여 효율성을 극대화합니다. 이 프로세스는 간혹 관련 쿼리를 다수 생성하여 단일 쿼리를 대체하기도 합니다.
4. 옵티마이저가 성능을 극대화하는 쿼리 실행 계획(이전 단계에서 다수의 쿼리를 생성한 경우에는 쿼리 실행 계획들)을 작성합니다. 쿼리 계획이 조인 유형, 조인 순서, 집계 옵션, 데이터 분산 요건 등 실행 옵션을 지정합니다.

   EXPLAIN 명령을 사용하면 쿼리 계획을 확인할 수 있습니다. 쿼리 계획은 복합 쿼리를 분석하여 튜닝할 수 있는 기본 도구입니다. 자세한 내용은 쿼리 계획 생성 및 해석 섹션을 참조하세요.
5. 실행 엔진이 쿼리 계획을 단계, 세그먼트 및 스트림으로 변환합니다.

   단계

   각 단계는 쿼리 실행 시 필요한 개별 작업을 의미합니다. 컴퓨팅 노드는 이러한 단계들을 조합하여 쿼리, 조인 또는 기타 데이터베이스 작업 등을 실행할 수 있습니다.

세그먼트

단일 프로세스에서 실행할 수 있는 몇몇 단계들의 조합인 동시에 컴퓨팅 노드 조각에서 실행할 수 있는 가장 작은 컴파일 유닛입니다. 여기서 조각이란 Amazon Redshift의 병렬 처리 유닛을 말합니다. 스트림을 구성하는 세그먼트들은 병렬로 실행됩니다.

스트림

사용 가능한 컴퓨팅 노드 조각 위로 분할되는 세그먼트 집합입니다.

실행 엔진은 단계, 세그먼트 및 스트림을 기준으로 컴파일 코드를 작성합니다. 컴파일 코드는 인터프리터 코드보다 실행 속도가 빠르고 사용하는 컴퓨팅 용량도 적습니다. 이렇게 작성된 컴파일 코드는 컴퓨팅 노드로 브로드캐스팅됩니다.

> **Note**
>
> 쿼리를 벤치마킹할 때는 항상 두 번째 쿼리의 실행 시간을 비교해야 합니다. 첫 번째 쿼리의 실행 시간에는 코드를 컴파일하는 오버헤드가 포함되기 때문입니다. 자세한 내용은 쿼리 성능에 영향을 미치는 요인 섹션을 참조하세요.

6. 컴퓨팅 노드 조각이 쿼리 세그먼트를 병렬 방식으로 실행합니다. 이때 Amazon Redshift는 최적화된 네트워크 통신, 메모리 및 디스크 관리를 활용하여 한 쿼리 계획 단계의 중간 결과를 다음 쿼리 계획 단계로 전달합니다. 이는 쿼리 실행 속도를 높이는 데도 유용합니다.

5단계와 6단계는 각 스트림마다 한 번씩 일어납니다. 엔진은 스트림마다 실행 가능한 세그먼트를 생성하여 컴퓨팅 노드로 전송합니다. 이렇게 스트림 하나의 세그먼트가 완료되면 다음 스트림을 위한 세그먼트가 엔진에서 생성됩니다. 이러한 방식으로 엔진은 이전 스트림의 작업 내용(작업의 디스크 기반 여부 등)을 분석하여 다음 스트림의 세그먼트 생성에도 영향을 미칩니다.

컴퓨팅 노드에서 작업을 마치면 최종 처리를 위해 쿼리 결과를 다시 리더 노드로 보냅니다. 리더 노드는 수신되는 데이터를 단일 결과 집합으로 병합하여 필요에 따라 정렬 또는 집계를 실행합니다. 그런 다음 결과를 클라이언트에게 돌려보냅니다.

> **Note**
>
> 컴퓨팅 노드는 쿼리 실행 도중 필요에 따라 일부 데이터를 리더 노드로 다시 보낼 수도 있습니다. 예를 들어 LIMIT 절에 하위 쿼리가 있는 경우에는 리더 노드에 먼저 제한이 적용된 후에 데이터가 추가 처리를 위해 클러스터로 재분산됩니다.

쿼리 계획 생성 및 해석

쿼리 계획은 쿼리를 실행하는 데 필요한 개별 작업에 대한 정보를 확인하는 용도로 사용할 수 있습니다. 하지만 쿼리 계획을 사용하기 앞서서 먼저 Amazon Redshift가 쿼리를 처리하거나 쿼리 계획을 작성하는 방식부터 이해하는 것이 좋습니다. 자세한 내용은 쿼리 계획 및 실행 워크플로우 섹션을 참조하세요.

쿼리 계획을 작성하려면 실제 쿼리 텍스트와 함께 EXPLAIN 명령을 실행하세요. 쿼리 계획은 다음과 같은 정보를 제공합니다.

- 실행 엔진이 수행한 작업(상향식 결과 확인)
- 각 작업이 수행한 단계 유형
- 각 작업에서 사용되는 테이블과 열
- 행의 수와 데이터 폭(바이트)을 기준으로 한 각 작업의 데이터 처리량
- 상대적 작업 비용. 여기에서 비용이란 계획 내 각 단계의 상대적 실행 시간을 비교하는 척도를 말합니다. 그렇다고 비용이 실제 실행 시간이나 메모리 사용량에 대해 정확한 정보를 제공하는 것은 아니며, 마찬가지로 실행 계획 간 비교가 유의미하지도 않습니다. 단지 쿼리에서 가장 많은 리소스를 사용하는 작업이 무엇인지 알려주는 역할을 합니다.

EXPLAIN 명령은 실제로 쿼리를 실행하지는 않습니다. 쿼리가 현재 작업 조건에서 실행되는 경우 Amazon Redshift의 실행 계획을 보여줄 뿐입니다. 테이블 스키마 또는 데이터를 변경한 후 ANALYZE를 다시 실행하여 통계 메타데이터를 업데이트하면 쿼리 계획도 달라질 수 있습니다.

EXPLAIN에서 출력되는 쿼리 계획은 쿼리 실행을 단순하지만 종합적으로 나타냅니다. 병렬 쿼리 처리에 대한 세부 정보까지 표시하지는 않습니다. 세부 정보를 보려면 쿼리 자체를 실행한 후 SVL_QUERY_SUMMARY 또는 SVL_QUERY_REPORT 뷰에서 쿼리 요약 정보를 가져옵니다. 이러한 뷰를 사용하는 자세한 방법은 쿼리 요약 분석 섹션을 참조하세요.

다음은 EVENT 테이블에 대한 GROUP BY 쿼리를 실행할 경우를 나타낸 EXPLAIN 출력 예입니다.

```
explain select eventname, count(*) from event group by eventname;

                            QUERY PLAN
------------------------------------------------------------------
XN HashAggregate  (cost=131.97..133.41 rows=576 width=17)
  ->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798 width=17)
```

EXPLAIN은 다음과 같이 각 작업에 대한 지표를 반환합니다.

비용

계획 내 작업을 비교하는데 유용한 상대 값입니다. 비용은 2개의 마침표로 구분된 소수 값 2개로 구성됩니다. 예를 들면 `cost=131.97..133.41`과 같습니다. 여기에서 첫 번째 값인 131.97은 위 작업의 첫 번째 행을 반환하는 상대적 비용을 나타냅니다. 그리고, 두 번째 값인 133.41은 위 작업을 완료하는 상대적 비용을 나타냅니다. 쿼리 계획 비용은 아래에서 위 방향으로 계획을 읽어가면서 누적됩니다. 따라서 위 예에서 HashAggregate 비용(131.97..133.41)은 아래 있는 Seq Scan 비용(0.00..87.98)이 포함된 것입니다.

행

반환될 것으로 예상되는 행의 수입니다. 위 예에서 스캔을 통해 반환된 것으로 예상되는 행의 수는 8,798개입니다. HashAggregate 연산자 자체에서는 576개의 행이 반환될 것으로 예상됩니다(결과 집합에서 중복 이벤트 이름은 무시했을 때).

> **Note**
>
> 예상되는 행의 수는 ANALYZE 명령으로 생성된 유효 통계를 기준으로 합니다. 최근에 ANALYZE를 실행하지 않았다면 예상되는 행의 수는 신뢰성이 떨어집니다.

너비

평균 행의 예상 폭(바이트)입니다. 위 예에서 평균 행은 폭이 17바이트가 될 것으로 보입니다.

EXPLAIN 연산자

이번 섹션에서는 EXPLAIN 출력 시 가장 자주 표시되는 연산자에 대해서 간략히 설명합니다. 전체 연산자 목록은 SQL 명령 섹션에서 EXPLAIN 섹션을 참조하세요.

순차적 스캔 연산자

순차적 스캔 연산자(Seq Scan)는 테이블 스캔을 나타냅니다. Seq Scan이 처음부터 끝까지 테이블의 열을 각각 순차적으로 스캔하고 WHERE 절에서 각 행의 쿼리 제약 조건을 평가합니다.

조인 연산자

Amazon Redshift는 조인되는 테이블의 물리적 설계, 조인에 필요한 데이터의 위치, 쿼리 자체의 특정 요건을 기반으로 여러 가지 조인 연산자를 선택합니다.

- 중첩 루프

중첩 루프는 가장 덜 최적화된 조인으로서 주로 크로스 조인(데카르트 곱)과 일부 부등식 조인에 사용됩니다.

- Hash Join 및 Hash

  일반적으로 중첩 루프 조인보다 빠른 해시 조인 및 해시는 내부 조인과 왼쪽 및 오른쪽 외부 조인에 사용됩니다. 이 연산자들은 조인 열이 둘 다 분산 키 및 정렬 키가 아닌 테이블을 조인할 때 사용됩니다. 해시 연산자는 조인의 이너 테이블에 대한 해시 테이블을 생성합니다. 해시 조인 연산자는 아우터 테이블을 읽고, 조인 열을 해시하고, 이너 해시 테이블과 일치하는 항목을 검색합니다.

- 병합 조인

  일반적으로 가장 빠른 조인인 병합 조인은 내부 조인과 외부 조인에 사용되지만 전체 조인에는 사용되지 않습니다. 이 연산자는 조인 열이 둘 다 분산 키 및 정렬 키인 테이블을 조인할 때, 그리고 조인 테이블에서 정렬되지 않은 비율이 20% 미만일 때 사용됩니다. 정렬된 테이블 2개를 순서대로 읽고 나서 일치하는 행을 찾습니다. 정렬되지 않은 행의 비율을 보려면 SVV_TABLE_INFO 시스템 테이블에 대한 쿼리를 실행하세요.

- 공간 조인

  일반적으로 공간 데이터의 근접성을 기반으로 한 고속 조인으로, GEOMETRY 및 GEOGRAPHY 데이터 유형에 사용됩니다.

집계 연산자

쿼리 계획은 집계 함수 및 GROUP BY 작업과 관련된 쿼리에서 다음과 같은 연산자를 사용합니다.

- 집계

  AVG, SUM 같은 스칼라 집계 함수에 사용되는 연산자입니다.

- HashAggregate

  정렬 없이 분류된 집계 함수에 사용되는 연산자입니다.

- GroupAggregate

  정렬과 함께 분류된 집계 함수에 사용되는 연산자입니다.

정렬 연산자

쿼리 계획은 쿼리가 결과 집합을 정렬하거나 병합해야 할 때 다음과 같은 연산자를 사용합니다.

- 정렬

  ORDER BY 절을 비롯해 UNION 쿼리 및 조인, SELECT DISTINCT 쿼리, 창 함수에서 필요한 정렬 등 기타 정렬 작업을 평가합니다.

- Merge

  병렬 작업을 통해 얻은 중간 정렬 결과에 따라 최종 정렬 결과를 산출합니다.

UNION, INTERSECT 및 EXCEPT 연산자

쿼리 계획은 UNION, INTERSECT 및 EXCEPT를 사용하는 집합 작업과 관련된 쿼리에서 다음과 같은 연산자를 사용합니다.

- Subquery

  UNION 쿼리를 실행하는 데 사용됩니다.

- Hash Intersect Distinct

  쿼리를 실행하는 데 사용됩니다.

- SetOp Except

  EXCEPT(또는 MINUS) 쿼리를 실행하는 데 사용됩니다.

기타 연산자

그 밖에다 다음과 같은 연산자가 일반 쿼리의 EXPLAIN 출력에 자주 사용됩니다.

- 고유

  SELECT DISTINCT 쿼리 및 UNION 쿼리에 대한 중복을 제거합니다.

- 제한

  LIMIT 절을 처리합니다.

- Window

  창 함수를 실행합니다.

- 결과

  어떤 테이블 액세스에도 관련되지 않는 스칼라 함수를 실행합니다.

- Subplan

  특정 하위 쿼리에 사용됩니다.

- 네트워크

  추가 처리를 위해 중간 결과를 리더 노드로 전송합니다.

- Materialize

  중첩 루프 조인과 일부 병합 조인에 대한 입력 행을 저장합니다.

EXPLAIN의 조인 유형

쿼리 옵티마이저는 쿼리 구조 및 기본 테이블에 따라 다른 조인 유형을 사용하여 테이블 데이터를 가져옵니다. 그러면 EXPLAIN 출력이 조인 유형과 사용된 테이블, 그리고 테이블 데이터의 클러스터 분산 방식을 참조하여 쿼리 처리 방법을 설명합니다.

조인 유형 예

다음 예들은 쿼리 옵티마이저가 사용할 수 있는 여러 가지 조인 유형을 보여주고 있습니다. 쿼리 계획에서 사용되는 조인 유형은 관련 테이블의 물리적 설계에 따라 다릅니다.

예: 두 테이블의 해시 조인

다음은 CATID 열을 기준으로 EVENT와 CATEGORY를 조인하는 쿼리입니다. CATID는 CATEGORY에서는 분산 및 정렬 키이지만 EVENT에서는 그렇지 않습니다. 이때는 해시 조인이 EVENT를 외부 테이블로, 그리고 CATEGORY를 내부 테이블로 실행됩니다. CATEGORY가 작은 테이블이기 때문에 플래너는 쿼리를 처리하면서 DS_BCAST_INNER를 사용하여 CATEGORY 테이블의 복사본을 컴퓨팅 노드로 브로드캐스팅합니다. 이 예에서는 조인 비용이 계획의 누적 비용 대부분을 차지합니다.

```
explain select * from category, event where category.catid=event.catid;

                              QUERY PLAN
------------------------------------------------------------------------
 XN Hash Join DS_BCAST_INNER  (cost=0.14..6600286.07 rows=8798 width=84)
   Hash Cond: ("outer".catid = "inner".catid)
   ->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798 width=35)
   ->  XN Hash  (cost=0.11..0.11 rows=11 width=49)
         ->  XN Seq Scan on category  (cost=0.00..0.11 rows=11 width=49)
```

> **Note**
>
> EXPLAIN 출력에서 정렬되어 있는 연산자 들여쓰기는 각 작업들이 서로 종속되지 않고 병렬 방식으로 시작될 수 있다는 것을 의미하기도 합니다. 위 예에서는 EVENT 테이블에 대한 스캔과 해시 작업이 정렬되어 있지만 EVENT 스캔을 시작하려면 해시 작업이 완전히 끝날 때까지 기다려야 합니다.

예: 두 테이블의 병합 조인

다음 쿼리 역시 SELECT *를 사용하지만 LISTID 열을 기준으로 SALES와 LISTING을 조인합니다. 이 때 LISTID는 두 테이블 모두에 분산 및 정렬 키로 설정된 열입니다. 병합 조인이 선택되고, 데이터 재분산은 필요 없습니다(DS_DIST_NONE).

```
explain select * from sales, listing where sales.listid = listing.listid;
QUERY PLAN
------------------------------------------------------------------------------
XN Merge Join DS_DIST_NONE  (cost=0.00..6285.93 rows=172456 width=97)
  Merge Cond: ("outer".listid = "inner".listid)
  ->  XN Seq Scan on listing  (cost=0.00..1924.97 rows=192497 width=44)
  ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=53)
```

다음은 동일한 쿼리 내에서 다른 유형의 조인을 설명하는 예입니다. 위 예에서 보이듯이 SALES와 LISTING이 병합 조인되지만, 세 번째 테이블인 EVENT는 병합 조인 결과와 해시 조인되어야 합니다. 그 결과, 여기에서도 해시 조인에 따른 브로드캐스팅 비용이 발생합니다.

```
explain select * from sales, listing, event
where sales.listid = listing.listid and sales.eventid = event.eventid;
                                  QUERY PLAN
------------------------------------------------------------------------------
XN Hash Join DS_BCAST_INNER  (cost=109.98..3871130276.17 rows=172456 width=132)
  Hash Cond: ("outer".eventid = "inner".eventid)
  ->  XN Merge Join DS_DIST_NONE  (cost=0.00..6285.93 rows=172456 width=97)
        Merge Cond: ("outer".listid = "inner".listid)
        ->  XN Seq Scan on listing  (cost=0.00..1924.97 rows=192497 width=44)
        ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=53)
  ->  XN Hash  (cost=87.98..87.98 rows=8798 width=35)
        ->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798 width=35)
```

예: 조인, 집계 및 정렬

다음은 SALES 및 EVENT 테이블에 대한 해시 조인과, 그 뒤를 이어 분류된 SUM 함수와 ORDER BY 절을 설명하기 위한 집계 및 정렬 작업이 실행되는 쿼리입니다. 초기 Sort 연산자는 컴퓨팅 노드에서 병렬로 실행됩니다. 그런 다음 Network 연산자가 결과를 리더 노드에게 보내고 이어서 Merge 연산자가 최종 정렬 결과를 리더 노드에 산출합니다.

```
explain select eventname, sum(pricepaid) from sales, event
where sales.eventid=event.eventid group by eventname
order by 2 desc;
                                            QUERY PLAN
------------------------------------------------------------------------------
 XN Merge  (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
  Merge Key: sum(sales.pricepaid)
  ->  XN Network  (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
        Send to leader
        ->  XN Sort  (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
              Sort Key: sum(sales.pricepaid)
              ->  XN HashAggregate  (cost=2815366577.07..2815366578.51 rows=576
 width=27)
                    ->  XN Hash Join DS_BCAST_INNER  (cost=109.98..2815365714.80
 rows=172456 width=27)
                          Hash Cond: ("outer".eventid = "inner".eventid)
                          ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456
 width=14)
                          ->  XN Hash  (cost=87.98..87.98 rows=8798 width=21)
                                ->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798
 width=21)
```

데이터 재분산

조인에 대한 EXPLAIN 출력은 조인을 용이하게 하기 위해 데이터의 클러스터 이동 방식에 대한 메서드를 지정합니다. 이 데이터 이동은 브로드캐스팅 또는 재분산으로 수행될 수 있습니다. 브로드캐스팅에서는 한쪽 조인의 데이터 값이 각 컴퓨팅 노드에서 나머지 모든 컴퓨팅 노드로 복사되어 모든 컴퓨팅 노드에 대한 데이터 복사가 완료됩니다. 재분산에서는 데이터 값이 현재 조각에서 새로운 조각(다른 노드의 조각일 가능성이 높음)으로 전송됩니다. 조인 열 중 하나가 분산 키인 경우에는 조인에 참여하는 나머지 테이블의 분산 키와 일치하도록 데이터가 재분산됩니다. 두 테이블 모두 조인 열 중 하나에 분산 키가 없는 경우에는 두 테이블 모두 분산되거나, 혹은 내부 테이블이 모든 노드로 브로드캐스팅됩니다.

EXPLAIN 출력 또한 내부 테이블과 외부 테이블을 참조합니다. 내부 테이블이 먼저 스캔되어 쿼리 계획 하단 가까운 곳에 표시됩니다. 내부 테이블은 일치하는 조건을 탐색하는 대상 테이블입니다. 보통은 메모리에 저장되며, 일반적으로 해시 처리를 위한 원본 테이블인 동시에 가능하다면 조인하는 두 테이블 중 더 작은 용량의 테이블이기도 합니다. 외부 테이블은 내부 테이블을 대상으로 일치하는 행의 원본입니다. 일반적으로 디스크에서 읽어옵니다. 쿼리 옵티마이저는 최근 실행한 ANALYZE 명령의 데이터베이스 통계를 기준으로 내부 테이블과 외부 테이블을 선택합니다. 쿼리의 FROM 절에서 테이블 순서로는 내부 테이블과 외부 테이블을 구분할 수 없습니다.

쿼리를 용이하게 하기 위해 데이터의 이동 방식은 쿼리 계획에서 다음 속성을 사용하여 구분합니다.

- DS_BCAST_INNER

  전체 내부 테이블의 복사본이 모든 컴퓨팅 노드로 브로드캐스팅됩니다.

- DS_DIST_ALL_NONE

  이미 내부 테이블이 DISTSTYLE ALL을 사용하여 모든 노드에 분산되었기 때문에 재분산이 필요 없습니다.

- DS_DIST_NONE

  두 테이블 모두 재분산되지 않습니다. 해당 조각이 노드 간 데이터 이동 없이 조인되기 때문에 공동 배치되는 조인도 가능합니다.

- DS_DIST_INNER

  내부 테이블이 재분산됩니다.

- DS_DIST_OUTER

  외부 테이블이 재분산됩니다.

- DS_DIST_ALL_INNER

  외부 테이블이 DISTSTYLE ALL을 사용하기 때문에 내부 테이블 전체가 단일 조각으로 재분산됩니다.

- DS_DIST_BOTH

  두 테이블 모두 재분산됩니다.

쿼리 계획 단계 검토

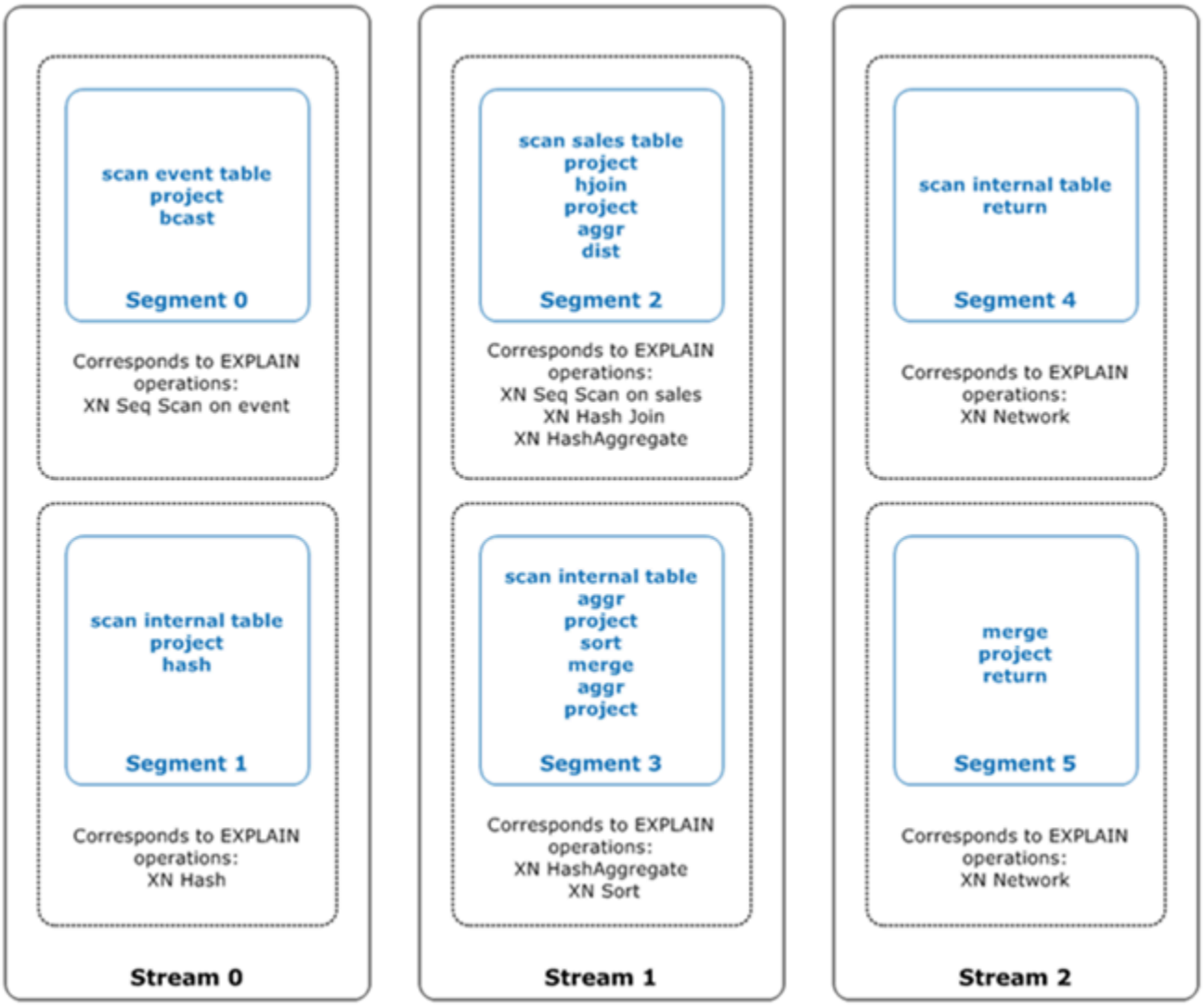
EXPLAIN 명령을 실행하면 쿼리 계획의 각 단계를 볼 수 있습니다. 다음 예는 SQL 쿼리를 보여 주며 출력을 설명합니다. 쿼리 계획을 아래에서 위로 읽어보면 쿼리를 실행하는 데 사용한 로직 작업을 개별적으로 확인할 수 있습니다. 자세한 내용은 쿼리 계획 생성 및 해석 섹션을 참조하세요.

```
explain
select eventname, sum(pricepaid) from sales, event
where sales.eventid = event.eventid
group by eventname
order by 2 desc;
```

```
XN Merge  (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
  Merge Key: sum(sales.pricepaid)
  ->  XN Network  (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
        Send to leader
        ->  XN Sort  (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
              Sort Key: sum(sales.pricepaid)
              ->  XN HashAggregate  (cost=2815366577.07..2815366578.51 rows=576
 width=27)
                    ->  XN Hash Join DS_BCAST_INNER  (cost=109.98..2815365714.80
 rows=172456 width=27)
                          Hash Cond: ("outer".eventid = "inner".eventid)
                          ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456
 width=14)
                          ->  XN Hash  (cost=87.98..87.98 rows=8798 width=21)
                                ->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798
 width=21)
```

쿼리 계획 생성의 일부로 쿼리 최적화 프로그램은 계획을 스트림, 세그먼트 및 단계로 나눕니다. 쿼리 최적화 프로그램은 컴퓨팅 노드에 데이터 및 쿼리 워크로드를 분산할 준비를 위해 계획을 세분화합니다. 스트림, 세그먼트 및 단계에 대한 자세한 내용은 쿼리 계획 및 실행 워크플로우 섹션을 참조하세요.

다음 그림은 위의 쿼리 및 관련 쿼리 계획을 보여줍니다. Amazon Redshift가 컴퓨팅 노드 조각에 대해 컴파일된 코드를 생성할 때 사용하는 단계에 쿼리 작업이 매핑되는 방법을 보여줍니다. 각 쿼리 계획 작업은 세그먼트 내 다수의 단계로 매핑되고, 간혹 스트림 내 다수의 세그먼트로 매핑되기도 합니다.



이 그림에서 쿼리 최적화 프로그램은 다음과 같이 쿼리 계획을 실행합니다.

1. Stream 0에서 쿼리는 순차적 스캔 작업과 함께 Segment 0을 실행하여 events 테이블을 스캔합니다. 쿼리는 해시 작업과 함께 Segment 1을 계속하여 조인에 내부 테이블에 대한 해시 테이블을 만듭니다.
2. Stream 1에서 쿼리는 순차적 스캔 작업과 함께 Segment 2을 실행하여 sales 테이블을 스캔합니다. 해시 조인과 함께 Segment 2를 계속하여 조인 열이 둘 다 분산 키 및 정렬 키가 아닌 테이블을 조인합니다. 해시 집계와 함께 Segment 2를 다시 계속하여 결과를 집계합니다. 그런 다음 쿼리는 해시 집계 작업과 함께 Segment 3을 실행하여 정렬되지 않은 그룹화된 집계 함수 및 정렬 작업을 수행하여 ORDER BY 절 및 다른 정렬 작업을 평가합니다.
3. Stream 2에서 쿼리는 Segment 4 및 Segment 5에서 네트워크 작업을 실행하여 추가 처리를 위해 중간 결과를 리더 노드로 보냅니다.

쿼리의 마지막 세그먼트는 데이터를 반환합니다. 반환 집합이 집계되거나 정렬되면 컴퓨팅 노드는 각각 중간 결과의 조각을 리더 노드로 보냅니다. 그런 다음 리더 노드는 최종 결과를 요청하는 클라이언트로 다시 전송할 수 있도록 데이터를 병합합니다.

EXPLAIN 연산자에 대한 자세한 내용은 EXPLAIN 섹션을 참조하세요.

쿼리 성능에 영향을 미치는 요인

쿼리 성능에 영향을 미칠 수 있는 요인은 매우 많습니다. 아래 설명하는 데이터, 클러스터 및 데이터베이스 작업 요소들은 모두 쿼리를 빠르게 처리하는 데 일조합니다.

- 노드, 프로세서 또는 조각 수 – 컴퓨팅 노드는 조각으로 분할됩니다. 노드가 많아질수록 프로세서와 조각의 수도 늘어난다는 것을 의미합니다. 그러면 쿼리를 다수의 조각으로 나눠 동시에 실행하기 때문에 쿼리의 처리 속도를 높일 수 있습니다. 하지만 노드 추가는 비용 증가를 의미하기 때문에 시스템에 적합한 가성비를 찾는 것이 중요합니다. Amazon Redshift 클러스터 아키텍처에 대한 자세한 내용은 데이터 웨어하우스 시스템 아키텍처 섹션을 참조하세요.
- 노드 유형 - Amazon Redshift 클러스터는 여러 노드 유형 중 하나를 사용할 수 있습니다. 각 노드 유형마다 크기와 제한이 다르기 때문에 상황에 따라 클러스터를 확장하는 데도 좋습니다. 노드 크기는 스토리지 용량, 메모리, CPU, 그리고 클러스터의 각 노드 요금을 결정합니다. 노드 유형에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 Amazon Redshift 클러스터의 개요 섹션을 참조하세요.
- 데이터 분산 - Amazon Redshift는 테이블의 배포 스타일에 따라 테이블 데이터를 컴퓨팅 노드에 저장합니다. 쿼리를 실행하면 쿼리 옵티마이저가 필요에 따라 데이터를 컴퓨팅 노드로 다시 분산시켜 조인 및 집계를 실행합니다. 테이블에 적합한 분산 스타일을 선택하면 조인에 앞서서 데이터를 필요한 것에 배치하여 재분산 단계의 영향을 최소화하는 데 효과적입니다. 자세한 내용은 쿼리 최적화를 위한 데이터 배포 섹션을 참조하세요.
- 데이터 정렬 순서 – Amazon Redshift는 테이블의 정렬 키에 따라 테이블 데이터를 정렬 순서대로 디스크에 저장합니다. 쿼리 옵티마이저와 쿼리 프로세서는 데이터 배치 정보를 사용하여 스캔할 블록 수를 줄이기 때문에 쿼리 속도를 높이는 데도 유용합니다. 자세한 내용은 정렬 키 섹션을 참조하세요.
- 데이터 집합 크기 – 클러스터의 데이터 볼륨이 높을수록 스캔 및 재분산이 필요한 행이 늘어나기 때문에 쿼리 성능이 느려질 수 있습니다. 이때는 주기적인 정리 작업(VACUUM)과 데이터 아카이빙, 그리고 조건자를 사용한 쿼리 데이터 세트 제한을 통해 이러한 성능 문제를 완화할 수 있습니다.
- 동시 작업 – 한 번에 다수의 작업을 실행하면 쿼리 성능에 영향을 미칠 수 있습니다. 각 작업마다 유효한 쿼리 대기열에서 슬롯을 1개 이상 사용할 뿐만 아니라 각 슬롯에 연결되어 있는 메모리를 사용하기 때문입니다. 다른 작업이 실행 중이라면 사용 가능한 쿼리 대기열 슬롯 수가 부족할 수 있습니다. 이로 인해 사용 가능한 슬롯이 열릴 때까지 기다린 후에 쿼리 처리를 시작할 수 있습니다. 쿼리 대기열의 생성 및 구성에 대한 자세한 내용은 워크로드 관리 섹션을 참조하세요.

- 쿼리 구조 – 쿼리 작성 방식 또한 성능에 영향을 미칩니다. 따라서 가능하다면 요건에 따라 최소한의 데이터를 처리하여 반환할 수 있도록 쿼리를 작성하는 것이 좋습니다. 자세한 내용은 Amazon Redshift 쿼리 설계 모범 사례 섹션을 참조하세요.
- 코드 컴파일 – Amazon Redshift는 각 쿼리 실행 계획마다 코드를 생성하여 컴파일합니다.

  컴파일된 코드는 인터프리터 사용에 따른 오버헤드를 제거하기 때문에 실행 속도가 더욱 빠릅니다. 단, 처음 코드를 생성 및 컴파일할 때는 오버헤드 비용이 일부 발생하기 마련입니다. 결과적으로 쿼리를 처음 실행할 때의 성능을 잘못 판단할 수가 있습니다. 오버헤드 비용은 특히 한 번 쿼리를 실행할 때 현저히 발생할 수 있습니다. 따라서 쿼리를 두 번 실행하여 일반적인 성능을 결정해야 합니다. Amazon Redshift는 서버리스 컴파일 서비스를 사용하여 Amazon Redshift 클러스터의 컴퓨팅 리소스 이상으로 쿼리 컴파일을 확장합니다. 컴파일된 코드 세그먼트는 클러스터와 거의 무제한 캐시에 로컬로 캐시됩니다. 이 캐시는 클러스터 재부팅 후에도 유지됩니다. 동일한 쿼리의 후속 실행은 컴파일 단계를 건너뛸 수 있기 때문에 더 빠르게 실행됩니다.

  캐시는 Amazon Redshift 버전 간에 호환되지 않으므로 컴파일 캐시가 플러시되고 버전 업그레이드 후 쿼리가 실행될 때 코드가 다시 컴파일됩니다. 쿼리에 엄격한 SLA가 있는 경우 클러스터 테이블의 데이터를 스캔하는 쿼리 세그먼트를 미리 실행하는 것이 좋습니다. 이를 통해 Amazon Redshift는 기본 테이블 데이터를 캐시하여 버전 업그레이드 후 쿼리를 계획하는 시간을 줄일 수 있습니다. Amazon Redshift는 확장 가능한 컴파일 서비스를 사용하여 코드를 병렬로 컴파일하여 일관되게 빠른 성능을 제공할 수 있습니다. 워크로드 속도 향상의 규모는 쿼리의 복잡성과 동시성에 따라 다릅니다.

쿼리 분석 및 개선 사항

Amazon Redshift 데이터 웨어하우스에서 정보를 가져올 때는 엄청난 대용량 데이터에 대한 복합 쿼리를 실행하기 때문에 프로세스 시간이 길어질 수 있습니다. 이러한 쿼리 프로세스의 속도를 최대한 높이기 위해 잠재적 성능 문제를 식별하는 데 사용할 수 있는 도구가 많이 있습니다.

주제

- 쿼리 분석 워크플로우
- 쿼리 알림 검토
- 쿼리 계획 분석
- 쿼리 요약 분석
- 쿼리 성능 개선
- 쿼리 튜닝을 위한 진단 쿼리

쿼리 분석 워크플로우

쿼리가 예상보다 오래 걸리면 다음 단계에 따라 쿼리 성능에 부정적인 영향을 미칠 수 있는 문제를 찾아 교정합니다. 시스템에서 튜닝을 통해 성능을 높일 수 있는 쿼리에 대해 확신이 없는 경우에는 먼저 튜닝에 가장 적합한 쿼리 식별 섹션의 진단 코드를 실행하세요.

1. 테이블이 모범 사례에 따라 설계되어 있는지 확인합니다. 자세한 내용은 Amazon Redshift 테이블 설계 모범 사례 섹션을 참조하세요.
2. 테이블에서 불필요한 데이터를 삭제하거나 아카이빙할 수 있는지 살펴봅니다. 예를 들어 쿼리가 항상 지난 6개월 분량의 데이터를 대상으로 하지만 테이블에는 지난 18개월 분량의 데이터가 저장되어 있다고 가정하겠습니다. 이러한 경우 이전 데이터를 삭제하거나 아카이브하여 스캔 및 분산해야 하는 레코드 수를 줄일 수 있습니다.
3. 쿼리에서 테이블에 대한 VACUUM 명령을 실행하여 공간을 회수한 다음 행을 재정렬합니다. 정렬되지 않은 영역이 크고, 쿼리가 조인 또는 조건자에서 정렬 키를 사용하는 경우, VACUUM을 실행하는 것이 좋습니다.
4. 쿼리에서 테이블에 대한 ANALYZE 명령을 실행하여 통계를 최신 상태로 유지합니다. 쿼리에서 테이블 하나라도 최근에 크기가 많이 변경된 경우에는 ANALYZE를 실행하는 것이 좋습니다. ANALYZE 명령 전체를 실행하는 데 시간이 너무 오래 걸린다면 단일 열에 대한 ANALYZE를 실행하여 처리 시간을 줄일 수 있습니다. 이러한 방법으로도 테이블 크기 통계가 업데이트됩니다. 테이블 크기는 쿼리 계획에서 중요한 요인입니다.
5. 각 유형별 클라이언트에 따라(클라이언트가 사용하는 연결 프로토콜 유형에 따라) 쿼리를 한 번씩 실행해야만 쿼리가 컴파일 및 캐싱됩니다. 그러면 이후 쿼리를 실행하는 속도가 빨라집니다. 자세한 내용은 쿼리 성능에 영향을 미치는 요인 섹션을 참조하세요.
6. STL_ALERT_EVENT_LOG 테이블을 확인하고 쿼리의 잠재적 문제를 찾아 교정합니다. 자세한 내용은 쿼리 알림 검토 섹션을 참조하세요.
7. EXPLAIN 명령을 실행하여 쿼리 계획을 가져와 쿼리를 최적화하는 데 사용합니다. 자세한 내용은 쿼리 계획 분석 섹션을 참조하세요.
8. SVL_QUERY_SUMMARY 및 SVL_QUERY_REPORT 뷰를 사용하여 요약 정보를 가져와 쿼리를 최적화하는 데 사용합니다. 자세한 내용은 쿼리 요약 분석 섹션을 참조하세요.

간혹 빠르게 실행해야 하데 다른 장기 실행 쿼리가 끝날 때까지 기다려야 하는 쿼리가 있는 경우도 있습니다. 이때는 쿼리 자체에서 개선할 방법은 없습니다. 하지만 쿼리 유형마다 다른 쿼리 대기열을 생성 및 사용하여 전체 시스템 성능을 높일 수는 있습니다. 쿼리 대기열의 대기 시간에 대한 자세한 내용은 쿼리의 대기열 대기 시간 검토 섹션을 참조하세요. 상태 확인 구성에 대한 자세한 내용은 워크로드 관리 섹션을 참조하세요.

쿼리 알림 검토

STL_ALERT_EVENT_LOG 시스템 테이블을 사용하여 쿼리의 잠재적 성능 문제를 찾아 교정하려면 다음 단계를 따릅니다.

1. 다음과 같이 실행하여 쿼리 ID를 확인합니다.

```
select query, elapsed, substring
from svl_qlog
order by query
desc limit 5;
```

   substring 필드에서 잘려있는 쿼리 텍스트를 검사하여 선택할 query 값을 결정합니다. 쿼리를 한 번 넘게 실행한 경우에는 query 값이 더 낮은 행의 elapsed 값을 사용하세요. 이 행이 컴파일 버전의 행입니다. 다수의 쿼리를 실행한 경우 쿼리의 포함 여부를 확인하는 데 사용하는 LIMIT 절에서 사용되는 값을 높일 수도 있습니다.

2. STL_ALERT_EVENT_LOG에서 쿼리에 사용할 행을 선택합니다.

```
Select * from stl_alert_event_log where query = MyQueryID;
```

userid	query	slice	segment	step	pid	xid	event	solution	event_time
100	32359	4	0	0	8780	71195	Very selective query filter:ratio=rows(2)/r	Review the choice of sort key to enable...	2015-02-10 17:40:50
100	32359	5	0	0	8781	71195	Very selective query filter:ratio=rows(2)/r	Review the choice of sort key to enable...	2015-02-10 17:40:50
100	109142	4	0	0	8780	302411	Very selective query filter:ratio=rows(2)/r	Review the choice of sort key to enable...	2015-02-24 20:32:28
100	109142	5	0	0	8781	302411	Very selective query filter:ratio=rows(2)/r	Review the choice of sort key to enable...	2015-02-24 20:32:28
100	109828	4	1	0	8746	304543	Very selective query filter:ratio=rows(3)/r	Review the choice of sort key to enable...	2015-02-24 23:27:52
100	109828	5	1	0	8747	304543	Very selective query filter:ratio=rows(3)/r	Review the choice of sort key to enable...	2015-02-24 23:27:52
100	109829	4	1	0	8760	304543	Very selective query filter:ratio=rows(3)/r	Review the choice of sort key to enable...	2015-02-24 23:28:01
100	109829	5	1	0	8761	304543	Very selective query filter:ratio=rows(3)/r	Review the choice of sort key to enable...	2015-02-24 23:28:01
100	113910	4	1	0	8774	316848	Very selective query filter:ratio=rows(3)/r	Review the choice of sort key to enable...	2015-02-25 17:14:58

3. 쿼리 결과를 평가합니다. 다음 표를 사용하여 식별된 문제를 해결할 수 있는 솔루션을 찾으세요.

> **Note**
>
> 모든 쿼리에 STL_ALERT_EVENT_LOG의 행이 있지는 않습니다. 식별된 문제가 있는 쿼리에만 행이 있습니다.

문제	이벤트 값	솔루션 값	권장 솔루션
쿼리에서 테이블 통계가 누락되었거나, 최신 상태가 아닙니다.	누락된 쿼리 플래너 통계	ANALYZE 명령 실행	테이블 통계 누락 또는 만료 섹

문제	이벤트 값	솔루션 값	권장 솔루션
			션을 참조하세요.
쿼리 계획에 중첩 루프 조인(가장 최적화되지 않은 조인)이 있습니다.	쿼리 계획에 중첩 루프 조인이 존재함	조인 조건자를 검토하여 데카르트 곱 회피	중첩 루프 섹션을 참조하세요.
스캔 작업이 정리가 아닌 삭제된 것으로 표시되었거나, 혹은 커밋되지 않고 삽입된 비교적 다수의 행을 건너뛰었습니다.	삭제된 행을 다수 스캔함	VACUUM 명령을 실행하여 삭제된 공간 회수	고스트 행 또는 커밋되지 않은 행 섹션을 참조하세요.
100만 개가 넘는 행이 해시 조인 또는 집계를 위해 재분산되었습니다.	다수의 행이 네트워크를 통해 분산됨(예: RowCount 행이 집계 처리를 위해 분산됨)	조인 또는 집계를 공동 배치하기 위한 분산 키의 선택 검토	최적이 아닌 데이터 분산 섹션을 참조하세요.
100만 개가 넘는 행이 해시 조인을 위해 브로드캐스팅되었습니다.	다수의 행이 네트워크를 통해 브로드캐스팅됨	조인을 공동 배치하기 위한 분산 키의 선택 검토 및 분산된 테이블의 사용 고려	최적이 아닌 데이터 분산 섹션을 참조하세요.
쿼리 계획에서 DS_DIST_ALL_INNER 재분산 스타일이 지정되면서 내부 테이블 전체가 단일 조각으로 재분산되었기 때문에 직렬 실행을 피할 수 없습니다.	쿼리 계획에 해시 조인을 위한 DS_DIST_ALL_INNER가 지정됨	외부보다는 내부 테이블 분산을 위한 전략 선택 검토	최적이 아닌 데이터 분산 섹션을 참조하세요.

쿼리 계획 분석

EXPLAIN 명령을 실행하여 쿼리 계획을 가져온 후

쿼리 계획을 분석하려면 먼저 쿼리 계획을 읽는 방법부터 알아야 합니다. 쿼리 계획을 읽는 방법에 대해서 잘 모르는 경우에는 본 섹션을 진행하기 전에 쿼리 계획 생성 및 해석부터 읽는 것이 좋습니다.

다음 단계에 따라 쿼리 계획에서 출력되는 데이터를 분석합니다.

1. 비용이 가장 높은 단계를 식별합니다. 나머지 단계를 진행하면서 식별된 단계를 최적화하는 데 집중해야 합니다.
2. 조인 유형을 살펴봅니다.
   - 중첩 루프: 이 조인이 주로 조인 조건이 생략되었을 때 발생합니다. 권장 솔루션은 중첩 루프 섹션을 참조하세요.
   - 해시 및 해시 조인: 해시 조인은 조인 열이 분산 키와 정렬 키가 아닌 테이블을 조인할 때 사용됩니다. 권장 솔루션은 해시 조인 섹션을 참조하세요.
   - 병합 조인: 변경할 필요 없습니다.
3. 내부 조인과 외부 조인에 어떤 테이블이 사용되는지 확인합니다. 쿼리 엔진은 일반적으로 작은 테이블을 내부 조인 용도로, 그리고 큰 테이블을 외부 조인 용도로 선택합니다. 이러한 선택과 다르다면 통계가 오랜 시간이 지났을 가능성이 높습니다. 권장 솔루션은 테이블 통계 누락 또는 만료 섹션을 참조하세요.
4. 비용이 높은 정렬 작업 유무를 확인합니다. 있을 경우 권장 솔루션은 정렬되지 않았거나 잘못 정렬된 행 섹션을 참조하세요.
5. 비용이 높은 작업이 있을 경우 다음과 같은 브로드캐스팅 연산자를 찾습니다.
   - DS_BCAST_INNER: 테이블이 모든 컴퓨팅 노드에 브로드캐스트되는지 나타냅니다. 이는 작은 테이블에는 좋지만 더 큰 테이블에는 이상적이지 않습니다.
   - DS_DIST_ALL_INNER: 모든 워크로드가 단일 조각으로 재분산되는 것을 의미합니다.
   - DS_DIST_BOTH: 과다한 재분산을 의미합니다.

   각 상황에 대한 권장 솔루션은 최적이 아닌 데이터 분산 섹션을 참조하세요.

쿼리 요약 분석

EXPLAIN에서 출력되는 쿼리 계획보다 더욱 자세하게 실행 단계 및 통계 정보를 가져오려면 SVL_QUERY_SUMMARY 및 SVL_QUERY_REPORT 시스템 뷰를 사용하세요.

SVL_QUERY_SUMMARY는 스트림 단위로 쿼리 통계 정보를 제공합니다. 이 정보를 사용하여 비용이 높은 단계, 장기 실행 단계, 그리고 디스크에 작성되는 단계의 문제를 식별할 수 있습니다.

SVL_QUERY_REPORT 시스템 뷰도 SVL_QUERY_SUMMARY와 유사한 정보를 제공하지만 스트림 단위가 아닌 컴퓨팅 노드 조각 단위로 이루어집니다. 이러한 조각 수준의 정보는 클러스터 간 고르지 않은 데이터 분산(데이터 분산 스큐라고도 함)을 감지하는 데 사용할 수 있습니다. 데이터 분산이 고르지 않으면 일부 노드가 다른 노드에 비해 작업량이 많아지면서 쿼리 성능이 떨어집니다.

주제

- SVL_QUERY_SUMMARY 뷰 사용
- SVL_QUERY_REPORT 뷰 사용
- 쿼리 계획을 쿼리 요약에 매핑

SVL_QUERY_SUMMARY 뷰 사용

SVL_QUERY_SUMMARY를 사용한 스트림의 쿼리 요약 정보를 분석하려면 다음과 같이 수행합니다.

1. 다음 쿼리를 실행하여 쿼리 ID를 확인합니다.

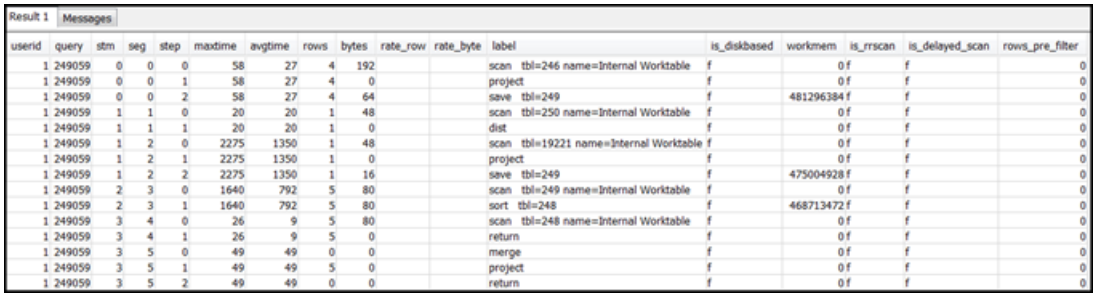
```
select query, elapsed, substring
from svl_qlog
order by query
desc limit 5;
```

   substring 필드에서 잘려있는 쿼리 텍스트를 검사하여 쿼리를 표현할 query 값을 결정합니다. 쿼리를 한 번 넘게 실행한 경우에는 query 값이 더 낮은 행의 elapsed 값을 사용하세요. 이 행이 컴파일 버전의 행입니다. 다수의 쿼리를 실행한 경우 쿼리의 포함 여부를 확인하는 데 사용하는 LIMIT 절에서 사용되는 값을 높일 수도 있습니다.

2. SVL_QUERY_SUMMARY에서 쿼리에 사용할 행을 선택합니다. 결과 순서는 stream, segment 및 step의 순으로 정하세요.

```
select * from svl_query_summary where query = MyQueryID order by stm, seg, step;
```

   다음은 예 결과입니다.



Result 1 | Messages

userid	query	stm	seg	step	maxtime	avgtime	rows	bytes	rate_row	rate_byte	label	is_diskbased	workmem	is_rrscan	is_delayed_scan	rows_pre_filter
1	249059	0	0	0	58	27	4	192			scan tbl=246 name=Internal Worktable	f	0	f	f	0
1	249059	0	0	1	58	27	4	0			project	f	0	f	f	0
1	249059	0	0	2	58	27	4	64			save tbl=249	f	481296384	f	f	0
1	249059	1	1	0	20	20	1	48			scan tbl=250 name=Internal Worktable	f	0	f	f	0
1	249059	1	1	1	20	20	1	0			dist	f	0	f	f	0
1	249059	1	2	0	2275	1350	1	48			scan tbl=19221 name=Internal Worktable	f	0	f	f	0
1	249059	1	2	1	2275	1350	1	0			project	f	0	f	f	0
1	249059	1	2	2	2275	1350	1	16			save tbl=249	f	475004928	f	f	0
1	249059	2	3	0	1640	792	5	80			scan tbl=249 name=Internal Worktable	f	0	f	f	0
1	249059	2	3	1	1640	792	5	80			sort tbl=248	f	468713472	f	f	0
1	249059	3	4	0	26	9	5	80			scan tbl=248 name=Internal Worktable	f	0	f	f	0
1	249059	3	4	1	26	9	5	0			return	f	0	f	f	0
1	249059	3	5	0	49	49	0	0			merge	f	0	f	f	0
1	249059	3	5	1	49	49	5	0			project	f	0	f	f	0
1	249059	3	5	2	49	49	0	0			return	f	0	f	f	0

3. 쿼리 계획을 쿼리 요약에 매핑 섹션의 정보를 사용하여 단계를 쿼리 계획의 작업으로 매핑합니다. 이때 각 단계는 행 값과 바이트 값이 대략적으로 동일해야 합니다(쿼리 계획의 행 * 폭). 그렇지 않은 경우 권장 솔루션은 테이블 통계 누락 또는 만료 섹션을 참조하세요.
4. 어떤 단계든지 `is_diskbased` 필드에 `t`(true) 값이 있는지 확인합니다. 쿼리 처리를 위해 시스템에 할당된 메모리가 충분하지 않을 경우 해시, 집계 및 정렬 연산자는 데이터를 디스크에 작성할 가능성이 높습니다.

   `is_diskbased`가 true인 경우 권장 솔루션은 쿼리에 할당되는 메모리 부족 섹션을 참조하세요.
5. `label` 필드 값을 살펴보면서 여러 단계 중 어디에서든지 AGG-DIST-AGG 시퀀스가 있는지 확인합니다. 이러한 시퀀스는 2단계 집계를 의미하는 것으로 비용이 높습니다. 이 문제를 해결하려면 GROUP BY 절을 변경하여 분산 키(키가 다수인 경우 첫 번째 키)를 사용하세요.
6. 각 세그먼트마다 `maxtime` 값을 살펴봅니다(이 값은 세그먼트를 구성하는 모든 단계에서 동일합니다). `maxtime` 값이 가장 높은 세그먼트를 찾아 각 단계에서 다음 연산자를 확인합니다.

> **Note**
>
> `maxtime` 값이 높다고 해서 반드시 세그먼트에 문제가 있다는 것을 의미하지는 않습니다. 높은 값에도 불구하고 세그먼트의 처리 시간이 오래 걸리지 않을 수도 있습니다. 스트림을 구성하는 세그먼트는 모두 시간을 동일하게 맞춥니다. 하지만 일부 다운스트림 세그먼트는 업스트림 세그먼트에서 데이터를 가져올 때까지 실행하지 못하는 경우도 있습니다. 이러한 세그먼트는 `maxtime` 값에 대기 시간과 처리 시간이 모두 포함되기 때문에 시간이 더 오래 걸리는 원인이 될 수도 있습니다.

- BCAST 또는 DIST: 두 경우 `maxtime` 값은 다수의 행을 재분산한 결과가 될 수 있습니다. 권장 솔루션은 최적이 아닌 데이터 분산 섹션을 참조하세요.
- HJOIN(해시 조인): 해당 단계의 `rows` 필드 값이 쿼리에서 최종 RETURN 단계의 `rows` 값에 비해 매우 높은 경우 권장하는 솔루션은 해시 조인 섹션을 참조하세요.

- SCAN/SORT: 조인 단계 바로 앞에서 SCAN, SORT, SCAN 및 MERGE 단계를 순서대로 찾습니다. 이 패턴은 정렬되지 않은 데이터가 스캔 및 정렬을 거쳐 정렬된 테이블 영역과 병합된다는 것을 의미합니다.

  SCAN 단계의 rows 값이 쿼리에서 최종 RETURN 단계의 rows 값에 비해 매우 높은지 확인합니다. 이러한 패턴은 실행 엔진이 나중에 무시되는 행을 스캔한다는 것을 의미하지만, 이는 비효율적입니다. 권장 솔루션은 불충분한 제한적 조건자 섹션을 참조하세요.

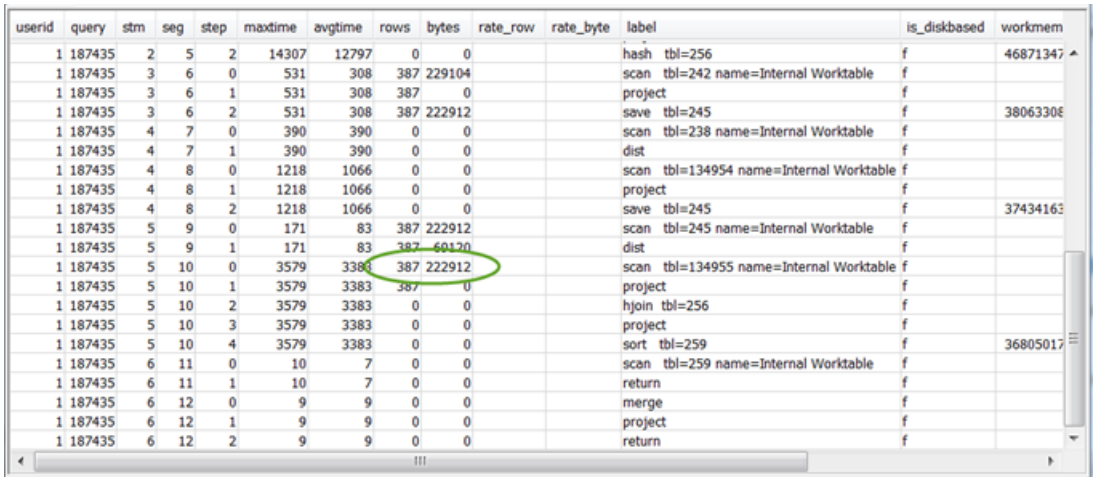
  SCAN 단계의 `maxtime` 값이 높은 경우 권장 솔루션은 최상이 아닌 WHERE 절 섹션을 참조하세요.

  SORT 단계의 `rows` 값이 0이 아닌 경우 권장 솔루션은 정렬되지 않았거나 잘못 정렬된 행 섹션을 참조하세요.

7. 최종 RETURN 단계 앞에 있는 5~10단계에서 `rows` 값과 `bytes` 값을 검토하여 클라이언트로 반환되는 데이터 크기를 파악합니다. 이 프로세스는 기교가 필요할 수 있습니다.

   예를 들어 다음 샘플 쿼리 요약을 보면 세 번째 PROJECT 단계에서 `bytes` 값이 아닌 `rows` 값이 제공됩니다. 이전 단계들에서 동일한 `rows` 값을 가진 단계를 살펴보며 행과 바이트 정보를 모두 제공하는 SCAN 단계를 찾습니다.

   다음은 결과 샘플입니다.



userid	query	stm	seg	step	maxtime	avgtime	rows	bytes	rate_row	rate_byte	label	is_diskbased	workmem
1	187435	2	5	2	14307	12797	0	0			hash tbl=256	f	46871347
1	187435	3	6	0	531	308	387	229104			scan tbl=242 name=Internal Worktable	f	
1	187435	3	6	1	531	308	387	0			project	f	
1	187435	3	6	2	531	308	387	222912			save tbl=245	f	38063308
1	187435	4	7	0	390	390	0	0			scan tbl=238 name=Internal Worktable	f	
1	187435	4	7	1	390	390	0	0			dist	f	
1	187435	4	8	0	1218	1066	0	0			scan tbl=134954 name=Internal Worktable	f	
1	187435	4	8	1	1218	1066	0	0			project	f	
1	187435	4	8	2	1218	1066	0	0			save tbl=245	f	37434163
1	187435	5	9	0	171	83	387	222912			scan tbl=245 name=Internal Worktable	f	
1	187435	5	9	1	171	83	387	69120			dist	f	
1	187435	5	10	0	3579	3383	387	222912			scan tbl=134955 name=Internal Worktable	f	
1	187435	5	10	1	3579	3383	387	0			project	f	
1	187435	5	10	2	3579	3383	0	0			hjoin tbl=256	f	
1	187435	5	10	3	3579	3383	0	0			project	f	
1	187435	5	10	4	3579	3383	0	0			sort tbl=259	f	36805017
1	187435	6	11	0	10	7	0	0			scan tbl=259 name=Internal Worktable	f	
1	187435	6	11	1	10	7	0	0			return	f	
1	187435	6	12	0	9	9	0	0			merge	f	
1	187435	6	12	1	9	9	0	0			project	f	
1	187435	6	12	2	9	9	0	0			return	f	

   비정상적으로 많은 데이터 볼륨을 반환하는 경우 권장 솔루션은 매우 큰 결과 집합 섹션을 참조하세요.

8. 어떤 단계에서든지 다른 단계와 비교하여 `bytes` 값이 `rows` 값에 비해 높은 경우가 있는지 확인합니다. 이 패턴은 다수의 열이 선택되어 있다는 것을 의미할 수 있습니다. 권장 솔루션은 큰 SELECT 목록 섹션을 참조하세요.

SVL_QUERY_REPORT 뷰 사용

SVL_QUERY_REPORT를 사용한 조각의 쿼리 요약 정보를 분석하려면 다음과 같이 수행합니다.

1. 다음과 같이 실행하여 쿼리 ID를 확인합니다.

```
select query, elapsed, substring
from svl_qlog
order by query
desc limit 5;
```

   substring 필드에서 잘려있는 쿼리 텍스트를 검사하여 쿼리를 표현할 query 값을 결정합니다. 쿼리를 한 번 넘게 실행한 경우에는 query 값이 더 낮은 행의 elapsed 값을 사용하세요. 이 행이 컴파일 버전의 행입니다. 다수의 쿼리를 실행한 경우 쿼리의 포함 여부를 확인하는 데 사용하는 LIMIT 절에서 사용되는 값을 높일 수도 있습니다.

2. SVL_QUERY_REPORT에서 쿼리에 사용할 행을 선택합니다. 결과 순서는 segment, step, elapsed_time 및 rows의 순으로 정하세요.

```
select * from svl_query_report where query = MyQueryID order by segment, step,
 elapsed_time, rows;
```

3. 각 단계마다 모든 조각이 대략적으로 동일한 수의 행을 처리하는지 확인합니다.

userid	query	slice	segment	step	start_time	end_time	elapsed_time	rows	bytes	label	is
100	141696	5	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	420	1100	31700	bcast	f
100	141696	1	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	437	1099	31812	bcast	f
100	141696	3	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	490	1066	30108	bcast	f
100	141696	6	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	576	1108	32316	bcast	f
100	141696	4	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	583	1128	32484	bcast	f
100	141696	0	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	726	1079	30804	bcast	f
100	141696	7	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	2109	1150	33300	bcast	f
100	141696	2	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	2406	1068	31056	bcast	f
100	141696	2	1	0	2014-09-12 18:45:33	2014-09-12 18:45:33	3441	8798	253580	scan tbl=95423 name=Internal Worktable	f

   또한 모든 조각에서 대략적으로 동일한 시간이 걸리는지도 확인합니다.

userid	query	slice	segment	step	start_time	end_time	elapsed_time	rows	bytes	label	is
100	141696	5	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	420	1100	31700	bcast	f
100	141696	1	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	437	1099	31812	bcast	f
100	141696	3	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	490	1066	30108	bcast	f
100	141696	6	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	576	1108	32316	bcast	f
100	141696	4	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	583	1128	32484	bcast	f
100	141696	0	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	726	1079	30804	bcast	f
100	141696	7	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	2109	1150	33300	bcast	f
100	141696	2	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	2406	1068	31056	bcast	f
100	141696	2	1	0	2014-09-12 18:45:33	2014-09-12 18:45:33	3441	8798	253580	scan tbl=95423 name=Internal Worktable	f

두 값의 차이가 클 경우에는 위의 특정 쿼리에 최적화되지 않은 분산 스타일로 인한 데이터 분산 스큐를 나타낼 수 있습니다. 권장 솔루션은 최적이 아닌 데이터 분산 섹션을 참조하세요.

쿼리 계획을 쿼리 요약에 매핑

쿼리 요약을 분석하는 경우 쿼리 계획의 작업에서 쿼리 요약의 단계(레이블 필드 값으로 식별)로 매핑하여 더 많은 세부 정보를 확보할 수 있습니다. 다음 테이블은 쿼리 계획 작업을 매핑하여 요약 단계를 쿼리합니다.

쿼리 계획 작업	레이블 필드 값	설명
Aggregate HashAggregate GroupAggregate	AGGR	집계 함수와 GROUP BY 조건을 평가합니다.
DS_BCAST_INNER	BCAST(브로드캐스팅)	테이블 전체 또는 일부 행 집합(테이블에서 필터링된 행 집합 등)을 모든 노드로 브로드캐스팅합니다.
쿼리 계획에 표시되지 않음	DELETE	행을 테이블에서 삭제합니다.
DS_DIST_NONE DS_DIST_ALL_NONE DS_DIST_INNER DS_DIST_ALL_INNER DS_DIST_ALL_BOTH	DIST(분산)	병렬 조인을 목적으로, 또는 기타 병렬 처리를 위해 행을 노드로 분산시킵니다.
HASH	HASH	해시 조인에서 사용할 목적으로 해시 테이블을 빌드합니다.
해시 조인	HJOIN(해시 조인)	두 테이블 또는 중간 결과 집합의 해시 조인을 수행합니다.

쿼리 계획 작업	레이블 필드 값	설명
쿼리 계획에 표시되지 않음	INSERT	행을 테이블에 삽입합니다.
Limit	LIMIT	LIMIT 절을 결과 집합에 적용합니다.
병합	MERGE	병렬 정렬 또는 조인 작업에서 도출되는 행을 병합합니다.
병합 조인	MJOIN(병합 조인)	두 테이블 또는 중간 결과 집합의 병합 조인을 수행합니다.
중첩 루프	NLOOP(중첩 루프)	두 테이블 또는 중간 결과 집합의 중첩 루프 조인을 수행합니다.
쿼리 계획에 표시되지 않음	PARSE	문자열 구문을 로드할 수 있는 이진 값으로 분석합니다.
프로젝트	PROJECT	표현식을 평가합니다.
네트워크	RETURN	행을 리더 또는 클라이언트로 반환합니다.
쿼리 계획에 표시되지 않음	SAVE	다음 처리 단계에서 사용할 수 있도록 행을 구체화합니다.
Seq Scan	SCAN	테이블 또는 중간 결과 집합을 스캔합니다.
정렬	SORT	이후 다른 작업(조인, 집계 등)에서 필요에 따라, 혹은 ORDER BY 절을 충족시킬 목적으로 행 또는 중간 결과 집합을 정렬합니다.

쿼리 계획 작업	레이블 필드 값	설명
고유	UNIQUE	다른 작업에서 필요에 따라 SELECT DISTINCT 절을 적용하거나, 혹은 중복을 제거합니다.
창	WINDOW	집계 및 순위 창 함수를 계산합니다.

쿼리 성능 개선

다음은 문제를 진단 및 해결하는 방법에 대한 설명과 함께 Amazon Redshift 쿼리 성능에 영향을 미치는 몇 가지 일반적인 문제입니다.

주제

- 테이블 통계 누락 또는 만료
- 중첩 루프
- 해시 조인
- 고스트 행 또는 커밋되지 않은 행
- 정렬되지 않았거나 잘못 정렬된 행
- 최적이 아닌 데이터 분산
- 쿼리에 할당되는 메모리 부족
- 최상이 아닌 WHERE 절
- 불충분한 제한적 조건자
- 매우 큰 결과 집합
- 큰 SELECT 목록

테이블 통계 누락 또는 만료

테이블 통계가 누락되었거나 이전 상태이면 다음과 같이 표시될 수 있습니다.

- EXPLAIN 명령 결과의 경고 메시지

- STL_ALERT_EVENT_LOG의 통계 누락 알림 이벤트. 자세한 내용은 쿼리 알림 검토 섹션을 참조하세요.

이 문제를 해결하려면 ANALYZE를 실행하세요.

중첩 루프

중첩 루프가 존재하는 경우 STL_ALERT_EVENT_LOG에 중첩 루프 알림 이벤트가 표시됩니다. 이러한 유형의 이벤트는 중첩 루프가 포함된 쿼리 식별에 있는 쿼리를 실행해도 식별할 수 있습니다. 자세한 내용은 쿼리 알림 검토 섹션을 참조하세요.

이 문제를 해결하려면 쿼리에서 크로스 조인 유무를 살펴본 후 가능하다면 제거하세요. 크로스 조인은 조인 조건이 없기 때문에 두 테이블의 데카르트 곱이 발생하는 원인이 됩니다. 또한 일반적으로 중첩 루프 조인으로 실행되기 때문에 가능한 조인 유형 중에서 속도가 가장 느립니다.

해시 조인

해시 조인이 존재하는 경우 다음과 같이 표시됩니다.

- 쿼리 계획에 해시 및 해시 조인 작업이 존재합니다. 자세한 내용은 쿼리 계획 분석 섹션을 참조하세요.
- SVL_QUERY_SUMMARY에 maxtime 값이 가장 높은 세그먼트의 HJOIN 단계가 존재합니다. 자세한 내용은 SVL_QUERY_SUMMARY 뷰 사용 섹션을 참조하세요.

이 문제를 해결하려면 다음과 같이 두 가지 방법을 사용할 수 있습니다.

- 가능하다면 쿼리를 재작성하여 병합 조인을 사용하세요. 분산 키인 동시에 정렬 키인 조인 열을 지정하면 가능합니다.
- SVL_QUERY_SUMMARY에서 HJOIN 단계의 rows 필드 값이 쿼리에서 최종 RETURN 단계의 rows 값에 비해 매우 높은 경우에는 쿼리를 재작성하여 고유한 열을 기준으로 조인할 수 있는지 확인하세요. 쿼리가 기본 키 같이 고유한 열을 기준으로 조인되지 않으면 조인에 참여하는 행의 수가 늘어납니다.

고스트 행 또는 커밋되지 않은 행

고스트 행 또는 커밋되지 않은 행이 존재하면 고스트 행이 지나치게 많다는 것을 나타내는 알림 이벤트가 STL_ALERT_EVENT_LOG에 표시됩니다. 자세한 내용은 쿼리 알림 검토 섹션을 참조하세요.

이 문제를 해결하려면 다음과 같이 두 가지 방법을 사용할 수 있습니다.

- Amazon Redshift 콘솔의 [로드(Loads)] 탭에서 쿼리 테이블에 대한 활성 로드 작업을 확인합니다. 활성 로드 작업이 있으면 끝날 때까지 기다린 후 다른 작업을 시작하세요.
- 활성 로드 작업이 없으면 쿼리 테이블에 대해 VACUUM을 실행하여 삭제된 행을 제거하세요.

정렬되지 않았거나 잘못 정렬된 행

정렬되지 않았거나 잘못 정렬된 행이 존재하면 STL_ALERT_EVENT_LOG에 선택의 폭이 매우 제한적인 필터 알림 이벤트가 표시됩니다. 자세한 내용은 쿼리 알림 검토 섹션을 참조하세요.

그 밖에 데이터 스큐 또는 미정렬 행이 포함된 테이블 식별에 있는 쿼리를 실행하여 쿼리 테이블 중 정렬되지 않은 영역이 많은 테이블이 있는지 알아보는 방법도 있습니다.

이 문제를 해결하려면 다음과 같이 두 가지 방법을 사용할 수 있습니다.

- 쿼리 테이블에 대해 VACUUM을 실행하여 행을 다시 정렬하세요.
- 쿼리 테이블의 정렬 키에서 개선할 수 있는 점이 있는지 살펴보세요. 단, 무엇이든 변경하기 전에 이 쿼리의 성능과 다른 중요한 쿼리 및 전반 시스템을 비교하여 검토해야 합니다. 자세한 내용은 정렬 키 섹션을 참조하세요.

최적이 아닌 데이터 분산

데이터 분산이 최적의 상태가 아니면 다음과 같이 표시됩니다.

- 직렬 실행, 대량 브로드캐스팅 또는 대량 분산 알림 이벤트가 STL_ALERT_EVENT_LOG에 표시됩니다. 자세한 내용은 쿼리 알림 검토 섹션을 참조하세요.
- 임의 단계에서 조각이 처리하는 행의 수가 대략적으로 동일하지 않습니다. 자세한 내용은 SVL_QUERY_REPORT 뷰 사용 섹션을 참조하세요.
- 임의 단계에서 조각이 처리하는 데 걸리는 시간이 대략적으로 동일하지 않습니다. 자세한 내용은 SVL_QUERY_REPORT 뷰 사용 섹션을 참조하세요.

위에서 설명한 것 중 하나라도 사실이 아니라면 데이터 스큐 또는 미정렬 행이 포함된 테이블 식별에 있는 쿼리를 실행하여 쿼리 테이블 중 데이터 스큐가 발생하는 테이블이 있는지 살펴보는 방법도 있습니다.

이 문제를 해결하려면 쿼리의 테이블에 대한 배포 스타일을 검토하고 개선할 수 있는 점이 있는지 확인합니다. 단, 무엇이든 변경하기 전에 이 쿼리의 성능과 다른 중요한 쿼리 및 전반 시스템을 비교하여 검토해야 합니다. 자세한 내용은 쿼리 최적화를 위한 데이터 배포 섹션을 참조하세요.

쿼리에 할당되는 메모리 부족

쿼리에 할당되는 메모리가 부족하면 SVL_QUERY_SUMMARY에 `is_diskbased` 값이 true인 단계가 표시됩니다. 자세한 내용은 SVL_QUERY_SUMMARY 뷰 사용 섹션을 참조하세요.

이 문제를 해결하려면 사용할 쿼리 수를 일시적으로 늘려서 쿼리에 더 많은 메모리를 할당하세요. 워크로드 관리(WLM)는 대기열에 설정되어 있는 동시성 레벨과 동일하게 쿼리 대기열의 슬롯을 보유합니다. 예를 들어 동시성 레벨이 5인 대기열은 슬롯 수도 5개입니다. 대기열에 할당되는 메모리는 균일하게 각 슬롯으로 분할됩니다. 하나의 쿼리에 다수의 슬롯을 할당하면 해당 쿼리는 할당된 모든 슬롯의 메모리에 대한 액세스 권한을 갖게 됩니다. 쿼리에 사용할 슬롯 수를 일시적으로 늘리는 방법에 대한 자세한 내용은 wlm_query_slot_count 섹션을 참조하세요.

최상이 아닌 WHERE 절

WHERE 절이 지나치게 많은 테이블 스캔을 초래하면 세그먼트에서 SCAN 단계가 `maxtime` 값이 가장 높은 것으로 SVL_QUERY_SUMMARY에 표시됩니다. 자세한 내용은 SVL_QUERY_SUMMARY 뷰 사용 섹션을 참조하세요.

이 문제를 해결하려면 가장 큰 테이블의 기본 정렬 열을 기준으로 WHERE 절을 쿼리에 추가하세요. 이 방법은 스캔 시간을 최소화하는 효과가 있습니다. 자세한 내용은 Amazon Redshift 테이블 설계 모범 사례 섹션을 참조하세요.

불충분한 제한적 조건자

쿼리에 불충분한 제한적 조건자가 있는 경우에는 세그먼트에서 SCAN 단계가 `maxtime` 값이 가장 높은 것으로 SVL_QUERY_SUMMARY에 표시되는 동시에 SVL_QUERY_SUMMARY에서 SCAN 단계의 `rows` 값이 쿼리에서 최종 RETURN 단계의 `rows` 값에 비해 매우 높습니다. 자세한 내용은 SVL_QUERY_SUMMARY 뷰 사용 섹션을 참조하세요.

이 문제를 해결하려면 쿼리에 조건자를 추가하거나, 혹은 기존 조건자의 제한을 높여서 출력 범위를 좁히세요.

매우 큰 결과 집합

쿼리가 매우 큰 결과 집합을 반환하는 경우에는 쿼리를 재작성하면서 UNLOAD를 사용하여 결과를 Amazon S3에 작성하는 것이 좋습니다. 이 방법은 병렬 처리를 이용해 RETURN 단계의 성능을 개선하는 효과가 있습니다. 매우 큰 결과 집합의 유무를 확인하는 방법에 대한 자세한 내용은 SVL_QUERY_SUMMARY 뷰 사용 섹션을 참조하세요.

큰 SELECT 목록

쿼리의 SELECT 목록이 비정상적으로 큰 경우에는 어떤 단계에서든지(다른 단계와 비교하여) bytes 값이 rows 값에 비해 비교적 높은 것으로 SVL_QUERY_SUMMARY에 표시됩니다. 이렇게 bytes 값이 높으면 선택한 열이 많다는 것을 나타낼 수 있습니다. 자세한 내용은 SVL_QUERY_SUMMARY 뷰 사용 섹션을 참조하세요.

이 문제를 해결하려면 선택한 열을 살펴보면서 제거할 수 있는 열이 있는지 확인하세요.

쿼리 튜닝을 위한 진단 쿼리

아래 쿼리들은 쿼리 성능에 영향을 미칠 수 있는 쿼리 또는 쿼리 테이블의 문제를 식별하는 데 사용됩니다. 이러한 쿼리는 쿼리 분석 및 개선 사항에서 언급한 쿼리 튜닝 프로세스와 함께 사용하는 것이 좋습니다.

> **Note**
>
> 이러한 쿼리는 Amazon Redshift 프로비저닝된 클러스터를 대상으로 합니다. 이러한 쿼리는 Redshift Serverless 작업 그룹에는 사용할 수 없습니다.

주제

- 튜닝에 가장 적합한 쿼리 식별
- 데이터 스큐 또는 미정렬 행이 포함된 테이블 식별
- 중첩 루프가 포함된 쿼리 식별
- 쿼리의 대기열 대기 시간 검토
- 테이블별 쿼리 알림 검토
- 통계가 누락된 테이블 식별

튜닝에 가장 적합한 쿼리 식별

다음은 지난 7일 동안 실행한 쿼리 문 중에서 가장 많은 시간이 소요된 문 50개를 구분하는 쿼리입니다. 결과를 사용하면 비정상적으로 오래 걸리는 쿼리를 식별할 수 있습니다. 또한 자주 실행되는 쿼리(결과 집합에 두 번 이상 나타나는 쿼리)를 식별할 수 있습니다. 이러한 쿼리들은 튜닝을 통해 시스템 성능을 개선하기 좋은 후보들입니다.

이 쿼리는 식별된 각 쿼리와 연결되어 있는 알림 이벤트의 수도 반환합니다. 이러한 알림 이벤트를 통해 쿼리 성능을 개선하는 데 필요한 세부 정보를 알아낼 수 있습니다. 자세한 내용은 쿼리 알림 검토 섹션을 참조하세요.

```
select trim(database) as db, count(query) as n_qry,
max(substring (qrytext,1,80)) as qrytext,
min(run_minutes) as "min" ,
max(run_minutes) as "max",
avg(run_minutes) as "avg", sum(run_minutes) as total,
max(query) as max_query_id,
max(starttime)::date as last_run,
sum(alerts) as alerts, aborted
from (select userid, label, stl_query.query,
trim(database) as database,
trim(querytxt) as qrytext,
md5(trim(querytxt)) as qry_md5,
starttime, endtime,
(datediff(seconds, starttime,endtime)::numeric(12,2))/60 as run_minutes,
alrt.num_events as alerts, aborted
from stl_query
left outer join
(select query, 1 as num_events from stl_alert_event_log group by query ) as alrt
on alrt.query = stl_query.query
where userid <> 1 and starttime >=  dateadd(day, -7, current_date))
group by database, label, qry_md5, aborted
order by total desc limit 50;
```

데이터 스큐 또는 미정렬 행이 포함된 테이블 식별

다음은 데이터 분산이 균일하지 못하거나(데이터 스큐), 정렬되지 않은 행의 비율이 높은 테이블을 찾아내는 쿼리입니다.

skew 값이 낮으면 테이블 데이터가 올바로 분산된 것을 의미합니다. 테이블의 skew 값이 4.00 이상이면 데이터 분산 스타일을 수정하는 것이 좋습니다. 자세한 내용은 최적이 아닌 데이터 분산 섹션을 참조하세요.

테이블의 pct_unsorted 값이 20%보다 높으면 VACUUM 명령을 실행하는 것이 좋습니다. 자세한 내용은 정렬되지 않았거나 잘못 정렬된 행 섹션을 참조하세요.

그 밖에도 각 테이블마다 mbytes 값과 pct_of_total 값을 살펴봐야 합니다. 이러한 열은 테이블 크기를 비롯해 원시 디스크에서 테이블이 사용하는 공간 비율을 나타냅니다. 원시 디스크 공간에는

Amazon Redshift가 내부 사용 목적으로 예약하는 공간도 포함되므로 사용자가 사용할 수 있는 디스크 공간 크기인 공칭 디스크 용량보다 더 커야 합니다. 이 정보를 사용하여 여유 디스크 공간이 가장 큰 테이블 크기의 2.5배 이상인지 확인합니다. 이 정도 크기의 공간을 사용할 수 있도록 유지하면 시스템이 복합 쿼리를 처리할 때도 중간 결과를 디스크에 작성할 수 있습니다.

```
select trim(pgn.nspname) as schema,
trim(a.name) as table, id as tableid,
decode(pgc.reldiststyle,0, 'even',1,det.distkey ,8,'all') as distkey,
 dist_ratio.ratio::decimal(10,4) as skew,
det.head_sort as "sortkey",
det.n_sortkeys as "#sks", b.mbytes,
decode(b.mbytes,0,0,((b.mbytes/part.total::decimal)*100)::decimal(5,2)) as
 pct_of_total,
decode(det.max_enc,0,'n','y') as enc, a.rows,
decode( det.n_sortkeys, 0, null, a.unsorted_rows ) as unsorted_rows ,
decode( det.n_sortkeys, 0, null, decode( a.rows,0,0, (a.unsorted_rows::decimal(32)/
a.rows)*100) )::decimal(5,2) as pct_unsorted
from (select db_id, id, name, sum(rows) as rows,
sum(rows)-sum(sorted_rows) as unsorted_rows
from stv_tbl_perm a
group by db_id, id, name) as a
join pg_class as pgc on pgc.oid = a.id
join pg_namespace as pgn on pgn.oid = pgc.relnamespace
left outer join (select tbl, count(*) as mbytes
from stv_blocklist group by tbl) b on a.id=b.tbl
inner join (select attrelid,
min(case attisdistkey when 't' then attname else null end) as "distkey",
min(case attsortkeyord when 1 then attname  else null end ) as head_sort ,
max(attsortkeyord) as n_sortkeys,
max(attencodingtype) as max_enc
from pg_attribute group by 1) as det
on det.attrelid = a.id
inner join ( select tbl, max(mbytes)::decimal(32)/min(mbytes) as ratio
from (select tbl, trim(name) as name, slice, count(*) as mbytes
from svv_diskusage group by tbl, name, slice )
group by tbl, name ) as dist_ratio on a.id = dist_ratio.tbl
join ( select sum(capacity) as  total
from stv_partitions where part_begin=0 ) as part on 1=1
where mbytes is not null
order by  mbytes desc;
```

중첩 루프가 포함된 쿼리 식별

다음은 중첩 루프에 대한 알림 이벤트가 기록된 쿼리를 식별하는 쿼리입니다. 중첩 루프 조건을 해결하는 방법에 대한 자세한 내용은 중첩 루프 섹션을 참조하세요.

```
select query, trim(querytxt) as SQL, starttime
from stl_query
where query in (
select distinct query
from stl_alert_event_log
where event like 'Nested Loop Join in the query plan%')
order by starttime desc;
```

쿼리의 대기열 대기 시간 검토

다음은 최근 쿼리가 실행에 앞서 쿼리 대기열의 슬롯이 열릴 때까지 대기한 시간을 나타내는 쿼리입니다. 대기 시간이 높은 추이가 발견되면 쿼리 대기열 구성을 수정하여 처리량을 개선하는 것이 좋습니다. 자세한 내용은 수동 WLM 구현 섹션을 참조하세요.

```
select trim(database) as DB , w.query,
substring(q.querytxt, 1, 100) as querytxt,  w.queue_start_time,
w.service_class as class, w.slot_count as slots,
w.total_queue_time/1000000 as queue_seconds,
w.total_exec_time/1000000 exec_seconds, (w.total_queue_time+w.total_Exec_time)/1000000
 as total_seconds
from stl_wlm_query w
left join stl_query q on q.query = w.query and q.userid = w.userid
where w.queue_start_Time >= dateadd(day, -7, current_Date)
and w.total_queue_Time > 0  and w.userid >1
and q.starttime >= dateadd(day, -7, current_Date)
order by w.total_queue_time desc, w.queue_start_time desc limit 35;
```

테이블별 쿼리 알림 검토

다음은 테이블 자체에 대한 알림 이벤트가 기록된 테이블을 찾아내는 동시에 가장 자주 기록되는 알림 유형을 식별하는 쿼리입니다.

식별된 테이블에서 행의 `minutes` 값이 높으면 테이블에서 해당 테이블에 대한 ANALYZE 또는 VACUUM 실행과 같은 일상적인 유지 관리가 필요한지를 확인합니다.

임의의 행에서 `count` 값이 높을 때 `table` 값이 NULL이라면 STL_ALERT_EVENT_LOG에서 연결된 `event` 값에 대한 쿼리를 실행하여 알림이 잦은 이유를 조사하세요.

```
select trim(s.perm_table_name) as table,
(sum(abs(datediff(seconds, s.starttime, s.endtime)))/60)::numeric(24,0) as minutes,
 trim(split_part(l.event,':',1)) as event,  trim(l.solution) as solution,
max(l.query) as sample_query, count(*)
from stl_alert_event_log as l
left join stl_scan as s on s.query = l.query and s.slice = l.slice
and s.segment = l.segment and s.step = l.step
where l.event_time >=  dateadd(day, -7, current_Date)
group by 1,3,4
order by 2 desc,6 desc;
```

통계가 누락된 테이블 식별

다음은 통계가 누락된 테이블에 대한 쿼리 수를 제공하는 쿼리입니다. 이 쿼리가 어떤 행이든 반환하는 경우에는 plannode 값을 살펴보면서 해당 테이블을 확인한 후 ANALYZE를 실행하세요.

```
select substring(trim(plannode),1,100) as plannode, count(*)
from stl_explain
where plannode like '%missing statistics%'
group by plannode
order by 2 desc;
```

쿼리 문제 해결

이 섹션은 Amazon Redshift 쿼리를 실행할 때 발생할 수 있는 가장 공통적이고 심각한 문제 몇 가지를 식별하여 해결할 수 있는 빠른 참조를 제공합니다.

주제

- 연결 실패
- 쿼리 중단
- 쿼리가 너무 오래 걸림
- 로드 실패
- 로드가 너무 오래 걸림
- 로드 데이터가 잘못됨
- JDBC Fetch Size 파라미터 설정

다음은 처음에 문제를 해결할 때 시도할 수 있는 권장 방법입니다. 자세한 내용은 다음 리소스를 참조할 수도 있습니다.

- Amazon Redshift 클러스터 및 데이터베이스 액세스
- 자동 테이블 최적화
- Amazon Redshift에서 데이터 로드
- 튜토리얼: Amazon S3에서 데이터 로드

연결 실패

다음과 같은 이유로 쿼리 연결이 실패할 수 있습니다. 먼저 다음 문제 해결 접근 방식을 따르는 것이 좋습니다.

클라이언트가 서버에 연결할 수 없음

SSL 또는 서버 인증서를 사용할 때는 먼저 연결 문제를 해결하면서 이러한 복잡성을 제거했다가 그런 다음 해결책을 발견하였을 때 SSL 또는 서버 인증서를 다시 추가합니다. 자세한 내용은 Amazon Redshift 관리 가이드의 연결에 대한 보안 옵션 구성 섹션을 참조하세요.

연결 거부

일반적으로 연결 구성에 실패했다는 오류 메시지가 수신되면 클러스터에 대한 액세스 권한과 관련된 문제를 의미합니다. 자세한 내용은 Amazon Redshift 관리 가이드의 연결이 거부되거나 실패함 섹션을 참조하세요.

쿼리 중단

다음과 같은 이유로 쿼리가 중단되거나 응답이 중지될 수 있습니다. 먼저 다음 문제 해결 접근 방식을 따르는 것이 좋습니다.

데이터베이스 연결 해제

최대 전송 단위(MTU)의 크기를 줄이세요. 단일 이더넷 프레임으로 네트워크 연결을 통해 전송할 수 있는 패킷의 최대 크기(바이트)는 MTU의 크기에 따라 결정됩니다. 자세한 내용은 Amazon Redshift 관리 가이드의 데이터베이스 연결이 끊어짐 섹션을 참조하세요.

데이터베이스 연결 시간 초과

COPY 명령 같은 긴 쿼리를 실행할 때는 데이터베이스에 대한 클라이언트 연결이 멈추거나 제한 시간에 걸릴 수 있습니다. 이런 경우 Amazon Redshift 콘솔에서 쿼리의 완료 여부를 관찰할 수 있지만 클라이언트 도구에는 쿼리가 여전히 실행 중인 것으로 표시됩니다. 쿼리 결과는 연결 중단 시점에 따라 누락되거나 불완전할 수도 있습니다. 이러한 문제는 중간 네트워크 구성 요소에서 유휴 상태의 연결을 종료할 때 발생합니다. 자세한 내용은 Amazon Redshift 관리 가이드의 방화벽 시간 제한 문제 섹션을 참조하세요.

ODBC에 클라이언트 측 메모리 부족 오류 발생

클라이언트 애플리케이션이 ODBC 연결을 사용하고 쿼리가 메모리에 비해 너무 큰 결과 집합을 생성하는 경우, 커서를 사용하여 결과 집합을 클라이언트 애플리케이션으로 스트리밍할 수 있습니다. 자세한 내용은 DECLARE 및 커서 사용 시 성능 고려사항 섹션을 참조하세요.

JDBC에 클라이언트 측 메모리 부족 오류 발생

JDBC 연결을 통해 대용량의 결과 집합을 가져오려고 하면 클라이언트 측 메모리 부족 오류가 발생할 수 있습니다. 자세한 내용은 JDBC Fetch Size 파라미터 설정 섹션을 참조하세요.

잠재적 교착 발생

잠재적 교착 상황이 발생하면 다음과 같이 시도하세요.

- STV_LOCKS 및 STL_TR_CONFLICT 시스템 테이블을 살펴보면서 테이블을 1개 이상 업데이트하는 데 따른 충돌 유무를 확인하세요.
- PG_CANCEL_BACKEND 함수를 사용하여 충돌하는 쿼리를 1개 이상 취소하세요.
- PG_TERMINATE_BACKEND 함수를 사용하여 세션을 종료하세요. 그러면 종료된 세션에서 실행 중이던 트랜잭션이 모든 잠금을 강제로 해제하여 트랜잭션을 롤백시킵니다.
- 주의하여 동시 쓰기 작업을 예약하세요. 자세한 내용은 동시 쓰기 작업 관리 섹션을 참조하세요.

쿼리가 너무 오래 걸림

다음과 같은 이유로 쿼리가 너무 오래 걸릴 수 있습니다. 먼저 다음 문제 해결 접근 방식을 따르는 것이 좋습니다.

테이블이 최적화되지 않음

테이블의 정렬 키, 분산 스타일 및 압축 인코딩을 설정하여 병렬 처리를 최대한 이용하세요. 자세한 내용은 자동 테이블 최적화 섹션을 참조하세요.

쿼리가 디스크에 쓰고 있음

쿼리는 쿼리 실행 중 일부분이라도 디스크에 작성할 수 있습니다. 자세한 내용은 쿼리 성능 개선 섹션을 참조하세요.

다른 쿼리가 끝날 때까지 쿼리가 대기해야 함

이때는 쿼리 대기열을 생성한 후 유형에 따라 쿼리를 적합한 대기열에 할당하면 전반적인 시스템 성능을 개선할 수 있습니다. 자세한 내용은 워크로드 관리 섹션을 참조하세요.

쿼리가 최적화되지 않음

실행 계획을 분석하여 쿼리를 재작성하거나 데이터베이스를 최적화하세요. 자세한 내용은 쿼리 계획 생성 및 해석 섹션을 참조하세요.

쿼리 실행에 더 많은 메모리 필요

특정 쿼리에 더 많은 메모리가 필요한 경우에는 wlm_query_slot_count 파라미터 값을 높여서 사용 가능한 메모리를 늘릴 수 있습니다.

데이터베이스에서 VACUUM 명령 실행 필요

정렬 키 순서에 따라 데이터를 로드하지 않는 경우에는 다수의 행을 추가, 삭제 또는 수정할 때마다 VACUUM 명령을 실행하세요. VACUUM 명령은 데이터를 재구성하여 정렬 순서를 유지하는 동시에 성능을 복원합니다. 자세한 내용은 테이블 Vacuum 단원을 참조하십시오.

장기 실행 쿼리 문제 해결을 위한 추가 리소스

다음은 쿼리 튜닝에 도움이 되는 시스템 뷰 주제 및 기타 설명서 섹션입니다.

- STV_INFLIGHT 시스템 뷰는 클러스터에서 실행 중인 쿼리를 보여줍니다. 현재 실행 중이거나 최근에 완료된 쿼리를 확인하려면 STV_RECENTS와 함께 사용하면 유용할 수 있습니다.
- SYS_QUERY_HISTORY는 문제 해결에 유용합니다. `running` 또는 `failed`와 같은 현재 상태, 각 쿼리가 실행되는 데 걸린 시간, 쿼리가 동시성 확장 클러스터에서 실행되었는지 여부와 같은 관련 속성과 함께 DDL 및 DML 쿼리를 표시합니다.
- STL_QUERYTEXT는 SQL 명령의 쿼리 텍스트를 수집합니다. 또한 STL_QUERYTEXT를 STV_INFLIGHT에 결합하는 SVV_QUERY_INFLIGHT는 더 많은 쿼리 메타데이터를 표시합니다.
- 트랜잭션 잠금 충돌은 쿼리 성능 문제의 원인이 될 수 있습니다. 현재 테이블에 잠금을 유지하고 있는 트랜잭션에 대한 자세한 내용은 SVV_TRANSACTIONS를 참조하세요.

- 조정에 가장 적합한 쿼리를 식별하면 최근에 실행한 쿼리 중 가장 많은 시간이 소요된 쿼리를 파악하는 데 도움이 되는 문제 해결 쿼리를 제공합니다. 이를 통해 개선이 필요한 쿼리에 노력을 집중할 수 있습니다.
- 쿼리 관리를 더 자세히 살펴보고 쿼리 대기열을 관리하는 방법을 이해하려면 워크로드 관리에서 그 방법을 확인하세요. 워크로드 관리는 고급 기능이며 대부분의 경우 자동화된 워크로드 관리를 권장합니다.

로드 실패

다음과 같은 이유로 데이터 로드가 실패할 수 있습니다. 먼저 다음 문제 해결 접근 방식을 따르는 것이 좋습니다.

데이터 원본이 다른 AWS 리전에 있습니다

기본적으로 COPY 명령에서 지정하는 Amazon S3 버킷 또는 Amazon DynamoDB 테이블은 클러스터와 동일한 AWS 리전에 있어야 합니다. 데이터와 클러스터가 서로 다른 리전에 있는 경우에는 다음과 유사한 오류 메시지가 표시됩니다.

```
The bucket you are attempting to access must be addressed using the specified endpoint.
```

가능하면 클러스터와 데이터 원본이 동일한 리전에 있어야 합니다. 그렇게 해도 COPY 명령에서 REGION 옵션을 사용하여 다른 리전을 지정할 수 있습니다.

> **Note**
>
> 클러스터와 데이터 원본이 서로 다른 AWS 리전에 있는 경우 데이터 전송 비용이 발생합니다. 또한 지연 시간이 더 깁니다.

COPY 명령 실패

STL_LOAD_ERRORS에 대한 쿼리를 실행하여 특정 로드 도중 발생한 오류를 식별합니다. 자세한 내용은 STL_LOAD_ERRORS 섹션을 참조하세요.

로드가 너무 오래 걸림

다음과 같은 이유로 로드 작업이 너무 오래 걸릴 수 있습니다. 먼저 다음 문제 해결 접근 방식을 따르는 것이 좋습니다.

COPY가 단일 파일에서 데이터를 로드함

로드 데이터를 여러 파일로 분할합니다. 하나의 큰 파일에서 모든 데이터를 로드하면 Amazon Redshift는 훨씬 느린 직렬화된 로드를 수행해야 합니다. 이때 파일 수는 클러스터 조각 수의 배수가 되어야 하며, 파일 크기는 압축 이후 1MB~1GB에서 대략적으로 동일해야 합니다. 자세한 내용은 Amazon Redshift 쿼리 설계 모범 사례 섹션을 참조하세요.

로드 작업에서 여러 개의 COPY 명령 사용

동시에 다수의 COPY 명령을 사용하여 여러 파일에서 테이블 하나를 로드하면 Amazon Redshift가 강제로 직렬화 로딩을 실행하여 속도가 느려집니다. 이때는 COPY 명령을 하나만 사용하세요.

로드 데이터가 잘못됨

COPY 작업은 다음과 같은 방법으로 잘못된 데이터를 로드할 수 있습니다. 먼저 다음 문제 해결 접근 방식을 따르는 것이 좋습니다.

잘못된 파일이 로드됨

객체 접두사를 사용하여 데이터 파일을 지정하면 원하지 않는 파일을 읽어올 수 있습니다. 이때는 매니페스트 파일을 사용하여 로드할 파일을 정확히 지정하세요. 자세한 내용은 COPY 명령의 copy_from_s3_manifest_file 옵션과 COPY 예의 Example: COPY from Amazon S3 using a manifest 섹션을 참조하세요.

JDBC Fetch Size 파라미터 설정

기본적으로 JDBC 드라이버는 모든 쿼리 결과를 한 번에 수집합니다. 그 결과, JDBC 연결을 통해 대용량의 결과 집합을 가져오려고 하면 클라이언트 측 메모리 부족 오류가 발생할 수 있습니다. 단 한 번에 모두 가져오거나 아무것도 가져오지 않는 방식이 아닌 배치(batch) 방식으로 결과 집합을 가져오려면 클라이언트 애플리케이션에서 JDBC Fetch Size 파라미터를 설정하세요.

> **Note**
>
> ODBC는 Fetch Size 파라미터가 지원되지 않습니다.

성능을 최적화하려면 메모리 부족 오류가 일어나지 않는 범위 내에서 페치 크기 값을 가장 높게 설정하세요. 페치 크기 값이 더 낮아지면 서버 전송이 늘어나서 실행 시간이 장기화될 수 있습니다. 서버는 클라이언트가 전체 결과 집합을 가져오거나 쿼리가 취소될 때까지 WLM 쿼리 슬롯이나 연결 메모리를 비

롯한 리소스를 예약합니다. 이때 Fetch Size 값을 적절히 조정하면 이러한 리소스가 더욱 빠르게 해제되어 다른 쿼리에서도 사용할 수 있게 됩니다.

> **Note**
>
> 대용량 데이터 집합을 추출해야 하는 경우에는 UNLOAD 문을 사용하여 데이터를 Amazon S3로 전송하는 것이 좋습니다. UNLOAD를 사용하면 컴퓨팅 노드가 병렬로 실행되어 데이터 전송 속도가 빨라집니다.

JDBC Fetch Size 파라미터 설정에 대한 자세한 내용은 PostgreSQL 설명서의 Getting results based on a cursor에서 확인할 수 있습니다.

워크로드 관리

자동 WLM 또는 수동 WLM으로 실행되도록 Amazon Redshift WLM을 구성할 수 있습니다.

Amazon Redshift를 사용하면 동시 쿼리 및 사용자 워크로드를 관리하고 우선순위를 지정하여 성능 및 리소스 사용률을 최적화할 수 있습니다. 워크로드 관리(WLM)를 사용하면 대기열, 사용자 그룹 및 기타 구문을 정의하여 다양한 유형의 쿼리 또는 사용자에게 할당된 리소스를 제어할 수 있습니다.

다음 섹션에서는 Amazon Redshift의 특정 워크로드 관리 기능을 간략하게 설명하고 구성 및 모니터링에 대해 알아봅니다.

자동 WLM

시스템 처리량을 극대화하고 리소스를 효율적으로 사용하려면 Amazon Redshift가 자동 WLM을 통해 동시 쿼리를 실행하기 위해 리소스를 나누는 방법을 관리하게 할 수 있습니다. 자동 WLM는 쿼리를 실행하는 데 필요한 리소스를 관리합니다. Amazon Redshift는 동시에 실행되는 쿼리의 수와 디스패치된 각 쿼리에 할당되는 메모리의 양을 결정합니다. 동시 쿼리를 실행하기 위해 리소스가 분할되는 방식을 Amazon Redshift가 관리하도록 하려면 자동 WLM을 사용하세요. 자세한 내용은 자동 WLM 구현 단원을 참조하십시오.

동시성 확장과 자동 WLM을 사용하면 일관되게 빠른 쿼리 성능으로 동시 사용자 및 동시 쿼리를 사실상 무제한 지원할 수 있습니다. 자세한 내용은 동시성 확장 단원을 참조하십시오.

> **Note**
>
> 대부분의 경우 자동 WLM을 사용하는 것이 좋습니다. 수동 WLM을 사용 중이고 자동 WLM으로 마이그레이션하려는 경우 수동 WLM에서 자동 WLM으로 마이그레이션 섹션을 참조하세요.

자동 WLM을 사용하면 대기열에 있는 워크로드의 쿼리 우선 순위를 정의할 수 있습니다. 쿼리 우선 순위에 대한 자세한 내용은 쿼리 우선 순위 섹션을 참조하세요.

수동 WLM

다수의 세션 또는 사용자가 동시에 쿼리를 실행할 수도 있습니다. 일부 쿼리는 장기간 클러스터 리소스를 소모하여 다른 쿼리의 성능에 영향을 미칠 수 있습니다. 특수한 사용 사례의 경우 수동 WLM을 통해 이를 관리할 수 있습니다. 동시성을 보다 세밀하게 제어하려면 수동 WLM을 사용하세요.

장기 실행 쿼리와 단기 실행 쿼리를 위한 별도의 대기열을 만들도록 WLM 구성을 수정하여 시스템 성능을 관리할 수 있습니다. 실행 시간에 쿼리를 사용자 그룹 또는 쿼리 그룹에 따라 이 대기열로 라우팅할 수 있습니다.

쿼리를 특정 대기열로 라우팅하는 규칙은 쿼리를 실행하는 사용자나 지정하는 레이블을 기준으로 설정할 수 있습니다. 또한 각 대기열에 할당되는 메모리 크기도 구성이 가능합니다. 따라서 대용량 쿼리를 실행하는 대기열에는 다른 대기열보다 많은 메모리를 할당하는 것이 좋습니다. 장기 실행 쿼리를 제한하기 위해 QMR(쿼리 모니터링 규칙)을 구성할 수도 있습니다. 자세한 내용은 수동 WLM 구현 단원을 참조하십시오.

> **Note**
>
> 총 15개 이하의 쿼리 슬롯이 있는 수동 WLM 쿼리 대기열을 구성하는 것이 좋습니다. 자세한 내용은 동시성 레벨 단원을 참조하십시오.

수동 WLM 구성과 관련하여 대기열에 할당할 수 있는 최대 슬롯은 50개라는 점에 유의하세요. 그러나 이는 자동 WLM 구성에서 Amazon Redshift 클러스터가 항상 50개의 쿼리를 동시에 실행한다는 의미는 아닙니다. 이는 클러스터의 메모리 요구 사항이나 기타 유형의 리소스 할당에 따라 변경될 수 있습니다.

주제

- WLM 모드 전환
- WLM 구성 수정
- 자동 WLM 구현
- 수동 WLM 구현
- 동시성 확장
- 단기 쿼리 가속화
- WLM 대기열 할당 규칙
- 대기열에 쿼리 할당
- WLM 동적 및 정적 구성 속성
- WLM 쿼리 모니터링 규칙
- WLM 시스템 테이블 및 뷰

WLM 모드 전환

Amazon Redshift 콘솔을 사용하여 자동 또는 수동 WLM을 활성화할 수 있습니다.

1. Switch WLM mode(WLM 모드 전환)를 선택합니다.
2. 자동 WLM으로 설정하려면 자동 WLM을 선택합니다. 그러면 최대 8개의 대기열이 쿼리를 관리하는 데 사용되고, 메모리 및 Concurrency on main(기본의 동시성) 필드가 모두 Auto(자동)로 설정됩니다. 또한 쿼리의 기본 우선 순위는 정상으로 설정됩니다.
3. Amazon Redshift 콘솔을 사용하여 수동 구성을 사용하려면 수동 WLM으로 전환합니다. 그런 다음 쿼리를 관리하는 데 사용되는 대기열과, Memory(메모리) 및 Concurrency on main(기본의 동시성) 필드 값을 지정합니다. 수동 구성에서는 쿼리 대기열을 최대 8개까지 구성하고, 각 대기열에서 동시에 실행할 수 있는 쿼리 수를 설정할 수 있습니다.

WLM 구성 수정

WLM 구성을 수정하는 가장 쉬운 방법은 Amazon Redshift 콘솔을 사용하는 것입니다. AWS CLI 또는 Amazon Redshift API를 사용할 수도 있습니다.

클러스터를 자동 WLM과 수동 WLM 간에 전환하면 클러스터가 `pending reboot` 상태가 됩니다. 변경 사항은 다음번 클러스터 재부팅까지 적용되지 않습니다.

WLM 구성 수정에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 워크로드 관리 구성 섹션을 참조하세요.

수동 WLM에서 자동 WLM으로 마이그레이션

시스템 처리량을 극대화하고 리소스를 최대한 효율적으로 사용하려면 해당 쿼리에 자동 WLM을 설정하는 것이 좋습니다. 수동 WLM에서 자동 WLM으로 원활하게 전환하도록 설정하려면 다음 접근 방식을 취하는 것을 고려하십시오.

수동 WLM에서 자동 WLM으로 마이그레이션하고 쿼리 우선 순위를 사용하려면 새로운 파라미터 그룹을 생성한 후 이 파라미터 그룹을 해당 클러스터에 연결하는 것이 좋습니다. 자세한 내용은 Amazon Redshift 관리 가이드의 Amazon Redshift 파라미터 그룹 섹션을 참조하세요.

> **Important**
>
> 파라미터 그룹을 변경하거나 수동 WLM에서 자동 WLM으로 전환하려면 클러스터 재부팅이 필요합니다. 자세한 내용은 WLM 동적 및 정적 구성 속성 단원을 참조하십시오.

예를 들어 세 개의 수동 WLM 대기열이 있는 경우를 살펴보겠습니다. 즉 ETL 워크로드, 분석 워크로드 및 데이터 과학 워크로드에 각각 한 개씩 있는 경우를 말합니다. ETL 워크로드는 6시간마다 실행되고, 분석 워크로드는 하루종일 실행되며, 데이터 과학 워크로드는 언제든지 급증할 수 있습니다. 수동 WLM을 사용해 비즈니스에 대한 각 워크로드의 중요도에 따라 각 워크로드 대기열이 차지할 메모리 및 동시성을 지정할 수 있습니다. 메모리 및 동시성을 지정하는 것은 까다로울 뿐만 아니라 클러스터 리소스가 통계적으로 분할되기 때문에 워크로드의 하위 집합만 실행되고 있을 때는 낭비되는 결과를 낳기도 합니다.

쿼리 우선 순위가 있는 자동 WLM을 사용하여 워크로드의 상대적 우선 순위를 나타냄으로써 이러한 문제를 피할 수 있습니다. 이와 같은 예의 경우 다음 절차를 따르십시오.

- 새로운 파라미터 그룹을 생성하고 Auto WLM(자동 WLM) 모드로 전환합니다.
- 세 가지 워크로드, 즉 ETL 워크로드, 분석 워크로드 및 데이터 과학 워크로드 각각에 대기열을 추가합니다. 수동 WLM 모드와 함께 사용된 각 워크로드에 동일한 사용자 그룹을 사용합니다.
- ETL 워크로드는 `High`, 분석 워크로드는 `Normal`, 데이터 과학 워크로드는 `Low`로 우선 순위를 설정합니다. 이러한 우선 순위는 다양한 워크로드 또는 사용자 그룹에 대한 비즈니스 우선 순위를 반영합니다.
- 선택적으로 ETL 워크로드가 6시간마다 실행되는 경우에도 이 대기열에 있는 쿼리가 일관된 성능을 발휘하도록 분석 또는 데이터 과학 대기열에 대해 동시성 확장을 활성화할 수 있습니다.

쿼리 우선 순위를 사용하면 분석 워크로드만 클러스터에서 실행 중인 경우 이 워크로드가 전체 시스템을 차지할 수 있습니다. 이렇게 하면 처리량이 많아지고 시스템 활용도가 향상됩니다. 그러나 ETL 워크로드가 시작하면 이 워크로드가 우선 순위가 더 높으므로 우선권을 얻습니다. ETL 워크로드의 일부로 실행되는 쿼리는 승인된 후에 우선적인 리소스 할당을 받을 뿐만 아니라 승인 중에도 우선권을 얻습니다. 결과적으로 ETL 워크로드는 시스템에서 다른 워크로드가 실행 중이어도 이와 상관없이 예상대로 성능을 발휘합니다. 우선 순위가 높은 워크로드가 예상대로 성능을 발휘하는 것은 더 오래 실행되는 더 낮은 우선 순위의 기타 워크로드로 인해 가능합니다. 이러한 워크로드의 쿼리는 더 중요한 쿼리가 완료되기를 뒤에서 기다리고 있거나 우선 순위가 더 높은 쿼리와 동시에 실행 중일 때 더 작은 리소스 조각을 얻기 때문입니다. Amazon Redshift에서 사용하는 예약 알고리즘을 통해 우선 순위가 더 낮은 쿼리가 리소스 결핍 없이 느린 속도로라도 계속 진행되도록 할 수 있습니다.

> **Note**
>
> - 제한 시간 필드는 자동 WLM에서 사용할 수 없습니다. 대신에 QMR 규칙인 `query_execution_time`을 사용합니다. 자세한 내용은 WLM 쿼리 모니터링 규칙 단원을 참조하십시오.

- QMR 작업인 HOP는 자동 WLM에 적용할 수 없습니다. 대신에 `change priority` 작업을 사용합니다. 자세한 내용은 WLM 쿼리 모니터링 규칙 단원을 참조하십시오.
- 클러스터는 자동 WLM과 수동 WLM 대기열을 서로 다르게 사용하므로 구성에 혼동이 발생할 수 있습니다. 예를 들어, 자동 WLM 대기열에서는 우선 순위 속성을 구성할 수 있지만 수동 WLM 대기열에서는 구성할 수 없습니다. 따라서 파라미터 그룹 내에서 자동 WLM 대기열과 수동 WLM 대기열을 섞지 마세요. 대신에 자동 WLM으로 마이그레이션할 때 새로운 파라미터 그룹을 생성하십시오.

자동 WLM 구현

자동 워크로드 관리(WLM)를 통해 Amazon Redshift는 쿼리 동시성과 메모리 할당을 관리합니다. 서비스 클래스 식별자 100~107로 최대 8개의 대기열을 생성할 수 없습니다. 대기열마다 우선 순위가 있습니다. 자세한 내용은 쿼리 우선 순위 단원을 참조하십시오.

자동 WLM을 통해 쿼리에 필요한 리소스의 양을 결정하고 워크로드에 근거하여 동시성을 조정합니다. 대량의 리소스가 필요한 쿼리가 시스템에 있는 경우(예: 큰 테이블 간 해시 조인) 동시성이 낮습니다. 더 가벼운 쿼리(예: 삽입, 삭제, 스캔 또는 간단한 집계 등)가 제출되는 경우 동시성이 높습니다.

자동 WLM은 단기 쿼리 가속화(SQA)와 분리되어 있으며 쿼리를 다른 방식으로 평가합니다. 자동 WLM 및 SQA는 함께 작동하여 장기 실행 중인 리소스 집약 쿼리가 활성화되어 있는 동안에도 단기 실행 및 경량 쿼리가 완료되게 합니다. SQA에 대한 자세한 내용은 단기 쿼리 가속화 섹션을 참조하세요.

Amazon Redshift에서는 파라미터 그룹을 통해 자동 WLM을 사용합니다.

- 클러스터에서 기본 파라미터 그룹을 사용하는 경우에서 Amazon Redshift는 클러스터에 대해 자동 WLM을 사용 설정합니다.
- 클러스터에서 사용자 지정 파라미터 그룹을 사용하는 경우 자동 WLM을 활성화하도록 클러스터를 구성할 수 있습니다. 자동 WLM 구성에 대해 별도의 파라미터 그룹을 생성하는 것이 좋습니다.

WLM을 구성하려면 1개 이상의 클러스터와 연결할 수 있는 파라미터 그룹에서 `wlm_json_configuration` 파라미터를 편집합니다. 자세한 내용은 WLM 구성 수정 단원을 참조하십시오.

WLM 구성 내에서 쿼리 대기열을 정의합니다. 쿼리 대기열은 기본 WLM 구성에 추가할 수 있으며, 이때 사용자 대기열의 최대 수는 8개입니다. 각 쿼리 대기열마다 다음과 같은 속성을 구성할 수 있습니다.

- 우선순위

- 동시성 확장 모드
- 사용자 그룹
- 쿼리 그룹
- 쿼리 모니터링 규칙

우선순위

우선 순위 값을 설정하여 워크로드 내에서 쿼리의 상대적 중요도를 정의할 수 있습니다. 우선 순위는 대기열에 대해 지정되고 이 대기열에 연결된 모든 쿼리에서 이 우선 순위를 상속합니다. 자세한 내용은 쿼리 우선 순위 단원을 참조하십시오.

동시성 확장 모드

동시성 크기 조정이 사용되면 동시 읽기 및 쓰기 쿼리의 증가를 처리하는 데 필요한 추가 클러스터 용량을 Amazon Redshift에서 자동으로 추가합니다. 쿼리가 기본 클러스터에서 실행되든 동시성 조정 클러스터에서 실행되든 사용자에게는 최신 데이터가 보입니다.

WLM 대기열을 구성하여 동시성 확장 클러스터에 보낸 쿼리를 관리합니다. 대기열에 동시성 크기 조정을 사용하면 대기열에서 기다리는 대신 적격 쿼리가 동시성 크기 조정 확장 클러스터로 전송됩니다. 자세한 내용은 동시성 확장 단원을 참조하십시오.

사용자 그룹

사용자 그룹은 사용자 그룹 이름을 지정하거나 와일드카드를 사용하여 대기열에 할당할 수 있습니다. 나열된 사용자 그룹의 멤버가 쿼리를 실행할 경우에는 해당 대기열에서 실행됩니다. 대기열에 할당할 수 있는 사용자 그룹의 수는 제한이 없습니다. 자세한 내용은 사용자 그룹을 기반으로 대기열에 쿼리 할당 단원을 참조하십시오.

사용자 역할

사용자 역할은 사용자 역할 이름을 지정하거나 와일드카드를 사용하여 대기열에 할당할 수 있습니다. 나열된 사용자 역할의 멤버가 쿼리를 실행할 경우에는 해당 대기열에서 실행됩니다. 대기열에 할당할 수 있는 사용자 역할의 수는 제한이 없습니다. 자세한 내용은 사용자 역할을 기반으로 대기열에 쿼리 할당 섹션을 참조하세요.

쿼리 그룹

쿼리 그룹은 쿼리 그룹 이름을 지정하거나 와일드카드를 사용하여 대기열에 할당할 수 있습니다. 쿼리 그룹이란 간단히 말해 레이블입니다. 실행 시간에 쿼리 그룹 레이블을 일련의 쿼리에 지정할 수 있습니

다. 목록에 있는 쿼리 그룹에 지정된 모든 쿼리는 해당 대기열에서 실행됩니다. 대기열에 할당할 수 있는 쿼리 그룹의 수는 제한이 없습니다. 자세한 내용은 쿼리 그룹에 쿼리 할당 단원을 참조하십시오.

와일드카드

WLM 대기열 구성에서 와일드카드가 사용 설정되어 있으면 사용자 그룹이나 쿼리 그룹을 개별적으로, 혹은 Unix 셸 스타일 와일드카드를 사용하여 할당할 수 있습니다. 패턴 일치는 대/소문자를 구분하지 않습니다.

예를 들어 '*' 와일드카드 문자는 모든 복수 문자와 일치합니다. 따라서 대기열의 사용자 그룹 목록에 dba_*를 추가하면 이름이 dba_로 시작되는 그룹에 속한 사용자 실행 쿼리는 모두 해당 대기열에 할당됩니다. dba_admin 또는 DBA_primary를 예로 들 수 있습니다. '?' 와일드카드 문자는 모든 단일 문자와 일치합니다. 따라서 대기열에 dba?1이라는 이름의 사용자 그룹이 할당되어 있으면 dba11이나 dba21 같은 사용자 그룹은 모두 일치하지만 dba12는 일치하지 않습니다.

기본적으로 와일드카드는 활성화되어 있지 않습니다.

쿼리 모니터링 규칙

쿼리 모니터링 규칙은 WLM 대기열의 지표 기반 성능 경계를 정의하고 쿼리가 해당 경계를 벗어날 때 실행할 작업을 지정합니다. 예를 들어 단시간 실행되는 쿼리 전용 대기열일 때는 60초 이상 실행되는 쿼리를 취소하는 규칙을 생성할 수도 있습니다. 그 밖에 잘못 설계된 쿼리를 추적할 목적으로 중첩 루프가 포함된 쿼리를 기록하는 규칙을 따로 만들 수도 있습니다. 자세한 내용은 WLM 쿼리 모니터링 규칙 단원을 참조하십시오.

자동 WLM 확인

자동 WLM이 활성화되었는지 확인하려면 다음 쿼리를 실행합니다. 쿼리에서 최소 1개의 행을 반환하면 자동 WLM이 활성화되어 있는 것입니다.

```
select * from stv_wlm_service_class_config
where service_class >= 100;
```

다음 쿼리는 각 쿼리 대기열(서비스 클래스)를 통해 전송된 쿼리의 수를 보여줍니다. 또한 평균 실행 시간, 90번째 퍼센타일에 대기 시간이 있는 쿼리 수, 평균 대기 시간도 보여줍니다. 자동 WLM 쿼리는 서비스 클래스 100~107을 사용합니다.

```
select final_state, service_class, count(*), avg(total_exec_time),
percentile_cont(0.9) within group (order by total_queue_time), avg(total_queue_time)
```

```
from stl_wlm_query where userid >= 100 group by 1,2 order by 2,1;
```

자동 WLM에서 실행했고 성공적으로 완료한 쿼리를 찾으려면 다음 쿼리를 실행합니다.

```
select a.queue_start_time, a.total_exec_time, label, trim(querytxt)
from stl_wlm_query a, stl_query b
where a.query = b.query and a.service_class >= 100 and a.final_state = 'Completed'
order by b.query desc limit 5;
```

쿼리 우선 순위

Amazon Redshift를 사용하면 워크로드 관리(WM)를 사용하여 동시 쿼리 및 워크로드 전반에서 쿼리 우선순위 지정 및 리소스 할당을 관리할 수 있습니다. 다음 섹션에서는 WM 쿼리 대기열을 구성하고, 메모리 할당 및 동시성 규모 조정 등의 대기열 속성을 정의하고, 워크로드 요구 사항에 맞는 우선순위 규칙을 구현하는 방법을 자세히 설명합니다.

모든 쿼리의 중요도가 동등한 것은 아니며 한 가지 워크로드 또는 사용자 집합의 성능이 더 중요한 경우가 많습니다. 자동 WLM을 활성화한 경우 우선 순위 값을 설정하여 워크로드 내에서 쿼리의 상대적 중요도를 정의할 수 있습니다. 우선 순위는 대기열에 대해 지정되고 이 대기열에 연결된 모든 쿼리에서 이 우선 순위를 상속합니다. 사용자 그룹 및 쿼리 그룹을 대기열에 매핑하여 쿼리를 대기열에 연결합니다. 우선 순위를 다음과 같이 설정할 수 있습니다(높은 순에서 낮은 순으로 나열됨).

1. HIGHEST
2. HIGH
3. NORMAL
4. LOW
5. LOWEST

동일한 리소스를 놓고 경합하는 다양한 우선 순위의 쿼리가 있을 때 관리자는 이러한 우선 순위를 사용해 워크로드의 상대적 중요도를 표시합니다. Amazon Redshift는 시스템에 쿼리를 허용할 때 우선 순위를 사용하고 쿼리에 할당된 리소스의 양을 결정합니다. 기본적으로 쿼리는 우선 순위가 NORMAL로 설정된 상태에서 실행됩니다.

HIGHEST보다 더 높은 우선 순위인 CRITICAL이라는 추가 우선 순위는 수퍼유저에게 제공됩니다. 이 우선 순위를 설정하려면 CHANGE_QUERY_PRIORITY, CHANGE_SESSION_PRIORITY 및 CHANGE_USER_PRIORITY 함수를 사용할 수 있습니다. 이 함수를 사용할 수 있는 권한을 데이터베이스 사용자에게 부여하려면 저장 프로시저를 생성하여 사용자에게 권한을 부여할 수 있습니다. 예시는 CHANGE_SESSION_PRIORITY을 확인하세요.

> **Note**
>
> 한 번에 하나의 CRITICAL 쿼리만 실행할 수 있습니다.
> 롤백은 항상 CRITICAL 우선순위로 실행됩니다.

ETL(추출, 변환 및 로드) 워크로드의 우선 순위가 분석 워크로드의 우선 순위보다 높은 경우를 예로 들어보겠습니다. ETL 워크로드는 6시간마다 실행되고, 분석 워크로드는 하루종일 실행됩니다. 분석 워크로드만 클러스터에서 실행 중인 경우 전체 이 워크로드는 전체 시스템을 차지함으로써 최적 시스템 사용률을 통해 높은 처리량을 산출합니다. 그러나 ETL 워크로드가 시작하면 이 워크로드가 우선 순위가 더 높으므로 우선권을 얻습니다. ETL 워크로드의 일부로 실행 중인 쿼리는 승인되는 중에, 그리고 승인된 후 기본 설정에 따른 리소스 할당 시에도 우선권을 얻습니다. 결과적으로 ETL 워크로드는 시스템에서 다른 워크로드가 실행 중이어도 이와 상관없이 예상대로 성능을 발휘합니다. 따라서 이를 통해 예측 가능한 성능을 얻을 수 있고 관리자는 비즈니스 사용자에게 SLA(서비스 수준 계약)를 제공할 수 있습니다.

특정 클러스터 내에서 우선 순위가 높은 워크로드의 예측 가능한 성능은 우선 순위가 더 낮은 기타 워크로드의 양보 덕분에 가능합니다. 우선 순위가 더 낮은 워크로드는 자체 쿼리가 중요도가 더 높은 쿼리가 완료될 때까지 뒤에서 대기하거나 우선 순위가 더 높은 쿼리와 동시에 실행 중일 때 더 작은 리소스 조각을 얻기 때문에 더 오래 실행될 수 있습니다. 우선 순위가 더 낮은 쿼리는 결핍에 시달리지 않고 더 느린 속도로 계속 진행됩니다.

앞의 예에서 관리자는 분석 워크로드에 대해 동시성 확장을 활성화할 수 있습니다. 이렇게 하면 ETL 워크로드가 높은 우선 순위에서 실행 중인 경우에도 분석 워크로드는 처리량을 유지할 수 있습니다.

대기열 우선 순위 구성

자동 WLM을 활성화한 경우 각 대기열에는 우선 순위 값이 있습니다. 쿼리는 사용자 그룹 및 쿼리 그룹에 따라 대기열에 라우팅됩니다. 대기열 우선 순위를 NORMAL로 설정하여 시작합니다. 대기열의 사용자 그룹 및 쿼리 그룹에 연결된 워크로드에 따라 더 높거나 더 낮은 우선 순위를 설정합니다.

Amazon Redshift 콘솔에서 대기열의 우선 순위를 변경할 수 있습니다. Amazon Redshift 콘솔의 [워크로드 관리(Workload Management)] 페이지에는 대기열이 표시되고 이 페이지에서 [우선 순위(Priority)]와 같은 대기열 속성을 편집할 수 있습니다. CLI 또는 API 작업을 사용해 우선 순위를 설정하려면 `wlm_json_configuration` 파라미터를 사용하십시오. 자세한 내용은 Amazon Redshift 관리 가이드의 워크로드 관리 구성 섹션을 참조하세요.

다음 wlm_json_configuration 예제에서는 세 가지 사용자 그룹(ingest, reporting 및 analytics)을 정의합니다. 이 그룹 중 하나에 속한 사용자가 제출한 쿼리는 각각 highest, normal 및 low 우선 순위로 실행됩니다.

```
[
    {
        "user_group": [
            "ingest"
        ],
        "priority": "highest",
        "queue_type": "auto"
    },
    {
        "user_group": [
            "reporting"
        ],
        "priority": "normal",
        "queue_type": "auto"
    },
    {
        "user_group": [
            "analytics"
        ],
        "priority": "low",
        "queue_type": "auto",
        "auto_wlm": true
    }
]
```

쿼리 모니터링 규칙을 이용해 쿼리 우선 순위 변경

QMR(쿼리 모니터링 규칙)을 통해 쿼리 실행 중에 쿼리의 동작에 근거하여 쿼리의 우선 순위를 변경할 수 있습니다. 이러한 변경은 QMR 조건자뿐 아니라 작업에서 우선 순위 속성을 지정하는 방식으로 수행할 수 있습니다. 자세한 내용은 WLM 쿼리 모니터링 규칙 단원을 참조하십시오.

예를 들어 10분 이상 실행되는 high 우선순위로 분류된 모든 쿼리를 취소하도록 규칙을 정의할 수 있습니다.

```
"rules" :[
  {
    "rule_name":"rule_abort",
    "predicate":[
```

```
      {
        "metric_name":"query_cpu_time",
        "operator":">",
        "value":600
      },
      {
        "metric_name":"query_priority",
        "operator":"=",
        "value":"high"
      }
    ],
    "action":"abort"
  }
]
```

또 다른 예는 1TB 이상을 디스크에 유출하며 현재 우선 순위가 normal인 모든 쿼리에 대해 쿼리 우선 순위를 lowest로 변경할 수 있는 규칙을 정의하는 것입니다.

```
"rules":[
  {
    "rule_name":"rule_change_priority",
    "predicate":[
      {
        "metric_name":"query_temp_blocks_to_disk",
        "operator":">",
        "value":1000000
      },
      {
        "metric_name":"query_priority",
        "operator":"=",
        "value":"normal"
      }
    ],
    "action":"change_query_priority",
    "value":"lowest"
  }
]
```

대기열 우선 순위 모니터링

쿼리 대기 및 실행에 대한 우선 순위를 표시하려면 stv_wlm_query_state 시스템 테이블에 있는 query_priority 열을 확인하십시오.

```
query    | service_cl | wlm_start_time             | state            | queue_time |
 query_priority
---------+------------+----------------------------+------------------+------------
+----------------
2673299  | 102        | 2019-06-24 17:35:38.866356 | QueuedWaiting    | 265116     |
 Highest
2673236  | 101        | 2019-06-24 17:35:33.313854 | Running          | 0          |
 Highest
2673265  | 102        | 2019-06-24 17:35:33.523332 | Running          | 0          |
 High
2673284  | 102        | 2019-06-24 17:35:38.477366 | Running          | 0          |
 Highest
2673288  | 102        | 2019-06-24 17:35:38.621819 | Running          | 0          |
 Highest
2673310  | 103        | 2019-06-24 17:35:39.068513 | QueuedWaiting    | 62970      |
 High
2673303  | 102        | 2019-06-24 17:35:38.968921 | QueuedWaiting    | 162560     |
 Normal
2673306  | 104        | 2019-06-24 17:35:39.002733 | QueuedWaiting    | 128691     |
 Lowest
```

완료된 쿼리의 쿼리 우선 순위를 나열하려면 stl_wlm_query 시스템 테이블에 있는 query_priority 열을 확인하십시오.

```
select query, service_class as svclass, service_class_start_time as starttime,
 query_priority
from stl_wlm_query order by 3 desc limit 10;
```

```
  query  | svclass |         starttime          |    query_priority
---------+---------+----------------------------+----------------------
 2723254 |     100 | 2019-06-24 18:14:50.780094 | Normal
 2723251 |     102 | 2019-06-24 18:14:50.749961 | Highest
 2723246 |     102 | 2019-06-24 18:14:50.725275 | Highest
 2723244 |     103 | 2019-06-24 18:14:50.719241 | High
 2723243 |     101 | 2019-06-24 18:14:50.699325 | Low
 2723242 |     102 | 2019-06-24 18:14:50.692573 | Highest
 2723239 |     101 | 2019-06-24 18:14:50.668535 | Low
 2723237 |     102 | 2019-06-24 18:14:50.661918 | Highest
 2723236 |     102 | 2019-06-24 18:14:50.643636 | Highest
```

워크로드의 처리량을 최적화하기 위해 Amazon Redshift는 사용자가 제출한 쿼리의 우선 순위를 수정할 수 있습니다. Amazon Redshift는 고급 기계 학습 알고리즘을 사용하여 이 최적화가 워크로드에 도움이 되는 시기를 결정하고 다음 조건이 모두 충족되면 자동으로 최적화를 적용합니다.

- 자동 WLM이 사용됩니다.
- 하나의 WLM 대기열만 정의됩니다.
- 쿼리 우선 순위를 설정하는 쿼리 모니터링 규칙(QMR)을 정의하지 않았습니다. 이러한 규칙에는 QMR 지표 `query_priority` 또는 QMR 작업 `change_query_priority`가 포함됩니다. 자세한 내용은 WLM 쿼리 모니터링 규칙 단원을 참조하십시오.

수동 WLM 구현

수동 WLM에서는 WLM 구성을 수정하여 장시간 실행되는 쿼리와 단시간 실행되는 쿼리에 대해 별도의 대기열을 생성함으로써 시스템 성능과 사용자 경험을 관리할 수 있습니다.

사용자가 Amazon Redshift에서 쿼리를 실행할 때는 쿼리가 쿼리 대기열로 라우팅됩니다. 각 쿼리 대기열에는 다수의 쿼리 슬롯이 포함되어 있습니다. 또한 클러스터에서 사용할 수 있는 메모리가 부분적으로 할당됩니다. 이렇게 할당된 대기열의 메모리는 다시 쿼리 슬롯으로 분할됩니다. Amazon Redshift가 자동 WLM으로 쿼리 동시성을 관리하도록 할 수 있습니다. 자세한 내용은 자동 WLM 구현 단원을 참조하십시오.

또는 쿼리 대기열마다 WLM 속성을 구성할 수 있습니다. 이를 통해 메모리가 여러 슬롯에 할당되는 방식과 런타임 시 쿼리가 특정 대기열로 라우팅되는 방식을 지정할 수 있습니다. 장기 실행되는 쿼리를 취소하도록 WLM 속성을 구성할 수도 있습니다.

Amazon Redshift에는 다음 쿼리 대기열이 기본적으로 구성되어 있습니다.

- 수퍼유저 대기열 1개

  수퍼유저 대기열은 수퍼유저 전용으로 예약되어 따로 구성할 수 없습니다. 시스템에 영향을 미치는 쿼리나 문제 해결이 목적인 쿼리를 실행할 때만 이 대기열을 사용합니다. 예를 들어 장시간 실행 중인 쿼리를 취소하거나 사용자에게 데이터베이스를 추가해야 할 때는 이 대기열을 사용하십시오. 일반적인 쿼리를 실행하는 데는 사용하지 마십시오. 대기열이 콘솔에는 표시되지 않지만 데이터베이스의 시스템 테이블에 5번째 대기열로 표시됩니다. 수퍼유저 대기열에서 쿼리를 실행하기 위해서는 사용자가 수퍼유저 권한으로 로그인한 후 사전 설정된 `superuser` 쿼리 그룹을 사용하여 쿼리를 실행해야 합니다.
- 기본 사용자 대기열 1개

기본 대기열에서는 동시에 실행할 수 있는 쿼리가 처음에 5개로 구성됩니다. 수동 WLM을 사용하면 기본 대기열의 동시성, 제한 시간 및 메모리 할당 속성을 변경할 수는 있지만 사용자 그룹이나 쿼리 그룹은 지정할 수 없습니다. 기본 대기열은 WLM 구성에서 마지막 대기열이 되어야 합니다. 다른 대기열로 라우팅되지 않는 쿼리는 모두 기본 대기열에서 실행됩니다.

쿼리 대기열은 WLM 구성에서 정의합니다. WLM 구성은 파라미터 그룹에서 편집이 가능한 파라미터(`wlm_json_configuration`)로서 클러스터 1개 이상과 연결할 수 있습니다. 자세한 내용은 Amazon Redshift 관리 가이드의 워크로드 관리 구성 섹션을 참조하세요.

쿼리 대기열은 기본 WLM 구성에 추가할 수 있으며, 이때 사용자 대기열의 최대 수는 8개입니다. 각 쿼리 대기열마다 다음과 같은 속성을 구성할 수 있습니다.

- 동시성 확장 모드
- 동시성 레벨
- 사용자 그룹
- 쿼리 그룹
- 사용할 WLM 메모리 비율
- WLM 제한 시간
- WLM 쿼리 대기열 건너뛰기
- 쿼리 모니터링 규칙

동시성 확장 모드

동시성 크기 조정이 사용되면 동시 읽기 및 쓰기 쿼리의 증가를 처리하는 데 필요한 추가 클러스터 용량을 Amazon Redshift에서 자동으로 추가합니다. 쿼리가 기본 클러스터에서 실행되든 동시성 확장 클러스터에서 실행되든 사용자에게는 최신 데이터가 보입니다.

WLM 대기열을 구성하여 동시성 확장 클러스터에 보낸 쿼리를 관리합니다. 대기열에 동시성 크기 조정을 사용하면 대기열에서 기다리는 대신 적격 쿼리가 동시성 크기 조정 확장 클러스터로 전송됩니다. 자세한 내용은 동시성 확장 단원을 참조하십시오.

동시성 레벨

대기열의 쿼리는 해당 대기열에 정의된 WLM 쿼리 슬롯 수 또는 동시성 레벨에 도달할 때까지 동시에 실행됩니다. 이후의 쿼리들은 대기열에서 대기합니다.

> **Note**
>
> WLM 동시 실행 수준은 한 클러스터에 만들 수 있는 동시 사용자 연결 수와 다릅니다. 자세한 내용은 Amazon Redshift 관리 가이드의 클러스터에 연결 섹션을 참조하세요.

자동 WLM 구성(권장)에는 동시성 수준이 자동으로 설정되어 있습니다. Amazon Redshift는 쿼리에 메모리를 동적으로 할당하고, 쿼리는 이후에 동시에 실행할 메모리를 결정합니다. 이는 실행 중인 쿼리와 대기열에 있는 쿼리에 모두 필요한 리소스를 기반으로 합니다. 자동 WLM은 구성할 수 없습니다. 자세한 내용은 자동 WLM 구현 단원을 참조하십시오.

수동 WLM 구성에서 Amazon Redshift는 고정된 양의 메모리를 각 대기열에 정적으로 할당합니다. 큐의 메모리는 쿼리 슬롯 간에 균등하게 분할됩니다. 예를 들어, 대기열에 클러스터 메모리의 20%가 할당되고 10개의 슬롯이 있는 경우 각 쿼리에는 클러스터 메모리의 2%가 할당됩니다. 메모리 할당은 동시에 실행되는 쿼리 수에 관계없이 고정된 상태로 유지됩니다. 고정된 메모리 할당으로 인해 슬롯 수가 5일 때 메모리에서만 실행되는 쿼리는 슬롯 수가 20으로 증가될 경우 중간 결과를 디스크에 작성해야 할 수도 있습니다. 이 경우 각 쿼리의 대기열 메모리 점유율은 1/5에서 1/20로 감소합니다. 결국 디스크 I/O가 추가로 발생하여 성능이 떨어지게 됩니다.

모든 사용자 정의 대기열의 최대 슬롯 수는 50입니다. 이렇게 하면 기본 대기열을 포함한 모든 대기열의 총 슬롯이 제한됩니다. 제한이 적용되지 않는 유일한 대기열은 예약된 수퍼유저 대기열입니다.

기본적으로 수동 WLM 대기열의 동시성 레벨은 5입니다. 하지만 다음과 같은 경우에는 동시성 레벨을 높여 워크로드의 이점을 얻을 수도 있습니다.

- 많은 수의 작은 쿼리가 장기 실행 쿼리를 대기해야 하는 경우 슬롯 수가 많은 별도의 대기열을 생성하고 작은 쿼리를 해당 대기열에 할당합니다. 동시성 레벨이 높은 대기열은 각 쿼리 슬롯에 할당되는 메모리 크기가 작기는 하지만 쿼리 용량이 낮을수록 필요한 메모리 크기 역시 작습니다.

  > **Note**
  >
  > 단기 쿼리 가속화(SQA)를 활성화하면 WLM은 자동으로 장기 실행 쿼리보다 단기 쿼리를 우선적으로 처리하므로, 대부분의 워크플로우에서 단기 쿼리에 별도의 대기열이 필요 없습니다. 자세한 내용은 단기 쿼리 가속화 단원을 참조하십시오.

- 단일 조각의 데이터에 각각 액세스하는 쿼리가 여러 개인 경우 이러한 쿼리를 동시에 실행하도록 별도의 WLM 대기열을 설정합니다. Amazon Redshift는 동시 쿼리를 별도의 조각에 할당하므로 여러 쿼리를 여러 조각에서 병렬로 실행할 수 있습니다. 예를 들어 분산 키를 기준으로 조건자를 사용한 단순 집계 쿼리인 경우에는 쿼리 데이터가 단일 조각에 저장됩니다.

수동 WLM 예제

이 예제는 슬롯과 메모리를 할당하는 방법을 보여주는 간단한 수동 WLM 시나리오입니다. 다음과 같은 세 개의 대기열로 수동 WLM을 구현합니다.

- data-ingestion 대기열 – 데이터 모으기를 위해 설정됩니다. 클러스터 메모리의 20%가 할당되고 5개의 슬롯이 있습니다. 이후 대기열에서 5개의 쿼리를 동시에 실행할 수 있으며 각 쿼리에는 메모리의 4% 가 할당됩니다.
- data-scientist 대기열 – 메모리를 많이 사용하는 쿼리를 위해 설계되었습니다. 클러스터 메모리의 40%가 할당되고 5개의 슬롯이 있습니다. 이후 5개의 쿼리를 동시에 실행할 수 있으며 각 쿼리에는 메모리의 8% 가 할당됩니다.
- default 대기열 – 조직 내 대다수 사용자를 위해 설계되었습니다. 여기에는 일반적으로 복잡하지 않은 단기 또는 중기 실행 쿼리를 사용하는 영업 및 회계 그룹이 포함됩니다. 클러스터 메모리의 40%가 할당되고 40개의 슬롯이 있습니다. 이 대기열에서는 40개의 쿼리를 동시에 실행할 수 있으며 각 쿼리에는 메모리의 1%가 할당됩니다. 대기열 전체의 제한은 50개이므로 이 대기열에 할당할 수 있는 최대 슬롯 수입니다.

자동 WLM을 실행 중이고 워크로드를 병렬로 실행하기 위해 15개가 넘는 쿼리가 필요한 경우 동시성 확장을 켜는 것이 좋습니다. 쿼리 슬롯 수가 15개를 초과하면 시스템 리소스에 대한 경합이 발생하여 단일 클러스터의 전체 처리량이 제한될 수 있기 때문입니다. 동시성 확장을 사용하면 동시성 확장 클러스터의 구성된 수까지 수백 개의 쿼리를 병렬로 실행할 수 있습니다. 동시성 확장 클러스터의 수는 max_concurrency_scaling_clusters에서 제어합니다. 동시성 조정에 대한 자세한 내용은 동시성 확장 섹션을 참조하세요.

자세한 내용은 쿼리 성능 개선 단원을 참조하십시오.

사용자 그룹

사용자 그룹은 사용자 그룹 이름을 지정하거나 와일드카드를 사용하여 대기열에 할당할 수 있습니다. 나열된 사용자 그룹의 멤버가 쿼리를 실행할 경우에는 해당 대기열에서 실행됩니다. 대기열에 할당할 수 있는 사용자 그룹의 수는 제한이 없습니다. 자세한 내용은 사용자 그룹을 기반으로 대기열에 쿼리 할당 단원을 참조하십시오.

사용자 역할

사용자 역할은 사용자 역할 이름을 지정하거나 와일드카드를 사용하여 대기열에 할당할 수 있습니다. 나열된 사용자 역할의 멤버가 쿼리를 실행할 경우에는 해당 대기열에서 실행됩니다. 대기열에 할당할

수 있는 사용자 역할의 수는 제한이 없습니다. 자세한 내용은 사용자 역할을 기반으로 대기열에 쿼리 할당 단원을 참조하십시오.

쿼리 그룹

쿼리 그룹은 쿼리 그룹 이름을 지정하거나 와일드카드를 사용하여 대기열에 할당할 수 있습니다. 쿼리 그룹이란 쉽게 말해서 레이블을 의미합니다. 실행 시간에 쿼리 그룹 레이블을 일련의 쿼리에 지정할 수 있습니다. 목록에 있는 쿼리 그룹에 지정된 모든 쿼리는 해당 대기열에서 실행됩니다. 대기열에 할당할 수 있는 쿼리 그룹의 수는 제한이 없습니다. 자세한 내용은 쿼리 그룹에 쿼리 할당 단원을 참조하십시오.

와일드카드

WLM 대기열 구성에서 와일드카드가 활성화되어 있으면 사용자 그룹이나 쿼리 그룹을 개별적으로, 혹은 Unix 셸 스타일 와일드카드를 사용하여 할당할 수 있습니다. 패턴 일치는 대/소문자를 구분하지 않습니다.

예를 들어 '*' 와일드카드 문자는 모든 복수 문자와 일치합니다. 따라서 대기열의 사용자 그룹 목록에 dba_*를 추가하면 이름이 dba_로 시작되는 그룹에 속한 사용자 실행 쿼리는 모두 해당 대기열에 할당됩니다. dba_admin 또는 DBA_primary를 예로 들 수 있습니다. '?' 와일드카드 문자는 모든 단일 문자와 일치합니다. 따라서 대기열에 dba?1이라는 이름의 사용자 그룹이 할당되어 있으면 dba11이나 dba21 같은 사용자 그룹은 모두 일치하지만 dba12는 일치하지 않습니다.

와일드카드는 기본적으로 해제되어 있습니다.

사용할 WLM 메모리 비율

자동 WLM 구성에서 메모리 비율은 **auto**로 설정되어 있습니다. 자세한 내용은 자동 WLM 구현 단원을 참조하십시오.

수동 WLM 구성에서 쿼리에 할당할 수 있는 메모리 크기를 지정하려면 WLM Memory Percent to Use 파라미터를 설정할 수 있습니다. 기본적으로 각 사용자 정의 대기열에는 사용자 정의 쿼리에 사용할 수 있는 메모리가 균일하게 할당됩니다. 예를 들어 사용자 정의 대기열이 4개라면 각 대기열마다 사용할 수 있는 메모리가 25%씩 할당됩니다. 수퍼유저 대기열에는 메모리가 따로 할당되며, 이를 수정할 수 없습니다. 할당 비율을 변경하려면 각 대기열마다 메모리를 총 100%까지 정수 비율로 할당하면 됩니다. 할당되지 않는 메모리는 Amazon Redshift에서 관리하며, 데이터 처리를 위해 추가 메모리를 요청하는 대기열에 임시로 할당됩니다.

예를 들어 대기열을 4개 구성할 경우 20%, 30%, 15%, 15%씩 메모리를 할당할 수 있습니다. 나머지 20%가 미할당 메모리가 되며, 서비스에서 관리합니다.

WLM 제한 시간

WLM 제한 시간(max_execution_time)은 더 이상 사용되지 않습니다. 대신 query_execution_time을 통해 QMR(쿼리 모니터링 규칙)을 생성하여 쿼리 실행 경과 시간을 제한하십시오. 자세한 내용은 WLM 쿼리 모니터링 규칙 단원을 참조하십시오.

임의의 WLM 대기열에서 쿼리가 사용할 수 있는 시간 크기를 제한할 때는 각 대기열마다 WLM 제한 시간 값을 설정할 수 있습니다. 시간 제한 파라미터는 Amazon Redshift가 쿼리를 취소하거나 건너뛸 때까지 쿼리 실행을 대기하는 시간(밀리초)을 지정합니다. 제한 시간은 쿼리 실행 시간에 따라 달라지며 대기열에서 대기하는 시간은 포함되지 않습니다.

WLM은 CREATE TABLE AS(CTAS) 문과 읽기 전용 쿼리(예: SELECT 문)만 건너뛰려고 합니다. 건너뛸 수 없는 쿼리는 취소됩니다. 자세한 내용은 WLM 쿼리 대기열 건너뛰기 단원을 참조하십시오.

WLM 제한 시간은 returning 상태에 도달한 쿼리에는 적용되지 않습니다. 커서의 상태를 보려면 STV_WLM_QUERY_STATE 시스템 테이블을 참조하십시오. COPY 문과 유지 관리 작업(예: ANALYZE 및 VACUUM)에는 WLM 제한 시간이 적용되지 않습니다.

WLM 제한 시간 함수는 statement_timeout 구성 파라미터와 비슷합니다. statement_timeout 구성 파라미터가 전체 클러스터에 적용되는 반면 WLM 제한 시간은 WLM 구성에 속한 단일 대기열에만 적용된다는 점이 다릅니다.

statement_timeout도 지정하는 경우에는 statement_timeout과 WLM 제한 시간(max_execution_time) 중에서 더 낮은 값이 사용됩니다.

쿼리 모니터링 규칙

쿼리 모니터링 규칙은 WLM 대기열의 지표 기반 성능 경계를 정의하고 쿼리가 해당 경계를 벗어날 때 실행할 작업을 지정합니다. 예를 들어 단시간 실행되는 쿼리 전용 대기열일 때는 60초 이상 실행되는 쿼리를 취소하는 규칙을 생성할 수도 있습니다. 그 밖에 잘못 설계된 쿼리를 추적할 목적으로 중첩 루프가 포함된 쿼리를 기록하는 규칙을 따로 만들 수도 있습니다. 자세한 내용은 WLM 쿼리 모니터링 규칙 단원을 참조하십시오.

WLM 쿼리 대기열 건너뛰기

Amazon Redshift를 사용하면 WLM(워크로드 관리) 쿼리 대기열 호핑을 사용하여 워크로드 동시성과 리소스 할당을 관리할 수 있습니다. 이 기능을 사용하면 리소스 사용이 가능할 때 쿼리가 할당된 대기열에서 우선순위가 더 높은 대기열로 일시적으로 '호핑'할 수 있으므로 전반적인 쿼리 성능과 시스템 사

용률이 향상됩니다. 다음 섹션에서는 Amazon Redshift에서 WLM 쿼리 대기열 호핑 구성 및 활용에 관한 자세한 가이드를 제공합니다.

WLM 시간 제한 또는 쿼리 모니터링 규칙(QMR) 건너뛰기 작업으로 인해 쿼리를 건너뛸 수 있습니다. 수동 WLM 구성에서는 쿼리를 건너뛸 수만 있습니다.

쿼리를 건너뛰면 WLM은 WLM 대기열 할당 규칙을 기반으로 다음에 일치하는 대기열에 쿼리를 라우팅하려고 합니다. 쿼리가 다른 어떤 대기열 정의와도 일치하지 않는 경우, 쿼리가 취소됩니다. 쿼리는 기본 대기열에 할당되지 않습니다.

WLM 제한 시간 작업

다음 표에는 다양한 유형의 쿼리 동작과 WLM 제한 시간이 요약되어 있습니다.

쿼리 유형	작업
INSERT, UPDATE 및 DELETE	취소
사용자 정의 함수(UDF)	취소
UNLOAD	취소
COPY	실행 계속
유지 관리 작업	실행 계속
`returning` 상태의 읽기 전용 쿼리	실행 계속
`running` 상태의 읽기 전용 쿼리	다시 할당 또는 다시 시작
CREATE TABLE AS (CTAS), SELECT INTO	다시 할당 또는 다시 시작

WLM 제한 시간 대기열 건너뛰기

WLM은 시간이 초과될 때 다음 유형의 쿼리를 건너뜁니다.

- WLM 상태가 `running`인 읽기 전용 쿼리(예: SELECT 문). 쿼리의 WLM 상태를 알고 싶다면 STV_WLM_QUERY_STATE 시스템 테이블에서 STATE 열을 확인하십시오.
- CREATE TABLE AS(CTAS) 문. WLM 대기열 건너뛰기는 사용자 정의 CTAS 문과 시스템 생성 CTAS 문을 모두 지원합니다.

- SELECT INTO 문

WLM 제한 시간이 적용되지 않는 쿼리는 완료될 때까지 원래 대기열에서 계속 실행됩니다. 다음 유형의 쿼리에는 WLM 제한 시간이 적용되지 않습니다.

- COPY 문
- 유지 관리 작업(예: ANALYZE 및 VACUUM)
- `returning` WLM 상태에 도달한 읽기 전용 쿼리(예: SELECT 문). 쿼리의 WLM 상태를 알고 싶다면 STV_WLM_QUERY_STATE 시스템 테이블에서 STATE 열을 확인하십시오.

WLM 시간 제한에 따른 건너뛰기를 사용할 수 없는 쿼리는 시간이 초과될 때 취소됩니다. 다음 유형의 쿼리는 WLM 제한 시간에 따른 건너뛰기를 사용할 수 없습니다.

- INSERT, UPDATE 및 DELETE 문
- UNLOAD 문
- 사용자 정의 함수(UDF)

WLM 제한 시간 다시 할당 및 다시 시작 쿼리

쿼리를 건너뛰고 일치하는 대기열을 찾을 수 없으면 쿼리가 취소됩니다.

쿼리를 건너뛰고 일치하는 대기열을 찾으면 WLM은 쿼리를 새 대기열에 다시 할당하려고 합니다. 쿼리를 다시 할당할 수 없으면 다음 설명과 같이 쿼리가 새 대기열에서 다시 시작됩니다.

다음이 모두 참인 경우에만 쿼리가 다시 할당됩니다.

- 일치하는 대기열이 발견되었습니다.
- 새로운 대기열에 쿼리를 실행할 수 있는 빈 슬롯이 있습니다. wlm_query_slot_count 파라미터를 1보다 큰 값으로 설정한 경우 한 쿼리에 여러 슬롯이 필요할 수 있습니다.
- 새 대기열에서 최소한 쿼리가 현재 사용하는 것과 같은 양의 메모리를 사용할 수 있습니다.

쿼리를 다시 할당하면 쿼리는 새 대기열에서 실행을 계속합니다. 중간 결과가 보존되므로, 총 실행 시간에 대한 효과는 최소화됩니다.

쿼리를 다시 할당할 수 없으면 쿼리가 취소되고 새 대기열에서 다시 시작됩니다. 중가나 결과는 삭제됩니다. 쿼리는 대기열에서 대기한 다음 충분한 슬롯을 사용할 수 있을 때 실행을 시작합니다.

QMR 건너뛰기 작업

다음 표에는 다양한 유형의 쿼리 동작과 QMR 건너뛰기 작업이 요약되어 있습니다.

쿼리 유형	작업
COPY	실행 계속
유지 관리 작업	실행 계속
사용자 정의 함수(UDF)	실행 계속
UNLOAD	다시 할당 또는 실행 계속
INSERT, UPDATE 및 DELETE	다시 할당 또는 실행 계속
`returning` 상태의 읽기 전용 쿼리	다시 할당 또는 실행 계속
`running` 상태의 읽기 전용 쿼리	다시 할당 또는 다시 시작
CREATE TABLE AS (CTAS), SELECT INTO	다시 할당 또는 다시 시작

QMR에서 건너뛴 쿼리가 다시 할당되었는지, 다시 시작되었는지 또는 취소되었는지 알아보려면 STL_WLM_RULE_ACTION 시스템 로그 테이블을 쿼리합니다.

QMR 건너뛰기 작업 다시 할당 및 다시 시작 쿼리

쿼리를 건너뛰고 일치하는 대기열을 찾을 수 없으면 쿼리가 취소됩니다.

쿼리를 건너뛰고 일치하는 대기열을 찾으면 WLM은 쿼리를 새 대기열에 다시 할당하려고 합니다. 쿼리를 다시 할당할 수 없으면 다음 설명과 같이 쿼리가 새 대기열에서 다시 시작되거나 원래 대기열에서 실행을 계속합니다.

다음이 모두 참인 경우에만 쿼리가 다시 할당됩니다.

- 일치하는 대기열이 발견되었습니다.
- 새로운 대기열에 쿼리를 실행할 수 있는 빈 슬롯이 있습니다. wlm_query_slot_count 파라미터를 1보다 큰 값으로 설정한 경우 한 쿼리에 여러 슬롯이 필요할 수 있습니다.
- 새 대기열에서 최소한 쿼리가 현재 사용하는 것과 같은 양의 메모리를 사용할 수 있습니다.

쿼리를 다시 할당하면 쿼리는 새 대기열에서 실행을 계속합니다. 중간 결과가 보존되므로, 총 실행 시간에 대한 효과는 최소화됩니다.

쿼리를 다시 할당할 수 없으면 쿼리가 다시 시작되거나 원래 대기열에서 실행을 계속합니다. 쿼리를 다시 시작하면 쿼리가 취소되고 새 대기열에서 다시 시작됩니다. 중가나 결과는 삭제됩니다. 쿼리는 대기열에서 대기한 다음 충분한 슬롯을 사용할 수 있을 때 실행을 시작합니다.

자습서: 수동 워크로드 관리(WLM) 대기열 구성

Amazon Redshift를 사용하면 수동 워크로드 관리(WLM) 대기열을 구성하여 다양한 유형의 쿼리 및 사용자에 대한 리소스의 우선순위를 지정하고 할당할 수 있습니다. 수동 WLM 대기열을 사용하면 특정 대기열에 대한 메모리 및 동시성 설정을 제어할 수 있으므로 중요한 워크로드가 필요한 리소스를 수신하면서 우선순위가 낮은 쿼리가 시스템을 독점하는 것을 방지할 수 있습니다. 다음 섹션에서는 워크로드 관리 요구 사항을 충족하기 위해 Amazon Redshift에서 수동 WLM 대기열을 생성하고 구성하는 프로세스를 안내합니다.

개요

Amazon Redshift에서 자동 워크로드 관리(WLM)를 구성하는 것이 좋습니다. 자동 WLM에 대한 자세한 내용은 워크로드 관리 섹션을 참조하세요. 그러나 WLM 대기열이 여러 개 필요한 경우 이 튜토리얼에서는 Amazon Redshift에서 수동 워크로드 관리(WLM)를 구성하는 과정을 안내합니다. 수동 WLM을 구성하면 클러스터의 쿼리 성능과 리소스 할당을 개선할 수 있습니다.

Amazon Redshift는 사용자 쿼리를 처리할 수 있도록 대기열로 라우팅합니다. 이러한 쿼리의 대기열 라우팅 방식을 정의하는 것이 바로 WLM입니다. 기본적으로 Amazon Redshift에는 슈퍼 사용자용 대기열, 사용자용 대기열 등 쿼리에 사용할 수 있는 대기열이 2개 있습니다. 수퍼유저 대기열은 따로 구성할 수 없으며 한 번에 처리할 수 있는 쿼리의 수도 1개로 제한됩니다. 이 대기열은 문제 해결 목적으로만 예약해야 합니다. 사용자 대기열은 한 번에 5개까지 쿼리를 처리할 수 있지만 필요하다면 대기열의 동시성 레벨을 변경하여 다르게 구성할 수도 있습니다.

데이터베이스에서 다수의 사용자가 쿼리를 실행하고 있다면 더욱 효율적인 구성이 있는지 찾아보십시오. 예를 들어 일부 사용자가 VACUUM 같이 리소스 집약적인 작업을 실행할 경우에는 보고서 같이 리소스를 덜 사용하는 쿼리는 부정적인 영향을 받을 수 있습니다. 이때는 대기열을 추가하여 다른 워크로드에 구성하는 것이 좋습니다.

예상 소요 시간: 75분

예상 비용: 50센트

사전 조건

Amazon Redshift 클러스터, 샘플 TICKIT 데이터베이스 및 psql 클라이언트 도구가 필요합니다. 아직 이 세 가지가 준비되지 않았다면 Amazon Redshift 시작 안내서와 Amazon Redshift RSQL을 참조하세요.

Sections

단원 1: 기본적인 대기열 처리 동작에 대한 이해

수동 WLM 구성을 시작하기 전에 먼저 Amazon Redshift의 기본적인 대기열 처리 동작에 대해서 알아두는 것이 좋습니다. 이번 단원에서는 몇 가지 시스템 테이블에서 정보를 반환하는 데이터베이스 뷰 2개를 생성합니다. 그런 다음 테스트 쿼리를 몇 차례 실행하여 기본적으로 쿼리가 라우팅되는 방식을 알아봅니다. 시스템 테이블에 대한 자세한 내용은 시스템 테이블 및 뷰 참조 섹션을 참조하세요.

1단계: WLM_QUEUE_STATE_VW 뷰 생성

이번 단계에서는 WLM_QUEUE_STATE_VW라는 이름의 뷰를 생성합니다. 이 뷰는 다음 시스템 테이블에서 정보를 반환합니다.

- STV_WLM_CLASSIFICATION_CONFIG
- STV_WLM_SERVICE_CLASS_CONFIG
- STV_WLM_SERVICE_CLASS_STATE

이 뷰는 자습서 전체에서 WLM 구성 변경 후 대기열을 모니터링하는 데 사용됩니다. 다음 표는 WLM_QUEUE_STATE_VW 뷰가 반환하는 데이터에 대해 설명한 것입니다.

열	설명
대기열	행과 연결되어 대기열을 나타내는 번호입니다. 대기열 번호에 따라 데이터베이스의 대기열 순서가 결정됩니다.

열	설명
설명	대기열을 특정 사용자 그룹 또는 특정 쿼리 그룹에게만 제공할지, 혹은 모든 유형의 쿼리에게 제공할지 설명하는 값입니다.
slots	대기열에 할당되는 슬롯 수입니다.
mem	대기열에 할당되는 슬롯당 메모리 크기(MB)입니다.
2013년 7월 15일	쿼리 종료 이전에 허용되는 실행 시간입니다.
user_*	사용자 그룹 일치를 위한 WLM 구성 시 와일드카드 문자의 허용 여부를 나타내는 값입니다.
query_*	쿼리 그룹 일치를 위한 WLM 구성 시 와일드카드 문자의 허용 여부를 나타내는 값입니다.
queued	대기열에서 처리를 기다리는 쿼리 수입니다.
executing	현재 실행 중인 쿼리 수.
executed	실행된 쿼리 수.

WLM_QUEUE_STATE_VW 뷰를 생성하려면

1. Amazon Redshift RSQL을 열고 TICKIT 샘플 데이터베이스에 연결합니다. 이 역할이 구성되지 않은 경우 사전 조건 섹션을 참조하세요.
2. 다음 쿼리를 실행하여 WLM_QUEUE_STATE_VW 뷰를 생성합니다.

```
create view WLM_QUEUE_STATE_VW as
select (config.service_class-5) as queue
, trim (class.condition) as description
, config.num_query_tasks as slots
, config.query_working_mem as mem
, config.max_execution_time as max_time
, config.user_group_wild_card as "user_*"
, config.query_group_wild_card as "query_*"
, state.num_queued_queries queued
, state.num_executing_queries executing
, state.num_executed_queries executed
```

```
from
STV_WLM_CLASSIFICATION_CONFIG class,
STV_WLM_SERVICE_CLASS_CONFIG config,
STV_WLM_SERVICE_CLASS_STATE state
where
class.action_service_class = config.service_class
and class.action_service_class = state.service_class
and config.service_class > 4
order by config.service_class;
```

3. 다음 쿼리를 실행하여 뷰에 포함된 정보를 확인합니다.

```
select * from wlm_queue_state_vw;
```

다음은 예 결과입니다.

```
query | description                                | slots | mem | max_time | user_* |
 query_* | queued | executing | executed
------+--------------------------------------------+-------+-----+----------+--------
+---------+--------+-----------+----------
    0 | (super user) and (query group: superuser) |     1 | 357 |        0 |  false |
 false   |      0 |         0 |        0
    1 | (querytype:any)                           |     5 | 836 |        0 |  false |
 false   |      0 |         1 |      160
```

2단계: WLM_QUERY_STATE_VW 뷰 생성

이번 단계에서는 WLM_QUERY_STATE_VW라는 이름의 뷰를 생성합니다. 이 뷰는 STV_WLM_QUERY_STATE 시스템 테이블에서 정보를 반환합니다.

이 뷰는 자습서 전체에서 실행 중인 쿼리를 모니터링하는 데 사용됩니다. 다음 표는 WLM_QUERY_STATE_VW 뷰가 반환하는 데이터에 대해 설명한 것입니다.

열	설명
쿼리	쿼리 ID.
대기열	대기열 번호입니다.
slot_count	쿼리에 할당되는 슬롯 수입니다.

열	설명
start_time	쿼리가 시작된 시간입니다.
state	쿼리 상태(executing 등)입니다.
queue_time	쿼리가 대기열에서 대기한 시간(마이크로초)입니다.
exec_time	쿼리가 실행된 시간(마이크로초)

WLM_QUERY_STATE_VW 뷰를 생성하려면

1. RSQL에서 다음 쿼리를 실행하여 WLM_QUERY_STATE_VW 뷰를 생성합니다.

```
create view WLM_QUERY_STATE_VW as
select query, (service_class-5) as queue, slot_count, trim(wlm_start_time) as
 start_time, trim(state) as state, trim(queue_time) as queue_time, trim(exec_time) as
 exec_time
from stv_wlm_query_state;
```

2. 다음 쿼리를 실행하여 뷰에 포함된 정보를 확인합니다.

```
select * from wlm_query_state_vw;
```

다음은 예 결과입니다.

```
query | queue | slot_count | start_time          | state     | queue_time | exec_time
------+-------+------------+---------------------+-----------+------------
+-----------
 1249 |     1 |          1 | 2014-09-24 22:19:16 | Executing | 0          | 516
```

3단계: 테스트 쿼리 실행

이번 단계에서는 RSQL에서 다중 연결을 통해 쿼리를 실행한 후 시스템 테이블을 살펴보면서 쿼리가 처리를 위해 어떻게 라우팅되었는지 확인합니다.

이번 단계를 진행하려면 RSQL 창 2개를 열어야 합니다.

- RSQL 창 1에서는 이번 자습서에서 이미 생성한 뷰를 사용하여 대기열과 쿼리의 상태를 모니터링하는 쿼리를 실행합니다.
- RSQL 창 2에서는 RSQL 창 1에서 구한 결과를 변경하는 쿼리를 장시간 실행합니다.

테스트 쿼리를 실행하려면

1. RSQL 창 2개를 엽니다. 이미 창을 1개 열었다면 두 번째 창만 열면 됩니다. 두 창을 모두 연결하는 데 동일한 사용자 계정을 사용할 수 있습니다.
2. RSQL 창 1에서는 다음 쿼리를 실행합니다.

   ```
   select * from wlm_query_state_vw;
   ```

   다음은 예 결과입니다.

   ```
   query | queue | slot_count | start_time          | state     | queue_time | exec_time
   ------+-------+------------+---------------------+-----------+------------
   +-----------
    1258 |     1 |          1 | 2014-09-24 22:21:03 | Executing | 0          | 549
   ```

   이 쿼리는 자기 참조적 결과를 반환합니다. 현재 실행 중인 쿼리는 이 보기의 SELECT 문입니다. 이 뷰에 대한 쿼리는 항상 1개 이상의 결과를 반환합니다. 이 결과를 다음 단계에서 장시간 실행 쿼리를 시작하여 나타나는 결과와 서로 비교합니다.
3. RSQL 창 2에서는 TICKIT 샘플 데이터베이스에서 쿼리를 실행합니다. 이 쿼리는 앞에서 생성한 WLM_QUEUE_STATE_VW 뷰와 WLM_QUERY_STATE_VW 뷰의 결과를 충분히 살펴볼 수 있도록 약 1분 동안 실행해야 합니다. 경우에 따라 두 뷰에 대한 쿼리를 실행할 수 있을 정도로 쿼리 실행 시간이 충분히 길지 않을 수 있습니다. 이러한 경우 `l.listid` 에 대한 필터 값을 높여서 실행 시간을 연장할 수 있습니다.

   > **Note**
   >
   > 쿼리 실행 시간을 줄이고 시스템 성능을 향상시키기 위해 Amazon Redshift는 특정 형식의 쿼리 결과를 리더 노드의 메모리에 캐시합니다. 결과 캐싱이 활성화되어 있으면 이후 쿼리의 실행 속도가 훨씬 빨라집니다. 쿼리가 빠르게 실행되는 것을 방지하려면 현재 세션에 대해 결과 캐싱을 비활성화합니다.

현재 세션에 대해 결과 캐싱을 해제하려면 다음과 같이 enable_result_cache_for_session 파라미터를 off로 설정합니다.

```
set enable_result_cache_for_session to off;
```

RSQL 창 2에서는 다음 쿼리를 실행합니다.

```
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <
 100000;
```

4. RSQL 창 1에서 WLM_QUEUE_STATE_VW 및 WLM_QUERY_STATE_VW에 대한 쿼리를 실행하여 결과를 앞선 결과와 비교합니다.

```
select * from wlm_queue_state_vw;
select * from wlm_query_state_vw;
```

다음은 예 결과입니다.

```
query | description                                | slots | mem | max_time | user_* |
 query_* | queued | executing | executed
------+--------------------------------------------+-------+-----+----------+--------
+---------+--------+-----------+----------
    0 | (super user) and (query group: superuser) |     1 | 357 |        0 |  false |
 false   |      0 |         0 |        0
    1 | (querytype:any)                           |     5 | 836 |        0 |  false |
 false   |      0 |         2 |      163

query | queue | slot_count | start_time          | state     | queue_time | exec_time
------+-------+------------+---------------------+-----------+------------
+-----------
 1267 |     1 |          1 | 2014-09-24 22:22:30 | Executing | 0          | 684
 1265 |     1 |          1 | 2014-09-24 22:22:36 | Executing | 0          | 4080859
```

이전 쿼리와 이번 단계의 결과가 다음과 같이 다른 점을 확인하십시오.

- 이제 WLM_QUERY_STATE_VW에는 행이 2개 있습니다. 결과 하나는 이 뷰에 대한 SELECT 작업을 실행하는 데 필요한 자기 참조적 쿼리입니다. 두 번째 결과는 이전 단계의 장기 실행 쿼리입니다.

- WLM_QUEUE_STATE_VW에서 executing 열이 1에서 2로 증가했습니다. 이 열의 입력 항목은 대기열에서 실행 중인 쿼리가 2개라는 것을 의미합니다.
- executed 열은 대기열에서 쿼리를 실행할 때마다 일정하게 증가합니다.

WLM_QUEUE_STATE_VW 뷰는 대기열을 비롯해 각 대기열에서 처리 중인 쿼리 수를 전체적으로 살펴보는 데 유용합니다. WLM_QUERY_STATE_VW 뷰는 현재 실행 중인 개별 쿼리를 더욱 자세히 살펴보는 데 유용합니다.

단원 2: WLM 쿼리 대기열 구성의 수정

대기열의 기본적인 동작에 대해 이해하였으므로 이제 수동 WLM을 사용하여 쿼리 대기열을 구성하는 방법에 대해 알아봅니다. 이번 단원에서는 새로운 파라미터 그룹을 생성하여 클러스터에 구성합니다. 또한 사용자 대기열을 2개 더 생성한 후 쿼리의 사용자 그룹 또는 쿼리 그룹 레이블에 따라 쿼리를 허용하도록 구성합니다. 이 두 대기열 중 하나로 라우팅되지 않는 쿼리는 실행 시간에 기본 대기열로 라우팅됩니다.

파라미터 그룹의 수동 WLM 구성을 생성하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 구성(Configurations) 및 워크로드 관리(Workload management)를 차례로 선택하여 워크로드 관리(Workload management) 페이지를 표시합니다.
3. 생성을 선택하여 파라미터 그룹 생성 창을 표시합니다.
4. Parameter group name(파라미터 그룹 이름)과 Description(설명) 모두에 대해 **`WLMTutorial`**을 입력한 다음 Create(생성)를 선택하여 파라미터 그룹을 생성합니다.

   > **Note**
   >
   > 파라미터 그룹 이름은 생성 시 모두 소문자 형식으로 변환됩니다.

5. 워크로드 관리 페이지에서 파라미터 그룹 **`wlmtutorial`**을 선택하여 파라미터 및 워크로드 관리 탭이 있는 세부 정보 페이지를 표시합니다.
6. 워크로드 관리 탭에 있는지 확인한 후 Switch WLM mode(WLM 모드 전환)를 선택하여 Concurrency settings(동시성 설정) 창을 표시합니다.
7. Manual WLM(수동 WLM)을 선택한 다음 저장을 선택하여 수동 WLM으로 전환합니다.
8. Edit workload queues(워크로드 대기열 편집)를 선택합니다.

9. 대기열 추가를 두 번 선택하여 대기열을 두 개 추가합니다. 이제 대기열 1, 대기열 2 및 기본 대기열이 있습니다.
10. 다음과 같이 각 대기열에 대한 정보를 입력합니다.

    - [대기열 1(Queue 1)]의 경우 [메모리(%)Memory (%)]에 대해 **30**, [기본의 동시성(Concurrency on main)]에 대해 **2**, [쿼리 그룹(Query groups)]에 대해 **test**를 입력합니다. 기타 설정은 기본값을 유지합니다.
    - [대기열 2(Queue 2)]의 경우 [메모리(%)Memory (%)]에 대해 **40**, [기본의 동시성(Concurrency on main)]에 대해 **3**, [사용자 그룹(User groups)]에 대해 **admin**를 입력합니다. 기타 설정은 기본값을 유지합니다.
    - 기본 대기열의 기본 값에 대한 동시성을 1보다 크거나 같은 값으로 설정합니다. 기본 대기열은 그 외에 아무것도 바꾸지 마세요. WLM은 기본 대기열에 미할당 메모리를 할당합니다.
11. 설정을 저장하려면 저장을 선택합니다.

그런 다음 수동 WLM 구성이 있는 파라미터 그룹을 클러스터와 연결합니다.

수동 WLM 구성이 있는 파라미터 그룹을 클러스터와 연결하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/redshiftv2/에서 Amazon Redshift 콘솔을 엽니다.
2. 탐색 메뉴에서 클러스터(Clusters)를 선택하고 클러스터(Clusters)를 선택하여 클러스터 목록을 표시합니다.
3. `examplecluster`와 같은 클러스터를 선택하여 클러스터의 세부 정보를 표시합니다. 그런 다음 [속성(Properties)] 탭을 선택하여 해당 클러스터의 속성을 표시합니다.
4. [데이터베이스 구성(Database configurations)] 섹션에서 [편집(Edit)], [파라미터 그룹 편집(Edit parameter group)]을 클릭하여 파라미터 그룹 창을 표시합니다.
5. [파라미터 그룹 수(Parameter groups)]에 대해 이전에 생성한 **wlmtutorial** 파라미터 그룹을 선택합니다.
6. [변경 사항 저장(Save changes)]을 선택하여 파라미터 그룹을 연결합니다.

   변경된 파라미터 그룹으로 클러스터가 수정됩니다. 그러나 데이터베이스에 변경 사항을 적용하려면 클러스터를 재부팅해야 합니다.
7. 클러스터를 선택한 [작업(Actions)]에 대해 [재부팅(Reboot)]을 선택합니다.

클러스터가 재부팅되면 상태가 사용 가능으로 돌아옵니다.

단원 3: 사용자 그룹 및 쿼리 그룹을 기반으로 쿼리를 대기열로 라우팅

이제 클러스터를 새로운 파라미터 그룹과 연결했고 WLM을 구성했습니다. 다음으로 몇 가지 쿼리를 실행하여 Amazon Redshift가 처리를 위해 쿼리를 대기열로 라우팅하는 방법에 대해 알아봅니다.

1단계: 데이터베이스에서 쿼리 대기열 구성 보기

먼저 데이터베이스에 원하는 WLM 구성이 있는지 확인합니다.

쿼리 대기열 구성을 보려면

1. RSQL을 열고 다음 쿼리를 실행합니다. 쿼리는 1단계: WLM_QUEUE_STATE_VW 뷰 생성에서 생성한 WLM_QUEUE_STATE_VW 뷰를 사용합니다. 클러스터 재부팅 이전에 세션이 이미 데이터베이스에 연결되어 있다면 다시 연결해야 합니다.

```
select * from wlm_queue_state_vw;
```

   다음은 예 결과입니다.

```
query | description                                 | slots | mem | max_time | user_* |
 query_* | queued | executing | executed
------+---------------------------------------------+-------+-----+----------+--------
+---------+--------+-----------+----------
    0 | (super user) and (query group: superuser) |     1 | 357 |        0 |  false |
 false   |      0 |         0 |        0
    1 | (query group: test)                       |     2 | 627 |        0 |  false |
 false   |      0 |         0 |        0
    2 | (suser group: admin)                      |     3 | 557 |        0 |  false |
 false   |      0 |         0 |        0
    3 | (querytype:any)                           |     5 | 250 |        0 |  false |
 false   |      0 |         1 |        0
```

   위 결과를 1단계: WLM_QUEUE_STATE_VW 뷰 생성에서 나온 결과와 비교합니다. 여기에서는 대기열 2개가 추가된 것을 알 수 있습니다. 대기열 1은 테스트 쿼리 그룹을 위한 대기열이고, 대기열 2는 관리자 사용자 그룹을 위한 대기열입니다.

   대기열 3은 이제 기본 대기열입니다. 목록에서 마지막 대기열은 항상 기본 대기열입니다. 즉 쿼리에서 사용자 그룹 또는 쿼리 그룹을 따로 지정하지 않으면 쿼리가 기본적으로 라우팅되는 대기열을 말합니다.

2. 다음 쿼리를 실행하여 쿼리가 대기열 3에서 실행되는지 확인합니다.

```
select * from wlm_query_state_vw;
```

다음은 예 결과입니다.

```
query | queue | slot_count | start_time          | state     | queue_time | exec_time
------+-------+------------+---------------------+-----------+------------
+-----------
 2144 |     3 |          1 | 2014-09-24 23:49:59 | Executing | 0          | 550430
```

2단계: 쿼리 그룹 대기열을 사용하여 쿼리 실행

쿼리 그룹 대기열을 사용하여 쿼리를 실행하려면

1. 다음 쿼리를 실행하여 `test` 쿼리 그룹으로 라우팅합니다.

```
set query_group to test;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

2. 나머지 RSQL 창에서 다음 쿼리를 실행합니다.

```
select * from wlm_query_state_vw;
```

다음은 예 결과입니다.

```
query | queue | slot_count | start_time          | state     | queue_time | exec_time
------+-------+------------+---------------------+-----------+------------
+-----------
 2168 |     1 |          1 | 2014-09-24 23:54:18 | Executing | 0          | 6343309
 2170 |     3 |          1 | 2014-09-24 23:54:24 | Executing | 0          | 847
```

쿼리가 테스트 쿼리 그룹인 대기열 1로 라우팅되었습니다.

3. 대기열 상태 뷰에서 모두 선택합니다.

```
select * from wlm_queue_state_vw;
```

다음과 같은 결과가 출력됩니다.

```
query | description                                | slots | mem | max_time | user_* |
 query_* | queued | executing | executed
------+--------------------------------------------+-------+-----+----------+--------
+---------+--------+-----------+----------
    0 | (super user) and (query group: superuser) |     1 | 357 |        0 |  false |
 false   |      0 |         0 |        0
    1 | (query group: test)                       |     2 | 627 |        0 |  false |
 false   |      0 |         1 |        0
    2 | (suser group: admin)                      |     3 | 557 |        0 |  false |
 false   |      0 |         0 |        0
    3 | (querytype:any)                           |     5 | 250 |        0 |  false |
 false   |      0 |         1 |        0
```

4. 이제 쿼리 그룹을 재설정한 후 장기(long) 쿼리를 다시 실행합니다.

```
reset query_group;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

5. 두 뷰에 대한 쿼리를 실행하여 결과를 확인합니다.

```
select * from wlm_queue_state_vw;
select * from wlm_query_state_vw;
```

다음은 예 결과입니다.

```
query | description                                | slots | mem | max_time | user_* |
 query_* | queued | executing | executed
------+--------------------------------------------+-------+-----+----------+--------
+---------+--------+-----------+----------
    0 | (super user) and (query group: superuser) |     1 | 357 |        0 |  false |
 false   |      0 |         0 |        0
    1 | (query group: test)                       |     2 | 627 |        0 |  false |
 false   |      0 |         0 |        1
    2 | (suser group: admin)                      |     3 | 557 |        0 |  false |
 false   |      0 |         0 |        0
    3 | (querytype:any)                           |     5 | 250 |        0 |  false |
 false   |      0 |         2 |        5

query | queue | slot_count | start_time          | state     | queue_time | exec_time
------+-------+------------+---------------------+-----------+------------
+-----------
```

```
 2186 |     3 |          1 | 2014-09-24 23:57:52 | Executing | 0          | 649
 2184 |     3 |          1 | 2014-09-24 23:57:48 | Executing | 0          | 4137349
```

결과는 위와 같이 쿼리가 다시 대기열 3에서 실행되고 있는 모습이어야 합니다.

3단계: 데이터베이스 사용자 및 그룹 생성

이 대기열에서 쿼리를 실행하려면 먼저 사용자 그룹을 데이터베이스에 생성한 후 사용자를 그룹에 추가해야 합니다. 그런 다음 새로운 사용자의 자격 증명을 사용해 RSQL에 로그인하여 쿼리를 실행합니다. 데이터베이스 사용자를 생성하려면 관리자와 같은 슈퍼 사용자 권한으로 쿼리를 실행해야 합니다.

새로운 데이터베이스 사용자와 사용자 그룹을 생성하려면

1. RSQL 창에서 다음 명령을 실행하여 adminwlm이라는 이름의 새로운 데이터베이스 사용자를 데이터베이스에 생성합니다.

```
create user adminwlm createuser password '123Admin';
```

2. 그러고 나서 다음 명령을 실행하여 새로운 사용자 그룹을 생성한 후 새로운 adminwlm 사용자를 그룹에 추가합니다.

```
create group admin;
alter group admin add user adminwlm;
```

4단계: 사용자 그룹 대기열을 사용하여 쿼리 실행

다음으로 쿼리를 실행하여 사용자 그룹 대기열에 라우팅합니다. 이는 실행할 쿼리 유형을 처리하도록 구성된 대기열에 쿼리를 라우팅할 때 필요합니다.

사용자 그룹 대기열을 사용하여 쿼리를 실행하려면

1. RSQL 창 2에서 다음 쿼리를 실행하여 adminwlm 계정으로 전환한 후 해당 사용자 권한으로 쿼리를 실행합니다.

```
set session authorization 'adminwlm';
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

2. RSQL 창 1에서 다음 쿼리를 실행하여 쿼리가 라우팅되는 대기열을 확인합니다.

```
select * from wlm_query_state_vw;
```

```
select * from wlm_queue_state_vw;
```

다음은 예 결과입니다.

```
query | description                                 | slots | mem | max_time | user_* |
 query_* | queued | executing | executed
------+---------------------------------------------+-------+-----+----------+--------
+---------+--------+-----------+----------
    0 | (super user) and (query group: superuser) |     1 | 357 |        0 |  false |
 false   |      0 |         0 |        0
    1 | (query group: test)                       |     2 | 627 |        0 |  false |
 false   |      0 |         0 |        1
    2 | (suser group: admin)                      |     3 | 557 |        0 |  false |
 false   |      0 |         1 |        0
    3 | (querytype:any)                           |     5 | 250 |        0 |  false |
 false   |      0 |         1 |        8

query | queue | slot_count | start_time          | state     | queue_time | exec_time
------+-------+------------+---------------------+-----------+------------
+-----------
 2202 |     2 |          1 | 2014-09-25 00:01:38 | Executing | 0          | 4885796
 2204 |     3 |          1 | 2014-09-25 00:01:43 | Executing | 0          | 650
```

이 쿼리가 실행된 대기열은 2번인 admin 사용자 대기열입니다. 이제 앞으로 이 사용자 권한으로 로그인하여 쿼리를 실행할 때마다 다른 쿼리 그룹을 지정하지 않는 한 대기열 2에서 쿼리가 실행됩니다. 선택한 대기열은 대기열 할당 규칙에 따라 달라집니다. 자세한 내용은 WLM 대기열 할당 규칙 단원을 참조하십시오.

3. 이제 RSQL 창 2에서 다음 쿼리를 실행합니다.

```
set query_group to test;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

4. RSQL 창 1에서 다음 쿼리를 실행하여 쿼리가 라우팅되는 대기열을 확인합니다.

```
select * from wlm_queue_state_vw;
select * from wlm_query_state_vw;
```

다음은 예 결과입니다.

```
query | description                                  | slots | mem | max_time | user_* |
 query_* | queued | executing | executed
------+----------------------------------------------+-------+-----+----------+--------
+---------+--------+-----------+----------
    0 | (super user) and (query group: superuser) |     1 | 357 |        0 |  false |
 false   |      0 |         0 |        0
    1 | (query group: test)                       |     2 | 627 |        0 |  false |
 false   |      0 |         1 |        1
    2 | (suser group: admin)                      |     3 | 557 |        0 |  false |
 false   |      0 |         0 |        1
    3 | (querytype:any)                           |     5 | 250 |        0 |  false |
 false   |      0 |         1 |       10

query | queue | slot_count | start_time          | state     | queue_time | exec_time
------+-------+------------+---------------------+-----------+------------
+-----------
 2218 |     1 |          1 | 2014-09-25 00:04:30 | Executing | 0          | 4819666
 2220 |     3 |          1 | 2014-09-25 00:04:35 | Executing | 0          | 685
```

5. 모두 마친 후에는 쿼리 그룹을 재설정합니다.

```
reset query_group;
```

단원 4: wlm_query_slot_count를 사용하여 대기열에서 동시성 레벨을 임시로 재정의

간혹 사용자들은 특정 쿼리에서 일시적으로 리소스가 더 필요한 경우도 있습니다. 이때는 wlm_query_slot_count 구성 설정을 사용하여 쿼리 대기열에 대한 슬롯 할당 방식을 일시적으로 재정의할 수 있습니다. 여기에서 슬롯이란 쿼리를 처리하는 데 사용되는 메모리 단위이자 CPU를 말합니다. 간혹 데이터베이스의 VACUUM 작업처럼 클러스터에서 많은 리소스가 필요한 쿼리를 실행할 때는 슬롯 수를 재정의할 수 있습니다.

간혹 특정 쿼리 유형에 맞춰 wlm_query_slot_count를 설정해야 하는 사용자가 있을 수 있습니다. 그런 경우 WLM 구성을 조정하여 해당 사용자의 쿼리 요건에 더욱 적합한 대기열을 제공할 수 있습니다. 슬롯 수를 사용하여 동시성 수준을 일시적으로 재정의하는 방법에 대한 자세한 내용은 wlm_query_slot_count 섹션을 참조하세요.

1단계: wlm_query_slot_count를 사용하여 동시성 레벨 재정의

본 자습서의 목적에 따라 이번에도 장기간 실행되는 SELECT 쿼리를 실행합니다. adminwlm 사용자 권한으로 이 쿼리를 실행하면서 wlm_query_slot_count를 사용해 쿼리에 사용할 수 있는 슬롯 수를 늘릴 것입니다.

wlm_query_slot_count를 사용하여 동시성 레벨을 재정의하려면

1. WLM_QUERY_STATE_VW 뷰에 대한 쿼리를 실행하여 결과까지 충분히 볼 수 있도록 쿼리에 대한 제한을 높입니다.

```
set wlm_query_slot_count to 3;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

2. 이제 관리자 계정을 사용하여 WLM_QUERY_STATE_VW에 대한 쿼리를 실행하고 쿼리가 어떻게 실행되고 있는지 확인합니다.

```
select * from wlm_query_state_vw;
```

   다음은 예 결과입니다.

```
query | queue | slot_count | start_time          | state     | queue_time | exec_time
------+-------+------------+---------------------+-----------+------------
+-----------
 2240 |     2 |          1 | 2014-09-25 00:08:45 | Executing | 0          | 3731414
 2242 |     3 |          1 | 2014-09-25 00:08:49 | Executing | 0          | 596
```

   쿼리의 슬롯 수는 현재 3개입니다. 이 수는 쿼리가 슬롯 3개를 모두 사용하여 쿼리를 처리하면서 대기열의 모든 리소스를 해당 쿼리에 할당하고 있다는 것을 의미합니다.

3. 이제 다음 쿼리를 실행합니다.

```
select * from WLM_QUEUE_STATE_VW;
```

   다음은 예 결과입니다.

```
query | description                                | slots | mem | max_time | user_* |
 query_* | queued | executing | executed
------+--------------------------------------------+-------+-----+----------+--------
+---------+--------+-----------+----------
```

```
    0 | (super user) and (query group: superuser) |     1 | 357 |        0 |  false |
 false   |      0 |         0 |        0
    1 | (query group: test)                       |     2 | 627 |        0 |  false |
 false   |      0 |         0 |        4
    2 | (suser group: admin)                      |     3 | 557 |        0 |  false |
 false   |      0 |         1 |        3
    3 | (querytype:any)                           |     5 | 250 |        0 |  false |
 false   |      0 |         1 |       25
```

wlm_query_slot_count 구성 설정은 현재 세션에서만 유효합니다. 이 세션이 종료되거나 다른 사용자가 쿼리를 실행할 경우에는 WLM 구성이 사용됩니다.

4. 슬롯 수를 재설정한 후 테스트를 다시 실행합니다.

```
reset wlm_query_slot_count;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

다음은 예 결과입니다.

```
query | description                                | slots | mem | max_time | user_* |
 query_* | queued | executing | executed
------+--------------------------------------------+-------+-----+----------+--------
+---------+--------+-----------+----------
    0 | (super user) and (query group: superuser) |     1 | 357 |        0 |  false |
 false   |      0 |         0 |        0
    1 | (query group: test)                       |     2 | 627 |        0 |  false |
 false   |      0 |         0 |        2
    2 | (suser group: admin)                      |     3 | 557 |        0 |  false |
 false   |      0 |         1 |        2
    3 | (querytype:any)                           |     5 | 250 |        0 |  false |
 false   |      0 |         1 |       14

query | queue | slot_count | start_time          | state     | queue_time | exec_time
------+-------+------------+---------------------+-----------+------------
+-----------
 2260 |     2 |          1 | 2014-09-25 00:12:11 | Executing | 0          | 4042618
 2262 |     3 |          1 | 2014-09-25 00:12:15 | Executing | 0          | 680
```

2단계: 다른 세션에서 쿼리 실행

이제는 다른 세션에서 쿼리를 실행합니다.

다른 세션에서 쿼리를 실행하려면

1. RSQL 창 1 및 2에서 다음과 같이 실행하여 테스트 쿼리 그룹을 사용합니다.

```
set query_group to test;
```

2. RSQL 창 1에서 다음 장기(long-running) 쿼리를 실행합니다.

```
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

3. 장기 쿼리가 여전히 RSQL 창 1에서 실행되고 있을 때 다음을 실행합니다. 이러한 명령은 대기열의 슬롯을 모두 사용할 수 있도록 슬롯 수를 높인 다음 장기 쿼리를 시작합니다.

```
set wlm_query_slot_count to 2;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

4. 세 번째 RSQL 창을 열고 뷰에 대한 쿼리를 실행하여 결과를 확인합니다.

```
select * from wlm_queue_state_vw;
select * from wlm_query_state_vw;
```

다음은 예 결과입니다.

```
query | description                                | slots | mem | max_time | user_* |
 query_* | queued | executing | executed
------+--------------------------------------------+-------+-----+----------+--------
+---------+--------+-----------+----------
    0 | (super user) and (query group: superuser) |     1 | 357 |        0 |  false |
 false   |      0 |         0 |        0
    1 | (query group: test)                       |     2 | 627 |        0 |  false |
 false   |      1 |         1 |        2
    2 | (suser group: admin)                      |     3 | 557 |        0 |  false |
 false   |      0 |         0 |        3
    3 | (querytype:any)                           |     5 | 250 |        0 |  false |
 false   |      0 |         1 |       18

query | queue | slot_count | start_time          | state         | queue_time |
 exec_time
------+-------+------------+---------------------+---------------+------------
+-----------
 2286 |     1 |          2 | 2014-09-25 00:16:48 | QueuedWaiting | 3758950    | 0
```

```
2282 |     1 |          1 | 2014-09-25 00:16:33 | Executing     | 0          |
19335850
2288 |     3 |          1 | 2014-09-25 00:16:52 | Executing     | 0          | 666
```

첫 번째 쿼리는 대기열 1에 할당된 슬롯 중 하나를 사용하여 쿼리를 실행합니다. 또한 대기열에서 대기 중인 쿼리가 하나 있습니다(여기에서 queued는 1이고 state는 QueuedWaiting임). 첫 번째 쿼리가 실행을 마치면 두 번째 쿼리가 실행을 시작합니다. 이렇게 실행되는 이유는 두 쿼리가 모두 test 쿼리 그룹으로 라우팅되었기 때문이며, 두 번째 쿼리는 처리를 시작할 정도로 충분한 수의 슬롯이 나올 때까지 대기해야 합니다.

단원 5: 리소스 정리

클러스터는 실행하는 동안 계속해서 요금이 발생합니다. 이 튜토리얼을 마치면 Amazon Redshift 시작 안내서의 추가 리소스 찾기 및 환경 재설정 단계에 따라 환경을 이전 상태로 되돌립니다.

WLM에 대한 자세한 내용은 워크로드 관리 섹션을 참조하세요.

동시성 확장

동시성 확장 기능을 사용하면 일관성 있게 빠른 쿼리 성능으로 수많은 동시 사용자 및 동시 쿼리를 지원할 수 있습니다. 동시성 크기 조정을 설정하면 Amazon Redshift가 자동으로 클러스터 용량을 추가하여 읽기 및 쓰기 쿼리의 증가를 처리합니다. 쿼리가 기본 클러스터에서 실행되든 동시성 조정 클러스터에서 실행되든 사용자에게는 최신 데이터가 보입니다.

WLM 대기열을 구성하여 동시성 조정 클러스터에 보낸 쿼리를 관리할 수 있습니다. 동시성 조정을 설정하면 대기열에서 기다리는 대신 적합한 쿼리가 동시성 조정 클러스터로 전송됩니다.

동시성 조정 클러스터가 쿼리를 활발하게 실행하는 시간에 대해서만 요금이 부과됩니다. 요금이 부과되는 방식 및 최소 요금 등 요금에 대한 자세한 내용은 동시성 크기 조정 요금을 참조하세요.

주제

- 동시성 조정 기능
- 동시성 조정에 대한 제한 사항
- 동시성 조정 지원 AWS 리전
- 동시성 확장 대상
- 동시성 확장 대기열 구성

- 동시성 확장 모니터링
- 동시성 확장 시스템 뷰

동시성 조정 기능

WLM 대기열에 대한 동시성 조정을 설정하면 대시보드 쿼리와 같은 읽기 작업에 사용할 수 있습니다. 또한 데이터 수집 및 처리를 위한 문과 같이 일반적으로 사용되는 쓰기 작업에도 사용할 수 있습니다.

쓰기 작업에 대한 동시성 조정 기능

동시성 조정은 추출, 변환 및 로드(ETL) 문과 같이 자주 사용하는 쓰기 작업을 지원합니다. 쓰기 작업에 대한 동시성 조정은 클러스터가 많은 수의 요청을 수신할 때 일관된 응답 시간을 유지하려는 경우 특히 유용합니다. 기본 클러스터의 리소스를 놓고 경합하는 쓰기 작업의 처리량을 향상시킵니다.

동시성 조정은 COPY, INSERT, DELETE, UPDATE 및 CREATE TABLE AS(CTAS) 문을 지원합니다. 또한 동시성 조정은 집계를 사용하지 않는 MV에 대한 구체화된 뷰 새로 고침을 지원합니다. 다른 데이터 조작 언어(DML) 문 및 데이터 정의 언어(DDL) 문은 지원되지 않습니다. TABLE AS가 없는 CREATE와 같이 지원되지 않는 쓰기 문이 지원되는 쓰기 문보다 먼저 명시적 트랜잭션에 포함되면 동시성 조정 클러스터에서는 쓰기 문이 아무것도 실행되지 않습니다.

동시성 조정을 위해 크레딧을 누적하면 이 크레딧 누적은 읽기 및 쓰기 작업 모두에 적용됩니다.

동시성 조정에 대한 제한 사항

다음은 Amazon Redshift 동시성 조정 사용에 대한 제한 사항입니다.

- 인터리브 정렬 키를 사용하는 테이블에 대한 쿼리를 지원하지 않습니다.
- 임시 테이블에 대한 쿼리를 지원하지 않습니다.
- 제한적인 네트워크 또는 Virtual Private Cloud(VPC) 구성으로 보호되는 외부 리소스에 액세스하는 쿼리를 지원하지 않습니다.
- Python 사용자 정의 함수(UDF) 및 람다 UDF가 포함된 쿼리를 지원하지 않습니다.
- 시스템 테이블, PostgreSQL 카탈로그 테이블 또는 비백업 테이블에 액세스하는 쿼리를 지원하지 않습니다.
- 제한적인 IAM 정책 권한이 있는 경우 외부 리소스에 액세스하는 COPY 또는 UNLOAD 쿼리는 지원하지 않습니다. 여기에는 Amazon S3 버킷 또는 DynamoDB 테이블과 같은 리소스 또는 소스에 적용되는 권한이 포함됩니다. IAM 소스에는 다음이 포함될 수 있습니다.

- `aws:sourceVpc` – 소스 VPC입니다.
- `aws:sourceVpce` - 소스 VPC 엔드포인트입니다.
- `aws:sourceIp` - 소스 IP 주소입니다.

경우에 따라 리소스에 액세스하는 COPY 및 UNLOAD 쿼리가 동시성 조정 클러스터로 전송되도록 리소스 또는 소스를 제한하는 권한을 제거해야 할 수 있습니다.

리소스 정책에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 정책 유형 및 버킷 정책을 사용하여 VPC 엔드포인트에서 액세스 제어를 참조하세요.

- 쓰기 작업에 대한 Amazon Redshift 동시성 조정은 CREATE TABLE 또는 ALTER TABLE과 같은 DDL 작업에 대해 지원되지 않습니다.
- COPY 명령에 대해 ANALYZE를 지원하지 않습니다.
- DISTSTYLE이 ALL로 설정된 대상 테이블에 대한 쓰기 작업을 지원하지 않습니다.
- 다음 파일 형식에서 COPY는 지원하지 않습니다.
  - PARQUET
  - ORC
- 자격 증명 열이 있는 테이블에 대한 쓰기 작업을 지원하지 않습니다.
- Amazon Redshift는 Amazon Redshift RA3 노드에서만 쓰기 작업에 대한 동시성 규모 조정을 지원합니다. 쓰기 작업에 대한 동시성 조정은 다른 노드 유형에서 지원되지 않습니다.

동시성 조정 지원 AWS 리전

Amazon Redshift를 사용하면 동시성 규모 조정을 사용하여 Redshift 클러스터 전반에서 동시 워크로드 수요를 관리할 수 있습니다. 이 주제에서는 Amazon Redshift로 동시성 규모 조정을 사용할 수 있는 리전에 대해 자세히 설명합니다.

다음 AWS 리전에서 동시성 크기 조정을 사용할 수 있습니다.

- 미국 동부(버지니아 북부) 리전(us-east-1)
- 미국 동부(오하이오) 리전(us-east-2)
- 미국 서부(캘리포니아 북부) 리전(us-west-1)
- 미국 서부(오레곤) 리전(us-west-2)
- 아시아 태평양(뭄바이) 리전(ap-south-1)
- 아시아 태평양(서울) 리전(ap-northeast-2)

- 아시아 태평양(싱가포르) 리전(ap-southeast-1)
- 아시아 태평양(시드니) 리전(ap-southeast-2)
- 아시아 태평양(도쿄) 리전(ap-northeast-1)
- 캐나다(중부) 리전(ca-central-1)
- 중국(베이징) 리전(cn-north-1)
- 중국(닝샤) 리전(cn-northwest-1)
- 유럽(프랑크푸르트) 리전(eu-central-1)
- 유럽(아일랜드) 리전(eu-west-1)
- 유럽(런던) 리전(eu-west-2)
- 유럽(파리) 리전(eu-west-3)
- 유럽(스톡홀름) 리전(eu-north-1)
- 유럽(취리히) 리전(eu-central-2)
- 유럽(스페인) 리전(eu-south-2)
- 남아메리카(상파울루) 리전(sa-east-1)
- AWS GovCloud(미국 동부)

동시성 확장 대상

Amazon Redshift를 사용하면 쿼리 처리를 스케일 아웃하여 동시 쿼리 실행을 가속화할 수 있습니다. 다음 주제에서는 Amazon Redshift가 동시성 규모 조정으로 라우팅할 쿼리를 결정하는 데 사용하는 기준에 대해 알아봅니다.

기본 클러스터가 다음 요구 사항을 충족하는 경우에만 쿼리가 동시성 확장 클러스터로 라우팅됩니다.

- EC2-VPC 플랫폼
- 노드 유형은 dc2.8xlarge, dc2.large, ra3.large, ra3.xlplus, ra3.4xlarge, ra3.16xlarge이어야 합니다. 쓰기 작업용 동시성 규모 조정은 Amazon Redshift RA3 노드에서만 지원됩니다.
- 노드 유형이 ra3.xlplus, ra3.4xlarge 또는 ra3.16xlarge인 클러스터의 경우 최대 32개의 컴퓨팅 노드가 허용됩니다. 또한 클러스터를 처음 생성할 때 기본 클러스터의 노드 수는 32개를 초과할 수 없습니다. 예를 들어 현재 클러스터에 20개의 노드가 있지만 원래 40개로 생성된 경우에도 동시성 확장에 대한 요구 사항을 충족하지 않습니다. 반대로 현재 DC2 클러스터에 40개의 노드가 있지만 원래 20개로 생성된 경우 동시성 확장에 대한 요구 사항을 충족합니다.
- 단일 노드 클러스터가 아닙니다.

동시성 확장 대기열 구성

Amazon Redshift를 사용하면 동시성 규모 조정을 구성하여 동시성 및 시스템 리소스를 관리할 수 있습니다. 동시성 규모 조정 대기열을 통해 동시에 실행할 수 있는 쿼리 또는 사용자 세션 수에 대해 제한을 설정하는 것이 가능합니다. 다음 섹션에서는 Amazon Redshift에서 동시성 규모 조정 대기열을 사용하여 동시 쿼리 및 사용자 세션을 효과적으로 처리할 수 있는 방법에 대한 설명을 제공합니다.

워크로드 관리자(WLM) 대기열에서 동시성 확장을 사용하도록 설정하여 쿼리를 동시성 확장 클러스터로 라우팅합니다. 대기열에 대해 동시성 조정을 설정하려면 [동시성 조정 모드(Concurrency Scaling mode)] 값을 [자동(auto)]으로 설정합니다.

동시성 확장이 활성화된 대기열로 라우팅되는 쿼리 수가 대기열의 동시성 용량을 초과하면, 용량이 수동으로 구성되었든 자동으로 결정되었든 상관없이 적격 쿼리가 동시성 확장 클러스터로 전송됩니다. 기본 클러스터에서 대기열 슬롯을 사용할 수 있게 되면 쿼리가 기본 클러스터로 라우팅되어 실행됩니다. 다른 WLM 대기열과 마찬가지로 사용자 그룹을 기준으로 또는 쿼리 그룹 레이블을 사용하여 쿼리에 레이블을 지정하거나 쿼리를 대기열에 할당에서 정의한 일치 조건에 따라 쿼리를 동시성 확장 대기열로 라우팅합니다. 또한 WLM 쿼리 모니터링 규칙을 정의하여 쿼리를 라우팅할 수 있습니다. 예를 들어 5초보다 오래 걸리는 모든 쿼리를 동시성 확장 대기열로 라우팅할 수 있습니다. 대기열 동작은 자동 WLM을 사용하는지 수동 WLM을 사용하는지에 따라 달라질 수 있다는 점에 유의하세요. 자세한 내용은 자동 WLM 구현 또는 수동 WLM 구현을 참조하세요.

동시성 확장 클러스터의 기본 수는 1입니다. 사용할 수 있는 동시성 확장 클러스터 수는 max_concurrency_scaling_clusters에서 제어합니다.

동시성 확장 모니터링

Amazon Redshift를 사용하면 동시성 규모 조정을 모니터링하고 관리하여 데이터 웨어하우징 워크로드의 성능과 비용 효율성을 최적화할 수 있습니다. 동시성 규모 조정을 통해 Amazon Redshift는 워크로드 수요가 증가할 경우 클러스터 용량을 자동으로 추가하고 수요가 줄어들 경우 해당 용량을 제거할 수 있습니다. 다음 섹션에서는 Amazon Redshift 클러스터의 동시성 규모 조정 모니터링에 대한 가이드를 제공합니다.

쿼리가 기본 클러스터에서 실행 중인지 아니면 동시성 조정 클러스터에서 실행 중인지 확인하려면 Amazon Redshift 콘솔에서 클러스터로 이동한 후 클러스터를 선택합니다. 그런 다음 쿼리 모니터링 탭과 워크로드 동시성을 선택하여 실행 중인 쿼리와 대기 중인 쿼리에 대한 정보를 확인합니다.

실행 시간을 찾으려면 `concurrency_scaling_status` 열에서 STL_QUERY 테이블 및 필터를 쿼리합니다. 다음 쿼리는 동시성 확장 클러스터에서 실행한 쿼리 및 기본 클러스터에서 실행한 쿼리에 대한 대기열 시간 및 실행 시간을 비교합니다.

```
SELECT w.service_class AS queue
, CASE WHEN q.concurrency_scaling_status = 1 THEN 'concurrency scaling cluster' ELSE
 'main cluster' END as concurrency_scaling_status
, COUNT( * ) AS queries
, SUM( q.aborted ) AS aborted
, SUM( ROUND( total_queue_time::NUMERIC / 1000000,2) ) AS queue_secs
, SUM( ROUND( total_exec_time::NUMERIC / 1000000,2) ) AS exec_secs
FROM stl_query q
JOIN stl_wlm_query w
USING (userid,query)
WHERE q.userid > 1
AND q.starttime > '2019-01-04 16:38:00'
AND q.endtime < '2019-01-04 17:40:00'
GROUP BY 1,2
ORDER BY 1,2;
```

요구 사항에 따라 `starttime` 및 `endtime` 값을 조정합니다.

동시성 확장 시스템 뷰

Amazon Redshift를 사용하면 동시성 규모 조정 시스템 뷰 기능으로 클러스터의 동시성 규모 조정 활동을 모니터링 및 관리할 수 있습니다. 다음 섹션에서는 이러한 시스템 뷰를 쿼리하고 결과를 해석하여 Amazon Redshift 환경에서 동시성 스케일 인을 효과적으로 활용하는 방법에 대해 설명합니다.

접두사가 SVCS인 시스템 뷰 집합은 기본 클러스터와 동시성 확장 클러스터 모두의 쿼리에 대한 시스템 로그 테이블의 세부 정보를 제공합니다.

다음 뷰의 정보는 해당되는 STL 뷰 또는 SVL 뷰의 정보와 유사합니다.

- SVCS_ALERT_EVENT_LOG
- SVCS_COMPILE
- SVCS_EXPLAIN
- SVCS_PLAN_INFO
- SVCS_QUERY_SUMMARY
- SVCS_STREAM_SEGS

다음 뷰는 동시성 확장에 고유한 뷰입니다.

- SVCS_CONCURRENCY_SCALING_USAGE

동시성 조정에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 다음 주제를 참조하세요.

- 동시성 확장 데이터 보기
- 쿼리 실행 중 클러스터 성능 보기
- 쿼리 세부 정보 보기

단기 쿼리 가속화

단기 쿼리 가속화(SQA)는 선택한 단기 실행 쿼리를 장기 실행 쿼리보다 우선적으로 적용합니다. SQA 쿼리가 대기열에서 장기 쿼리 뒤에서 대기해야 하지 않도록 SQA는 전용 공간에서 단기 실행 쿼리를 실행합니다. SQA는 단기 실행되고 사용자 정의 대기열에 있는 쿼리에만 우선순위를 부여합니다. SQA가 있으면 단기 실행 쿼리가 더 빠르게 실행하기 시작하며 사용자가 더 빨리 결과를 확인합니다.

SQA를 활성화하면 단기 쿼리 실행에 전용되는 워크로드 관리(WLM) 대기열을 축소할 수 있습니다. 또한 장기 실행 쿼리가 단기 실행 쿼리와 대기열의 슬롯에 대해 경합할 필요가 없으므로, 더 적은 쿼리 슬롯을 사용하도록 WLM 대기열을 구성할 수 있습니다. 더 적은 동시성을 사용하면 대부분의 워크로드에 대해 쿼리 처리량이 증가되고 전체 시스템 성능이 향상됩니다.

CREATE TABLE AS(CTAS) 문과 읽기 전용 쿼리(예: SELECT 문)는 SQA를 사용할 수 있습니다.

Amazon Redshift는 기계 학습 알고리즘으로 사용 가능한 각 쿼리를 분석하고 쿼리의 실행 시간을 예측합니다. 기본적으로 WLM은 클러스터의 워크로드를 분석한 정보에 근거하여 SQA 최대 실행 시간에 대한 값을 동적으로 지정합니다. 또는 1~20초의 고정값을 지정할 수 있습니다. 쿼리의 예측 실행 시간이 정의되거나 동적으로 할당된 SQA의 최대 런타임보다 짧고 쿼리가 WLM 대기열에서 대기 중인 경우 SQA는 WLM 대기열에서 쿼리를 분리하고 우선 실행되도록 예약합니다. 쿼리가 SQA 최대 실행 시간보다 오래 실행되면 WLM 대기열 할당 규칙에 따라 WLM은 쿼리를 첫 번째로 일치하는 WLM 대기열로 이동합니다. 시간이 지나면서 SQA가 쿼리 패턴에서 학습하면 예측이 향상합니다.

SQA는 기본 파라미터 그룹과 모든 새 파라미터 그룹에 대해 기본적으로 활성화되어 있습니다. Amazon Redshift 콘솔에서 SQA를 사용 중지하려면 파라미터 그룹에 맞게 WLM 구성을 편집하고 [단기 쿼리 가속화 사용(Enable short query acceleration)]을 선택 해제합니다. 모범 사례로, 최적의 전체 시스템 성능을 유지하려면 15개 이하의 WLM 쿼리 슬롯 수를 사용하는 것이 좋습니다. WLM 구성 수정에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 워크로드 관리 구성 섹션을 참조하세요.

단기 쿼리의 최대 실행 시간

SQA를 활성화하면 WLM이 단기 쿼리에 대한 최대 실행 시간을 '동적(dynamic)'으로 기본 설정합니다. SQA 최대 실행 시간에 대해서는 동적 설정을 유지하는 것이 좋습니다. 1~20초의 고정값을 지정하여 기본 설정을 재정의할 수 있습니다.

경우에 따라 시스템 성능을 개선하기 위해 SQA 최대 실행 시간에 다양한 값을 사용하는 방안을 고려할 수 있습니다. 이 경우 워크로드를 분석하여 대부분의 단기 실행 쿼리에 대한 최대 실행 시간을 확인하십시오. 다음 쿼리는 약 70 백분위수에서 쿼리의 최대 실행 시간을 반환합니다.

```
select least(greatest(percentile_cont(0.7)
within group (order by total_exec_time / 1000000) + 2, 2), 20)
from stl_wlm_query
where userid >= 100
and final_state = 'Completed';
```

워크로드에 효과적으로 작동하는 최대 실행 시간 값을 식별한 후에는 워크로드가 크게 변경되지 않는 한 값을 변경할 필요가 없습니다.

SQA 모니터링

SQA가 활성화되었는지 확인하려면 다음 쿼리를 실행합니다. 쿼리가 행을 반환하면 SQA가 활성화된 것입니다.

```
select * from stv_wlm_service_class_config
where service_class = 14;
```

다음 쿼리는 각 쿼리 대기열(서비스 클래스)를 통해 전송된 쿼리의 수를 보여줍니다. 또한 평균 실행 시간, 90번째 퍼센타일에 대기 시간이 있는 쿼리 수, 평균 대기 시간도 보여줍니다. SQA 쿼리는 서비스 중 클래스 14를 사용합니다.

```
select final_state, service_class, count(*), avg(total_exec_time),
percentile_cont(0.9) within group (order by total_queue_time), avg(total_queue_time)
from stl_wlm_query where userid >= 100 group by 1,2 order by 2,1;
```

SQA에서 선택되고 성공적으로 완료된 쿼리를 찾으려면 다음 쿼리를 실행합니다.

```
select a.queue_start_time, a.total_exec_time, label, trim(querytxt)
from stl_wlm_query a, stl_query b
where a.query = b.query and a.service_class = 14 and a.final_state = 'Completed'
order by b.query desc limit 5;
```

SQA에서 가져왔지만 시간이 초과된 쿼리를 확인하려면 다음 쿼리를 실행합니다.

```
select a.queue_start_time, a.total_exec_time, label, trim(querytxt)
```

```
from stl_wlm_query a, stl_query b
where a.query = b.query and a.service_class = 14 and a.final_state = 'Evicted'
order by b.query desc limit 5;
```

제거된 쿼리 및 일반적으로 쿼리에 대해 수행할 수 있는 규칙 기반 작업에 대한 자세한 내용은 WLM 쿼리 모니터링 규칙 섹션을 참조하세요.

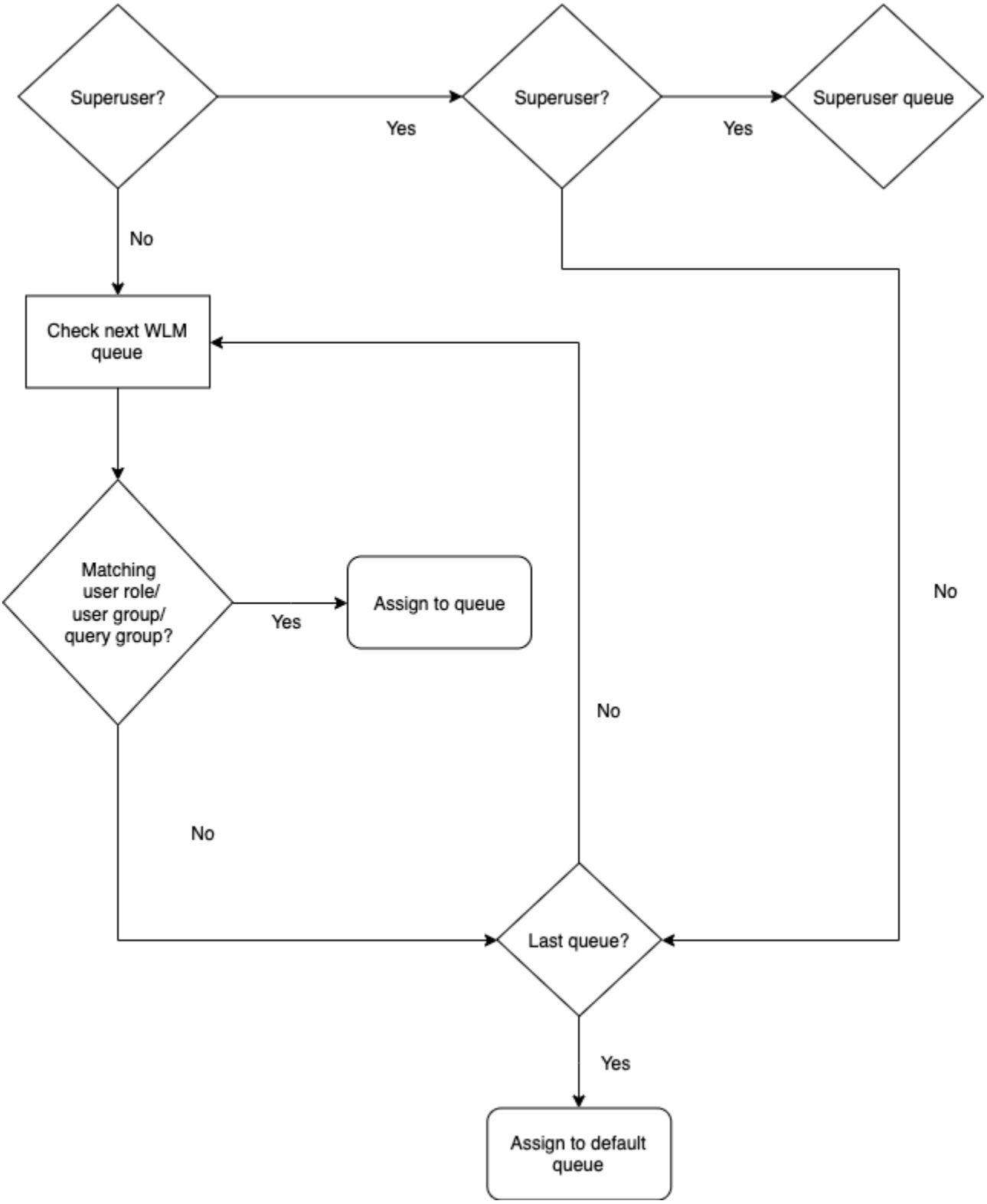
WLM 대기열 할당 규칙

Amazon Redshift를 사용하면 워크로드 관리(WLM) 구성에서 대기열 할당 규칙을 정의하여 사용자 쿼리에 대한 메모리 및 CPU 리소스 할당을 제어할 수 있습니다. 다음 섹션에서는 효율적인 리소스 할당을 달성하고 Amazon Redshift의 다양한 워크로드에 대한 서비스 수준 계약을 충족하기 위해 WLM 대기열 할당 규칙을 생성 및 관리하는 방법을 설명합니다.

사용자가 쿼리를 실행하면 아래와 같이 WLM이 WLM 대기열 할당 규칙에 따라 쿼리를 첫 번째로 일치하는 대기열에 할당합니다.

1. 사용자가 수퍼유저로 로그인하여 레이블이 수퍼유저인 쿼리 그룹에서 쿼리를 실행할 경우에는 쿼리가 수퍼유저 대기열로 할당됩니다.
2. 사용자가 역할의 일부이거나, 나열된 사용자 그룹에 속하거나, 나열된 쿼리 그룹에 속한 쿼리를 실행하는 경우에는 쿼리가 첫 번째 일치하는 대기열에 할당됩니다.
3. 쿼리가 어떠한 기준과도 일치하지 않으면 WLM 구성에서 마지막에 정의되어 있는 대기열인 기본 대기열에 할당됩니다.

다음 다이어그램은 이 규칙이 작동하는 방식을 나타낸 것입니다.

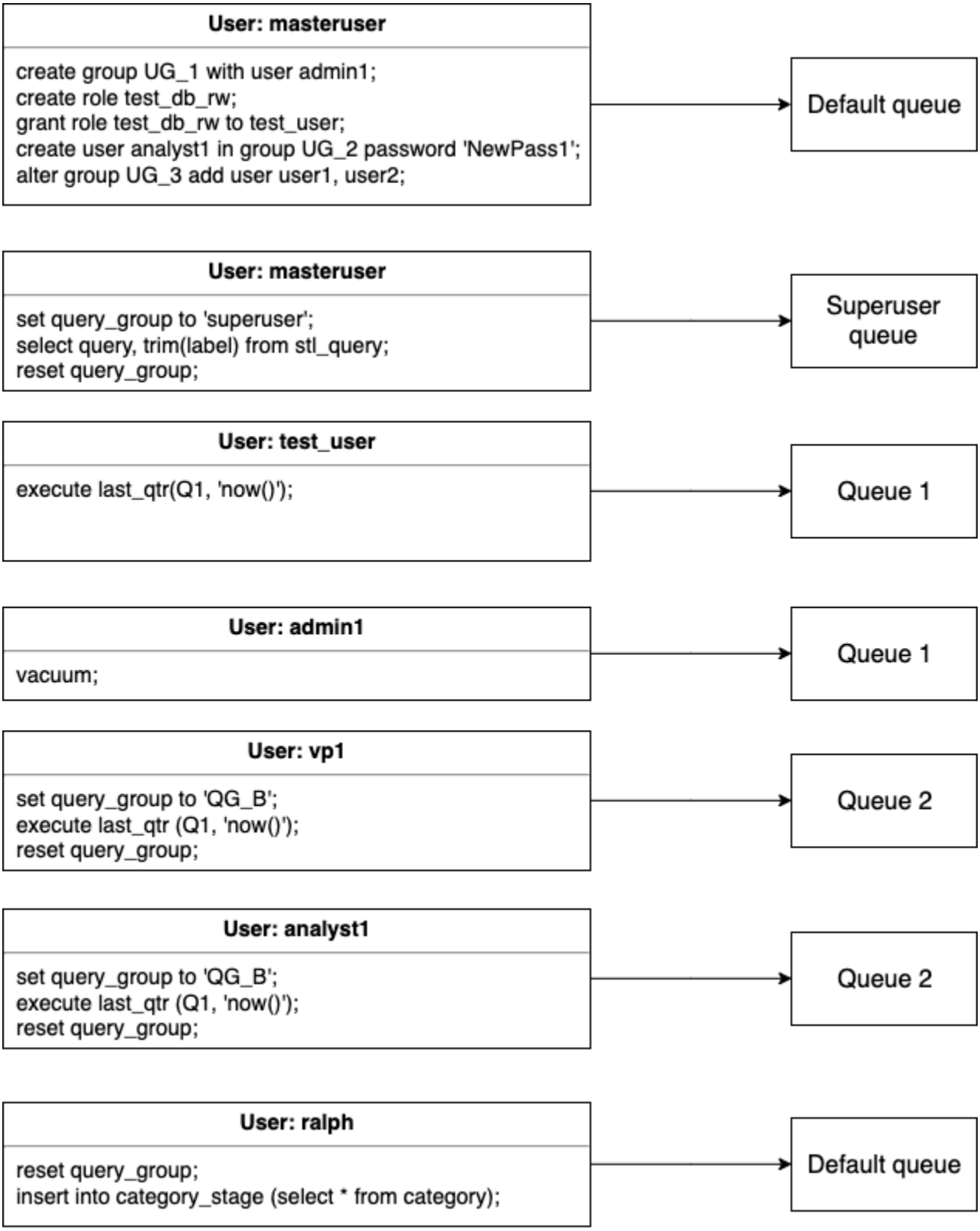


쿼리 할당 예

다음 표는 수퍼유저 대기열과 사용자 정의 대기열 4개가 포함된 WLM 구성을 나타낸 것입니다.

대기열	동시성	사용자 역할	사용자 그룹	쿼리 그룹
수퍼유저	1			수퍼유저
1	5	test_db_rw	UG_1	
2	5			QG_B
3	5		UG_2	QG_C
기본값	5			

아래 도해는 사용자 그룹 및 쿼리 그룹에 따라 쿼리가 위 표의 대기열에 할당되는 방식을 나타낸 것입니다. 런타임에 쿼리를 사용자 그룹과 쿼리 그룹에 할당하는 방법에 대한 자세한 내용은 본 섹션 후반부의 대기열에 쿼리 할당 섹션을 참조하세요.



위의 예에서 WLM의 할당 방식은 다음과 같습니다.

1. 첫 번째 문 집합은 사용자를 사용자 그룹에 할당하는 세 가지 방법을 나타냅니다. 여기에 있는 문들을 실행하는 사용자는 adminuser이며, WLM 대기열에 나열된 사용자 그룹의 멤버가 아닙니다. 또한 쿼리 그룹도 설정되어 있지 않기 때문에 문들이 모두 기본 대기열로 라우팅됩니다.
2. 사용자 adminuser가 수퍼유저이고, 쿼리 그룹이 'superuser'로 설정되어 있기 때문에 쿼리가 수퍼유저 대기열로 할당됩니다.
3. test_user 사용자가 대기열 1에 나열된 test_db_rw 역할에 할당되었으므로 쿼리가 대기열 1로 할당됩니다.
4. 사용자 admin1이 대기열 1에 나열된 사용자 그룹의 멤버이므로 쿼리가 대기열 1로 할당됩니다.
5. 사용자 vp1은 나열된 사용자 그룹의 멤버가 아닙니다. 또한 쿼리 그룹이 'QG_B'로 설정되어 있으므로 쿼리가 대기열 2로 할당됩니다.
6. 사용자 analyst1이 대기열 3에 나열된 사용자 그룹의 멤버이지만 'QG_B'가 대기열 2와 일치하므로 쿼리가 대기열 2로 할당됩니다.
7. 사용자 ralph가 나열된 사용자 그룹의 멤버가 아니고, 쿼리 그룹이 재설정되었으므로 일치하는 대기열이 없습니다. 따라서 쿼리는 기본 대기열로 할당됩니다.

대기열에 쿼리 할당

Amazon Redshift를 사용하면 워크로드 동시성을 관리하고 대기열에 할당하여 쿼리의 우선순위를 지정할 수 있습니다. 대기열을 사용하면 메모리 및 CPU 등의 리소스를 다양한 유형의 쿼리 또는 사용자에 할당하여 중요도가 덜한 쿼리에 비해 중요한 쿼리에 우선순위를 지정할 수 있습니다. 다음 섹션에서는 대기열을 생성하고, 해당 속성을 구성하고, 정의한 기준에 따라 유입되는 쿼리를 할당하는 방법을 설명합니다.

다음은 사용자 역할, 사용자 그룹 및 쿼리 그룹에 따라 쿼리를 대기열에 할당하는 예입니다.

사용자 역할을 기반으로 대기열에 쿼리 할당

사용자에게 역할이 할당되고 해당 역할이 큐에 연결되어 있는 경우 해당 사용자가 실행하는 쿼리가 해당 큐에 할당됩니다. 다음 예에서는 sales_rw라는 사용자 역할을 만들고 해당 역할에 test_user 사용자를 할당합니다.

```
create role sales_rw;
grant role sales_rw to test_user;
```

한 역할을 다른 역할에 명시적으로 부여하여 두 역할의 권한을 결합할 수도 있습니다. 사용자에게 중첩된 역할을 할당하면 사용자에게 두 역할의 권한이 모두 부여됩니다.

```
create role sales_rw;
create role sales_ro;
grant role sales_ro to role sales_rw;
grant role sales_rw to test_user;
```

클러스터에서 역할이 부여된 사용자 목록을 보려면 SVV_USER_GRANTS 테이블을 쿼리하세요. 클러스터에서 역할이 부여된 역할 목록을 보려면 SVV_ROLE_GRANTS 테이블을 쿼리하세요.

```
select * from svv_user_grants;
select * from svv_role_grants;
```

사용자 그룹을 기반으로 대기열에 쿼리 할당

사용자 그룹 이름이 대기열 정의에 나열되어 있으면 나열된 사용자 그룹의 멤버가 실행하는 쿼리는 해당 대기열에 할당됩니다. 다음은 SQL 명령(사용자 생성, create group 및 ALTER GROUP)을 사용하여 사용자 그룹을 생성한 후 사용자를 그룹에 추가하는 예입니다.

```
create group admin_group with user admin246, admin135, sec555;
create user vp1234 in group ad_hoc_group password 'vpPass1234';
alter group admin_group add user analyst44, analyst45, analyst46;
```

쿼리 그룹에 쿼리 할당

적절한 쿼리 그룹에 쿼리를 할당하여 실행 시간에 쿼리를 대기열에 할당할 수 있습니다. SET 명령을 사용하여 쿼리 그룹을 시작합니다.

```
SET query_group TO group_label
```

여기서 *group_label*은 WLM 구성에 나열된 쿼리 그룹 레이블입니다.

SET query_group 명령 이후에 실행하는 쿼리는 모두 쿼리 그룹을 재설정하거나, 혹은 현재 로그인 세션을 종료할 때까지는 지정된 쿼리 그룹의 멤버로서 실행됩니다. Amazon Redshift 객체의 설정 및 재설정에 대한 자세한 내용은 SQL 명령 참조의 SET 및 reset 섹션을 참조하세요.

지정하는 쿼리 그룹 레이블은 WLM 구성에도 포함되어야 합니다. 그렇지 않으면 SET query_group 명령을 실행해도 쿼리 대기열에 아무런 영향도 미치지 못합니다.

TO 절에서 정의하는 레이블은 쿼리 로그에도 수집되기 때문에 레이블을 문제 해결에도 사용할 수 있습니다. query_group 구성 파라미터에 대한 자세한 내용은 구성 참조에서 query_group 섹션을 참조하세요.

다음은 쿼리 그룹 'priority'에 속한 쿼리 2개를 실행한 후 쿼리 그룹을 재설정하는 예입니다.

```
set query_group to 'priority';
select count(*)from stv_blocklist;
select query, elapsed, substring from svl_qlog order by query desc limit 5;
reset query_group;
```

수퍼유저 대기열에 쿼리 할당

슈퍼 사용자 대기열에 쿼리를 할당하려면 먼저 슈퍼 사용자 권한으로 Amazon Redshift에 로그온한 후 슈퍼 사용자 그룹에서 쿼리를 실행합니다. 쿼리 실행을 모두 마치고 나면 이후 쿼리부터는 수퍼유저 대기열에서 실행되지 않도록 쿼리 그룹을 재설정해야 합니다.

다음은 실행할 명령 2개를 수퍼유저 대기열에서 할당하는 것을 보여주는 예입니다.

```
set query_group to 'superuser';

analyze;
vacuum;
reset query_group;
```

수퍼유저 목록을 보려면 PG_USER 시스템 카탈로그 테이블에 대해 쿼리합니다.

```
select * from pg_user where usesuper = 'true';
```

WLM 동적 및 정적 구성 속성

WLM 구성 속성은 동적이거나 정적입니다. 동적 속성은 클러스터 재부팅 없이도 데이터베이스에 적용할 수 있지만 정적 속성의 경우 변경 사항이 적용되려면 클러스터를 재부팅해야 합니다. 하지만 동적 속성과 정적 속성을 동시에 변경할 경우 속성 변경 사항을 모두 적용하려면 클러스터를 재부팅해야 합니다. 이는 변경된 속성이 동적이든 정적이든 상관없습니다.

동적 속성을 적용하는 동안 클러스터 상태는 `modifying`입니다. 자동 WLM과 수동 WLM 간의 전환은 정적 변경이기 때문에 적용하려면 클러스터를 재부팅해야 합니다.

다음 표에서는 자동 WLM 또는 수동 WLM 사용 시 어떤 WLM 속성이 동적이고 정적인지 알려줍니다.

WLM 속성	자동 WLM	수동 WLM
쿼리 그룹	동적	정적
쿼리 그룹 와일드카드	동적	정적
사용자 그룹	동적	정적
사용자 그룹 와일드카드	동적	정적
사용자 역할	동적	정적
사용자 역할 와일드카드	동적	정적
기본 클러스터의 동시성	해당 사항 없음	동적
동시성 확장 모드	동적	동적
단기 쿼리 가속화 활성화	해당 사항 없음	동적
단기 쿼리의 최대 실행 시간	동적	동적
사용할 메모리 비율	해당 사항 없음	동적
제한 시간	해당 사항 없음	동적
우선순위	동적	해당 사항 없음
대기열 추가 또는 제거	동적	정적

쿼리 모니터링 규칙(QMR)을 추가하거나 기존 QMR을 수정 또는 삭제하면 클러스터를 다시 시작할 필요 없이 자동으로 변경됩니다.

> **Note**
>
> 수동 WLM 사용 시 제한 시간 값이 변경되면 값이 변경된 후 실행이 시작되는 모든 쿼리에 새로운 값이 적용됩니다. 동시성 또는 사용할 메모리 비율이 변경되면 Amazon Redshift가 동적으로 새 구성으로 변경됩니다. 단, 현재 실행 중인 쿼리는 변경의 영향을 받지 않습니다. 자세한 내용은 WLM 동적 메모리 할당 섹션을 참조하세요.

주제

WLM 동적 메모리 할당

WLM은 각 대기열마다 동시성 레벨에 맞춰 다수의 쿼리 슬롯을 생성합니다. 쿼리 슬롯에 할당되는 메모리 크기는 슬롯 수를 기준으로 분할된 대기열에 할당되는 메모리 비율과 동일합니다. 메모리 할당 또는 동시성을 변경하면 Amazon Redshift가 새로운 WLM 구성으로의 전환을 동적으로 관리합니다. 따라서 활성 쿼리가 현재 할당된 메모리 양을 사용하여 완료될 때까지 실행될 수 있습니다. 동시에 Amazon Redshift는 총 메모리 사용량이 사용 가능한 메모리의 100%를 초과하지 않도록 합니다.

워크로드 관리자는 다음과 같은 프로세스에 따라 구성 전환을 관리합니다.

1. WLM이 새로운 쿼리 슬롯마다 메모리 할당을 재계산합니다.
2. 실행 중인 쿼리에서 사용하지 않는 쿼리 슬롯이 있으면 WLM이 슬롯을 제거하여 새로운 슬롯에서 해당 메모리를 사용할 수 있게 합니다.
3. 쿼리 슬롯이 현재 사용 중일 경우에는 쿼리가 끝날 때까지 WLM이 대기합니다.
4. 현재 실행 중인 쿼리가 종료되면 빈 슬롯은 제거되고 할당된 메모리도 해제됩니다.
5. 슬롯을 하나 이상 추가할 정도로 메모리가 충분하다면 새로운 슬롯이 추가됩니다.
6. 변경 시점에 실행 중이던 쿼리가 모두 종료되면 슬롯 수가 새로운 동시성 레벨과 동일해지면서 새로운 WLM 구성 전환이 완료됩니다.

실제로 변경 시 실행 중인 쿼리는 계속해서 원래 메모리 할당을 사용합니다. 변경 시 대기열에서 대기 중인 쿼리는 실행할 때가 되면 새로운 슬롯으로 라우팅됩니다.

전환 프로세스 도중 WLM 동적 속성이 변경되면 WLM이 현재 상태부터 즉시 새로운 구성 전환을 시작합니다. 전환 상태를 보려면 STV_WLM_SERVICE_CLASS_CONFIG 시스템 테이블에 대한 쿼리를 실행하십시오.

동적 WLM 예제

Amazon Redshift를 사용하면 동적 WLM(워크로드 관리)을 사용하여 Amazon Redshift 클러스터 전반에서 워크로드 배포 및 리소스 할당을 자동으로 관리할 수 있습니다. 동적 WLM은 워크로드 요구 사항에 따라 메모리 할당을 동적으로 조정하여 최적의 동시성과 성능을 제공하는 워크로드 관리(WLM) 구

성의 예제입니다. 다음 섹션에서는 Amazon Redshift 클러스터에서 동적 WLM을 구현하고 구성하는 방법에 대한 세부 정보를 제공합니다.

클러스터 WLM이 다음과 같은 동적 속성을 사용하여 2개의 대기열로 구성되어 있다고 가정하겠습니다.

대기열	동시성	사용할 메모리 비율(%)
1	4	50%
2	4	50%

이번에는 클러스터에 쿼리 처리에 사용할 수 있는 메모리가 200GB 있다고 가정하겠습니다. 이 수치는 임의적인 것으로 이해를 돕기 위해 사용되었습니다. 다음 방정식과 같이 각 슬롯에 할당되는 메모리는 25GB입니다.

```
(200 GB * 50% ) / 4 slots  = 25 GB
```

이번에는 다음과 같은 동적 속성을 사용하도록 WLM을 변경합니다.

대기열	동시성	사용할 메모리 비율(%)
1	3	75%
2	4	25%

다음 방정식과 같이 대기열 1에서는 각 슬롯마다 새롭게 할당되는 메모리는 50GB입니다.

```
(200 GB * 75% ) / 3 slots = 50 GB
```

새로운 구성을 적용할 때 쿼리 A1, A2, A3 및 A4가 실행 중이고, 쿼리 B1, B2, B3 및 B4가 대기 중이라고 가정하겠습니다. WLM이 쿼리 슬롯을 다음과 같이 동적으로 재구성합니다.

단계	실행 중인 쿼리	현재 슬롯 수	목표 슬롯 수	할당되는 메모리	사용할 수 있는 메모리
1	A1, A2, A3, A4	4	0	100GB	50GB
2	2	3	0	75GB	75GB
3	A3, A4	2	0	50GB	100GB
4	A3, A4, B1	2	1	100GB	50GB
5	5	1	1	75GB	75GB
6	6	1	2	125GB	25GB
7	7	0	2	100GB	50GB
8	8	0	3	150GB	0GB

1. WLM이 쿼리 슬롯마다 메모리 할당을 재계산합니다. 처음에 대기열 1에 할당된 메모리는 100GB였습니다. 하지만 새로운 대기열에 할당되는 총 메모리는 150GB이기 때문에 50GB를 즉시 사용할 수 있습니다. 현재 대기열 1에서 사용하는 슬롯 수는 4개이고, 새로운 동시성 레벨은 3슬롯이므로 새로운 슬롯이 추가되지는 않습니다.
2. 하나의 쿼리가 종료되면 슬롯이 제거되고 25GB가 비워집니다. 대기열 1은 이제 슬롯이 3개이며, 75GB의 메모리를 사용할 수 있습니다. 새로운 구성에서는 각 슬롯마다 50GB가 필요하지만 동시성 레벨이 3슬롯이기 때문에 슬롯이 새롭게 추가되지는 않습니다.
3. 두 번째 쿼리가 종료되면 슬롯이 제거되고 25GB가 비워집니다. 대기열 1은 이제 슬롯이 2개이며, 100GB가 사용할 수 있는 여유 메모리입니다.
4. 여유 메모리 50GB를 사용하여 새로운 슬롯이 추가됩니다. 대기열 1은 이제 슬롯이 3개이며, 50GB가 사용할 수 있는 여유 메모리입니다. 이제 대기열에서 대기 중이던 쿼리가 새로운 슬롯으로 라우팅될 수 있습니다.
5. 세 번째 쿼리가 종료되면 슬롯이 제거되고 25GB가 비워집니다. 대기열 1은 이제 슬롯이 2개이며, 75GB가 사용할 수 있는 여유 메모리입니다.
6. 여유 메모리 50GB를 사용하여 새로운 슬롯이 추가됩니다. 대기열 1은 이제 슬롯이 3개이며, 25GB가 사용할 수 있는 여유 메모리입니다. 이제 대기열에서 대기 중이던 쿼리가 새로운 슬롯으로 라우팅될 수 있습니다.

7. 네 번째 쿼리가 종료되면 슬롯이 제거되고 25GB가 비워집니다. 대기열 1은 이제 슬롯이 2개이며, 50GB가 사용할 수 있는 여유 메모리입니다.
8. 여유 메모리 50GB를 사용하여 새로운 슬롯이 추가됩니다. 대기열 1은 이제 슬롯이 3개이며 각각 50GB씩 사용할 수 있는 메모리가 모두 할당되었습니다.

전환이 완료되어 대기 중이던 쿼리가 모든 쿼리 슬롯을 사용할 수 있습니다.

WLM 쿼리 모니터링 규칙

Amazon Redshift 워크로드 관리(WLM)에서는 쿼리 모니터링 규칙에 따라 지표를 기준으로 WLM 대기열의 성능 경계를 정의한 후 쿼리가 이 경계를 벗어났을 때 필요한 작업을 지정합니다. 예를 들어 단시간 실행되는 쿼리 전용 대기열일 때는 60초 이상 실행되는 쿼리를 취소하는 규칙을 생성할 수도 있습니다. 그 밖에 잘못 설계된 쿼리를 추적할 목적으로 중첩 루프가 포함된 쿼리를 기록하는 규칙을 따로 만들 수도 있습니다.

쿼리 모니터링 규칙은 워크로드 관리(WLM) 구성 시 정의합니다. 대기열 1개에 최대 25개까지 규칙을 정의할 수 있으며 모든 대기열의 규칙 수도 25개로 제한됩니다. 각 규칙은 세 가지 조건, 즉 조건자와 한 가지 작업이 포함됩니다. 조건자는 지표와 비교 조건(=, <, >) 그리고 값으로 구성됩니다. 임의 규칙에서 모든 조건자가 충족되면 해당 규칙의 작업이 트리거됩니다. 가능한 규칙 작업으로는 아래에서도 설명하겠지만 log, hop 및 abort가 있습니다.

임의 대기열에 적용되는 규칙은 해당 대기열에서 실행되는 쿼리에만 적용됩니다. 규칙끼리는 서로 독립되어 있습니다.

WLM은 10초마다 지표를 평가합니다. Amazon Redshift는 쿼리가 자동으로 다시 작성될 때 하위 쿼리 수준에서 쿼리 모니터링 규칙을 적용합니다. 같은 기간 동안 2개 이상의 규칙이 트리거된 경우 WLM은 가장 심각한 작업이 연결되어 있는 규칙을 선택합니다. 두 규칙에 대한 작업의 심각도가 동일한 경우 WLM은 규칙 이름을 기준으로 규칙을 알파벳 순서로 실행합니다. 작업이 건너뛰기나 중단인 경우에는 작업을 기록한 후 쿼리가 대기열에서 제거됩니다. 기록 작업인 경우 쿼리가 대기열에서 계속 실행됩니다. WLM은 규칙에 따라 쿼리 1개마다 시작할 수 있는 기록 작업이 1개로 제한됩니다. 대기열에 다른 규칙이 포함되어 있으면 다른 규칙들도 계속해서 적용됩니다. 작업이 건너뛰기이고 쿼리가 다른 대기열로 라우팅된다면 새로운 대기열의 규칙이 적용됩니다. 쿼리 모니터링 및 특정 쿼리에 대해 수행된 작업 추적에 대한 자세한 내용은 단기 쿼리 가속화의 샘플 컬렉션을 참조하세요.

규칙의 조건자가 모두 충족되면 WLM이 한 행을 STL_WLM_RULE_ACTION 시스템 테이블에 작성합니다. 또한 Amazon Redshift가 현재 실행 중인 쿼리의 지표를 STV_QUERY_METRICS에 기록합니다. 완료된 쿼리의 지표는 STL_QUERY_METRICS에 저장됩니다.

쿼리 모니터링 규칙 정의

쿼리 모니터링 규칙은 WLM 구성, 즉 클러스터의 파라미터 그룹을 정의할 때 함께 정의합니다.

AWS Management Console을 사용하거나 프로그래밍 방식으로 JSON을 사용하여 규칙을 생성할 수 있습니다.

> **Note**
>
> 프로그래밍 방식으로 규칙을 생성하려면 콘솔을 사용하여 파라미터 그룹 정의에 추가할 JSON을 생성하는 것이 가장 좋습니다. 자세한 내용은 Amazon Redshift 관리 설명서의 콘솔을 사용하여 쿼리 모니터링 규칙 생성 또는 수정 및 AWS CLI를 사용하여 파라미터 값 구성을 참조하세요.

쿼리 모니터링 규칙을 정의하려면 다음 요소를 지정합니다.

- 규칙 이름 - 규칙 이름은 WLM 구성 내에서 고유해야 합니다. 최대 32자의 영숫자 또는 밑줄로 구성되며, 공백이나 인용 부호는 포함될 수 없습니다. 대기열 1개당 규칙 수는 최대 25개까지 가능하며 모든 대기열의 총 규칙 수도 25개로 제한됩니다.
- 1개 이상의 조건자 - 규칙 1개당 최대 3개의 조건자를 가질 수 있습니다. 임의 규칙에서 모든 조건자가 충족되면 연결되어 있는 작업이 트리거됩니다. 조건자는 지표 이름과 연산자(=, <, >) 그리고 값으로 정의됩니다. 예를 들면, `query_cpu_time > 100000`입니다. 지표 목록과 지표에 따른 값의 예는 이번 단원에서 아래 프로비저닝된 Amazon Redshift에 대한 쿼리 모니터링 지표를 참조하십시오.
- 작업 - 다수의 규칙이 트리거되면 WLM이 가장 심각한 작업이 연결되어 있는 규칙을 선택합니다. 가능한 작업은 심각도의 오름차순에 따라 다음과 같습니다.
  - 로그 - 쿼리에 대한 정보를 STL_WLM_RULE_ACTION 시스템 테이블에 기록합니다. 기록하기 작업은 로그 레코드만 작성하려고 할 때 사용합니다. WLM은 규칙에 따라 쿼리 1개마다 기록하기 작업이 1개로 제한되어 있습니다. 기록하기 작업을 마치면 다른 규칙이 적용되고 WLM이 계속해서 쿼리를 모니터링합니다.
  - 건너뛰기(수동 WLM에서만 사용할 수 있음) - 작업을 기록하고 쿼리를 건너뛰어 일치하는 그다음 대기열로 이동합니다. 일치하는 대기열이 더 이상 없으면 쿼리가 취소됩니다. QMR은 CREATE TABLE AS(CTAS) 문과 읽기 전용 쿼리(예: SELECT 문)만 건너뜁니다. 자세한 내용은 WLM 쿼리 대기열 건너뛰기 단원을 참조하십시오.
  - 중단 - 작업을 로그하고 쿼리를 취소합니다. QMR은 COPY 문과 유지 관리 작업(예: ANALYZE 및 VACUUM)을 중지하지 않습니다.

- 우선 순위 변경(자동 WLM에서만 사용할 수 있음) - 쿼리의 우선 순위를 변경합니다.

쿼리 실행 시간을 제한하려면 WLM 제한 시간 대신 쿼리 모니터링 규칙을 생성하는 것이 좋습니다. 예를 들어 다음 JSON 코드 조각과 같이 `max_execution_time`을 50,000밀리초로 설정할 수 있습니다.

```
"max_execution_time": 50000
```

하지만 대신 동등한 쿼리 모니터링 규칙을 정의하는 것이 좋습니다. 다음 예제에서는 `query_execution_time`을 50초로 설정하는 쿼리 모니터링 규칙을 보여 줍니다.

```
"rules":
[
    {
        "rule_name": "rule_query_execution",
        "predicate": [
            {
                "metric_name": "query_execution_time",
                "operator": ">",
                "value": 50
            }
        ],
        "action": "abort"
    }
]
```

쿼리 모니터링 규칙을 생성하거나 수정하는 단계는 Amazon Redshift 관리 설명서의 콘솔을 사용하여 쿼리 모니터링 규칙 생성 또는 수정 및 wlm_json_configuration 파라미터의 속성을 참조하세요.

쿼리 모니터링 규칙에 대한 자세한 내용은 다음 주제에서 찾아볼 수 있습니다.

- 프로비저닝된 Amazon Redshift에 대한 쿼리 모니터링 지표
- 쿼리 모니터링 규칙 템플릿
- 콘솔을 사용한 규칙 생성
- 워크로드 관리 구성
- 쿼리 모니터링 규칙에 대한 시스템 테이블 및 뷰

프로비저닝된 Amazon Redshift에 대한 쿼리 모니터링 지표

다음 표는 쿼리 모니터링 규칙에서 사용하는 지표를 설명한 것입니다. 아래 지표는 STV_QUERY_METRICS 및 STL_QUERY_METRICS 시스템 테이블에 저장되는 지표와 다릅니다.

임의 지표의 성능 임계값은 쿼리 수준에서 또는 세그먼트 수준에서 추적됩니다. 세그먼트 및 단계에 대한 자세한 내용은 쿼리 계획 및 실행 워크플로우 섹션을 참조하세요.

> **Note**
>
> WLM 제한 시간 파라미터는 쿼리 모니터링 규칙과 다릅니다.

지표	명칭	설명
쿼리 CPU 시간	query_cpu_time	쿼리에 사용된 CPU 시간(초)입니다. CPU time은 Query execution time가 다릅니다. 유효한 값은 0~999,999입니다.
읽은 블록	query_blocks_read	쿼리가 읽은 1MB 데이터 블록의 수입니다. 유효한 값은 0~1,048,575입니다.
스캔하는 행의 수	scan_row_count	스캔 단계에 포함되는 행의 수입니다. 행 개수는 삭제 대기 행(고스트 행)을 필터링하고 사용자 정의 쿼리 필터를 적용하기 전에 내보낸 행의 총 수입니다. 유효한 값은 0~999,999,999,999,999입니다.
쿼리 실행 시간	query_execution_time	쿼리를 실행하고 경과된 시간(초)입니다. 실행 시간에는 대기열에서 대기하는 데 소모한 시간은 포함되지 않습니다. 유효한 값은 0~86,399입니다.
쿼리 대기열 시간	query_queue_time	대기열에서 기다리는 데 소요된 시간(초)입니다.

지표	명칭	설명
		유효한 값은 0~86,399입니다.
CPU 사용량	query_cpu_usage_percent	쿼리에 사용된 CPU 용량의 비율입니다. 유효한 값은 0~6,399입니다.
디스크 메모리	query_temp_blocks_to_disk	중간 결과를 작성하는 데 사용되는 임시 디스크 공간(1MB 블록)입니다. 유효한 값은 0~319,815,679입니다.
CPU 스큐	cpu_skew	임의 조각의 최대 CPU 사용량과 모든 조각의 평균 CPU 사용량을 비교한 비율입니다. 이 지표는 세그먼트 수준에서 정의됩니다. 유효한 값은 0~99입니다.
I/O 스큐	io_skew	임의 조각에서 읽은 최대 블록 수(I/O)와 모든 조각에서 읽은 평균 블록 수를 비교한 비율입니다. 이 지표는 세그먼트 수준에서 정의됩니다. 유효한 값은 0~99입니다.
조인된 행	join_row_count	조인 단계에서 처리한 행의 수입니다. 유효한 값은 0~999,999,999,999,999입니다.
중첩 루프 조인 행의 수	nested_loop_join_row_count	중첩 루프 조인에 포함된 행의 수입니다. 유효한 값은 0~999,999,999,999,999입니다.
반환 행의 수	return_row_count	쿼리에서 반환되는 행의 수입니다. 유효한 값은 0~999,999,999,999,999입니다.

지표	명칭	설명
세그먼트 실행 시간	segment_execution_time	단일 세그먼트를 실행하고 경과된 시간(초)입니다. 샘플링 오류를 피하거나 줄이려면 규칙에 segment_execution_time > 10 을 추가하십시오. 유효한 값은 0~86,388입니다.
Spectrum 스캔 행 개수(행	spectrum_scan_row_count	Amazon Redshift Spectrum 쿼리에서 스캔한 Amazon S3의 데이터 행 수입니다. 유효한 값은 0~999,999,999,999,999입니다.
Spectrum 스캔 크기	spectrum_scan_size_mb	Amazon Redshift Spectrum 쿼리에서 스캔한 Amazon S3의 데이터 크기(MB)입니다. 유효한 값은 0~999,999,999,999,999입니다.
쿼리 우선 순위	query_priority	쿼리의 우선순위입니다. 유효한 값은 HIGHEST, HIGH, NORMAL, LOW 및 LOWEST입니다. 보다 큼(>) 및 보다 작음(<) 연산자를 사용해 query_priority 를 비교하는 경우 HIGHEST는 HIGH보다 크고 HIGH는 NORMAL보다 큽니다.

Note

- 건너뛰기 작업은 query_queue_time 조건자에서 지원되지 않습니다. 즉, query_queue_time 조건자가 충족될 때 건너뛰도록 정의된 규칙은 무시됩니다.
- 세그먼트 실행 시간이 짧으면 io_skew, query_cpu_usage_percent 같은 일부 지표에서 샘플링 오류가 발생할 수 있습니다. 이러한 샘플링 오류를 피하거나 줄이려면 규칙에 세그먼트 실행 시간을 추가하십시오. 처음에는 segment_execution_time > 10으로 시작하는 것이 좋습니다.

SVL_QUERY_METRICS 뷰에는 완료된 쿼리의 지표가 표시됩니다. 그리고 SVL_QUERY_METRICS_SUMMARY 뷰에는 완료된 쿼리의 최대 지표 값이 표시됩니다. 쿼리 모니터링 규칙을 정의하기 위한 임계값을 결정할 때는 이 두 가지 뷰의 값들을 사용하십시오.

Amazon Redshift Serverless에 대한 쿼리 모니터링 지표

다음 테이블은 Amazon Redshift Serverless의 쿼리 모니터링 규칙에서 사용하는 지표를 설명한 것입니다.

지표	명칭	설명
쿼리 CPU 시간	`max_query_cpu_time`	쿼리에 사용된 CPU 시간(초)입니다. `CPU time`은 `Query execution time`가 다릅니다. 유효한 값은 0~999,999입니다.
읽은 블록	`max_query_blocks_read`	쿼리가 읽은 1MB 데이터 블록의 수입니다. 유효한 값은 0~1,048,575입니다.
스캔하는 행의 수	`max_scan_row_count`	스캔 단계에 포함되는 행의 수입니다. 행 개수는 삭제 대기 행(고스트 행)을 필터링하고 사용자 정의 쿼리 필터를 적용하기 전에 내보낸 행의 총 수입니다. 유효한 값은 0~999,999,999,999,999입니다.
쿼리 실행 시간	max_query_execution_time	쿼리를 실행하고 경과된 시간(초)입니다. 실행 시간에는 대기열에서 대기하는 데 소모한 시간은 포함되지 않습니다. 쿼리가 설정된 실행 시간을 초과하면 Amazon Redshift Serverless가 쿼리를 중단합니다. 유효한 값은 0~86,399입니다.
쿼리 대기열 시간	`max_query_queue_time`	대기열에서 기다리는 데 소요된 시간(초)입니다. 유효한 값은 0~86,399입니다.

지표	명칭	설명
CPU 사용량	max_query_cpu_usage_percent	쿼리에 사용된 CPU 용량의 비율입니다. 유효한 값은 0~6,399입니다.
디스크 메모리	max_query_temp_blocks_to_disk	중간 결과를 작성하는 데 사용되는 임시 디스크 공간(1MB 블록)입니다. 유효한 값은 0~319,815,679입니다.
조인된 행	max_join_row_count	조인 단계에서 처리한 행의 수입니다. 유효한 값은 0~999,999,999,999,999입니다.
중첩 루프 조인 행의 수	max_nested_loop_join_row_count	중첩 루프 조인에 포함된 행의 수입니다. 유효한 값은 0~999,999,999,999,999입니다.

> **Note**
>
> - 건너뛰기 작업은 max_query_queue_time 조건자에서 지원되지 않습니다. 즉, max_query_queue_time 조건자가 충족될 때 건너뛰도록 정의된 규칙은 무시됩니다.
> - 세그먼트 실행 시간이 짧으면 max_io_skew, max_query_cpu_usage_percent 같은 일부 지표에서 샘플링 오류가 발생할 수 있습니다.

쿼리 모니터링 규칙 템플릿

Amazon Redshift 콘솔을 사용하여 규칙을 추가할 때 사전 정의된 템플릿에서 규칙을 생성하도록 선택할 수 있습니다. Amazon Redshift는 일련의 조건자로 새 규칙을 생성하고 조건자를 기본값으로 채웁니다. 이때 기본 작업은 기록하기입니다. 조건자와 작업은 사용 사례에 따라 수정할 수 있습니다.

다음 표는 사용할 수 있는 템플릿을 모아놓은 목록입니다.

템플릿 이름	Predicates	설명
중첩 루프 조인	`nested_loop_join_row_count > 100`	중첩 루프 조인은 조인 조건자가 불완전하여 종종 대용량 반환 집합(데카르트 곱)을 야기한다는 의미가 될 수도 있습니다. 이때는 행의 수를 낮춰서 잠재적 런어웨이 쿼리를 조기에 발견하십시오.
쿼리는 많은 수의 행을 반환함	`return_row_count > 1000000`	대기열을 단시간 실행되는 단순 쿼리에 지정할 경우에는 높은 행의 수를 반환하는 쿼리를 찾아내는 규칙도 추가할 수 있습니다. 이 템플릿에서 기본적으로 사용하는 행의 수는 100만 개입니다. 시스템에 따라 100만 개의 행이 높다고 생각할 수도 있지만 대용량 시스템에서는 행의 수가 10억 개 이상은 되어야 높다고 할 수 있습니다.
많은 수의 행을 통해 조인	`join_row_count > 1000000000`	조인 단계에서 행의 수가 비정상적으로 높아지는 경우에는 더욱 제한적인 필터의 필요성을 나타낼 수도 있습니다. 이 템플릿에서 기본적으로 사용하는 행의 수는 10억 개입니다. 빠르고 간단한 쿼리를 위한 임시(일회성) 대기열의 경우 더 낮은 숫자를 사용할 수 있습니다.
중간 결과를 작성할 때의 높은 디스크 사용량	`query_temp_blocks_to_disk > 100000`	현재 실행 중인 쿼리가 가능한 시스템 RAM보다 더 많은 메모리를 사용할 경우에는 쿼리 실행 엔진이 중간 결과를 디스크에 작성합니다(메모리 가득 참). 일반적으로 이러한 상황은 대부분 디스크 공간을 사용하는 악의적인 쿼리로 인해 발생합니다. 디스크 사용량의 허용 임계값은 클러스터 노드 유형과 노드 수에 따라 다릅니다. 이 템플릿에서 기본적으로 사용하는 블록의 수는 10만 개, 즉 100GB입니다. 클러스터가 작은 규모일수록 블록의 수를 낮추는 것이 좋습니다.
I/O 스큐(skew)가 높은 장기 실행 쿼리	`segment_execution_time`	노드 조각 1개가 나머지 전체 조각보다 I/O 속도가 훨씬 높을 때는 I/O 스큐가 발생합니다. 처음부터 스큐가 1.30(평균의 1.3배)이면 높다고 생

템플릿 이름	Predicates	설명
	`> 120` 및 `io_skew > 1.30`	각할 수 있습니다. I/O 스큐가 높다고 해서 항상 문제가 되는 것은 아니지만 동시에 쿼리 실행이 장시간 길어지면 분산 스타일이나 정렬 키에 문제가 있는 것일 수도 있습니다.

쿼리 모니터링 규칙에 대한 시스템 테이블 및 뷰

규칙의 조건자가 모두 충족되면 WLM이 한 행을 STL_WLM_RULE_ACTION 시스템 테이블에 작성합니다. 이 행에는 규칙을 트리거한 쿼리와 결과 작업에 대한 세부 정보가 포함되어 있습니다.

또한 Amazon Redshift는 쿼리 지표를 다음 시스템 테이블 및 뷰에 기록합니다.

- STV_QUERY_METRICS 테이블에는 현재 실행 중인 쿼리의 지표가 표시됩니다.
- STL_QUERY_METRICS 테이블에는 완료된 쿼리의 지표가 기록됩니다.
- SVL_QUERY_METRICS 뷰에는 완료된 쿼리의 지표가 표시됩니다.
- 그리고 SVL_QUERY_METRICS_SUMMARY 뷰에는 완료된 쿼리의 최대 지표 값이 표시됩니다.

WLM 시스템 테이블 및 뷰

WLM은 내부적으로 정의된 WLM 서비스 클래스에 따라 쿼리 대기열을 구성합니다. Amazon Redshift는 WLM 구성에 정의된 대기열과 함께 이러한 서비스 클래스에 따라 여러 내부 대기열을 생성합니다. 대기열과 서비스 클래스라는 용어는 시스템 테이블에서 서로 통용되기도 합니다. 수퍼유저 대기열은 서비스 클래스 5를 사용합니다. 사용자 정의 대기열은 서비스 클래스 6 이상을 사용합니다.

쿼리, 대기열 및 서비스 클래스의 상태는 WLM 시스템 테이블을 사용하여 확인할 수 있습니다. 다음은 쿼리를 통해서 아래와 같은 작업이 가능한 시스템 테이블입니다.

- 추적 중인 쿼리와 워크로드 관리자에서 할당하는 리소스를 확인합니다.
- 쿼리가 할당된 대기열을 확인합니다.
- 현재 워크로드 관리자에서 추적 중인 쿼리의 상태를 확인합니다.

테이블 이름	설명
STL_WLM_ERROR	WLM 관련 오류 이벤트에 대한 로그를 저장합니다.
STL_WLM_QUERY	WLM에서 추적 중인 쿼리를 나열합니다.
STV_WLM_CLASSIFICATION_CONFIG	현재 WLM 분류 규칙을 표시합니다.
STV_WLM_QUERY_QUEUE_STATE	쿼리 대기열의 현재 상태를 기록합니다.
STV_WLM_QUERY_STATE	WLM에서 추적 중인 쿼리의 현재 상태를 스냅샷으로 제공합니다.
STV_WLM_QUERY_TASK_STATE	쿼리 작업의 현재 상태를 저장합니다.
STV_WLM_SERVICE_CLASS_CONFIG	WLM의 서비스 클래스 구성을 기록합니다.
STV_WLM_SERVICE_CLASS_STATE	서비스 클래스의 현재 상태를 저장합니다.
STL_WLM_RULE_ACTION	사용자 정의 대기열과 연결된 WLM 쿼리 모니터링 규칙에서 발생하는 작업 세부 정보를 기록합니다.
STV_WLM_QMR_CONFIG	WLM 쿼리 모니터링 규칙(QMR)의 구성을 기록합니다.

시스템 테이블에서 쿼리를 추적하려면 작업 ID를 사용하면 됩니다. 다음은 가장 최근에 제출된 사용자 쿼리의 작업 ID를 가져오는 예입니다.

```
select task from stl_wlm_query where exec_start_time =(select max(exec_start_time) from
 stl_wlm_query);

task
------
137
(1 row)
```

다음은 현재 실행 중이거나 여러 서비스 클래스(대기열)에서 대기 중인 쿼리를 표시하는 예입니다. 다음 쿼리는 Amazon Redshift에서 동시에 실행 중인 전체 워크로드를 추적하는 데 유용합니다.

```
select * from stv_wlm_query_state order by query;


xid |task|query|service_| wlm_start_  |  state  |queue_ | exec_
    |    |     |class   | time        |         |time   | time
----+----+-----+--------+-------------+---------+-------+--------
2645| 84 | 98  | 3      | 2010-10-... |Returning|   0   | 3438369
2650| 85 | 100 | 3      | 2010-10-... |Waiting  |   0   | 1645879
2660| 87 | 101 | 2      | 2010-10-... |Executing|   0   | 916046
2661| 88 | 102 | 1      | 2010-10-... |Executing|   0   | 13291
(4 rows)
```

WLM 서비스 클래스 ID

다음 표에 서비스 클래스에 할당된 ID가 나와 있습니다.

ID	서비스 클래스
1~4	시스템에서 사용하도록 예약됩니다.
5	수퍼유저 대기열에서 사용합니다.
6~13	WLM 구성에 정의된 수동 WLM 대기열에서 사용합니다.
14	단기 쿼리 가속화에서 사용합니다.
15	Amazon Redshift에서 실행하는 유지 관리 작업용으로 예약됩니다.
100~107	auto_wlm이 true인 경우 자동 WLM 대기열에서 사용합니다.

데이터베이스 보안

어느 사용자가 어느 데이터베이스 객체에 액세스할 수 있는지 제어하여 데이터베이스 보안을 관리합니다. 사용자에게 역할이나 그룹을 할당할 수 있으며 사용자, 역할 또는 그룹에 부여하는 권한에 따라 액세스할 수 있는 데이터베이스 객체가 결정됩니다.

주제

데이터베이스 객체에 대한 액세스는 사용자 또는 역할에 부여하는 권한에 따라 달라집니다. 다음 지침은 데이터베이스 보안의 작동 원리를 요약합니다.

- 기본적으로 객체 소유자에게만 권한이 부여됩니다.
- Amazon Redshift 데이터베이스 사용자는 데이터베이스에 연결할 수 있는 명명된 사용자입니다. 사용자는 두 가지 방법으로 권한이 부여됩니다. 권한을 계정에 직접 할당하는 명시적인 방법과 권한이 부여되는 그룹의 구성원으로 만드는 묵시적인 방법입니다.
- 그룹은 보안 유지 관리를 간소화하기 위해 집단적으로 권한이 할당될 수 있는 사용자의 모음입니다.
- 스키마는 데이터베이스 테이블과 그 밖의 데이터베이스 객체의 모음입니다. 스키마는 중첩될 수 없다는 점만 빼면 파일 시스템 디렉터리와 비슷합니다. 사용자에게 단일 스키마 또는 여러 스키마에 대한 액세스 권한을 부여할 수 있습니다.

또한 Amazon Redshift는 다음 기능을 사용하여 어떤 사용자가 어떤 데이터베이스 객체에 액세스할 수 있는지를 보다 세밀하게 제어할 수 있습니다.

- 역할 기반 액세스 제어(RBAC)를 사용하면 역할에 권한을 할당한 다음 사용자에게 적용할 수 있으므로 대규모 사용자 그룹에 대한 권한을 제어할 수 있습니다. 그룹과 달리 역할은 다른 역할의 권한을 상속할 수 있습니다.

  행 수준 보안(RLS)을 사용하면 선택한 행에 대한 액세스를 제한하는 정책을 정의한 다음 해당 정책을 사용자 또는 그룹에 적용할 수 있습니다.

  동적 데이터 마스킹(DDM)은 쿼리 런타임에 데이터를 변환하여 사용자가 민감한 세부 정보를 노출하지 않고 데이터에 액세스할 수 있도록 하여 데이터를 추가로 보호합니다.

보안 구현의 예는 사용자 및 그룹 액세스 제어 예 섹션을 참조하세요.

데이터 보호에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 Amazon Redshift의 보안 섹션을 참조하세요.

Amazon Redshift 보안 개요

Amazon Redshift 데이터베이스 보안은 다른 Amazon Redshift 보안 유형과 구분됩니다. Amazon Redshift는 이 섹션에서 설명하는 데이터베이스 보안 외에 다음과 같은 보안 관리 기능을 제공합니다.

- 로그인 자격 증명 - Amazon Redshift AWS 관리 콘솔에 대한 액세스는 AWS 계정 권한에 의해 제어됩니다. 자세한 내용은 로그인 자격 증명을 참조하세요.
- 액세스 관리 - 특정 Amazon Redshift 리소스에 대한 액세스를 제어하려면 AWS Identity and Access Management(IAM) 계정을 정의합니다. 자세한 내용은 Amazon Redshift 리소스에 대한 액세스 제어 섹션을 참조하세요.
- 클러스터 보안 그룹 - 다른 사용자에게 Amazon Redshift 클러스터에 대한 인바운드 액세스 권한을 부여하려면 클러스터 보안 그룹을 정의하고 클러스터에 연결합니다. 자세한 내용은 Amazon Redshift 클러스터 보안 그룹 섹션을 참조하세요.
- VPC - 가상 네트워킹 환경을 사용하여 클러스터에 대한 액세스를 보호하기 위해 Amazon Virtual Private Cloud(VPC)에서 클러스터를 시작할 수 있습니다. 자세한 내용은 Virtual Private Cloud(VPC)에서 클러스터 관리를 참조하세요.
- 클러스터 암호화 - 모든 사용자 생성 테이블에서 데이터를 암호화하기 위해 클러스터를 시작할 때 클러스터 암호화를 활성화할 수 있습니다. 자세한 내용은 Amazon Redshift 클러스터 섹션을 참조하세요.
- SSL 연결 - SQL 클라이언트와 클러스터 간 연결을 암호화하기 위해 보안 소켓 계층(SSL) 암호화를 사용할 수 있습니다. 자세한 내용은 SSL을 사용하여 클러스터에 연결을 참조하세요.

- 로드 데이터 암호화 - Amazon S3에 업로드할 때 테이블 로드 데이터를 암호화하기 위해 서버 측 암호화 또는 클라이언트 측 암호화를 사용할 수 있습니다. 서버 측 암호화된 데이터에서 로드할 때는 Amazon S3가 복호화를 투명하게 처리합니다. 클라이언트 측 암호화된 데이터에서 로드할 때는 Amazon Redshift COPY 명령이 테이블을 로드하면서 데이터를 복호화합니다. 자세한 내용은 Amazon S3에 암호화된 데이터 업로드 섹션을 참조하세요.
- 전송 중인 데이터 - AWS 클라우드 내에서 전송 중인 데이터를 보호하기 위해 Amazon Redshift는 COPY, UNLOAD, 백업 및 복원 작업을 위해 하드웨어 가속 SSL을 사용하여 Amazon S3 또는 Amazon DynamoDB와 통신합니다.
- 열 수준 액세스 제어 - Amazon Redshift의 데이터에 대해 열 수준 액세스 제어를 사용하려면 보기 기반 액세스 제어를 구현하거나 다른 시스템을 사용할 필요 없이 열 수준 권한 부여 및 취소 문을 사용합니다.
- 행 수준 보안 제어 - Amazon Redshift의 데이터에 대한 행 수준 보안 제어를 수행하려면 정책에 정의된 행에 대한 액세스를 제한하는 정책을 생성하여 역할 또는 사용자에게 연결합니다.

기본 데이터베이스 사용자 권한

데이터베이스 객체를 생성하면 해당 객체의 소유자가 됩니다. 기본적으로 슈퍼 사용자 또는 객체의 소유자만 객체를 쿼리, 수정하거나 객체에 대한 권한을 부여할 수 있습니다. 사용자가 객체를 사용하려면 사용자 또는 해당 사용자를 포함하는 그룹에 필요한 권한을 부여해야 합니다. 데이터베이스 슈퍼 사용자는 데이터베이스 소유자와 동일한 권한을 갖습니다.

Amazon Redshift는 SELECT, INSERT, UPDATE, DELETE, REFERENCES, CREATE, TEMPORARY, USAGE 권한을 지원합니다. 권한마다 연결되는 객체 유형이 다릅니다. Amazon Redshift가 지원하는 데이터베이스 객체 권한에 대해서는 GRANT 명령을 참조하세요.

소유자에게만 객체를 수정 또는 삭제할 수 있는 권한이 있습니다.

기본적으로 모든 사용자는 데이터베이스의 PUBLIC 스키마에서 CREATE 및 USAGE 권한을 갖습니다. 사용자가 데이터베이스의 PUBLIC 스키마에 객체를 생성하는 것을 허용하지 않으려면 REVOKE 명령을 사용하여 해당 권한을 제거하세요.

이전에 부여한 권한을 취소하려면 REVOKE 명령을 사용하세요. DROP, GRANT, REVOKE 권한 등 객체 소유자의 권한은 묵시적이며 부여하거나 취소할 수 없습니다. 객체 소유자는 본인의 일반적 권한을 취소하여 예컨대 테이블을 본인 및 다른 사용자에게 읽기 전용으로 만들 수 있습니다. 슈퍼 사용자는 GRANT 및 REVOKE 명령과 상관없이 모든 권한을 보유합니다.

슈퍼 사용자

데이터베이스 슈퍼 사용자는 모든 데이터베이스에 대해 데이터베이스 소유자와 동일한 권한을 갖습니다.

클러스터를 시작할 때 생성한 사용자인 관리자는 슈퍼 사용자입니다.

슈퍼 사용자만이 슈퍼 사용자를 만들 수 있습니다.

Amazon Redshift 시스템 테이블과 시스템 뷰는 슈퍼 사용자에게만 표시되거나 모든 사용자에게 표시됩니다. "슈퍼 사용자에게 보임"으로 지정된 시스템 테이블 및 시스템 보기는 슈퍼 사용자만이 쿼리할 수 있습니다. 자세한 내용은 SYS 모니터링 뷰 섹션을 참조하세요.

슈퍼 사용자는 모든 카탈로그 테이블을 볼 수 있습니다. 자세한 내용은 시스템 카탈로그 테이블 섹션을 참조하세요.

데이터베이스 슈퍼 사용자는 모든 권한 검사를 우회합니다. 슈퍼 사용자는 GRANT 및 REVOKE 명령과 상관없이 모든 권한을 보유합니다. 슈퍼 사용자 역할을 사용할 때는 주의하세요. 대부분의 작업은 슈퍼 사용자가 아닌 역할로 수행하는 것이 좋습니다. 보다 제한적인 권한이 있는 관리자 역할을 만들 수 있습니다. 역할 생성에 대한 자세한 내용은 역할 기반 액세스 제어(RBAC) 섹션을 참조하세요.

새 데이터베이스 슈퍼 사용자를 만들려면 슈퍼 사용자로 데이터베이스에 로그온하고 CREATEUSER 권한을 사용하여 CREATE USER 명령이나 ALTER USER 명령을 실행합니다.

```
CREATE USER adminuser CREATEUSER PASSWORD '1234Admin';
ALTER USER adminuser CREATEUSER;
```

슈퍼 사용자를 생성, 변경 또는 삭제하려면 동일한 명령을 사용하여 사용자를 관리합니다. 자세한 내용은 사용자 생성, 변경 및 삭제 섹션을 참조하세요.

사용자

Amazon Redshift SQL 명령 CREATE USER 및 ALTER USER 명령을 사용하여 데이터베이스 사용자를 만들고 관리할 수 있습니다. 또는 사용자 정의 Amazon Redshift JDBC 또는 ODBC 드라이버를 사용하여 SQL 클라이언트를 구성할 수 있습니다. 이들은 데이터베이스 로그온 프로세스의 일환으로 데이터베이스 사용자 및 임시 암호 생성 프로세스를 관리합니다.

드라이버는 AWS Identity and Access Management(IAM) 인증을 바탕으로 데이터베이스 사용자를 인증합니다. 이미 AWS 밖에서 사용자 자격 증명을 관리하고 있는 경우 SAML 2.0을 준수하는 자격 증명

공급자(IdP)를 통해 Amazon Redshift 리소스에 대한 액세스를 관리할 수 있습니다. IAM 역할을 사용해 IdP 및 AWS를 구성하고 페더레이션 사용자가 임시 데이터베이스 자격 증명을 새로 만들고 Amazon Redshift 데이터베이스에 로그온하도록 허용할 수 있습니다. 자세한 내용은 IAM 인증을 이용한 데이터베이스 사용자 자격 증명 생성을 참조하세요.

Amazon Redshift 사용자는 데이터베이스 슈퍼 사용자만 생성하고 삭제할 수 있습니다. 사용자는 Amazon Redshift에 로그온할 때 인증됩니다. 데이터베이스 및 데이터베이스 객체(예: 테이블)를 소유할 수 있습니다. 또한 이러한 객체에 대한 권한을 사용자, 그룹 및 스키마에 부여하여 누가 어느 객체에 액세스할 수 있는지 제어할 수 있습니다. CREATE DATABASE 권리가 있는 사용자는 데이터베이스를 만들고 이러한 데이터베이스에 대한 권한을 부여할 수 있습니다. 슈퍼 사용자는 모든 데이터베이스에 대해 데이터베이스 소유 권한을 갖습니다.

사용자 생성, 변경 및 삭제

데이터베이스 사용자는 데이터 웨어하우스 클러스터에서 전역적입니다(개별 데이터베이스에 따라 달라지지 않음).

- 사용자를 만들려면 사용자 생성 명령을 사용하세요.
- 슈퍼 사용자를 만들려면 CREATEUSER 옵션과 함께 사용자 생성 명령을 사용하세요.
- 기존 사용자를 제거하려면 DROP USER 명령을 사용하세요.
- 사용자를 변경하려면(예: 암호 변경) ALTER USER 명령을 사용하세요.
- 사용자 목록을 보려면 PG_USER 카탈로그 테이블을 쿼리하세요.

```
select * from pg_user;

  usename   | usesysid | usecreatedb | usesuper | usecatupd |  passwd  | valuntil |
 useconfig
------------+----------+-------------+----------+-----------+----------+----------
+-----------
 rdsdb      |        1 | t           | t        | t         | ******** |          |
 masteruser |      100 | t           | t        | f         | ******** |          |
 dwuser     |      101 | f           | f        | f         | ******** |          |
 simpleuser |      102 | f           | f        | f         | ******** |          |
 poweruser  |      103 | f           | t        | f         | ******** |          |
 dbuser     |      104 | t           | f        | f         | ******** |          |
(6 rows)
```

그룹

그룹은 그룹에 연결된 어떤 권한이건 모두 부여되는 사용자들의 모음입니다. 그룹을 사용하여 권한을 부여할 수 있습니다. 예를 들어 영업, 관리, 지원을 위한 다양한 그룹을 생성하여 각 그룹의 사용자에게 업무에 필요한 데이터에 액세스할 수 있는 적절한 권한을 부여할 수 있습니다. 그룹 수준에서 권한을 부여하거나 취소할 수 있으며, 이러한 변경 사항은 슈퍼 사용자를 제외한 그룹의 모든 구성원에게 적용됩니다.

모든 사용자 그룹을 보려면 PG_GROUP 시스템 카탈로그 테이블을 쿼리합니다.

```
select * from pg_group;
```

예를 들어 모든 데이터베이스 사용자를 그룹별로 나열하려면 다음 SQL을 실행합니다.

```
SELECT u.usesysid
,g.groname
,u.usename
FROM pg_user u
LEFT JOIN pg_group g ON u.usesysid = ANY (g.grolist)
```

그룹 생성, 변경 및 삭제

슈퍼 사용자만이 그룹을 생성, 변경 또는 삭제할 수 있습니다.

다음 작업을 수행할 수 있습니다.

- 그룹을 만들려면 create group 명령을 사용하세요.
- 기존 그룹에 사용자를 추가하거나 기존 그룹에서 제거하려면 ALTER GROUP 명령을 사용하세요.
- 그룹을 삭제하려면 DROP GROUP 명령을 사용하세요. 이 명령은 그룹만을 삭제하며 그룹 구성원인 사용자는 삭제하지 않습니다.

사용자 및 그룹 액세스 제어 예

이 예는 사용자 그룹과 사용자를 만든 다음 이들에게 웹 애플리케이션 클라이언트에 연결되는 Amazon Redshift 데이터베이스에 대한 권한을 부여합니다. 이 예에서는 웹 애플리케이션 일반 사용자, 웹 애플리케이션 파워 유저, 웹 개발자 등 세 가지 사용자 그룹을 가정합니다.

그룹에서 사용자를 제거하는 방법에 대한 자세한 내용은 ALTER GROUP 섹션을 참조하세요.

1. 사용자가 할당될 그룹을 만듭니다. 다음 명령 세트는 세 가지 사용자 그룹을 만듭니다.

```
create group webappusers;

create group webpowerusers;

create group webdevusers;
```

2. 권한이 서로 다른 몇몇 데이터베이스 사용자를 만들어 그룹에 추가합니다.

   a. 두 명의 사용자를 만들어 WEBAPPUSERS 그룹에 추가합니다.

```
create user webappuser1 password 'webAppuser1pass'
in group webappusers;

create user webappuser2 password 'webAppuser2pass'
in group webappusers;
```

   b. 웹 개발자용 계정을 만들어 WEBDEVUSERS 그룹에 추가합니다.

```
create user webdevuser1 password 'webDevuser2pass'
in group webdevusers;
```

   c. 슈퍼 사용자를 생성합니다. 이 사용자는 다른 사용자를 생성하는 관리 권한을 갖습니다.

```
create user webappadmin  password 'webAppadminpass1'
createuser;
```

3. 웹 애플리케이션이 사용하는 데이터베이스 테이블에 연결될 스키마를 생성해 다양한 사용자 그룹에게 이 스키마에 액세스할 권한을 부여합니다.

   a. WEBAPP 스키마를 생성합니다.

```
create schema webapp;
```

   b. WEBAPPUSERS 그룹에 USAGE 권한을 부여합니다.

```
grant usage on schema webapp to group webappusers;
```

   c. WEBPOWERUSERS 그룹에 USAGE 권한을 부여합니다.

```
grant usage on schema webapp to group webpowerusers;
```

d. WEBDEVUSERS 그룹에 ALL 권한을 부여합니다.

```
grant all on schema webapp to group webdevusers;
```

기본적인 사용자 및 그룹이 이제 설정되었습니다. 이제 사용자와 그룹을 변경할 수 있습니다.

4. 예를 들어 다음 명령은 WEBAPPUSER1의 search_path 파라미터를 변경합니다.

```
alter user webappuser1 set search_path to webapp, public;
```

SEARCH_PATH는 테이블과 함수 같은 데이터베이스 객체를 스키마가 지정되지 않은 단순한 이름으로 참조할 때 객체에 대한 스키마 검색 순서를 지정합니다.

5. 그룹을 만든 후 사용자를 그룹에 추가할 수 있습니다. 예를 들어 WEBAPPUSER2를 WEBPOWERUSERS 그룹에 추가할 수 있습니다.

```
alter group webpowerusers add user webappuser2;
```

스키마

데이터베이스에는 하나 이상의 명명된 스키마가 포함되어 있습니다. 데이터베이스의 각 스키마에는 테이블과 그 밖의 명명된 객체가 포함됩니다. 기본적으로 데이터베이스에는 PUBLIC이라는 이름의 스키마가 하나 있습니다. 스키마를 사용하여 일반 이름으로 데이터베이스 객체를 그룹화할 수 있습니다. 스키마는 중첩될 수 없다는 점만 빼면 파일 시스템 디렉터리와 비슷합니다.

같은 데이터베이스 내의 다른 스키마에서 동일한 데이터베이스 객체 이름을 충돌 없이 사용할 수 있습니다. 예를 들어 MY_SCHEMA와 YOUR_SCHEMA에 MYTABLE이라는 이름의 테이블이 모두 포함될 수 있습니다. 필요한 권한이 있는 사용자는 데이터베이스의 여러 스키마에서 객체에 액세스할 수 있습니다.

기본적으로 데이터베이스의 검색 경로의 첫 번째 스키마 내에서 객체가 생성됩니다. 자세한 내용은 이 섹션의 후반부에서 검색 경로 섹션을 참조하세요.

스키마는 다음과 같은 방법으로 다중 사용자 환경에서 조직 및 동시성 문제에 도움이 될 수 있습니다.

- 여러 개발자가 서로 간섭 없이 같은 데이터베이스에서 작업이 가능.
- 보다 관리하기 쉽게 데이터베이스 객체를 논리적 그룹으로 체계화.
- 다른 애플리케이션에서 사용하는 객체의 이름과 충돌하지 않도록 객체를 별도의 스키마에 넣는 능력을 애플리케이션에 제공.

검색 경로

검색 경로는 쉼표로 분리된 스키마 이름 목록을 사용하여 search_path 파라미터에서 정의됩니다. 검색 경로는 테이블이나 함수와 같은 객체를 스키마 한정자가 포함되지 않은 단순한 이름으로 참조할 때 스키마가 검색되는 순서를 지정합니다.

대상 스키마를 지정하지 않고 생성된 객체는 검색 경로에 나열되는 첫 번째 스키마에 추가됩니다. 동일한 이름의 객체가 다른 스키마에 존재하는 경우, 스키마를 지정하지 않은 객체 이름은 검색 경로에서 해당 이름의 객체가 포함된 첫 번째 스키마를 참조합니다.

현재 세션의 기본 스키마를 변경하려면 SET 명령을 사용합니다.

자세한 내용은 구성 참조의 search_path 설명을 참조하세요.

스키마 생성, 변경 및 삭제

모든 사용자는 직접 스키마를 생성하고 변경하거나 삭제할 수 있습니다.

다음 작업을 수행할 수 있습니다.

- 스키마를 생성하려면 CREATE SCHEMA 명령을 사용합니다.
- 스키마의 소유자를 변경하려면 ALTER SCHEMA 명령을 사용합니다.
- 스키마와 그 객체를 삭제하려면 DROP SCHEMA 명령을 사용합니다.
- 스키마 내에 테이블을 생성하려면 schema_name.table_name 형식으로 테이블을 생성합니다.

모든 스키마 목록을 보려면 PG_NAMESPACE 시스템 카탈로그 테이블을 쿼리합니다.

```
select * from pg_namespace;
```

스키마에 속한 테이블의 목록을 보려면 PG_TABLE_DEF 시스템 카탈로그 테이블을 쿼리합니다. 예를 들어 다음 쿼리는 PG_CATALOG 스키마에 있는 테이블의 목록을 반환합니다.

```
select distinct(tablename) from pg_table_def
where schemaname = 'pg_catalog';
```

스키마 기반 권한

스키마 기반 권한은 스키마 소유자에 의해 결정됩니다.

- 기본적으로 모든 사용자는 데이터베이스의 PUBLIC 스키마에서 CREATE 및 USAGE 권한을 갖습니다. 사용자가 데이터베이스의 PUBLIC 스키마에 객체를 생성하는 것을 허용하지 않으려면 REVOKE 명령을 사용하여 해당 권한을 제거하세요.
- 객체 소유자에 의해 USAGE 권한이 부여되지 않은 경우, 사용자는 소유하지 않은 스키마의 어떤 객체에도 액세스할 수 없습니다.
- 다른 사용자가 생성한 스키마에 대한 CREATE 권한을 부여받은 사용자는 해당 스키마에서 객체를 만들 수 있습니다.

역할 기반 액세스 제어(RBAC)

역할 기반 액세스 제어(RBAC)를 사용하여 Amazon Redshift에서 데이터베이스 권한을 관리하면 Amazon Redshift에서 보안 권한 관리를 간소화할 수 있습니다. 사용자가 광범위하거나 세밀한 수준에서 수행할 수 있는 작업을 제어하여 민감한 데이터에 대한 액세스를 보호할 수 있습니다. 또한 일반적으로 슈퍼 사용자로 제한되는 작업에 대한 사용자 액세스를 제어할 수도 있습니다. 서로 다른 역할에 서로 다른 권한을 할당하고 이를 다른 사용자에게 할당하여 사용자 액세스를 보다 세부적으로 제어할 수 있습니다.

역할이 할당된 사용자는 권한이 부여된 할당된 역할에 의해 지정된 작업만 수행할 수 있습니다. 예를 들어 CREATE TABLE 및 DROP TABLE 권한이 있는 할당된 역할이 있는 사용자는 이러한 작업을 수행할 수 있는 권한만 부여됩니다. 서로 다른 사용자가 작업에 필요한 데이터에 액세스할 수 있도록 서로 다른 수준의 보안 권한을 부여하여 사용자의 액세스를 제어할 수 있습니다.

RBAC는 관련된 객체 유형에 상관없이 역할 요구 사항에 따라 사용자에게 최소 권한의 원칙을 적용합니다. 권한 부여 및 취소는 개별 데이터베이스 객체에 대한 권한을 업데이트할 필요 없이 역할 수준에서 수행됩니다.

RBAC를 사용하면 슈퍼 사용자 권한을 요구했던 명령을 실행할 수 있는 권한을 갖는 역할을 만들 수 있습니다. 사용자는 이러한 권한을 포함하는 역할로 권한이 부여되는 한 이러한 명령을 실행할 수 있습니다. 마찬가지로 역할을 만들어 특정 명령에 대한 액세스를 제한하고 역할을 통해 권한이 부여된 슈퍼 사용자 또는 사용자에게 역할을 할당할 수도 있습니다.

Amazon Redshift RBAC 작동 방식에 대해 알아보려면 다음 동영상을 시청하세요. Amazon Redshift Redshift의 역할 기반 액세스 제어(RBAC) 소개

역할 계층 구조

역할은 사용자 또는 다른 역할에 할당할 수 있는 권한 모음입니다. 역할에 시스템 또는 데이터베이스 권한을 할당할 수 있습니다. 사용자는 할당된 역할에서 권한을 상속합니다.

RBAC에서 사용자는 중첩된 역할을 가질 수 있습니다. 사용자와 역할 모두에 역할을 부여할 수 있습니다. 사용자에게 역할을 부여할 때 이 역할이 포함하는 모든 권한을 사용자에게 부여합니다. 사용자에게 역할 r1을 부여할 때 r1의 권한을 사용자에게 부여합니다. 이제 사용자는 r1의 권한과 이미 가지고 있는 모든 기존 권한을 갖습니다.

역할(r1)을 다른 역할(r2)에 부여할 때 r1의 모든 권한을 r2에 부여합니다. 또한 r2를 다른 역할(r3)에 부여할 때 r3의 권한은 r1 및 r2의 권한의 조합입니다. 역할 계층 구조는 r2가 r1에서 권한을 상속하도록 합니다. Amazon Redshift는 각 역할 권한 부여로 권한을 전파합니다. r1을 r2에 부여한 다음 r2를 r3에 부여하면 세 역할의 모든 권한을 r3에 부여합니다. 따라서 사용자에게 r3을 부여하면 사용자는 세 역할의 모든 권한을 갖습니다.

Amazon Redshift는 역할 권한 부여 순환 생성을 허용하지 않습니다. 역할 권한 부여 순환은 중첩된 역할이 역할 계층 구조에서 이전 역할에 다시 할당될 때 발생합니다(예: r3이 다시 r1에 할당됨). 역할 생성 및 관리에 대한 자세한 내용은 RBAC에서 역할 관리 단원을 참조하세요.

역할 할당

CREATE ROLE 권한이 있는 슈퍼 사용자 및 일반 사용자는 CREATE ROLE 문을 사용하여 역할을 만들 수 있습니다. 슈퍼 사용자 및 역할 관리자는 GRANT ROLE 문을 사용하여 다른 사용자에게 역할을 부여할 수 있습니다. REVOKE ROLE 문을 사용하여 다른 사용자의 역할을 취소하고 DROP ROLE 문을 사용하여 역할을 삭제할 수 있습니다. 역할 관리자에는 ADMIN OPTION 권한이 있는 역할이 부여된 역할 소유자 및 사용자가 포함됩니다.

슈퍼 사용자 또는 역할 관리자만이 역할을 부여 및 취소할 수 있습니다. 하나 이상의 역할 또는 사용자에게 하나 이상의 역할을 부여하거나 취소할 수 있습니다. GRANT ROLE 문에서 WITH ADMIN OPTION 옵션을 사용하여 모든 피부여자에게 부여된 모든 역할에 대한 관리 옵션을 제공하세요.

Amazon Redshift는 여러 역할을 부여하거나 여러 피부여자를 갖는 등 다양한 역할 할당 조합을 지원합니다. WITH ADMIN OPTION은 사용자에게만 적용되며 역할에는 적용되지 않습니다. 마찬가지로 REVOKE ROLE 문에서 WITH ADMIN OPTION 옵션을 사용하여 피부여자의 역할 및 관리 권한 부여를 제거하세요. ADMIN OPTION과 함께 사용하면 역할의 관리 권한 부여만 취소됩니다.

다음 예에는 `user2`에서 `sample_role2` 역할의 관리 권한 부여를 취소합니다.

```
REVOKE ADMIN OPTION FOR sample_role2 FROM user2;
```

역할 생성 및 관리에 대한 자세한 내용은 RBAC에서 역할 관리 단원을 참조하세요.

Amazon Redshift 시스템 정의 역할

Amazon Redshift는 특정 권한으로 정의된 몇 가지 시스템 정의 역할을 제공합니다. 시스템 관련 역할은 sys: 접두사로 시작합니다. 적절하게 액세스하는 사용자만이 시스템 정의 역할을 변경하거나 사용자 지정 시스템 정의 역할을 만들 수 있습니다. 사용자 지정 시스템 정의 역할에 sys: 접두사를 사용할 수 없습니다.

다음 표에는 역할과 그 권한이 요약되어 있습니다.

역할 이름	설명		
sys:monitor	이 역할에는 카탈로그 또는 시스템 테이블에 액세스할 수 있는 권한이 있습니다.		
sys:operator	이 역할에는 카탈로그 또는 시스템 테이블에 액세스하거나 쿼리를 분석, 정리 또는 취소할 수 있는 권한이 있습니다.		
sys:dba	이 역할에는 스키마 생성, 테이블 생성, 스키마 삭제, 테이블 삭제 및 테이블을 자를 수 있는 권한이 있습니다. 저장된 프로시저를 만들거나 바꾸고, 프로시저를 삭제하고, 함수를 만들거나 바꾸고, 외부 함수를 만들거나 바꾸고, 보기를 만들고, 보기를 삭제할 수 있는 권한이 있습니다. 또한 이 역할은 sys:opera		

역할 이름	설명		
	tor 역할의 모든 권한을 상속합니다.		
sys:superuser	이 역할에는 RBAC에 대한 시스템 권한에 정의된 지원되는 모든 시스템 권한이 있습니다.		
sys:secadmin	• 이 역할에는 사용자를 만들고, 사용자를 변경하고, 사용자를 삭제하고, 역할을 만들고, 역할을 삭제하고, 역할을 부여할 수 있는 권한이 있습니다. • 이 역할에는 관계에서 RLS를 켜거나 끌 수 있는 권한과 RLS 및 DDM 정책(CREATE, DROP, ATTACH, DETACH, ALTER)을 관리할 권한이 있습니다. 또한 이 역할에는 기본적으로 EXPLAIN RLS, IGNORE RLS 및 EXPLAIN MASKING 권한이 부여됩니다. • 이 역할은 권한이 명시적으로 역할에 부여된 경우에만 사용자 테이블에 액세스할 수 있습니다.		

데이터 공유를 위한 시스템 정의 역할 및 사용자

Amazon Redshift는 데이터 공유 및 데이터 공유 소비자에 해당하는 내부 용도의 역할과 사용자를 생성합니다. 각 내부 역할 이름과 사용자 이름에는 예약된 네임스페이스 접두사 `ds:`가 있습니다. 다음과 같은 형식을 보유합니다.

명칭	설명		
`ds:`*`sharename`*	데이터 공유에 해당하는 시스템 역할입니다.		
`ds:`*`sharename_consumer`*	데이터 공유 소비자에 해당하는 시스템 사용자입니다.		

각 데이터 공유별로 데이터 공유 역할이 생성됩니다. 현재 데이터 공유에 부여된 모든 권한을 보유합니다. 데이터 공유의 각 소비자별로 데이터 공유 사용자가 생성됩니다. 단일 데이터 공유 역할에 대한 권한이 부여됩니다. 여러 데이터 공유에 추가된 소비자는 각 데이터 공유별로 데이터 공유 사용자를 한 명씩 생성하게 됩니다.

데이터 공유가 제대로 작동하려면 이러한 사용자와 역할이 필요합니다. 수정하거나 삭제할 수 없으며 고객이 실행하는 태스크에 액세스하거나 사용할 수 없습니다. 이 내용은 안심하고 무시해도 됩니다. 데이터 공유에 대한 자세한 내용은 Amazon Redshift에서 클러스터 간 데이터 공유를 참조하세요.

> **Note**
>
> `ds:` 접두사를 사용하여 사용자 정의 역할 또는 사용자를 생성할 수는 없습니다.

RBAC에 대한 시스템 권한

다음은 역할에서 부여하거나 취소할 수 있는 시스템 권한 목록입니다.

Command	명령을 실행하려면 다음 방법 중 하나로 권한을 받아야 합니다.		
역할 생성	• 슈퍼 사용자.		

Command	명령을 실행하려면 다음 방법 중 하나로 권한을 받아야 합니다.		
	• CREATE ROLE 권한이 있는 사용자.		
DROP ROLE	• 슈퍼 사용자. • 역할을 만들었거나 WITH ADMIN OPTION 권한이 있는 역할이 부여된 사용자인 역할 소유자.		
사용자 생성	• 슈퍼 사용자. • CREATE USER 권한이 있는 사용자. 이러한 사용자는 슈퍼 사용자를 만들 수 없습니다.		
DROP USER	• 슈퍼 사용자. • DROP USER 권한이 있는 사용자.		
ALTER USER	• 슈퍼 사용자. • ALTER USER 권한이 있는 사용자. 이러한 사용자는 사용자를 슈퍼 사용자로 변경하거나 슈퍼 사용자를 사용자로 변경할 수 없습니다. • 자신의 암호를 변경하려는 현재 사용자.		

Command	명령을 실행하려면 다음 방법 중 하나로 권한을 받아야 합니다.		
CREATE SCHEMA	• 슈퍼 사용자. • CREATE SCHEMA 권한이 있는 사용자.		
DROP SCHEMA	• 슈퍼 사용자. • DROP SCHEMA 권한이 있는 사용자. • 스키마 소유자.		
ALTER DEFAULT PRIVILEGES	• 슈퍼 사용자. • ALTER DEFAULT PRIVILEGES 권한이 있는 사용자. • 자신의 기본 액세스 권한을 변경하는 사용자. • 액세스 권한이 있는 스키마에 대한 권한을 설정하는 사용자.		
ACCESS CATALOG	• 슈퍼 사용자. • ACCESS CATALOG 권한이 있는 사용자.		
ACCESS SYSTEM TABLE	• 슈퍼 사용자. • ACCESS SYSTEM TABLE 권한이 있는 사용자.		

Command	명령을 실행하려면 다음 방법 중 하나로 권한을 받아야 합니다.		
CREATE TABLE	• 슈퍼 사용자. • CREATE TABLE 권한이 있는 사용자. • 스키마에 대한 CREATE 권한이 있는 사용자.		
DROP TABLE	• 슈퍼 사용자. • DROP TABLE 권한이 있는 사용자. • 스키마에 대한 USAGE 권한이 있는 테이블 소유자.		
ALTER TABLE	• 슈퍼 사용자. • ALTER TABLE 권한이 있는 사용자. • 스키마에 대한 USAGE 권한이 있는 테이블 소유자.		

Command	명령을 실행하려면 다음 방법 중 하나로 권한을 받아야 합니다.		

CREATE OR REPLACE FUNCTION	• CREATE FUNCTION의 경우
	  • 슈퍼 사용자.
	  • CREATE OR REPLACE FUNCTION 권한이 있는 사용자.
	  • 언어에 대한 USAGE 권한이 있는 사용자.
	• REPLACE FUNCTION의 경우
	  • 슈퍼 사용자.
	  • CREATE OR REPLACE FUNCTION 권한이 있는 사용자.
	  • 함수 소유자.		

CREATE OR REPLACE EXTERNAL FUNCTION	• 슈퍼 사용자. • CREATE OR REPLACE EXTERNAL FUNCTION 권한이 있는 사용자.		
DROP FUNCTION	• 슈퍼 사용자. • DROP FUNCTION 권한이 있는 사용자. • 함수 소유자.		

Command	명령을 실행하려면 다음 방법 중 하나로 권한을 받아야 합니다.		

CREATE OR REPLACE PROCEDURE	• CREATE PROCEDURE의 경우
	  • 슈퍼 사용자.
	  • CREATE OR REPLACE PROCEDURE 권한이 있는 사용자.
	  • 언어에 대한 USAGE 권한이 있는 사용자.
	• REPLACE PROCEDURE의 경우
	  • 슈퍼 사용자.
	  • CREATE OR REPLACE PROCEDURE 권한이 있는 사용자.
	  • 프로시저 소유자.		

DROP PROCEDURE	• 슈퍼 사용자. • DROP PROCEDURE 권한이 있는 사용자. • 프로시저 소유자.		

Command	명령을 실행하려면 다음 방법 중 하나로 권한을 받아야 합니다.		

CREATE OR REPLACE VIEW	• CREATE VIEW의 경우
	  • 슈퍼 사용자.
	  • CREATE OR REPLACE VIEW 권한이 있는 사용자.
	  • 스키마에 대한 CREATE 권한이 있는 사용자.
	• REPLACE VIEW의 경우
	  • 슈퍼 사용자.
	  • CREATE OR REPLACE VIEW 권한이 있는 사용자.
	  • 보기 소유자.		

DROP VIEW	• 슈퍼 사용자. • DROP VIEW 권한이 있는 사용자. • 보기 소유자.		

Command	명령을 실행하려면 다음 방법 중 하나로 권한을 받아야 합니다.		
CREATE MODEL	• 슈퍼 사용자. • CREATE MODEL 시스템 권한이 있는 사용자. CREATE MODEL의 관계를 읽을 수 있어야 합니다. • CREATE MODEL 권한이 있는 사용자.		
DROP MODEL	• 슈퍼 사용자. • DROP MODEL 권한이 있는 사용자. • 모델 소유자. • 스키마 소유자.		
CREATE DATASHARE	• 슈퍼 사용자. • CREATE DATASHARE 권한이 있는 사용자. • 데이터베이스 소유자.		

Command	명령을 실행하려면 다음 방법 중 하나로 권한을 받아야 합니다.		
ALTER DATASHARE	• 슈퍼 사용자. • ALTER DATASHARE 권한이 있는 사용자. • datashare에 대한 ALTER 또는 ALL 권한이 있는 사용자. • datashare에 특정 객체를 추가하려면 이러한 사용자에게 객체에 대한 권한이 있어야 합니다. 사용자는 객체 소유자이거나 객체에 대한 SELECT, USAGE 또는 ALL 권한이 있어야 합니다.		
DROP DATASHARE	• 슈퍼 사용자. • DROP DATASHARE 권한이 있는 사용자. • 데이터베이스 소유자.		
CREATE LIBRARY	• 슈퍼 사용자. • CREATE LIBRARY 권한이 있거나 지정된 언어의 권한이 있는 사용자.		

Command	명령을 실행하려면 다음 방법 중 하나로 권한을 받아야 합니다.		
DROP LIBRARY	• 슈퍼 사용자. • DROP LIBRARY 권한이 있는 사용자. • 라이브러리 소유자.		
ANALYZE	• 슈퍼 사용자. • ANALYZE 권한이 있는 사용자. • 관계 소유자. • 테이블이 공유되는 데이터베이스 소유자.		
CANCEL	• 자신의 쿼리를 취소하는 슈퍼 사용자. • 사용자의 쿼리를 취소하는 슈퍼 사용자. • 사용자의 쿼리를 취소하는 CANCEL 권한이 있는 사용자. • 자신의 쿼리를 취소하는 사용자.		
TRUNCATE TABLE	• 슈퍼 사용자. • TRUNCATE TABLE 권한이 있는 사용자. • 테이블 소유자.		

Command	명령을 실행하려면 다음 방법 중 하나로 권한을 받아야 합니다.		
VACUUM	• 슈퍼 사용자. • VACUUM 권한이 있는 사용자. • 테이블 소유자. • 테이블이 공유되는 데이터베이스 소유자.		
IGNORE RLS	• 슈퍼 사용자. • `sys:secadmin` 역할 내 사용자.		
EXPLAIN RLS	• 슈퍼 사용자. • `sys:secadmin` 역할 내 사용자.		
EXPLAIN MASKING	• 슈퍼 사용자. • `sys:secadmin` 역할 내 사용자.		

데이터베이스 객체 권한

시스템 권한 외에도 Amazon Redshift는 액세스 옵션을 정의하는 데이터베이스 객체 권한을 포함합니다. 테이블 및 보기의 데이터 읽기, 데이터 쓰기, 테이블 생성 및 테이블 삭제 기능과 같은 옵션을 포함합니다. 자세한 내용은 GRANT 단원을 참조하십시오.

RBAC를 사용하여 시스템 권한과 비슷한 방법으로 데이터베이스 객체 권한을 역할에 할당할 수 있습니다. 그런 다음 사용자에게 역할을 할당하고, 시스템 권한이 있는 사용자에게 권한을 부여하고, 데이터베이스 권한이 있는 사용자에게 권한을 부여할 수 있습니다.

RBAC에 대한 ALTER DEFAULT PRIVILEGES

ALTER DEFAULT PRIVILEGES 문을 사용하여 앞으로 지정된 사용자가 만들 객체에 적용되는 기본 액세스 권한 집합을 정의하세요. 기본적으로 사용자는 자신의 기본 액세스 권한만 변경할 수 있습니다. RBAC를 사용하여 역할의 기본 액세스 권한을 설정할 수 있습니다. 자세한 내용은 ALTER DEFAULT PRIVILEGES 명령을 참조하세요.

RBAC를 사용하여 시스템 권한과 비슷하게 데이터베이스 객체 권한을 역할에 할당할 수 있습니다. 그런 다음 사용자에게 역할을 할당하고 시스템 및/또는 데이터베이스 권한이 있는 사용자에게 권한을 부여할 수 있습니다.

RBAC에서 역할 사용에 대한 고려 사항

RBAC 역할을 사용하여 작업 시 다음 사항을 고려하세요.

- Amazon Redshift는 역할 권한 부여 순환을 허용하지 않습니다. r1을 r2에 부여한 다음 r2를 r1에 부여할 수 없습니다.
- RBAC는 네이티브 Amazon Redshift 객체와 Amazon Redshift Spectrum 테이블 모두에서 작동합니다.
- Amazon Redshift 관리자는 클러스터를 최신 유지 관리 패치로 업그레이드함으로써 RBAC를 활성화하여 시작할 수 있습니다.
- CREATE ROLE 시스템 권한이 있는 슈퍼 사용자 및 사용자만 역할을 만들 수 있습니다.
- 슈퍼 사용자 및 역할 관리자만 역할을 수정하거나 삭제할 수 있습니다.
- 역할 이름은 사용자 이름과 같을 수 없습니다.
- 역할 이름은 “:/\n.”과 같은 잘못된 문자를 포함할 수 없습니다.
- 역할 이름은 PUBLIC과 같은 예약어가 될 수 없습니다.
- 역할 이름은 기본 역할에 대해 예약된 접두사 `sys:`로 시작할 수 없습니다.
- 다른 역할에 부여된 경우 RESTRICT 파라미터가 있는 역할은 삭제할 수 없습니다. 기본 설정은 RESTRICT입니다. 다른 역할을 상속한 역할을 삭제하려고 하면 Amazon Redshift에서 오류가 발생합니다.
- 역할에 대한 관리자 권한이 없는 사용자는 역할을 부여하거나 취소할 수 없습니다.
- RBAC는 시스템 테이블 및 뷰에서 완전히 지원되지 않습니다. 시스템 테이블 및 뷰에서의 RBAC 권한은 업그레이드, 다운그레이드 또는 크기 조정을 통해 유지되지 않습니다. Amazon Redshift 시스템 정의 역할를 사용하여 시스템 테이블을 관리하고 권한을 확인하는 것이 좋습니다. 시스템 테이블에 관한 자세한 내용은 시스템 테이블 및 뷰 참조 섹션으로 이동해 참조하시기 바랍니다.

RBAC에서 역할 관리

다음 작업을 수행하려면 다음 명령을 사용하세요.

- 역할을 만들려면 역할 생성 명령을 사용하세요.
- 역할의 이름을 바꾸거나 역할의 소유자를 변경하려면 역할 변경 명령을 사용하세요.
- 역할을 삭제하려면 DROP ROLE 명령을 사용하세요.
- 사용자에게 역할을 부여하려면 GRANT 명령을 사용하세요.
- 사용자에게서 역할을 취소하려면 REVOKE 명령을 사용하세요.
- 역할에 시스템 권한을 부여하려면 GRANT 명령을 사용하세요.
- 역할에서 시스템 권한을 취소하려면 REVOKE 명령을 사용하세요.

클러스터 또는 작업 그룹의 역할 목록을 보려면 SVV_ROLES 섹션을 참조하세요.

자습서: RBAC를 사용한 역할 생성 및 쿼리

RBAC를 사용하면 슈퍼 사용자 권한을 요구했던 명령을 실행할 수 있는 권한을 갖는 역할을 만들 수 있습니다. 사용자는 이러한 권한을 포함하는 역할로 권한이 부여되는 한 이러한 명령을 실행할 수 있습니다.

이 자습서에서는 생성하는 데이터베이스에서 역할 기반 액세스 제어(RBAC)를 사용하여 권한을 관리합니다. 그런 다음 데이터베이스에 연결하고 서로 다른 두 역할로 데이터베이스를 쿼리하여 RBAC의 기능을 테스트합니다.

데이터베이스를 쿼리할 때 만들고 사용하는 두 가지 역할은 `sales_ro`와 `sales_rw`입니다. `sales_ro` 역할을 생성하고 `sales_ro` 역할을 가진 사용자로 데이터를 쿼리합니다. `sales_ro` 사용자는 SELECT 명령만 사용할 수 있고 UPDATE 명령은 사용할 수 없습니다. 그런 다음 `sales_rw` 역할을 생성하고 `sales_rw` 역할을 가진 사용자로 데이터를 쿼리합니다. `sales_rw` 사용자는 SELECT 명령과 UPDATE 명령을 사용할 수 있습니다.

마찬가지로 역할을 생성하여 특정 명령으로만 액세스를 제한하고 슈퍼 사용자 또는 사용자에게 역할을 할당합니다.

업무

- 사전 조건
- 1단계: 관리 사용자 생성
- 2단계: 스키마 설정

- 3단계: 읽기 전용 사용자 생성
- 4단계: 읽기 전용 사용자로 데이터 쿼리
- 5단계: 읽기-쓰기 사용자 생성
- 6단계: 상속한 읽기 전용 역할을 가진 사용자로 데이터 쿼리
- 7단계: 읽기-쓰기 역할에 업데이트 및 삽입 권한 부여
- 8단계: 읽기-쓰기 사용자로 데이터 쿼리
- 9단계: 관리 사용자로 데이터베이스의 테이블 분석 및 정리
- 10단계: 읽기-쓰기 사용자로 테이블 자르기
- RBAC용 시스템 함수(선택 사항)
- RBAC용 시스템 보기(선택 사항)
- RBAC와 함께 행 수준 보안 사용(선택 사항)

사전 조건

- TICKIT 샘플 데이터베이스와 함께 로드되는 Amazon Redshift 클러스터 또는 서버리스 작업 그룹을 생성합니다. Serverless 작업 그룹을 만들려면 Amazon Redshift Serverless를 참조하세요. 클러스터를 생성하려면 샘플 Amazon Redshift 클러스터 생성을 참조하세요. TICKIT 샘플 데이터베이스에 대한 자세한 내용은 샘플 데이터베이스를 참조하세요.
- 슈퍼 사용자 또는 역할 관리자 권한을 가진 사용자에 대한 액세스를 확보합니다. 슈퍼 사용자 또는 역할 관리자만 역할을 부여하거나 취소할 수 있습니다. RBAC에 필요한 권한에 대한 자세한 내용은 RBAC에 대한 시스템 권한 섹션을 참조하세요.
- RBAC에서 역할 사용에 대한 고려 사항을(를) 검토합니다.

1단계: 관리 사용자 생성

이 자습서를 진행하기 위해 설정하려면 이 단계에서 데이터베이스 관리자 역할을 만들어 데이터베이스 관리 사용자에게 연결합니다. 슈퍼 사용자 또는 역할 관리자로서 데이터베이스 관리자를 생성해야 합니다.

Amazon Redshift 쿼리 에디터 v2에서 모든 쿼리를 실행합니다.

1. 관리자 역할 db_admin을 생성하려면 다음 예시를 사용하세요.

```
CREATE ROLE db_admin;
```

2. dbadmin이라는 데이터베이스 사용자를 생성하려면 다음 예시를 사용하세요.

```
CREATE USER dbadmin PASSWORD 'Test12345';
```

3. sys:dba라는 시스템 정의 역할을 db_admin 역할에 부여하려면 다음 예시를 사용하세요. sys:dba 역할이 부여되면 dbadmin 사용자는 스키마와 테이블을 생성할 수 있습니다. 자세한 내용은 Amazon Redshift 시스템 정의 역할 단원을 참조하십시오.

2단계: 스키마 설정

이 단계에서는 데이터베이스 관리자로 데이터베이스에 연결합니다. 그런 다음 두 개의 스키마를 만들고 여기에 데이터를 추가합니다.

1. 쿼리 에디터 v2를 사용하여 dbadmin 사용자로 dev 데이터베이스에 연결합니다. 데이터베이스에 연결하는 방법에 관한 자세한 내용은 쿼리 에디터 v2 작업을 참조하세요.
2. 영업 및 마케팅 데이터베이스 스키마를 생성하려면 다음 예시를 사용하세요.

```
CREATE SCHEMA sales;
CREATE SCHEMA marketing;
```

3. 영업 스키마의 테이블을 생성하고 값을 삽입하려면 다음 예시를 사용하세요.

```
CREATE TABLE sales.cat(
catid smallint,
catgroup varchar(10),
catname varchar(10),
catdesc varchar(50)
);
INSERT INTO sales.cat(SELECT * FROM category);

CREATE TABLE sales.dates(
dateid smallint,
caldate date,
day char(3),
week smallint,
month char(5),
qtr char(5),
year smallint,
holiday boolean
);
INSERT INTO sales.dates(SELECT * FROM date);
```

```
CREATE TABLE sales.events(
eventid integer,
venueid smallint,
catid smallint,
dateid smallint,
eventname varchar(200),
starttime timestamp
);
INSERT INTO sales.events(SELECT * FROM event);

 CREATE TABLE sales.sale(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp
);
INSERT INTO sales.sale(SELECT * FROM sales);
```

4. 마케팅 스키마의 테이블을 생성하고 값을 삽입하려면 다음 예시를 사용하세요.

```
CREATE TABLE marketing.cat(
catid smallint,
catgroup varchar(10),
catname varchar(10),
catdesc varchar(50)
);
INSERT INTO marketing.cat(SELECT * FROM category);

CREATE TABLE marketing.dates(
dateid smallint,
caldate date,
day char(3),
week smallint,
month char(5),
qtr char(5),
year smallint,
holiday boolean
```

```
);
INSERT INTO marketing.dates(SELECT * FROM date);

CREATE TABLE marketing.events(
eventid integer,
venueid smallint,
catid smallint,
dateid smallint,
eventname varchar(200),
starttime timestamp
);
INSERT INTO marketing.events(SELECT * FROM event);

CREATE TABLE marketing.sale(
marketingid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp
);
INSERT INTO marketing.sale(SELECT * FROM marketing);
```

3단계: 읽기 전용 사용자 생성

이 단계에서는 읽기 전용 역할과 읽기 전용 역할에 대한 salesanalyst 사용자를 생성합니다. 영업 분석가는 커미션이 가장 큰 이벤트를 찾는 임무를 수행하기 위해 영업 스키마의 테이블에 대한 읽기 전용 액세스만 있으면 됩니다.

1. dbadmin 사용자로 데이터베이스에 연결합니다.
2. sales_ro 역할을 생성하려면 다음 예시를 사용하세요.

```
CREATE ROLE sales_ro;
```

3. salesanalyst 사용자를 생성하려면 다음 예시를 사용하세요.

```
CREATE USER salesanalyst PASSWORD 'Test12345';
```

4. sales_ro 역할 사용과 영업 스키마의 객체에 대한 선택 액세스를 허용하려면 다음 예시를 사용하세요.

```
GRANT USAGE ON SCHEMA sales TO ROLE sales_ro;
GRANT SELECT ON ALL TABLES IN SCHEMA sales TO ROLE sales_ro;
```

5. salesanalyst 사용자에게 sales_ro 역할을 부여하려면 다음 예시를 사용하세요.

```
GRANT ROLE sales_ro TO salesanalyst;
```

4단계: 읽기 전용 사용자로 데이터 쿼리

이 단계에서 salesanalyst 사용자는 영업 스키마의 데이터를 쿼리합니다. 그런 다음 salesanalyst 사용자는 테이블 업데이트를 시도하고 마케팅 스키마의 테이블을 읽으려고 합니다.

1. salesanalyst 사용자로 데이터베이스에 연결합니다.
2. 커미션이 가장 큰 10개의 영업을 찾으려면 다음 예시를 사용하세요.

```
SET SEARCH_PATH TO sales;
SELECT DISTINCT events.dateid, sale.commission, cat.catname
FROM sale, events, dates, cat
WHERE events.dateid=dates.dateid AND events.dateid=sale.dateid AND events.catid =
 cat.catid
ORDER BY 2 DESC LIMIT 10;

+--------+------------+----------+
| dateid | commission | catname  |
+--------+------------+----------+
|   1880 |     1893.6 | Pop      |
|   1880 |     1893.6 | Opera    |
|   1880 |     1893.6 | Plays    |
|   1880 |     1893.6 | Musicals |
|   1861 |       1500 | Plays    |
|   2003 |       1500 | Pop      |
|   1861 |       1500 | Opera    |
|   2003 |       1500 | Plays    |
|   1861 |       1500 | Musicals |
|   1861 |       1500 | Pop      |
+--------+------------+----------+
```

3. 영업 스키마의 이벤트 테이블에서 10개의 이벤트를 선택하려면 다음 예시를 사용하세요.

```
SELECT * FROM sales.events LIMIT 10;

+---------+---------+-------+--------+--------------------+---------------------+
| eventid | venueid | catid | dateid |     eventname      |      starttime      |
+---------+---------+-------+--------+--------------------+---------------------+
|    4836 |      73 |     9 |   1871 | Soulfest           | 2008-02-14 19:30:00 |
|    5739 |      41 |     9 |   1871 | Fab Faux           | 2008-02-14 19:30:00 |
|     627 |     229 |     6 |   1872 | High Society       | 2008-02-15 14:00:00 |
|    2563 |     246 |     7 |   1872 | Hamlet             | 2008-02-15 20:00:00 |
|    7703 |      78 |     9 |   1872 | Feist              | 2008-02-15 14:00:00 |
|    7903 |      90 |     9 |   1872 | Little Big Town    | 2008-02-15 19:30:00 |
|    7925 |     101 |     9 |   1872 | Spoon              | 2008-02-15 19:00:00 |
|    8113 |      17 |     9 |   1872 | Santana            | 2008-02-15 15:00:00 |
|     463 |     303 |     8 |   1873 | Tristan und Isolde | 2008-02-16 19:00:00 |
|     613 |     236 |     6 |   1873 | Pal Joey           | 2008-02-16 15:00:00 |
+---------+---------+-------+--------+--------------------+---------------------+
```

4. eventid 1의 eventname을 업데이트하려고 시도하려면 다음 예시를 실행하세요. 이 예시는 권한 거부됨 오류를 반환합니다. salesanalyst 사용자에게 영업 스키마의 이벤트 테이블에 대한 SELECT 권한만 있기 때문입니다. 이벤트 테이블을 업데이트하려면 sales_ro 역할에 UPDATE 권한을 부여해야 합니다. 테이블 업데이트 권한 부여에 대한 자세한 내용은 GRANT의 UPDATE 파라미터를 참조하세요. UPDATE 명령에 대한 자세한 내용은 UPDATE 섹션을 참조하세요.

```
UPDATE sales.events
SET eventname = 'Comment event'
WHERE eventid = 1;

ERROR: permission denied for relation events
```

5. 마케팅 스키마의 모든 이벤트 테이블을 선택하려고 시도하려면 다음 예시를 사용하세요. 이 예시는 권한 거부됨 오류를 반환합니다. salesanalyst 사용자에게 영업 스키마의 이벤트 테이블에 대한 SELECT 권한만 있기 때문입니다. 마케팅 스키마의 이벤트 테이블에서 데이터를 선택하려면 sales_ro 역할에 마케팅 스키마의 이벤트 테이블에 대한 SELECT 권한을 부여해야 합니다.

```
SELECT * FROM marketing.events;

                ERROR: permission denied for schema marketing
```

5단계: 읽기-쓰기 사용자 생성

이 단계에서는 영업 스키마의 데이터 처리를 위한 추출, 전환, 적재(ETL) 파이프라인 구축을 담당하는 영업 엔지니어에게 읽기 전용 액세스 권한을 부여하지만 나중에는 업무를 수행할 수 있도록 읽기 및 쓰기 액세스 권한을 부여합니다.

1. dbadmin 사용자로 데이터베이스에 연결합니다.
2. 영업 스키마에서 sales_rw 역할을 생성하려면 다음 예시를 사용하세요.

```
CREATE ROLE sales_rw;
```

3. salesengineer 사용자를 생성하려면 다음 예시를 사용하세요.

```
CREATE USER salesengineer PASSWORD 'Test12345';
```

4. sales-ro 역할을 할당하여 sales_rw 역할 사용과 영업 스키마의 객체에 대한 선택 액세스를 허용하려면 다음 예시를 사용하세요. Amazon Redshift에서 역할이 권한을 상속하는 방법에 대한 자세한 내용은 역할 계층 구조 섹션을 참조하세요.

```
GRANT ROLE sales_ro TO ROLE sales_rw;
```

5. salesengineer 사용자에게 sales_rw 역할을 할당하려면 다음 예시를 사용하세요.

```
GRANT ROLE sales_rw TO salesengineer;
```

6단계: 상속한 읽기 전용 역할을 가진 사용자로 데이터 쿼리

이 단계에서 salesengineer 사용자는 읽기 권한을 부여받기 전에 이벤트 테이블을 업데이트하려고 합니다.

1. salesengineer 사용자로 데이터베이스에 연결합니다.
2. salesengineer 사용자는 영업 스키마의 이벤트 테이블에서 데이터를 성공적으로 읽을 수 있습니다. 영업 스키마의 이벤트 테이블에서 eventid가 1인 이벤트를 선택하려면 다음 예시를 사용하세요.

```
SELECT * FROM sales.events where eventid=1;

+---------+---------+-------+--------+----------------+---------------------+
| eventid | venueid | catid | dateid |    eventname   |      starttime      |
+---------+---------+-------+--------+----------------+---------------------+
```

```
|       1 |     305 |     8 |   1851 | Gotterdammerung | 2008-01-25 14:30:00 |
+---------+---------+-------+--------+-----------------+---------------------+
```

3. 마케팅 스키마의 모든 이벤트 테이블을 선택하려고 시도하려면 다음 예시를 사용하세요. salesengineer 사용자에게는 마케팅 스키마의 테이블에 대한 권한이 없으므로 이 쿼리는 권한 거부됨 오류를 반환합니다. 마케팅 스키마의 이벤트 테이블에서 데이터를 선택하려면 sales_rw 역할에 마케팅 스키마의 이벤트 테이블에 대한 SELECT 권한을 부여해야 합니다.

```
SELECT * FROM marketing.events;

ERROR: permission denied for schema marketing
```

4. eventid 1의 eventname을 업데이트하려고 시도하려면 다음 예시를 실행하세요. 이 예시는 권한 거부됨 오류를 반환합니다. salesengineer 사용자에게 영업 스키마의 이벤트 테이블에 대한 선택 권한만 있기 때문입니다. 이벤트 테이블을 업데이트하려면 sales_rw 역할에 UPDATE 권한을 부여해야 합니다.

```
UPDATE sales.events
SET eventname = 'Comment event'
WHERE eventid = 1;

ERROR: permission denied for relation events
```

7단계: 읽기-쓰기 역할에 업데이트 및 삽입 권한 부여

이 단계에서는 sales_rw 역할에 업데이트 및 삽입 권한을 부여합니다.

1. dbadmin 사용자로 데이터베이스에 연결합니다.
2. sales_rw 역할에 UPDATE, INSERT, DELETE 권한을 부여하려면 다음 예시를 사용하세요.

```
GRANT UPDATE, INSERT, ON ALL TABLES IN SCHEMA sales TO role sales_rw;
```

8단계: 읽기-쓰기 사용자로 데이터 쿼리

이 단계에서는 salesengineer의 역할에 삽입 및 업데이트 권한이 부여된 후 salesengineer가 테이블을 성공적으로 업데이트합니다. 그런 다음 salesengineer는 이벤트 테이블을 분석하고 정리하려고 시도하지만 실패합니다.

1. salesengineer 사용자로 데이터베이스에 연결합니다.
2. eventid 1의 eventname을 업데이트하려면 다음 예시를 실행하세요.

```
UPDATE sales.events
SET eventname = 'Comment event'
WHERE eventid = 1;
```

3. 이전 쿼리에서 변경한 내용을 보려면 다음 예시를 사용하여 영업 스키마의 이벤트 테이블에서 eventid가 1인 이벤트를 선택하세요.

```
SELECT * FROM sales.events WHERE eventid=1;

+---------+---------+-------+--------+--------------+---------------------+
| eventid | venueid | catid | dateid |   eventname  |      starttime      |
+---------+---------+-------+--------+--------------+---------------------+
|       1 |     305 |     8 |   1851 | Comment event | 2008-01-25 14:30:00 |
+---------+---------+-------+--------+--------------+---------------------+
```

4. 영업 스키마의 업데이트된 이벤트 테이블을 분석하려면 다음 예시를 사용하세요. 이 예시는 권한 거부됨 오류를 반환합니다. salesengineer 사용자에게 필요한 권한이 없으며 salesengineer는 영업 스키마에 있는 이벤트 테이블의 소유자가 아니기 때문입니다. 이벤트 테이블을 분석하려면 GRANT 명령을 사용하여 sales_rw 역할에 ANALYZE 권한을 부여해야 합니다. ANALYZE 명령에 대한 자세한 내용은 ANALYZE 섹션을 참조하세요.

```
ANALYZE sales.events;

               ERROR: skipping "events" --- only table or database owner can analyze
```

5. 업데이트된 이벤트 테이블을 정리하려면 다음 예시를 사용하세요. 이 예시는 권한 거부됨 오류를 반환합니다. salesengineer 사용자에게 필요한 권한이 없으며 salesengineer는 영업 스키마에 있는 이벤트 테이블의 소유자가 아니기 때문입니다. 이벤트 테이블을 정리하려면 GRANT 명령을 사용하여 sales_rw 역할에 VACUUM 권한을 부여해야 합니다. VACUUM 명령에 대한 자세한 내용은 VACUUM 섹션을 참조하세요.

```
VACUUM sales.events;

ERROR: skipping "events" --- only table or database owner can vacuum it
```

9단계: 관리 사용자로 데이터베이스의 테이블 분석 및 정리

이 단계에서 dbadmin 사용자는 모든 테이블을 분석하고 정리합니다. 이 사용자는 이 데이터베이스에 대한 관리자 권한을 가지고 있으므로 이러한 명령을 실행할 수 있습니다.

1. dbadmin 사용자로 데이터베이스에 연결합니다.
2. 영업 스키마의 이벤트 테이블을 분석하려면 다음 예시를 사용하세요.

```
ANALYZE sales.events;
```

3. 영업 스키마의 이벤트 테이블을 정리하려면 다음 예시를 사용하세요.

```
VACUUM sales.events;
```

4. 마케팅 스키마의 이벤트 테이블을 분석하려면 다음 예시를 사용하세요.

```
ANALYZE marketing.events;
```

5. 마케팅 스키마의 이벤트 테이블을 정리하려면 다음 예시를 사용하세요.

```
VACUUM marketing.events;
```

10단계: 읽기-쓰기 사용자로 테이블 자르기

이 단계에서 salesengineer 사용자는 영업 스키마의 이벤트 테이블을 자르려고 시도하지만 dbadmin 사용자가 자르기 권한을 부여한 경우에만 성공합니다.

1. salesengineer 사용자로 데이터베이스에 연결합니다.
2. 영업 스키마의 이벤트 테이블에서 모든 행을 삭제하려면 다음 예시를 사용하세요. 이 예시는 오류를 반환합니다. salesengineer 사용자에게 필요한 권한이 없으며 salesengineer는 영업 스키마에 있는 이벤트 테이블의 소유자가 아니기 때문입니다. 이벤트 테이블을 자르려면 GRANT 명령을 사용하여 sales_rw 역할에 TRUNCATE 권한을 부여해야 합니다. TRUNCATE 명령에 대한 자세한 내용은 TRUNCATE 섹션을 참조하세요.

```
TRUNCATE sales.events;

ERROR: must be owner of relation events
```

3. dbadmin 사용자로 데이터베이스에 연결합니다.

4. sales_rw 역할에 테이블 자르기 권한을 부여하려면 다음 예시를 사용하세요.

```
GRANT TRUNCATE TABLE TO role sales_rw;
```

5. 쿼리 에디터 v2를 사용하여 salesengineer 사용자로 데이터베이스에 연결합니다.
6. 영업 스키마의 이벤트 테이블에서 처음 10개의 이벤트를 읽으려면 다음 예시를 사용하세요.

```
SELECT * FROM sales.events ORDER BY eventid LIMIT 10;

+---------+---------+-------+--------+----------------------------
+---------------------+
| eventid | venueid | catid | dateid |          eventname          |      starttime
    |
+---------+---------+-------+--------+----------------------------
+---------------------+
|       1 |     305 |     8 |   1851 | Comment event               | 2008-01-25
 14:30:00 |
|       2 |     306 |     8 |   2114 | Boris Godunov               | 2008-10-15
 20:00:00 |
|       3 |     302 |     8 |   1935 | Salome                      | 2008-04-19
 14:30:00 |
|       4 |     309 |     8 |   2090 | La Cenerentola (Cinderella) | 2008-09-21
 14:30:00 |
|       5 |     302 |     8 |   1982 | Il Trovatore                | 2008-06-05
 19:00:00 |
|       6 |     308 |     8 |   2109 | L Elisir d Amore            | 2008-10-10
 19:30:00 |
|       7 |     309 |     8 |   1891 | Doctor Atomic               | 2008-03-06
 14:00:00 |
|       8 |     302 |     8 |   1832 | The Magic Flute             | 2008-01-06
 20:00:00 |
|       9 |     308 |     8 |   2087 | The Fly                     | 2008-09-18
 19:30:00 |
|      10 |     305 |     8 |   2079 | Rigoletto                   | 2008-09-10
 15:00:00 |
+---------+---------+-------+--------+----------------------------
+---------------------+
```

7. 영업 스키마의 이벤트 테이블을 자르려면 다음 예시를 사용하세요.

```
TRUNCATE sales.events;
```

8. 영업 스키마의 업데이트된 이벤트 테이블에서 데이터를 읽으려면 다음 예시를 사용하세요.

```
SELECT * FROM sales.events ORDER BY eventid LIMIT 10;

+---------+---------+-------+--------+----------------------------
+---------------------+
| eventid | venueid | catid | dateid |          eventname          |      starttime
    |
+---------+---------+-------+--------+----------------------------
+---------------------+
```

마케팅 스키마에 대한 읽기 전용 및 읽기-쓰기 역할 생성(선택 사항)

이 단계에서는 마케팅 스키마에 대한 읽기 전용 및 읽기-쓰기 역할을 생성합니다.

1. dbadmin 사용자로 데이터베이스에 연결합니다.
2. 마케팅 스키마에 대한 읽기 전용 및 읽기-쓰기 역할을 생성하려면 다음 예시를 사용하세요.

```
CREATE ROLE marketing_ro;

CREATE ROLE marketing_rw;

GRANT USAGE ON SCHEMA marketing TO ROLE marketing_ro, ROLE marketing_rw;

GRANT SELECT ON ALL TABLES IN SCHEMA marketing TO ROLE marketing_ro;

GRANT ROLE marketing_ro TO ROLE marketing_rw;

GRANT INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA marketing TO ROLE marketing_rw;

CREATE USER marketinganalyst PASSWORD 'Test12345';

CREATE USER marketingengineer PASSWORD 'Test12345';

GRANT ROLE marketing_ro TO marketinganalyst;

GRANT ROLE marketing_rw TO marketingengineer;
```

RBAC용 시스템 함수(선택 사항)

Amazon Redshift에는 추가 그룹 또는 역할의 사용자 멤버십 및 역할 멤버십에 대한 시스템 정보를 제공하는 두 가지 함수가 있습니다. 바로 role_is_member_of와 user_is_member_of입니다. 이러한 함수는 슈퍼 사용자와 일반 사용자가 사용할 수 있습니다. 슈퍼 사용자는 모든 역할 멤버십을 확인할 수 있습니다. 일반 사용자는 자신에게 액세스 권한이 부여된 역할의 멤버십만 확인할 수 있습니다.

role_is_member_of 함수를 사용하는 방법

1. salesengineer 사용자로 데이터베이스에 연결합니다.
2. sales_rw 역할이 sales_ro 역할의 멤버인지 확인하려면 다음 예시를 사용하세요.

```
SELECT role_is_member_of('sales_rw', 'sales_ro');

+-------------------+
| role_is_member_of |
+-------------------+
| true              |
+-------------------+
```

3. sales_ro 역할이 sales_rw 역할의 멤버인지 확인하려면 다음 예시를 사용하세요.

```
SELECT role_is_member_of('sales_ro', 'sales_rw');

+-------------------+
| role_is_member_of |
+-------------------+
| false             |
+-------------------+
```

user_is_member_of 함수를 사용하는 방법

1. salesengineer 사용자로 데이터베이스에 연결합니다.
2. 다음 예시에서는 salesanalyst 사용자의 사용자 멤버십을 확인하려고 시도합니다. 이 쿼리는 오류를 반환합니다. salesengineer에게 salesanalyst에 대한 액세스 권한이 없기 때문입니다. 이 명령을 성공적으로 실행하려면 salesanalyst 사용자로 데이터베이스에 연결하고 예시를 사용하세요.

```
SELECT user_is_member_of('salesanalyst', 'sales_ro');
```

```
ERROR
```

3. 슈퍼 사용자로 데이터베이스에 연결합니다.
4. 슈퍼 사용자로 연결했을 때 salesanalyst 사용자의 멤버십을 확인하려면 다음 예시를 사용하세요.

```
SELECT user_is_member_of('salesanalyst', 'sales_ro');

+-------------------+
| user_is_member_of |
+-------------------+
| true              |
+-------------------+
```

5. dbadmin 사용자로 데이터베이스에 연결합니다.
6. salesengineer 사용자의 멤버십을 확인하려면 다음 예시를 사용하세요.

```
SELECT user_is_member_of('salesengineer', 'sales_ro');

+-------------------+
| user_is_member_of |
+-------------------+
| true              |
+-------------------+

SELECT user_is_member_of('salesengineer', 'marketing_ro');

+-------------------+
| user_is_member_of |
+-------------------+
| false             |
+-------------------+

SELECT user_is_member_of('marketinganalyst', 'sales_ro');

+-------------------+
| user_is_member_of |
+-------------------+
| false             |
+-------------------+
```

RBAC용 시스템 보기(선택 사항)

역할, 사용자에 대한 역할 할당, 역할 계층 구조 및 역할을 통한 데이터베이스 객체 권한을 보려면 Amazon Redshift의 시스템 보기를 사용하세요. 이러한 보기는 슈퍼 사용자와 일반 사용자가 사용할 수 있습니다. 슈퍼 사용자는 모든 역할 세부 정보를 확인할 수 있습니다. 일반 사용자는 자신에게 액세스 권한이 부여된 역할의 세부 정보만 확인할 수 있습니다.

1. 클러스터에서 명시적으로 역할이 부여된 사용자 목록을 보려면 다음 예시를 사용하세요.

```
SELECT * FROM svv_user_grants;
```

2. 클러스터에서 명시적으로 역할이 부여된 역할 목록을 보려면 다음 예시를 사용하세요.

```
SELECT * FROM svv_role_grants;
```

시스템 보기의 전체 목록은 SVV 메타데이터 뷰 섹션을 참조하세요.

RBAC와 함께 행 수준 보안 사용(선택 사항)

민감한 데이터에 대한 액세스를 세부적으로 제어하려면 행 수준 보안(RLS)을 사용하세요. RLS에 대한 자세한 내용은 행 수준 보안 섹션을 참조하세요.

이 섹션에서는 Major League Baseball의 `catdesc` 값이 있는 `cat` 테이블의 행만 볼 수 있는 권한을 `salesengineer` 사용자에게 부여하는 RLS 정책을 생성합니다. 그런 다음 `salesengineer` 사용자로 데이터베이스를 쿼리합니다.

1. `salesengineer` 사용자로 데이터베이스에 연결합니다.
2. `cat` 테이블의 처음 5개 항목을 보려면 다음 예시를 사용하세요.

```
SELECT *
FROM sales.cat
ORDER BY catid ASC
LIMIT 5;

+-------+----------+---------+----------------------------------+
| catid | catgroup | catname |             catdesc              |
+-------+----------+---------+----------------------------------+
|     1 | Sports   | MLB     | Major League Baseball            |
|     2 | Sports   | NHL     | National Hockey League           |
|     3 | Sports   | NFL     | National Football League         |
```

```
|     4 | Sports   | NBA     | National Basketball Association |
|     5 | Sports   | MLS     | Major League Soccer             |
+-------+----------+---------+---------------------------------+
```

3. dbadmin 사용자로 데이터베이스에 연결합니다.
4. cat 테이블의 catdesc 열에 대한 RLS 정책을 생성하려면 다음 예시를 사용하세요.

```
CREATE RLS POLICY policy_mlb_engineer
WITH (catdesc VARCHAR(50))
USING (catdesc = 'Major League Baseball');
```

5. RLS 정책을 sales_rw 역할에 연결하려면 다음 예시를 사용하세요.

```
ATTACH RLS POLICY policy_mlb_engineer ON sales.cat TO ROLE sales_rw;
```

6. RLS를 활성화하도록 테이블을 변경하려면 다음 예시를 사용하세요.

```
ALTER TABLE sales.cat ROW LEVEL SECURITY ON;
```

7. salesengineer 사용자로 데이터베이스에 연결합니다.
8. cat 테이블의 처음 5개 항목을 보려고 시도하려면 다음 예시를 사용하세요. catdesc 열이 Major League Baseball일 때만 항목이 표시된다는 점을 유의하세요.

```
SELECT *
FROM sales.cat
ORDER BY catid ASC
LIMIT 5;

+-------+----------+---------+----------------------+
| catid | catgroup | catname |        catdesc       |
+-------+----------+---------+----------------------+
|     1 | Sports   | MLB     | Major League Baseball |
+-------+----------+---------+----------------------+
```

9. salesanalyst 사용자로 데이터베이스에 연결합니다.
10. cat 테이블의 처음 5개 항목을 보려고 시도하려면 다음 예시를 사용하세요. 기본값인 모두 거부 정책이 적용되므로 항목이 표시되지 않는다는 점을 유의하세요.

```
SELECT *
FROM sales.cat
ORDER BY catid ASC
```

```
LIMIT 5;

+-------+----------+---------+----------------------+
| catid | catgroup | catname |        catdesc       |
+-------+----------+---------+----------------------+
```

11. dbadmin 사용자로 데이터베이스에 연결합니다.
12. sales_ro 역할에 IGNORE RLS 권한을 부여하려면 다음 예시를 사용하세요. 이렇게 하면 salesanalyst 사용자가 sales_ro 역할의 구성원이므로 RLS 정책을 무시할 수 있는 권한이 부여됩니다.

```
GRANT IGNORE RLS TO ROLE sales_ro;
```

13. salesanalyst 사용자로 데이터베이스에 연결합니다.
14. cat 테이블의 처음 5개 항목을 보려면 다음 예시를 사용하세요.

```
SELECT *
FROM sales.cat
ORDER BY catid ASC
LIMIT 5;

+-------+----------+---------+---------------------------------+
| catid | catgroup | catname |             catdesc             |
+-------+----------+---------+---------------------------------+
|     1 | Sports   | MLB     | Major League Baseball           |
|     2 | Sports   | NHL     | National Hockey League          |
|     3 | Sports   | NFL     | National Football League        |
|     4 | Sports   | NBA     | National Basketball Association |
|     5 | Sports   | MLS     | Major League Soccer             |
+-------+----------+---------+---------------------------------+
```

15. dbadmin 사용자로 데이터베이스에 연결합니다.
16. sales_ro 역할에서 IGNORE RLS 권한을 취소하려면 다음 예시를 사용하세요.

```
REVOKE IGNORE RLS FROM ROLE sales_ro;
```

17. salesanalyst 사용자로 데이터베이스에 연결합니다.
18. cat 테이블의 처음 5개 항목을 보려고 시도하려면 다음 예시를 사용하세요. 기본값인 모두 거부 정책이 적용되므로 항목이 표시되지 않는다는 점을 유의하세요.

```
SELECT *
```

```
FROM sales.cat
ORDER BY catid ASC
LIMIT 5;

+-------+----------+---------+----------------------+
| catid | catgroup | catname |        catdesc       |
+-------+----------+---------+----------------------+
```

19. dbadmin 사용자로 데이터베이스에 연결합니다.

20. cat 테이블에서 RLS 정책을 분리하려면 다음 예시를 사용하세요.

```
DETACH RLS POLICY policy_mlb_engineer ON cat FROM ROLE sales_rw;
```

21. salesanalyst 사용자로 데이터베이스에 연결합니다.

22. cat 테이블의 처음 5개 항목을 보려고 시도하려면 다음 예시를 사용하세요. 기본값인 모두 거부 정책이 적용되므로 항목이 표시되지 않는다는 점을 유의하세요.

```
SELECT *
FROM sales.cat
ORDER BY catid ASC
LIMIT 5;

+-------+----------+---------+---------------------------------+
| catid | catgroup | catname |             catdesc             |
+-------+----------+---------+---------------------------------+
|     1 | Sports   | MLB     | Major League Baseball           |
|     2 | Sports   | NHL     | National Hockey League          |
|     3 | Sports   | NFL     | National Football League        |
|     4 | Sports   | NBA     | National Basketball Association |
|     5 | Sports   | MLS     | Major League Soccer             |
+-------+----------+---------+---------------------------------+
```

23. dbadmin 사용자로 데이터베이스에 연결합니다.

24. RLS 정책을 삭제하려면 다음 예시를 사용하세요.

```
DROP RLS POLICY policy_mlb_engineer;
```

25. RLS를 제거하려면 다음 예시를 사용하세요.

```
ALTER TABLE cat ROW LEVEL SECURITY OFF;
```

관련 주제

RBAC에 대한 자세한 내용은 다음 설명서를 참조하세요.

- 역할 계층 구조
- 역할 할당
- 데이터베이스 객체 권한
- RBAC에 대한 ALTER DEFAULT PRIVILEGES

행 수준 보안

Amazon Redshift의 행 수준 보안(RLS)을 사용하면 민감한 데이터에 대한 액세스를 세부적으로 제어할 수 있습니다. 데이터베이스 객체 수준에서 정의된 보안 정책에 따라, 스키마 또는 테이블 내의 특정 데이터 레코드에 액세스할 수 있는 사용자나 역할을 결정할 수 있습니다. 사용자에게 열 하위 집합에 대한 권한을 부여할 수 있는 열 수준 보안 외에도, RLS 정책을 사용하여 표시되는 열의 특정 행에 대한 액세스를 추가로 제한합니다. 열 수준 보안에 대한 자세한 내용은 열 수준 액세스 제어 사용 시 주의 사항 섹션을 참조하세요.

테이블에 RLS 정책을 적용할 때 쿼리 실행 시에 반환되는 결과 집합을 제한할 수 있습니다.

RLS 정책을 생성할 때 Amazon Redshift Redshift가 쿼리에서 테이블의 기존 행을 반환할지 여부를 결정하는 표현식을 지정할 수 있습니다. 액세스를 제한하는 RLS 정책을 만들면 쿼리에 조건을 추가하거나 추가 조건을 외부화할 필요가 없습니다.

RLS 정책을 만들 때는 정책을 간단하게 만들고 정책에 복잡한 문은 사용하지 않는 것이 좋습니다. RLS 정책을 정의할 때 정책을 기반으로 하는 테이블 조인을 정책 정의에 과도하게 사용하지 마세요.

정책이 조회 테이블을 참조하는 경우 Amazon Redshift는 정책이 존재하는 테이블 외에도 추가 테이블을 스캔합니다. RLS 정책이 연결된 사용자와 정책이 연결되지 않은 사용자에 대한 동일한 쿼리 간에 성능 차이가 나타날 수 있습니다.

SQL 문에 RLS 정책 사용

SQL 문에 RLS 정책을 사용할 때 Amazon Redshift는 다음 규칙을 적용합니다.

- Amazon Redshift는 기본적으로 SELECT, UPDATE 및 DELETE 문에 RLS 정책을 적용합니다.
- SELECT 및 UNLOAD의 경우 Amazon Redshift는 정의된 정책에 따라 행을 필터링합니다.

- UPDATE의 경우 Amazon Redshift는 사용자에게 표시되는 행만 업데이트합니다. 정책이 테이블 행의 하위 집합을 제한하는 경우 해당 행을 업데이트할 수 없습니다.
- DELETE의 경우 표시되는 행만 삭제할 수 있습니다. 정책이 테이블 행의 하위 집합을 제한하는 경우 해당 행을 삭제할 수 없습니다. TRUNCATE의 경우 테이블을 자를 수 있습니다.
- CREATE TABLE LIKE의 경우 LIKE 옵션을 사용하여 생성된 테이블은 소스 테이블에서 권한 설정을 상속하지 않습니다. 마찬가지로, 대상 테이블은 소스 테이블에서 RLS 정책을 상속하지 않습니다.

사용자별로 여러 정책 결합

Amazon Redshift에서 RLS는 사용자 및 객체별로 여러 정책을 연결하는 기능을 지원합니다. 한 사용자에 대해 여러 정책이 정의되어 있는 경우, Amazon Redshift는 테이블에 대한 RLS CONJUNCTION TYPE 설정에 따라 AND 또는 OR 구문을 사용하여 모든 정책을 적용합니다. 접속사 유형에 대한 자세한 정보는 ALTER TABLE 섹션을 참고하세요.

테이블에 대한 여러 정책이 사용자에게 연결될 수 있습니다. 사용자에게 여러 정책이 직접 연결되어 있거나 사용자가 여러 역할에 속해 있으며, 역할별로 서로 다른 정책이 연결되어 있습니다.

주어진 관계에서 여러 정책이 행 액세스를 제한해야 하는 경우 관계의 RLS CONJUNCTION TYPE을 AND로 설정할 수 있습니다. 다음 예제를 살펴보세요. Alice는 지정된 정책으로 'catname'이 NBA인 Sports 이벤트만 볼 수 있습니다.

```
-- Create an analyst role and grant it to a user named Alice.
CREATE ROLE analyst;
CREATE USER alice WITH PASSWORD 'Name_is_alice_1';
GRANT ROLE analyst TO alice;

-- Create an RLS policy that only lets the user see sports.
CREATE RLS POLICY policy_sports
WITH (catgroup VARCHAR(10))
USING (catgroup = 'Sports');

-- Create an RLS policy that only lets the user see NBA.
CREATE RLS POLICY policy_nba
WITH (catname VARCHAR(10))
USING (catname = 'NBA');

-- Attach both to the analyst role.
ATTACH RLS POLICY policy_sports ON category TO ROLE analyst;
ATTACH RLS POLICY policy_nba ON category TO ROLE analyst;
```

```
-- Activate RLS on the category table with AND CONJUNCTION TYPE.
ALTER TABLE category ROW LEVEL SECURITY ON CONJUNCTION TYPE AND;

-- Change session to Alice.
SET SESSION AUTHORIZATION alice;

-- Select all from the category table.
SELECT catgroup, catname
FROM category;

 catgroup | catname
---------+---------
 Sports   | NBA
(1 row)
```

주어진 관계에서 여러 정책이 사용자가 더 많은 행을 보도록 허용해야 하는 경우 관계의 RLS CONJUNCTION TYPE을 OR로 설정할 수 있습니다. 다음 예제를 살펴보세요. Alice는 지정된 정책으로 'Concerts'와 'Sports'만 볼 수 있습니다.

```
-- Create an analyst role and grant it to a user named Alice.
CREATE ROLE analyst;
CREATE USER alice WITH PASSWORD 'Name_is_alice_1';
GRANT ROLE analyst TO alice;

-- Create an RLS policy that only lets the user see concerts.
CREATE RLS POLICY policy_concerts
WITH (catgroup VARCHAR(10))
USING (catgroup = 'Concerts');

-- Create an RLS policy that only lets the user see sports.
CREATE RLS POLICY policy_sports
WITH (catgroup VARCHAR(10))
USING (catgroup = 'Sports');

-- Attach both to the analyst role.
ATTACH RLS POLICY policy_concerts ON category TO ROLE analyst;
ATTACH RLS POLICY policy_sports ON category TO ROLE analyst;

-- Activate RLS on the category table with OR CONJUNCTION TYPE.
ALTER TABLE category ROW LEVEL SECURITY ON CONJUNCTION TYPE OR;

-- Change session to Alice.
SET SESSION AUTHORIZATION alice;
```

```
-- Select all from the category table.
SELECT catgroup, count(*)
FROM category
GROUP BY catgroup ORDER BY catgroup;

 catgroup | count
---------+-------
 Concerts |  3
 Sports   |  5
(2 rows)
```

RLS 정책 소유권 및 관리

슈퍼 사용자, 보안 관리자 또는 sys:secadmin 역할이 부여된 사용자는 RLS 정책을 만들고 수정하고 연결하고 분리할 수 있습니다. RLS 정책은 테이블, 뷰, 지연 바인딩 뷰(LBV) 및 구체화된 뷰(MV)에 연결될 수 있습니다. 테이블의 스키마 정의를 수정하지 않고, 객체 수준에서 행 수준 보안을 설정하거나 해제할 수 있습니다.

행 수준 보안을 시작하기 위해 사용할 수 있는 SQL 문은 다음과 같습니다.

- ALTER TABLE 문을 사용하여 테이블, 뷰 또는 지연 바인딩 뷰에 대해 RLS를 설정하거나 해제합니다. 자세한 내용은 ALTER TABLE 단원을 참조하십시오.
- ALTER MATERIALIZED VIEW 문을 사용하여 구체화된 뷰(MV)에서 RLS를 설정하거나 해제합니다. 자세한 내용은 ALTER MATERIALIZED VIEW 단원을 참조하십시오.
- CREATE RLS POLICY 문을 사용하여 하나 이상의 테이블에 대한 보안 정책을 생성하고, 정책에서 하나 이상의 사용자 또는 역할을 지정합니다.

  자세한 내용은 CREATE RLS POLICY 단원을 참조하십시오.
- ALTER RLS POLICY 명령문을 사용하여 정책 정의 변경과 같은 정책을 변경할 수 있습니다. 여러 테이블 또는 뷰에 동일한 정책을 사용할 수 있습니다.

  자세한 내용은 ALTER RLS POLICY 단원을 참조하십시오.
- ATTACH RLS POLICY 문을 사용하여 하나 이상의 관계, 하나 이상의 사용자 또는 역할에 정책을 연결합니다.

  자세한 내용은 ATTACH RLS POLICY 단원을 참조하십시오.
- DETACH RLS POLICY 문을 사용하여 하나 이상의 관계, 하나 이상의 사용자 또는 역할에서 정책을 분리합니다.

자세한 내용은 DETACH RLS POLICY 단원을 참조하십시오.

- DROP RLS POLICY 문을 사용하여 정책을 삭제합니다.

  자세한 내용은 DROP RLS POLICY 단원을 참조하십시오.

- GRANT 및 REVOKE 문을 사용하여 조회 테이블을 참조하는 RLS 정책에 SELECT 권한을 명시적으로 부여하고 취소합니다. 자세한 내용은 GRANT 및 REVOKE 섹션을 참조하세요.

sys:secadmin은 SVV_RLS_POLICY 및 SVV_RLS_ATTACHED_POLICY을 통해 생성된 정책을 모니터링할 수 있습니다.

RLS로 보호된 관계를 나열하기 위해 sys:secadmin은 SVV_RLS_RELATION을 볼 수 있습니다.

수퍼유저, sys:operator 또는 ACCESS SYSTEM TABLE 시스템 권한이 있는 모든 사용자는 SVV_RLS_APPLIED_POLICY를 통해 RLS로 보호되는 관계를 참조하는 쿼리에 대한 RLS 정책의 적용을 추적할 수 있습니다. sys:secadmin에게는 기본적으로 이러한 권한이 부여되지 않습니다.

사용자에게 RLS로 보호된 관계에 대한 전체 액세스 권한을 허용하려면 IGNORE RLS 권한을 부여하면 됩니다. 슈퍼 사용자 또는 sys:secadmin에게는 자동으로 IGNORE RLS 권한이 부여됩니다. 자세한 내용은 GRANT 단원을 참조하십시오.

EXPLAIN 계획에서 쿼리의 RLS 정책 필터를 설명하여 RLS 관련 쿼리 문제를 해결하려면, 사용자에게 EXPLAIN RLS 권한을 부여합니다. 자세한 내용은 GRANT 및 EXPLAIN 섹션을 참조하세요.

정책 종속 객체 및 원칙

애플리케이션의 보안을 제공하고 정책 객체가 오래되거나 잘못되는 것을 방지하기 위해, Amazon Redshift는 RLS 정책에서 참조하는 객체를 삭제하거나 변경하는 것을 허용하지 않습니다.

다음은 Amazon Redshift가 RLS 정책에 대해 추적하는 스키마 객체 종속성의 목록입니다.

- 대상 테이블의 스키마 객체 종속성을 추적할 때 Amazon Redshift는 다음 규칙을 따릅니다.
  - Amazon Redshift는 대상 테이블을 삭제할 때 관계, 사용자, 역할 또는 퍼블릭 객체로부터 정책을 분리합니다.
  - 대상 테이블의 이름을 바꾸더라도 연결된 정책에는 영향을 주지 않습니다.
  - 먼저 정책을 삭제하거나 분리하면, 정책 정의 내에서 참조되는 대상 테이블의 열만 삭제할 수 있습니다. 이는 CASCADE 옵션이 지정된 경우에도 적용됩니다. 대상 테이블의 다른 열은 삭제할 수 있습니다.

  - 대상 테이블에서 참조된 열의 이름은 바꿀 수 없습니다. 참조된 열의 이름을 바꾸려면 먼저 정책을 분리합니다. 이는 CASCADE 옵션이 지정된 경우에도 적용됩니다.
  - CASCADE 옵션을 지정하는 경우에도 참조된 열의 유형은 변경할 수 없습니다.
- 조회 테이블의 스키마 객체 종속성을 추적할 때 Amazon Redshift는 다음 규칙을 따릅니다.
  - 조회 테이블은 삭제할 수 없습니다. 조회 테이블을 삭제하려면 조회 테이블을 참조하는 정책을 먼저 삭제합니다.
  - 조회 테이블의 이름을 바꿀 수 없습니다. 조회 테이블의 이름을 바꾸려면 조회 테이블을 참조하는 정책을 먼저 삭제합니다. 이는 CASCADE 옵션이 지정된 경우에도 적용됩니다.
  - 정책 정의에 사용된 조회 테이블 열은 삭제할 수 없습니다. 정책 정의에 사용된 조회 테이블 열을 삭제하려면 조회 테이블을 참조하는 정책을 먼저 삭제합니다. 이는 ALTER TABLE DROP COLUMN 문에 CASCADE 옵션이 지정되어 있는 경우에도 적용됩니다. 조회 테이블의 다른 열은 삭제할 수 있습니다.
  - 조회 테이블에서 참조된 열의 이름은 바꿀 수 없습니다. 참조된 열의 이름을 바꾸려면 조회 테이블을 참조하는 정책을 먼저 삭제합니다. 이는 CASCADE 옵션이 지정된 경우에도 적용됩니다.
  - 참조된 열의 유형은 변경할 수 없습니다.
- 사용자 또는 역할이 삭제되면 Amazon Redshift가 사용자 또는 역할에 연결된 모든 정책을 자동으로 분리합니다.
- DROP SCHEMA 문에 CASCADE 옵션을 사용하면 Amazon Redshift가 스키마의 관계도 삭제합니다. 또한 삭제된 스키마의 관계에 종속된 다른 스키마의 관계도 삭제합니다. 정책의 조회 테이블인 관계의 경우, Amazon Redshift에서 DROP SCHEMA DDL이 실패합니다. DROP SCHEMA 문에 의해 삭제된 관계의 경우, Amazon Redshift는 해당 관계에 연결된 모든 정책을 분리합니다.
- 해당 정책도 삭제하는 경우에만 조회 함수(정책 정의 내에서 참조되는 함수)를 삭제할 수 있습니다. 이는 CASCADE 옵션이 지정된 경우에도 적용됩니다.
- 정책이 테이블에 연결되면 Amazon Redshift가 이 테이블이 다른 정책에서 조회 테이블인지 확인합니다. 이 경우 Amazon Redshift는 이 테이블에 정책을 연결하도록 허용하지 않습니다.
- RLS 정책을 생성할 때 Amazon Redshift는 이 테이블이 다른 RLS 정책의 대상 테이블인지 확인합니다. 이 경우 Amazon Redshift는 이 테이블에 대한 정책을 생성하도록 허용하지 않습니다.

예제

다음 예제는 스키마 종속성이 추적되는 방법을 보여 줍니다.

```
-- The CREATE and ATTACH policy statements for `policy_events` references some
-- target and lookup tables.
```

```
-- Target tables are tickit_event_redshift and target_schema.target_event_table.
-- Lookup table is tickit_sales_redshift.
-- Policy `policy_events` has following dependencies:
--   table tickit_sales_redshift column eventid, qtysold
--   table tickit_event_redshift column eventid
--   table target_event_table column eventid
--   schema public and target_schema
CREATE RLS POLICY policy_events
WITH (eventid INTEGER)
USING (
    eventid IN (SELECT eventid FROM tickit_sales_redshift WHERE qtysold <3)
);

ATTACH RLS POLICY policy_events ON tickit_event_redshift TO ROLE analyst;

ATTACH RLS POLICY policy_events ON target_schema.target_event_table TO ROLE consumer;
```

RLS 정책 사용 고려 사항 및 제한 사항

고려 사항

다음은 RLS 정책 작업에 대한 고려 사항입니다.

- Amazon Redshift는 SELECT, UPDATE 또는 DELETE 문에 RLS 정책을 적용합니다.
- Amazon Redshift는 INSERT, COPY, ALTER TABLE APPEND 문에 RLS 정책을 적용하지 않습니다.
- RLS 정책은 테이블, 뷰, 지연 바인딩 뷰(LBV) 및 구체화된 뷰(MV)에 연결될 수 있습니다.
- 행 수준 보안은 열 수준 보안과 함께 작동하며 데이터를 보호합니다.
- 소스 관계에 대해 RLS가 활성화되면 Amazon Redshift는 수퍼유저, 시스템 권한 IGNORE RLS 또는 sys:secadmin 역할이 명시적으로 부여된 사용자에 대해 ALTER TABLE APPEND 문을 지원합니다. 이 경우 ALTER TABLE APPEND 문을 실행하여 기존 소스 테이블에서 데이터를 이동하는 방법으로 대상 테이블에 행을 추가할 수 있습니다. Amazon Redshift는 소스 관계의 모든 튜플을 대상 관계로 이동합니다. 대상 관계의 RLS 상태는 ALTER TABLE APPEND 문에는 영향을 미치지 않습니다.
- 다른 데이터 웨어하우스 시스템에서 손쉽게 마이그레이션하려면, 변수 이름 및 값을 지정하여 연결에 대한 사용자 지정 세션 컨텍스트 변수를 설정하고 검색하면 됩니다.

  다음 예는 행 수준 보안(RLS) 정책에 대한 세션 컨텍스트 변수를 설정합니다.

  ```
  -- Set a customized context variable.
  ```

```
SELECT set_config('app.category', 'Concerts', FALSE);

-- Create a RLS policy using current_setting() to get the value of a customized
 context variable.
CREATE RLS POLICY policy_categories
WITH (catgroup VARCHAR(10))
USING (catgroup = current_setting('app.category', FALSE));

-- Set correct roles and attach the policy on the target table to one or more roles.
ATTACH RLS POLICY policy_categories ON tickit_category_redshift TO ROLE analyst, ROLE
 dbadmin;
```

사용자 지정 세션 컨텍스트 변수를 설정 및 검색하는 방법을 자세히 알아보려면 SET, SET_CONFIG, SET, CURRENT_SETTING, reset 섹션으로 이동하세요. 일반적인 서버 구성 수정 방법을 자세히 알아보려면 서버 구성 수정 섹션으로 이동하세요.

> **Important**
>
> RLS 정책 내에서 세션 컨텍스트 변수를 사용하는 경우 보안 정책은 정책을 간접 호출하는 사용자 또는 역할에 따라 달라집니다. RLS 정책에서 세션 컨텍스트 변수를 사용할 때는 보안 취약성이 발생하지 않도록 주의하세요.

- DECLARE와 FETCH 사이 또는 후속 FETCH 문 사이에서 SET SESSION AUTHORIZATION을 사용하여 세션 사용자를 변경하면 DECLARE 시간에 사용자 정책을 기반으로 이미 준비된 계획이 새로 고쳐지지 않습니다. 커서가 RLS 보호 테이블과 함께 사용되는 경우 세션 사용자를 변경하지 마세요.
- 뷰 객체 내의 기본 객체가 RLS로 보호되는 경우 쿼리를 실행하는 사용자에게 연결된 정책이 각 기본 객체에 적용됩니다. 이는 뷰 기본 객체에 대해 뷰 소유자의 권한을 확인하는 객체 수준 권한 확인과 다릅니다. EXPLAIN 계획 출력에서 쿼리의 RLS 보호 관계를 볼 수 있습니다.
- 사용자에게 연결된 관계의 RLS 정책에서 사용자 정의 함수 (UDF) 를 참조하는 경우 사용자는 UDF에 대한 EXECUTE 권한이 있어야 관계를 쿼리할 수 있습니다.
- 행 수준 보안은 쿼리 최적화를 제한할 수 있습니다. 대규모 데이터 세트에 RLS로 보호되는 뷰를 배포하기 전에 쿼리 성능을 신중하게 평가하는 것이 좋습니다.
- 지연 바인딩 뷰에 적용된 행 수준 보안 정책은 페더레이션된 테이블로 푸시될 수 있습니다. 이러한 RLS 정책은 외부 처리 엔진 로그에서 볼 수 있습니다.

제한 사항

RLS 정책과 관련한 작업을 할 때의 제한 사항은 다음과 같습니다.

- RLS 정책은 외부 테이블 및 다른 여러 관계 유형에 연결할 수 없습니다. 자세한 내용은 ATTACH RLS POLICY 단원을 참조하십시오.
- Amazon Redshift는 복잡한 조인이 있는 조회를 사용하는 특정 RLS 정책에 대해 SELECT 문을 지원하지만 UPDATE 또는 DELETE 문은 지원하지 않습니다. UPDATE 또는 DELETE 문의 경우 Amazon Redshift는 다음 오류를 반환합니다.

```
ERROR: One of the RLS policies on target relation is not supported in UPDATE/DELETE.
```

- 사용자에게 연결된 관계의 RLS 정책에서 사용자 정의 함수(UDF)를 참조할 때마다, 사용자는 UDF에 대한 EXECUTE 권한이 있어야 관계를 쿼리할 수 있습니다.
- 연관된 하위 쿼리는 지원되지 않습니다. Amazon Redshift는 다음 오류를 반환합니다.

```
ERROR: RLS policy could not be rewritten.
```

- Amazon Redshift Redshift는 RLS와의 데이터 공유를 지원하지 않습니다. 관계에서 데이터 공유에 대한 RLS를 끄지 않은 경우, 다음 오류가 표시되면서 소비자 클러스터에서 쿼리가 실패하게 됩니다.

```
RLS-protected relation "rls_protected_table" cannot be accessed via datasharing
 query.
```

  ROW LEVEL SECURITY OFF FOR DATASHARES 파라미터와 함께 ALTER TABLE 명령을 사용하여 데이터 공유에 대한 RLS를 해제할 수 있습니다. 테이블 변경을 사용하여 RLS를 활성화 또는 비활성화하는 방법에 대한 자세한 내용은 ALTER TABLE 섹션을 참조하세요.
- 데이터베이스 간 쿼리에서 Amazon Redshift는 RLS 보호 관계에 대한 읽기를 차단합니다. IGNORE RLS 권한이 있는 사용자는 데이터베이스 간 쿼리를 사용하여 보호된 관계에 액세스할 수 있습니다. IGNORE RLS 권한이 없는 사용자가 데이터베이스 간 쿼리를 통해 RLS 보호 관계에 액세스하면 다음 오류가 나타납니다.

```
RLS-protected relation "rls_protected_table" cannot be accessed via cross-database
 query.
```

- ALTER RLS POLICY는 USING(using_predicate_exp) 절을 사용하여 RLS 정책을 수정하는 것만 지원합니다. ALTER RLS POLICY를 실행할 때는 WITH 절을 사용하여 RLS 정책을 수정할 수 없습니다.
- 다음 구성 옵션의 값이 세션의 기본값과 일치하지 않는 경우 행 수준 보안이 설정된 관계를 쿼리할 수 없습니다.
  - `enable_case_sensitive_super_attribute`

  - `enable_case_sensitive_identifier`
  - `downcase_delimited_identifier`

  행 수준 보안이 설정된 상태에서 관계를 쿼리하려고 할 때 "대/소문자 구분이 기본값과 다른 경우 RLS 보호 관계가 세션 수준 구성을 지원하지 않습니다."라는 메시지가 표시되는 경우 세션의 구성 옵션을 재설정해 보세요.
- 프로비저닝된 클러스터 또는 서버리스 네임스페이스에 행 수준 보안 정책이 있는 경우 일반 사용자에게는 다음 명령이 차단됩니다.

```
ALTER <current_user> SET enable_case_sensitive_super_attribute/
enable_case_sensitive_identifier/downcase_delimited_identifier
```

  RLS 정책을 만들 때 일반 사용자의 기본 구성 옵션 설정을 정책을 만들 당시의 세션의 구성 옵션 설정과 일치하도록 변경하는 것이 좋습니다. 슈퍼유저 및 ALTER USER 권한이 있는 사용자는 파라미터 그룹 설정 또는 ALTER USER 명령을 사용하여 이 작업을 수행할 수 있습니다. 파라미터 그룹에 대한 자세한 내용은 Amazon Redshift 관리 안내서의 Amazon Redshift 파라미터 그룹을 참조하세요. ALTER USER 명령에 대한 자세한 내용은 ALTER USER 섹션을 참조하세요.
- 행 수준 보안 정책이 적용된 뷰와 지연 바인딩 뷰는 일반 사용자가 CREATE VIEW 명령을 사용하여 교체할 수 없습니다. RLS 정책이 적용된 뷰 또는 LBV를 바꾸려면 먼저 연결된 RLS 정책을 분리하고 뷰 또는 LBV를 교체한 다음, 정책을 다시 연결합니다. 슈퍼 사용자 및 `sys:secadmin permission`이 있는 사용자는 정책을 분리하지 않고도 RLS 정책이 적용된 뷰 또는 LBV에 CREATE VIEW를 사용할 수 있습니다.
- 행 수준 보안 정책이 적용된 뷰는 시스템 테이블과 시스템 뷰를 참조할 수 없습니다.
- 일반 뷰에서 참조하는 지연 바인딩 뷰는 RLS로 보호될 수 없습니다.
- RLS로 보호되는 관계와 데이터 레이크의 중첩 데이터는 동일한 쿼리에서 액세스할 수 없습니다.

RLS 성능 관련 모범 사례

다음은 RLS로 보호되는 테이블에서 높은 Amazon Redshift 성능을 보장하기 위한 모범 사례입니다.

연산자 및 함수의 안전

RLS 보호 테이블을 쿼리할 때 특정 연산자나 함수를 사용하면 성능이 저하될 수 있습니다. Amazon Redshift는 RLS 보호 테이블을 쿼리할 때 연산자와 함수를 안전하거나 안전하지 않은 것으로 분류합니다. 함수나 연산자는 입력에 따라 관찰 가능한 부작용이 없는 경우 RLS 안전으로 분류됩니다. 구체적으로 살펴보면, RLS 안전 함수 또는 연산자는 다음 중 하나일 수 없습니다.

- 오류 메시지를 포함하거나 포함하지 않은 상태로, 입력 값 또는 입력 값에 종속된 값을 출력합니다.
- 입력 값에 종속된 오류를 실패 처리하거나 반환합니다.

RLS 불안전 연산자는 다음과 같습니다.

- 산술 연산자: +, -, /, *, %.
- 텍스트 연산자: LIKE 및 SIMILAR TO
- 변환 운영자
- UDF

다음 SELECT 문을 사용하여 연산자와 함수의 안전성을 검사합니다.

```
SELECT proname, proc_is_rls_safe(oid) FROM pg_proc;
```

Amazon Redshift는 RLS 보호 테이블에 대한 쿼리를 계획할 때 RLS 불안전 연산자 및 함수를 포함하는 사용자 조건자의 평가 순서를 제한합니다. RLS 불안전 연산자 또는 함수를 참조하는 쿼리는 RLS 보호 테이블을 쿼리할 때 성능이 저하될 수 있습니다. Amazon Redshift가 정렬 키를 활용하기 위해 RLS 불안전 조건자를 기본 테이블 스캔으로 푸시다운할 수 없는 경우 성능이 크게 저하될 수 있습니다. 더 나은 성능을 얻으려면 정렬 키를 활용하는 RLS 불안전 조건자를 사용하는 쿼리는 사용하지 마세요. Amazon Redshift가 연산자와 함수를 푸시다운할 수 있는지 확인하려면 EXPLAIN 문을 시스템 권한 EXPLAIN RLS와 함께 사용하면 됩니다.

결과 캐싱

쿼리 런타임을 줄이고 시스템 성능을 향상시키기 위해 Amazon Redshift는 특정 형식의 쿼리 결과를 리더 노드의 메모리에 캐시합니다.

Amazon Redshift는 보호되지 않은 테이블에 대한 조건이 모두 충족되고 다음 조건이 모두 충족될 경우, 새 쿼리 스캐닝 RLS 보호 테이블에 캐싱된 결과를 사용합니다.

- 정책의 테이블 또는 보기가 수정되지 않았습니다.
- 정책은 실행 시마다 평가되어야 하는 함수(예: GETDATE 또는 CURRENT_USER)를 사용하지 않습니다.

성능을 높이려면, 이전 조건을 충족하지 않는 정책 조건자를 사용하지 마세요.

Amazon Redshift의 결과 캐싱에 대한 자세한 내용은 결과 캐싱 섹션을 참조하세요.

복잡한 정책

성능을 높이려면 여러 테이블을 조인하는 하위 쿼리에 복잡한 정책을 사용하지 마세요.

행 수준 보안 엔드 투 엔드 예제

다음은 수퍼유저가 몇 가지 사용자와 역할을 만드는 방법을 보여주는 자세한 예입니다. 그러면 secadmin 역할이 부여된 사용자가 RLS 정책을 생성, 연결, 분리 및 삭제합니다. 이 예에서는 tickit 샘플 데이터베이스를 사용합니다. 자세한 내용은 Amazon Redshift 시작 안내서의 Amazon S3에서 Amazon Redshift로 데이터 로드를 참조하세요.

```
-- Create users and roles referenced in the policy statements.
CREATE ROLE analyst;
CREATE ROLE consumer;
CREATE ROLE dbadmin;
CREATE ROLE auditor;
CREATE USER bob WITH PASSWORD 'Name_is_bob_1';
CREATE USER alice WITH PASSWORD 'Name_is_alice_1';
CREATE USER joe WITH PASSWORD 'Name_is_joe_1';
CREATE USER molly WITH PASSWORD 'Name_is_molly_1';
CREATE USER bruce WITH PASSWORD 'Name_is_bruce_1';
GRANT ROLE sys:secadmin TO bob;
GRANT ROLE analyst TO alice;
GRANT ROLE consumer TO joe;
GRANT ROLE dbadmin TO molly;
GRANT ROLE auditor TO bruce;
GRANT ALL ON TABLE tickit_category_redshift TO PUBLIC;
GRANT ALL ON TABLE tickit_sales_redshift TO PUBLIC;
GRANT ALL ON TABLE tickit_event_redshift TO PUBLIC;

-- Create table and schema referenced in the policy statements.
CREATE SCHEMA target_schema;
GRANT ALL ON SCHEMA target_schema TO PUBLIC;
CREATE TABLE target_schema.target_event_table (LIKE tickit_event_redshift);
GRANT ALL ON TABLE target_schema.target_event_table TO PUBLIC;

-- Change session to analyst alice.
SET SESSION AUTHORIZATION alice;

-- Check the tuples visible to analyst alice.
-- Should contain all 3 categories.
SELECT catgroup, count(*)
FROM tickit_category_redshift
```

```
GROUP BY catgroup ORDER BY catgroup;

-- Change session to security administrator bob.
SET SESSION AUTHORIZATION bob;

CREATE RLS POLICY policy_concerts
WITH (catgroup VARCHAR(10))
USING (catgroup = 'Concerts');

SELECT poldb, polname, polalias, polatts, polqual, polenabled, polmodifiedby FROM
 svv_rls_policy WHERE poldb = CURRENT_DATABASE();

ATTACH RLS POLICY policy_concerts ON tickit_category_redshift TO ROLE analyst, ROLE
 dbadmin;

ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON;

SELECT * FROM svv_rls_attached_policy;

-- Change session to analyst alice.
SET SESSION AUTHORIZATION alice;

-- Check that tuples with only `Concert` category will be visible to analyst alice.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Change session to consumer joe.
SET SESSION AUTHORIZATION joe;

-- Although the policy is attached to a different role, no tuples will be
-- visible to consumer joe because the default deny all policy is applied.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Change session to dbadmin molly.
SET SESSION AUTHORIZATION molly;

-- Check that tuples with only `Concert` category will be visible to dbadmin molly.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;
```

```
-- Check that EXPLAIN output contains RLS SecureScan to prevent disclosure of
-- sensitive information such as RLS filters.
EXPLAIN SELECT catgroup, count(*) FROM tickit_category_redshift GROUP BY catgroup ORDER
 BY catgroup;

-- Change session to security administrator bob.
SET SESSION AUTHORIZATION bob;

-- Grant IGNORE RLS permission so that RLS policies do not get applicable to role
 dbadmin.
GRANT IGNORE RLS TO ROLE dbadmin;

-- Grant EXPLAIN RLS permission so that anyone in role auditor can view complete
 EXPLAIN output.
GRANT EXPLAIN RLS TO ROLE auditor;

-- Change session to dbadmin molly.
SET SESSION AUTHORIZATION molly;

-- Check that all tuples are visible to dbadmin molly because `IGNORE RLS` is granted
 to role dbadmin.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Change session to auditor bruce.
SET SESSION AUTHORIZATION bruce;

-- Check explain plan is visible to auditor bruce because `EXPLAIN RLS` is granted to
 role auditor.
EXPLAIN SELECT catgroup, count(*) FROM tickit_category_redshift GROUP BY catgroup ORDER
 BY catgroup;

-- Change session to security administrator bob.
SET SESSION AUTHORIZATION bob;

DETACH RLS POLICY policy_concerts ON tickit_category_redshift FROM ROLE analyst, ROLE
 dbadmin;

-- Change session to analyst alice.
SET SESSION AUTHORIZATION alice;

-- Check that no tuples are visible to analyst alice.
-- Although the policy is detached, no tuples will be visible to analyst alice
```

```
-- because of default deny all policy is applied if the table has RLS on.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Change session to security administrator bob.
SET SESSION AUTHORIZATION bob;

CREATE RLS POLICY policy_events
WITH (eventid INTEGER) AS ev
USING (
    ev.eventid IN (SELECT eventid FROM tickit_sales_redshift WHERE qtysold <3)
);

ATTACH RLS POLICY policy_events ON tickit_event_redshift TO ROLE analyst;
ATTACH RLS POLICY policy_events ON target_schema.target_event_table TO ROLE consumer;

RESET SESSION AUTHORIZATION;

-- Can not cannot alter type of dependent column.
ALTER TABLE target_schema.target_event_table ALTER COLUMN eventid TYPE float;
ALTER TABLE tickit_event_redshift ALTER COLUMN eventid TYPE float;
ALTER TABLE tickit_sales_redshift ALTER COLUMN eventid TYPE float;
ALTER TABLE tickit_sales_redshift ALTER COLUMN qtysold TYPE float;

-- Can not cannot rename dependent column.
ALTER TABLE target_schema.target_event_table RENAME COLUMN eventid TO renamed_eventid;
ALTER TABLE tickit_event_redshift RENAME COLUMN eventid TO renamed_eventid;
ALTER TABLE tickit_sales_redshift RENAME COLUMN eventid TO renamed_eventid;
ALTER TABLE tickit_sales_redshift RENAME COLUMN qtysold TO renamed_qtysold;

-- Can not drop dependent column.
ALTER TABLE target_schema.target_event_table DROP COLUMN eventid CASCADE;
ALTER TABLE tickit_event_redshift DROP COLUMN eventid CASCADE;
ALTER TABLE tickit_sales_redshift DROP COLUMN eventid CASCADE;
ALTER TABLE tickit_sales_redshift DROP COLUMN qtysold CASCADE;

-- Can not drop lookup table.
DROP TABLE tickit_sales_redshift CASCADE;

-- Change session to security administrator bob.
SET SESSION AUTHORIZATION bob;

DROP RLS POLICY policy_concerts;
```

```
DROP RLS POLICY IF EXISTS policy_events;

ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY OFF;

RESET SESSION AUTHORIZATION;

-- Drop users and roles.
DROP USER bob;
DROP USER alice;
DROP USER joe;
DROP USER molly;
DROP USER bruce;
DROP ROLE analyst;
DROP ROLE consumer;
DROP ROLE auditor FORCE;
DROP ROLE dbadmin FORCE;
```

메타데이터 보안

Amazon Redshift의 행 수준 보안과 마찬가지로 메타데이터 보안을 사용하면 메타데이터를 보다 세밀하게 제어할 수 있습니다. 프로비저닝된 클러스터 또는 서버리스 작업 그룹에 메타데이터 보안이 활성화된 경우 사용자는 보기 액세스 권한이 있는 객체의 메타데이터를 볼 수 있습니다. 메타데이터 보안을 사용하면 필요에 따라 가시성을 구분할 수 있습니다. 예를 들어 단일 데이터 웨어하우스를 사용하여 모든 데이터 스토리지를 중앙 집중화할 수 있습니다. 그러나 여러 섹터에 데이터를 저장하는 경우 보안 관리가 번거로울 수 있습니다. 메타데이터 보안을 활성화하면 가시성을 구성할 수 있습니다. 한 섹터의 사용자는 자신의 객체에 대해 더 큰 가시성을 가지는 동시에 다른 섹터 사용자에게는 보기 액세스를 제한할 수 있습니다. 메타데이터 보안은 스키마, 테이블, 뷰, 구체화된 뷰, 저장 프로시저, 사용자 정의 함수 및 기계 학습 모델과 같은 모든 객체 유형을 지원합니다.

사용자는 다음과 같은 상황에서 객체의 메타데이터를 볼 수 있습니다.

- 사용자에게 객체 액세스 권한이 부여된 경우
- 사용자가 속한 그룹 또는 역할에 객체 액세스 권한이 부여된 경우
- 객체는 공개됩니다(퍼블릭).
- 사용자는 데이터베이스 객체의 소유자입니다.

메타데이터 보안을 활성화하려면 ALTER SYSTEM 명령을 사용합니다. 다음은 ALTER SYSTEM 명령을 메타데이터 보안과 함께 사용하는 방법을 나타내는 구문입니다.

```
ALTER SYSTEM SET metadata_security=[true|t|on|false|f|off];
```

메타데이터 보안을 활성화하면 필요한 권한을 가진 모든 사용자가 액세스 권한이 있는 객체의 관련 메타데이터를 볼 수 있습니다. 특정 사용자만 메타데이터 보안을 볼 수 있게 하려면 역할에 ACCESS CATALOG 권한을 부여한 다음 사용자에게 역할을 할당하세요. 역할을 사용하여 보안을 강화하는 방법에 대한 자세한 내용은 역할 기반 액세스 제어를 참조하세요.

다음 예시는 역할에 ACCESS CATALOG 권한을 부여한 다음 사용자에게 역할을 할당하는 방법을 보여줍니다. 권한을 부여하는 방법에 대한 자세한 내용은 GRANT 명령을 참조하세요

```
CREATE ROLE sample_metadata_viewer;

GRANT ACCESS CATALOG TO ROLE sample_metadata_viewer;

GRANT ROLE sample_metadata_viewer to salesadmin;
```

이미 정의된 역할을 사용하려는 경우 시스템 정의 역할인 operator, secadmin, dba, superuser가 모두 객체 메타데이터를 보는 데 필요한 권한이 있습니다. 기본적으로 슈퍼 사용자는 전체 카탈로그를 볼 수 있습니다.

```
GRANT ROLE operator to sample_user;
```

역할을 사용하여 메타데이터 보안을 제어하는 경우 역할 기반 액세스 제어와 함께 제공되는 모든 시스템 뷰 및 기능에 액세스할 수 있습니다. 예를 들어, SVV_ROLES 뷰를 쿼리하여 모든 역할을 확인할 수 있습니다. 사용자가 역할 또는 그룹의 구성원인지 확인하려면 USER_IS_MEMBER_OF 함수를 사용하세요. SVV 뷰의 전체 목록은 SVV 메타데이터 뷰를 참조하세요. 시스템 정보 함수 목록은 시스템 정보 함수를 참조하세요.

동적 데이터 마스킹

Amazon Redshift에서 동적 데이터 마스킹(DDM)을 사용하면 데이터 웨어하우스의 민감한 데이터를 보호할 수 있습니다. Amazon Redshift가 데이터베이스에서 데이터를 변환하지 않고 쿼리 시 민감한 데이터를 사용자에게 표시하는 방식을 조작할 수 있습니다. 지정된 사용자 또는 역할에 사용자 지정 난독화 규칙을 적용하는 마스킹 정책을 통해 데이터에 대한 액세스를 제어합니다. 이렇게 하면 기본 데이터를 변경하거나 SQL 쿼리를 편집하지 않고도 변화하는 개인 정보 보호 요구 사항에 대응할 수 있습니다.

동적 데이터 마스킹 정책은 지정된 형식과 일치하는 데이터를 숨기거나 난독화하거나 익명화합니다. 테이블에 첨부하면 마스킹 표현식이 하나 이상의 해당 열에 적용됩니다. 마스킹 정책을 추가로 수정하여 특정 사용자 또는 역할 기반 액세스 제어(RBAC)로 생성할 수 있는 사용자 정의 역할에만 적용할 수 있습니다. 또한 마스킹 정책을 생성할 때 조건부 열을 사용하여 셀 수준에서 DDM을 적용할 수 있습니다. 조건부 마스킹에 대한 자세한 설명은 조건부 동적 데이터 마스킹 섹션을 참조하세요.

난독화 수준이 다른 여러 마스킹 정책을 테이블의 동일한 열에 적용하고 다른 역할에 할당할 수 있습니다. 하나의 열에 다른 정책을 적용하는 다른 역할이 있는 경우 충돌을 방지하기 위해 각 응용 프로그램에 대한 우선 순위를 설정할 수 있습니다. 이러한 방식으로 지정된 사용자 또는 역할이 액세스할 수 있는 데이터를 제어할 수 있습니다. DDM 정책은 데이터를 부분적으로 또는 완전히 수정하거나 SQL, Python 또는 AWS Lambda로 작성된 사용자 정의 함수를 사용하여 데이터를 해시할 수 있습니다. 해시를 사용하여 데이터를 마스킹하면 잠재적으로 민감한 정보에 액세스하지 않고도 이 데이터에 조인을 적용할 수 있습니다.

동적 데이터 마스킹 정책 관리용 SQL 명령

다음 작업을 수행하여 동적 데이터 마스킹 정책을 생성, 연결, 분리 및 삭제할 수 있습니다.

- DDM 정책을 만들려면 마스킹 정책 생성 명령을 사용합니다.

  다음은 SHA-2 해시함수를 이용하여 마스킹 정책을 생성한 예이다.

  ```
  CREATE MASKING POLICY hash_credit
  WITH (credit_card varchar(256))
  USING (sha2(credit_card + 'testSalt', 256));
  ```

- 기존 DDM 정책을 변경하려면 마스킹 정책 변경 명령을 사용합니다.

  다음은 기존 마스킹 정책을 변경하는 예제입니다.

  ```
  ALTER MASKING POLICY hash_credit
  USING (sha2(credit_card + 'otherTestSalt', 256));
  ```

- 테이블에 대한 DDM 정책을 하나 이상의 사용자 또는 역할에 연결하려면 마스킹 정책 연결 명령을 사용합니다.

  다음은 마스킹 정책을 열/역할 쌍에 연결하는 예제입니다.

  ```
   ATTACH MASKING POLICY hash_credit
  ON credit_cards (credit_card)
  TO ROLE science_role
  ```

```
PRIORITY 30;
```

PRIORITY 절은 여러 정책이 동일한 열에 첨부될 때 사용자 세션에 적용되는 마스킹 정책을 결정합니다. 예를 들어 앞의 예에서 사용자가 동일한 신용 카드 열에 우선 순위가 20인 다른 마스킹 정책을 연결한 경우 science_role의 정책이 우선 순위가 30보다 높으므로 적용됩니다.

- 하나 이상의 사용자 또는 역할에서 테이블의 DDM 정책을 분리하려면 DETACH 마스킹 정책 분리 명령을 사용합니다.

  다음은 마스킹 정책을 열/역할 쌍에서 분리하는 예제입니다.

```
DETACH MASKING POLICY hash_credit
ON credit_cards(credit_card)
FROM ROLE science_role;
```

- 모든 데이터베이스에서 DDM 정책을 삭제하려면 마스킹 정책 삭제 명령을 사용하세요.

  다음은 모든 데이터베이스에서 마스킹 정책을 삭제하는 예제입니다.

```
DROP MASKING POLICY hash_credit;
```

동적 데이터 마스킹 정책 계층 구조

마스킹 정책을 여러 개 연결할 때는 다음 사항을 고려하세요.

- 여러 마스킹 정책을 단일 열에 연결할 수 있습니다.
- 쿼리에 여러 마스킹 정책을 적용할 경우 각 열에 첨부된 가장 높은 우선 순위의 정책이 적용됩니다. 다음 예제를 살펴보세요.

```
ATTACH MASKING POLICY partial_hash
ON credit_cards(address, credit_card)
TO ROLE analytics_role
PRIORITY 20;

ATTACH MASKING POLICY full_hash
ON credit_cards(credit_card, ssn)
TO ROLE auditor_role
PRIORITY 30;

SELECT address, credit_card, ssn
```

```
FROM credit_cards;
```

SELECT 문을 실행할 때 분석 및 감사자 역할을 모두 가진 사용자는 partial_hash 마스킹 정책이 적용된 주소 열을 보게 됩니다. 신용카드 열에서 full_hash 정책의 우선순위가 더 높기 때문에 full_hash 마스킹 정책이 적용된 신용카드 및 SSN 열이 표시됩니다.

- 마스킹 정책을 연결할 때 우선 순위를 지정하지 않으면 기본 우선 순위는 0입니다.
- 우선 순위가 동일한 동일한 열에 두 개의 정책을 연결할 수 없습니다.
- 사용자와 열 또는 역할과 열의 동일한 조합에 두 개의 정책을 연결할 수 없습니다.
- 동일한 사용자 또는 역할에 연결된 상태에서 동일한 SUPER 경로를 따라 여러 마스킹 정책을 적용할 경우 우선순위가 가장 높은 첨부 파일만 적용됩니다. 다음 예제를 살펴보세요.

  첫 번째 예는 동일한 경로에 연결된 두 개의 마스킹 정책을 보여 주며 우선순위가 더 높은 정책이 적용됩니다.

```
ATTACH MASKING POLICY hide_name
ON employees(col_person.name)
TO PUBLIC
PRIORITY 20;

ATTACH MASKING POLICY hide_last_name
ON employees(col_person.name.last)
TO PUBLIC
PRIORITY 30;

--Only the hide_last_name policy takes effect.
SELECT employees.col_person.name FROM employees;
```

  두 번째 예에서는 정책 간 충돌 없이 동일한 SUPER 객체의 서로 다른 경로에 연결된 두 마스킹 정책을 보여줍니다. 두 첨부 파일이 동시에 적용됩니다.

```
ATTACH MASKING POLICY hide_first_name
ON employees(col_person.name.first)
TO PUBLIC
PRIORITY 20;

ATTACH MASKING POLICY hide_last_name
ON employees(col_person.name.last)
TO PUBLIC
PRIORITY 20;
```

```
--Both col_person.name.first and col_person.name.last are masked.
SELECT employees.col_person.name FROM employees;
```

주어진 사용자와 열 또는 역할과 열 조합에 적용되는 마스킹 정책을 확인하기 위해 `sys:secadmin` 역할이 있는 사용자는 SVV_ATTACHED_MASKING_POLICY 시스템 뷰에서 열/역할 또는 열/사용자 쌍을 조회할 수 있습니다. 자세한 내용은 동적 데이터 마스킹 시스템 뷰 단원을 참조하십시오.

SUPER 데이터 유형 경로와 함께 동적 데이터 마스킹 사용

Amazon Redshift는 동적 데이터 마스킹 정책을 SUPER 유형 열의 경로에 연결하는 것을 지원합니다. SUPER 데이터 형식에 대한 자세한 내용은 Amazon Redshift의 반정형 데이터 섹션을 참조하세요.

SUPER 유형 열의 경로에 마스킹 정책을 연결할 때는 다음 사항을 고려하세요.

- 마스킹 정책을 열의 경로에 연결하는 경우 해당 열을 SUPER 데이터 유형으로 정의해야 합니다. SUPER 경로의 스칼라 값에만 마스킹 정책을 적용할 수 있습니다. 복잡한 구조나 배열에는 마스킹 정책을 적용할 수 없습니다.
- SUPER 경로가 충돌하지 않는 한 단일 SUPER 열의 여러 스칼라 값에 서로 다른 마스킹 정책을 적용할 수 있습니다. 예를 들어 SUPER 경로 `a.b` 및 `a.b.c`는 `a.b`가 `a.b.c`의 상위 구조로 동일한 경로에 있기 때문에 충돌이 발생합니다. SUPER 경로 `a.b.c` 및 `a.b.d`는 충돌하지 않습니다.
- Amazon Redshift는 사용자 쿼리 런타임 시 정책이 적용되기 전까지는 마스킹 정책이 연결된 경로가 데이터에 있고 예상 유형인지 확인할 수 없습니다. 예를 들어, INT 값이 포함된 SUPER 경로에 TEXT 값을 마스킹하는 마스킹 정책을 연결하면 Amazon Redshift는 해당 경로에 있는 값의 유형을 캐스팅하려고 시도합니다.

  이러한 상황에서 런타임 시 Amazon Redshift의 동작은 SUPER 객체를 쿼리하기 위한 구성 설정에 따라 달라집니다. 기본적으로 Amazon Redshift는 lax 모드이며 지정된 SUPER 경로에서 누락된 경로와 잘못된 캐스트를 NULL로 해석합니다. SUPER와 관련한 구성 설정에 대한 자세한 내용은 SUPER 구성 섹션을 참조하세요.
- SUPER는 스키마가 없는 유형입니다. 즉, Amazon Redshift는 지정된 SUPER 경로에서 값의 존재를 확인할 수 없습니다. 존재하지 않는 SUPER 경로에 마스킹 정책을 연결하고 Amazon Redshift가 lax 모드인 경우 Amazon Redshift는 경로를 NULL 값으로 해석합니다. SUPER 열 경로에 마스킹 정책을 연결할 때는 SUPER 객체의 예상 형식과 이러한 객체에 예상치 못한 속성이 있을 가능성을 고려하는 것이 좋습니다. SUPER 열에 예상치 못한 스키마가 있다고 생각되면 마스킹 정책을 SUPER 열에 직접 연결하는 것을 고려해 보세요. SUPER 유형 정보 함수를 사용하여 속성 및 유형을 확인하고 값을 마스킹하는 데 `OBJECT_TRANSFORM`을 사용할 수 있습니다. SUPER 유형 정보 함수에 대한 자세한 내용은 SUPER 형식 정보 함수 섹션을 참조하세요.

예시

SUPER 경로에 마스킹 정책 연결

다음 예에서는 여러 마스킹 정책을 한 열의 여러 SUPER 유형 경로에 연결합니다.

```
CREATE TABLE employees (
    col_person SUPER
);

INSERT INTO employees
VALUES
    (
        json_parse('
            {
                "name": {
                    "first": "John",
                    "last": "Doe"
                },
                "age": 25,
                "ssn": "111-22-3333",
                "company": "Company Inc."
            }
        ')
    ),
    (
        json_parse('
            {
                "name": {
                    "first": "Jane",
                    "last": "Appleseed"
                },
                "age": 34,
                "ssn": "444-55-7777",
                "company": "Organization Org."
            }
        ')
    )
;
GRANT ALL ON ALL TABLES IN SCHEMA "public" TO PUBLIC;

-- Create the masking policies.

-- This policy converts the given name to all uppercase letters.
```

```
CREATE MASKING POLICY mask_first_name
WITH(first_name TEXT)
USING ( UPPER(first_name) );

-- This policy replaces the given name with the fixed string 'XXXX'.
CREATE MASKING POLICY mask_last_name
WITH(last_name TEXT)
USING ( 'XXXX'::TEXT );

-- This policy rounds down the given age to the nearest 10.
CREATE MASKING POLICY mask_age
WITH(age INT)
USING ( (FLOOR(age::FLOAT / 10) * 10)::INT );

-- This policy converts the first five digits of the given SSN to 'XXX-XX'.
CREATE MASKING POLICY mask_ssn
WITH(ssn TEXT)
USING ( 'XXX-XX-'::TEXT || SUBSTRING(ssn::TEXT FROM 8 FOR 4) );

-- Attach the masking policies to the employees table.
ATTACH MASKING POLICY mask_first_name
ON employees(col_person.name.first)
TO PUBLIC;

ATTACH MASKING POLICY mask_last_name
ON employees(col_person.name.last)
TO PUBLIC;

ATTACH MASKING POLICY mask_age
ON employees(col_person.age)
TO PUBLIC;

ATTACH MASKING POLICY mask_ssn
ON employees(col_person.ssn)
TO PUBLIC;

-- Verify that your masking policies are attached.
SELECT
    policy_name,
    TABLE_NAME,
    priority,
    input_columns,
    output_columns
FROM
```

```
    svv_attached_masking_policy;

   policy_name   | table_name | priority |           input_columns           |
 output_columns
-----------------+------------+----------+-----------------------------------
+-----------------------------------
 mask_age        | employees  |        0 | ["col_person.\"age\""]            |
 ["col_person.\"age\""]
 mask_first_name | employees  |        0 | ["col_person.\"name\".\"first\""] |
 ["col_person.\"name\".\"first\""]
 mask_last_name  | employees  |        0 | ["col_person.\"name\".\"last\""]  |
 ["col_person.\"name\".\"last\""]
 mask_ssn        | employees  |        0 | ["col_person.\"ssn\""]            |
 ["col_person.\"ssn\""]
(4 rows)

-- Observe the masking policies taking effect.
SELECT col_person FROM employees ORDER BY col_person.age;

-- This result is formatted for ease of reading.
         col_person
--------------------------------
{
    "name": {
        "first": "JOHN",
        "last": "XXXX"
    },
    "age": 20,
    "ssn": "XXX-XX-3333",
    "company": "Company Inc."
}
{
    "name": {
        "first": "JANE",
        "last": "XXXX"
    },
    "age": 30,
    "ssn": "XXX-XX-7777",
    "company": "Organization Org."
}
```

다음은 SUPER 경로에 잘못된 마스킹 정책을 첨부한 몇 가지 예입니다.

```
-- This attachment fails because there is already a policy
-- with equal priority attached to employees.name.last, which is
-- on the same SUPER path as employees.name.
ATTACH MASKING POLICY mask_ssn
ON employees(col_person.name)
TO PUBLIC;
ERROR:  DDM policy "mask_last_name" is already attached on relation "employees" column
 "col_person."name"."last"" with same priority

-- Create a masking policy that masks DATETIME objects.
CREATE MASKING POLICY mask_date
WITH(INPUT DATETIME)
USING ( INPUT );

-- This attachment fails because SUPER type columns can't contain DATETIME objects.
ATTACH MASKING POLICY mask_date
ON employees(col_person.company)
TO PUBLIC;
ERROR:  cannot attach masking policy for output of type "timestamp without time zone"
 to column "col_person."company"" of type "super
```

다음은 마스킹 정책을 존재하지 않는 SUPER 경로에 연결하는 예입니다. 기본적으로 Amazon Redshift는 경로를 NULL로 해석합니다.

```
ATTACH MASKING POLICY mask_first_name
ON employees(col_person.not_exists)
TO PUBLIC;

SELECT col_person FROM employees LIMIT 1;

-- This result is formatted for ease of reading.
         col_person
----------------------------------
{
    "name": {
        "first": "JOHN",
        "last": "XXXX"
    },
    "age": 20,
    "ssn": "XXX-XX-3333",
    "company": "Company Inc.",
    "not_exists": null
```

```
}
```

조건부 동적 데이터 마스킹

마스킹 표현식에서 조건식을 사용하여 마스킹 정책을 생성하여 셀 수준에서 데이터를 마스킹할 수 있습니다. 예를 들어 해당 행에 있는 다른 열의 값에 따라 다른 마스크를 값에 적용하는 마스킹 정책을 만들 수 있습니다.

다음은 조건부 데이터 마스킹을 사용하여 사기와 관련된 신용 카드 번호를 부분적으로 수정하고 다른 모든 신용 카드 번호를 완전히 숨기는 마스킹 정책을 만들고 첨부하는 예입니다. 이 예제를 실행하려면 수퍼유저이거나 sys:secadmin 역할이 있어야 합니다.

```
--Create an analyst role.
CREATE ROLE analyst;

--Create a credit card table. The table contains an is_fraud boolean column,
--which is TRUE if the credit card number in that row was involved in a fraudulent
 transaction.
CREATE TABLE credit_cards (id INT, is_fraud BOOLEAN, credit_card_number VARCHAR(16));

--Create a function that partially redacts credit card numbers.
CREATE FUNCTION REDACT_CREDIT_CARD (credit_card VARCHAR(16))
RETURNS VARCHAR(16) IMMUTABLE
AS $$
    import re
    regexp = re.compile("^([0-9]{6})[0-9]{5,6}([0-9]{4})")

    match = regexp.search(credit_card)
    if match != None:
        first = match.group(1)
        last = match.group(2)
    else:
        first = "000000"
        last = "0000"

    return "{}XXXXX{}".format(first, last)
$$ LANGUAGE plpythonu;

--Create a masking policy that partially redacts credit card numbers if the is_fraud
 value for that row is TRUE,
--and otherwise blanks out the credit card number completely.
CREATE MASKING POLICY card_number_conditional_mask
```

```
    WITH (fraudulent BOOLEAN, pan varchar(16))
    USING (CASE WHEN fraudulent THEN REDACT_CREDIT_CARD(pan)
                ELSE Null
           END);

--Attach the masking policy to the credit_cards/analyst table/role pair.
ATTACH MASKING POLICY card_number_conditional_mask ON credit_cards (credit_card_number)
 USING (is_fraud, credit_card_number)
 TO ROLE analyst PRIORITY 100;
```

동적 데이터 마스킹 시스템 뷰

수퍼유저, `sys:operator` 역할이 있는 사용자, ACCESS SYSTEM TABLE 권한이 있는 사용자는 다음과 같은 DDM 관련 시스템 뷰에 액세스할 수 있습니다.

- SVV_MASKING_POLICY

  SVV_MASKING_POLICY를 사용하여 클러스터 또는 작업 그룹에 생성된 모든 마스킹 정책을 볼 수 있습니다.
- SVV_ATTACHED_MASKING_POLICY

  SVV_ATTACHED_MASKING_POLICY를 사용하여 현재 연결된 데이터베이스에 정책이 연결된 모든 관계 및 사용자 또는 역할을 봅니다.
- SYS_APPLIED_MASKING_POLICY_LOG

  SYS_APPLIED_MASKING_POLICY_LOG를 사용하여 DDM로 보호되는 관계를 참조하는 쿼리에서 마스킹 정책의 적용을 추적합니다.

다음은 시스템 뷰를 사용하여 찾을 수 있는 정보의 몇 가지 예제입니다.

```
--Select all policies associated with specific users, as opposed to roles
SELECT policy_name,
       schema_name,
       table_name,
       grantee
FROM svv_attached_masking_policy
WHERE grantee_type = 'user';

--Select all policies attached to a specific user
SELECT policy_name,
       schema_name,
```

```
       table_name,
       grantee
FROM svv_attached_masking_policy
WHERE grantee = 'target_grantee_name'

--Select all policies attached to a given table
SELECT policy_name,
       schema_name,
       table_name,
       grantee
FROM svv_attached_masking_policy
WHERE table_name = 'target_table_name'
      AND schema_name = 'target_schema_name';

--Select the highest priority policy attachment for a given role
SELECT samp.policy_name,
       samp.priority,
       samp.grantee,
       smp.policy_expression
FROM svv_masking_policy AS smp
JOIN svv_attached_masking_policy AS samp
    ON samp.policy_name = smp.policy_name
WHERE
    samp.grantee_type = 'role' AND
    samp.policy_name = mask_get_policy_for_role_on_column(
        'target_schema_name',
        'target_table_name',
        'target_column_name',
        'target_role_name')
ORDER BY samp.priority desc
LIMIT 1;

--See which policy a specific user will see on a specific column in a given relation
SELECT samp.policy_name,
       samp.priority,
       samp.grantee,
       smp.policy_expression
FROM svv_masking_policy AS smp
JOIN svv_attached_masking_policy AS samp
    ON samp.policy_name = smp.policy_name
WHERE
    samp.grantee_type = 'role' AND
    samp.policy_name = mask_get_policy_for_user_on_column(
        'target_schema_name',
```

```
        'target_table_name',
        'target_column_name',
        'target_user_name')
ORDER BY samp.priority desc;

 --Select all policies attached to a given relation.
SELECT policy_name,
schema_name,
relation_name,
database_name
FROM sys_applied_masking_policy_log
WHERE relation_name = 'relation_name'
AND schema_name = 'schema_name';
```

동적 데이터 마스킹 사용 시 고려 사항

동적 데이터 마스킹을 사용할 때 다음 사항을 고려하세요.

- 뷰와 같은 테이블에서 생성된 객체를 쿼리할 때 사용자는 객체를 생성한 사용자의 정책이 아닌 자신의 마스킹 정책에 따라 결과를 보게 됩니다. 예를 들어 secadmin이 생성한 뷰를 쿼리하는 분석가 역할을 가진 사용자는 분석가 역할에 연결된 마스킹 정책이 있는 결과를 볼 수 있습니다.
- EXPLAIN 명령이 잠재적으로 중요한 마스킹 정책 필터를 노출하지 않도록 하기 위해 SYS_EXPLAIN_DDM 권한이 있는 사용자만 EXPLAIN 출력에 적용된 마스킹 정책을 볼 수 있습니다. 사용자에게는 기본적으로 SYS_EXPLAIN_DDM 권한이 없습니다.

  다음은 역할에 권한을 부여할 때 사용하는 구문입니다.

  ```
  GRANT EXPLAIN MASKING TO ROLE rolename
  ```

  EXPLAIN 명령에 대한 자세한 내용은 EXPLAIN 섹션을 참조하세요.
- 역할이 다른 사용자는 사용된 필터 조건 또는 조인 조건에 따라 다른 결과를 볼 수 있습니다. 예를 들어 명령을 실행하는 사용자에게 해당 열을 난독화하는 마스킹 정책이 적용된 경우 특정 열 값을 사용하여 테이블에서 SELECT 명령을 실행하면 실패합니다.
- DDM 정책은 조건자 작업 또는 예측보다 먼저 적용되어야 합니다. 마스킹 정책에는 다음이 포함될 수 있습니다.
  - 값을 null로 변환하는 것과 같은 저비용 상수 연산
  - HMAC 해싱과 같은 중간 비용 연산
  - 외부 Lambda 사용자 정의 함수 호출과 같은 고비용 연산

따라서 가능하면 간단한 마스킹 표현식을 사용하는 것이 좋습니다.

- 행 수준 보안 정책이 있는 역할에 DDM 정책을 사용할 수 있지만 RLS 정책은 DDM보다 먼저 적용됩니다. 동적 데이터 마스킹 표현식은 RLS로 보호된 행을 읽을 수 없습니다. RLS에 대한 자세한 내용은 행 수준 보안 섹션을 참조하세요.
- COPY 명령을 사용하여 Parquet에서 보호된 대상 테이블로 복사할 때는 COPY 문에 열을 명시적으로 지정해야 합니다. COPY를 사용한 열 매핑에 대한 자세한 내용은 열 매핑 옵션 섹션을 참조하세요.
- DDM 정책은 다음 관계에 연결될 수 없습니다.
  - 시스템 테이블 및 카탈로그
  - 외부 테이블
  - 데이터 공유 테이블
  - 구체화된 뷰
  - 교차 데이터베이스 관계
  - 임시 테이블
  - 상관관계가 있는 쿼리
- DDM 정책에는 조회 테이블이 포함될 수 있습니다. 조회 테이블은 USING 절에 있을 수 있습니다. 다음 관계 유형은 조회 테이블로 사용할 수 없습니다.
  - 시스템 테이블 및 카탈로그
  - 외부 테이블
  - 데이터 공유 테이블
  - 뷰, 구체화된 뷰 및 지연 바인딩 뷰
  - 교차 데이터베이스 관계
  - 임시 테이블
  - 상관관계가 있는 쿼리

  다음은 마스킹 정책을 조회 테이블에 연결하는 예제입니다.

```
--Create a masking policy referencing a lookup table
CREATE MASKING POLICY lookup_mask_credit_card WITH (credit_card TEXT) USING (
  CASE
    WHEN
      credit_card IN (SELECT credit_card_lookup FROM credit_cards_lookup)
    THEN '000000XXXX0000'
    ELSE REDACT_CREDIT_CARD(credit_card)
```

```
    END
  );

--Provides access to the lookup table via a policy attached to a role
GRANT SELECT ON TABLE credit_cards_lookup TO MASKING POLICY lookup_mask_credit_card;
```

- 대상 열의 유형 및 크기와 호환되지 않는 출력을 생성하는 마스킹 정책을 연결할 수 없습니다. 예를 들어 12자 길이의 문자열을 VARCHAR(10) 열에 출력하는 마스킹 정책을 연결할 수 없습니다. Amazon Redshift Redshift는 다음 예외를 지원합니다.
  - 입력 유형이 INTN인 마스킹 정책은 M < N인 한 크기가 INTM인 정책에 연결할 수 있습니다. 예를 들어 BIGINT(INT8) 입력 정책은 smallint(INT4) 열에 연결할 수 있습니다.
  - 입력 유형이 NUMERIC 또는 DECIMAL인 마스킹 정책은 항상 FLOAT 열에 연결할 수 있습니다.
- DDM 정책은 데이터 공유와 함께 사용할 수 없습니다. 데이터 공유의 데이터 생산자가 데이터 공유의 테이블에 DDM 정책을 연결하면, 테이블을 쿼리하려는 데이터 소비자의 사용자는 해당 테이블에 액세스할 수 없게 됩니다. DDM 정책이 연결된 테이블은 데이터 공유에 추가할 수 없습니다.
- 다음 구성 옵션의 값이 세션의 기본값과 일치하지 않으면 DDM 정책을 첨부한 관계를 쿼리할 수 없습니다.
  - `enable_case_sensitive_super_attribute`
  - `enable_case_sensitive_identifier`
  - `downcase_delimited_identifier`

  DDM 정책이 첨부된 관계를 쿼리하려고 할 때 "대/소문자 구분이 기본값과 다른 DDM 보호 관계에서 세션 수준 구성을 지원하지 않습니다."라는 메시지가 표시되면 세션의 구성 옵션을 재설정하는 것이 좋습니다.
- 프로비저닝된 클러스터 또는 서버리스 네임스페이스에 동적 데이터 마스킹 정책이 있는 경우 일반 사용자에게는 다음 명령이 차단됩니다.

  ```
  ALTER <current_user> SET enable_case_sensitive_super_attribute/
  enable_case_sensitive_identifier/downcase_delimited_identifier
  ```

  DDM 정책을 만들 때 일반 사용자의 기본 구성 옵션 설정을 정책을 만들 당시의 세션의 구성 옵션 설정과 일치하도록 변경하는 것이 좋습니다. 슈퍼유저 및 ALTER USER 권한이 있는 사용자는 파라미터 그룹 설정 또는 ALTER USER 명령을 사용하여 이 작업을 수행할 수 있습니다. 파라미터 그룹에 대한 자세한 내용은 Amazon Redshift 관리 안내서의 Amazon Redshift 파라미터 그룹을 참조하세요. ALTER USER 명령에 대한 자세한 내용은 ALTER USER 섹션을 참조하세요.

- DDM 정책이 연결된 뷰와 지연 바인딩 뷰는 일반 사용자가 CREATE VIEW 명령을 사용하여 교체할 수 없습니다. DDM 정책이 연결된 뷰 또는 LBV를 바꾸려면 먼저 연결된 DDM 정책을 분리하고 뷰 또는 LBV를 교체한 다음, 정책을 다시 연결합니다. 슈퍼 사용자 및 `sys:secadmin` 권한이 있는 사용자는 정책을 분리하지 않고도 DDM 정책이 연결된 뷰 또는 LBV에 CREATE VIEW를 사용할 수 있습니다.
- DDM 정책이 연결된 뷰는 시스템 테이블과 뷰를 참조할 수 없습니다. 지연 바인딩 뷰는 시스템 테이블 및 뷰를 참조할 수 있습니다.
- DDM 정책이 연결된 지연 바인딩 뷰는 데이터 레이크의 중첩된 데이터(예: JSON 문서)를 참조할 수 없습니다.
- 지연 바인딩 뷰를 참조하는 뷰가 있는 경우 해당 지연 바인딩 뷰에는 DDM 정책을 연결할 수 없습니다.
- 지연 바인딩 뷰에 연결된 DDM 정책은 열 이름을 기준으로 첨부됩니다. 쿼리 시 Amazon Redshift는 지연 바인딩 뷰에 연결된 모든 마스킹 정책이 성공적으로 적용되었는지, 그리고 지연 바인딩 뷰의 출력 열 유형이 연결된 마스킹 정책의 유형과 일치하는지 확인합니다. 검증이 실패하면 Amazon Redshift가 쿼리에 대한 오류를 반환합니다.
- DDM 정책을 생성할 때 사용자 지정된 세션 컨텍스트 변수를 사용할 수 있습니다. 다음 예제에서는 DDM 정책의 세션 컨텍스트 변수를 설정합니다.

```
-- Set a customized context variable.
SELECT set_config('app.city', 'XXXX', FALSE);

-- Create a MASKING policy using current_setting() to get the value of a customized
 context variable.
CREATE MASKING POLICY city_mask
WITH (city VARCHAR(30))
USING (current_setting('app.city')::VARCHAR(30));

-- Attach the policy on the target table to one or more roles.
ATTACH MASKING POLICY city_mask
ON tickit_users_redshift(city)
TO ROLE analyst, ROLE dbadmin;
```

사용자 지정 세션 컨텍스트 변수를 설정 및 검색하는 방법을 자세히 알아보려면 SET, SET_CONFIG, SET, CURRENT_SETTING, reset 섹션으로 이동하세요. 일반적인 서버 구성 수정 방법을 자세히 알아보려면 서버 구성 수정 섹션으로 이동하세요.

> **Important**
>
> DDM 정책 내에서 세션 컨텍스트 변수를 사용하는 경우 보안 정책은 정책을 간접 호출하는 사용자 또는 역할에 따라 달라집니다. DDM 정책에서 세션 컨텍스트 변수를 사용할 때는 보안 취약성이 발생하지 않도록 주의하세요.

동적 데이터 마스킹 엔드 투 엔드 예제

다음은 마스킹 정책을 생성하고 열에 연결하는 방법을 보여주는 종합 예제입니다. 이러한 정책을 통해 사용자는 역할에 연결된 정책의 난독화 정도에 따라 열에 액세스하고 다른 값을 볼 수 있습니다. 이 예제를 실행하려면 수퍼유저이거나 `sys:secadmin` 역할이 있어야 합니다.

마스킹 정책 생성

먼저 테이블을 만들고 신용 카드 값으로 채웁니다.

```
--create the table
CREATE TABLE credit_cards (
  customer_id INT,
  credit_card TEXT
);

--populate the table with sample values
INSERT INTO credit_cards
VALUES
  (100, '4532993817514842'),
  (100, '4716002041425888'),
  (102, '5243112427642649'),
  (102, '6011720771834675'),
  (102, '6011378662059710'),
  (103, '373611968625635')
;

--run GRANT to grant permission to use the SELECT statement on the table
GRANT SELECT ON credit_cards TO PUBLIC;

--create two users
CREATE USER regular_user WITH PASSWORD '1234Test!';

CREATE USER analytics_user WITH PASSWORD '1234Test!';
```

```
--create the analytics_role role and grant it to analytics_user
--regular_user does not have a role
CREATE ROLE analytics_role;

GRANT ROLE analytics_role TO analytics_user;
```

다음으로 분석 역할에 적용할 마스킹 정책을 생성합니다.

```
--create a masking policy that fully masks the credit card number
CREATE MASKING POLICY mask_credit_card_full
WITH (credit_card VARCHAR(256))
USING ('000000XXXX0000'::TEXT);

--create a user-defined function that partially obfuscates credit card data
CREATE FUNCTION REDACT_CREDIT_CARD (credit_card TEXT)
RETURNS TEXT IMMUTABLE
AS $$
    import re
    regexp = re.compile("^([0-9]{6})[0-9]{5,6}([0-9]{4})")

    match = regexp.search(credit_card)
    if match != None:
        first = match.group(1)
        last = match.group(2)
    else:
        first = "000000"
        last = "0000"

    return "{}XXXXX{}".format(first, last)
$$ LANGUAGE plpythonu;

--create a masking policy that applies the REDACT_CREDIT_CARD function
CREATE MASKING POLICY mask_credit_card_partial
WITH (credit_card VARCHAR(256))
USING (REDACT_CREDIT_CARD(credit_card));

--confirm the masking policies using the associated system views
SELECT * FROM svv_masking_policy;

SELECT * FROM svv_attached_masking_policy;
```

마스킹 정책 연결

마스킹 정책을 신용 카드 테이블에 연결합니다.

```
--attach mask_credit_card_full to the credit card table as the default policy
--all users will see this masking policy unless a higher priority masking policy is
 attached to them or their role
ATTACH MASKING POLICY mask_credit_card_full
ON credit_cards(credit_card)
TO PUBLIC;

--attach mask_credit_card_partial to the analytics role
--users with the analytics role can see partial credit card information
ATTACH MASKING POLICY mask_credit_card_partial
ON credit_cards(credit_card)
TO ROLE analytics_role
PRIORITY 10;

--confirm the masking policies are applied to the table and role in the associated
 system view
SELECT * FROM svv_attached_masking_policy;

--confirm the full masking policy is in place for normal users by selecting from the
 credit card table as regular_user
SET SESSION AUTHORIZATION regular_user;

SELECT * FROM credit_cards;

--confirm the partial masking policy is in place for users with the analytics role by
 selecting from the credit card table as analytics_user
SET SESSION AUTHORIZATION analytics_user;

SELECT * FROM credit_cards;
```

마스킹 정책 변경

다음 섹션은 동적 데이터 마스킹 정책을 변경하는 방법을 설명합니다.

```
--reset session authorization to the default
RESET SESSION AUTHORIZATION;

--alter the mask_credit_card_full policy
ALTER MASKING POLICY mask_credit_card_full
```

```
USING ('00000000000000'::TEXT);

--confirm the full masking policy is in place after altering the policy, and that
 results are altered from '000000XXXX0000' to '00000000000000'
SELECT * FROM credit_cards;
```

마스킹 정책 분리 및 삭제

다음 섹션에서는 테이블에서 모든 동적 데이터 마스킹 정책을 제거하여 마스킹 정책을 분리 및 삭제하는 방법을 보여줍니다.

```
--reset session authorization to the default
RESET SESSION AUTHORIZATION;

--detach both masking policies from the credit_cards table
DETACH MASKING POLICY mask_credit_card_full
ON credit_cards(credit_card)
FROM PUBLIC;

DETACH MASKING POLICY mask_credit_card_partial
ON credit_cards(credit_card)
FROM ROLE analytics_role;

--drop both masking policies
DROP MASKING POLICY mask_credit_card_full;

DROP MASKING POLICY mask_credit_card_partial;
```

범위가 지정된 권한

범위가 지정된 권한을 사용하면 데이터베이스 또는 스키마 내 특정 유형의 모든 객체에 대한 권한을 사용자 또는 역할에 부여할 수 있습니다. 범위가 지정된 권한이 있는 사용자와 역할은 데이터베이스 또는 스키마 내의 모든 현재 및 미래 객체에 대한 지정된 권한을 갖습니다.

SVV_DATABASE_PRIVILEGES에서 데이터베이스 수준 범위 지정 권한의 범위를 볼 수 있습니다. SVV_SCHEMA_PRIVILEGES에서 스키마 수준 범위 지정 권한의 범위를 볼 수 있습니다.

범위가 지정된 권한 적용에 대한 자세한 내용은 GRANT 및 REVOKE 섹션을 참조하세요.

범위가 지정된 권한 사용 시 고려 사항

범위가 지정된 권한을 사용할 때 다음 사항을 고려하세요.

- 범위가 지정된 권한을 사용하여 지정된 사용자 또는 역할에게 데이터베이스 또는 스키마 범위에 대한 권한을 부여하거나 취소할 수 있습니다.
- 범위가 지정된 권한을 사용자 그룹에는 부여할 수 없습니다.
- 범위가 지정된 권한을 부여하거나 취소하면 범위 내 모든 현재 및 미래 객체에 대한 권한이 변경됩니다.
- 범위가 지정된 권한과 객체 수준 권한은 서로 독립적으로 작동합니다. 예를 들어 사용자는 다음 두 경우 모두에서 테이블에 대한 권한을 유지합니다.
  - 사용자에게 schema1.table1 테이블에 대한 SELECT 권한과 schema1에 대한 SELECT 범위 지정 권한이 부여됩니다. 그런 다음 사용자에게 schema1 스키마의 모든 테이블에 대해 SELECT가 취소됩니다. 사용자에게 schema1.table1에 대한 SELECT가 유지됩니다.
  - 사용자에게 schema1.table1 테이블에 대한 SELECT 권한과 schema1에 대한 SELECT 범위 지정 권한이 부여됩니다. 그런 다음 사용자에게 schema1.table1에 대해 SELECT가 취소됩니다. 사용자에게 schema1.table1에 대한 SELECT가 유지됩니다.
- 범위가 지정된 권한을 부여하거나 취소하려면 다음 기준 중 하나를 충족해야 합니다.
  - 슈퍼 사용자
  - 해당 권한에 대한 권한 부여 옵션이 있는 사용자 권한 부여 옵션에 대한 자세한 내용은 GRANT의 WITH GRANT OPTION 파라미터를 참조하세요.
- 범위가 지정된 권한은 연결된 데이터베이스의 객체 또는 데이터 공유에서 가져온 데이터베이스에만 부여하거나 취소할 수 있습니다.
- 범위가 지정된 권한을 사용하여 데이터 공유에서 만든 데이터베이스에 기본 권한을 설정할 수 있습니다. 공유 데이터베이스에 대해 범위가 지정된 권한이 부여된 소비자 측 데이터 공유 사용자는 생산자 측에서 데이터 공유에 추가하는 모든 새 객체에 대해 해당 권한을 자동으로 얻게 됩니다.
- 생산자는 스키마 내 객체에 대한 범위가 지정된 권한을 데이터 공유에 부여할 수 있습니다. (미리 보기)

SQL 참조

Amazon Redshift를 사용하면 SQL을 활용하여 데이터 웨어하우스에 저장된 방대한 양의 데이터를 효율적으로 쿼리하고 분석할 수 있습니다. SQL 참조에서는 SQL 명령, 데이터 유형, 함수, 연산자 등의 구문 및 사용량을 다루므로 인사이트를 추출하고 데이터 기반 의사 결정을 내릴 수 있습니다. 이 참조를 통해 최적화된 쿼리를 작성하고, 데이터베이스 객체를 생성 및 관리하고, 복잡한 데이터 변환을 수행할 수 있습니다. 다음 참조에서는 Amazon Redshift 환경 내에서 SQL을 활용하여 데이터 분석 요구 사항을 충족하는 방법에 대한 포괄적인 세부 정보를 제공합니다.

주제

Amazon Redshift SQL

주제

Amazon Redshift는 업계 표준 SQL을 중심으로 개발되었으며, 여기에 대용량 데이터 집합을 관리할 뿐만 아니라 이러한 데이터에 대한 고성능 분석 및 보고를 지원하는 기능까지 추가되었습니다.

> **Note**
>
> 단일 Amazon Redshift SQL 문의 최대 크기는 16MB입니다.

리더 노드에서 지원되는 SQL 함수

일부 Amazon Redshift 쿼리는 분산을 통해 컴퓨팅 노드에서 실행되는 반면 리더 노드에서만 실행되는 쿼리도 있습니다.

리더 노드는 쿼리가 사용자 생성 테이블 또는 시스템 테이블(STL 또는 STV 접두사가 첨부된 테이블과 SVL 또는 SVV 접두사가 첨부된 시스템 뷰)을 참조할 때마다 SQL을 컴퓨팅 노드로 분산시킵니다. 카탈로그 테이블(PG_TABLE_DEF처럼 PG 접두사가 첨부되어 리더 노드에 저장되는 테이블)만 참조하거나 어떤 테이블도 참조하지 않는 쿼리는 리더 노드에서만 실행됩니다.

일부 Amazon Redshift SQL 함수는 리더 노드에서만 지원되고, 컴퓨팅 노드에서는 지원되지 않습니다. 따라서 리더 노드 함수를 사용하는 쿼리는 컴퓨팅 노드가 아닌 리더 노드에서만 실행해야 합니다. 그렇지 않으면 오류를 반환합니다.

리더 노드에서만 실행해야 하는 각 함수의 설명서에는 이 함수가 사용자 정의 테이블이나 Amazon Redshift 시스템 테이블을 참조할 경우 오류를 반환한다는 설명이 포함되어 있습니다. 리더 노드에서만 실행해야 하는 함수 목록은 리더 노드 전용 함수 섹션을 참조하세요.

예시

다음 예제에서는 샘플 TICKIT 데이터베이스를 사용합니다. 샘플 데이터베이스에 대한 자세한 내용은 샘플 데이터베이스 섹션을 참조하세요.

CURRENT_SCHEMA

CURRENT_SCHEMA 함수는 리더 노드 전용 함수입니다. 다음 예에서는 쿼리가 테이블을 참조하지 않기 때문에 리더 노드에서만 실행됩니다.

```
select current_schema();

current_schema
---------------
public
```

다음 예에서는 쿼리가 시스템 카탈로그 테이블을 참조하기 때문에 리더 노드에서만 실행됩니다.

```
select * from pg_table_def
where schemaname = current_schema() limit 1;

 schemaname | tablename | column |   type   | encoding | distkey | sortkey | notnull
------------+-----------+--------+----------+----------+---------+---------+---------
 public     | category  | catid  | smallint | none     | t       |       1 | t
```

다음 예에서는 쿼리가 컴퓨팅 노드에 저장되는 Amazon Redshift 시스템 테이블을 참조하기 때문에 오류를 반환합니다.

```
select current_schema(), userid from users;

INFO:  Function "current_schema()" not supported.
ERROR:  Specified types or functions (one per INFO message) not supported on Amazon
 Redshift tables.
```

SUBSTR

SUBSTR 또한 리더 노드 전용 함수입니다. 다음 예에서는 쿼리가 테이블을 참조하지 않기 때문에 리더 노드에서만 실행됩니다.

```
SELECT SUBSTR('amazon', 5);

+--------+
| substr |
+--------+
| on     |
+--------+
```

다음 예에서는 쿼리가 컴퓨팅 노드에 있는 테이블을 참조합니다. 이 코드를 실행하면 오류가 발생합니다.

```
SELECT SUBSTR(catdesc, 1) FROM category LIMIT 1;

ERROR: SUBSTR() function is not supported (Hint: use SUBSTRING instead)
```

이전 쿼리를 성공적으로 실행하려면 SUBSTRING을 사용하세요.

```
SELECT SUBSTRING(catdesc, 1) FROM category LIMIT 1;

+---------------------------------+
|            substring            |
+---------------------------------+
| National Basketball Association |
+---------------------------------+
```

FACTORIAL()

FACTORIAL()은 리더 노드 전용 함수입니다. 다음 예에서는 쿼리가 테이블을 참조하지 않기 때문에 리더 노드에서만 실행됩니다.

```
SELECT FACTORIAL(5);

 factorial
-------------
 120
```

다음 예에서는 쿼리가 컴퓨팅 노드에 있는 테이블을 참조합니다. 쿼리 에디터 v2를 사용하여 실행하면 오류가 발생합니다.

```
create table t(a int);
insert into t values (5);
select factorial(a) from t;

ERROR: Specified types or functions (one per INFO message) not supported on Redshift
 tables.
Info: Function "factorial(bigint)" not supported.
```

Amazon Redshift 및 PostgreSQL

주제

- Amazon Redshift와 PostgreSQL JDBC 및 ODBC
- 다르게 구현되는 기능
- 지원되지 않는 PostgreSQL 기능
- 지원되지 않는 PostgreSQL 데이터 유형
- 지원되지 않는 PostgreSQL 함수

Amazon Redshift는 PostgreSQL을 기반으로 합니다. Amazon Redshift와 PostgreSQL은 데이터웨어하우스 애플리케이션을 설계하고 개발할 때 숙지해야 할 몇 가지 매우 중요한 차이점이 있습니다.

Amazon Redshift는 대용량 데이터 집합을 대상으로 복잡한 쿼리가 필요한 온라인 분석 처리(OLAP) 또는 비즈니스 인텔리전스(BI) 애플리케이션 전용으로 설계되었습니다. 해결하는 요건도 매우 다양하기 때문에 Amazon Redshift의 전용 데이터 스토리지 스키마 및 쿼리 실행 엔진도 PostgreSQL 구현체와 완전히 다릅니다. 예를 들어 OLTP(온라인 트랜잭션 처리) 애플리케이션이 일반적으로 데이터를 행에 저장하는 반면 Amazon Redshift는 최적의 메모리 사용 및 디스크 I/O를 위한 특수 데이터 압축 인코딩을 사용하여 데이터를 열에 저장합니다. 보조 인덱스 및 효율적인 단일 행 데이터 조작 작업과 같이 소규모 OLTP 처리에 적합한 일부 PostgreSQL 기능은 성능 향상을 위해 생략되었습니다.

Amazon Redshift 데이터 웨어하우스 시스템 아키텍처에 대한 자세한 내용은 Amazon Redshift 아키텍처 섹션을 참조하세요.

PostgreSQL 9.x에는 Amazon Redshift에서 지원되지 않는 기능이 일부 포함되어 있습니다. 그 밖에도 Amazon Redshift SQL과 PostgreSQL 사이에는 반드시 알고 있어야 할 중요한 차이점들이 있습니다. 이번 섹션에서는 Amazon Redshift와 PostgreSQL의 차이점에 대해 살펴보고, Amazon Redshift SQL 구현체를 최대한 이용하여 데이터 웨어하우스를 개발할 수 있는 지침까지 제공합니다.

Amazon Redshift와 PostgreSQL JDBC 및 ODBC

Amazon Redshift는 PostgreSQL을 기반으로 하기 때문에 이전에는 JDBC4 Postgresql 드라이버 버전 8.4.703 및 psqlODBC 버전 9.x 드라이버 사용이 권장되었습니다. 현재 이러한 드라이버를 사용하는 경우 앞으로 새로운 Amazon Redshift 전용 드라이버로 옮기는 것이 좋습니다. 드라이버 및 연결 구성에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 Amazon Redshift JDBC 또는 ODBC 드라이버 섹션을 참조하세요.

JDBC를 사용하여 대용량 데이터 세트를 가져올 때 클라이언트 측 메모리 부족 오류를 방지하려면 클라이언트에서 JDBC fetch size 파라미터를 설정하여 데이터를 일괄적으로 가져올 수 있습니다. 자세한 내용은 JDBC Fetch Size 파라미터 설정 단원을 참조하십시오.

Amazon Redshift는 JDBC maxRows 파라미터를 인식하지 못합니다. 따라서 결과 집합을 제한할 때는 LIMIT 절을 지정하십시오. 또한 OFFSET 절을 사용하여 결과 집합에서 특정 시작점으로 건너뛸 수도 있습니다.

다르게 구현되는 기능

Amazon Redshift SQL 언어 요소는 서로 성능 특성이 다를 뿐만 아니라 상응하는 PostgreSQL 구현체와 완전히 다른 구문과 의미를 사용합니다.

> **Important**
>
> Amazon Redshift와 PostgreSQL의 공통 요소라고 해서 의미가 동일할 것이라고 미리 짐작해서는 안 됩니다. 반드시 Amazon Redshift 개발자 안내서 SQL 명령 섹션을 참조하여 미묘한 차이가 무엇인지 확인하세요.

한 가지 특정한 예가 테이블을 정리하여 재구성할 때 사용되는 VACUUM 명령입니다. VACUUM은 PostgreSQL 버전과 다르게 동작할 뿐만 아니라 사용하는 파라미터 집합도 다릅니다. Amazon Redshift에서 VACUUM 사용에 대한 자세한 내용은 테이블 Vacuum 섹션을 참조하세요.

종종 데이터베이스 관리 및 운영 기능과 도구도 다릅니다. 예를 들어 Amazon Redshift는 시스템 작동 방식에 대한 정보가 저장되어 있는 시스템 테이블 및 뷰가 많습니다. 자세한 정보는 SYS 모니터링 뷰을 참조하세요.

다음은 Amazon Redshift에서 다르게 구현되는 SQL 기능 중 몇 가지 예입니다.

- CREATE TABLE

  Amazon Redshift는 테이블스페이스, 테이블 분할, 상속 및 일부 제약 조건을 지원하지 않습니다. CREATE TABLE 기능을 Amazon Redshift에서 구현하면 테이블을 정렬 및 분산 알고리즘을 정의하여 병렬 처리를 최적화할 수 있습니다.

  Amazon Redshift Spectrum은 CREATE EXTERNAL TABLE 명령을 사용하여 테이블 파티셔닝을 지원합니다.

- ALTER TABLE

  ALTER COLUMN 작업의 하위 집합만 지원됩니다.

  ADD COLUMN은 각 ALTER TABLE 문마다 열을 하나만 추가할 수 있도록 지원합니다.

- COPY

  Amazon Redshift COPY 명령은 Amazon S3 버킷 및 Amazon DynamoDB 테이블에서 데이터를 로드하여 자동 압축을 지원할 수 있도록 고도로 특화되었습니다. 자세한 내용은 Amazon Redshift에서 데이터 로드 및 COPY 명령 참조를 참조하십시오.

- VACUUM

  VACUUM 파라미터는 완전히 다릅니다. 예를 들어 PostgreSQL에서는 기본적으로 VACUUM 작업이 공간을 재사용할 수 있도록 회수하는 데 그치는 반면에서 Amazon Redshift는 VACUUM FULL이 기본 VACUUM 작업으로서 디스크 공간을 회수한 후 모든 행을 재정렬합니다.

- 문자열 값을 서로 비교할 때 VARCHAR 값의 후행 공백은 무시됩니다. 자세한 내용은 후행 공백의 중요성 단원을 참조하십시오.

지원되지 않는 PostgreSQL 기능

다음 PostgreSQL 기능은 Amazon Redshift에서 지원되지 않습니다.

> **Important**
>
> Amazon Redshift와 PostgreSQL의 공통 요소라고 해서 의미가 동일할 것이라고 미리 짐작해서는 안 됩니다. 반드시 Amazon Redshift 개발자 안내서 SQL 명령 섹션을 참조하여 미묘한 차이가 무엇인지 확인하세요.

- 쿼리 도구 psql은 지원되지 않습니다. Amazon Redshift RSQL 클라이언트가 지원됩니다.
- 테이블 분할(범위 및 목록 분할)
- 테이블스페이스
- 제약 조건
  - 고유
  - 외래 키
  - 프라이머리 키
  - 제약 조건 확인
  - 예외 제약 조건

  고유성, 기본 키 및 외래 키 제약 조건은 허용되지만 오직 참고용입니다. 따라서 시스템에서 적용되지는 않지만 쿼리 플래너에서 사용할 수는 있습니다.
- 상속
- PostgreSQL 시스템 열

  Amazon Redshift SQL은 시스템 열을 묵시적으로 정의하지 않습니다. 하지만 다음 PostgreSQL 시스템 열 이름은 사용자 정의 열인 `oid`, `tableoid`, `xmin`, `cmin`, `xmax`, `cmax`, `ctid`의 이름으로 사용할 수 없습니다. 자세한 내용은 https://www.postgresql.org/docs/8.0/static/ddl-system-columns.html에서 확인할 수 있습니다.
- 인덱스
- 창 함수의 NULLS 절
- 콜레이션

  Amazon Redshift는 로케일 또는 사용자 정의 데이터 정렬 순서를 지원하지 않습니다. 콜레이션 시퀀스 섹션을 참조하세요.
- 값 표현식
  - 구독 표현식

  - 배열 생성자
  - 행 생성자
- 트리거
- 외부 데이터 관리(SQL/MED)
- 테이블 함수
- 상수 테이블로 사용되는 VALUES 목록
- 시퀀스
- 시퀀스
- RULE 및 TRIGGER 권한

  Amazon Redshift는 GRANT ALL 또는 REVOKE ALL를 실행할 때 이러한 권한을 부여하거나 취소하지만, RULE 및 TRIGGER 권한의 유무는 피부여자의 액세스 권한에 어떤 식으로든 영향을 미치지 않습니다.

지원되지 않는 PostgreSQL 데이터 유형

일반적으로 쿼리가 명시적 또는 묵시적 변환을 포함하여 지원되지 않는 데이터 형식을 사용하려고 하면 오류가 반환됩니다. 하지만 지원되지 않는 데이터 형식을 사용하는 쿼리 중에는 컴퓨팅 노드가 아닌 리더 노드에서 실행되는 쿼리도 있습니다. 리더 노드에서 지원되는 SQL 함수 섹션을 참조하세요.

지원되는 데이터 형식의 전체 목록은 데이터 타입 섹션을 참조하세요.

다음 PostgreSQL 데이터 형식은 Amazon Redshift에서 지원되지 않습니다.

- 배열
- BIT, BIT VARYING
- BYTEA
- 복합 형식
- 열거 형식
- 기하학적 유형(기하학적 유형의 Amazon Redshift 구현은 PostgreSQL과 다름)
- HSTORE
- JSON
- 네트워크 주소 형식

- 숫자형
  - SERIAL, BIGSERIAL, SMALLSERIAL
  - MONEY
- 객체 식별자 형식
- 의사 형식
- 범위 형식
- 특수 문자 유형
  - "char" - 단일 바이트 내부 유형(여기서 char라는 데이터 형식은 인용 부호로 묶입니다).
  - 이름 - 객체 이름의 내부 형식.

  이러한 유형에 대한 자세한 내용은 PostgreSQL 설명서의 Special Character Types를 참조하십시오.
- 텍스트 검색 형식
- TXID_SNAPSHOT
- UUID
- XML

지원되지 않는 PostgreSQL 함수

여기에서 언급하는 함수들도 의미 또는 사용법이 서로 다릅니다. 예를 들어 지원되는 함수 중 일부는 리더 노드에서만 실행됩니다. 또한 지원되지 않는 함수라고 해도 일부는 리더 노드에서 실행할 경우 오류를 반환하지 않습니다. 이러한 함수들이 일부 경우 오류를 반환하지 않는다고 해서 Amazon Redshift에서 지원되는 함수라고 간주해서는 안 됩니다.

> **Important**
>
> Amazon Redshift와 PostgreSQL의 공통 요소라고 해서 의미가 동일할 것이라고 미리 짐작해서는 안 됩니다. 반드시 Amazon Redshift 데이터베이스 개발자 안내서 SQL 명령 섹션을 참조하여 미묘한 차이가 무엇인지 확인하세요.

자세한 내용은 리더 노드에서 지원되는 SQL 함수 단원을 참조하십시오.

다음 PostgreSQL 함수는 Amazon Redshift에서 지원되지 않습니다.

- 액세스 권한 조회 함수

- 권고 잠금 함수
- 집계 함수
  - STRING_AGG()
  - ARRAY_AGG()
  - EVERY()
  - XML_AGG()
  - CORR()
  - COVAR_POP()
  - COVAR_SAMP()
  - REGR_AVGX(), REGR_AVGY()
  - REGR_COUNT()
  - REGR_INTERCEPT()
  - REGR_R2()
  - REGR_SLOPE()
  - REGR_SXX(), REGR_SXY(), REGR_SYY()
- 배열 함수와 연산자
- 백업 제어 함수
- 주석 정보 함수
- 데이터베이스 객체 위치 함수
- 데이터베이스 객체 크기 함수
- 날짜/시간 함수 및 연산자
  - CLOCK_TIMESTAMP()
  - JUSTIFY_DAYS(), JUSTIFY_HOURS(), JUSTIFY_INTERVAL()
  - PG_SLEEP()
  - TRANSACTION_TIMESTAMP()
- ENUM 지원 함수
- 기하 함수 및 연산자
- 제네릭 파일 액세스 함수
- IS DISTINCT FROM
- 네트워크 주소 함수 및 연산자

- 수학 함수
  - DIV()
  - SETSEED()
  - WIDTH_BUCKET()
- 설정 반환 함수
  - GENERATE_SERIES()
  - GENERATE_SUBSCRIPTS()
- 범위 함수 및 연산자
- 복구 제어 함수
- 복구 정보 함수
- ROLLBACK TO SAVEPOINT 함수
- 스키마 가시성 조회 함수
- 서버 신호 전송 함수
- 스냅샷 동기화 함수
- 시퀀스 조작 함수
- 문자열 함수
  - BIT_LENGTH()
  - OVERLAY()
  - CONVERT(), CONVERT_FROM(), CONVERT_TO()
  - ENCODE()
  - FORMAT()
  - QUOTE_NULLABLE()
  - REGEXP_MATCHES()
  - REGEXP_SPLIT_TO_ARRAY()
  - REGEXP_SPLIT_TO_TABLE()
- 시스템 카탈로그 정보 함수
- 시스템 정보 함수
  - CURRENT_CATALOG CURRENT_QUERY()
  - INET_CLIENT_ADDR()
  - INET_CLIENT_PORT()

  - INET_SERVER_ADDR() INET_SERVER_PORT()
  - PG_CONF_LOAD_TIME()
  - PG_IS_OTHER_TEMP_SCHEMA()
  - PG_LISTENING_CHANNELS()
  - PG_MY_TEMP_SCHEMA()
  - PG_POSTMASTER_START_TIME()
  - PG_TRIGGER_DEPTH()
  - SHOW VERSION()
- 텍스트 검색 함수 및 연산자
- 트랜잭션 ID 및 스냅샷 함수
- 트리거 함수
- XML 함수

SQL 사용

주제

- SQL 참조 규칙
- 기본 요소
- Expressions
- 조건

SQL 언어는 데이터베이스 및 데이터베이스 객체 작업 시 사용하는 명령과 함수로 구성되어 있습니다. 또한 데이터 형식, 표현식 및 리터럴의 사용에 대한 규칙을 적용하기도 합니다.

SQL 참조 규칙

이번 단원에서는 SQL 참조 단원에서 설명한 SQL 표현식, 명령 및 함수의 구문을 작성하는 데 사용되는 규칙에 대해 설명합니다.

문자	설명
CAPS	대문자로 된 단어는 키워드입니다.

문자	설명
[ ]	대괄호는 선택적 인수를 나타냅니다. 다수의 인수가 대괄호로 묶이면 인수를 얼마든지 선택할 수 있다는 것을 의미합니다. 또한 별도의 라인에서 대괄호로 묶이는 인수는 Amazon Redshift 구문 분석기에서 인수가 구문에 나열된 순서를 그대로 따른다는 것을 의미합니다. 예시는 SELECT을 확인하세요.
{ }	중괄호는 중괄호 안의 인수 중 하나를 선택해야 함을 나타냅니다.
|	파이프는 인수 중에서 하나를 선택할 수 있다는 것을 의미합니다.
기울임꼴	기울임꼴 단어는 자리 표시자를 나타냅니다. 기울임꼴 단어 자리에 적절한 값을 넣어야 합니다.
. . .	줄임표는 앞의 요소를 반복할 수 있음을 나타냅니다.
'	작은따옴표로 묶인 단어는 따옴표를 입력해야 함을 나타냅니다.

기본 요소

주제

- 이름 및 식별자
- 리터럴
- NULL
- 데이터 타입
- 콜레이션 시퀀스

이번 단원에서는 데이터베이스 객체 이름, 리터럴, NULL, 데이터 형식 등의 작업 규칙에 대해서 설명합니다.

이름 및 식별자

이름은 사용자와 암호 외에도 테이블이나 열 같은 데이터베이스 객체를 식별하는 역할을 합니다. 이름과 식별자는 서로 통용되는 용어입니다. 식별자는 표준 식별자와 인용 또는 구분 식별자, 2가지 유형이 있습니다. 또한 인쇄 가능한 UTF-8 문자로만 구성되어야 합니다. 표준 및 구분 식별자의 ASCII 문자는 대/소문자를 구분하지 않고 데이터베이스에서 모두 소문자로 변환됩니다. 쿼리 결과에서 열 이름은

기본적으로 소문자로 반환됩니다. 열 이름을 대문자로 반환하려면 describe_field_name_in_uppercase 구성 파라미터를 **true**로 설정하십시오.

표준 식별자

표준 SQL 식별자는 규칙 집합을 준수하고, 다음과 같이 따라야 합니다.

- ASCII 단일 바이트 알파벳 문자 또는 밑줄 문자나, 2~4바이트 길이의 UTF-8 멀티바이트 문자로 시작합니다.
- 후속 문자는 ASCII 단일 바이트 영숫자 문자, 밑줄 또는 달러 기호, 혹은 2~4바이트 길이의 UTF-8 멀티바이트 문자가 될 수 있습니다.
- 길이는 1~127바이트가 되어야 하며, 여기에 구분 식별자의 인용 부호는 포함되지 않습니다.
- 인용 부호나 공백이 포함되어서는 안 됩니다.
- 예약된 SQL 키워드가 되어서는 안 됩니다.

구분 식별자

구분 식별자(인용 식별자로도 불림)는 큰 따옴표(")로 시작해서 끝납니다. 구분 식별자를 사용하는 경우에는 해당하는 객체 참조마다 큰 따옴표를 사용해야 합니다. 이 식별자에는 큰따옴표 외에 인쇄 가능한 표준 UTF-8 문자라면 무엇이든 포함될 수 있습니다. 따라서 그 밖에 공백이나 퍼센트 기호 같이 잘못된 문자까지 포함한 열 또는 테이블 이름도 생성할 수 있습니다.

구분 식별자의 ASCII 문자는 대/소문자를 구분하지 않고 모두 소문자로 변환됩니다. 문자열에 큰따옴표를 사용하려면 다른 큰따옴표를 앞에 입력해야 합니다.

대/소문자 구분 식별자

대/소문자 구분 식별자(대/소문자 혼합 식별자라고도 함)에는 대문자와 소문자가 모두 포함될 수 있습니다. 대/소문자 구분 식별자를 사용하려면 구성 `enable_case_sensitive_identifier`를 `true`로 설정합니다. 클러스터나 세션에 대해 이 구성을 설정할 수 있습니다. 자세한 내용은 enable_case_sensitive_identifier 및 Amazon Redshift 관리 가이드의 기본 파라미터 값 섹션을 참조하세요.

시스템 열 이름

다음 PostgreSQL 시스템 열 이름은 사용자 정의 열의 열 이름으로 사용할 수 없습니다. 자세한 내용은 https://www.postgresql.org/docs/8.0/static/ddl-system-columns.html에서 확인할 수 있습니다.

- `oid`

- tableoid
- xmin
- cmin
- xmax
- cmax
- ctid

예시

다음 표에 구분 식별자의 예, 출력 결과 및 설명이 나와 있습니다.

구문	결과	토론
"그룹"	그룹	GROUP은 예약어이기 때문에 식별자에 사용하기 위해서는 큰따옴표가 필요합니다.
"""WHERE"""	"""WHERE"""	WHERE 역시 예약어입니다. 문자열에 인용 부호를 포함하려면 큰따옴표를 한 번 더 입력하여 큰따옴표를 각각 이스케이프 처리합니다.
"This name"	this name	공백을 그대로 유지하기 위해서는 큰따옴표가 필요합니다.
"This ""IS IT"""	this "is it"	IS IT을 묶는 인용 부호가 이름의 일부가 되기 위해서는 각각 별도의 인용 부호 뒤에 입력되어야 합니다.

다음은 이 "is it"이라는 열을 포함하여 group이라는 이름의 테이블을 생성하는 예입니다.

```
create table "group" (
"This ""IS IT""" char(10));
```

다음은 동일한 결과를 반환하는 쿼리입니다.

```
select "This ""IS IT"""
from "group";

this "is it"
--------------
```

```
(0 rows)
```

```
select "this ""is it"""
from "group";

this "is it"
--------------
(0 rows)
```

다음은 동일한 결과를 반환하도록 정규화된 `table.column` 구문입니다.

```
select "group"."this ""is it"""
from "group";

this "is it"
--------------
(0 rows)
```

다음은 열 이름에 슬래시가 있는 테이블을 생성하는 CREATE TABLE 명령입니다.

```
create table if not exists city_slash_id(
                  "city/id" integer not null,
                  state char(2) not null);
```

리터럴

리터럴 또는 상수는 연속된 문자 또는 숫자 상수로 구성된 데이터 고정 값입니다. Amazon Redshift는 다음과 같은 여러 가지 형식의 리터럴을 지원합니다.

- 숫자 리터럴(정수, 소수, 부동 소수점 수) 자세한 내용은 정수 및 부동 소수점 리터럴 단원을 참조하십시오.
- 문자 리터럴(문자열, 문자 문자열 또는 문자 상수)
- 날짜/시간 및 간격 리터럴(날짜/시간 데이터 형식과 함께 사용됨) 자세한 내용은 날짜, 시간 및 타임스탬프 리터럴 및 간격 데이터 유형 및 리터럴 단원을 참조하세요.

NULL

행의 열이 누락되었거나, 알 수 없거나, 혹은 적용되지 않는 경우에는 NULL 값이거나, NULL을 포함한다고 얘기합니다. NULL은 모든 데이터 형식에서 기본 키 또는 NOT NULL 제약 조건의 제한을 받지 않는 필드에 나타날 수 있습니다. 0의 값이나 빈 문자열하고는 다릅니다.

NULL이 포함된 산술 표현식은 항상 NULL로 평가됩니다. NULL 인수 또는 피연산자가 지정되면 연결 연산자(concatenation)를 제외한 모든 연산자가 NULL을 반환합니다.

NULL 유무를 테스트할 때는 비교 조건인 IS NULL과 IS NOT NULL을 사용합니다. NULL은 데이터 부족을 의미하기 때문에 임의의 값이나 다른 NULL과 같을 수는 없습니다.

데이터 타입

주제

- 멀티바이트 문자
- 숫자형
- 문자 형식
- 날짜/시간 형식
- 부울 유형
- HLLSKETCH 형식
- SUPER 형식
- VARBYTE 형식
- 형식 호환성 및 변환

Amazon Redshift가 저장하거나 가져오는 값은 각각 고정적으로 연결된 속성 집합이 포함된 데이터 형식을 가지고 있습니다. 데이터 형식은 테이블을 생성할 때 선언됩니다. 열이나 인수에 포함될 수 있는 값도 이 데이터 형식에 따라 결정됩니다.

다음 표는 Amazon Redshift 테이블에서 사용할 수 있는 데이터 형식을 나열한 것입니다.

데이터 유형	에일리어스	설명
SMALLINT	INT2	2바이트 부호화 정수
INTEGER	INT, INT4	4바이트 부호화 정수

데이터 유형	에일리어스	설명
BIGINT	INT8	8바이트 부호화 정수
DECIMAL	NUMERIC	정밀도를 선택할 수 있는 정확한 숫자
REAL	FLOAT4	단정밀도 부동 소수점 수
DOUBLE PRECISION	FLOAT8, FLOAT	배정밀도 부동 소수점 수
CHAR	CHARACTER, NCHAR, BPCHAR	고정 길이 문자열
VARCHAR	CHARACTER VARYING, NVARCHAR, TEXT	사용자 정의 제한이 포함된 가변 길이 문자열
날짜		날짜(년, 월, 일)
TIME	TIME WITHOUT TIME ZONE	Time of day
TIMETZ	TIME WITH TIME ZONE	Time of day with time zone
TIMESTAMP	TIMESTAMP WITHOUT TIME ZONE	날짜/시간(시간대 제외)
TIMESTAMPTZ	TIMESTAMP(시간대 사용)	날짜/시간(시간대 포함)
INTERVAL YEAR TO MONTH		년-월 단위의 기간
INTERVAL DAY TO SECOND		일-초 단위의 기간
BOOLEAN	BOOL	논리적 부울(true/false)
HLLSKETCH		HyperLogLog 스케치와 함께 사용되는 형식입니다.

데이터 유형	에일리어스	설명
SUPER		ARRAY 및 STRUCTS와 같은 복합 유형을 포함하여 Amazon Redshift의 모든 스칼라 유형을 포괄하는 상위 집합 데이터 유형입니다.
VARBYTE	VARBINARY, BINARY VARYING	가변 길이 이진 값
GEOMETRY		공간 데이터
GEOGRAPHY		공간 데이터

> **Note**
>
> “char”(char는 인용 부호로 묶임) 등 지원되지 않는 데이터 형식에 대한 자세한 내용은 지원되지 않는 PostgreSQL 데이터 유형 섹션을 참조하세요.

멀티바이트 문자

VARCHAR 데이터 형식은 최대 4바이트의 UTF-8 멀티바이트 문자를 지원합니다. 5바이트 이상의 문자는 지원되지 않습니다. 멀티바이트 문자가 포함된 VARCHAR 열의 크기를 계산하려면 문자 수를 문자당 바이트 수와 곱셈합니다. 예를 들어 문자열에 한자가 4개 포함되어 있고, 각 문자의 길이가 3바이트라면 문자열을 저장하는 데 VARCHAR(12) 열이 필요합니다.

VARCHAR 데이터 유형은 다음과 같이 잘못된 UTF-8 코드포인트를 지원하지 않습니다.

`0xD800 - 0xDFFF`(바이트 시퀀스: `ED A0 80~ED BF BF`)

CHAR 데이터 유형은 멀티바이트 문자를 지원하지 않습니다.

숫자형

주제

- 정수 형식
- DECIMAL 또는 NUMERIC 형식

숫자 데이터 형식으로는 정수, 소수 및 부동 소수점 수가 있습니다.

정수 형식

SMALLINT, INTEGER 및 BIGINT 데이터 형식을 사용하여 다양한 범위의 정수를 저장합니다. 각 형식마다 허용 범위를 벗어나는 값은 저장할 수 없습니다.

명칭	스토리지	Range
SMALLINT 또는 INT2	2 bytes	2 bytes
INTEGER, INT 또는 INT4	4 bytes	4 bytes
BIGINT 또는 INT8	8 bytes	8 bytes

DECIMAL 또는 NUMERIC 형식

소수 또는 숫자 데이터 형식을 사용하여 사용자 정의 정밀도가 포함된 값을 저장합니다. 여기에서 소수와 숫자 키워드는 동일한 의미로 통용됩니다. 하지만 본 문서에서는 소수가 이 데이터 형식에서 우선적으로 사용되는 용어입니다. 실제로 숫자는 일반적으로 정수, 소수 및 부동 소수점 데이터 형식을 일컬을 때 사용됩니다.

스토리지	Range
가변적, 비압축 소수 형식인 경우 최대 128비트	최대 38자리 정밀도의 128비트 부호화 정수

테이블에서 DECIMAL 열은 precision과 scale을 지정하여 다음과 같이 정의합니다.

```
decimal(precision, scale)
```

precision

정수에서 전체 유효 자릿수, 즉 소수점 양변의 자릿수를 말합니다. 예를 들어 숫자 48.2891의 정밀도는 6이고, 소수점 자릿수는 4입니다. 정밀도를 따로 지정하지 않을 경우 기본 정밀도는 18입니다. 최대 정밀도는 38입니다.

입력 값에서 소수점 왼쪽의 자릿수가 열 정밀도에서 소수점 자릿수를 뺀 값보다 큰 경우에는 값을 열에 복사하거나, 삽입하거나 혹은 업데이트할 수 없습니다. 이 규칙은 열 정의의 범위를 벗어나는 모든 값에 적용됩니다. 예를 들어 numeric(5,2) 열에서는 허용되는 값의 범위가 -999.99~999.99입니다.

사용

값의 소수부, 즉 소수점 오른쪽의 소수 자릿수를 말합니다. 정수는 소수 자릿수가 0입니다. 열 명세에서 소수점 자릿수 값은 정밀도 값보다 작거나 같아야 합니다. 소수점 자릿수를 따로 지정하지 않을 경우 기본 소수점 자릿수는 18입니다. 최대 소수점 자릿수는 37입니다.

테이블에 로드되는 입력 값의 소수점 자릿수가 열의 소수점 자릿수보다 큰 경우에는 값이 지정한 자릿수로 반올림됩니다. 예를 들어 SALES 테이블의 PRICEPAID 열이 DECIMAL(8,2) 열이라고 가정하겠습니다. 이때 DECIMAL(8,4) 값이 PRICEPAID 열에 삽입되면 값의 소수점 자릿수가 2로 반올림됩니다.

```
insert into sales
values (0, 8, 1, 1, 2000, 14, 5, 4323.8951, 11.00, null);

select pricepaid, salesid from sales where salesid=0;

pricepaid | salesid
-----------+---------
4323.90 |       0
(1 row)
```

하지만 테이블에서 선택한 값의 명시적인 변환 결과는 반올림되지 않습니다.

> **Note**
>
> DECIMAL(19,0) 열에 삽입할 수 있는 최대 양의 값은 9223372036854775807(2^{63} -1)입니다. 음의 최댓값은 -9223372036854775808입니다. 예를 들어 9999999999999999999(919개) 값을 삽입하려고 하면 오버플로우 오류가 발생합니다. 소수점 위치에 상관없이 Amazon Redshift에서 DECIMAL 숫자로 표현할 수 있는 가장 큰 문자열은

9223372036854775807입니다. 예를 들어 DECIMAL(19,18) 열에 로드할 수 있는 가장 큰 값은 9.223372036854775807입니다.
유효 자릿수가 19자리 이하인 DECIMAL 값은 내부적으로 8바이트 정수로 저장되지만, 유효 자릿수가 20~38자리인 DECIMAL 값은 16바이트 정수로 저장되기 때문에 이러한 규칙이 적용됩니다.

128비트 DECIMAL 또는 NUMERIC 열 사용에 대한 주의 사항

애플리케이션에 해당 전체 자릿수가 필요한지 확실하지 않은 경우 DECIMAL 열에 최대 전체 자릿수를 임의로 지정하지 않도록 합니다. 128비트 값은 64비트 값보다 두 배 많은 디스크 공간을 사용하므로 쿼리 실행 시간이 느려질 수 있습니다.

부동 소수점 형식

REAL 및 DOUBLE PRECISION 데이터 형식을 사용하여 가변 정밀도의 숫자 값을 저장합니다. 부동 소수점 형식은 부정확합니다. 이 말은 일부 값이 근사치로 저장되어 특정 값을 저장하거나 반환할 때 약간 불일치가 발생할 수 있다는 것을 의미합니다. 따라서 정확한 저장 및 계산이 필요하다면(금전적 액수 등) DECIMAL 데이터 형식을 사용하십시오.

REAL은 이진 부동 소수점 산술에 대한 IEEE 표준 754에 따른 단정밀도 부동 소수점 형식을 나타냅니다. 정밀도는 약 6자리이며 범위는 약 1E-37~1E+37입니다. 이 데이터 유형을 FLOAT4로 지정할 수도 있습니다.

DOUBLE PRECISION은 이진 부동 소수점 산술에 대한 IEEE 표준 754에 따른 배정밀도 부동 소수점 형식을 나타냅니다. 정밀도는 약 15자리이며 범위는 약 1E-307~1E+308입니다. 이 데이터 유형을 FLOAT 또는 FLOAT8로 지정할 수도 있습니다.

부동 소수점 유형에는 일반 숫자 값 외에도 몇 가지 특수 값이 있습니다. SQL에서 이러한 값을 사용할 때는 다음 값 주위에 작은따옴표를 사용하세요.

- `NaN` – 숫자가 아님
- `Infinity` - 무한대
- `-Infinity` - 음의 무한대

예를 들어, `customer_activity` 테이블의 `day_charge` 열에 숫자가 아닌 항목을 삽입하려면 다음 SQL을 실행합니다.

```
insert into customer_activity(day_charge) values('NaN');
```

숫자 값 계산

여기에서 계산이란 더하기, 빼기, 곱하기, 나누기 등 수학적인 이진 연산을 말합니다. 이번 단원에서는 이러한 연산에 따라 예상되는 반환 형식을 비롯해 DECIMAL 데이터 형식이 포함되어 있을 경우 정밀도와 소수점 자릿수의 계산 공식에 대해서 설명합니다.

쿼리 처리 시 숫자 값을 계산할 때는 계산이 불가능하거나 쿼리가 숫자 오버플로우 오류를 반환하는 상황이 발생할 수 있습니다. 그 밖에도 계산된 값의 소수점 자릿수가 바뀌거나 예상과 다를 수도 있습니다. 일부 연산에서는 명시적 캐스팅(형식 승격) 또는 Amazon Redshift 구성 파라미터를 사용하여 이러한 문제를 피할 수 있습니다.

SQL 함수를 사용하는 비슷한 계산 결과에 대한 자세한 내용은 집계 함수 섹션을 참조하세요.

계산 반환 형식

다음 표는 Amazon Redshift에서 지원되는 숫자 데이터 형식을 고려하여 더하기, 빼기, 곱하기 및 나누기 연산에 따라 예상되는 반환 형식을 나타낸 것입니다. 표 왼쪽에서 첫 번째 열은 계산에 포함되는 첫 번째 피연산자이고, 맨 위의 행은 두 번째 피연산자를 의미합니다.

피연산자 1	피연산자 1	반환 타입
INT2	INT2	INT2
INT2	INT4	INT4
INT2	INT8	INT8
INT2	DECIMAL	DECIMAL
INT2	FLOAT4	FLOAT8
INT2	FLOAT8	FLOAT8
INT4	INT4	INT4
INT4	INT8	INT8
INT4	DECIMAL	DECIMAL
INT4	FLOAT4	FLOAT8
INT4	FLOAT8	FLOAT8

피연산자 1	피연산자 1	반환 타입
INT8	INT8	INT8
INT8	DECIMAL	DECIMAL
INT8	FLOAT4	FLOAT8
INT8	FLOAT8	FLOAT8
DECIMAL	DECIMAL	DECIMAL
DECIMAL	FLOAT4	FLOAT8
DECIMAL	FLOAT8	FLOAT8
FLOAT4	FLOAT8	FLOAT8
FLOAT8	FLOAT8	FLOAT8

계산된 DECIMAL 결과의 정밀도와 소수점 자릿수

다음 표는 수학 연산이 DECIMAL 결과를 반환할 때 정밀도와 소수점 자릿수를 계산하는 규칙을 요약한 것입니다. 이 표에서 p1과 s1은 계산 시 첫 번째 피연산자의 정밀도 및 소수점 자릿수를, 그리고 p2와 s2는 두 번째 피연산자의 정밀도 및 소수점 자릿수를 의미합니다. 이 계산과 상관없이 최대 결과 정밀도는 38이고, 최대 결과 소수점 자릿수도 38입니다.

Operation	결과 정밀도 및 소수점 자릿수
+ 또는 -	Scale = `max(s1,s2)` precision = `max(p1-s1,p2-s2)+1+scale`
*	Scale = `s1+s2` precision = `p1+p2+1`
/	Scale = `max(4,s1+p2-s2+1)` precision = `p1-s1+ s2+scale`

예를 들어 SALES 테이블의 PRICEPAID 열과 COMMISSION 열이 모두 DECIMAL(8,2) 열이라고 가정하겠습니다. 이때 PRICEPAID를 COMMISSION으로(혹은 그 반대로) 나누면 다음과 같은 공식이 적용됩니다.

```
Precision = 8-2 + 2 + max(4,2+8-2+1)
= 6 + 2 + 9 = 17

Scale = max(4,2+8-2+1) = 9

Result = DECIMAL(17,9)
```

다음 계산은 UNION, INTERSECT, EXCEPT 같은 집합 연산자나 COALESCE, DECODE 같은 함수를 사용해 DECIMAL 값에 대한 연산 결과 정밀도와 소수점 자릿수를 계산하기 위한 일반 규칙입니다.

```
Scale = max(s1,s2)
Precision = min(max(p1-s1,p2-s2)+scale,19)
```

예를 들어 DECIMAL(7,2) 열 1개가 포함된 DEC1 테이블이 DECIMAL(15,3) 열 1개가 포함된 DEC2 테이블과 조인되어 DEC3 테이블을 생성한다고 가정할 때, DEC3의 스키마를 보면 NUMERIC(15,3) 열이 되는 것을 알 수 있습니다.

```
create table dec3 as select * from dec1 union select * from dec2;
```

결과

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'dec3';

column |     type      | encoding | distkey | sortkey
-------+---------------+----------+---------+---------
c1     | numeric(15,3) | none     | f       | 0
```

위 예에서 적용되는 공식은 다음과 같습니다.

```
Precision = min(max(7-2,15-3) + max(2,3), 19)
= 12 + 3 = 15

Scale = max(2,3) = 3
```

```
Result = DECIMAL(15,3)
```

나누기 연산에 대한 주의 사항

나누기 연산에서 0으로 나누는(divide-by-zero) 조건은 오류를 반환합니다.

정밀도와 소수점 자릿수를 계산한 후에는 소수점 자릿수 한계로 100이 적용됩니다. 계산된 결과 소수점 자릿수가 100보다 크면 나누기 결과가 다음과 같이 조정됩니다.

- precision = `precision - (scale - max_scale)`
- Scale = `max_scale`

계산된 정밀도가 최대 정밀도(38)보다 크면 정밀도가 38로 줄어들고, 소수점 자릿수는 다음 공식의 결과 값이 됩니다. `max((38 + scale - precision), min(4, 100))`

오버플로우 조건

모든 수치 계산에서는 오버플로우 여부를 확인합니다. 정밀도가 19 이하인 DECIMAL 데이터는 64비트 정수로 저장됩니다. 정밀도가 19보다 큰 DECIMAL 데이터는 128비트 정수로 저장됩니다. DECIMAL 값은 모두 최대 정밀도가 38이고, 최대 소수점 자릿수가 37입니다. 오버플로우 오류는 값이 이러한 제한을 초과할 때 발생하며, 중간 결과 집합과 최종 결과 집합 모두에 적용됩니다.

- 특정 데이터 값이 CAST 함수에서 지정하는 정밀도 또는 소수점 자릿수와 맞지 않으면 명시적 변환을 실행해도 오버플로우 오류를 반환합니다. 예를 들어 SALES 테이블의 PRICEPAID 열(DECIMAL(8,2) 열)에 있는 값을 모두 변환할 수는 없기 때문에 다음과 같이 DECIMAL(7,3) 결과를 반환합니다.

  ```
  select pricepaid::decimal(7,3) from sales;
  ERROR:  Numeric data overflow (result precision)
  ```

  이러한 오류가 발생하는 이유는 PRICEPAID 열에서 일부 더 큰 값은 변환할 수 없기 때문입니다.
- 곱하기 연산은 결과 소수점 자릿수가 각 피연산자의 소수점 자릿수 합인 결과를 산출합니다. 예를 들어 두 피연산자의 소수점 자릿수가 4라고 한다면 결과 소수점 자릿수는 8이 되고 소수점 왼쪽에는 10자리만 남게 됩니다. 따라서 둘 다 유효 소수점 자릿수를 가지고 있는 큰 수 2개를 곱할 경우에는 비교적 오버플로우 조건이 발생하기 쉽습니다.

  다음 예는 오버플로 오류를 유발합니다.

  ```
  SELECT CAST(1 AS DECIMAL(38, 20)) * CAST(10 AS DECIMAL(38, 20));
  ```

```
ERROR: 128 bit numeric data overflow (multiplication)
```

곱하기 대신 나누기를 사용하여 오버플로 오류를 해결할 수 있습니다. 다음 예를 사용하여 원래 제수로 나눈 1로 나눕니다.

```
SELECT CAST(1 AS DECIMAL(38, 20)) / (1 / CAST(10 AS DECIMAL(38, 20)));
+----------+
| ?column? |
+----------+
| 10       |
+----------+
```

INTEGER 및 DECIMAL 형식을 사용한 숫자 계산

계산 시 피연산자 하나가 INTEGER 데이터 형식이고, 나머지 피연산자가 DECIMAL 데이터 형식인 경우에는 INTEGER 피연산자가 묵시적으로 DECIMAL로 변환됩니다.

- INT2(SMALLINT)는 DECIMAL(5,0)로 변환됨
- INT4(INTEGER)는 DECIMAL(10,0)로 변환됨
- INT8(BIGINT)은 DECIMAL(19,0)로 변환됨

예를 들어 SALES.COMMISSION(DECIMAL(8,2) 열)과 SALES.QTYSOLD(SMALLINT 열)를 곱하면 다음과 같이 변환됩니다.

```
DECIMAL(8,2) * DECIMAL(5,0)
```

정수 및 부동 소수점 리터럴

숫자를 나타내는 리터럴이나 상수는 정수 또는 부동 소수점이 될 수 있습니다.

정수 리터럴

정수 상수는 0부터 9까지 연속된 숫자로서 옵션으로 양(+) 또는 음(-)의 부호가 숫자 앞에 추가됩니다.

구문

```
[ + | - ] digit ...
```

예시

유효한 정수를 예로 들면 다음과 같습니다.

```
23
-555
+17
```

부동 소수점 리터럴

부동 소수점 리터럴(소수, 숫자 또는 분수 리터럴)은 소수점과 옵션으로 지수 표시(e)까지 포함할 수 있는 연속된 숫자입니다.

구문

```
[ + | - ] digit ... [ . ] [ digit ...]
[ e | E [ + | - ] digit ... ]
```

인수

ee

e 또는 E는 숫자가 유효숫자 표기법으로 지정되는 것을 나타냅니다.

예시

유효한 부동 소수점 리터럴을 예로 들면 다음과 같습니다.

```
3.14159
-37.
2.0e19
-2E-19
```

숫자 형식의 예제

CREATE TABLE 문

다음은 여러 가지 숫자 데이터 형식을 선언하는 CREATE TABLE 문입니다.

```
create table film (
film_id integer,
```

```
language_id smallint,
original_language_id smallint,
rental_duration smallint default 3,
rental_rate numeric(4,2) default 4.99,
length smallint,
replacement_cost real default 25.00);
```

범위를 벗어난 정수를 삽입하려고 시도

다음은 값 33000을 SMALLINT 열에 삽입하는 예입니다.

```
insert into film(language_id) values(33000);
```

SMALLINT의 범위가 -32768~+32767이기 때문에 Amazon Redshift가 오류를 반환합니다.

```
An error occurred when executing the SQL command:
insert into film(language_id) values(33000)

ERROR: smallint out of range [SQL State=22003]
```

소수 값을 정수 열에 삽입

다음은 소수 값을 INT 열에 삽입하는 예입니다.

```
insert into film(language_id) values(1.5);
```

이 값은 정수 값 2로 반올림하여 삽입됩니다.

소수점 자릿수를 반올림하여 소수 값 삽입

다음은 정밀도가 열보다 높은 소수 값을 삽입하는 예입니다.

```
insert into film(rental_rate) values(35.512);
```

이 경우에는 35.51 값이 열에 삽입됩니다.

범위를 벗어나는 소수 값을 삽입하려고 시도

이 경우에는 350.10 값이 범위를 벗어납니다. DECIMAL 열에서 값의 자릿수는 열의 정밀도에서 소수점 자릿수를 뺀 값과 동일합니다(RENTAL_RATE의 경우 4 - 2). 다시 말해서 DECIMAL(4,2) 열의 허용 범위는 -99.99~99.99입니다.

```
insert into film(rental_rate) values (350.10);
ERROR:  numeric field overflow
DETAIL:  The absolute value is greater than or equal to 10^2 for field with precision
 4, scale 2.
```

REAL 열에 가변 정밀도 값 삽입

다음은 가변 정밀도 값을 REAL 열에 삽입하는 예입니다.

```
insert into film(replacement_cost) values(1999999.99);

insert into film(replacement_cost) values(1999.99);

select replacement_cost from film;

+------------------+
| replacement_cost |
+------------------+
| 2000000          |
| 1999.99          |
+------------------+
```

1999999.99 값은 REAL으로 변환되어 2000000 열의 정밀도 요건을 충족합니다. 1999.99 값은 그대로 로드됩니다.

문자 형식

주제

- 스토리지 및 범위
- CHAR 또는 CHARACTER
- VARCHAR 또는 CHARACTER VARYING
- NCHAR 및 NVARCHAR 형식
- TEXT 및 BPCHAR 형식
- 후행 공백의 중요성
- 문자 형식의 예제

문자 데이터 형식에는 CHAR(문자)와 VARCHAR(가변 문자)가 포함됩니다.

스토리지 및 범위

CHAR 및 VARCHAR 데이터 형식은 문자가 아닌 바이트로 정의됩니다. CHAR 열에는 단일 바이트 문자만 포함되기 때문에 예를 들어 CHAR(10) 열이라고 하면 최대 10바이트의 문자열이 포함될 수 있습니다. VARCHAR에는 멀티바이트 문자가 포함되어 문자당 최대 4바이트까지 가능합니다. 예를 들어 VARCHAR(12)라고 하면 단일 바이트 문자 12개, 2바이트 문자 6개, 3바이트 문자 4개, 또는 4바이트 문자 3개가 포함될 수 있습니다.

명칭	스토리지	범위(열의 너비)
CHAR, CHARACTER 또는 NCHAR	후행 공백(있는 경우)을 포함한 문자열 길이	4096 bytes
VARCHAR, CHARACTER VARYING 또는 NVARCHAR	4바이트 + 전체 문자 바이트, 여기에서 각 문자의 길이는 1~4바이트가 될 수 있습니다.	65535바이트(64K -1)
BPCHAR	고정 길이 CHAR(256)로 변환됨	256 bytes
TEXT	VARCHAR(256)로 변환됨	260 bytes

> **Note**
>
> CREATE TABLE 구문은 문자 데이터 형식에 MAX 키워드를 지원합니다. 예:
>
> ```
> create table test(col1 varchar(max));
> ```
>
> MAX 설정은 열의 폭을 CHAR일 때는 4096바이트로, 그리고 VARCHAR일 때는 65535바이트로 정의합니다.

CHAR 또는 CHARACTER

CHAR 또는 CHARACTER 열은 고정 길이 문자열을 저장하는 데 사용됩니다. 이 문자열은 공백으로 채워지므로 CHAR(10) 열은 항상 10바이트의 스토리지를 차지합니다.

```
char(10)
```

길이 명세가 없는 CHAR 열은 CHAR(1) 열이 됩니다.

VARCHAR 또는 CHARACTER VARYING

VARCHAR 또는 CHARACTER VARYING 열은 제한이 고정되어 있는 가변 길이 문자열을 저장하는 데 사용됩니다. 이 문자열은 공백으로 채워지지 않으므로 VARCHAR(120) 열은 단일 바이트 문자 120개, 2바이트 문자 60개, 3바이트 문자 40개 또는 4바이트 문자 30개까지 구성됩니다.

```
varchar(120)
```

CREATE TABLE 문에서 길이 지정자 없이 VARCHAR 데이터 형식을 사용하는 경우 기본 길이는 256입니다. 표현식에 사용되는 경우 출력 크기는 입력 표현식(최대 65535)을 사용하여 결정됩니다.

NCHAR 및 NVARCHAR 형식

NCHAR 및 NVARCHAR 형식(NATIONAL CHARACTER 및 NATIONAL CHARACTER VARYING 형식으로도 불림)의 열을 생성할 수 있습니다. 이 두 형식은 각각 CHAR 및 VARCHAR 형식으로 변환된 후 지정한 바이트 수로 저장됩니다.

길이 명세가 없는 NCHAR 열은 CHAR(1) 열로 변환됩니다.

길이 명세가 없는 NVARCHAR 열은 VARCHAR(256) 열로 변환됩니다.

TEXT 및 BPCHAR 형식

TEXT 열이 포함된 Amazon Redshift 테이블을 생성할 수는 있지만 이 열은 VARCHAR(256) 열로 변환되어 최대 256개 문자까지 가변 길이 값을 허용합니다.

BPCHAR(공백 채움 문자) 형식의 Amazon Redshift 열을 생성할 수 있으며, 이 열은 Amazon Redshift에서 고정 길이 CHAR(256) 열로 변환됩니다.

후행 공백의 중요성

CHAR 및 VARCHAR 데이터 형식은 모두 n 바이트 길이까지 문자열을 저장합니다. 더욱 긴 문자열을 이러한 형식의 열에 저장하려고 하면 오류를 반환합니다. 단, 추가 문자가 모두 공백일 경우에는 문자

열이 최대 길이까지 잘립니다. 문자열이 최대 길이보다 짧을 경우 CHAR 값은 공백으로 채워지지만 VARCHAR 값은 공백 없이 문자열을 저장합니다.

CHAR 값에서 후행 공백은 언제나 의미상 유의적이지 않습니다. CHAR 값 2개를 비교할 때는 무시되고, LENGTH 계산에 포함되지 않으며, 그리고 CHAR 값을 다른 문자열 형식으로 변환할 때는 제거됩니다.

값을 서로 비교할 경우 VARCHAR 값과 CHAR 값의 후행 공백은 의미상 유의적이지 않습니다.

LENGTH 계산을 실행하면 길이에 포함된 후행 공백까지 합쳐서 VARCHAR 문자열의 길이를 반환합니다. 하지만 고정 길이 문자열에서는 후행 공백을 길이에 포함하여 계산하지 않습니다.

문자 형식의 예제

CREATE TABLE 문

다음은 VARCHAR 및 CHAR 데이터 형식을 사용하는 CREATE TABLE 문입니다.

```
create table address(
address_id integer,
address1 varchar(100),
address2 varchar(50),
district varchar(20),
city_name char(20),
state char(2),
postal_code char(5)
);
```

다음 예부터는 위에서 생성한 테이블을 사용합니다.

가변 길이 문자열의 후행 공백

ADDRESS1이 VARCHAR 열이기 때문에 두 번째 삽입된 주소의 후행 공백은 의미상 유의적이지 않습니다. 다시 말해서 삽입된 주소 2개 모두 일치합니다.

```
insert into address(address1) values('9516 Magnolia Boulevard');

insert into address(address1) values('9516 Magnolia Boulevard  ');
```

```
select count(*) from address
where address1='9516 Magnolia Boulevard';

count
```

```
-------
2
(1 row)
```

ADDRESS1 열이 CHAR 열이고 동일한 값이 삽입되었다면 COUNT(*) 쿼리는 문자열을 동일하게 인식하여 2를 반환할 것입니다.

LENGTH 함수의 결과

LENGTH 함수는 VARCHAR 열의 후행 공백을 인식합니다.

```
select length(address1) from address;

length
--------
23
25
(2 rows)
```

CITY_NAME 열(CHAR 열)에서 Augusta 값은 입력 문자열의 후행 공백 유무에 상관없이 항상 7자의 문자 길이를 반환합니다.

열의 길이를 초과하는 값

다음과 같은 경우에는 문자열이 선언한 열의 폭에 맞춰 잘리지 않습니다.

```
insert into address(city_name) values('City of South San Francisco');
ERROR: value too long for type character(20)
```

이러한 문제는 아래와 같이 값을 열의 크기로 변환하여 해결할 수 있습니다.

```
insert into address(city_name)
values('City of South San Francisco'::char(20));
```

이 경우 문자열의 첫 20자(City of South San Fr)가 열에 로드됩니다.

날짜/시간 형식

주제

- 스토리지 및 범위
- 날짜

- TIME
- TIMETZ
- TIMESTAMP
- TIMESTAMPTZ
- 날짜/시간 형식의 예제
- 날짜, 시간 및 타임스탬프 리터럴
- 간격 데이터 유형 및 리터럴

날짜/시간 데이터 형식으로는 DATE, TIME, TIMETZ, TIMESTAMP 및 TIMESTAMPTZ가 있습니다.

스토리지 및 범위

명칭	스토리지	Range	해결 방법
날짜	4 bytes	4713 BC~294276 AD	1일
TIME	8 bytes	00:00:00~24:00:00	1마이크로초
TIMETZ	8 bytes	00:00:00+1459~00:00:00+1459	1마이크로초
TIMESTAMP	8 bytes	4713 BC~294276 AD	1마이크로초
TIMESTAMPTZ	8 bytes	4713 BC~294276 AD	1마이크로초

날짜

DATE 데이터 형식은 타임스탬프 없이 날짜만 저장하는 데 사용됩니다.

TIME

TIME은 TIME WITHOUT TIME ZONE의 별칭입니다.

TIME 데이터 형식을 사용하여 시간을 저장합니다.

TIME 열에는 초의 소수점 이하 자릿수가 최대 6자리인 값이 저장됩니다.

기본적으로 TIME 값은 사용자 테이블과 Amazon Redshift 시스템 테이블 모두에서 협정 세계시(UTC)입니다.

TIMETZ

TIMETZ는 TIME WITH TIME ZONE의 별칭입니다.

TIMETZ 데이터 형식을 사용하여 시간대와 함께 시간을 저장합니다.

TIMETZ 열에는 초의 소수점 이하 자릿수가 최대 6자리인 값이 저장됩니다.

기본적으로 TIMETZ 값은 사용자 테이블과 Amazon Redshift 시스템 테이블 모두에서 UTC입니다.

TIMESTAMP

TIMESTAMP는 TIMESTAMP WITHOUT TIME ZONE의 별칭입니다.

TIMESTAMP 데이터 형식은 날짜와 시간이 모두 포함된 완전한 타임스탬프 값을 저장하는 데 사용됩니다.

TIMESTAMP 열에는 초의 소수점 이하 자릿수가 최대 6자리인 값이 저장됩니다.

TIMESTAMP 열에 날짜를 삽입하거나 일부 타임스탬프 값이 있는 날짜를 삽입하면 값이 암시적으로 전체 타임스탬프 값으로 변환됩니다. 이 전체 타임스탬프 값에는 누락된 시간, 분, 초에 대한 기본값(00)이 있습니다. 입력 문자열의 시간대 값은 무시됩니다.

기본적으로 TIMESTAMP 값은 사용자 테이블과 Amazon Redshift 시스템 테이블 모두에서 UTC입니다.

TIMESTAMPTZ

TIMESTAMPTZ는 TIMESTAMP WITH TIME ZONE의 별칭입니다.

TIMESTAMPTZ 데이터 형식은 날짜, 시간 및 시간대가 모두 포함된 완전한 타임스탬프 값을 입력하는 데 사용됩니다. 입력 값에 시간대가 포함되어 있으면 Amazon Redshift가 해당 시간대를 사용하여 값을 UTC로 변환하여 UTC 값을 저장합니다.

지원되는 시간대 이름 목록을 보려면 다음 명령을 실행합니다.

```
select pg_timezone_names();
```

지원되는 시간대 이름 약어 목록을 보려면 다음 명령을 실행합니다.

```
select pg_timezone_abbrevs();
```

시간대에 대한 현재 정보는 IANA Time Zone Database에서도 확인할 수 있습니다.

다음 표는 각 시간대 형식의 예를 나타낸 것입니다.

형식	예제
dd mon hh:mi:ss yyyy tz	17 Dec 07:37:16 1997 PST
mm/dd/yyyy hh:mi:ss.ss tz	12/17/1997 07:37:16.00 PST
mm/dd/yyyy hh:mi:ss.ss tz	12/17/1997 07:37:16.00 US/Pacific
yyyy-mm-dd hh:mi:ss+/-tz	1997-12-17 07:37:16-08
dd.mm.yyyy hh:mi:ss tz	17.12.1997 07:37:16.00 PST

TIMESTAMPTZ 열에는 초의 소수점 이하 자릿수가 최대 6자리인 값이 저장됩니다.

TIMESTAMPTZ 열에 날짜를 삽입하거나 일부 타임스탬프가 있는 날짜를 삽입하면 값이 암시적으로 전체 타임스탬프 값으로 변환됩니다. 이 전체 타임스탬프 값에는 누락된 시간, 분, 초에 대한 기본값(00)이 있습니다.

TIMESTAMPTZ 값은 사용자 테이블에서 UTC입니다.

날짜/시간 형식의 예제

다음으로 Amazon Redshift에서 지원되는 날짜/시간 형식 작업 예를 찾아봅니다.

날짜 예제

다음은 다른 형식의 날짜를 삽입한 후 출력을 표시하는 예입니다.

```
create table datetable (start_date date, end_date date);
```

```
insert into datetable values ('2008-06-01','2008-12-31');

insert into datetable values ('Jun 1,2008','20081231');
```

```
select * from datetable order by 1;

start_date |  end_date
-----------------------
```

```
2008-06-01 | 2008-12-31
2008-06-01 | 2008-12-31
```

타임스탬프 값을 DATE 열에 삽입하면 시간 구간은 무시하고 날짜만 로드됩니다.

시간 예제

다음은 다른 형식의 TIME 및 TIMETZ 값을 삽입한 후 출력을 표시하는 예입니다.

```
create table timetable (start_time time, end_time timetz);
```

```
insert into timetable values ('19:11:19','20:41:19 UTC');
insert into timetable values ('191119', '204119 UTC');
```

```
select * from timetable order by 1;
start_time |  end_time
-----------------------
 19:11:19  | 20:41:19+00
 19:11:19  | 20:41:19+00
```

타임스탬프 예제

날짜를 TIMESTAMP 또는 TIMESTAMPTZ 열에 삽입할 경우 시간이 자정으로 기본 설정됩니다. 예를 들어 20081231 리터럴을 삽입하면 2008-12-31 00:00:00으로 값이 저장됩니다.

현재 세션의 시간대를 변경하려면 SET 명령을 사용하여 timezone 구성 파라미터를 설정하면 됩니다.

다음은 다른 형식의 타임스탬프를 삽입한 후 결과 테이블을 표시하는 예입니다.

```
create table tstamp(timeofday timestamp, timeofdaytz timestamptz);

insert into tstamp values('Jun 1,2008  09:59:59', 'Jun 1,2008 09:59:59 EST' );
insert into tstamp values('Dec 31,2008 18:20','Dec 31,2008 18:20');
insert into tstamp values('Jun 1,2008  09:59:59 EST', 'Jun 1,2008 09:59:59');

SELECT * FROM tstamp;

+---------------------+------------------------+
|      timeofday      |      timeofdaytz       |
+---------------------+------------------------+
| 2008-06-01 09:59:59 | 2008-06-01 14:59:59+00 |
```

```
| 2008-12-31 18:20:00 | 2008-12-31 18:20:00+00 |
| 2008-06-01 09:59:59 | 2008-06-01 09:59:59+00 |
+---------------------+------------------------+
```

날짜, 시간 및 타임스탬프 리터럴

다음은 Amazon Redshift에서 지원하는 날짜, 시간 및 타임스탬프 리터럴 작업 규칙입니다.

날짜

다음 입력 날짜는 모두 Amazon Redshift 테이블에 로드할 수 있는 DATE 날짜 유형에 대한 리터럴 날짜 값의 유효한 예입니다. 기본 `MDY DateStyle` 모드가 유효한 것으로 간주됩니다. 이 모드는 `1999-01-08`, `01/02/00`과 같은 문자열에서 월 값이 일 값에 선행함을 의미합니다.

> **Note**
>
> 날짜 또는 타임스탬프 리터럴은 테이블에 로드할 때 인용 부호로 묶어야 합니다.

입력 날짜	전체 날짜
1999년 1월 8일	1999년 1월 8일
1999년 1월 8일	1999년 1월 8일
1999년 1월 8일	1999년 1월 8일
1999년 1월 8일	1999년 1월 8일
1999년 1월 8일	2000년 1월 31일
2000년 1월 31일	2000년 1월 31일
2000년 1월 31일	2000년 1월 31일
20080215	2008년 2월 15일
080215	2008년 2월 15일
2008.366	2008년 12월 31일(날짜에서 3자리 구간에 입력할 수 있는 숫자는 001부터 366까지임)

Times

다음 입력 시간은 모두 Amazon Redshift 테이블에 로드할 수 있는 TIME 및 TIMETZ 데이터 유형에 대한 리터럴 시간 값의 유효한 예입니다.

입력 시간	설명(시간 구간)
04:05:06.789	오전 4시 5분 6.789초
04:05:06	오전 4시 5분 6초
04:05	오전 4시 5분 정각
040506	오전 4시 5분 6초
오전 4시 5분	오전 4시 5분 정각(오전은 옵션)
04:05 PM	오후 4시 5분 정각(시간 값은 12보다 작아야 함)
16:05	오후 4시 5분 정각

타임스탬프

다음 입력 타임스탬프는 모두 Amazon Redshift 테이블에 로드할 수 있는 TIMESTAMP 및 TIMESTAMPTZ 데이터 유형에 대한 리터럴 시간 값의 유효한 예입니다. 날짜 리터럴만 유효하다면 모두 아래 시간 리터럴과 함께 사용할 수 있습니다.

입력 타임스탬프(연결된 날짜 및 시간)	설명(시간 구간)
20080215 04:05:06.789	오전 4시 5분 6.789초
20080215 04:05:06	오전 4시 5분 6초
20080215 04:05	오전 4시 5분 정각
20080215 040506	오전 4시 5분 6초
20080215 04:05 AM	오전 4시 5분 정각(오전은 옵션)
20080215 04:05 PM	오후 4시 5분 정각(시간 값은 12보다 작아야 함)

입력 타임스탬프(연결된 날짜 및 시간)	설명(시간 구간)
20080215 16:05	오후 4시 5분 정각
20080215	자정(기본 설정)

특수한 날짜/시간 값

다음 특수 값은 날짜/시간 리터럴로, 혹은 날짜 함수의 인수로 사용됩니다. 이 특수 값을 사용하려면 작은따옴표가 필요하며, 쿼리 처리 시 타임스탬프 정규 값으로 변환됩니다.

특수 값	설명
now	현재 트랜잭션의 시작 시간으로 평가되며, 마이크로초 정밀도로 타임스탬프를 반환합니다.
today	해당하는 날짜로 평가되며, 시간 부분을 0으로 타임스탬프를 반환합니다.
tomorrow	해당하는 날짜로 평가되며, 시간 부분을 0으로 타임스탬프를 반환합니다.
yesterday	해당하는 날짜로 평가되며, 시간 부분을 0으로 타임스탬프를 반환합니다.

다음은 now 및 today가 DATEADD 함수와 함께 작동하는 방식을 보여주는 예입니다.

```
select dateadd(day,1,'today');

date_add
---------------------
2009-11-17 00:00:00
(1 row)

select dateadd(day,1,'now');

date_add
----------------------------
2009-11-17 10:45:32.021394
```

```
(1 row)
```

간격 데이터 유형 및 리터럴

간격 데이터 유형을 사용하여 기간을 seconds, minutes, hours, days, months 및 years 단위로 저장할 수 있습니다. 날짜 및 타임스탬프에 간격 추가, 간격 합산, 날짜 또는 타임스탬프에서 간격 빼기 등 날짜/시간 계산에 간격 데이터 유형 및 리터럴을 사용할 수 있습니다. 간격 리터럴은 테이블의 간격 데이터 유형 열에 대한 입력 값으로 사용할 수 있습니다.

간격 데이터 유형 구문

기간을 년 및 월 단위로 저장하도록 간격 데이터 유형을 지정하려면:

```
INTERVAL year_to_month_qualifier
```

기간을 일, 시간, 분, 초 단위로 저장하도록 간격 데이터 유형을 지정하려면:

```
INTERVAL day_to_second_qualifier [ (fractional_precision) ]
```

간격 리터럴 구문

기간을 년 및 월 단위로 정의하도록 간격 리터럴을 지정하려면:

```
INTERVAL quoted-string year_to_month_qualifier
```

기간을 일, 시간, 분, 초 단위로 정의하도록 간격 리터럴을 지정하려면:

```
INTERVAL quoted-string day_to_second_qualifier [ (fractional_precision) ]
```

인수

quoted-string

수량과 날짜/시간 단위를 입력 문자열로 지정하여 양수 또는 음수 값을 지정합니다. quoted-string에 숫자만 포함된 경우 Amazon Redshift는 year_to_month_qualifier 또는 day_to_second_qualifier에서 단위를 결정합니다. 예를 들어 `'23' MONTH`는 `1 year 11 months`를 나타내고, `'-2' DAY`는 `-2 days 0 hours 0 minutes 0.0 seconds`를 나타내며, `'1-2' MONTH`는 `1 year 2 months`를 나타내고, `'13 day 1 hour 1 minute 1.123 seconds' SECOND`는 `13 days 1 hour 1 minute 1.123 seconds`를 나타냅니다. 간격 출력 형식에 대한 자세한 내용은 간격 스타일 섹션을 참조하세요.

year_to_month_qualifier

간격 범위를 지정합니다. 한정자를 사용하고 시간 단위가 한정자보다 작은 간격을 생성하면 Amazon Redshift는 간격의 더 작은 부분을 잘라서 삭제합니다. year_to_month_qualifier의 유효한 값은 다음과 같습니다.

- YEAR
- MONTH
- YEAR TO MONTH

day_to_second_qualifier

간격 범위를 지정합니다. 한정자를 사용하고 시간 단위가 한정자보다 작은 간격을 생성하면 Amazon Redshift는 간격의 더 작은 부분을 잘라서 삭제합니다. day_to_second_qualifier의 유효한 값은 다음과 같습니다.

- DAY
- HOUR
- MINUTE
- SECOND
- DAY TO HOUR
- DAY TO MINUTE
- DAY TO SECOND
- HOUR TO MINUTE
- HOUR TO SECOND
- MINUTE TO SECOND

INTERVAL 리터럴의 출력은 지정된 가장 작은 INTERVAL 구성 요소까지 잘립니다. 예를 들어 MINUTE 한정자를 사용하는 경우 Amazon Redshift는 MINUTE 미만의 시간 단위를 삭제합니다.

```
select INTERVAL '1 day 1 hour 1 minute 1.123 seconds' MINUTE
```

결과 값은 '1 day 01:01:00'까지 잘립니다.

fractional_precision

간격에 허용되는 소수 자릿수를 지정하는 파라미터 옵션입니다. fractional_precision 인수는 간격에 SECOND가 포함된 경우에만 지정해야 합니다. 예를 들어 SECOND(3)는 세 자리 소수만 허용하는 간격을 생성합니다(예: 1.234초). 최대 소수 자릿수는 6자리입니다.

세션 구성 `interval_forbid_composite_literals`는 YEAR TO MONTH 및 DAY TO SECOND 부분을 모두 사용하여 간격을 지정하는 경우 오류가 반환되는지 여부를 결정합니다. 자세한 내용은 interval_forbid_composite_literals 단원을 참조하십시오.

간격 산술

간격 값을 다른 날짜/시간 값과 함께 사용하여 산술 연산을 수행할 수 있습니다. 다음 테이블에는 사용 가능한 작업과 각 작업에서 얻을 수 있는 데이터 유형이 설명되어 있습니다.

> **Note**
>
> `date` 및 `timestamp` 결과를 모두 생성할 수 있는 작업은 방정식과 관련된 최소 시간 단위를 기반으로 합니다. 예를 들어, `interval`을 `date`에 추가하는 경우 YEAR TO MONTH 간격이면 `date`이 반환되고, DAY TO SECOND 간격이면 timestamp가 반환됩니다.

첫 번째 피연산자가 `interval`인 작업은 지정된 두 번째 피연산자에 대해 다음과 같은 결과를 생성합니다.

연산자	날짜	Timestamp	간격	숫자
-	N/A	N/A	간격	N/A
+	날짜	Date/Timestamp	간격	N/A
*	해당 사항 없음	해당 사항 없음	N/A	간격
/	N/A	해당 사항 없음	N/A	간격

첫 번째 피연산자가 `date`인 작업은 지정된 두 번째 피연산자에 대해 다음과 같은 결과를 생성합니다.

연산자	날짜	Timestamp	간격	숫자
-	숫자	간격	Date/Timestamp	날짜
+	N/A	해당 사항 없음	해당 사항 없음	N/A

첫 번째 피연산자가 timestamp인 작업은 지정된 두 번째 피연산자에 대해 다음과 같은 결과를 생성합니다.

연산자	날짜	Timestamp	간격	숫자
-	숫자	간격	Timestamp	Timestamp
+	N/A	해당 사항 없음	해당 사항 없음	N/A

간격 스타일

SQL the section called “SET” 명령을 사용하여 간격 값의 출력 표시 형식을 변경할 수 있습니다. SQL에서 간격 데이터 유형을 사용하는 경우 텍스트로 변환하여 예상 간격 스타일(예: YEAR TO MONTH::text)을 확인합니다. IntervalStyle 값을 설정하는 데 사용할 수 있는 값은 다음과 같습니다.

- postgres - PostgreSQL 스타일을 따릅니다. 이 값이 기본값입니다.
- postgres_verbose - PostgreSQL 상세 정보 표시 스타일을 따릅니다.
- sql_standard - SQL 표준 간격 리터럴 스타일을 따릅니다.

다음 명령은 간격 스타일을 sql_standard로 설정합니다.

```
SET IntervalStyle to 'sql_standard';
```

postgres 출력 형식

다음은 postgres 간격 스타일의 출력 형식입니다. 각 숫자 값은 음수일 수 있습니다.

```
'<numeric> <unit> [, <numeric> <unit> ...]'
```

```
select INTERVAL '1-2' YEAR TO MONTH::text

varchar
---------------
1 year 2 mons
```

```
select INTERVAL '1 2:3:4.5678' DAY TO SECOND::text
```

```
varchar
------------------
1 day 02:03:04.5678
```

postgres_verbose 출력 형식

postgres_verbose 구문은 postgres와 비슷하지만 postgres_verbose 출력에는 시간 단위도 포함됩니다.

```
'[@] <numeric> <unit> [, <numeric> <unit> ...] [direction]'
```

```
select INTERVAL '1-2' YEAR TO MONTH::text

varchar
-----------------
@ 1 year 2 mons
```

```
select INTERVAL '1 2:3:4.5678' DAY TO SECOND::text

varchar
---------------------------
@ 1 day 2 hours 3 mins 4.56 secs
```

sql_standard 출력 형식

간격 연도-월 값의 형식은 다음과 같습니다. 간격 앞에 음수 부호를 지정하면 간격이 음수임을 나타내며 전체 간격에 적용됩니다.

```
'[-]yy-mm'
```

간격 일-초 값의 형식은 다음과 같습니다.

```
'[-]dd hh:mm:ss.ffffff'
```

```
SELECT INTERVAL '1-2' YEAR TO MONTH::text

varchar
-------
```

```
1-2
```

```
select INTERVAL '1 2:3:4.5678' DAY TO SECOND::text

varchar
---------------
1 2:03:04.5678
```

간격 데이터 유형의 예

다음 예제에서는 테이블을 사용해 INTERVAL 데이터 유형을 사용하는 방법을 보여 줍니다.

```
create table sample_intervals (y2m interval month, h2m interval hour to minute);
insert into sample_intervals values (interval '20' month, interval '2 days
 1:1:1.123456' day to second);
select y2m::text, h2m::text from sample_intervals;


      y2m      |       h2m
---------------+-----------------
 1 year 8 mons | 2 days 01:01:00
```

```
update sample_intervals set y2m = interval '2' year where y2m = interval '1-8' year to
 month;
select * from sample_intervals;

   y2m   |       h2m
---------+-----------------
 2 years | 2 days 01:01:00
```

```
delete from sample_intervals where h2m = interval '2 1:1:0' day to second;
select * from sample_intervals;

 y2m | h2m
-----+-----
```

간격 리터럴의 예

다음 예제는 간격 스타일을 postgres로 설정한 상태에서 실행됩니다.

다음 예제에서는 1년의 INTERVAL 리터럴을 생성하는 방법을 보여 줍니다.

```
select INTERVAL '1' YEAR

intervaly2m
---------------
1 years 0 mons
```

한정자를 초과하는 quoted-string을 지정하면 간격에서 남은 시간 단위가 잘립니다. 다음 예제에서는 13개월의 간격이 1년 1개월이 되지만 나머지 1개월은 YEAR 한정자로 인해 제외됩니다.

```
select INTERVAL '13 months' YEAR

intervaly2m
---------------
1 years 0 mons
```

간격 문자열보다 낮은 한정어를 사용하는 경우 남은 단위가 포함됩니다.

```
select INTERVAL '13 months' MONTH

intervaly2m
---------------
1 years 1 mons
```

간격에 정밀도를 지정하면 소수 자릿수가 잘려 지정된 정밀도까지 줄어듭니다.

```
select INTERVAL '1.234567' SECOND (3)

intervald2s
--------------------------------
0 days 0 hours 0 mins 1.235 secs
```

정밀도를 지정하지 않는 경우 Amazon Redshift는 최대 정밀도 6을 사용합니다.

```
select INTERVAL '1.23456789' SECOND

intervald2s
---------------------------------
0 days 0 hours 0 mins 1.234567 secs
```

다음 예제에서는 범위가 지정된 간격을 생성하는 방법을 보여 줍니다.

```
select INTERVAL '2:2' MINUTE TO SECOND

intervald2s
------------------------------
0 days 0 hours 2 mins 2.0 secs
```

한정자에 따라 지정하는 단위가 결정됩니다. 예를 들어 다음 예제가 이전 예제와 동일한 '2:2' quoted-string을 사용하더라도 Amazon Redshift는 한정자 때문에 다른 시간 단위를 사용한다고 인식합니다.

```
select INTERVAL '2:2' HOUR TO MINUTE

intervald2s
------------------------------
0 days 2 hours 2 mins 0.0 secs
```

각 단위의 약어 및 복수형도 지원됩니다. 예를 들어 5s, 5 second, 5 seconds는 동일한 간격입니다. 지원되는 단위는 연, 월, 시간, 분, 초입니다.

```
select INTERVAL '5s' SECOND

intervald2s
------------------------------
0 days 0 hours 0 mins 5.0 secs
```

```
select INTERVAL '5 HOURS' HOUR

intervald2s
------------------------------
0 days 5 hours 0 mins 0.0 secs
```

```
select INTERVAL '5 h' HOUR

intervald2s
------------------------------
0 days 5 hours 0 mins 0.0 secs
```

한정자 구문이 없는 간격 리터럴의 예

> **Note**
>
> 다음 예에서는 YEAR TO MONTH 또는 DAY TO SECOND 한정자 없이 간격 리터럴을 사용하는 방법을 보여줍니다. 한정자와 함께 권장 간격 리터럴을 사용하는 방법에 대한 자세한 내용은 간격 데이터 유형 및 리터럴 섹션을 참조하세요.

간격 리티럴은 12 hours 또는 6 months 같이 일정한 기간을 구분할 때 사용됩니다. 또한 날짜/시간 표현식이 포함된 조건이나 계산에서도 간격 리터럴을 사용할 수 있습니다.

간격 리터럴은 INTERVAL 키워드와 숫자 기간, 그리고 지원되는 날짜 부분으로 표현됩니다. 예를 들면 INTERVAL '7 days' 또는 INTERVAL '59 minutes'와 같습니다. 더욱 정확한 간격을 표현하기 위해 여러 수량과 단위를 연결할 수도 있습니다(예: INTERVAL '7 days, 3 hours, 59 minutes'). 각 단위의 약어와 복수 표현도 지원됩니다. 예를 들어 5 s, 5 second 및 5 seconds 모두 동일한 간격입니다.

날짜 부분을 지정하지 않을 경우 간격 값은 초를 의미합니다. 기간 값은 소수로 지정할 수도 있습니다(예: 0.5 days).

다음은 모두 간격 값을 다르게 하여 계산하는 예들입니다.

다음은 지정된 날짜에 1초를 더합니다.

```
select caldate + interval '1 second' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2008-12-31 00:00:01
(1 row)
```

다음은 지정된 날짜에 1분을 더합니다.

```
select caldate + interval '1 minute' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2008-12-31 00:01:00
(1 row)
```

다음은 지정한 날짜에 3시간 35분을 더합니다.

```
select caldate + interval '3 hours, 35 minutes' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2008-12-31 03:35:00
(1 row)
```

다음은 지정된 날짜에 52주를 더합니다.

```
select caldate + interval '52 weeks' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2009-12-30 00:00:00
(1 row)
```

다음은 지정한 날짜에 1주, 1시간, 1분, 1초를 더합니다.

```
select caldate + interval '1w, 1h, 1m, 1s' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2009-01-07 01:01:01
(1 row)
```

다음은 지정한 날짜에 12시간(반일)을 더합니다.

```
select caldate + interval '0.5 days' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2008-12-31 12:00:00
(1 row)
```

다음은 2023년 2월 15일에서 4개월을 뺀 값이며 결과는 2022년 10월 15일입니다.

```
select date '2023-02-15' - interval '4 months';

?column?
```

```
---------------------
2022-10-15 00:00:00
```

다음은 2023년 3월 31일에서 4개월을 뺀 값이며 결과는 2022년 11월 30일입니다. 계산에서는 한 달의 일수를 고려합니다.

```
select date '2023-03-31' - interval '4 months';

?column?
---------------------
2022-11-30 00:00:00
```

부울 유형

부울 데이터 형식은 단일 바이트 열에 true 또는 false 값을 저장하는 데 사용됩니다. 다음 표는 부울 값에서 가능한 세 가지 상태와 이러한 상태를 나타내는 리터럴 값에 대해 설명한 것입니다. 입력 문자열에 상관없이 Boolean 열은 true일 때는 "t"를, 그리고 false일 때는 "f"를 저장 및 출력합니다.

State	유효한 리터럴 값	스토리지
True	`TRUE 't' 'true' 'y' 'yes' '1'`	1바이트
False	`FALSE 'f' 'false' 'n' 'no' '0'`	1바이트
알 수 없음	`NULL`	1바이트

IS 비교를 사용해 WHERE 절의 조건자로 부울 값만 확인할 수 있습니다. IS 비교는 SELECT 목록의 부울 값에는 사용할 수 없습니다.

예시

BOOLEAN 열을 사용하여 각 고객의 "Active/Inactive" 상태를 CUSTOMER 테이블에 저장할 수 있습니다.

```
create table customer(
```

```
custid int,
active_flag boolean default true);
```

```
insert into customer values(100, default);
```

```
select * from customer;
custid | active_flag
-------+-------------
   100 | t
```

CREATE TABLE 문에서 기본값(`true` 또는 `false`)을 지정하지 않은 경우에는 기본값을 삽입하더라도 NULL 값을 삽입하는 것과 똑같습니다.

다음은 USERS 테이블에서 스포츠는 좋아하지만 영화를 좋아하지 않는 사용자를 선택하는 쿼리 예입니다.

```
select firstname, lastname, likesports, liketheatre
from users
where likesports is true and liketheatre is false
order by userid limit 10;

firstname |  lastname  | likesports | liketheatre
----------+------------+------------+-------------
Lars      | Ratliff    | t          | f
Mufutau   | Watkins    | t          | f
Scarlett  | Mayer      | t          | f
Shafira   | Glenn      | t          | f
Winifred  | Cherry     | t          | f
Chase     | Lamb       | t          | f
Liberty   | Ellison    | t          | f
Aladdin   | Haney      | t          | f
Tashya    | Michael    | t          | f
Lucian    | Montgomery | t          | f
(10 rows)
```

다음은 USERS 테이블에서 록 음악을 좋아하는지 알 수 없는 사용자를 선택하는 예입니다.

```
select firstname, lastname, likerock
from users
where likerock is unknown
```

```
order by userid limit 10;

firstname | lastname | likerock
----------+----------+----------
Rafael    | Taylor   |
Vladimir  | Humphrey |
Barry     | Roy      |
Tamekah   | Juarez   |
Mufutau   | Watkins  |
Naida     | Calderon |
Anika     | Huff     |
Bruce     | Beck     |
Mallory   | Farrell  |
Scarlett  | Mayer    |
(10 rows)
```

다음 예에서는 SELECT 목록에 IS 비교를 사용했기 때문에 오류를 반환합니다.

```
select firstname, lastname, likerock is true as "check"
from users
order by userid limit 10;

[Amazon](500310) Invalid operation: Not implemented
```

다음 예는 SELECT 목록에서 IS 비교 대신에 같음 비교(=)를 사용했기 때문에 성공합니다.

```
select firstname, lastname, likerock = true as "check"
from users
order by userid limit 10;

firstname | lastname  | check
----------+-----------+------
Rafael    | Taylor    |
Vladimir  | Humphrey  |
Lars      | Ratliff   | true
Barry     | Roy       |
Reagan    | Hodge     | true
Victor    | Hernandez | true
Tamekah   | Juarez    |
Colton    | Roy       | false
Mufutau   | Watkins   |
Naida     | Calderon  |
```

HLLSKETCH 형식

HyperLogLog 스케치에 HLLSKETCH 데이터 형식을 사용합니다. Amazon Redshift는 희소 또는 밀집 HyperLogLog 스케치 표현을 지원합니다. 스케치는 희소로 시작하여 밀집 형식이 사용되는 메모리 공간을 최소화하는 데 더 효율적일 때 밀집으로 전환됩니다.

Amazon Redshift는 다음 JSON 형식으로 스케치를 가져오거나 내보내거나 인쇄할 때 희소 HyperLogLog 스케치를 자동으로 전환합니다.

```
{"logm":15,"sparse":{"indices":[4878,9559,14523],"values":[1,2,1]}}
```

Amazon Redshift는 Base64 형식의 문자열 표현을 사용하여 밀집 HyperLogLog 스케치를 표현합니다.

Amazon Redshift는 Base64 형식의 다음 문자열 표현을 사용하여 밀집 HyperLogLog 스케치를 표현합니다.

```
"ABAABA..."
```

HLLSKETCH 객체의 최대 크기는 원시 압축에 사용 시 24,580바이트입니다.

SUPER 형식

SUPER 데이터 형식을 사용하여 비정형 데이터나 문서를 값으로 저장합니다.

비정형 데이터는 SQL 데이터베이스에 사용되는 관계형 데이터 모델의 엄격한 테이블형 구조를 따르지 않습니다. 여기에는 데이터 내의 고유한 엔티티를 참조하는 태그가 포함됩니다. 여기에는 배열, 중첩 구조 및 JSON과 같은 직렬화 형식과 연결된 기타 복잡한 구조와 같은 복소수 값이 포함될 수 있습니다. SUPER 데이터 형식은 Amazon Redshift의 다른 모든 스칼라 형식을 포함하는 스키마 없는 배열 및 구조 값 집합입니다.

SUPER 데이터 형식은 개별 SUPER 객체에 대해 최대 16MB의 데이터를 지원합니다. 테이블에 이를 구현하는 예를 포함하여 SUPER 데이터 유형에 대한 자세한 내용은 Amazon Redshift의 반정형 데이터을 참조하십시오.

1MB보다 큰 SUPER 객체는 다음 파일 형식에서만 수집할 수 있습니다.

- PARQUET
- JSON
- TEXT
- CSV

SUPEWER 데이터 형식은 다음과 같은 속성을 가집니다.

- Amazon Redshift 스칼라 값:
  - null
  - 부울
  - smallint, integer, bigint, decimal 또는 부동 소수점(예: float4 또는 float8)과 같은 숫자
  - varchar 또는 char와 같은 문자열 값
- 복소수 값:
  - 스칼라 또는 복소수를 포함한 값의 배열
  - 속성 이름 및 값(스칼라 또는 복소수)의 맵인 구조체(튜플 또는 객체라고도 함)

두 가지 형식의 복소수 값은 규칙성에 대한 제한 없이 자체 스칼라 또는 복소수 값을 포함합니다.

SUPER 데이터 형식은 스키마 없는 형태로 비정형 데이터의 지속성을 지원합니다. 계층적 데이터 모델은 변경될 수 있지만 이전 버전의 데이터는 동일한 SUPER 열에 공존할 수 있습니다.

Amazon Redshift는 PartiQL을 사용하여 배열 및 구조체에 대한 탐색을 활성화합니다. Amazon Redshift는 SUPER 배열을 반복하는 데도 PartiQL 구문을 사용합니다. 자세한 내용은 탐색 및 쿼리 중첩 해제 단원을 참조하세요.

Amazon Redshift는 동적 형식 지정을 사용하여 쿼리에 사용하기 전에 데이터 유형을 선언할 필요 없이 스키마 없는 SUPER 데이터를 처리합니다. 자세한 내용은 동적 형식 지정 단원을 참조하십시오.

SUPER 유형 열의 경로에 있는 `scalar` 값에 동적 데이터 마스킹 정책을 적용할 수 있습니다. 동적 데이터 마스킹에 대한 자세한 내용은 동적 데이터 마스킹 섹션을 참조하세요. SUPER 데이터 유형에 동적 데이터 마스킹을 사용하는 방법에 대한 자세한 내용은 SUPER 데이터 유형 경로와 함께 동적 데이터 마스킹 사용 섹션을 참조하세요.

VARBYTE 형식

VARBYTE, VARBINARY 또는 BINARY VARYING 열을 사용하여 고정 제한이 있는 가변 길이 이진 값을 저장합니다.

```
varbyte [ (n) ]
```

최대 바이트 수(n)의 범위는 1~16,777,216입니다. 기본값은 64,000입니다.

VARBYTE 데이터 유형을 사용할 수 있는 몇 가지 예는 다음과 같습니다.

- VARBYTE 열의 테이블 조인.
- VARBYTE 열을 포함하는 구체화된 보기 생성. VARBYTE 열을 포함하는 구체화된 보기의 증분 새로 고침이 지원됩니다. 그러나 VARBYTE 열의 COUNT, MIN, MAX 및 GROUP BY 이외의 집계 함수는 증분 새로 고침을 지원하지 않습니다.

모든 바이트가 인쇄 가능한 문자인지 확인하기 위해 Amazon Redshift는 16진수 형식을 사용하여 VARBYTE 값을 인쇄합니다. 예를 들어 다음 SQL은 16진수 문자열 6162를 이진 값으로 변환합니다. 반환된 값이 이진 값이더라도 결과는 16진수 6162로 출력됩니다.

```
select from_hex('6162');

 from_hex
----------
 6162
```

Amazon Redshift는 VARBYTE와 다음 데이터 유형 간의 캐스팅을 지원합니다.

- CHAR
- VARCHAR
- SMALLINT
- INTEGER
- BIGINT

CHAR 및 VARCHAR로 캐스팅할 때 UTF-8 형식이 사용됩니다. UTF-8 형식에 대한 자세한 내용은 TO_VARBYTE 섹션을 참조하세요. SMALLINT, INTEGER 및 BIGINT에서 캐스팅할 때 원래 데이터 유형의 바이트 수가 유지됩니다. SMALLINT의 경우 2바이트, INTEGER의 경우 4바이트, BIGINT의 경우 8바이트입니다.

다음 SQL 문은 VARCHAR 문자열을 VARBYTE로 캐스팅합니다. 반환된 값이 이진 값이더라도 결과는 16진수 616263로 출력됩니다.

```
select 'abc'::varbyte;

 varbyte
---------
 616263
```

다음 SQL 문은 열의 CHAR 값을 VARBYTE로 캐스팅합니다. 이 예에서는 CHAR(10) 열(c)이 있는 테이블을 생성하고 길이가 10보다 짧은 문자 값을 삽입합니다. 결과 캐스트는 결과를 정의된 열 크기로 공백 문자(hex'20')로 채웁니다. 반환된 값이 이진 값이더라도 결과는 16진수로 출력됩니다.

```
create table t (c char(10));
insert into t values ('aa'), ('abc');
select c::varbyte from t;
          c
----------------------
 61612020202020202020
 61626320202020202020
```

다음 SQL 문은 SMALLINT 문자열을 VARBYTE로 캐스팅합니다. 반환된 값이 이진 값이더라도 결과는 2바이트 또는 4개의 16진수 문자인 16진수 0005로 인쇄됩니다.

```
select 5::smallint::varbyte;

 varbyte
---------
 0005
```

다음 SQL 문은 INTEGER를 VARBYTE로 캐스팅합니다. 반환된 값이 이진 값이더라도 결과는 4바이트 또는 8개의 16진수 문자인 16진수 00000005로 인쇄됩니다.

```
select 5::int::varbyte;

 varbyte
----------
 00000005
```

다음 SQL 문은 BIGINT를 VARBYTE로 캐스팅합니다. 반환된 값이 이진 값이더라도 결과는 8바이트 또는 16개의 16진수 문자인 16진수 0000000000000005로 인쇄됩니다.

```
select 5::bigint::varbyte;

     varbyte
------------------
 0000000000000005
```

VARBYTE 데이터 유형을 지원하는 Amazon Redshift 특성은 다음과 같습니다.

- VARBYTE 연산자
- CONCAT
- LEN
- LENGTH 함수
- OCTET_LENGTH
- SUBSTRING 함수
- FROM_HEX
- TO_HEX
- FROM_VARBYTE
- TO_VARBYTE
- GETBIT
- VARBYTE 데이터 유형의 열 로드
- VARBYTE 데이터 유형의 열 언로드

Amazon Redshift와 함께 VARBYTE 데이터 유형을 사용할 때의 제한 사항

다음은 Amazon Redshift와 함께 VARBYTE 데이터 유형을 사용할 때의 제한 사항입니다.

- Amazon Redshift Spectrum은 Parquet 및 ORC 파일에 대해서만 VARBYTE 데이터 형식을 지원합니다.
- Amazon Redshift 쿼리 편집기와 Amazon Redshift 쿼리 편집기 v2는 아직 VARBYTE 데이터 유형을 완전히 지원하지 않습니다. 따라서 VARBYTE 표현식으로 작업할 때는 다른 SQL 클라이언트를 사용합니다.

  쿼리 편집기를 사용하기 위한 해결 방법으로 데이터 길이가 64KB 미만이고 콘텐츠가 유효한 UTF-8인 경우 VARBYTE 값을 VARCHAR로 캐스팅할 수 있습니다. 예를 들면 다음과 같습니다.

  ```
  select to_varbyte('6162', 'hex')::varchar;
  ```
- Python 또는 Lambda 사용자 정의 함수(UDF)에는 VARBYTE 데이터 유형을 사용할 수 없습니다.
- VARBYTE 열에서 HLLSKETCH 열을 생성하거나 VARBYTE 열에서 APPROXIMATE COUNT DISTINCT를 사용할 수 없습니다.
- 1MB보다 큰 VARBYTE 값은 다음 파일 형식에서만 수집할 수 있습니다.
  - PARQUET

- 텍스트
- CSV(쉼표로 분리된 값)

형식 호환성 및 변환

이번 섹션에서는 Amazon Redshift의 형식 변환 규칙과 데이터 형식 호환성의 적용 방식에 대한 설명을 살펴볼 수 있습니다.

호환성

데이터 형식 일치, 즉 리터럴 값 및 상수를 데이터 형식과 일치시키는 것은 아래 작업을 포함해 다양한 데이터베이스 작업에서 발생합니다.

- 테이블에 대한 데이터 조작 언어(DML) 작업
- UNION, INTERSECT 및 EXCEPT 쿼리
- CASE 표현식
- LIKE, IN 등 조건자 평가
- 데이터를 비교하거나 추출하는 SQL 함수에 대한 평가
- 수학 연산자를 사용한 비교

위의 작업 결과는 형식 변환 규칙과 데이터 형식 호환성에 따라 달라집니다. 호환성에는 특정 값과 특정 데이터 형식의 1대 1 일치가 항상 필요한 것은 아니라는 의미가 내포되어 있습니다. 일부 데이터 형식은 호환이 가능하기 때문에 묵시적 변환, 즉 강제 변환이 가능합니다(자세한 내용은 묵시적인 변환 형식 단원 참조). 데이터 형식이 호환되지 않을 때는 명시적인 변환 함수를 사용하여 다른 데이터 형식으로 값을 변환할 수 있는 경우도 있습니다.

일번적인 호환성 및 변환 규칙

호환성 및 변환 규칙은 다음과 같습니다.

- 일반적으로 동일한 형식 카테고리에 해당하는 데이터 형식(여러 가지 숫자 데이터 형식 등)은 서로 호환이 가능하기 때문에 묵시적으로 변환할 수 있습니다.

  예를 들어 묵시적인 변환을 통해 소수 값을 정수 열에 삽입할 수 있습니다. 이때 소수는 정수로 반올림됩니다. 또는 날짜에서 2008 같은 숫자 값을 추출하여 정수 열에 삽입하는 것도 가능합니다.
- 숫자 데이터 형식은 범위 외 값을 삽입하려고 할 때 오버플로우 조건이 발생할 가능성이 높습니다. 예를 들어 정밀도가 5인 소수 값은 정밀도가 4로 정의된 DECIMAL 열에 맞지 않습니다. 소수에서 정

수부는 절대로 잘리지 않지만 소수부는 상황에 따라 반올림 또는 내림 처리할 수 있습니다. 하지만 테이블에서 선택한 값의 명시적인 변환 결과는 반올림되지 않습니다.

- 다른 형식의 문자열도 호환이 가능합니다. 예를 들어 단일 바이트 데이터가 포함된 VARCHAR 열과 CHAR 열 문자열은 서로 호환이 되어 묵시적으로 변환할 수 있습니다. 멀티바이트 데이터가 포함되는 VARCHAR 문자열은 호환되지 않습니다. 그 밖에 문자열이 적합한 리터럴 값인 경우에는 문자열을 날짜, 시간, 타임스탬프 또는 숫자 값으로 변환할 수도 있습니다. 이때 선행 또는 후행 공백은 무시됩니다. 반대로 날짜, 시간, 타임스탬프 및 숫자 값을 고정 길이 또는 가변 길이 문자열로 변환하는 것도 가능합니다.

  > **Note**
  >
  > 문자열을 숫자 형식으로 변환하려면 문자열에 숫자를 표현한 문자가 있어야 합니다. 예를 들어 `'1.0'`이나 `'5.9'` 같은 문자열은 소수 값으로 변환할 수 있지만 문자열 `'ABC'`는 어떤 숫자 형식으로도 변환할 수 없습니다.

- DECIMAL 값을 문자열과 비교하면 Amazon Redshift는 문자열을 DECIMAL 값으로 변환하려고 시도합니다. 모든 다른 숫자 값을 문자열과 비교하는 경우 숫자 값이 문자열로 변환됩니다. 반대 변환(예: 문자열을 정수로 변환하거나 DECIMAL 값을 문자열로 변환)을 적용하려면 CAST과 같은 명시적 함수를 사용하세요.
- 64비트 DECIMAL 또는 NUMERIC 값의 정밀도를 높여서 변환하려면 CAST 또는 CONVERT 같은 명시적인 변환 함수를 사용해야 합니다.
- DATE 또는 TIMESTAMP를 TIMESTAMPTZ로 변환하거나 TIME을 TIMETZ로 변환할 때 시간대는 현재 세션 시간대로 설정됩니다. 세션 시간대는 기본적으로 UTC입니다. 세션 시간대 설정에 대한 자세한 내용은 timezone 섹션을 참조하세요.
- 마찬가지로 TIMESTAMPTZ 역시 현재 세션 시간대에 따라 DATE, TIME 또는 TIMESTAMP로 변환됩니다. 세션 시간대는 기본적으로 UTC입니다. 변환 후에는 시간대 정보가 삭제됩니다.
- 시간대를 지정하여 타임스탬프를 표현한 문자열은 현재 세션 시간대(기본적으로 UTC)에 따라 TIMESTAMPTZ로 변환됩니다. 마찬가지로 시간대가 지정된 시간을 표현하는 문자열은 현재 세션 시간대(기본값 UTC)를 사용하여 TIMETZ로 변환됩니다.

묵시적인 변환 형식

묵시적인 변환 유형은 다음과 같이 두 가지입니다.

- 인수의 묵시적 변환(INSERT 또는 UPDATE 명령의 값 설정 등)
- 표현식의 묵시적 변환(WHERE 절의 비교 등)

다음 표는 인수 또는 표현식에서 묵시적으로 변환할 수 있는 데이터 형식을 나열한 것입니다. 그 밖에 명시적인 변환 함수를 통한 변환도 가능합니다.

입력 형식	출력 형식
BIGINT (INT8)	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	INTEGER (INT, INT4)
	REAL (FLOAT4)
	SMALLINT (INT2)
	VARCHAR
CHAR	VARCHAR
날짜	CHAR
	VARCHAR
	TIMESTAMP
	TIMESTAMPTZ
DECIMAL (NUMERIC)	BIGINT (INT8)
	CHAR
	DOUBLE PRECISION (FLOAT8)
	INTEGER (INT, INT4)
	REAL (FLOAT4)
	SMALLINT (INT2)

입력 형식	출력 형식
	VARCHAR
DOUBLE PRECISION (FLOAT8)	BIGINT (INT8)
	CHAR
	DECIMAL (NUMERIC)
	INTEGER (INT, INT4)
	REAL (FLOAT4)
	SMALLINT (INT2)
	VARCHAR
INTEGER (INT, INT4)	BIGINT (INT8)
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	REAL (FLOAT4)
	SMALLINT (INT2)
	VARCHAR
REAL (FLOAT4)	BIGINT (INT8)
	CHAR
	DECIMAL (NUMERIC)
	INTEGER (INT, INT4)

입력 형식	출력 형식
	SMALLINT (INT2)
	VARCHAR
SMALLINT (INT2)	BIGINT (INT8)
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	INTEGER (INT, INT4)
	REAL (FLOAT4)
	VARCHAR
TIMESTAMP	CHAR
	날짜
	VARCHAR
	TIMESTAMPTZ
	TIME
TIMESTAMPTZ	CHAR
	날짜
	VARCHAR
	TIMESTAMP
	TIMETZ

입력 형식	출력 형식
TIME	VARCHAR
	TIMETZ
	INTERVAL DAY TO SECOND
TIMETZ	VARCHAR
	TIME
GEOMETRY	GEOGRAPHY
GEOGRAPHY	GEOMETRY

> **Note**
>
> TIMESTAMPTZ, TIMESTAMP, DATE, TIME, TIMETZ 또는 문자열 사이의 묵시적인 변환은 현재 세션 시간대를 사용합니다. 현재 시간대 설정에 대한 자세한 내용은 timezone 섹션을 참조하세요.
> GEOMETRY 및 GEOGRAPHY 데이터 유형은 서로를 제외하고 다른 데이터 유형으로 암시적으로 변환될 수 없습니다. 자세한 내용은 CAST 함수 단원을 참조하십시오.
> VARBYTE 데이터 유형은 암시적으로 다른 데이터 유형으로 변환될 수 없습니다. 자세한 내용은 CAST 함수 단원을 참조하십시오.

SUPER 데이터 형식에 동적 형식 지정 사용

Amazon Redshift는 동적 형식 지정을 사용하여 쿼리에 사용하기 전에 데이터 형식을 선언할 필요 없이 스키마 없는 SUPER 데이터를 처리합니다. 동적 형식 지정은 Amazon Redshift 형식으로 명시적으로 캐스팅하지 않고도 SUPER 데이터 열로 이동한 결과를 사용합니다. SUPER 데이터 형식에 동적 입력 사용에 대한 자세한 내용은 동적 형식 지정 섹션을 참조하세요.

몇 가지 예외를 제외하고는 SUPER 값을 다른 데이터 형식과 캐스팅할 수 있습니다. 자세한 내용은 제한 사항 단원을 참조하십시오.

콜레이션 시퀀스

Amazon Redshift는 로케일 또는 사용자 정의 데이터 정렬 순서를 지원하지 않습니다. 일반적으로 어떤 상황에서든지 데이터 값을 정렬하여 비교하는 로케일 규칙이 없으면 조건자의 결과에 영향을 미칠 수 있습니다. 예를 들어 MIN, MAX, RANK 같은 ORDER BY 표현식과 함수는 데이터의 이진 UTF8 순서에 따라 로케일 문자를 고려하지 않고 결과를 반환합니다.

Expressions

주제

표현식이란 하나 이상의 값, 연산자 또는 값으로 평가되는 함수의 조합을 말합니다. 표현식의 데이터 형식은 일반적으로 구성하는 각 요소의 데이터 형식과 일치합니다.

단순 표현식

단순 표현식은 다음과 같습니다.

- 상수 또는 리터럴 값
- 열 이름 또는 열 참조
- 스칼라 함수
- 집계(집합) 함수
- 창 함수
- 스칼라 하위 쿼리

단순 표현식의 예는 다음과 같습니다.

```
5+12
dateid
sales.qtysold * 100
sqrt (4)
```

```
max (qtysold)
(select max (qtysold) from sales)
```

복합 표현식

복합 표현식이란 연속된 단순 표현식이 산술 연산자로 결합되어 있는 것을 말합니다. 복합 표현식에 사용되는 단순 표현식은 숫자 값을 반환해야 합니다.

구문

```
expression
operator
expression | (compound_expression)
```

인수

expression

값으로 평가되는 단순 표현식입니다.

연산자

복합 산술 표현식은 다음 연산자의 순서에 따라 작성할 수 있습니다.

- ( ) : 평가 순서를 제어하기 위한 괄호
- + , - : 양 및 음의 부호/연산자
- ^ , |/ , ||/ : 지수, 제곱근, 세제곱근
- * , / , % : 곱하기, 나누기 및 모듈로 연산자
- @ : 절대값
- + , - : 더하기, 빼기
- & , |, #, ~, <<, >> : AND, OR, NOT, 왼쪽 이동, 오른쪽 이동 비트 연산자
- ||: 연결

(compound_expression)

복합 표현식은 괄호를 사용해 중첩될 수 있습니다.

예시

복합 표현식의 예는 다음과 같습니다.

```
('SMITH' || 'JONES')
sum(x) / y
sqrt(256) * avg(column)
rank() over (order by qtysold) / 100
(select (pricepaid - commission) from sales where dateid = 1882) * (qtysold)
```

일부 함수는 다른 함수 내에서 중첩될 수도 있습니다. 예를 들어 스칼라 함수는 다른 스칼라 함수 내에서 중첩이 가능합니다. 다음은 숫자 집합의 절대값 총합을 반환하는 예입니다.

```
sum(abs(qtysold))
```

창 함수는 집계 함수 또는 기타 창 함수의 인수로 사용할 수 없습니다. 다음과 같은 표현식은 오류를 반환합니다.

```
avg(rank() over (order by qtysold))
```

창 함수에 중첩된 집계 함수가 포함될 수 있습니다. 다음은 값 집합의 총합을 구한 후 순위를 정하는 표현식입니다.

```
rank() over (order by sum(qtysold))
```

표현식 목록

표현식 목록이란 표현식의 조합을 말하며, 멤버십 및 비교 조건(WHERE 절)과 GROUP BY 절에서 지정할 수 있습니다.

구문

```
expression , expression , ... | (expression, expression, ...)
```

인수

expression

값으로 평가되는 단순 표현식입니다. 표현식 목록에는 쉼표로 구분된 표현식 또는 쉼표로 구분된 표현식 집합이 1개 이상 포함됩니다. 표현식 집합이 다수일 때는 각 집합마다 표현식의 수가 동일하고, 괄호를 사용하여 집합을 구분해야 합니다. 각 집합의 표현식 수는 조건에서 연산자 앞에 있는 표현식 수와 일치해야 합니다.

예시

다음은 조건에서 지정하는 표현식 목록의 예입니다.

```
(1, 5, 10)
('THESE', 'ARE', 'STRINGS')
(('one', 'two', 'three'), ('blue', 'yellow', 'green'))
```

각 집합의 표현식 수는 문의 첫 번째 부분에 있는 수와 일치해야 합니다.

```
select * from venue
where (venuecity, venuestate) in (('Miami', 'FL'), ('Tampa', 'FL'))
order by venueid;

venueid |        venuename        | venuecity | venuestate | venueseats
---------+-------------------------+-----------+------------+------------
28 | American Airlines Arena | Miami     | FL         |          0
54 | St. Pete Times Forum    | Tampa     | FL         |          0
91 | Raymond James Stadium   | Tampa     | FL         |      65647
(3 rows)
```

스칼라 하위 쿼리

스칼라 하위 쿼리는 정확히 1개의 값, 즉 열 1개가 포함된 행 1개를 반환하는 정규 SELECT 쿼리이며, 괄호로 묶입니다. 쿼리를 실행하여 반환되는 값은 바깥쪽 쿼리에 사용됩니다. 하위 쿼리가 행을 0개 반환하는 경우 하위 쿼리 표현식의 값은 NULL입니다. 행을 2개 이상 반환하면 Amazon Redshift가 오류를 반환합니다. 하위 쿼리는 상위 쿼리의 변수를 참조할 수 있으며, 하위 쿼리를 한 번 호출할 때마다 상수 역할을 합니다.

스칼라 하위 쿼리는 표현식이 필요한 대부분 문에서 사용됩니다. 하지만 다음과 같은 경우에는 유효한 표현식이 아닙니다.

- 표현식의 기본값으로 사용되는 경우
- GROUP BY 및 HAVING 절에서 사용되는 경우

예제

다음은 2008년 한 해 판매 한 건당 지불된 평균 가격을 계산하는 하위 쿼리입니다. 그러면 바깥쪽 쿼리가 출력 값을 사용하여 분기별로 판매 한 건당 평균 가격과 비교합니다.

```
select qtr, avg(pricepaid) as avg_saleprice_per_qtr,
(select avg(pricepaid)
from sales join date on sales.dateid=date.dateid
where year = 2008) as avg_saleprice_yearly
from sales join date on sales.dateid=date.dateid
where year = 2008
group by qtr
order by qtr;
qtr  | avg_saleprice_per_qtr | avg_saleprice_yearly
-------+----------------------+----------------------
1     |               647.64 |               642.28
2     |               646.86 |               642.28
3     |               636.79 |               642.28
4     |               638.26 |               642.28
(4 rows)
```

함수 표현식

구문

내장 함수는 모두 표현식으로 사용할 수 있습니다. 함수 호출 구문이 함수의 이름이며, 그 뒤에 인수 목록이 괄호로 묶여서 입력됩니다.

```
function ( [expression [, expression...]] )
```

인수

함수

내장 함수입니다. 예시 함수에 대한 자세한 내용은 SQL 함수 참조 섹션을 참조하세요.

expression

함수에서 예상되는 데이터 형식 및 파라미터 수와 일치하는 표현식입니다.

예시

```
abs (variable)
select avg (qtysold + 3) from sales;
select dateadd (day,30,caldate) as plus30days from date;
```

조건

주제

조건이란 1개 이상의 표현식과 true, false 또는 unknown으로 평가되는 논리 연산자로 구성된 하나의 문을 말합니다. 조건은 종종 조건자로 불리기도 합니다.

> **Note**
>
> 모든 문자열 비교와 LIKE 패턴 일치는 대/소문자를 구분합니다. 예를 들어 'A'와 'a'는 일치하지 않습니다. 하지만 ILIKE 조건자를 사용하면 패턴 일치에서 대/소문자를 구분하지 않을 수도 있습니다.

구문

```
comparison_condition
| logical_condition
| range_condition
| pattern_matching_condition
| null_condition
| EXISTS_condition
| IN_condition
```

비교 조건

비교 조건이란 두 값의 논리적 관계를 말합니다. 모든 비교 조건은 반환 형식이 부울인 이진 연산자입니다. Amazon Redshift는 다음 표에 설명된 비교 연산자를 지원합니다.

연산자	구문	설명
<	`a < b`	값 a가 값 b보다 작습니다.
>	`a > b`	값 a가 값 b보다 큽니다.
<=	`a <= b`	값 a가 값 b보다 작거나 같습니다.
>=	`a >= b`	값 a가 값 b보다 크거나 같습니다.
=	`a = b`	값 a가 값 b와 같습니다.
<> 또는 !=	`a <> b or a != b`	값 a가 값 b와 같지 않습니다.
ANY | SOME	`a = ANY(subquery)`	값 a가 하위 쿼리에서 반환되는 모든 값과 같습니다.
ALL	`a <> ALL or != ALL (subquery))`	값 a가 하위 쿼리에서 반환되는 모든 값과 같지 않습니다.
IS TRUE | FALSE | UNKNOWN	`a IS TRUE`	값이 부울 TRUE입니다.

사용 노트

= ANY | SOME

ANY 및 SOME 키워드는 IN 조건과 동의어입니다. 즉, 반환 값이 1개 이상인 하위 쿼리에서 반환되는 값 중에서 1개라도 비교 결과가 true라면 true를 반환합니다. Amazon Redshift는 ANY와 SOME일 때 =(같음) 조건만 지원합니다. 부등식 조건은 지원하지 않습니다.

> **Note**
>
> ALL 조건자는 지원되지 않습니다.

<> ALL

ALL 키워드는 NOT IN(IN 조건 조건 참조)과 동의어이기 때문에 하위 쿼리 결과에 표현식이 포함되어 있지 않을 때 true를 반환합니다. Amazon Redshift는 ALL일 때 <> 또는 !=(같지 않음) 조건만 지원합니다. 기타 비교 조건은 지원하지 않습니다.

IS TRUE/FALSE/UNKNOWN

0이 아닌 값은 TRUE와, 0은 FALSE와, 그리고 NULL은 UNKNOWN과 같습니다. 부울 유형 데이터 형식을 참조하십시오.

예시

다음은 몇 가지 간단한 비교 조건 예입니다.

```
a = 5
a < b
min(x) >= 5
qtysold = any (select qtysold from sales where dateid = 1882
```

다음은 VENUE 테이블에서 좌석 수가 10000석 이상인 장소를 반환하는 쿼리입니다.

```
select venueid, venuename, venueseats from venue
where venueseats > 10000
order by venueseats desc;

venueid |           venuename            | venueseats
---------+--------------------------------+------------
83 | FedExField                     |      91704
 6 | New York Giants Stadium        |      80242
79 | Arrowhead Stadium              |      79451
78 | INVESCO Field                  |      76125
69 | Dolphin Stadium                |      74916
67 | Ralph Wilson Stadium           |      73967
76 | Jacksonville Municipal Stadium |      73800
89 | Bank of America Stadium        |      73298
72 | Cleveland Browns Stadium       |      73200
86 | Lambeau Field                  |      72922
...
(57 rows)
```

다음은 USERS 테이블에서 록 음악을 좋아하는 사용자(USERID)를 선택하는 예입니다.

```
select userid from users where likerock = 't' order by 1 limit 5;

userid
--------
3
5
6
13
16
(5 rows)
```

다음은 USER 테이블에서 록 음악을 좋아하는지 알 수 없는 사용자(USERID)를 선택하는 예입니다.

```
select firstname, lastname, likerock
from users
where likerock is unknown
order by userid limit 10;

firstname | lastname | likerock
----------+----------+----------
Rafael    | Taylor   |
Vladimir  | Humphrey |
Barry     | Roy      |
Tamekah   | Juarez   |
Mufutau   | Watkins  |
Naida     | Calderon |
Anika     | Huff     |
Bruce     | Beck     |
Mallory   | Farrell  |
Scarlett  | Mayer    |
(10 rows
```

TIME 열이 있는 예

다음 예제 테이블 TIME_TEST에는 3개의 값이 삽입된 TIME_VAL(TIME 형식) 열이 있습니다.

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
```

```
00:58:00
```

다음 예에서는 각 timetz_val에서 시간을 추출합니다.

```
select time_val from time_test where time_val < '3:00';
   time_val
---------------
 00:00:00.5550
 00:58:00
```

다음 예에서는 2개의 시간 리터럴을 비교합니다.

```
select time '18:25:33.123456' = time '18:25:33.123456';
 ?column?
----------
 t
```

TIMETZ 열이 있는 예

다음 예제 테이블 TIMETZ_TEST에는 3개의 값이 삽입된 TIMETZ_VAL(TIMETZ 형식) 열이 있습니다.

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

다음 예에서는 `3:00:00 UTC`보다 작은 TIMETZ 값만 선택합니다. 값을 UTC로 변환한 후 비교합니다.

```
select timetz_val from timetz_test where timetz_val < '3:00:00 UTC';

   timetz_val
---------------
 00:00:00.5550+00
```

다음 예에서는 2개의 TIMETZ 리터럴을 비교합니다. 비교를 위해 시간대는 무시됩니다.

```
select time '18:25:33.123456 PST' < time '19:25:33.123456 EST';

 ?column?
```

```
----------
 t
```

논리 조건

논리 조건은 두 조건 결과를 결합하여 단일 결과를 산출합니다. 모든 논리 조건은 반환 형식이 부울인 이진 연산자입니다.

구문

```
expression
{ AND | OR }
expression
NOT expression
```

논리 조건은 값이 3개인 부울 논리를 사용하며, 여기에서 NULL 값은 알 수 없는 관계를 의미합니다. 다음 표는 논리 조건 결과를 설명한 것으로서 E1과 E2는 표현식을 의미합니다.

E1	E	E1 AND E2	E1 OR E2	NOT E2
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE
TRUE	UNKNOWN	UNKNOWN	TRUE	UNKNOWN
FALSE	TRUE	FALSE	TRUE	
FALSE	FALSE	FALSE	FALSE	
FALSE	UNKNOWN	FALSE	UNKNOWN	
UNKNOWN	TRUE	UNKNOWN	TRUE	
UNKNOWN	FALSE	FALSE	UNKNOWN	
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	

NOT 연산자는 AND 이전에, 그리고 AND 연산자는 OR 이전에 평가됩니다. 하지만 괄호를 사용하면 이러한 기본 평가 순서를 재정의할 수 있습니다.

예시

다음은 USERS 테이블에서 Las Vegas와 스포츠를 모두 좋아하는 사용자의 USERID 및 USERNAME을 반환하는 예입니다.

```
select userid, username from users
where likevegas = 1 and likesports = 1
order by userid;

userid | username
--------+----------
1 | JSG99FHE
67 | TWU10MZT
87 | DUF19VXU
92 | HYP36WEQ
109 | FPL38HZK
120 | DMJ24GUZ
123 | QZR22XGQ
130 | ZQC82ALK
133 | LBN45WCH
144 | UCX04JKN
165 | TEY68OEB
169 | AYQ83HGO
184 | TVX65AZX
...
(2128 rows)
```

다음은 USERS 테이블에서 Las Vegas 또는 스포츠를, 혹은 둘 다 좋아하는 사용자의 USERID 및 USERNAME을 반환하는 예입니다. 이 쿼리는 이전 예의 모든 출력에 더하여 Las Vegas 또는 스포츠만 좋아하는 사용자까지 반환합니다.

```
select userid, username from users
where likevegas = 1 or likesports = 1
order by userid;

userid | username
--------+----------
1 | JSG99FHE
2 | PGL08LJI
3 | IFT66TXU
5 | AEB55QTM
6 | NDQ15VBM
9 | MSD36KVR
```

```
10 | WKW41AIW
13 | QTF33MCG
15 | OWU78MTR
16 | ZMG93CDD
22 | RHT62AGI
27 | KOY02CVE
29 | HUH27PKK
...
(18968 rows)
```

다음은 OR 조건에 괄호를 사용하여 New York 또는 California에서 Macbeth를 공연한 장소를 찾는 예입니다.

```
select distinct venuename, venuecity
from venue join event on venue.venueid=event.venueid
where (venuestate = 'NY' or venuestate = 'CA') and eventname='Macbeth'
order by 2,1;

venuename                |   venuecity
-----------------------------------------+---------------
Geffen Playhouse                         | Los Angeles
Greek Theatre                            | Los Angeles
Royce Hall                               | Los Angeles
American Airlines Theatre                | New York City
August Wilson Theatre                    | New York City
Belasco Theatre                          | New York City
Bernard B. Jacobs Theatre                | New York City
...
```

위 예에서 괄호를 제거하면 쿼리의 논리 및 결과가 바뀝니다.

다음 예에서는 NOT 스크립트를 사용합니다:

```
select * from category
where not catid=1
order by 1;

catid | catgroup |  catname  |                 catdesc
-------+----------+-----------+--------------------------------------------
2 | Sports   | NHL       | National Hockey League
3 | Sports   | NFL       | National Football League
4 | Sports   | NBA       | National Basketball Association
```

```
5 | Sports   | MLS       | Major League Soccer
...
```

다음은 NOT 조건에 이어 AND 조건을 사용하는 예입니다.

```
select * from category
where (not catid=1) and catgroup='Sports'
order by catid;

catid | catgroup | catname |             catdesc
-------+----------+---------+---------------------------------
2 | Sports   | NHL     | National Hockey League
3 | Sports   | NFL     | National Football League
4 | Sports   | NBA     | National Basketball Association
5 | Sports   | MLS     | Major League Soccer
(4 rows)
```

패턴 일치 조건

주제

- LIKE
- SIMILAR TO
- POSIX 연산자

패턴 일치 연산자는 조건 표현식에서 지정한 패턴을 문자열에서 검색하여 일치하는 패턴 유무에 따라 true 또는 false를 반환합니다. Amazon Redshift는 패턴 일치에 세 가지 방법을 사용합니다.

- LIKE 표현식

  LIKE 연산자는 열 이름 같은 문자열 표현식과 %(퍼센트) 및 _(밑줄) 와일드카드 문자의 사용 패턴을 서로 비교합니다. LIKE 패턴 일치는 항상 전체 문자열을 검색합니다. LIKE는 대/소문자를 구분하여 패턴을 일치시키는 반면 ILIKE는 대/소문자를 구분하지 않고 패턴을 일치시킵니다.

- SIMILAR TO 정규 표현식

  SIMILAR TO 연산자는 문자열 표현식과 SQL 표준 정규 표현식 패턴을 일치시킵니다. 여기에는 LIKE 연산자에서 지원하는 문자 2개는 물론이고 패턴 일치 메타 문자까지 포함될 수 있습니다. SIMILAR TO는 전체 문자열을 일치시키며, 대/소문자를 구분합니다.

- POSIX-스타일 정규 표현식

POSIX 정규 표현식은 LIKE 및 SIMILAR TO 연산자보다 더욱 강력한 패턴 일치 수단을 제공합니다. POSIX 정규 표현식 패턴은 대/소문자를 구분하여 문자열의 어느 구간이든 일치시킬 수 있습니다.

SIMILAR TO 또는 POSIX 연산자를 사용하는 정규 표현식 일치는 계산에 따른 리소스 비용이 높습니다. 따라서 특히 다수의 행을 처리할 때는 최대한 LIKE를 사용하는 것이 좋습니다. 예를 들어 다음 두 쿼리는 기능면에서 동일하지만 LIKE를 사용하는 쿼리의 실행 속도가 정규 표현식을 사용하는 쿼리보다 몇 배 더 빠릅니다.

```
select count(*) from event where eventname SIMILAR TO '%(Ring|Die)%';
select count(*) from event where eventname LIKE '%Ring%' OR eventname LIKE '%Die%';
```

LIKE

LIKE 연산자는 열 이름 같은 문자열 표현식과 %(퍼센트) 및 _(밑줄) 와일드카드 문자의 사용 패턴을 서로 비교합니다. LIKE 패턴 일치는 항상 전체 문자열을 검색합니다. 문자열 내 아무 곳에서나 시퀀스를 일치시키려면 패턴이 퍼센트 기호로 시작해서 끝나야 합니다.

LIKE는 대/소문자를 구분하지만 ILIKE는 대/소문자를 구분하지 않습니다.

구문

```
expression [ NOT ] LIKE | ILIKE pattern [ ESCAPE 'escape_char' ]
```

인수

expression

열 이름 같이 유효한 UTF-8 문자 표현식입니다.

LIKE | ILIKE

LIKE는 대/소문자를 구분하여 패턴을 일치시킵니다. ILIKE는 단일 바이트 UTF-8(ASCII) 문자일 때 대/소문자를 구분하지 않고 패턴을 일치시킵니다. 멀티바이트 문자에 대해 대/소문자를 구분하지 않는 패턴 일치를 수행하려면 LIKE 조건이 있는 expression 및 pattern에 LOWER 함수를 사용합니다.

비교 조건자(예: = 및 <>)와 달리 LIKE 및 ILIKE 조건자는 후행 공백을 묵시적으로 무시하지 않습니다. 후행 공백을 무시하려면 RTRIM을 사용하거나 CHAR 열을 VARCHAR로 명시적으로 캐스트합니다.

~~ 연산자는 LIKE와 동일하며 ~~*는 ILIKE와 동일합니다. 또한 !~~ 및 !~~* 연산자는 NOT LIKE 및 NOT ILIKE와 동일합니다.

패턴

일치시킬 패턴이 포함된, 유효한 UTF-8 문자 표현식입니다.

escape_char

패턴의 메타 문자를 이스케이프 처리하는 문자 표현식입니다. 기본값은 백슬래시 2개('\\')입니다.

pattern에 메타 문자가 포함되어 있지 않으면 패턴이 문자열 자체만 의미합니다. 이런 경우에는 LIKE가 등호 연산자와 동일한 역할을 합니다.

문자 표현식 중 하나는 CHAR 또는 VARCHAR 데이터 형식이 될 수 있습니다. 데이터 형식이 서로 다른 경우에는 Amazon Redshift가 pattern을 expression의 데이터 형식으로 변환합니다.

LIKE에서 지원되는 패턴 일치 메타 문자는 다음과 같습니다.

연산자	설명
%	0개 이상의 문자 시퀀스를 일치시킵니다.
_	모든 문자를 일치시킵니다.

예시

다음 표는 LIKE를 사용한 패턴 일치의 예를 나타낸 것입니다.

표현식	반환
`'abc' LIKE 'abc'`	True
`'abc' LIKE 'a%'`	True
`'abc' LIKE '_B_'`	False
`'abc' ILIKE '_B_'`	True
`'abc' LIKE 'c%'`	False

다음은 이름이 "E"로 시작하는 도시를 모두 찾는 예입니다.

```
select distinct city from users
where city like 'E%' order by city;
city
---------------
East Hartford
East Lansing
East Rutherford
East St. Louis
Easthampton
Easton
Eatontown
Eau Claire
...
```

다음은 성에 "ten"이 포함된 사용자를 찾는 예입니다.

```
select distinct lastname from users
where lastname like '%ten%' order by lastname;
lastname
-------------
Christensen
Wooten
...
```

다음 예제에서는 여러 패턴을 매칭하는 방법을 보여줍니다.

```
select distinct lastname from tickit.users
where lastname like 'Chris%' or lastname like '%Wooten' order by lastname;
lastname
-------------
Christensen
Christian
Wooten
...
```

다음은 세 번째와 네 번째 문자가 "ea"인 도시를 찾는 예입니다. 이 명령에서는 ILIKE를 사용하여 대/소문자를 구분하지 않고 있습니다.

```
select distinct city from users where city ilike '__EA%' order by city;
```

```
city
-------------
Brea
Clearwater
Great Falls
Ocean City
Olean
Wheaton
(6 rows)
```

다음은 기본 이스케이프 문자열(\\)을 사용하여 'start_'(텍스트 start 뒤에 밑줄 _가 붙음)가 포함된 문자열을 찾는 예입니다.

```
select tablename, "column" from pg_table_def
where "column" like '%start\\_%'
limit 5;

     tablename     |    column
-------------------+---------------
 stl_s3client      | start_time
 stl_tr_conflict   | xact_start_ts
 stl_undone        | undo_start_ts
 stl_unload_log    | start_time
 stl_vacuum_detail | start_row
(5 rows)
```

다음은 '^'을 이스케이프 문자로 지정한 후 이 이스케이프 문자를 사용하여 'start_'(텍스트 start 뒤에 밑줄 _가 붙음)가 포함된 문자열을 찾는 예입니다.

```
select tablename, "column" from pg_table_def
where "column" like '%start^_%' escape '^'
limit 5;

     tablename     |    column
-------------------+---------------
 stl_s3client      | start_time
 stl_tr_conflict   | xact_start_ts
 stl_undone        | undo_start_ts
 stl_unload_log    | start_time
 stl_vacuum_detail | start_row
(5 rows)
```

다음 예시에서는 ~~* 연산자를 사용하여 'Ag'로 시작하는 도시에 대해 대소문자를 구분하지 않는 (ILIKE) 검색을 수행합니다.

```
select distinct city from users where city ~~* 'Ag%' order by city;

city
------------
Agat
Agawam
Agoura Hills
Aguadilla
```

SIMILAR TO

SIMILAR TO 연산자는 열 이름 같은 문자열 표현식과 SQL 표준 정규 표현식 패턴을 일치시킵니다. SQL 정규 표현식 패턴에는 LIKE 연산자에서 지원되는 문자 2개는 물론이고 패턴 일치 문자까지 포함될 수 있습니다.

SIMILAR TO 연산자는 패턴이 문자열 구간과 일치하는 POSIX 정규 표현식과 달리 패턴이 전체 문자열과 일치하는 경우에만 true를 반환합니다.

SIMILAR TO는 대/소문자를 구분하여 패턴을 일치시킵니다.

> **Note**
>
> SIMILAR TO를 사용하는 정규 표현식 일치는 계산에 따른 리소스 비용이 높습니다. 따라서 특히 다수의 행을 처리할 때는 최대한 LIKE를 사용하는 것이 좋습니다. 예를 들어 다음 두 쿼리는 기능면에서 동일하지만 LIKE를 사용하는 쿼리의 실행 속도가 정규 표현식을 사용하는 쿼리보다 몇 배 더 빠릅니다.
>
> ```
> select count(*) from event where eventname SIMILAR TO '%(Ring|Die)%';
> select count(*) from event where eventname LIKE '%Ring%' OR eventname LIKE '%Die
> %';
> ```

구문

```
expression [ NOT ] SIMILAR TO pattern [ ESCAPE 'escape_char' ]
```

인수

expression

열 이름 같이 유효한 UTF-8 문자 표현식입니다.

SIMILAR TO

SIMILAR TO는 대/소문자를 구분하여 expression의 전체 문자열과 패턴을 일치시킵니다.

패턴

SQL 표준 정규 표현식 패턴을 나타내는, 유효한 UTF-8 문자 표현식입니다.

escape_char

패턴의 메타 문자를 이스케이프 처리하는 문자 표현식입니다. 기본값은 백슬래시 2개('\\')입니다.

pattern에 메타 문자가 포함되어 있지 않으면 패턴이 문자열 자체만 의미합니다.

문자 표현식 중 하나는 CHAR 또는 VARCHAR 데이터 형식이 될 수 있습니다. 데이터 형식이 서로 다른 경우에는 Amazon Redshift가 pattern을 expression의 데이터 형식으로 변환합니다.

SIMILAR TO에서 지원되는 패턴 일치 메타 문자는 다음과 같습니다.

연산자	설명
%	0개 이상의 문자 시퀀스를 일치시킵니다.
_	모든 문자를 일치시킵니다.
|	대체(둘 중 하나)를 나타냅니다.
*	이전 항목을 0번 이상 반복합니다.
+	이전 항목을 1번 이상 반복합니다.
?	이전 항목을 0번 또는 1번 반복합니다.
{m}	이전 항목을 정확히 m번 반복합니다.
{m,}	이전 항목을 m번 이상 반복합니다.
{m,n}	이전 항목을 m번 이상, n번 미만 반복합니다.

연산자	설명
()	그룹 항목을 괄호로 묶어 단일 논리 항목으로 처리합니다.
[...]	대괄호 표현식은 POSIX 정규 표현식처럼 문자 클래스를 지정합니다.

예시

다음 표는 SIMILAR TO를 사용한 패턴 일치의 예를 나타낸 것입니다.

표현식	반환
'abc' SIMILAR TO 'abc'	True
'abc' SIMILAR TO '_b_'	True
'abc' SIMILAR TO '_A_'	False
'abc' SIMILAR TO '%(b|d)%'	True
'abc' SIMILAR TO '(b|c)%'	False
'AbcAbcdefgefg12efgefg12' SIMILAR TO '((Ab)?c)+d((efg)+(12))+'	True
'aaaaaab11111xy' SIMILAR TO 'a{6}_[0-9]{5}(x|y){2}'	True
'$0.87' SIMILAR TO '$[0-9]+(.[0-9][0-9])?'	True

다음은 이름에 "E" 또는 "H"가 포함되는 도시를 찾는 예입니다.

```
SELECT DISTINCT city FROM users
WHERE city SIMILAR TO '%E%|%H%' ORDER BY city LIMIT 5;

      city
-----------------
 Agoura Hills
```

```
 Auburn Hills
 Benton Harbor
 Beverly Hills
 Chicago Heights
```

다음은 기본 이스케이프 문자열('\\')을 사용하여 "_"이 포함된 문자열을 찾는 예입니다.

```
SELECT tablename, "column" FROM pg_table_def
WHERE "column" SIMILAR TO '%start\\_%'
ORDER BY tablename, "column" LIMIT 5;

        tablename         |       column
--------------------------+---------------------
 stcs_abort_idle          | idle_start_time
 stcs_abort_idle          | txn_start_time
 stcs_analyze_compression | start_time
 stcs_auto_worker_levels  | start_level
 stcs_auto_worker_levels  | start_wlm_occupancy
```

다음은 '^'을 이스케이프 문자열로 지정한 후 이 이스케이프 문자열을 사용하여 "_"이 포함된 문자열을 찾는 예입니다.

```
SELECT tablename, "column" FROM pg_table_def
WHERE "column" SIMILAR TO '%start^_%' ESCAPE '^'
ORDER BY tablename, "column" LIMIT 5;

        tablename         |       column
--------------------------+---------------------
 stcs_abort_idle          | idle_start_time
 stcs_abort_idle          | txn_start_time
 stcs_analyze_compression | start_time
 stcs_auto_worker_levels  | start_level
 stcs_auto_worker_levels  | start_wlm_occupancy
```

POSIX 연산자

POSIX 정규 표현식은 일치 패턴을 지정하는 일련의 문자입니다. 문자열이 정규 표현식에 설명된 정규 집합의 멤버인 경우 해당 문자열은 정규 표현식과 일치합니다.

POSIX 정규 표현식은 LIKE 및 SIMILAR TO 연산자보다 더욱 강력한 패턴 일치 수단을 제공합니다. POSIX 정규 표현식은 패턴은 전체 문자열과 일치하는 경우에만 true를 반환하는 SIMILAR TO 연산자와 달리 문자열 구간과 일치시킬 수 있습니다.

> **Note**
>
> POSIX 연산자를 사용하는 정규 표현식 일치는 계산에 따른 리소스 비용이 높습니다. 따라서 특히 다수의 행을 처리할 때는 최대한 LIKE를 사용하는 것이 좋습니다. 예를 들어 다음 두 쿼리는 기능면에서 동일하지만 LIKE를 사용하는 쿼리의 실행 속도가 정규 표현식을 사용하는 쿼리보다 몇 배 더 빠릅니다.
>
> ```
> select count(*) from event where eventname ~ '.*(Ring|Die).*';
> select count(*) from event where eventname LIKE '%Ring%' OR eventname LIKE '%Die
> %';
> ```

구문

```
expression [ ! ] ~ pattern
```

인수

expression

열 이름 같이 유효한 UTF-8 문자 표현식입니다.

!

부정 연산자입니다. 정규 표현식과 일치하지 않습니다.

~

대/소문자를 구분하여 expression의 하위 문자열과 일치시킵니다.

> **Note**
>
> ~~는 LIKE의 동의어입니다.

패턴

정규 표현식 패턴을 나타내는 문자열 리터럴입니다.

pattern에 와일드카드 문자가 포함되어 있지 않으면 패턴이 문자열 자체만 의미합니다.

‘. * | ? ‘ 등 메타 문자가 포함된 문자열을 찾으려면 백슬래시 2개(' \\')를 사용하여 문자를 이스케이프 처리하십시오. SIMILAR TO나 LIKE와 달리 POSIX 정규 표현식 구문은 사용자 정의 이스케이프 문자를 지원하지 않습니다.

문자 표현식 중 하나는 CHAR 또는 VARCHAR 데이터 형식이 될 수 있습니다. 데이터 형식이 서로 다른 경우에는 Amazon Redshift가 pattern을 expression의 데이터 형식으로 변환합니다.

문자 표현식 모두 CHAR 또는 VARCHAR 데이터 형식이 될 수 있습니다. 표현식의 데이터 형식이 다르면 Amazon Redshift가 expression의 데이터 형식으로 변환합니다.

POSIX 패턴 일치에서 지원되는 메타 문자는 다음과 같습니다.

POSIX	설명
.	모든 문자를 일치시킵니다.
*	0개 이상의 발생 패턴을 일치시킵니다.
+	1개 이상의 발생 패턴을 일치시킵니다.
?	0개 또는 1개의 발생 패턴을 일치시킵니다.
|	대체할 일치 패턴을 지정합니다. 예를 들어 E | H 는 E 또는 H를 의미합니다.
^	라인 시작 문자를 일치시킵니다.
$	라인 끝 문자를 일치시킵니다.
$	문자열 끝을 일치시킵니다.
[]	대괄호는 일치하는, 즉 목록의 표현식 1개와 일치해야 하는 목록을 지정합니다. 일치하지 않는, 즉 목록의 표현식을 제외한 모든 문자와 일치하는 목록 앞에는(^)이 입력됩니다.
( )	그룹 항목을 괄호로 묶어 단일 논리 항목으로 처리합니다.
{m}	이전 항목을 정확히 m번 반복합니다.
{m,}	이전 항목을 m번 이상 반복합니다.

POSIX	설명
`{m,n}`	이전 항목을 m번 이상, n번 미만 반복합니다.
`[: :]`	POSIX 문자 클래스 안에 있는 모든 문자를 일치시킵니다. `[:alnum:]` , `[:alpha:]` , `[:lower:]` , `[:upper:]` 문자 클래스에서는 Amazon Redshift가 ASCII 문자만 지원합니다.

다음은 Amazon Redshift에서 지원되는 POSIX 문자 클래스입니다.

문자 클래스	설명
`[[:alnum:]]`	모든 ASCII 영숫자 문자
`[[:alpha:]]`	모든 ASCII 알파벳 문자
`[[:blank:]]`	모든 공백 문자
`[[:cntrl:]]`	모든 제어 문자(비인쇄)
`[[:digit:]]`	모든 숫자
`[[:lower:]]`	모든 ASCII 알파벳 소문자
`[[:punct:]]`	모든 구두점 문자
`[[:space:]]`	모든 공백 문자(비인쇄)
`[[:upper:]]`	모든 ASCII 알파벳 대문자
`[[:xdigit:]]`	모든 유효 16진수 문자

Amazon Redshift는 정규 표현식에서 다음과 같이 Perl의 영향을 받는 연산자를 지원합니다. 이러한 연산자는 백슬래시 2개('\\')를 사용하여 이스케이프 처리합니다.

연산자	설명	등가의 문자 클래스 표현식
`\\d`	숫자 문자	`[[:digit:]]`

연산자	설명	등가의 문자 클래스 표현식
\\D	숫자를 제외한 문자	[^[:digit:]]
\\w	단어 문자	[[:word:]]
\\W	단어를 제외한 문자	[^[:word:]]
\\s	공백 문자	[[:space:]]
\\S	공백을 제외한 문자	[^[:space:]]
\\b	A boundary word	

예시

다음 표는 POSIX 연산자를 사용한 패턴 일치의 예를 나타낸 것입니다.

표현식	반환
'abc' ~ 'abc'	True
'abc' ~ 'a'	True
'abc' ~ 'A'	False
'abc' ~ '.*(b|d).*'	True
'abc' ~ '(b|c).*'	True
'AbcAbcdefgefg12efgefg12' ~ '((Ab)?c)+d((efg)+(12))+'	True
'aaaaaab11111xy' ~ 'a{6}.[1]{5}(x|y){2}'	True
'$0.87' ~ '\\$[0-9]+(\\.[0-9][0-9])?'	True
'ab c' ~ '[[:space:]]'	True

표현식	반환
'ab c' ~ '\\s'	True
' ' ~ '\\S'	False

다음은 이름에 E 또는 H가 포함된 도시를 찾는 예입니다.

```
SELECT DISTINCT city FROM users
WHERE city ~ '.*E.*|.*H.*' ORDER BY city LIMIT 5;

      city
-----------------
 Agoura Hills
 Auburn Hills
 Benton Harbor
 Beverly Hills
 Chicago Heights
```

다음 예는 이름에 E 또는 H가 포함되지 않은 도시를 찾습니다.

```
SELECT DISTINCT city FROM users WHERE city !~ '.*E.*|.*H.*' ORDER BY city LIMIT 5;

      city
-----------------
 Aberdeen
 Abilene
 Ada
 Agat
 Agawam
```

다음은 기본 이스케이프 문자열('\\')을 사용하여 "마침표"가 포함된 문자열을 찾는 예입니다.

```
SELECT venuename FROM venue
WHERE venuename ~ '.*\\..*'
ORDER BY venueid;

          venuename
------------------------------
 St. Pete Times Forum
 Jobing.com Arena
```

```
 Hubert H. Humphrey Metrodome
 U.S. Cellular Field
 Superpages.com Center
 E.J. Nutter Center
 Bernard B. Jacobs Theatre
 St. James Theatre
```

BETWEEN 범위 조건

BETWEEN 조건은 키워드 BETWEEN과 AND를 사용하여 값의 범위에 대한 표현식의 포함 여부를 테스트합니다.

구문

```
expression [ NOT ] BETWEEN expression AND expression
```

표현식은 숫자, 문자 또는 날짜/시간 데이터 형식이 될 수 있지만 서로 호환 가능해야 합니다. 범위는 모든 값을 포함합니다.

예시

다음은 티켓 2장, 3장 또는 4장 중에서 판매가 등록된 티켓의 거래 수를 계산하는 예입니다.

```
select count(*) from sales
where qtysold between 2 and 4;

count
--------
104021
(1 row)
```

범위 조건에는 시작 값과 종료 값이 모두 포함됩니다.

```
select min(dateid), max(dateid) from sales
where dateid between 1900 and 1910;

min  | max
-----+-----
1900 | 1910
```

범위 조건에서 첫 번째 표현식은 두 번째 표현식보다 값이 작아야 하고, 두 번째 표현식은 첫 번째 표현식보다 값이 커야 합니다. 다음은 표현식의 값으로 인해 항상 0개의 행을 반환하는 예입니다.

```
select count(*) from sales
where qtysold between 4 and 2;

count
-------
0
(1 row)
```

하지만 NOT 한정자를 적용하면 논리가 반전되어 모든 행의 수를 반환합니다.

```
select count(*) from sales
where qtysold not between 4 and 2;

count
--------
172456
(1 row)
```

다음은 좌석 수가 20,000~50,000석인 장소 목록을 반환하는 쿼리입니다.

```
select venueid, venuename, venueseats from venue
where venueseats between 20000 and 50000
order by venueseats desc;

venueid |       venuename               | venueseats
---------+-------------------------------+------------
116 | Busch Stadium                   |      49660
106 | Rangers BallPark in Arlington |      49115
96 | Oriole Park at Camden Yards   |      48876
...
(22 rows)
```

다음 예에서는 날짜 값에 BETWEEN을 사용하는 방법을 보여 줍니다.

```
select salesid, qtysold, pricepaid, commission, saletime
from sales
where eventid between 1000 and 2000
   and saletime between '2008-01-01' and '2008-01-03'
order by saletime asc;
```

```
salesid | qtysold | pricepaid | commission |   saletime
--------+---------+-----------+------------+---------------
  65082 |       4 |       472 |       70.8 | 1/1/2008 06:06
 110917 |       1 |       337 |      50.55 | 1/1/2008 07:05
 112103 |       1 |       241 |      36.15 | 1/2/2008 03:15
 137882 |       3 |      1473 |     220.95 | 1/2/2008 05:18
  40331 |       2 |        58 |        8.7 | 1/2/2008 05:57
 110918 |       3 |      1011 |     151.65 | 1/2/2008 07:17
  96274 |       1 |       104 |       15.6 | 1/2/2008 07:18
 150499 |       3 |       135 |      20.25 | 1/2/2008 07:20
  68413 |       2 |       158 |       23.7 | 1/2/2008 08:12
```

BETWEEN의 범위는 포함되지만 날짜는 기본적으로 00:00:00의 시간 값을 가집니다. 샘플 쿼리에 대해 유일하게 유효한 1월 3일 행은 판매 시간이 `1/3/2008 00:00:00`인 행입니다.

NULL 조건

NULL 조건은 값이 누락되거나 알 수 없는 경우 NULL 여부를 테스트합니다.

구문

```
expression IS [ NOT ] NULL
```

인수

expression

열 같은 모든 표현식입니다.

IS NULL

표현식의 값이 NULL일 때는 true이고, 표현식에 값이 있을 때는 false입니다.

IS NOT NULL

표현식의 값이 NULL일 때는 false이고, 표현식에 값이 있을 때는 true입니다.

예제

다음은 SALES 테이블의 QTYSOLD 필드에 NULL이 포함된 횟수를 나타내는 예입니다.

```
select count(*) from sales
```

```
where qtysold is null;
count
-------
0
(1 row)
```

EXISTS 조건

EXISTS 조건은 하위 쿼리에 대한 행의 존재 유무를 테스트한 후 하위 쿼리에서 행이 1개 이상 존재하면 true를 반환합니다. NOT을 지정하는 경우에는 하위 쿼리에 행이 없을 때 true를 반환합니다.

구문

```
[ NOT ] EXISTS (table_subquery)
```

인수

exists

table_subquery가 행을 1개 이상 반환하면 true입니다.

not_exists

table_subquery가 행을 하나도 반환하지 않으면 true입니다.

table_subquery

열이 1개 이상, 그리고 행이 1개 이상 포함된 테이블로 평가되는 하위 쿼리입니다.

예제

다음은 유형에 상관없이 판매가 이루어진 날짜마다 각각 한 번씩 날짜 식별자를 모두 반환하는 예입니다.

```
select dateid from date
where exists (
select 1 from sales
where date.dateid = sales.dateid
)
order by dateid;

dateid
--------
```

```
1827
1828
1829
...
```

IN 조건

IN 조건은 값 집합 또는 하위 쿼리에서 값의 멤버십 여부를 테스트합니다.

구문

```
expression [ NOT ] IN (expr_list | table_subquery)
```

인수

expression

expr_list 또는 table_subquery를 대상으로 평가되는 숫자, 문자 또는 날짜/시간 표현식으로서 해당 목록이나 하위 쿼리의 데이터 형식과 호환되어야 합니다.

expr_list

쉼표로 구분된 표현식 1개 이상, 또는 쉼표로 구분된 표현식 집합 1개 이상이며 괄호로 경계를 표시합니다.

table_subquery

행이 1개 이상 포함되어 있지만 select 목록의 열은 1개로 제한된 테이블로 평가되는 하위 쿼리입니다.

IN | NOT IN

IN은 표현식이 표현식 목록 또는 쿼리의 멤버일 때 true를 반환합니다. NOT IN은 표현식이 멤버가 아닐 때 true를 반환합니다. IN과 NOT IN은 expression이 NULL을 산출하는 경우, 혹은 expr_list 또는 table_subquery 값이 하나도 일치하지 않고 두 비교 행 중 적어도 하나가 NULL을 산출하는 경우에는 NULL과 함께 아무런 행도 반환되지 않습니다.

예시

다음 조건은 나열된 값일 때만 true로 평가됩니다.

```
qtysold in (2, 4, 5)
date.day in ('Mon', 'Tues')
```

```
date.month not in ('Oct', 'Nov', 'Dec')
```

대용량 IN 목록의 최적화

값이 10개 이상인 IN 목록은 쿼리 성능의 최적화를 위해 내부에서 스칼라 배열로 평가됩니다. 값이 10개 미만인 IN 목록은 OR 조건자의 연속으로 평가됩니다. 이러한 최적화는 SMALLINT, INTEGER, BIGINT, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE, TIMESTAMP 및 TIMESTAMPTZ 데이터 형식에서 지원됩니다.

이러한 최적화의 효과는 다음 쿼리에서 EXPLAIN 출력을 보면 알 수 있습니다. 예:

```
explain select * from sales
QUERY PLAN
------------------------------------------------------------------------
XN Seq Scan on sales  (cost=0.00..6035.96 rows=86228 width=53)
Filter: (salesid = ANY ('{1,2,3,4,5,6,7,8,9,10,11}'::integer[]))
(2 rows)
```

SQL 명령

SQL 언어는 데이터베이스 객체 생성 및 조작, 쿼리 실행, 테이블 로드, 테이블에 있는 데이터 수정에 사용하는 명령으로 구성됩니다.

Amazon Redshift는 PostgreSQL을 기반으로 합니다. Amazon Redshift와 PostgreSQL은 데이터웨어하우스 애플리케이션을 설계하고 개발할 때 숙지해야 할 몇 가지 중요한 차이점이 있습니다. Amazon Redshift SQL이 PostgreSQL과 어떻게 다른지 자세히 알아보려면 Amazon Redshift 및 PostgreSQL 섹션을 참조하세요.

> **Note**
>
> 단일 SQL 문의 최대 크기는 16MB입니다.

주제

- ABORT
- ALTER DATABASE
- ALTER DATASHARE
- ALTER DEFAULT PRIVILEGES

- ALTER EXTERNAL VIEW
- ALTER FUNCTION
- ALTER GROUP
- ALTER IDENTITY PROVIDER
- 마스킹 정책 변경
- ALTER MATERIALIZED VIEW
- ALTER RLS POLICY
- 역할 변경
- ALTER PROCEDURE
- ALTER SCHEMA
- ALTER SYSTEM
- ALTER TABLE
- ALTER TABLE APPEND
- ALTER USER
- ANALYZE
- ANALYZE COMPRESSION
- 마스킹 정책 연결
- ATTACH RLS POLICY
- BEGIN
- CALL
- CANCEL
- CLOSE
- COMMENT
- COMMIT
- COPY
- 데이터베이스 생성
- CREATE DATASHARE
- CREATE EXTERNAL FUNCTION
- CREATE EXTERNAL MODEL
- CREATE EXTERNAL SCHEMA

- CREATE EXTERNAL TABLE
- CREATE EXTERNAL VIEW
- CREATE FUNCTION
- create group
- CREATE IDENTITY PROVIDER
- CREATE LIBRARY
- 마스킹 정책 생성
- CREATE MATERIALIZED VIEW
- CREATE MODEL
- CREATE PROCEDURE
- CREATE RLS POLICY
- 역할 생성
- CREATE SCHEMA
- CREATE TABLE
- CREATE TABLE AS
- 사용자 생성
- CREATE VIEW
- DEALLOCATE
- DECLARE
- DELETE
- DESC DATASHARE
- DESC IDENTITY PROVIDER
- 마스킹 정책 분리
- DETACH RLS POLICY
- DROP DATABASE
- DROP DATASHARE
- DROP EXTERNAL VIEW
- DROP FUNCTION
- DROP GROUP
- DROP IDENTITY PROVIDER

- DROP LIBRARY
- 마스킹 정책 삭제
- DROP MODEL
- DROP MATERIALIZED VIEW
- DROP PROCEDURE
- DROP RLS POLICY
- DROP ROLE
- DROP SCHEMA
- DROP TABLE
- DROP USER
- DROP VIEW
- END
- EXECUTE
- EXPLAIN
- FETCH
- GRANT
- INSERT
- INSERT(외부 테이블)
- LOCK
- MERGE
- PREPARE
- REFRESH MATERIALIZED VIEW
- reset
- REVOKE
- ROLLBACK
- SELECT
- SELECT INTO
- SET
- SET SESSION AUTHORIZATION
- SET SESSION CHARACTERISTICS

- SET
- SHOW COLUMNS
- SHOW EXTERNAL TABLE
- SHOW DATABASES
- SHOW GRANTS
- SHOW MODEL
- SHOW DATASHARES
- SHOW PROCEDURE
- SHOW SCHEMAS
- SHOW TABLE
- SHOW TABLES
- SHOW VIEW
- START TRANSACTION
- TRUNCATE
- UNLOAD
- UPDATE
- VACUUM

ABORT

현재 실행 중인 트랜잭션을 중지하고 그 트랜잭션에서 이루어진 모든 업데이트를 삭제합니다. ABORT는 이미 완료된 트랜잭션에는 아무런 영향도 미치지 않습니다.

이 명령은 ROLLBACK 명령과 똑같은 기능을 수행합니다. 자세한 내용은 ROLLBACK을 참조하세요.

구문

```
ABORT [ WORK | TRANSACTION ]
```

파라미터

Work

선택적 키워드입니다.

TRANSACTION

선택적 키워드: WORK와 TRANSACTION은 동의어입니다.

예제

다음 예에서는 테이블을 생성한 다음에 데이터가 테이블에 삽입되는 트랜잭션을 시작합니다. 그런 다음 ABORT 명령으로 데이터 삽입을 롤백하여 테이블이 비어 있는 상태로 둡니다.

다음 명령을 실행하면 MOVIE_GROSS라는 예 테이블이 생성됩니다.

```
create table movie_gross( name varchar(30), gross bigint );
```

다음 명령 세트는 테이블에 2개의 데이터 행을 삽입하는 트랜잭션을 시작합니다.

```
begin;

insert into movie_gross values ( 'Raiders of the Lost Ark', 23400000);

insert into movie_gross values ( 'Star Wars', 10000000 );
```

다음으로, 아래 명령을 실행하면 데이터가 올바로 삽입되었음을 보여주기 위해 테이블에서 해당 데이터가 선택됩니다.

```
select * from movie_gross;
```

명령 출력에는 두 행 모두 올바로 삽입된 것으로 표시됩니다.

```
         name           |  gross
------------------------+----------
Raiders of the Lost Ark | 23400000
Star Wars               | 10000000
(2 rows)
```

이제는 다음 명령으로 트랜잭션이 시작된 지점으로 데이터 변경 내용을 롤백합니다.

```
abort;
```

테이블에서 데이터를 선택하면 빈 테이블이 표시됩니다.

```
select * from movie_gross;

 name | gross
------+-------
(0 rows)
```

ALTER DATABASE

데이터베이스의 속성을 변경합니다.

필수 권한

ALTER DATABASE를 사용하려면 다음 권한 중 하나가 필요합니다.

- 수퍼유저
- ALTER DATABASE 권한을 가진 사용자
- 데이터베이스 소유자

구문

```
ALTER DATABASE database_name
{ RENAME TO new_name
| OWNER TO new_owner
| CONNECTION LIMIT { limit | UNLIMITED }
| COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE }
| ISOLATION LEVEL { SERIALIZABLE | SNAPSHOT }
| INTEGRATION {{SET REFRESH_INTERVAL <interval>} | REFRESH {{ ALL | INERROR } TABLES
 [IN SCHEMA schema [, ...]] | TABLE schema.table [, ...]}}
}
```

파라미터

database_name

변경할 데이터베이스의 이름입니다. 일반적으로, 현재 연결되어 있지 않은 데이터베이스를 변경하며, 경우에 따라 후속 세션에서만 변경 사항이 적용됩니다. 현재 데이터베이스의 소유자를 변경할 수는 있지만 이름을 바꿀 수는 없습니다.

```
alter database tickit rename to newtickit;
```

```
ERROR:  current database may not be renamed
```

RENAME TO

지정된 데이터베이스의 이름을 바꿉니다. 유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요. dev, padb_harvest, template0, template1 또는 sys:internal 데이터베이스의 이름을 바꿀 수 없고, 현재 데이터베이스의 이름도 바꿀 수 없습니다. 데이터베이스 소유자 또는 superuser (p. 881)만이 데이터베이스 이름을 바꿀 수 있고, 수퍼유저가 아닌 소유자 역시 CREATEDB 권한을 가져야 합니다.

new_name

새 데이터베이스 이름입니다.

OWNER TO

지정된 데이터베이스의 소유자를 변경합니다. 현재 데이터베이스 또는 다른 데이터베이스의 소유자를 변경할 수 있습니다. 수퍼유저만이 소유자를 변경할 수 있습니다.

new_owner

새 데이터베이스 소유자입니다. 새 소유자는 쓰기 권한을 가진 기존 데이터베이스 사용자여야 합니다. 사용자 권한에 대한 자세한 내용은 GRANT 섹션을 참조하세요.

CONNECTION LIMIT { limit | UNLIMITED }

사용자가 동시에 열어놓을 수 있는 데이터베이스 연결의 최대 개수입니다. 수퍼유저에 대해서는 이 제한이 적용되지 않습니다. 최대 동시 연결 수를 허용하려면 UNLIMITED 키워드를 사용하십시오. 각 사용자에 대한 연결 개수 제한이 적용될 수도 있습니다. 자세한 내용은 사용자 생성 단원을 참조하십시오. 기본값은 UNLIMITED입니다. 현재 연결을 보려면 STV_SESSIONS 시스템 뷰를 쿼리하십시오.

> **Note**
>
> 사용자 및 데이터베이스 연결 제한이 모두 적용되는 경우 사용되지 않는 연결 슬롯은 사용자가 연결 시도 시 양쪽 제한 범위 내에서 모두 사용 가능해야 합니다.

COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE }

문자열 검색 또는 비교가 대/소문자를 구분하는지, 구분하지 않는지를 지정하는 절입니다.

비어 있는 현재 데이터베이스의 대/소문자 구분을 변경할 수 있습니다.

대/소문자 구분을 변경하려면 현재 데이터베이스에 대한 권한이 있어야 합니다. CREATE DATABASE 권한을 가진 슈퍼 사용자 또는 데이터베이스 소유자는 데이터베이스 대/소문자 구분을 변경할 수도 있습니다.

ISOLATION LEVEL { SERIALIZABLE | SNAPSHOT }

데이터베이스에 대해 쿼리가 실행될 때 사용되는 격리 수준을 지정하는 절입니다.

- SERIALIZABLE 격리 - 동시 트랜잭션에 대한 완전한 직렬화 기능을 제공합니다. 자세한 내용은 직렬화 가능 격리 단원을 참조하십시오.
- SNAPSHOT 격리 - 업데이트 및 삭제 충돌로부터 보호하는 격리 수준을 제공합니다.

격리 수준에 대한 자세한 내용은 데이터베이스 생성 섹션을 참조하세요.

데이터베이스의 격리 수준을 변경할 경우 다음 사항을 고려하세요.

- 데이터베이스 격리 수준을 변경하려면 현재 데이터베이스에 대해 슈퍼 사용자 또는 데이터베이스 생성 권한이 있어야 합니다.
- dev 데이터베이스의 격리 수준은 변경할 수 없습니다.
- 트랜잭션 블록 내에서는 격리 수준을 변경할 수 없습니다.
- 다른 사용자가 데이터베이스에 연결되어 있으면 격리 수준 변경 명령이 실패합니다.
- 격리 수준 변경 명령으로 현재 세션의 격리 수준 설정을 변경할 수 있습니다.

INTEGRATION REFRESH {{ ALL | INERROR } TABLES [IN SCHEMA schema [, ...]] | TABLE schema.table [, ...]}

Amazon Redshift가 지정된 스키마 또는 테이블에서 오류가 있는 테이블을 모두 새로 고칠지를 지정하는 절입니다. 새로 고침은 지정된 스키마 또는 테이블의 테이블이 소스 데이터베이스에서 완전히 복제되도록 트리거합니다.

자세한 내용은 Amazon Redshift 관리 안내서의 제로 ETL 통합 작업을 참조하세요. 통합 상태에 대한 자세한 내용은 SVV_INTEGRATION_TABLE_STATE 및 SVV_INTEGRATION 섹션을 참조하세요.

INTEGRATION {SET REFRESH_INTERVAL <interval>}

SET REFRESH_INTERVAL 절은 제로 ETL 소스에서 대상 데이터베이스로 데이터를 새로고침하기 위해 대략적인 시간 간격을 초 단위로 설정합니다. 소스 유형이 Aurora MySQL, Aurora PostgreSQL 또는 RDS for MySQL인 제로 ETL 통합의 경우 `interval`을 0~432,000초(5일)로 설정할 수 있습니다. Amazon DynamoDB 제로 ETL 통합의 경우 `interval`을 900~432,000초(15분~5일)로 설정할 수 있습니다.

제로 ETL 통합을 사용하여 데이터베이스를 생성하는 방법에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 Amazon Redshift에서 대상 데이터베이스 생성을 참조하세요.

사용 노트

ALTER DATABASE 명령은 현재 세션이 아닌 후속 세션에 적용됩니다. 변경 효과를 보려면 변경된 데이터베이스에 다시 연결해야 합니다.

예시

다음 예에서는 TICKIT_SANDBOX라는 데이터베이스 이름을 TICKIT_TEST로 바꿉니다.

```
alter database tickit_sandbox rename to tickit_test;
```

다음 예에서는 TICKIT 데이터베이스(현재 데이터베이스)의 소유자를 DWUSER로 변경합니다.

```
alter database tickit owner to dwuser;
```

다음 예에서는 sampledb 데이터베이스의 데이터베이스 대/소문자 구분을 변경합니다.

```
ALTER DATABASE sampledb COLLATE CASE_INSENSITIVE;
```

다음 예에서는 **sampledb** 데이터베이스를 SNAPSHOT 격리 수준으로 변경합니다.

```
ALTER DATABASE sampledb ISOLATION LEVEL SNAPSHOT;
```

다음 예시는 제로 ETL 통합에서 **sample_integration_db** 데이터베이스의 테이블 **schema1.sample_table1** 및 **schema2.sample_table2**를 새로 고칩니다.

```
ALTER DATABASE sample_integration_db INTEGRATION REFRESH TABLE schema1.sample_table1,
 schema2.sample_table2;
```

다음 예시는 제로 ETL 통합 내에서 동기화된 테이블과 실패한 테이블을 모두 새로 고칩니다.

```
ALTER DATABASE sample_integration_db INTEGRATION REFRESH ALL tables;
```

다음 예제에서는 제로 ETL 통합의 새로고침 간격을 600초로 설정합니다.

```
ALTER DATABASE sample_integration_db INTEGRATION SET REFRESH_INTERVAL 600;
```

다음 예제는 **sample_schema** 스키마에서 ErrorState인 모든 테이블을 새로고침합니다.

```
ALTER DATABASE sample_integration_db INTEGRATION REFRESH INERROR TABLES in SCHEMA
 sample_schema;
```

ALTER DATASHARE

datashare의 정의를 변경합니다. ALTER DATASHARE를 사용하여 객체를 추가하거나 제거할 수 있습니다. 현재 데이터베이스의 데이터 공유만 변경할 수 있습니다. 연결된 데이터베이스에서 객체를 데이터 공유에 추가하거나 제거합니다. 추가하거나 제거할 datashare 객체에 대한 필수 권한이 있는 datashare 소유자는 datashare를 변경할 수 있습니다.

필수 권한

ALTER DATASHARE에 필요한 권한은 다음과 같습니다.

- 슈퍼 사용자.
- ALTER DATASHARE 권한을 가진 사용자.
- datashare에 대한 ALTER 또는 ALL 권한을 가진 사용자.
- datashare에 특정 객체를 추가하려면 사용자에게 객체에 대한 권한이 있어야 합니다. 이 경우 사용자는 객체 소유자이거나 객체에 대한 SELECT, USAGE 또는 ALL 권한이 있어야 합니다.

구문

다음 구문은 datashare에 객체를 추가하거나 제거하는 방법을 보여줍니다.

```
ALTER DATASHARE datashare_name { ADD | REMOVE } {
TABLE schema.table [, ...]
| SCHEMA schema [, ...]
| FUNCTION schema.sql_udf (argtype,...) [, ...]
| ALL TABLES IN SCHEMA schema [, ...]
| ALL FUNCTIONS IN SCHEMA schema [, ...] }
```

다음 구문은 datashare의 속성을 구성하는 방법을 보여줍니다.

```
ALTER DATASHARE datashare_name {
[ SET PUBLICACCESSIBLE [=] TRUE | FALSE ]
[ SET INCLUDENEW [=] TRUE | FALSE FOR SCHEMA schema ] }
```

파라미터

datashare_name

변경할 datashare의 이름입니다.

ADD | REMOVE

datashare에서 객체를 추가할지 아니면 제거할지 지정하는 절입니다.

TABLE schema.table [, ...]

datashare에 추가할 지정된 스키마의 테이블 또는 뷰의 이름입니다.

SCHEMA schema [, ...]

datashare에 추가할 스키마의 이름입니다.

FUNCTION schema.sql_udf (argtype,...) [, ...]

datashare에 추가할 인수 유형이 있는 사용자 정의 SQL 함수의 이름입니다.

ALL TABLES IN SCHEMA schema [, ...]

지정된 스키마의 모든 테이블과 뷰를 datashare에 추가할지 여부를 지정하는 절입니다.

ALL FUNCTIONS IN SCHEMA schema [, ...] }

지정된 스키마의 모든 함수를 datashare에 추가하도록 지정하는 절입니다.

[SET PUBLICACCESSIBLE [=] TRUE | FALSE]

공개적으로 액세스할 수 있는 클러스터와 datashare를 공유할 수 있는지 여부를 지정하는 절입니다.

[SET INCLUDENEW [=] TRUE | FALSE FOR SCHEMA schema]

지정된 스키마에서 생성된 향후 테이블, 뷰 또는 SQL 사용자 정의 함수(UDF)를 datashare에 추가할지 여부를 지정하는 절입니다. 지정된 스키마의 현재 테이블, 뷰 또는 SQL UDF는 datashare에 추가되지 않습니다. 슈퍼 사용자만 각 datashare-스키마 페어에 대해 이 속성을 변경할 수 있습니다. 기본적으로 INCLUDENEW 절은 false입니다.

ALTER DATASHARE 사용 참고 사항

- 다음 사용자가 datashare를 변경할 수 있습니다.
  - 슈퍼 사용자
  - datashare의 소유자입니다.

  - datashare에 대한 ALTER 또는 ALL 권한을 가진 사용자
- datashare에 특정 객체를 추가하려면 이러한 사용자에게 객체에 대한 올바른 권한이 있어야 합니다. 사용자는 객체 소유자이거나 객체에 대한 SELECT, USAGE 또는 ALL 권한이 있어야 합니다.
- 스키마, 테이블, 일반 뷰, 후기 바인딩 뷰, 구체화된 뷰 및 SQL 사용자 정의 함수(UDF)를 공유할 수 있습니다. 스키마에 객체를 추가하기 전에 먼저 datashare에 스키마를 추가합니다.

  스키마를 추가할 때 Amazon Redshift는 스키마 아래에 모든 객체를 추가하지 않습니다. 명시적으로 스키마를 추가해야 합니다.
- 공개적으로 액세스할 수 있는 설정이 지정된 상태에서 AWS Data Exchange datashare를 생성하는 것이 좋습니다.
- 일반적으로 ALTER DATASHARE 문을 사용하여 퍼블릭 액세스 가능성을 끄도록 AWS Data Exchange datashare를 변경하지 않는 것이 좋습니다. 그렇게 하면 클러스터에 공개적으로 액세스할 수 있는 경우 datashare에 액세스할 수 있는 AWS 계정의 액세스 권한이 상실됩니다. 이러한 유형의 변경을 수행하면 AWS Data Exchange의 데이터 제품 조건을 위반할 수 있습니다. 이 권장 사항에 대한 예외는 다음을 참조하세요.

  다음 예는 설정이 해제된 상태에서 AWS Data Exchange datashare가 생성될 경우 오류를 보여줍니다.

```
ALTER DATASHARE salesshare SET PUBLICACCESSIBLE FALSE;
ERROR:  Alter of ADX-managed datashare salesshare requires session variable
 datashare_break_glass_session_var to be set to value 'c670ba4db22f4b'
```

  공개적으로 액세스 가능한 설정을 해제하도록 AWS Data Exchange datashare를 변경하려면 다음 변수를 설정하고 ALTER DATASHARE 문을 다시 실행합니다.

```
SET datashare_break_glass_session_var to 'c670ba4db22f4b';
```

```
ALTER DATASHARE salesshare SET PUBLICACCESSIBLE FALSE;
```

  이 경우 Amazon Redshift는 임의의 일회성 값을 생성하여 AWS Data Exchange datashare에 대해 ALTER DATASHARE SET PUBLICACCESSIBLE FALSE를 허용하도록 세션 변수를 설정합니다.

예시

다음 예에서는 datashare `salesshare`에 `public` 스키마를 추가합니다.

```
ALTER DATASHARE salesshare ADD SCHEMA public;
```

다음 예에서는 datashare salesshare에 public.tickit_sales_redshift 테이블을 추가합니다.

```
ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;
```

다음 예에서는 datashare salesshare에 모든 테이블을 추가합니다.

```
ALTER DATASHARE salesshare ADD ALL TABLES IN SCHEMA PUBLIC;
```

다음 예에서는 datashare salesshare에서 public.tickit_sales_redshift 테이블을 제거합니다.

```
ALTER DATASHARE salesshare REMOVE TABLE public.tickit_sales_redshift;
```

ALTER DEFAULT PRIVILEGES

앞으로 지정된 사용자가 생성하는 객체에 적용될 기본 액세스 권한 세트를 정의합니다. 기본적으로 사용자는 자신의 기본 액세스 권한만 변경할 수 있습니다. 수퍼유저만이 다른 사용자의 기본 권한을 지정할 수 있습니다.

역할, 사용자 또는 사용자 그룹에 기본 권한을 적용할 수 있습니다. 현재 데이터베이스에서 생성되는 모든 객체나 지정된 스키마에서만 생성되는 객체에 대해 기본 권한을 전역적으로 설정할 수 있습니다.

기본 권한은 새 객체에만 적용됩니다. ALTER DEFAULT PRIVILEGES를 실행하면 기존 객체에 대한 권한은 변경되지 않습니다. 데이터베이스 또는 스키마 내에서 사용자가 생성하는 모든 현재 및 미래 객체에 대한 권한을 부여하려면 범위 지정 권한을 참조하세요.

데이터베이스 사용자의 기본 권한에 대한 정보를 보려면 PG_DEFAULT_ACL 시스템 카탈로그 테이블을 쿼리하십시오.

권한에 대한 자세한 내용은 GRANT 섹션을 참조하세요.

필수 권한

다음은 ALTER DEFAULT PRIVILEGES에 필요한 권한입니다.

- 수퍼유저

- ALTER DEFAULT PRIVILEGES 권한을 가진 사용자
- 자신의 기본 액세스 권한을 변경하는 사용자
- 액세스할 수 있는 스키마에 대한 권한을 설정하는 사용자

구문

```
ALTER DEFAULT PRIVILEGES
    [ FOR USER target_user [, ...] ]
    [ IN SCHEMA schema_name [, ...] ]
    grant_or_revoke_clause

where grant_or_revoke_clause is one of:

GRANT { { SELECT | INSERT | UPDATE | DELETE | DROP | REFERENCES | TRUNCATE } [,...] |
 ALL [ PRIVILEGES ] }
 ON TABLES
 TO { user_name [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
 ON FUNCTIONS
 TO { user_name [ WITH GRANT OPTION ] |  ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
 ON PROCEDURES
 TO { user_name [ WITH GRANT OPTION ] |  ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

REVOKE [ GRANT OPTION FOR ] { { SELECT | INSERT | UPDATE | DELETE | REFERENCES |
 TRUNCATE } [,...] | ALL [ PRIVILEGES ] }
 ON TABLES
 FROM user_name [, ...] [ RESTRICT ]

REVOKE  { { SELECT | INSERT | UPDATE | DELETE | REFERENCES | TRUNCATE } [,...] | ALL
 [ PRIVILEGES ] }
 ON TABLES
 FROM { ROLE role_name | GROUP group_name | PUBLIC } [, ...] [ RESTRICT ]

REVOKE [ GRANT OPTION FOR ] { EXECUTE | ALL [ PRIVILEGES ] }
 ON FUNCTIONS
 FROM user_name [, ...] [ RESTRICT ]
```

```
REVOKE { EXECUTE | ALL [ PRIVILEGES ] }
 ON FUNCTIONS
 FROM { ROLE role_name | GROUP group_name | PUBLIC } [, ...] [ RESTRICT ]

REVOKE [ GRANT OPTION FOR ] { EXECUTE | ALL [ PRIVILEGES ] }
 ON PROCEDURES
 FROM user_name [, ...] [ RESTRICT ]

REVOKE { EXECUTE | ALL [ PRIVILEGES ] }
 ON PROCEDURES
 FROM { ROLE role_name | GROUP group_name | PUBLIC } [, ...] [ RESTRICT ]
```

파라미터

FOR USER target_user

선택 사항. 기본 권한이 정의되는 사용자의 이름입니다. 수퍼유저만이 다른 사용자의 기본 권한을 지정할 수 있습니다. 기본값은 현재 사용자입니다.

IN SCHEMA schema_name

선택 사항. IN SCHEMA 절이 나타나면 지정된 schema_name에 생성되는 새 객체에 지정된 기본 권한이 적용됩니다. 이 경우, ALTER DEFAULT PRIVILEGES의 대상인 사용자 또는 사용자 그룹은 지정된 스키마에 대해 CREATE 권한을 가져야 합니다. 스키마에 특정한 기본 권한이 기존 전역 기본 권한에 추가됩니다. 기본적으로 전체 데이터베이스에 기본 권한이 전역적으로 적용됩니다.

GRANT

지정된 사용자가 생성하는 모든 새로운 테이블, 뷰, 함수 또는 저장 프로시저에 대해 지정된 사용자나 그룹에게 부여하는 권한 세트입니다. GRANT 명령과 함께 사용할 수 있는 GRANT 절을 이용해 동일한 권한과 옵션을 설정할 수 있습니다.

WITH GRANT OPTION

권한을 받은 사용자가 이번에는 똑같은 권한을 다른 사용자에게 허용할 수 있음을 나타내는 절입니다. 그룹 또는 PUBLIC에는 WITH GRANT OPTION을 허용할 수 없습니다.

TO user_name | ROLE role_name | GROUP group_name

지정된 기본 권한이 적용될 사용자, 역할 또는 사용자 그룹의 이름입니다.

REVOKE

지정된 사용자가 생성하는 모든 새로운 테이블, 함수 또는 저장 프로시저에 대해 지정된 사용자나 그룹에서 취소하는 권한 세트입니다. REVOKE 명령과 함께 사용할 수 있는 REVOKE 절을 이용해 동일한 권한과 옵션을 설정할 수 있습니다.

GRANT OPTION FOR

다른 사용자에게 지정된 권한을 허용하는 옵션만 취소하고 권한 자체를 취소하지는 않는 절입니다. 그룹 또는 PUBLIC에서 GRANT OPTION을 취소할 수 없습니다.

FROM user_name | ROLE role_name | GROUP group_name

지정된 권한이 기본적으로 취소되는 사용자, 역할 또는 사용자 그룹의 이름입니다.

RESTRICT

RESTRICT 옵션은 사용자가 직접 허용한 권한만 취소합니다. 이 값이 기본값입니다.

예시

사용자 그룹 `report_readers`의 사용자가 사용자 `report_admin`에 의해 생성된 모든 테이블과 뷰를 볼 수 있도록 허용하려는 경우를 생각해 봅시다. 이 경우에는 슈퍼 사용자 권한으로 다음 명령을 실행합니다.

```
alter default privileges for user report_admin grant select on tables to group
 report_readers;
```

다음 예에서 첫 번째 명령은 사용자 자신이 생성하는 모든 새로운 테이블과 뷰에 SELECT 권한을 허용합니다.

```
alter default privileges grant select on tables to public;
```

다음 예에서는 사용자 자신이 `sales_admin` 스키마에서 생성하는 모든 새로운 테이블과 뷰에 대해 `sales` 사용자 그룹에 INSERT 권한을 허용합니다.

```
alter default privileges in schema sales grant insert on tables to group sales_admin;
```

다음 예에서는 이전 예의 ALTER DEFAULT PRIVILEGES 명령을 반전합니다.

```
alter default privileges in schema sales revoke insert on tables from group
 sales_admin;
```

기본적으로, PUBLIC 사용자 그룹은 모든 새로운 사용자 정의 함수에 대한 실행 권한이 있습니다. 새 함수에 대한 public 실행 권한을 취소한 다음 dev_test 사용자 그룹에만 실행 권한을 허용하려면 다음 명령을 실행합니다.

```
alter default privileges revoke execute on functions from public;
alter default privileges grant execute on functions to group dev_test;
```

ALTER EXTERNAL VIEW

ALTER EXTERNAL VIEW 명령을 사용하여 외부 뷰를 업데이트하세요. 사용하는 파라미터에 따라 이 뷰를 참조할 수 있는 Amazon Athena 및 Amazon EMR Spark와 같은 다른 SQL 엔진도 영향을 받을 수 있습니다. Data Catalog 뷰에 대한 자세한 내용은 AWS Glue Data Catalog 뷰를 참조하세요.

구문

```
ALTER EXTERNAL VIEW schema_name.view_name
{catalog_name.schema_name.view_name | awsdatacatalog.dbname.view_name |
 external_schema_name.view_name}
[FORCE] { AS (query_definition) | REMOVE DEFINITION }
```

파라미터

schema_name.view_name

AWS Glue 데이터베이스에 연결된 스키마이며, 뷰 이름이 뒤따릅니다.

catalog_name.schema_name.view_name | awsdatacatalog.dbname.view_name | external_schema_name.view_name

뷰를 변경할 때 사용하는 스키마의 표기법입니다. 직접 만든 Glue 데이터베이스인 AWS Glue Data Catalog 또는 직접 만든 외부 스키마를 사용하도록 지정할 수 있습니다. 자세한 내용은 CREATE DATABASE 및 CREATE EXTERNAL SCHEMA를 참조하세요.

FORCE

테이블에서 참조된 객체가 다른 SQL 엔진과 일치하지 않는 경우에도 AWS Lake Formation이 뷰의 정의를 업데이트해야 하는지를 나타냅니다. Lake Formation이 뷰를 업데이트하면 다른 SQL 엔진도 업데이트하기 전까지는 다른 SQL 엔진에 대해서도 해당 뷰가 유효하지 않은 것으로 간주됩니다.

AS query_definition

뷰를 변경하기 위해 Amazon Redshift가 실행하는 SQL 쿼리의 정의입니다.

REMOVE DEFINITION

뷰를 삭제하고 다시 만들지를 나타냅니다. 뷰를 PROTECTED로 표시하려면 삭제한 후 다시 만들어야 합니다.

예시

다음 예시에서는 sample_schema.glue_data_catalog_view라는 데이터 카탈로그 뷰를 변경합니다.

```
ALTER EXTERNAL VIEW sample_schema.glue_data_catalog_view
FORCE
REMOVE DEFINITION
```

ALTER FUNCTION

함수의 이름을 바꾸거나 소유자를 변경합니다. 함수 이름과 데이터 형식은 모두 필수입니다. 소유자 또는 수퍼유저만 함수 이름을 바꿀 수 있습니다. 수퍼유저만 함수의 소유자를 변경할 수 있습니다.

구문

```
ALTER FUNCTION function_name ( { [ py_arg_name py_arg_data_type | sql_arg_data_type }
 [ , ... ] ] )
    RENAME TO new_name
```

```
ALTER FUNCTION function_name ( { [ py_arg_name py_arg_data_type | sql_arg_data_type }
 [ , ... ] ] )
    OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
```

파라미터

function_name

대체할 함수의 이름입니다. 특정 스키마를 사용하려면 현재 검색 경로에 함수의 이름만 지정하거나, schema_name.function_name 형식을 사용합니다.

py_arg_name py_arg_data_type | sql_arg_data_type

선택 사항. Python 사용자 정의 함수의 입력 인수 이름 및 데이터 유형 목록 또는 SQL 사용자 정의 함수의 입력 인수 데이터 유형 목록입니다.

new_name

사용자 정의 함수의 새 이름입니다.

new_owner | CURRENT_USER | SESSION_USER

사용자 정의 함수의 새 소유자입니다.

예시

다음 예에서는 함수의 이름을 first_quarter_revenue에서 quarterly_revenue로 변경합니다.

```
ALTER FUNCTION first_quarter_revenue(bigint, numeric, int)
         RENAME TO quarterly_revenue;
```

다음 예에서는 quarterly_revenue 함수의 소유자를 etl_user로 바꿉니다.

```
ALTER FUNCTION quarterly_revenue(bigint, numeric) OWNER TO etl_user;
```

ALTER GROUP

사용자 그룹을 변경합니다. 이 명령을 사용하여 그룹에 사용자를 추가하거나, 그룹에서 사용자를 삭제하거나, 그룹의 이름을 바꿀 수 있습니다.

구문

```
ALTER GROUP group_name
{
ADD USER username [, ... ] |
DROP USER username [, ... ] |
RENAME TO new_name
}
```

파라미터

group_name

수정할 사용자 그룹의 이름입니다.

ADD

사용자 그룹에 사용자를 추가합니다.

DROP

사용자 그룹에서 사용자를 제거합니다.

사용자 이름

그룹에 추가하거나 그룹에서 삭제할 사용자의 이름입니다.

RENAME TO

사용자 그룹의 이름을 바꿉니다. 2개의 밑줄로 시작하는 그룹 이름은 Amazon Redshift 내부 용도로 예약되어 있습니다. 유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요.

new_name

사용자 그룹의 새 이름입니다.

예시

다음 예에서는 DWUSER로 명명된 사용자를 ADMIN_GROUP 그룹에 추가합니다.

```
ALTER GROUP admin_group
ADD USER dwuser;
```

다음 예에서는 그룹 ADMIN_GROUP의 이름을 ADMINISTRATORS로 바꿉니다.

```
ALTER GROUP admin_group
RENAME TO administrators;
```

다음 예에서는 ADMIN_GROUP 그룹에 두 명의 사용자를 추가합니다.

```
ALTER GROUP admin_group
ADD USER u1, u2;
```

다음 예에서는 ADMIN_GROUP 그룹에서 두 명의 사용자를 삭제합니다.

```
ALTER GROUP admin_group
DROP USER u1, u2;
```

ALTER IDENTITY PROVIDER

자격 증명 공급자를 변경하여 새 파라미터 및 값을 할당합니다. 이 명령을 실행하면 새 값이 할당되기 전에 이전에 설정한 모든 파라미터 값이 삭제됩니다. 슈퍼 사용자만 자격 증명 공급자를 변경할 수 있습니다.

구문

```
ALTER IDENTITY PROVIDER identity_provider_name
[PARAMETERS parameter_string]
[NAMESPACE namespace]
[IAM_ROLE iam_role]
[AUTO_CREATE_ROLES
    [ TRUE [ { INCLUDE | EXCLUDE } GROUPS LIKE filter_pattern] |
      FALSE
    ]
[DISABLE | ENABLE]
```

파라미터

identity_provider_name

새 자격 증명 공급자 이름입니다. 유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요.

parameter_string

특정 자격 증명 공급자에 필요한 파라미터와 값이 포함된 올바른 형식의 JSON 객체를 포함하는 문자열입니다.

네임스페이스

조직 네임스페이스입니다.

iam_role

IAM Identity Center에 대한 연결 권한을 제공하는 IAM 역할입니다. 이 파라미터는 ID 제공업체 유형이 AWSIDC인 경우에만 적용됩니다.

auto_create_roles

역할 자동 생성 기능을 활성화하거나 비활성화합니다. SQL에서 옵션이 제공되지 않으면 기본값은 FALSE이고, 값 없이 옵션이 제공되는 경우 기본값은 TRUE입니다.

그룹을 포함하려면 INCLUDE를 지정합니다. 기본값은 비어 있습니다. 즉, AUTO_CREATES_ROLES가 켜져 있는 경우 모든 그룹이 포함됩니다.

그룹을 제외하려면 EXCLUDE를 지정합니다. 기본값은 비어 있습니다. 즉, AUTO_CREATES_ROLES가 켜져 있는 경우 그룹을 제외하지 않습니다.

DISABLE 또는 ENABLE

ID 제공업체를 켜거나 끕니다. 기본값은 ENABLE입니다.

예시

다음 예에서는 oauth_standard라는 자격 증명 공급자를 변경합니다. 이는 특히 Microsoft Azure AD가 ID 제공업체인 경우에 적용됩니다.

```
ALTER IDENTITY PROVIDER oauth_standard
PARAMETERS '{"issuer":"https://sts.windows.net/2sdfdsf-d475-420d-b5ac-667adad7c702/",
"client_id":"87f4aa26-78b7-410e-bf29-57b39929ef9a",
"client_secret":"BUAH~ewrqewrqwerUUY^%tHe1oNZShoiU7",
"audience":["https://analysis.windows.net/powerbi/connector/AmazonRedshift"]
}'
```

다음 샘플은 ID 제공업체 네임스페이스를 설정하는 방법을 보여줍니다. 이는 Microsoft Azure AD(이전 샘플과 같은 명령문을 따르는 경우) 또는 다른 ID 제공업체에 적용될 수 있습니다. 또한 관리형 애플리케이션을 통해 연결을 설정한 경우 기존 Amazon Redshift 프로비저닝된 클러스터 또는 Amazon Redshift Serverless 작업 그룹을 IAM Identity Center에 연결하는 경우에도 적용할 수 있습니다.

```
ALTER IDENTITY PROVIDER "my-redshift-idc-application"
NAMESPACE 'MYCO';
```

다음 샘플은 IAM 역할을 설정하며 IAM Identity Center와의 Redshift 통합을 구성하는 사용 사례에서 작동합니다.

```
ALTER IDENTITY PROVIDER "my-redshift-idc-application"
IAM_ROLE 'arn:aws:iam::123456789012:role/myadministratorrole';
```

Redshift에서 IAM Identity Center에 대한 연결을 설정하는 방법에 대한 자세한 내용은 Redshift를 IAM Identity Center와 연결하여 사용자에게 Single Sign-On 경험을 제공합니다를 참조하세요.

ID 제공업체 비활성화

다음 샘플은 ID 제공업체를 비활성화하는 방법을 보여줍니다. 비활성화되면 ID 제공업체의 페더레이션 사용자는 클러스터가 다시 활성화될 때까지 클러스터에 로그인할 수 없습니다.

```
ALTER IDENTITY PROVIDER "redshift-idc-app" DISABLE;
```

마스킹 정책 변경

기존 동적 데이터 마스킹 정책을 변경합니다. 동적 데이터 마스킹에 대한 자세한 내용은 동적 데이터 마스킹(동적 데이터 마스킹(미리 보기)을 참조하세요.

sys:secadmin 역할이 부여된 수퍼유저와 사용자 또는 역할은 마스킹 정책을 변경할 수 있습니다.

구문

```
ALTER MASKING POLICY policy_name
   USING (masking_expression);
```

파라미터

policy_name

마스킹 정책의 이름입니다. 이는 데이터베이스에 이미 있는 마스킹 정책의 이름이어야 합니다.

masking_expression

대상 열을 변환하는 데 사용되는 SQL 표현식입니다. 문자열 조작 함수와 같은 데이터 조작 함수를 사용하거나 SQL, Python 또는 AWS Lambda로 작성된 사용자 정의 함수와 함께 작성할 수 있습니다.

표현식은 원래 표현식의 입력 열 및 데이터 유형과 일치해야 합니다. 예를 들어 원래 마스킹 정책의 입력 열이 `sample_1 FLOAT` 및 `sample_2 VARCHAR(10)`인 경우 세 번째 열을 사용하거나 정책이 FLOAT 및 BOOLEAN을 사용하도록 마스킹 정책을 변경할 수 없습니다. 상수를 마스킹 표현식으로 사용하는 경우 입력 유형과 일치하는 유형으로 상수를 명시적으로 변환해야 합니다.

마스킹 표현식에서 사용하는 모든 사용자 정의 함수에 대한 USAGE 권한이 있어야 합니다.

ALTER MATERIALIZED VIEW

구체화된 뷰의 속성을 변경합니다.

구문

```
ALTER MATERIALIZED VIEW mv_name
{
AUTO REFRESH { YES | NO }
| ALTER DISTKEY column_name
| ALTER DISTSTYLE ALL
| ALTER DISTSTYLE EVEN
| ALTER DISTSTYLE KEY DISTKEY column_name
| ALTER DISTSTYLE AUTO
| ALTER [COMPOUND] SORTKEY ( column_name [,...] )
| ALTER SORTKEY AUTO
| ALTER SORTKEY NONE
| ROW LEVEL SECURITY { ON | OFF } [ CONJUNCTION TYPE { AND | OR } ] [FOR DATASHARES]
};
```

파라미터

mv_name

변경할 구체화된 뷰의 이름입니다.

AUTO REFRESH { YES | NO }

구체화된 뷰의 자동 새로 고침을 켜거나 끄는 절입니다. 구체화된 뷰의 자동 새로 고침에 대한 자세한 내용은 구체화된 뷰 새로 고침 섹션을 참조하세요.

ALTER DISTSTYLE ALL

관계의 기존 분산 스타일을 ALL로 변경하는 절입니다. 다음을 고려하세요.

- ALTER DISTSTYLE, ALTER SORTKEY, VACUUM은 동일한 관계에서 동시에 실행할 수 없습니다.
  - VACUUM이 현재 실행 중인 경우 ALTER DISTSTYLE ALL을 실행하면 오류가 반환됩니다.
  - ALTER DISTSTYLE ALL이 실행 중인 경우 백그라운드 VACCUM이 관계에서 시작되지 않습니다.
- 인터리브 정렬 키 및 임시 테이블이 있는 관계에는 ALTER DISTYLE ALL 명령이 지원되지 않습니다.
- 배포 스타일이 이전에 AUTO로 정의된 경우 관계는 더 이상 자동 테이블 최적화의 후보가 아닙니다.

DISTSTYLE ALL에 대한 자세한 내용은 CREATE MATERIALIZED VIEW 섹션으로 이동하세요.

ALTER DISTSTYLE EVEN

관계의 기존 분산 스타일을 EVEN으로 변경하는 절입니다. 다음을 고려하세요.

- ALTER DISTSYTLE, ALTER SORTKEY, VACUUM은 동일한 관계에서 동시에 실행할 수 없습니다.
  - VACUUM이 현재 실행 중인 경우 ALTER DISTSTYLE EVEN을 실행하면 오류가 반환됩니다.
  - ALTER DISTSTYLE EVEN이 실행 중인 경우 백그라운드 VACCUM이 관계에서 시작되지 않습니다.
- 인터리브 정렬 키 및 임시 테이블이 있는 관계에는 ALTER DISTYLE EVEN 명령이 지원되지 않습니다.
- 배포 스타일이 이전에 AUTO로 정의된 경우 관계는 더 이상 자동 테이블 최적화의 후보가 아닙니다.

DISTSTYLE EVEN에 대한 자세한 내용은 CREATE MATERIALIZED VIEW 섹션으로 이동하세요.

ALTER DISTKEY column_name 또는 ALTER DISTSTYLE KEY DISTKEY column_name

관계의 배포 키로 사용되는 열을 변경하는 절입니다. 다음을 고려하세요.

- VACUUM 및 ALTER DISTKEY는 동일한 관계에서 동시에 실행할 수 없습니다.
  - VACUUM이 이미 실행 중인 경우 ALTER DISTKEY가 오류를 반환합니다.
  - ALTER DISTKEY가 실행 중인 경우 백그라운드 VACCUM이 관계에서 시작되지 않습니다.
  - ALTER DISTKEY가 실행 중인 경우 포그라운드 VACCUM이 오류를 반환합니다.
- 관계에서 한 번에 하나의 ALTER DISTKEY 명령만 실행할 수 있습니다.
- 인터리브 정렬 키가 있는 관계에는 ALTER DISTKEY 명령이 지원되지 않습니다.
- 배포 스타일이 이전에 AUTO로 정의된 경우 관계는 더 이상 자동 테이블 최적화의 후보가 아닙니다.

DISTSTYLE KEY를 지정할 때 데이터는 DISTKEY 열에 있는 값을 기준으로 배포됩니다.
DISTSTYLE에 대한 자세한 내용은 CREATE MATERIALIZED VIEW 섹션으로 이동하세요.

ALTER DISTSTYLE AUTO

관계의 기존 분산 스타일을 AUTO로 변경하는 절입니다.

배포 스타일을 AUTO로 변경하면 관계의 배포 스타일이 다음과 같이 설정됩니다.

- DISTSTYLE ALL이 있는 작은 관계는 AUTO(ALL)로 변환됩니다.
- DISTSTYLE EVEN이 있는 작은 관계는 AUTO(ALL)로 변환됩니다.

- DISTSTYLE KEY가 있는 작은 관계는 AUTO(ALL)로 변환됩니다.
- DISTSTYLE ALL이 있는 큰 관계는 AUTO(EVEN)로 변환됩니다.
- DISTSTYLE EVEN이 있는 큰 관계는 AUTO(EVEN)로 변환됩니다.
- DISTSTYLE KEY가 있는 큰 관계는 AUTO(KEY)로 변환되고 DISTKEY는 보존됩니다. 이 경우 Amazon Redshift는 관계를 변경하지 않습니다.

새로운 배포 스타일 또는 키가 쿼리 성능을 향상시킬 것이라고 판단되면 Amazon Redshift는 향후 관계의 배포 스타일이나 키를 변경할 수 있습니다. 예를 들어 Amazon Redshift는 DISTSTYLE이 AUTO(KEY)인 관계를 AUTO(EVEN)로 또는 그 반대로 변환할 수 있습니다. 데이터 재배포 및 잠금을 포함하여 배포 키가 변경될 때의 동작에 대한 자세한 내용은 Amazon Redshift Advisor 권장 사항으로 이동하세요.

DISTSTYLE AUTO에 대한 자세한 내용은 CREATE MATERIALIZED VIEW 섹션으로 이동하세요.

관계의 배포 스타일을 보려면 SVV_TABLE_INFO 시스템 카탈로그 뷰를 쿼리합니다. 자세한 내용은 SVV_TABLE_INFO 섹션을 참조하세요. 관계에 대한 Amazon Redshift Advisor 권장 사항을 보려면 SVV_ALTER_TABLE_RECOMMENDATIONS 시스템 카탈로그 뷰를 쿼리합니다. 자세한 내용은 SVV_ALTER_TABLE_RECOMMENDATIONS 섹션을 참조하세요. Amazon Redshift에서 수행한 작업을 보려면 SVL_AUTO_WORKER_ACTION 시스템 카탈로그 뷰를 쿼리합니다. 자세한 내용은 SVL_AUTO_WORKER_ACTION 섹션을 참조하세요.

ALTER [COMPOUND] SORTKEY (column_name [,...])

관계에 사용되는 정렬 키를 변경하거나 추가하는 절입니다. 임시 테이블에는 ALTER SORTKEY가 지원되지 않습니다.

정렬 키를 변경하면 새 정렬 키 또는 원래 정렬 키에 있는 열의 압축 인코딩이 변경될 수 있습니다. 관계에 대해 명시적으로 정의된 인코딩이 없는 경우 Amazon Redshift는 다음과 같이 압축 인코딩을 자동으로 할당합니다.

- 정렬 키로 정의된 열은 RAW 압축이 할당됩니다.
- BOOLEAN, REAL 또는 DOUBLE PRECISION 데이터 형식으로 정의된 열은 RAW 압축이 할당됩니다.
- SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIME, TIMETZ, TIMESTAMP 또는 TIMESTAMPTZ로 정의된 열에는 AZ64 압축이 할당됩니다.
- CHAR 또는 VARCHAR로 정의된 열에는 LZO 압축이 할당됩니다.

다음을 고려하세요.

- 관계당 정렬 키에 최대 400개의 열을 정의할 수 있습니다.

- 인터리브 정렬 키를 복합 정렬 키 또는 정렬 키 없음으로 변경할 수 있습니다. 그러나 복합 정렬 키를 인터리브 정렬 키로 변경할 수는 없습니다.
- 정렬 키가 이전에 AUTO로 정의된 경우 관계는 더 이상 자동 테이블 최적화의 후보가 아닙니다.
- Amazon Redshift는 정렬 키로 정의된 열에 대해 RAW 인코딩(압축 없음)을 사용하는 것이 좋습니다. 열을 변경하여 정렬 키로 선택하면 열의 압축이 RAW 압축(압축 없음)으로 변경됩니다. 이렇게 하면 관계에 필요한 스토리지 양이 늘어날 수 있습니다. 관계 크기 증가량은 특정 관계 정의와 관계 내용에 따라 다릅니다. 압축에 대한 자세한 내용은 압축 인코딩 섹션으로 이동하세요.

데이터가 관계에 로드되면 데이터는 정렬 키의 순서로 로드됩니다. 정렬 키를 변경하면 Amazon Redshift는 데이터를 재정렬합니다. SORTKEY에 대한 자세한 내용은 CREATE MATERIALIZED VIEW 섹션으로 이동하세요.

ALTER SORTKEY AUTO

대상 관계의 정렬 키를 AUTO로 변경하거나 추가하는 절입니다. 임시 테이블에는 ALTER SORTKEY AUTO가 지원되지 않습니다.

정렬 키를 AUTO로 변경하면 Amazon Redshift는 관계의 기존 정렬 키를 보존합니다.

새로운 정렬 키가 쿼리 성능을 향상시킬 것이라고 판단되면 Amazon Redshift는 향후 관계의 정렬 키를 변경할 수 있습니다.

SORTKEY AUTO에 대한 자세한 내용은 CREATE MATERIALIZED VIEW 섹션으로 이동하세요.

관계의 정렬 키를 보려면 SVV_TABLE_INFO 시스템 카탈로그 뷰를 쿼리합니다. 자세한 내용은 SVV_TABLE_INFO 섹션을 참조하세요. 관계에 대한 Amazon Redshift Advisor 권장 사항을 보려면 SVV_ALTER_TABLE_RECOMMENDATIONS 시스템 카탈로그 뷰를 쿼리합니다. 자세한 내용은 SVV_ALTER_TABLE_RECOMMENDATIONS 섹션을 참조하세요. Amazon Redshift에서 수행한 작업을 보려면 SVL_AUTO_WORKER_ACTION 시스템 카탈로그 뷰를 쿼리합니다. 자세한 내용은 SVL_AUTO_WORKER_ACTION 섹션을 참조하세요.

ALTER SORTKEY NONE

대상 관계의 정렬 키를 제거하는 절입니다.

정렬 키가 이전에 AUTO로 정의된 경우 관계는 더 이상 자동 테이블 최적화의 후보가 아닙니다.

ROW LEVEL SECURITY { ON | OFF } [CONJUNCTION TYPE { AND | OR }] [FOR DATASHARES]

관계에 대한 행 수준 보안을 켜거나 끄는 절입니다.

관계에 대해 행 수준 보안이 켜 있으면 행 수준 보안 정책에서 액세스를 허용하는 행만 읽을 수 있습니다. 관계에 대한 액세스 권한을 부여하는 정책이 없으면 관계에서 행을 볼 수 없습니다. 수퍼유저

및 sys:secadmin 역할을 가진 사용자 또는 역할만 ROW LEVEL SECURITY 절을 설정할 수 있습니다. 자세한 내용은 행 수준 보안 단원을 참조하십시오.

- [CONJUNCTION TYPE { AND | OR }]

  관계에 대해 행 수준 보안 정책의 접속사 유형을 선택할 수 있도록 하는 절입니다. 여러 행 수준 보안 정책이 관계에 연결된 경우 정책을 AND 또는 OR 절과 결합할 수 있습니다. 기본적으로 Amazon Redshift는 RLS 정책을 AND 절과 결합합니다. sys:secadmin 역할이 있는 슈퍼 사용자, 사용자 또는 역할은 이 절을 사용하여 관계에 대한 행 수준 보안 정책의 접속사 유형을 정의할 수 있습니다. 자세한 내용은 사용자별로 여러 정책 결합 단원을 참조하십시오.

- FOR DATASHARES

  RLS로 보호되는 관계에 데이터 공유를 통해 액세스할 수 있는지를 결정하는 절입니다. 기본적으로 RLS로 보호되는 관계는 데이터 공유를 통해 액세스할 수 없습니다. ALTER MATERIALIZED VIEW ROW LEVEL SECURITY 명령을 이 절과 함께 실행하면 관계의 데이터 공유 접근성 속성에만 영향을 줍니다. ROW LEVEL SECURITY 속성은 변경되지 않습니다.

  RLS로 보호되는 관계에 데이터 공유를 통해 액세스할 수 있도록 설정하면 소비자 측 데이터 공유 데이터베이스에서 해당 관계에 행 수준 보안이 적용되지 않습니다. 이 관계는 생산자 측에서 해당 RLS 속성을 유지합니다.

예시

다음 예에서는 tickets_mv 구체화된 뷰를 자동으로 새로 고칠 수 있게 합니다.

```
ALTER MATERIALIZED VIEW tickets_mv AUTO REFRESH YES
```

ALTER MATERIALIZED VIEW에 대한 DISTSTYLE 및 SORTKEY 예시

이 주제의 예제는 ALTER MATERIALIZED VIEW를 사용하여 DISTSTYLE 및 SORTKEY 변경을 수행하는 방법을 보여줍니다.

다음 예제 쿼리는 샘플 기본 테이블을 사용하여 DISTSTYLE KEY DISTKEY 열을 변경하는 방법을 보여줍니다.

```
CREATE TABLE base_inventory(
  inv_date_sk int4 NOT NULL,
  inv_item_sk int4 NOT NULL,
  inv_warehouse_sk int4 NOT NULL,
  inv_quantity_on_hand int4
```

```
);

INSERT INTO base_inventory VALUES(1,1,1,1);

CREATE materialized VIEW inventory diststyle even AS SELECT * FROM base_inventory;
SELECT "table", diststyle FROM svv_table_info WHERE "table" = 'inventory';

ALTER materialized VIEW inventory ALTER diststyle KEY distkey inv_warehouse_sk;
SELECT "table", diststyle FROM svv_table_info WHERE "table" = 'inventory';

ALTER materialized VIEW inventory ALTER distkey inv_item_sk;
SELECT "table", diststyle FROM svv_table_info WHERE "table" = 'inventory';

DROP TABLE base_inventory CASCADE;
```

구체화된 뷰를 DISTSTYLE ALL로 변경:

```
CREATE TABLE base_inventory(
  inv_date_sk int4 NOT NULL,
  inv_item_sk int4 NOT NULL,
  inv_warehouse_sk int4 NOT NULL,
  inv_quantity_on_hand int4
);

INSERT INTO base_inventory VALUES(1,1,1,1);

CREATE materialized VIEW inventory diststyle even AS SELECT * FROM base_inventory;
SELECT "table", diststyle FROM svv_table_info WHERE "table" = 'inventory';

ALTER MATERIALIZED VIEW inventory ALTER diststyle ALL;
SELECT "table", diststyle FROM svv_table_info WHERE "table" = 'inventory';

DROP TABLE base_inventory CASCADE;
```

다음 명령은 샘플 기본 테이블을 사용한 ALTER MARTIALIZED VIEW SORTKEY 예시를 보여줍니다.

```
CREATE TABLE base_inventory (c0 int, c1 int);

INSERT INTO base_inventory VALUES(1,1);

CREATE materialized VIEW inventory interleaved sortkey(c0, c1) AS SELECT * FROM
 base_inventory;
SELECT "table", sortkey1 FROM svv_table_info WHERE "table" = 'inventory';
```

```
ALTER materialized VIEW inventory ALTER sortkey(c0, c1);
SELECT "table", diststyle, sortkey_num FROM svv_table_info WHERE "table" = 'inventory';

ALTER materialized VIEW inventory ALTER sortkey NONE;
SELECT "table", diststyle, sortkey_num FROM svv_table_info WHERE "table" = 'inventory';

ALTER materialized VIEW inventory ALTER sortkey(c0);
SELECT "table", diststyle, sortkey_num FROM svv_table_info WHERE "table" = 'inventory';

DROP TABLE base_inventory CASCADE;
```

ALTER RLS POLICY

테이블의 기존 행 수준 보안 정책을 변경합니다.

슈퍼유저와 `sys:secadmin` 역할이 부여된 사용자 또는 역할은 정책을 변경할 수 있습니다.

구문

```
ALTER RLS POLICY policy_name
USING ( using_predicate_exp );
```

파라미터

policy_name

정책의 이름입니다.

USING (using_predicate_exp)

쿼리의 WHERE 절에 적용되는 필터를 지정합니다. Amazon Redshift는 쿼리 수준 사용자 조건자를 지정하기 전에 정책 조건자를 적용합니다. 예를 들어 **`current_user = 'joe' and price > 10`**은 가격이 10 USD보다 큰 레코드만 Joe에게 표시하도록 제한합니다.

표현식에서는 이름이 policy_name인 정책을 생성하는 데 사용된 CREATE RLS POLICY 명령문의 WITH 절에 선언된 변수에 액세스할 수 있습니다.

예시

다음 예에서는 RLS 정책을 변경합니다.

```
-- First create an RLS policy that limits access to rows where catgroup is 'concerts'.
CREATE RLS POLICY policy_concerts
WITH (catgroup VARCHAR(10))
USING (catgroup = 'concerts');

-- Then, alter the RLS policy to only show rows where catgroup is 'piano concerts'.
ALTER RLS POLICY policy_concerts
USING (catgroup = 'piano concerts');
```

역할 변경

역할의 이름을 바꾸거나 소유자를 변경합니다. Amazon Redshift 시스템 정의 역할 목록은 the section called “Amazon Redshift 시스템 정의 역할” 섹션을 참조하세요.

필수 권한

ALTER ROLE에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- ALTER USER 권한이 있는 사용자

구문

```
ALTER ROLE role [ WITH ]
  { { RENAME TO role } | { OWNER TO user_name } }[, ...]
  [ EXTERNALID TO external_id ]
```

파라미터

역할

변경할 역할의 이름.

RENAME TO

역할의 새 이름.

user_name 소유자

역할의 새 소유자.

EXTERNALID TO external_id

자격 증명 공급자와 연결된 역할의 새 외부 ID입니다. 자세한 내용은 Native identity provider (IdP) federation for Amazon Redshift(Amazon Redshift용 네이티브 자격 증명 공급자(IdP) 페더레이션)를 참조하세요.

예시

다음 예에서는 역할의 이름을 sample_role1에서 sample_role2로 변경합니다.

```
ALTER ROLE sample_role1 RENAME TO sample_role2;
```

다음 예에서는 역할의 소유자를 바꿉니다.

```
ALTER ROLE sample_role1 WITH OWNER TO user1
```

ALTER ROLE의 구문은 다음과 같이 ALTER PROCEDURE와 유사합니다.

```
ALTER PROCEDURE first_quarter_revenue(bigint, numeric) RENAME TO quarterly_revenue;
```

다음 예에서는 프로시저의 소유자를 etl_user로 변경합니다.

```
ALTER PROCEDURE quarterly_revenue(bigint, numeric) OWNER TO etl_user;
```

다음 예에서는 자격 증명 공급자와 연결된 새 외부 ID로 역할 sample_role1을 업데이트합니다.

```
ALTER ROLE sample_role1 EXTERNALID TO "XYZ456";
```

ALTER PROCEDURE

프로시저의 이름을 바꾸거나 소유자를 변경합니다. 프로시저 이름과 데이터 형식 또는 서명은 모두 필수입니다. 소유자 또는 수퍼유저만 프로시저의 이름을 바꿀 수 있습니다. 수퍼유저만 프로시저의 소유자를 변경할 수 있습니다.

구문

```
ALTER PROCEDURE sp_name [ ( [ [ argname ] [ argmode ] argtype [, ...] ] ) ]
    RENAME TO new_name
```

```
ALTER PROCEDURE sp_name [ ( [ [ argname ] [ argmode ] argtype [, ...] ] ) ]
    OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
```

파라미터

sp_name

변경할 프로시저의 이름입니다. 특정 스키마를 사용하려면 현재 검색 경로에 프로시저의 이름만 지정하거나, schema_name.sp_procedure_name 형식을 사용하십시오.

[argname] [argmode] argtype

인수 이름, 인수 모드 및 데이터 형식의 목록입니다. 입력 데이터 형식만 필수입니다. 입력 데이터 형식은 저장 프로시저를 식별하는 데 사용됩니다. 모드와 함께 입력 및 출력 파라미터를 포함하여 프로시저를 생성하는 데 사용된 전체 서명을 제공할 수도 있습니다.

new_name

저장 프로시저의 새 이름입니다.

new_owner | CURRENT_USER | SESSION_USER

저장 프로시저의 새 소유자입니다.

예시

다음 예제에서는 프로시저의 이름을 first_quarter_revenue에서 quarterly_revenue로 변경합니다.

```
ALTER PROCEDURE first_quarter_revenue(volume INOUT bigint, at_price IN numeric,
 result OUT int) RENAME TO quarterly_revenue;
```

이 예는 다음과 동일합니다.

```
ALTER PROCEDURE first_quarter_revenue(bigint, numeric) RENAME TO quarterly_revenue;
```

다음 예에서는 프로시저의 소유자를 etl_user로 변경합니다.

```
ALTER PROCEDURE quarterly_revenue(bigint, numeric) OWNER TO etl_user;
```

ALTER SCHEMA

기존 스키마의 정의를 변경합니다. 스키마의 소유자를 변경하거나 스키마의 이름을 바꾸려면 이 명령을 사용하십시오. 예를 들어, 기존 스키마의 새 버전을 만들 계획인 경우 그 스키마의 백업 복사본을 보존하기 위해 스키마의 이름을 바꿉니다. 스키마에 대한 자세한 내용은 CREATE SCHEMA 섹션을 참조하세요.

구성된 스키마 할당량을 보려면 SVV_SCHEMA_QUOTA_STATE 섹션을 참조하세요.

스키마 할당량이 초과된 레코드를 보려면 STL_SCHEMA_QUOTA_VIOLATIONS 섹션을 참조하세요.

필수 권한

ALTER SCHEMA에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- ALTER SCHEMA 권한이 있는 사용자
- 스키마 소유자

스키마 이름을 변경할 때는 저장 프로시저나 구체화된 뷰와 같이 이전 이름을 사용하는 객체를 새 이름을 사용하도록 업데이트해야 한다는 점에 유의하세요.

구문

```
ALTER SCHEMA schema_name
{
RENAME TO new_name |
OWNER TO new_owner |
QUOTA { quota [MB | GB | TB] | UNLIMITED }
}
```

파라미터

schema_name

변경할 데이터베이스 스키마의 이름입니다.

RENAME TO

스키마의 이름을 바꾸는 절입니다.

new_name

스키마의 새 이름입니다. 유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요.

OWNER TO

스키마의 소유자를 변경하는 절입니다.

new_owner

스키마의 새 소유자입니다.

QUOTA

지정된 스키마에서 사용할 수 있는 최대 디스크 공간 양입니다. 이 공간은 지정된 스키마 아래에 있는 모든 테이블의 총체적 크기입니다. Amazon Redshift는 선택한 값을 메가바이트로 변환합니다. 값을 지정하지 않으면 기가바이트가 기본 측정 단위입니다.

스키마 할당량 구성에 대한 자세한 내용은 CREATE SCHEMA 섹션을 참조하세요.

예시

다음 예에서는 SALES 스키마의 이름을 US_SALES로 바꿉니다.

```
alter schema sales
rename to us_sales;
```

다음 예에서는 US_SALES 스키마의 소유권을 사용자 DWUSER에게 제공합니다.

```
alter schema us_sales
owner to dwuser;
```

다음 예에서는 할당량을 300GB로 변경하고 할당량을 제거합니다.

```
alter schema us_sales QUOTA 300 GB;
alter schema us_sales QUOTA UNLIMITED;
```

ALTER SYSTEM

Amazon Redshift 클러스터 또는 Redshift Serverless 작업 그룹에 대한 시스템 수준 구성 옵션을 변경합니다.

필수 권한

다음 사용자 유형 중 하나가 ALTER SYSTEM 명령을 실행할 수 있습니다.

- 수퍼유저
- 관리자

구문

```
ALTER SYSTEM SET system-level-configuration = {true| t | on | false | f | off}
```

파라미터

시스템 수준 구성

시스템 수준 구성입니다. 유효한 값: `data_catalog_auto_mount` 및 `metadata_security`.

{true| t | on | false | f | off}

시스템 수준 구성을 활성화 또는 비활성화할 값입니다. `true`,`t` 또는`on`은 구성을 활성화함을 나타냅니다. `false`,`f` 또는`off`는 구성을 비활성화함을 나타냅니다.

사용 노트

프로비저닝된 클러스터의 경우 `data_catalog_auto_mount`의 변경 사항은 클러스터를 다음에 재부팅할 때 적용됩니다. 자세한 내용은 Amazon Redshift 관리 안내서의 클러스터 재부팅 섹션을 참조하세요.

서버가 없는 작업 그룹의 경우 `data_catalog_auto_mount` 변경 사항이 즉시 적용되지 않습니다.

예시

다음 예제에서는 AWS Glue Data Catalog 자동 마운트를 활성화합니다.

```
ALTER SYSTEM SET data_catalog_auto_mount = true;
```

다음 예시에서는 메타데이터 보안을 활성화합니다.

```
ALTER SYSTEM SET metadata_security = true;
```

기본 ID 네임스페이스 설정

이 예는 ID 제공업체 작업과 관련된 것입니다. Redshift를 IAM Identity Center 및 ID 제공업체와 통합하여 Redshift 및 기타 AWS 서비스에 대한 ID 관리를 중앙 집중화할 수 있습니다.

다음 샘플은 시스템의 기본 ID 네임스페이스를 설정하는 방법을 보여줍니다. 이렇게 하면 각 ID의 접두사로 네임스페이스를 포함할 필요가 없기 때문에 이후에 GRANT 및 CREATE 문을 더 간단하게 실행할 수 있습니다.

```
ALTER SYSTEM SET default_identity_namespace = 'MYCO';
```

이 명령을 실행한 후 다음과 같은 명령문을 실행할 수 있습니다.

```
GRANT SELECT ON TABLE mytable TO alice;

GRANT UPDATE ON TABLE mytable TO salesrole;

CREATE USER bob password 'md50c983d1a624280812631c5389e60d48c';
```

기본 ID 네임스페이스를 설정하면 각 ID에 접두사로 네임스페이스가 필요 없게 되는 효과가 있습니다. 이 예제에서는 `alice`가 `MYCO:alice`로 대체됩니다. 이는 ID가 포함된 모든 경우에 발생합니다. Redshift에서 ID 제공업체를 사용하는 방법에 대한 자세한 내용은 Redshift를 IAM Identity Center와 연결하여 사용자에게 Single Sign-On 경험을 제공합니다를 참조하세요.

IAM Identity Center를 사용한 Redshift 구성과 관련된 설정에 대한 자세한 내용은 SET 및 ALTER IDENTITY PROVIDER 섹션을 참조하세요.

ALTER TABLE

이 명령은 Amazon Redshift 테이블 또는 Amazon Redshift Spectrum 외부 테이블의 정의를 변경합니다. 이 명령은 CREATE TABLE 또는 CREATE EXTERNAL TABLE에서 설정한 값과 속성을 업데이트합니다.

트랜잭션 블록(BEGIN ... END) 내의 외부 테이블에서 ALTER TABLE을 실행할 수 없습니다. 버전 관리에 대한 자세한 내용은 직렬화 가능 격리 단원을 참조하십시오.

ALTER TABLE은 ALTER TABLE 작업 관련한 트랜잭션이 완료될 때까지 테이블 읽기 및 쓰기 을 잠금 설정합니다. 다만 변경하는 동안 테이블에서 데이터를 쿼리하거나 다른 작업을 수행할 수 있다고 설명서에 나와 있는 경우는 예외입니다.

필수 권한

명령이 성공하려면 테이블을 변경하는 사용자에게 적절한 권한이 있어야 합니다. ALTER TABLE 명령에 따라 다음 권한 중 하나가 필요합니다.

- 수퍼유저
- ALTER TABLE 권한이 있는 사용자
- 스키마에 대한 USAGE 권한이 있는 테이블 소유자

구문

```
ALTER TABLE table_name
{
ADD table_constraint
| DROP CONSTRAINT constraint_name [ RESTRICT | CASCADE ]
| OWNER TO new_owner
| RENAME TO new_name
| RENAME COLUMN column_name TO new_name
| ALTER COLUMN column_name TYPE updated_varchar_data_type_size
| ALTER COLUMN column_name ENCODE new_encode_type
| ALTER COLUMN column_name ENCODE encode_type,
| ALTER COLUMN column_name ENCODE encode_type, .....;
| ALTER DISTKEY column_name
| ALTER DISTSTYLE ALL
| ALTER DISTSTYLE EVEN
| ALTER DISTSTYLE KEY DISTKEY column_name
| ALTER DISTSTYLE AUTO
| ALTER [COMPOUND] SORTKEY ( column_name [,...] )
| ALTER SORTKEY AUTO
| ALTER SORTKEY NONE
| ALTER ENCODE AUTO
| ADD [ COLUMN ] column_name column_type
  [ DEFAULT default_expr ]
  [ ENCODE encoding ]
  [ NOT NULL | NULL ]
  [ COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE } ] |
| DROP [ COLUMN ] column_name [ RESTRICT | CASCADE ]
| ROW LEVEL SECURITY { ON | OFF } [ CONJUNCTION TYPE { AND | OR } ] [ FOR DATASHARES ]}

where table_constraint is:
```

```
[ CONSTRAINT constraint_name ]
{ UNIQUE ( column_name [, ... ] )
| PRIMARY KEY ( column_name [, ... ] )
| FOREIGN KEY (column_name [, ... ] )
   REFERENCES  reftable [ ( refcolumn ) ]}

The following options apply only to external tables:

SET LOCATION { 's3://bucket/folder/' | 's3://bucket/manifest_file' }
| SET FILE FORMAT format |
| SET TABLE PROPERTIES ('property_name'='property_value')
| PARTITION ( partition_column=partition_value [, ...] )
  SET LOCATION { 's3://bucket/folder' |'s3://bucket/manifest_file' }
| ADD [IF NOT EXISTS]
    PARTITION ( partition_column=partition_value [, ...] ) LOCATION
 { 's3://bucket/folder' |'s3://bucket/manifest_file' }
    [, ... ]
| DROP PARTITION ( partition_column=partition_value [, ...] )
```

ALTER TABLE 명령을 실행하는 시간을 줄이려면 ALTER TABLE 명령의 일부 절을 결합할 수 있습니다.

Amazon Redshift는 ALTER TABLE 절의 다음 조합을 지원합니다.

```
ALTER TABLE tablename ALTER SORTKEY (column_list), ALTER DISTKEY column_Id;
ALTER TABLE tablename ALTER DISTKEY column_Id, ALTER SORTKEY (column_list);
ALTER TABLE tablename ALTER SORTKEY (column_list), ALTER DISTSTYLE ALL;
ALTER TABLE tablename ALTER DISTSTYLE ALL, ALTER SORTKEY (column_list);
```

파라미터

table_name

수정할 테이블 이름. 특정 스키마를 사용하려면 테이블의 이름만 지정하거나 schema_name.table_name 형식을 사용하십시오. 외부 테이블은 외부 스키마 이름으로 정규화해야 합니다. ALTER TABLE 문을 사용하여 뷰의 이름을 바꾸거나 그 소유자를 변경하려는 경우 뷰 이름을 지정할 수도 있습니다. 테이블 이름의 최대 길이는 127바이트이며, 이보다 긴 이름은 127바이트까지 표시되고 나머지는 잘립니다. 최대 4바이트까지 UTF-8 멀티바이트 문자를 사용할 수 있습니다. 유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요.

ADD table_constraint

테이블에 지정된 제약 조건을 추가하는 절입니다. 유효한 table_constraint 값에 대한 설명은 CREATE TABLE 섹션을 참조하세요.

> **Note**
>
> Null 허용 열에 기본 키 제약 조건을 추가할 수 없습니다. 이 열이 원래 NOT NULL 제약 조건으로 생성된 경우에는 기본 키 제약 조건을 추가할 수 있습니다.

DROP CONSTRAINT constraint_name

테이블에서 명명된 제약 조건을 삭제하는 절입니다. 제약 조건을 삭제하려면 제약 조건 유형이 아니라 제약 조건 이름을 지정하십시오. 테이블 제약 조건 이름을 보려면 다음 쿼리를 실행합니다.

```
select constraint_name, constraint_type
from information_schema.table_constraints;
```

RESTRICT

지정된 제약 조건만 제거하는 절입니다. RESTRICT는 DROP CONSTRAINT에 대한 옵션입니다. RESTRICT는 CASCADE와 함께 사용할 수 없습니다.

CASCADE

지정된 제약 조건과 그 제약 조건에 종속된 모든 것을 제거하는 절입니다. CASCADE는 DROP CONSTRAINT에 대한 옵션입니다. CASCADE는 RESTRICT와 함께 사용할 수 없습니다.

OWNER TO new_owner

테이블(또는 뷰)의 소유자를 new_owner 값으로 변경하는 절입니다.

RENAME TO new_name

테이블(또는 뷰)의 이름을 new_name에 지정된 값으로 바꾸는 절입니다. 최대 테이블 이름 길이는 127바이트이며, 이보다 긴 이름은 127바이트까지 표시되고 나머지는 잘립니다.

영구 테이블 이름을 '#'로 시작하는 이름으로 바꿀 수 없습니다. '#'로 시작하는 테이블 이름은 임시 테이블을 나타냅니다.

외부 테이블의 이름을 변경할 수 없습니다.

ALTER COLUMN column_name TYPE updated_varchar_data_type_size

VARCHAR 데이터 형식으로 정의된 열 크기를 변경하는 절입니다. 이 절은 VARCHAR 데이터 유형의 크기 변경만 지원합니다. 다음 제한을 고려하십시오.

- 압축 인코딩 BYTEDICT, RUNLENGTH, TEXT255 또는 TEXT32K를 사용하는 열은 변경할 수 없습니다.
- 기존 데이터의 최대 크기보다 작게 크기를 줄일 수 없습니다.
- 기본값이 있는 열은 변경할 수 없습니다.
- UNIQUE, PRIMARY KEY 또는 FOREIGN KEY가 있는 열은 변경할 수 없습니다.
- 트랜잭션 블록(BEGIN ... END) 내에서 열을 변경할 수 없습니다. 버전 관리에 대한 자세한 내용은 직렬화 가능 격리 단원을 참조하십시오.

ALTER COLUMN column_name ENCODE new_encode_type

열의 압축 인코딩을 변경하는 절입니다. 열에 압축 인코딩을 지정하면 테이블이 더 이상 ENCODE AUTO로 설정되지 않습니다. 압축 인코딩에 대한 자세한 내용은 열 압축으로 저장된 데이터 크기 축소 섹션을 참조하세요.

열에 압축 인코딩을 변경하면 테이블을 쿼리할 수 있습니다.

다음 제한을 고려하십시오.

- 열에 대해 현재 정의된 것과 동일한 인코딩으로 열을 변경할 수 없습니다.
- 인터리브 정렬 키가 있는 테이블의 열에 대한 인코딩을 변경할 수 없습니다.

ALTER COLUMN column_name ENCODE encode_type, ALTER COLUMN column_name ENCODE encode_type, ...;

단일 명령에서 여러 열의 압축 인코딩을 변경하는 절입니다. 압축 인코딩에 대한 자세한 내용은 열 압축으로 저장된 데이터 크기 축소 섹션을 참조하세요.

열에 압축 인코딩을 변경하면 테이블을 쿼리할 수 있습니다.

다음 제한을 고려하십시오.

- 단일 명령에서 열을 동일하거나 다른 인코딩 형식으로 여러 번 변경할 수 없습니다.
- 열에 대해 현재 정의된 것과 동일한 인코딩으로 열을 변경할 수 없습니다.
- 인터리브 정렬 키가 있는 테이블의 열에 대한 인코딩을 변경할 수 없습니다.

ALTER DISTSTYLE ALL

테이블의 기존 분산 스타일을 ALL으로 변경하는 절입니다. 다음을 고려하세요.

- ALTER DISTSTYLE, ALTER SORTKEY, VACUUM은 동일한 테이블에서 동시에 실행할 수 없습니다.
  - VACUUM이 현재 실행 중인 경우 ALTER DISTSTYLE ALL을 실행하면 오류가 반환됩니다.
  - ALTER DISTSTYLE ALL이 실행 중인 경우 백그라운드 VACCUM이 테이블에서 시작되지 않습니다.
- 인터리브 정렬 키 및 임시 테이블이 있는 테이블에는 ALTER DISTYLE ALL 명령이 지원되지 않습니다.
- 배포 스타일이 이전에 AUTO로 정의된 경우 테이블은 더 이상 자동 테이블 최적화의 후보가 아닙니다.

DISTSTYLE ALL에 대한 자세한 내용은 CREATE TABLE 섹션을 참조하세요.

ALTER DISTSTYLE EVEN

테이블의 기존 분산 스타일을 EVEN으로 변경하는 절입니다. 다음을 고려하세요.

- ALTER DISTSYTLE, ALTER SORTKEY, VACUUM은 동일한 테이블에서 동시에 실행할 수 없습니다.
  - VACUUM이 현재 실행 중인 경우 ALTER DISTSTYLE EVEN을 실행하면 오류가 반환됩니다.
  - ALTER DISTSTYLE EVEN이 실행 중인 경우 백그라운드 VACCUM이 테이블에서 시작되지 않습니다.
- 인터리브 정렬 키 및 임시 테이블이 있는 테이블에는 ALTER DISTYLE EVEN 명령이 지원되지 않습니다.
- 배포 스타일이 이전에 AUTO로 정의된 경우 테이블은 더 이상 자동 테이블 최적화의 후보가 아닙니다.

DISTSTYLE EVEN에 대한 자세한 내용은 CREATE TABLE 섹션을 참조하세요.

ALTER DISTKEY column_name 또는 ALTER DISTSTYLE KEY DISTKEY column_name

테이블의 배포 키로 사용되는 열을 변경하는 절입니다. 다음을 고려하세요.

- VACUUM 및 ALTER DISTKEY는 동일한 테이블 상에서 동시에 실행할 수 없습니다.
  - VACUUM이 이미 실행 중인 경우 ALTER DISTKEY가 오류를 반환합니다.
  - ALTER DISTKEY가 실행 중인 경우 백그라운드 VACCUM이 테이블에서 시작되지 않습니다.
  - ALTER DISTKEY가 실행 중인 경우 포그라운드 VACCUM이 오류를 반환합니다.
- 테이블에서 한 번에 하나의 ALTER DISTKEY 명령만 실행할 수 있습니다.
- 인터리브 정렬 키가 있는 테이블에는 ALTER DISTKEY 명령이 지원되지 않습니다.

- 배포 스타일이 이전에 AUTO로 정의된 경우 테이블은 더 이상 자동 테이블 최적화의 후보가 아닙니다.

DISTSTYLE KEY를 지정할 때 데이터는 DISTKEY 열에 있는 값을 기준으로 배포됩니다. DISTSTYLE에 대한 자세한 내용은 CREATE TABLE 섹션을 참조하세요.

ALTER DISTSTYLE AUTO

테이블의 기존 배포 스타일을 AUTO로 변경하는 절입니다.

배포 스타일을 AUTO로 변경하면 테이블의 배포 스타일이 다음과 같이 설정됩니다.

- DISTSTYLE ALL이 있는 작은 테이블은 AUTO(ALL)로 변환됩니다.
- DISTSTYLE EVEN이 있는 작은 테이블은 AUTO(ALL)로 변환됩니다.
- DISTSTYLE KEY가 있는 작은 테이블은 AUTO(ALL)로 변환됩니다.
- DISTSTYLE ALL이 있는 큰 테이블은 AUTO(EVEN)로 변환됩니다.
- DISTSTYLE EVEN이 있는 큰 테이블은 AUTO(EVEN)로 변환됩니다.
- DISTSTYLE KEY가 있는 큰 테이블은 AUTO(KEY)로 변환되고 DISTKEY는 보존됩니다. 이 경우 Amazon Redshift는 테이블을 변경하지 않습니다.

새로운 배포 스타일 또는 키가 쿼리 성능을 향상시킬 것이라고 판단되면 Amazon Redshift는 향후 테이블의 배포 스타일이나 키를 변경할 수 있습니다. 예를 들어 Amazon Redshift는 DISTSTYLE이 AUTO(KEY)인 테이블을 AUTO(EVEN)로 또는 그 반대로 변환할 수 있습니다. 데이터 재배포 및 잠금을 포함하여 배포 키가 변경될 때의 동작에 대한 자세한 내용은 Amazon Redshift Advisor 권장 사항을 참조하세요.

DISTSTYLE AUTO에 대한 자세한 내용은 CREATE TABLE 섹션을 참조하세요.

테이블의 배포 스타일을 보려면 SVV_TABLE_INFO 시스템 카탈로그 뷰를 쿼리합니다. 자세한 내용은 SVV_TABLE_INFO 단원을 참조하십시오. 테이블에 대한 Amazon Redshift Advisor 권장 사항을 보려면 SVV_ALTER_TABLE_RECOMMENDATIONS 시스템 카탈로그 뷰를 쿼리합니다. 자세한 내용은 SVV_ALTER_TABLE_RECOMMENDATIONS 단원을 참조하십시오. Amazon Redshift에서 수행한 작업을 보려면 SVL_AUTO_WORKER_ACTION 시스템 카탈로그 뷰를 쿼리합니다. 자세한 내용은 SVL_AUTO_WORKER_ACTION 단원을 참조하십시오.

ALTER [COMPOUND] SORTKEY (column_name [,...])

테이블에 사용되는 정렬 키를 변경하거나 추가하는 절입니다. 임시 테이블에는 ALTER SORTKEY가 지원되지 않습니다.

정렬 키를 변경하면 새 정렬 키 또는 원래 정렬 키에 있는 열의 압축 인코딩이 변경될 수 있습니다. 테이블에 대해 명시적으로 정의된 인코딩이 없는 경우 Amazon Redshift는 다음과 같이 압축 인코딩을 자동으로 할당합니다.

- 정렬 키로 정의된 열은 RAW 압축이 할당됩니다.
- BOOLEAN, REAL 또는 DOUBLE PRECISION 데이터 형식으로 정의된 열은 RAW 압축이 할당됩니다.
- SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIME, TIMETZ, TIMESTAMP 또는 TIMESTAMPTZ로 정의된 열에는 AZ64 압축이 할당됩니다.
- CHAR 또는 VARCHAR로 정의된 열에는 LZO 압축이 할당됩니다.

다음을 고려하세요.

- 테이블당 정렬 키에 최대 400개의 열을 정의할 수 있습니다.
- 인터리브 정렬 키를 복합 정렬 키 또는 정렬 키 없음으로 변경할 수 있습니다. 그러나 복합 정렬 키를 인터리브 정렬 키로 변경할 수는 없습니다.
- 정렬 키가 이전에 AUTO로 정의된 경우 테이블은 더 이상 자동 테이블 최적화의 후보가 아닙니다.
- Amazon Redshift는 정렬 키로 정의된 열에 대해 RAW 인코딩(압축 없음)을 사용하는 것이 좋습니다. 열을 변경하여 정렬 키로 선택하면 열의 압축이 RAW 압축(압축 없음)으로 변경됩니다. 이렇게 하면 테이블에 필요한 스토리지 양이 늘어날 수 있습니다. 테이블 크기 증가량은 특정 테이블 정의와 테이블 내용에 따라 다릅니다. 압축에 대한 자세한 내용은 압축 인코딩 섹션을 참조하세요.

데이터가 테이블에 로드되면 데이터는 정렬 키의 순서로 로드됩니다. 정렬 키를 변경하면 Amazon Redshift는 데이터를 재정렬합니다. SORTKEY에 대한 자세한 내용은 CREATE TABLE 섹션을 참조하세요.

ALTER SORTKEY AUTO

대상 테이블의 정렬 키를 AUTO로 변경하거나 추가하는 절입니다. 임시 테이블에는 ALTER SORTKEY AUTO가 지원되지 않습니다.

정렬 키를 AUTO로 변경하면 Amazon Redshift는 테이블의 기존 정렬 키를 보존합니다.

새로운 정렬 키가 쿼리 성능을 향상시킬 것이라고 판단되면 Amazon Redshift는 향후 테이블의 정렬 키를 변경할 수 있습니다.

SORTKEY AUTO에 대한 자세한 내용은 CREATE TABLE 섹션을 참조하세요.

테이블의 정렬 키를 보려면 SVV_TABLE_INFO 시스템 카탈로그 뷰를 쿼리합니다. 자세한 내용은 SVV_TABLE_INFO 단원을 참조하십시오. 테이블에 대한 Amazon Redshift Advisor 권장 사항을 보려면 SVV_ALTER_TABLE_RECOMMENDATIONS 시스템 카탈로그 뷰를 쿼리합니다. 자세한 내용은 SVV_ALTER_TABLE_RECOMMENDATIONS 단원을 참조하십시오. Amazon Redshift에서 수행한 작업을 보려면 SVL_AUTO_WORKER_ACTION 시스템 카탈로그 뷰를 쿼리합니다. 자세한 내용은 SVL_AUTO_WORKER_ACTION 단원을 참조하십시오.

ALTER SORTKEY NONE

대상 테이블의 정렬 키를 제거하는 절입니다.

정렬 키가 이전에 AUTO로 정의된 경우 테이블은 더 이상 자동 테이블 최적화의 후보가 아닙니다.

ALTER ENCODE AUTO

대상 테이블 열의 인코딩 형식을 AUTO로 변경하는 절입니다. 인코딩을 AUTO로 변경하면 Amazon Redshift는 테이블에 있는 열의 기존 인코딩 형식이 보존됩니다. 그런 다음 새 인코딩 형식이 쿼리 성능을 향상시킬 수 있다고 판단되면 Amazon Redshift가 테이블 열의 인코딩 형식을 변경할 수 있습니다.

인코딩을 지정하기 위해 하나 이상의 열을 변경하는 경우 Amazon Redshift는 더 이상 테이블의 모든 열에 대한 인코딩을 자동으로 조정하지 않습니다. 열은 현재 인코딩 설정을 유지합니다.

다음 작업은 테이블의 ENCODE AUTO 설정에 영향을 주지 않습니다.

- 테이블 이름 바꾸기.
- 테이블에 대한 DISTSTYLE 또는 SORTKEY 설정 변경.
- ENCODE 설정으로 열 추가 또는 삭제.
- COPY 명령의 COMPUPDATE 옵션 사용. 자세한 내용은 데이터 로드 작업 단원을 참조하십시오.

테이블의 인코딩을 보려면 SVV_TABLE_INFO 시스템 카탈로그 뷰를 쿼리합니다. 자세한 내용은 SVV_TABLE_INFO 단원을 참조하십시오.

RENAME COLUMN column_name TO new_name

열의 이름을 new_name에 지정된 값으로 바꾸는 절입니다. 최대 열 이름 길이는 127바이트이며, 이보다 긴 이름은 127바이트까지 표시되고 나머지는 잘립니다. 유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요.

ADD [COLUMN] column_name

테이블에 지정된 이름과 함께 열을 추가하는 절입니다. 각각의 ALTER TABLE 문에 한 개의 열만 추가할 수 있습니다.

테이블의 배포 키(DISTKEY) 또는 정렬 키(SORTKEY)인 열은 추가할 수 없습니다.

ALTER TABLE ADD COLUMN 명령을 사용하여 다음 테이블 및 열 속성을 수정할 수 없습니다.

- UNIQUE
- PRIMARY KEY
- REFERENCES(외래 키)
- IDENTITY 또는 GENERATED BY DEFAULT AS IDENTITY

최대 열 이름 길이는 127바이트이며, 이보다 긴 이름은 127바이트까지 표시되고 나머지는 잘립니다. 단일 테이블에서 정의할 수 있는 최대 열 수는 1,600개입니다.

외부 테이블에 열을 추가할 때는 다음과 같은 제한 사항이 적용됩니다.

- 열 제약 조건 DEFAULT, ENCODE, NOT NULL, NULL이 있는 외부 테이블에 열을 추가할 수 없습니다.
- AVRO 파일 형식으로 정의된 외부 테이블에 열을 추가할 수 없습니다.
- 가상 열이 활성화되어 있으면 외부 테이블 하나에서 정의할 수 있는 최대 열 개수는 1,598개입니다. 가상 열이 활성화되어 있지 않으면 테이블 하나에서 정의할 수 있는 최대 열 개수는 1,600개입니다.

자세한 내용은 CREATE EXTERNAL TABLE 단원을 참조하십시오.

column_type

추가되는 열의 데이터 형식입니다. CHAR 및 VARCHAR 열의 경우 최대 길이를 선언하는 대신 MAX 키워드를 사용할 수 있습니다. MAX는 최대 길이를 CHAR의 경우 4,096바이트, VARCHAR의 경우 65,535바이트로 설정합니다. GEOMETRY 객체의 최대 크기는 1,048,447바이트입니다.

Amazon Redshift에서 지원하는 데이터 형식에 대한 자세한 내용은 데이터 타입 섹션을 참조하세요.

DEFAULT default_expr

열의 기본 데이터 값을 할당하는 절입니다. default_expr의 데이터 형식은 열의 데이터 형식과 일치해야 합니다. DEFAULT 값은 변수가 없는 표현식이어야 합니다. 하위 쿼리, 현재 테이블에 있는 다른 열과의 상호 참조, 사용자 정의 함수는 허용되지 않습니다.

default_expr은 열의 값을 지정하지 않는 INSERT 작업에 사용됩니다. 기본값이 지정되지 않은 경우 열의 기본값은 Null입니다.

COPY 작업에서 DEFAULT 값과 NOT NULL 제약 조건을 가진 열에 null 필드가 나타나는 경우 COPY 명령은 default_expr의 값을 삽입합니다.

외부 테이블에는 DEFAULT가 지원되지 않습니다.

ENCODE encoding

열에 대한 압축 인코딩입니다. 기본적으로 Amazon Redshift는 테이블의 열에 대해 압축 인코딩을 지정하지 않거나 테이블에 대해 ENCODE AUTO 옵션을 지정하는 경우 테이블의 모든 열에 대한 압축 인코딩을 자동으로 관리합니다.

테이블의 열에 대해 압축 인코딩을 지정하거나 테이블에 대해 ENCODE AUTO 옵션을 지정하지 않으면 Amazon Redshift는 다음과 같이 압축 인코딩을 지정하지 않은 열에 압축 인코딩을 자동으로 할당합니다.

- 임시 테이블의 모든 열은 기본적으로 RAW 압축으로 할당됩니다.
- 정렬 키로 정의된 열은 RAW 압축이 할당됩니다.
- BOOLEAN, REAL, DOUBLE PRECISION, GEOMETRY 또는 GEOGRAPHY 데이터 유형으로 정의된 열은 RAW 압축이 할당됩니다.
- SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIME, TIMETZ, TIMESTAMP 또는 TIMESTAMPTZ로 정의된 열에는 AZ64 압축이 할당됩니다.
- CHAR, VARCHAR 또는 VARBYTE로 정의된 열에는 LZO 압축이 할당됩니다.

> **Note**
>
> 열을 압축하지 않으려면 RAW 인코딩을 명시적으로 지정하십시오.

다음 모듈을 지원합니다.compression encodings (p. 59)

- AZ64
- BYTEDICT
- 델타
- 델타
- LZO
- LZO
- LZO
- LZO
- RAW(압축 없음)
- RUNLENGTH

- TEXT255
- TEXT32K
- ZSTD

외부 테이블에는 ENCODE가 지원되지 않습니다.

NOT NULL | NULL

NOT NULL은 열이 null 값을 포함하도록 허용되지 않도록 지정합니다. 기본값인 NULL은 열이 null 값을 허용하도록 지정합니다.

외부 테이블에는 NOT NULL 및 NULL가 지원되지 않습니다.

COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE }

열의 문자열 검색 또는 비교가 CASE_SENSITIVE인지, CASE_INSENSITIVE인지를 지정하는 절입니다. 기본값은 데이터베이스의 현재 대/소문자 구분 구성과 동일합니다.

데이터베이스 데이터 정렬 정보를 찾으려면 다음 명령을 사용합니다.

```
SELECT db_collation();

db_collation
----------------
 case_sensitive
(1 row)
```

DROP [COLUMN] column_name

테이블에서 삭제할 열의 이름입니다.

테이블의 마지막 열은 삭제할 수 없습니다. 테이블에는 열이 하나 이상 있어야 합니다.

테이블의 배포 키(DISTKEY) 또는 정렬 키(SORTKEY)인 열은 삭제할 수 없습니다. 열에 뷰, 기본 키, 외래 키 또는 UNIQUE 제한과 같이 종속적 객체가 있는 경우 DROP COLUMN의 기본 동작은 RESTRICT입니다.

외부 테이블에서 열을 삭제할 경우 다음과 같은 제한 사항이 적용됩니다.

- 외부 테이블에서 파티션 열로 사용되는 열은 삭제할 수 없습니다.
- AVRO 파일 형식으로 정의된 외부 테이블에서는 열을 삭제할 수 없습니다.
- 외부 테이블에서는 RESTRICT 및 CASCADE를 무시합니다.

- 정책을 삭제하거나 분리하지 않으면 정책 정의 내에서 참조되는 정책 테이블의 열을 삭제할 수 없습니다. 이는 CASCADE 옵션이 지정된 경우에도 적용됩니다. 정책 테이블의 다른 열은 삭제할 수 있습니다.

자세한 내용은 CREATE EXTERNAL TABLE 단원을 참조하십시오.

RESTRICT

DROP COLUMN과 함께 사용하는 경우 RESTRICT는 다음과 같은 경우 삭제하려는 열이 삭제되지 않음을 나타냅니다.

- 정의된 보기가 삭제할 열을 참조하는 경우
- 외래 키가 삭제할 열을 참조하는 경우
- 삭제할 열이 멀티파트 키에 포함된 경우

RESTRICT는 CASCADE와 함께 사용할 수 없습니다.

외부 테이블에서는 RESTRICT 및 CASCADE를 무시합니다.

CASCADE

DROP COLUMN과 함께 사용될 때, 지정된 열과 그 열에 종속된 모든 것을 제거합니다. CASCADE는 RESTRICT와 함께 사용할 수 없습니다.

외부 테이블에서는 RESTRICT 및 CASCADE를 무시합니다.

다음 옵션은 외부 테이블에만 적용됩니다.

SET LOCATION { 's3://bucket/folder/' | 's3://bucket/manifest_file' }

데이터 파일이 포함된 Amazon S3 폴더 또는 Amazon S3 객체 경로 목록이 포함된 매니페스트 파일의 경로. 버킷은 Amazon Redshift 클러스터와 동일한 AWS 리전에 있어야 합니다. 지원되는 AWS 리전 목록은 Amazon Redshift Spectrum 제한 사항 섹션을 참조하세요. 매니페스트 파일 사용에 대한 자세한 내용은 CREATE EXTERNAL TABLE 파라미터 참조의 LOCATION을 참조하십시오.

SET FILE FORMAT format

외부 데이터 파일의 파일 형식입니다.

유효한 형식은 다음과 같습니다.

- AVRO
- PARQUET

- RCFILE
- SEQUENCEFILE
- TEXTFILE

SET TABLE PROPERTIES ('property_name'='property_value')

외부 테이블에 대한 테이블 속성의 테이블 정의를 설정하는 절입니다.

> **Note**
>
> 테이블 속성은 대/소문자를 구분합니다.

'numRows'='row_count'

테이블 정의를 위해 numRows 값을 설정하는 속성입니다. 외부 테이블의 통계를 명시적으로 업데이트하려면 테이블의 크기를 나타내도록 numRows 속성을 설정합니다. Amazon Redshift는 쿼리 옵티마이저가 쿼리 계획을 생성하는 데 사용하는 테이블 통계를 생성하기 위해 외부 테이블을 분석하지 않습니다. 테이블 통계가 외부 테이블에 대해 설정되지 않은 경우 Amazon Redshift는 쿼리 실행 계획을 실행합니다. 이 계획은 외부 테이블이 더 큰 테이블이고 로컬 테이블이 더 작은 테이블이라는 가정에 기초하여 생성됩니다.

'skip.header.line.count'='line_count'

각 원본 파일의 시작 부분에서 건너 뛸 행 개수를 설정하는 속성입니다.

PARTITION (partition_column=partition_value [, ...] SET LOCATION { 's3://bucket/folder' | 's3://bucket/manifest_file' }

하나 이상의 파티션 열의 위치를 새로 설정하는 절입니다.

ADD [IF NOT EXISTS] PARTITION (partition_column=partition_value [, ...]) LOCATION { 's3://bucket/folder' | 's3://bucket/manifest_file' } [, ...]

파티션을 하나 이상 추가하는 절입니다. 단일 ALTER TABLE … ADD 문을 사용하여 여러 PARTITION 절을 지정할 수 있습니다.

> **Note**
>
> AWS Glue 카탈로그를 사용하는 경우 단일 ALTER TABLE 문을 사용하여 파티션을 최대 100개까지 추가할 수 있습니다.

IF NOT EXISTS 절은 지정한 파티션이 이미 있으면 명령이 아무 것도 변경하지 않아야 함을 나타냅니다. 또한 명령 실행 시 오류 메시지와 함께 종료되지 않고 파티션이 존재한다는 메시지를 반환해야 함을 나타냅니다. 이 절은 스크립트 작성 시 유용하므로, ALTER TABLE이 이미 존재하는 파티션의 추가를 시도하는 경우 스크립트가 실패하지 않습니다.

DROP PARTITION (partition_column=partition_value [, ...])

지정된 파티션을 삭제하는 절입니다. 파티션을 삭제하는 외부 테이블 메타데이터만 변경됩니다. Amazon S3의 데이터는 영향을 받지 않습니다.

ROW LEVEL SECURITY { ON | OFF } [CONJUNCTION TYPE { AND | OR }] [FOR DATASHARES]

관계에 대한 행 수준 보안을 켜거나 끄는 절입니다.

관계에 대해 행 수준 보안이 켜 있으면 행 수준 보안 정책에서 액세스를 허용하는 행만 읽을 수 있습니다. 관계에 대한 액세스 권한을 부여하는 정책이 없으면 관계에서 행을 볼 수 없습니다. 수퍼유저 및 `sys:secadmin` 역할을 가진 사용자 또는 역할만 ROW LEVEL SECURITY 절을 설정할 수 있습니다. 자세한 내용은 행 수준 보안 단원을 참조하십시오.

- [CONJUNCTION TYPE { AND | OR }]

  관계에 대해 행 수준 보안 정책의 접속사 유형을 선택할 수 있도록 하는 절입니다. 여러 행 수준 보안 정책이 관계에 연결된 경우 정책을 AND 또는 OR 절과 결합할 수 있습니다. 기본적으로 Amazon Redshift는 RLS 정책을 AND 절과 결합합니다. `sys:secadmin` 역할이 있는 슈퍼 사용자, 사용자 또는 역할은 이 절을 사용하여 관계에 대한 행 수준 보안 정책의 접속사 유형을 정의할 수 있습니다. 자세한 내용은 사용자별로 여러 정책 결합 단원을 참조하십시오.

- FOR DATASHARES

  RLS로 보호되는 관계에 데이터 공유를 통해 액세스할 수 있는지를 결정하는 절입니다. 기본적으로 RLS로 보호되는 관계는 데이터 공유를 통해 액세스할 수 없습니다. ALTER TABLE ROW LEVEL SECURITY 명령을 이 절과 함께 실행하면 관계의 데이터 공유 접근성 속성에만 영향을 줍니다. ROW LEVEL SECURITY 속성은 변경되지 않습니다.

  RLS로 보호되는 관계에 데이터 공유를 통해 액세스할 수 있도록 설정하면 소비자 측 데이터 공유 데이터베이스에서 해당 관계에 행 수준 보안이 적용되지 않습니다. 이 관계는 생산자 측에서 해당 RLS 속성을 유지합니다.

예시

ALTER TABLE 명령을 사용하는 방법을 보여주는 예제는 다음을 참조하세요.

- ALTER TABLE 예
- ALTER EXTERNAL TABLE 예
- ALTER TABLE ADD 및 DROP COLUMN 예

ALTER TABLE 예

다음은 ALTER TABLE 명령의 기본적인 사용법을 보여주는 예입니다.

테이블 또는 뷰의 이름 바꾸기

다음 명령을 실행하면 USERS 테이블의 이름이 USERS_BKUP로 바뀝니다.

```
alter table users
rename to users_bkup;
```

이 형식의 명령을 사용하여 뷰의 이름을 바꿀 수도 있습니다.

테이블 또는 뷰의 소유자 변경

다음 명령을 실행하면 VENUE 테이블 소유자가 사용자 DWUSER로 변경됩니다.

```
alter table venue
owner to dwuser;
```

다음 명령을 실행하면 뷰가 생성된 다음, 소유자가 변경됩니다.

```
create view vdate as select * from date;
alter table vdate owner to vuser;
```

열 이름 바꾸기

다음 명령을 실행하면 VENUE 테이블에 있는 VENUESEATS 열의 이름이 VENUESIZE로 바뀝니다.

```
alter table venue
rename column venueseats to venuesize;
```

테이블 제약 조건 삭제

기본 키, 외래 키 또는 고유한 제약 조건과 같은 테이블 제약 조건을 삭제하려면, 먼저 제약 조건의 내부 이름을 찾습니다. 그런 다음 ALTER TABLE 명령에서 제약 이름을 지정합니다. 다음 예제에서는

CATEGORY 테이블에 대한 제약 조건을 찾은 다음, 이름이 category_pkey인 기본 키를 삭제합니다.

```
select constraint_name, constraint_type
from information_schema.table_constraints
where constraint_schema ='public'
and table_name = 'category';

constraint_name | constraint_type
----------------+----------------
category_pkey   | PRIMARY KEY

alter table category
drop constraint category_pkey;
```

VARCHAR 열 변경

스토리지를 절약하려면 처음에 현재 데이터 요구 사항에 필요한 최소 크기인 VARCHAR 열로 테이블을 정의하면 됩니다. 나중에 더 긴 문자열을 수용해야 하는 경우 테이블을 변경하여 열 크기를 늘릴 수 있습니다.

다음 예에서는 EVENTNAME 열 크기를 VARCHAR(300)로 변경합니다.

```
alter table event alter column eventname type varchar(300);
```

열에 대한 압축 인코딩 변경

열의 압축 인코딩을 변경할 수 있습니다. 아래에서 이 접근 방식을 보여주는 일련의 예를 찾아볼 수 있습니다. 이러한 예에 대한 테이블 정의는 다음과 같습니다.

```
create table t1(c0 int encode lzo, c1 bigint encode zstd, c2 varchar(16) encode lzo, c3
 varchar(32) encode zstd);
```

다음 문은 열 c0에 대한 압축 인코딩을 LZO 인코딩에서 AZ64 인코딩으로 변경합니다.

```
alter table t1 alter column c0 encode az64;
```

다음 문은 열 c1에 대한 압축 인코딩을 Zstandard 인코딩에서 AZ64 인코딩으로 변경합니다.

```
alter table t1 alter column c1 encode az64;
```

다음 문은 열 c2에 대한 압축 인코딩을 LZO 인코딩에서 Byte-dictionary 인코딩으로 변경합니다.

```
alter table t1 alter column c2 encode bytedict;
```

다음 문은 열 c3에 대한 압축 인코딩을 Zstandard 인코딩에서 Runlength 인코딩으로 변경합니다.

```
alter table t1 alter column c3 encode runlength;
```

DISTSTYLE KEY DISTKEY 열 변경

다음 예에서는 테이블의 DISTSTYLE 및 DISTKEY를 변경하는 방법을 보여 줍니다.

EVEN 배포 스타일의 테이블을 생성합니다. SVV_TABLE_INFO 보기는 DISTSTYLE이 EVEN임을 보여 줍니다.

```
create table inventory(
  inv_date_sk int4 not null ,
  inv_item_sk int4 not null ,
  inv_warehouse_sk int4 not null ,
  inv_quantity_on_hand int4
) diststyle even;

Insert into inventory values(1,1,1,1);

select "table", "diststyle" from svv_table_info;

   table   |   diststyle
-----------+---------------
 inventory |     EVEN
```

DISTKEY 테이블을 inv_warehouse_sk로 변경합니다. SVV_TABLE_INFO 보기는 inv_warehouse_sk 열을 결과 배포 키로 보여 줍니다.

```
alter table inventory alter diststyle key distkey inv_warehouse_sk;

select "table", "diststyle" from svv_table_info;

   table   |       diststyle
-----------+-----------------------
 inventory | KEY(inv_warehouse_sk)
```

DISTKEY 테이블을 inv_item_sk로 변경합니다. SVV_TABLE_INFO 보기는 inv_item_sk 열을 결과 배포 키로 보여 줍니다.

```
alter table inventory alter distkey inv_item_sk;

select "table", "diststyle" from svv_table_info;

   table   |       diststyle
-----------+----------------------
 inventory | KEY(inv_item_sk)
```

테이블을 DISTSTYLE ALL로 변경

다음 예에서는 테이블을 DISTSTYLE ALL로 변경하는 방법을 보여 줍니다.

EVEN 배포 스타일의 테이블을 생성합니다. SVV_TABLE_INFO 보기는 DISTSTYLE이 EVEN임을 보여 줍니다.

```
create table inventory(
  inv_date_sk int4 not null ,
  inv_item_sk int4 not null ,
  inv_warehouse_sk int4 not null ,
  inv_quantity_on_hand int4
) diststyle even;

Insert into inventory values(1,1,1,1);

select "table", "diststyle" from svv_table_info;

   table   |   diststyle
-----------+---------------
 inventory |     EVEN
```

DISTSTYLE 테이블을 ALL로 변경합니다. SVV_TABLE_INFO 보기에는 변경된 DISTSYTLE이 표시됩니다.

```
alter table inventory alter diststyle all;

select "table", "diststyle" from svv_table_info;

   table   |   diststyle
-----------+---------------
```

```
 inventory |     ALL
```

테이블 SORTKEY 변경

복합 정렬 키가 있거나 정렬 키가 없도록 테이블을 변경할 수 있습니다.

다음 테이블 정의에서 테이블 t1은 인터리브 정렬 키로 정의됩니다.

```
create table t1 (c0 int, c1 int) interleaved sortkey(c0, c1);
```

다음 명령은 테이블을 인터리브 정렬 키에서 복합 정렬 키로 변경합니다.

```
alter table t1 alter sortkey(c0, c1);
```

다음 명령은 인터리브 정렬 키를 제거하도록 테이블을 변경합니다.

```
alter table t1 alter sortkey none;
```

다음 테이블 정의에서 테이블 t1은 정렬 키로 열 c0을 사용하여 정의됩니다.

```
create table t1 (c0 int, c1 int) sortkey(c0);
```

다음 명령은 테이블 t1을 복합 정렬 키로 변경합니다.

```
alter table t1 alter sortkey(c0, c1);
```

테이블을 ENCODE AUTO로 변경

다음 예에서는 테이블을 ENCODE AUTO로 변경하는 방법을 보여줍니다.

이 예에 대한 테이블 정의는 다음과 같습니다. 열 c0은 인코딩 형식 AZ64로 정의되고 열 c1은 인코딩 형식 LZO로 정의됩니다.

```
create table t1(c0 int encode AZ64, c1 varchar encode LZO);
```

이 테이블의 경우 다음 문은 인코딩을 AUTO로 변경합니다.

```
alter table t1 alter encode auto;
```

다음 예에서는 ENCODE AUTO 설정을 제거하기 위해 테이블을 변경하는 방법을 보여줍니다.

이 예에 대한 테이블 정의는 다음과 같습니다. 테이블 열은 인코딩 없이 정의됩니다. 이 경우 인코딩은 ENCODE AUTO로 기본 설정됩니다.

```
create table t2(c0 int, c1 varchar);
```

이 테이블의 경우 다음 문은 c0 열의 인코딩을 LZO로 변경합니다. 테이블 인코딩이 더 이상 ENCODE AUTO로 설정되지 않습니다.

```
alter table t2 alter column c0 encode lzo;;
```

행 수준 보안 통제 변경

다음 명령은 테이블에 대해 RLS를 해제합니다.

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY OFF;
```

다음 명령은 테이블에 대해 RLS를 설정합니다.

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON;
```

다음 명령은 테이블에 대해 RLS를 활성화하고 데이터 공유를 통해 액세스할 수 있도록 합니다.

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON;
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY FOR DATASHARES OFF;
```

다음 명령은 테이블에 대해 RLS를 활성화하고 데이터 공유를 통해 액세스할 수 없도록 합니다.

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON;
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY FOR DATASHARES ON;
```

다음 명령은 테이블에 대해 RLS를 활성화하고 RLS 접속사 유형을 OR로 설정합니다.

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON CONJUNCTION TYPE OR;
```

다음 명령은 테이블에 대해 RLS를 활성화하고 RLS 접속사 유형을 AND로 설정합니다.

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON CONJUNCTION TYPE AND;
```

ALTER EXTERNAL TABLE 예

다음 예시에서는 미국 동부(버지니아 북부) 리전(us-east-1) AWS 리전에 있는 Amazon S3 버킷과 예시에서 CREATE TABLE에 대해 생성한 예시 테이블을 사용합니다. 이 외부 테이블 포함 파티션을 사용하는 방법에 대한 자세한 내용은 Redshift Spectrum 외부 테이블 파티셔닝 섹션을 참조하세요.

다음 예에서는 SPECTRUM.SALES 외부 테이블의 numRows 테이블 속성을 170,000개 행으로 설정합니다.

```
alter table spectrum.sales
set table properties ('numRows'='170000');
```

다음 예에서는 SPECTRUM.SALES 외부 테이블의 위치를 변경합니다.

```
alter table spectrum.sales
set location 's3://redshift-downloads/tickit/spectrum/sales/';
```

다음 예에서는 SPECTRUM.SALES 외부 테이블의 형식을 Parquet로 변경합니다.

```
alter table spectrum.sales
set file format parquet;
```

다음 예에서는 테이블 SPECTRUM.SALES_PART에 대한 파티션을 한 개 추가합니다.

```
alter table spectrum.sales_part
add if not exists partition(saledate='2008-01-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-01/';
```

다음 예에서는 테이블 SPECTRUM.SALES_PART에 대한 파티션을 세 개 추가합니다.

```
alter table spectrum.sales_part add if not exists
partition(saledate='2008-01-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-01/'
partition(saledate='2008-02-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-02/'
partition(saledate='2008-03-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-03/';
```

다음 예에서는 saledate='2008-01-01''를 포함한 파티션을 삭제하도록 SPECTRUM.SALES_PART를 변경합니다.

```
alter table spectrum.sales_part
drop partition(saledate='2008-01-01');
```

다음 예에서는 saledate='2008-01-01'을 포함한 파티션에 대한 Amazon S3 경로를 새로 설정합니다.

```
alter table spectrum.sales_part
partition(saledate='2008-01-01')
set location 's3://redshift-downloads/tickit/spectrum/sales_partition/
saledate=2008-01-01/';
```

다음 예에서는 sales_date의 이름을 transaction_date로 바꿉니다.

```
alter table spectrum.sales rename column sales_date to transaction_date;
```

다음 예에서는 열 매핑을 ORC(Optimized Row Columnar) 형식을 사용하는 외부 테이블의 위치 매핑으로 설정합니다.

```
alter table spectrum.orc_example
set table properties('orc.schema.resolution'='position');
```

다음 예에서는 열 매핑을 ORC 형식을 사용하는 외부 테이블의 이름 매핑으로 설정합니다.

```
alter table spectrum.orc_example
set table properties('orc.schema.resolution'='name');
```

ALTER TABLE ADD 및 DROP COLUMN 예

다음 예에서는 ALTER TABLE을 사용하여 기본 테이블 열을 추가한 다음에 삭제하는 방법과 종속 객체가 있는 열을 삭제하는 방법을 보여줍니다.

기본 열을 ADD한 다음 DROP

다음 예에서는 독립형 FEEDBACK_SCORE 열을 USERS 테이블에 추가합니다. 이 열에는 간단하게 정수만 포함되어 있는데, 이 열의 기본값은 NULL(피드백 점수 없음)입니다.

먼저, PG_TABLE_DEF 카탈로그 테이블을 쿼리하여 USERS 테이블의 스키마를 봅니다.

```
column        | type                    | encoding | distkey | sortkey
```

```
-------------+------------------------+----------+---------+--------
userid       | integer                | delta    | true    |       1
username     | character(8)           | lzo      | false   |       0
firstname    | character varying(30)  | text32k  | false   |       0
lastname     | character varying(30)  | text32k  | false   |       0
city         | character varying(30)  | text32k  | false   |       0
state        | character(2)           | bytedict | false   |       0
email        | character varying(100) | lzo      | false   |       0
phone        | character(14)          | lzo      | false   |       0
likesports   | boolean                | none     | false   |       0
liketheatre  | boolean                | none     | false   |       0
likeconcerts | boolean                | none     | false   |       0
likejazz     | boolean                | none     | false   |       0
likeclassical | boolean               | none     | false   |       0
likeopera    | boolean                | none     | false   |       0
likerock     | boolean                | none     | false   |       0
likevegas    | boolean                | none     | false   |       0
likebroadway | boolean                | none     | false   |       0
likemusicals | boolean                | none     | false   |       0
```

이제 feedback_score 열을 추가합니다.

```
alter table users
add column feedback_score int
default NULL;
```

USERS에서 FEEDBACK_SCORE 열을 선택하여 열이 추가되었는지 확인합니다.

```
select feedback_score from users limit 5;

feedback_score
----------------
NULL
NULL
NULL
NULL
NULL
```

열을 삭제하여 원래 DDL을 복구합니다.

```
alter table users drop column feedback_score;
```

종속 객체가 있는 열 삭제

다음 예에서는 종속 객체가 있는 열을 삭제합니다. 결과적으로, 종속 객체도 삭제됩니다.

시작하려면 FEEDBACK_SCORE 열을 USERS 테이블에 다시 추가합니다.

```
alter table users
add column feedback_score int
default NULL;
```

다음으로, USERS_VIEW라는 USERS 테이블에서 뷰를 생성합니다.

```
create view users_view as select * from users;
```

이제, FEEDBACK_SCORE 열을 USERS 테이블에서 삭제해 봅니다. 다음 DROP 문에서는 기본 동작(RESTRICT)을 사용합니다.

```
alter table users drop column feedback_score;
```

다른 객체가 이 열에 종속되어 있으므로 Amazon Redshift가 이 열을 삭제할 수 없다는 오류 메시지를 표시합니다.

이번에는 모든 종속 객체를 삭제하도록 CASCADE를 지정하여 FEEDBACK_SCORE 열을 다시 삭제해 봅니다.

```
alter table users
drop column feedback_score cascade;
```

ALTER TABLE APPEND

기존 원본 테이블에서 데이터를 이동시켜 대상 테이블에 행을 추가합니다. 원본 테이블의 데이터가 대상 테이블에서 일치하는 열로 이동됩니다. 열 순서는 중요하지 않습니다. 대상 테이블에 데이터가 성공적으로 추가되면 원본 테이블은 비게 됩니다. ALTER TABLE APPEND는 일반적으로 데이터가 복제되지 않고 이동되므로 유사한 CREATE TABLE AS 또는 INSERT INTO 작업보다 훨씬 빠릅니다.

> **Note**
>
> ALTER TABLE APPEND는 원본 테이블과 대상 테이블 간에 데이터 블록을 이동합니다. 성능 개선을 위해 ALTER TABLE APPEND는 추가 작업을 수행할 때 스토리지를 압축하지 않습니

다. 따라서 스토리지 사용량이 일시적으로 증가합니다. 공간을 회수하려면 VACUUM 작업을 실행합니다.

이름이 같은 열은 열 속성도 같아야 합니다. 원본 테이블 또는 대상 테이블에 다른 테이블에는 없는 열이 포함되어 있는 경우, IGNOREEXTRA 또는 FILLTARGET 파라미터를 사용하여 추가 열을 관리하는 방법을 지정하십시오.

자격 증명 열을 추가할 수 없습니다. 두 테이블 모두 자격 증명 열을 포함한 경우 이 명령은 실패합니다. 한 테이블에만 자격 증명 열이 있는 경우 FILLTARGET 또는 IGNOREEXTRA 파라미터를 포함하십시오. 자세한 내용은 ALTER TABLE APPEND 사용 시 주의 사항 단원을 참조하십시오.

GENERATED BY DEFAULT AS IDENTITY 열을 추가할 수 있습니다. GENERATED BY DEFAULT AS IDENTITY로 정의된 열을 직접 입력하는 값으로 업데이트할 수 있습니다. 자세한 내용은 ALTER TABLE APPEND 사용 시 주의 사항 단원을 참조하십시오.

대상 테이블은 영구 테이블이어야 합니다. 그러나 소스는 영구 테이블이거나 스트리밍 수집을 위해 구성된 구체화된 뷰일 수 있습니다. 두 객체 모두 동일한 배포 스타일과 배포 키(정의된 경우)를 사용해야 합니다. 객체가 정렬되어 있는 경우 두 객체 모두 동일한 정렬 스타일을 사용하고 정렬 키와 동일한 열을 정의해야 합니다.

ALTER TABLE APPEND 명령은 작업 완료 즉시 자동으로 커밋합니다. 롤백할 수 없습니다. 트랜잭션 블록(BEGIN ... END) 내에서 ALTER TABLE APPEND를 실행할 수 없습니다. 버전 관리에 대한 자세한 내용은 직렬화 가능 격리 단원을 참조하십시오.

필수 권한

ALTER TABLE APPEND 명령에 따라 다음 권한 중 하나가 필요합니다.

- 수퍼유저
- ALTER TABLE 시스템 권한이 있는 사용자
- 원본 테이블에 대한 DELETE 및 SELECT 권한과 대상 테이블에 대한 INSERT 권한을 가진 사용자

구문

```
ALTER TABLE target_table_name APPEND FROM [ source_table_name
 | source_materialized_view_name ]
[ IGNOREEXTRA | FILLTARGET ]
```

구체화된 뷰에서 추가하는 것은 구체화된 뷰가 구체화된 뷰로 스트리밍 모으기에 대해 구성된 경우에만 작동합니다.

파라미터

target_table_name

행이 추가된 테이블의 이름입니다. 특정 스키마를 사용하려면 테이블의 이름만 지정하거나 schema_name.table_name 형식을 사용하십시오. 대상 테이블은 기존의 영구 테이블이어야 합니다.

FROM source_table_name

추가될 행을 제공하는 테이블의 이름입니다. 특정 스키마를 사용하려면 테이블의 이름만 지정하거나 schema_name.table_name 형식을 사용하십시오. 원본 테이블은 기존의 영구 테이블이어야 합니다.

FROM source_materialized_view_name

추가될 행을 제공하는 구체화된 뷰의 이름입니다. 구체화된 뷰에서 추가하는 것은 구체화된 뷰가 구체화된 뷰로 스트리밍 모으기에 대해 구성된 경우에만 작동합니다. 소스 구체화된 뷰가 이미 있어야 합니다.

IGNOREEXTRA

원본 테이블이 대상 테이블에 없는 열을 포함하는 경우 추가 열에 있는 데이터를 삭제해야 함을 지정하는 키워드입니다. FILLTARGET과 함께 IGNOREEXTRA를 사용할 수 없습니다.

FILLTARGET

대상 테이블이 원본 테이블에 없는 열을 포함하는 경우 해당 열을 DEFAULT 열 값으로 채워야 함을 지정하는 키워드입니다(하나가 정의되어 있거나 NULL인 경우). FILLTARGET과 함께 IGNOREEXTRA를 사용할 수 없습니다.

ALTER TABLE APPEND 사용 시 주의 사항

ALTER TABLE APPEND는 원본 테이블에서 대상 테이블로 동일한 열만 이동합니다. 열 순서는 중요하지 않습니다.

원본 테이블 또는 대상 테이블에 추가 열이 포함되어 있는 경우 다음 규칙에 따라 FILLTARGET 또는 IGNOREEXTRA를 사용합니다.

- 대상 테이블에 없는 열이 원본 테이블에 있는 경우 IGNOREEXTRA를 포함합니다. 이 명령은 원본 테이블에 있는 여분의 열은 무시합니다.
- 원본 테이블에 없는 열이 대상 테이블에 있는 경우 FILLTARGET을 포함합니다. 이 명령을 실행하면 대상 테이블의 추가 열에 기본 열 값이나 IDENTITY 값이 채워집니다(하나가 정의되어 있거나 NULL인 경우).
- 원본 테이블과 대상 테이블에 모두 추가 열이 있는 경우에는 명령이 실패합니다. FILLTARGET과 IGNOREEXTRA를 모두 사용할 수는 없습니다.

두 테이블에 모두 이름은 같지만 속성이 다른 열이 있는 경우 명령은 실패합니다. 이름이 같은 열들은 다음의 공통된 속성을 가져야 합니다.

- 데이터 유형
- 열 크기
- 압축 인코딩
- Null이 아님
- 정렬 스타일
- 정렬 키 열
- 분산 스타일
- 분산 키 열

자격 증명 열을 추가할 수 없습니다. 원본 테이블과 대상 테이블에 모두 자격 증명 열이 있는 경우에는 명령이 실패합니다. 원본 테이블에만 자격 증명 열이 있는 경우 자격 증명 열이 무시되도록 IGNOREEXTRA 파라미터를 포함하십시오. 대상 테이블에만 자격 증명 열이 있는 경우 테이블에 대해 정의된 IDENTITY 절에 따라 자격 증명 열이 채워지도록 FILLTARGET 파라미터를 포함하십시오. 자세한 내용은 DEFAULT 단원을 참조하십시오.

ALTER TABLE APPEND 문으로 기본 자격 증명 열을 추가할 수 있습니다. 자세한 내용은 CREATE TABLE 단원을 참조하십시오.

ALTER TABLE APPEND 예

회사에서 현재의 판매 거래를 파악하기 위해 SALES_MONTHLY라는 테이블을 관리하고 있다고 해봅시다. 매달 거래 테이블에서 SALES 테이블로 데이터를 이동하려고 합니다.

이럴 때 다음과 같이 INSERT INTO 및 TRUNCATE 명령을 사용하여 이 작업을 완수할 수 있습니다.

```
insert into sales (select * from sales_monthly);
truncate sales_monthly;
```

하지만 ALTER TABLE APPEND 명령을 사용하면 같은 작업을 훨씬 더 효율적으로 수행할 수 있습니다.

먼저 PG_TABLE_DEF 시스템 카탈로그 테이블을 쿼리하여 두 테이블에 모두 같은 열 속성을 가진 같은 열이 있는지 확인합니다.

```
select trim(tablename) as table, "column", trim(type) as type,
encoding, distkey, sortkey, "notnull"
from pg_table_def where tablename like 'sales%';

table      | column     | type                          | encoding | distkey | sortkey |
 notnull
-----------+------------+-------------------------------+----------+---------+---------
+--------
sales      | salesid    | integer                       | lzo      | false   |       0 |
 true
sales      | listid     | integer                       | none     | true    |       1 |
 true
sales      | sellerid   | integer                       | none     | false   |       2 |
 true
sales      | buyerid    | integer                       | lzo      | false   |       0 |
 true
sales      | eventid    | integer                       | mostly16 | false   |       0 |
 true
sales      | dateid     | smallint                      | lzo      | false   |       0 |
 true
sales      | qtysold    | smallint                      | mostly8  | false   |       0 |
 true
sales      | pricepaid  | numeric(8,2)                  | delta32k | false   |       0 |
 false
sales      | commission | numeric(8,2)                  | delta32k | false   |       0 |
 false
sales      | saletime   | timestamp without time zone | lzo      | false   |       0 |
 false
salesmonth | salesid    | integer                       | lzo      | false   |       0 |
 true
salesmonth | listid     | integer                       | none     | true    |       1 |
 true
salesmonth | sellerid   | integer                       | none     | false   |       2 |
 true
```

```
salesmonth | buyerid    | integer                     | lzo      | false   |       0 |
 true
salesmonth | eventid    | integer                     | mostly16 | false   |       0 |
 true
salesmonth | dateid     | smallint                    | lzo      | false   |       0 |
 true
salesmonth | qtysold    | smallint                    | mostly8  | false   |       0 |
 true
salesmonth | pricepaid  | numeric(8,2)                | delta32k | false   |       0 |
 false
salesmonth | commission | numeric(8,2)                | delta32k | false   |       0 |
 false
salesmonth | saletime   | timestamp without time zone | lzo      | false   |       0 |
 false
```

다음으로, 각 테이블의 크기를 살펴봅니다.

```
select count(*) from sales_monthly;
 count
-------
  2000
(1 row)

select count(*) from sales;
 count
-------
 412,214
(1 row)
```

이제는 다음 ALTER TABLE APPEND 명령을 실행합니다.

```
alter table sales append from sales_monthly;
```

각 테이블의 크기를 다시 살펴봅니다. 지금 SALES_MONTHLY 테이블의 행 개수는 0개인데 반해, SALES 테이블의 행 개수는 2,000개로 늘었습니다.

```
select count(*) from sales_monthly;
 count
-------
     0
(1 row)
```

```
select count(*) from sales;
 count
-------
 414214
(1 row)
```

원본 테이블의 열 개수가 대상 테이블의 열 개수보다 많은 경우 IGNOREEXTRA 파라미터를 지정합니다. 다음 예에서는 IGNOREEXTRA 파라미터를 사용하여 SALES 테이블에 추가할 때 SALES_LISTING 테이블에 있는 추가 열을 무시합니다.

```
alter table sales append from sales_listing ignoreextra;
```

대상 테이블의 열 개수가 원본 테이블의 열 개수보다 많은 경우 FILLTARGET 파라미터를 지정합니다. 다음 예에서는 FILLTARGET 파라미터를 사용하여 SALES_MONTH 테이블에는 없는 열을 SALES_REPORT 테이블에 채웁니다.

```
alter table sales_report append from sales_month filltarget;
```

다음 예는 구체화된 뷰를 소스로 사용하여 ALTER TABLE APPEND를 사용하는 방법을 보여줍니다.

```
ALTER TABLE target_tbl APPEND FROM my_streaming_materialized_view;
```

이 예의 테이블 및 구체화된 뷰 이름은 샘플입니다. 구체화된 뷰에서 추가하는 것은 구체화된 뷰가 구체화된 뷰로 스트리밍 모으기에 대해 구성된 경우에만 작동합니다. 소스 구체화된 뷰의 모든 레코드를 구체화된 뷰와 동일한 스키마를 가진 대상 테이블로 이동하고 구체화된 뷰는 그대로 둡니다. 이는 데이터 소스가 테이블인 경우와 동일한 동작입니다.

ALTER USER

데이터베이스 사용자를 변경합니다.

필수 권한

ALTER USER에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- ALTER USER 권한이 있는 사용자
- 자신의 암호를 변경하려는 현재 사용자

구문

```
ALTER USER username [ WITH ] option [, ... ]

where option is

CREATEDB | NOCREATEDB
| CREATEUSER | NOCREATEUSER
| SYSLOG ACCESS { RESTRICTED | UNRESTRICTED }
| PASSWORD { 'password' | 'md5hash' | DISABLE }
[ VALID UNTIL 'expiration_date' ]
| RENAME TO new_name |
| CONNECTION LIMIT { limit | UNLIMITED }
| SESSION TIMEOUT limit | RESET SESSION TIMEOUT
| SET parameter { TO | = } { value | DEFAULT }
| RESET parameter
| EXTERNALID external_id
```

파라미터

사용자 이름

사용자의 이름입니다.

WITH

선택적 키워드입니다.

CREATEDB | NOCREATEDB

CREATEDB 옵션을 사용하여 새 데이터베이스를 만들 수 있습니다. NOCREATEDB가 기본값입니다.

CREATEUSER | NOCREATEUSER

CREATEUSER 옵션으로 CREATE USER를 포함한 모든 데이터베이스 권한을 가진 수퍼유저를 생성합니다. 기본값은 NOCREATEUSER입니다. 자세한 내용은 superuser 단원을 참조하십시오.

SYSLOG ACCESS { RESTRICTED | UNRESTRICTED }

Amazon Redshift 시스템 테이블 및 뷰에 대한 사용자의 액세스 수준을 지정합니다.

SYSLOG ACCESS RESTRICTED 권한이 있는 일반 사용자는 사용자 가시성 시스템 테이블 및 뷰에서 해당 사용자가 생성한 행만 볼 수 있습니다. 기본값은 RESTRICTED입니다.

SYSLOG ACCESS UNRESTRICTED 권한이 있는 일반 사용자는 사용자 가시성 시스템 테이블 및 뷰에서 다른 사용자가 생성한 행을 포함한 모든 행을 볼 수 있습니다. UNRESTRICTED는 수퍼유저 가시성 테이블에 대한 일반 사용자 액세스를 부여하지 않습니다. 수퍼유저만 수퍼유저 가시성 테이블을 볼 수 있습니다.

> **Note**
>
> 사용자에게 시스템 테이블에 대한 무제한 액세스를 제공하면 다른 사용자가 생성한 데이터에 대한 가시성이 사용자에게 제공됩니다. 예를 들어, STL_QUERY 및 STL_QUERYTEXT에는 INSERT, UPDATE 및 DELETE 문의 전체 텍스트가 포함되며, 여기에는 사용자가 생성한 민감한 데이터가 포함될 수 있습니다.

SVV_TRANSACTIONS의 모든 행은 모든 사용자에게 표시됩니다.

자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

PASSWORD { 'password' | 'md5hash' | DISABLE }

사용자의 암호를 설정합니다.

기본적으로, 암호가 비활성화 상태가 아닌 이상 사용자는 암호를 변경할 수 있습니다. 사용자의 암호를 비활성화하려면 DISABLE을 지정하십시오. 사용자 암호를 비활성화하면 시스템에서 해당 암호가 삭제되고 사용자는 임시 AWS Identity and Access Management(IAM) 사용자 자격 증명만 이용해 로그인할 수 있습니다. 자세한 내용은 IAM 인증을 이용한 데이터베이스 사용자 자격 증명 생성을 참조하세요. 수퍼유저만이 암호를 활성화 또는 비활성화할 수 있습니다. 수퍼유저의 암호를 비활성화할 수는 없습니다. 암호를 활성화하려면 ALTER USER를 실행하고 암호를 지정하십시오.

PASSWORD 파라미터 사용에 대한 자세한 내용은 사용자 생성 섹션을 참조하세요.

VALID UNTIL 'expiration_date'

암호에 만료 날짜가 있음을 지정합니다. 만료 날짜가 없도록 하려면 값 `'infinity'`를 사용하십시오. 이 파라미터에 유효한 데이터 형식은 타임스탬프입니다.

슈퍼유저만 이 파라미터를 사용할 수 있습니다.

RENAME TO

사용자 이름을 새로 지정합니다.

new_name

사용자의 새 이름입니다. 유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요.

> **Important**
>
> 사용자의 이름을 재설정할 때는 사용자의 암호도 변경해야 합니다. 재설정 암호는 이전 암호와 다르지 않아도 됩니다. 사용자 이름은 암호를 암호화하는 과정의 일부로 사용되므로, 사용자 이름을 바꾸면 해당 암호가 삭제됩니다. 사용자가 암호를 재설정해야 로그인할 수 있습니다. 예:
>
> ```
> alter user newuser password 'EXAMPLENewPassword11';
> ```

CONNECTION LIMIT { limit | UNLIMITED }

사용자가 동시에 열어놓을 수 있는 데이터베이스 연결의 최대 개수입니다. 슈퍼 사용자에 대해서는 이 제한이 적용되지 않습니다. 최대 동시 연결 수를 허용하려면 UNLIMITED 키워드를 사용하십시오. 각 데이터베이스에 대한 연결 개수 제한이 적용될 수도 있습니다. 자세한 내용은 데이터베이스 생성 단원을 참조하십시오. 기본값은 UNLIMITED입니다. 현재 연결을 보려면 STV_SESSIONS 시스템 뷰를 쿼리하십시오.

> **Note**
>
> 사용자 및 데이터베이스 연결 제한이 모두 적용되는 경우 사용되지 않는 연결 슬롯은 사용자가 연결 시도 시 양쪽 제한 범위 내에서 모두 사용 가능해야 합니다.

SESSION TIMEOUT limit | RESET SESSION TIMEOUT

세션이 비활성 또는 유휴 상태로 유지되는 최대 시간(초)입니다. 범위는 60초(1분)~1,728,000초(20일)입니다. 사용자에 대해 세션 시간 제한이 설정되지 않은 경우 클러스터 설정이 적용됩니다. 자세한 내용은 Amazon Redshift 관리 가이드의 Amazon Redshift의 할당량 및 제한 섹션을 참조하세요.

세션 시간 제한을 설정하면 새 세션에만 적용됩니다.

시작 시간, 사용자 이름 및 세션 시간 제한을 포함하여 활성 사용자 세션에 대한 정보를 보려면 STV_SESSIONS 시스템 뷰를 쿼리합니다. 사용자 세션 기록에 대한 정보를 보려면 STL_SESSIONS 뷰를 쿼리합니다. 세션 시간 제한 값을 포함하여 데이터베이스 사용자에 대한 정보를 검색하려면 SVL_USER_INFO 뷰를 쿼리합니다.

SET

지정된 사용자가 실행하는 모든 세션에 대한 새로운 기본값으로 구성 파라미터를 설정합니다.

reset

지정된 사용자에 대한 원래 기본값으로 구성 파라미터를 재설정합니다.

파라미터

설정할 파라미터의 이름.

USD 상당

파라미터의 새 값입니다.

DEFAULT

지정된 사용자가 실행하는 모든 세션에 대한 기본값으로 구성 파라미터를 설정합니다.

EXTERNALID external_id

자격 증명 공급자와 연결된 사용자의 식별자입니다. 사용자는 암호를 비활성화해야 합니다. 자세한 내용은 Amazon Redshift용 네이티브 자격 증명 공급자(IdP) 페더레이션을 참조하세요.

사용 노트

- rdsdb 변경 시도 - 이름이 `rdsdb`인 사용자를 변경할 수 없습니다.
- 알 수 없는 암호 생성 - AWS Identity and Access Management(IAM) 인증을 이용해 데이터베이스 사용자 보안 인증 정보를 만들 때는 임시 보안 인증 정보만을 이용해 로그인할 수 있는 수퍼유저를 만드는 것이 좋습니다. 수퍼유저의 암호를 비활성화할 수는 없지만 임의로 생성되는 MD5 해시 문자열을 이용해 알 수 없는 암호를 만들 수는 있습니다.

```
alter user iam_superuser password 'md5123456789012345678012345678901 2';
```

- search_path 설정 - ALTER USER 명령으로 search_path 파라미터를 설정하면 수정된 사항은 지정된 사용자가 다음에 로그인할 때 적용됩니다. 현재 사용자와 세션에 대해 search_path 값을 변경하려면 SET 명령을 사용합니다.
- 시간대 설정 - ALTER USER 명령으로 SET TIMEZONE을 사용하면 수정된 사항은 지정된 사용자가 다음에 로그인할 때 적용됩니다.
- 동적 데이터 마스킹 및 행 수준 보안 정책 작업 - 프로비저닝된 클러스터 또는 서버리스 네임스페이스에 동적 데이터 마스킹 또는 행 수준 보안 정책이 있는 경우, 일반 사용자에게는 다음 명령이 차단됩니다.

```
ALTER <current_user> SET enable_case_sensitive_super_attribute/
enable_case_sensitive_identifier/downcase_delimited_identifier
```

슈퍼유저와 ALTER USER 권한이 있는 사용자만 이러한 구성 옵션을 설정할 수 있습니다. 행 수준 보안에 대한 자세한 내용은 행 수준 보안 섹션을 참조하세요. 동적 데이터 마스킹에 대한 자세한 내용은 동적 데이터 마스킹 섹션을 참조하세요.

예시

다음 예에서는 데이터베이스 생성 권한을 사용자 ADMIN에게 부여합니다.

```
alter user admin createdb;
```

다음 예에서는 사용자 ADMIN의 암호를 adminPass9로 설정하고 암호에 대한 만료 날짜와 시간을 설정합니다.

```
alter user admin password 'adminPass9'
valid until '2017-12-31 23:59';
```

다음 예에서는 사용자 ADMIN의 이름을 SYSADMIN으로 바꿉니다.

```
alter user admin rename to sysadmin;
```

다음 예에서는 사용자에 대한 유휴 세션 시간 제한을 300초로 업데이트합니다.

```
ALTER USER dbuser SESSION TIMEOUT 300;
```

사용자의 유휴 세션 시간 제한을 재설정합니다. 재설정하면 클러스터 설정이 적용됩니다. 이 명령을 실행하려면 데이터베이스 슈퍼 사용자여야 합니다. 자세한 내용은 Amazon Redshift 관리 가이드의 Amazon Redshift의 할당량 및 제한 섹션을 참조하세요.

```
ALTER USER dbuser RESET SESSION TIMEOUT;
```

다음 예에서는 이름이 bob인 사용자의 외부 ID를 업데이트합니다. 네임스페이스는 myco_aad입니다. 네임스페이스가 등록된 자격 증명 공급자와 연결되지 않은 경우 오류가 발생합니다.

```
ALTER USER myco_aad:bob EXTERNALID "ABC123" PASSWORD DISABLE;
```

다음 예는 특정 데이터베이스 사용자가 실행하는 모든 세션의 시간대를 설정합니다. 현재 세션이 아닌 이후 모든 세션의 시간대가 변경됩니다.

```
ALTER USER odie SET TIMEZONE TO 'Europe/Zurich';
```

다음 예시에서는 사용자 bob이 열 수 있는 최대 데이터베이스 연결 수를 설정합니다.

```
ALTER USER bob CONNECTION LIMIT 10;
```

ANALYZE

쿼리 플래너가 사용할 테이블 통계를 업데이트합니다.

필수 권한

ANALYZE에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- ANALYZE 권한이 있는 사용자
- 관계 소유자
- 테이블이 공유되는 데이터베이스 소유자

구문

```
ANALYZE [ VERBOSE ]
[ [ table_name [ ( column_name [, ...] ) ] ]
[ PREDICATE COLUMNS | ALL  COLUMNS ]
```

파라미터

상세 표시

ANALYZE 작업의 진행률 정보 메시지를 반환하는 절입니다. 이 옵션은 테이블을 지정하지 않을 때 유용합니다.

table_name

임시 테이블을 포함한 특정 테이블을 분석할 수 있습니다. 테이블은 테이블의 스키마 이름으로 정규화할 수 있습니다. 선택적으로 단일 테이블을 분석하도록 table_name을 지정할 수 있습니다. 단일 ANALYZE table_name 문으로 table_name을 여러 개 지정할 수 없습니다. table_name 값을 지

정하지 않으면 시스템 카탈로그의 영구 테이블을 포함하여 현재 연결된 데이터베이스의 모든 테이블이 분석됩니다. Amazon Redshift는 마지막 ANALYZE 이후 변경된 행의 비율이 분석 임계값보다 낮은 경우 테이블 분석을 건너뜁니다. 자세한 내용은 분석 임계값 단원을 참조하십시오.

Amazon Redshift 시스템 테이블(STL 및 STV 테이블)을 분석할 필요는 없습니다.

column_name

table_name을 지정하는 경우 테이블에 있는 하나 이상의 열을 지정할 수도 있습니다(괄호 안의 열로 구분된 목록으로). 열 목록을 지정하면 나열된 열만 분석됩니다.

PREDICATE COLUMNS | ALL COLUMNS

ANALYZE에 조건자 열만 포함시킬지 나타내는 절입니다. 이전 쿼리에서 조건자로 사용되었거나 조건자로 사용될 수 있는 열만 분석하려면 PREDICATE COLUMNS를 지정합니다. 모든 열을 분석하려면 ALL COLUMNS를 지정합니다. 기본값은 ALL COLUMNS입니다.

열이 조건자 열 집합에 포함되기 위해서는 다음 중 한 가지만 만족하면 됩니다.

- 열이 쿼리에서 filter, join condition 또는 group by 절의 일부로 사용되었습니다.
- 분산 키인 열
- 정렬 키에 포함되는 열

예를 들어 테이블에 대한 쿼리를 아직 실행하지 않아서 조건자 열로 표시되는 열이 하나도 없는 경우에는 PREDICATE COLUMNS로 지정하더라도 모든 열이 분석됩니다. 이 경우 Amazon Redshift는 '*table-name*'에 대한 조건자 열을 찾을 수 없음과 같은 메시지로 응답할 수 있습니다. 모든 열을 분석하고 있습니다. 조건자 열에 대한 자세한 내용은 테이블 분석 섹션을 참조하세요.

사용 노트

Amazon Redshift는 다음 명령으로 생성하는 테이블에서 ANALYZE를 자동으로 실행합니다.

- CREATE TABLE AS
- CREATE TEMP TABLE AS
- SELECT INTO

외부 테이블은 분석할 수 없습니다.

이 테이블들이 처음 생성될 때는 테이블에서 ANALYZE 명령을 실행할 필요가 없습니다. 테이블을 수정하는 경우, 다른 테이블과 같은 방법으로 분석해야 합니다.

분석 임계값

처리 시간을 단축하고 전체 시스템 성능을 개선하기 위해 Amazon Redshift는 마지막 ANALYZE 명령 실행 이후로 변경된 행의 비율이 analyze_threshold_percent 파라미터에 의해 지정된 분석 임계값보다 낮은 경우 테이블 ANALYZE를 건너뜁니다. 기본적으로 analyze_threshold_percent는 10입니다. 현재 세션에 대한 analyze_threshold_percent를 변경하려면 SET 명령을 실행합니다. 다음 예에서는 analyze_threshold_percent를 20퍼센트로 변경합니다.

```
set analyze_threshold_percent to 20;
```

소수의 행만 변경되었을 때 테이블을 분석하려면 analyze_threshold_percent를 임의의 작은 수로 설정하십시오. 예를 들어 analyze_threshold_percent를 0.01로 설정하면 10,000개 이상의 행이 변경된 경우 100,000,000개의 행이 있는 테이블을 건너뛰지 않습니다.

```
set analyze_threshold_percent to 0.01;
```

테이블이 분석 임계값을 충족하지 못해 ANALYZE가 테이블을 건너뛰는 경우 Amazon Redshift는 다음 메시지를 반환합니다.

```
ANALYZE SKIP
```

변경된 행이 없더라도 모든 테이블을 분석하려면 analyze_threshold_percent를 0으로 설정하십시오.

ANALYZE 작업의 결과를 보려면 STL_ANALYZE 시스템 테이블을 쿼리하십시오.

테이블 분석에 대한 자세한 내용은 테이블 분석 섹션을 참조하세요.

예시

TICKIT 데이터베이스에 있는 모든 테이블을 분석하고 진행률 정보를 반환합니다.

```
analyze verbose;
```

LISTING 테이블만 분석합니다.

```
analyze listing;
```

VENUE 테이블에서 VENUEID 및 VENUENAME 열을 분석합니다.

```
analyze venue(venueid, venuename);
```

VENUE 테이블에서 조건자 열만 분석합니다.

```
analyze venue predicate columns;
```

ANALYZE COMPRESSION

분석 대상 테이블에 대해 제안되는 압축 인코딩으로 압축 분석을 수행하고 보고서를 생성합니다. 각각의 열에 대해, 보고서에는 원시 인코딩에 비해 디스크 공간의 잠재적 감소 추정치가 포함됩니다.

구문

```
ANALYZE COMPRESSION
[ [ table_name ]
[ ( column_name [, ...] ) ] ]
[COMPROWS numrows]
```

파라미터

table_name

임시 테이블을 포함한 특정 테이블에 대한 압축을 분석할 수 있습니다. 테이블은 테이블의 스키마 이름으로 정규화할 수 있습니다. 선택적으로 단일 테이블을 분석하도록 table_name을 지정할 수 있습니다. table_name을 지정하지 않으면 현재 연결된 데이터베이스에 있는 모든 테이블이 분석됩니다. 단일 ANALYZE COMPRESSION 문으로 table_name을 두 개 이상 지정할 수 없습니다.

column_name

table_name을 지정하는 경우 테이블에 있는 하나 이상의 열을 지정할 수도 있습니다(괄호 안의 열로 구분된 목록으로).

COMPROWS

압축 분석을 위한 샘플 크기로 사용할 행의 개수입니다. 분석은 각 데이터 조각의 행에서 실행됩니다. 예를 들어, COMPROWS 1000000(1,000,000)을 지정하고 시스템에 총 4개의 조각이 있는 경우 조각당 250,000개의 행만 읽히고 분석됩니다. COMPROWS가 지정되지 않은 경우 샘플 크기는 기본적으로 조각당 100,000개입니다. 기본값인 조각당 100,000개의 행보다 낮은 COMPROWS 값은

자동으로 기본값으로 업그레이드됩니다. 하지만 테이블에 있는 데이터의 양이 유의미한 샘플을 생성하기에 부족한 경우 압축 분석에서는 권장 사항을 제시하지 않습니다. COMPROWS 숫자가 테이블의 행 개수보다 큰 값이면 ANALYZE COMPRESSION 명령이 계속 진행하고 사용 가능한 모든 행에 대한 압축 분석을 실행합니다. 테이블이 지정되지 않은 경우 COMPROWS를 사용하면 오류가 발생합니다.

numrows

압축 분석을 위한 샘플 크기로 사용할 행의 개수입니다. numrows의 허용 범위는 1000과 1000000000(1,000,000,000) 사이의 수입니다.

사용 노트

ANALYZE COMPRESSION은 배타적인 테이블 잠금을 획득하여 테이블에 대한 동시 읽기 및 쓰기를 방지합니다. 테이블이 유휴 상태일 때만 ANALYZE COMPRESSION을 실행하십시오.

테이블 내용의 샘플을 바탕으로 열 인코딩 체계에 대한 권장 사항을 받으려면 ANALYZE COMPRESSION을 실행하십시오. ANALYZE COMPRESSION은 자문 도구로서 테이블의 열 인코딩을 수정하지 않습니다. 테이블을 다시 생성하거나 같은 스키마를 가진 새 테이블을 생성하여 제안되는 인코딩을 적용할 수 있습니다. 적절한 인코딩 체계를 사용하여 압축되지 않은 테이블을 다시 생성하면 디스크 상의 공간을 크게 줄일 수 있습니다. 이 방법을 사용하면 디스크 공간을 절약하고 I/O 바인딩 워크로드의 쿼리 성능을 향상시킬 수 있습니다.

ANALYZE COMPRESSION은 실제 분석 단계를 건너뛰고, SORTKEY로 지정된 모든 열에서 원래 인코딩 형식을 직접 반환합니다. SORTKEY 열이 다른 열보다 훨씬 많이 압축되면 범위 제한 스캔이 제대로 수행되지 않을 수 있기 때문입니다.

예시

다음 예는 LISTING 테이블 내 열에 대해서만 인코딩 및 추정 백분율 감소를 보여 줍니다.

```
analyze compression listing;

  Table  |     Column     | Encoding | Est_reduction_pct
---------+----------------+----------+-------------------
 listing | listid         | az64     | 40.96
 listing | sellerid       | az64     | 46.92
 listing | eventid        | az64     | 53.37
 listing | dateid         | raw      | 0.00
 listing | numtickets     | az64     | 65.66
 listing | priceperticket | az64     | 72.94
```

```
 listing | totalprice     | az64     | 68.05
 listing | listtime       | az64     | 49.74
```

다음 예는 SALES 테이블에서 QTYSOLD, COMMISSION 및 SALETIME 열을 분석합니다.

```
analyze compression sales(qtysold, commission, saletime);

 Table |   Column   | Encoding | Est_reduction_pct
-------+------------+----------+-------------------
 sales | salesid    | N/A      | 0.00
 sales | listid     | N/A      | 0.00
 sales | sellerid   | N/A      | 0.00
 sales | buyerid    | N/A      | 0.00
 sales | eventid    | N/A      | 0.00
 sales | dateid     | N/A      | 0.00
 sales | qtysold    | az64     | 83.06
 sales | pricepaid  | N/A      | 0.00
 sales | commission | az64     | 71.85
 sales | saletime   | az64     | 49.63
```

마스킹 정책 연결

기존 동적 데이터 마스킹 정책을 열에 연결합니다. 동적 데이터 마스킹에 대한 자세한 내용은 동적 데이터 마스킹(동적 데이터 마스킹(미리 보기)을 참조하세요.

sys:secadmin 역할이 부여된 수퍼유저와 사용자 또는 역할은 마스킹 정책을 연결할 수 있습니다.

구문

```
ATTACH MASKING POLICY policy_name
   ON { relation_name }
   ( {output_columns_names | output_path} ) [ USING ( {input_column_names | input_path
 )} ]
   TO { user_name | ROLE role_name | PUBLIC }
   [ PRIORITY priority ];
```

파라미터

policy_name

연결할 마스킹 정책 이름입니다.

relation_name

마스킹 정책을 연결할 관계의 이름입니다.

output_column_names

마스킹 정책이 적용될 열의 이름입니다.

output_paths

열 이름을 포함하여 마스킹 정책이 적용될 SUPER 객체의 전체 경로입니다. 예를 들어 이름이 `person`인 SUPER 유형 열이 있는 관계의 경우 output_path는 `person.name.first_name`일 것입니다.

input_column_names

마스킹 정책에서 입력으로 사용할 열의 이름입니다. 이 파라미터는 선택 사항입니다. 지정하지 않으면 마스킹 정책은 output_column_names를 입력으로 사용합니다.

input_paths

마스킹 정책에서 입력으로 사용할 SUPER 객체의 전체 경로입니다. 이 파라미터는 선택 사항입니다. 지정하지 않으면 마스킹 정책은 output_path를 입력으로 사용합니다.

user_name

마스킹 정책이 연결될 사용자의 이름입니다. 사용자와 열 또는 역할과 열의 동일한 조합에 두 개의 정책을 연결할 수 없습니다. 정책을 사용자에게 연결하고 다른 정책을 사용자의 역할에 연결할 수 있습니다. 이 경우 우선 순위가 높은 정책이 적용됩니다.

단일 ATTACH MASKING POLICY 명령에서 user_name, role_name 및 PUBLIC 중 하나만 설정할 수 있습니다.

role_name

마스킹 정책이 연결될 역할의 이름입니다. 동일한 열/역할 쌍에 두 개의 정책을 연결할 수 없습니다. 정책을 사용자에게 연결하고 다른 정책을 사용자의 역할에 연결할 수 있습니다. 이 경우 우선 순위가 높은 정책이 적용됩니다.

단일 ATTACH MASKING POLICY 명령에서 user_name, role_name 및 PUBLIC 중 하나만 설정할 수 있습니다.

PUBLIC

테이블에 액세스하는 모든 사용자에게 마스킹 정책을 연결합니다. 특정 열/사용자 또는 열/역할 쌍에 연결된 다른 마스킹 정책을 적용하려면 PUBLIC 정책보다 높은 우선 순위를 부여해야 합니다.

단일 ATTACH MASKING POLICY 명령에서 user_name, role_name 및 PUBLIC 중 하나만 설정할 수 있습니다.

우선순위

마스킹 정책의 우선 순위입니다. 주어진 사용자의 쿼리에 여러 마스킹 정책이 적용되는 경우 우선순위가 가장 높은 정책이 적용됩니다.

우선순위가 동일한 열에 서로 다른 두 개의 정책을 연결할 수 없습니다. 두 개의 정책이 서로 다른 사용자 또는 역할에 연결되어 있더라도 마찬가지입니다. 정책이 연결되는 사용자 또는 역할이 매번 다르면 동일한 테이블, 출력 열, 입력 열 및 우선순위 파라미터 세트에 동일한 정책을 여러 번 연결할 수 있습니다.

역할이 다르더라도 해당 열에 연결된 다른 정책과 우선 순위가 같은 열에는 정책을 적용할 수 없습니다. 이 필드는 선택 사항입니다. 우선 순위를 지정하지 않을 경우 마스킹 정책의 기본값은 0입니다.

ATTACH RLS POLICY

테이블에 대한 행 수준 보안 정책을 하나 이상의 사용자 또는 역할에 연결합니다.

`sys:secadmin` 역할이 부여된 수퍼유저와 사용자 또는 역할은 정책을 연결할 수 있습니다.

구문

```
ATTACH RLS POLICY policy_name ON [TABLE] table_name [, ...]
TO { user_name | ROLE role_name | PUBLIC } [, ...]
```

파라미터

policy_name

정책의 이름입니다.

ON [TABLE] table_name [, ...]

행 수준 보안 정책이 연결된 관계입니다.

TO { user_name | ROLE role_name | PUBLIC} [, ...]

정책이 하나 이상의 지정된 사용자 또는 역할에 연결되는지 여부를 지정합니다.

사용 노트

ATTACH RLS POLICY 문과 관련한 작업을 수행할 때는 다음을 준수하세요.

- 연결되는 테이블에는 정책 생성 문의 WITH 절에 나열된 모든 열이 있어야 합니다.
- Amazon Redshift RLS는 RLS 정책을 다음 객체에 연결하는 것을 지원합니다.
  - 표
  - 보기
  - Late Binding 보기
  - 구체화된 뷰
- Amazon Redshift RLS는 RLS 정책을 다음 객체에 연결하는 것을 지원하지 않습니다.
  - 카탈로그 테이블
  - 교차 데이터베이스 관계
  - 외부 테이블
  - 임시 테이블
  - 정책 조회 테이블
  - 구체화된 뷰 기본 테이블
- 슈퍼 사용자 또는 `sys:secadmin` 권한이 있는 사용자에게 연결된 RLS 정책은 무시됩니다.

예시

다음 예에서는 테이블에 대한 정책을 역할에 연결합니다.

```
ATTACH RLS POLICY policy_concerts ON tickit_category_redshift TO ROLE analyst, ROLE
 dbadmin;
```

BEGIN

트랜잭션을 시작합니다. START TRANSACTION과 동의어입니다.

트랜잭션은 하나의 명령으로 구성되든 여러 명령으로 구성되든, 단 하나의 논리적 작업 단위입니다. 일반적으로 한 트랜잭션에 있는 모든 명령은 `transaction_snapshot_begin` 시스템 구성 파라미터에 대해 설정된 값으로 시작 시간이 결정되는 데이터베이스의 스냅샷에서 실행됩니다.

기본적으로 개별 Amazon Redshift 작업(쿼리, DDL 문, 로드)은 데이터베이스에 자동으로 커밋됩니다. 이후의 작업이 완료될 때까지 작업에 대한 커밋을 일시 중단하려는 경우 BEGIN 문으로 트랜잭션

을 열고 필요한 명령을 실행한 다음, COMMIT 또는 END 문으로 트랜잭션을 닫아야 합니다. 필요한 경우 ROLLBACK 문을 사용하여 진행 중인 트랜잭션을 중지할 수 있습니다. 이 동작에 대한 예외가 TRUNCATE 명령으로, 명령이 실행되는 트랜잭션을 커밋하고 롤백은 불가능합니다.

구문

```
BEGIN [ WORK | TRANSACTION ] [ ISOLATION LEVEL option ] [ READ WRITE | READ ONLY ]

START TRANSACTION [ ISOLATION LEVEL option ] [ READ WRITE | READ ONLY ]

Where option is

SERIALIZABLE
| READ UNCOMMITTED
| READ COMMITTED
| REPEATABLE READ

Note: READ UNCOMMITTED, READ COMMITTED, and REPEATABLE READ have no
operational impact and map to SERIALIZABLE in Amazon Redshift. You can see database
 isolation levels on your cluster
by querying the stv_db_isolation_level table.
```

파라미터

Work

선택적 키워드입니다.

TRANSACTION

선택적 키워드: WORK와 TRANSACTION은 동의어입니다.

ISOLATION LEVEL SERIALIZABLE

직렬화 가능 격리가 기본적으로 지원되므로, 트랜잭션의 동작은 이 구문이 명령문에 포함되어 있는지에 상관없이 동일합니다. 자세한 내용은 동시 쓰기 작업 관리 단원을 참조하십시오. 다른 격리 수준은 지원되지 않습니다.

> **Note**
>
> SQL 표준에서는 더티 읽기(트랜잭션이 동시 커밋되지 않은 트랜잭션에 의해 쓰인 데이터를 읽는 경우), 반복 불가능한 읽기(트랜잭션이 이전에 읽은 데이터를 다시 읽었는데 처음 읽은 이후에 커밋된 다른 트랜잭션에 의해 해당 데이터가 변경되었음을 발견하는 경우), 가

상 읽기(트랜잭션이 쿼리를 다시 실행하고, 검색 조건을 만족하는 행 집합을 반환한 다음, 최근에 커밋된 다른 트랜잭션 때문에 행 집합이 변경된 것을 발견하는 경우)를 방지하기 위해 네 가지 트랜잭션 격리 수준을 정의합니다.

- 커밋되지 않은 데이터 읽기: 더티 읽기, 반복 불가능한 읽기 및 가상 읽기가 가능합니다.
- 커밋된 데이터 읽기: 반복 불가능한 읽기 및 가상 읽기가 가능합니다.
- 반복 가능한 읽기: 가상 읽기가 가능합니다.
- 직렬화 가능: 더티 읽기, 반복 불가능한 읽기 및 가상 읽기를 방지합니다.

4가지 트랜잭션 격리 수준 중 어떤 수준이든 사용할 수 있지만 Amazon Redshift는 모든 격리 수준을 직렬화 가능한 수준으로 처리합니다.

READ WRITE

트랜잭션 읽기 및 쓰기 권한을 제공합니다.

READ ONLY

트랜잭션 읽기 전용 권한을 제공합니다.

예시

다음 예에서는 직렬화 가능 트랜잭션 블록을 시작합니다.

```
begin;
```

다음 예에서는 직렬화 가능 격리 수준과 읽기 및 쓰기 권한으로 트랜잭션 블록을 시작합니다.

```
begin read write;
```

CALL

저장 프로시저를 실행합니다. CALL 명령에는 프로시저 이름과 입력 인수 값이 포함되어야 합니다. CALL 문을 사용하여 저장 프로시저를 호출해야 합니다.

> **Note**
>
> CALL은 정규 쿼리의 일부일 수 없습니다.

구문

```
CALL sp_name ( [ argument ] [, ...] )
```

파라미터

sp_name

실행할 프로시저의 이름입니다.

인수

입력 인수의 값입니다. 이 파라미터도 함수 이름일 수 있습니다(예: `pg_last_query_id()`). 쿼리를 CALL 인수로 사용할 수 없습니다.

사용 노트

Amazon Redshift 저장 프로시저는 다음 설명에 따라 중첩 및 재귀 호출을 지원합니다. 또한 다음 설명에 따라 드라이버 지원이 최신인지 확인합니다.

주제

- 중첩 호출
- 드라이버 지원

중첩 호출

Amazon Redshift 저장 프로시저는 중첩 및 재귀 호출을 지원합니다. 허용되는 중첩 수준의 최대 수는 16개입니다. 중첩 호출은 비즈니스 로직을 더 작은 프로시저로 캡슐화할 수 있습니다. 그러면 이를 여러 호출자가 공유할 수 있습니다.

출력 파라미터가 있는 중첩 프로시저를 호출하면 내부 프로시저가 INOUT 인수를 정의해야 합니다. 이 경우 내부 프로시저는 상수가 아닌 변수로 전달됩니다. OUT 인수는 허용되지 않습니다. 내부 호출의 출력을 보관하는 데 변수가 필요하기 때문에 이 동작이 발생합니다.

내부 프로시저와 외부 프로시저 간의 관계는 SVL_STORED_PROC_CALL의 `from_sp_call` 열에 기록됩니다.

다음 예에서는 INOUT 인수를 통해 변수를 중첩 프로시저 호출로 전달하는 것을 보여 줍니다.

```
CREATE OR REPLACE PROCEDURE inner_proc(INOUT a int, b int, INOUT c int) LANGUAGE
 plpgsql
AS $$
BEGIN
  a := b * a;
  c := b * c;
END;
$$;

CREATE OR REPLACE PROCEDURE outer_proc(multiplier int) LANGUAGE plpgsql
AS $$
DECLARE
  x int := 3;
  y int := 4;
BEGIN
  DROP TABLE IF EXISTS test_tbl;
  CREATE TEMP TABLE test_tbl(a int, b varchar(256));
  CALL inner_proc(x, multiplier, y);
  insert into test_tbl values (x, y::varchar);
END;
$$;

CALL outer_proc(5);

SELECT * from test_tbl;
 a  | b
----+----
 15 | 20
(1 row)
```

드라이버 지원

Java Database Connectivity(JDBC) 및 Open Database Connectivity(ODBC) 드라이버를 Amazon Redshift 저장 프로시저를 지원하는 최신 버전으로 업그레이드하는 것이 좋습니다.

클라이언트 도구가 CALL 문을 통해 서버로 전달하는 드라이버 API 작업을 사용하는 경우 기존 드라이버를 사용할 수 있습니다. 출력 파라미터(있는 경우)는 한 행의 결과 세트로 반환됩니다.

Amazon Redshift JDBC 및 ODBC 드라이버의 최신 버전은 저장 프로시저 검색에 대해 메타데이터 지원을 제공합니다. 또한 사용자 지정 Java 애플리케이션에 대한 `CallableStatement` 지원도 제공합니다. 드라이버에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 SQL 클라이언트 도구를 사용하여 Amazon Redshift 클러스터에 연결 섹션을 참조하세요.

다음 예제에서는 저장 프로시저 호출에 JDBC 드라이버의 여러 API 작업을 사용하는 방법을 보여 줍니다.

```
void statement_example(Connection conn) throws SQLException {
  statement.execute("CALL sp_statement_example(1)");
}

void prepared_statement_example(Connection conn) throws SQLException {
  String sql = "CALL sp_prepared_statement_example(42, 84)";
  PreparedStatement pstmt = conn.prepareStatement(sql);
  pstmt.execute();
}

void callable_statement_example(Connection conn) throws SQLException {
  CallableStatement cstmt = conn.prepareCall("CALL sp_create_out_in(?,?)");
  cstmt.registerOutParameter(1, java.sql.Types.INTEGER);
  cstmt.setInt(2, 42);
  cstmt.executeQuery();
  Integer out_value = cstmt.getInt(1);
}
```

예시

다음 예제에서는 프로시저 이름 `test_spl`를 호출합니다.

```
call test_sp1(3,'book');
INFO:  Table "tmp_tbl" does not exist and will be skipped
INFO:  min_val = 3, f2 = book
```

다음 예제에서는 프로시저 이름 `test_spl2`를 호출합니다.

```
call test_sp2(2,'2019');

         f2          | column2
---------------------+---------
 2019+2019+2019+2019 | 2
(1 row)
```

CANCEL

현재 실행 중인 데이터베이스 쿼리를 취소합니다.

CANCEL 명령은 실행 중인 쿼리의 프로세스 ID 또는 세션 ID가 필요하고 쿼리가 취소되었는지 검증하기 위한 확인 메시지를 표시합니다.

필수 권한

CANCEL에 필요한 권한은 다음과 같습니다.

- 자신의 쿼리를 취소하는 슈퍼 사용자
- 사용자의 쿼리를 취소하는 슈퍼 사용자
- 사용자의 쿼리를 취소하는 CANCEL 권한이 있는 사용자
- 자신의 쿼리를 취소하는 사용자

구문

```
CANCEL process_id [ 'message' ]
```

파라미터

process_id

Amazon Redshift 클러스터에서 실행 중인 쿼리를 취소하려면 취소하려는 쿼리에 해당하는 STV_RECENTS에서 pid(프로세스 ID)를 사용합니다.

Amazon Redshift Serverless 작업 그룹에서 실행 중인 쿼리를 취소하려면 취소하려는 쿼리에 해당하는 SYS_QUERY_HISTORY에서 session_id(프로세스 ID)를 사용합니다.

'message'

쿼리 취소 완료 시 표시되는 선택적 확인 메시지입니다. 메시지를 지정하지 않을 경우 Amazon Redshift는 기본 메시지를 확인으로 표시합니다. 이 메시지는 작은따옴표로 묶어야 합니다.

사용 노트

쿼리 ID를 지정해서는 쿼리를 취소할 수 없습니다. 반드시 쿼리의 프로세스 ID(PID) 또는 세션 ID를 지정해야 합니다. 사용자가 현재 실행하고 있는 쿼리만 취소할 수 있습니다. 수퍼유저는 모든 쿼리를 취소할 수 있습니다.

여러 세션의 쿼리가 동일한 테이블에 대한 잠금을 보유하는 경우 PG_TERMINATE_BACKEND 함수를 사용하여 세션 중 하나를 종료할 수 있습니다. 그러면 종료된 세션에서 실행 중이던 트랜잭션이 모

든 잠금을 강제로 해제하여 트랜잭션을 롤백시킵니다. 현재 보유한 잠금을 보려면 STV_LOCKS 시스템 테이블을 쿼리합니다.

Amazon Redshift는 특정한 내부 이벤트 이후에 활성 세션을 다시 시작하고 새 PID를 할당할 수도 있습니다. PID가 변경된 경우 다음 오류 메시지가 나타날 수 있습니다.

```
Session <PID> does not exist. The session PID might have changed. Check the
 stl_restarted_sessions system table for details.
```

새 PID를 찾으려면 STL_RESTARTED_SESSIONS 시스템 테이블을 쿼리하고 oldpid 열에서 필터링하십시오.

```
select oldpid, newpid from stl_restarted_sessions where oldpid = 1234;
```

예시

Amazon Redshift 클러스터에서 현재 실행 중인 쿼리를 취소하려면 먼저 취소하려는 쿼리에 대한 프로세스 ID를 검색합니다. 현재 실행 중인 모든 쿼리의 프로세스 ID를 확인하려면 다음 명령을 입력하십시오.

```
select pid, starttime, duration,
trim(user_name) as user,
trim (query) as querytxt
from stv_recents
where status = 'Running';

pid |         starttime          | duration |   user   |    querytxt
-----+----------------------------+----------+----------+-----------------
802 | 2008-10-14 09:19:03.550885 |      132 | dwuser | select
venuename from venue where venuestate='FL', where venuecity not in
('Miami' , 'Orlando');
834 | 2008-10-14 08:33:49.473585 |  1250414 | dwuser | select *
from listing;
964 | 2008-10-14 08:30:43.290527 |   326179 | dwuser | select
sellerid from sales where qtysold in (8, 10);
```

쿼리 텍스트를 확인하여 어떤 프로세스 ID(PID)가 취소하려는 쿼리에 해당하는지 확인합니다.

다음 명령을 입력해 PID 802를 사용하여 해당 쿼리를 취소합니다.

```
cancel 802;
```

쿼리가 실행 중이던 세션에서 다음 메시지를 표시합니다.

```
ERROR:  Query (168) cancelled on user's request
```

여기에서 168은 쿼리 ID입니다(쿼리 취소에 사용되는 프로세스 ID가 아님).

또는 기본 메시지 대신 표시할 사용자 지정 확인 메시지를 지정할 수 있습니다. 사용자 지정 메시지를 지정하려면 CANCEL 명령의 끝에 메시지를 작은따옴표로 묶습니다.

```
cancel 802 'Long-running query';
```

쿼리가 실행 중이던 세션에서 다음 메시지를 표시합니다.

```
ERROR:  Long-running query
```

CLOSE

(선택 사항) 열려 있는 커서와 연결되어 사용 가능한 모든 리소스를 닫습니다. COMMIT, END 및 ROLLBACK은 커서를 자동으로 닫으므로, CLOSE 명령을 사용하여 커서를 명시적으로 닫을 필요가 없습니다.

자세한 내용은 DECLARE, FETCH 섹션을 참조하세요.

구문

```
CLOSE cursor
```

파라미터

cursor

닫을 커서의 이름입니다.

CLOSE 예

다음과 같은 명령은 커서를 닫고 커밋을 수행하여 트랜잭션을 종료합니다.

```
close movie_cursor;
commit;
```

COMMENT

데이터베이스 객체에 대한 설명을 생성하거나 변경합니다.

구문

```
COMMENT ON
{
TABLE object_name |
COLUMN object_name.column_name |
CONSTRAINT constraint_name ON table_name |
DATABASE object_name |
VIEW object_name
}
IS 'text' | NULL
```

파라미터

객체 이름

설명 대상이 되는 데이터베이스 객체의 이름입니다. 다음 객체에 설명을 추가할 수 있습니다.

- TABLE
- COLUMN(column_name도 취함).
- CONSTRAINT(constraint_name 및 table_name도 취함).
- 데이터베이스
- VIEW
- 스키마

IS 'text' | NULL

지정된 객체에 대해 추가하거나 바꿀 설명 텍스트입니다. text 문자열은 TEXT 데이터 형식입니다. 설명은 작은따옴표로 묶으십시오. 설명 텍스트를 제거하려면 값을 NULL로 설정합니다.

column_name

설명 대상이 되는 열의 이름입니다. COLUMN의 파라미터입니다. `object_name`에 지정된 테이블을 따릅니다.

constraint_name

설명 대상이 되는 제약 조건의 이름입니다. CONSTRAINT의 파라미터입니다.

table_name

제약 조건을 포함한 테이블의 이름입니다. CONSTRAINT의 파라미터입니다.

사용 노트

슈퍼유저 또는 데이터베이스 객체의 소유자만 설명을 추가하거나 업데이트할 수 있습니다.

데이터베이스에 대한 설명은 현재 데이터베이스에만 적용될 수 있습니다. 다른 데이터베이스에 대해 설명하려는 경우 경고 메시지가 표시됩니다. 존재하지 않는 데이터베이스에 대한 설명을 하려고 할 때도 같은 경고가 표시됩니다.

외부 테이블, 외부 열 및 후기 바인딩 뷰의 열에 대한 댓글은 지원되지 않습니다.

예시

다음 예에서는 SALES 테이블에 새 열을 추가합니다.

```
COMMENT ON TABLE sales IS 'This table stores tickets sales data';
```

다음 예에서는 SALES 테이블에 설명을 표시합니다.

```
select obj_description('public.sales'::regclass);

obj_description
-------------------------------------
This table stores tickets sales data
```

다음 예에서는 SALES 테이블에서 설명을 제거합니다.

```
COMMENT ON TABLE sales IS NULL;
```

다음 예에서는 SALES 테이블의 EVENTID 열에 설명을 추가합니다.

```
COMMENT ON COLUMN sales.eventid IS 'Foreign-key reference to the EVENT table.';
```

다음 예에서는 SALES 테이블의 EVENTID 열(열 번호 5)에 설명을 표시합니다.

```
select col_description( 'public.sales'::regclass, 5::integer );
```

```
col_description
----------------------------------------
Foreign-key reference to the EVENT table.
```

다음 예에서는 설명문을 EVENT 테이블에 추가합니다.

```
comment on table event is 'Contains listings of individual events.';
```

설명을 보려면 PG_DESCRIPTION 시스템 테이블을 쿼리합니다. 다음 예에서는 EVENT 테이블에 대한 설명을 반환합니다.

```
select * from pg_catalog.pg_description
where objoid =
(select oid from pg_class where relname = 'event'
and relnamespace =
(select oid from pg_catalog.pg_namespace where nspname = 'public') );

objoid | classoid | objsubid | description
-------+----------+----------+----------------------------------------
116658 |     1259 |        0 | Contains listings of individual events.
```

COMMIT

데이터베이스에 현재 트랜잭션을 커밋합니다. 이 명령을 실행하면 트랜잭션에서의 데이터베이스 업데이트가 영구적인 것이 됩니다.

구문

```
COMMIT [ WORK | TRANSACTION ]
```

파라미터

Work

선택적 키워드입니다. 이 키워드는 저장 프로시저 내에서 지원되지 않습니다.

TRANSACTION

선택적 키워드입니다. WORK와 TRANSACTION은 동의어입니다. 둘 다 저장 프로시저 내에서 지원되지 않습니다.

저장 프로시저 내의 COMMIT 사용에 대한 자세한 내용은 트랜잭션 관리를 참조하십시오.

예시

다음 각 예에서는 데이터베이스에 현재 트랜잭션을 커밋합니다.

```
commit;
```

```
commit work;
```

```
commit transaction;
```

COPY

데이터 파일 또는 Amazon DynamoDB 테이블의 데이터를 테이블에 로드합니다. 이 파일은 Amazon Simple Storage Service(Amazon S3) 버킷과 Amazon EMR 클러스터 또는 SSH(Secure Shell) 연결을 사용하여 액세스하는 원격 호스트에서 찾을 수 있습니다.

> **Note**
>
> Amazon Redshift Spectrum 외부 테이블은 읽기 전용입니다. 외부 테이블로 복사할 수 없습니다.

COPY 명령은 입력 데이터를 테이블에 추가 행으로 추가합니다.

어떤 원본이든지 상관없이 단일 입력 행의 최대 크기는 4MB입니다.

주제

- 필수 권한
- COPY 구문
- 필수 파라미터
- 선택적 파라미터
- COPY 명령에 대한 사용 참고 사항 및 추가 리소스
- COPY 명령 예시
- COPY JOB

- COPY 파라미터 참조
- 사용 노트
- COPY 예

필수 권한

COPY 명령을 사용하려면 Amazon Redshift 테이블에 대한 INSERT 권한이 필요합니다.

COPY 구문

```
COPY table-name
[ column-list ]
FROM data_source
authorization
[ [ FORMAT ] [ AS ] data_format ]
[ parameter [ argument ] [, ... ] ]
```

COPY 작업은 테이블 이름과 데이터 원본, 그리고 데이터에 대한 액세스 권한 등 최소 3개 파라미터만 있으면 실행할 수 있습니다.

Amazon Redshift는 COPY 명령의 기능을 연장하여 다수의 데이터 원본에서 몇 가지 데이터 형식으로 데이터를 로드하거나, 데이터 로드를 위한 액세스 권한을 제어하거나, 데이터 변환 및 로드 작업을 관리하는 데도 사용 가능합니다.

이번 섹션에서는 필요한 COPY 명령 파라미터를 제시하면서 함수에 따라 옵션 파라미터를 분류합니다. 각 파라미터와 함께 다양한 옵션 조합의 사용 방법에 대해서도 설명합니다. 알파벳 순서로 정렬된 파라미터 목록을 사용해 파라미터 설명으로 직접 이동할 수 있습니다.

필수 파라미터

COPY 명령에는 세 가지 파라미터가 필요합니다.

- Table Name
- Data Source
- Authorization

COPY 명령을 가장 단순하게 사용하면 다음 형식과 같습니다.

```
COPY table-name
```

```
FROM data-source
authorization;
```

다음은 CATDEMO라는 이름의 테이블을 생성한 다음 category_pipe.txt라는 이름의 Amazon S3 데이터 파일에서 샘플 데이터를 테이블에 로드하는 예입니다.

```
create table catdemo(catid smallint, catgroup varchar(10), catname varchar(10), catdesc
 varchar(50));
```

다음 예에서 COPY 명령의 데이터 원본은 이름이 redshift-downloads인 Amazon S3 버킷의 tickit 폴더에 저장된 category_pipe.txt라는 데이터 파일입니다. COPY 명령은 AWS Identity and Access Management(IAM) 역할을 통해 Amazon S3 버킷에 액세스할 수 있는 권한이 부여됩니다. 클러스터에 연결되어 있는 기존 IAM 역할에 Amazon S3에 액세스할 수 있는 권한이 부여되어 있으면 다음 COPY 명령에서 해당 역할의 Amazon 리소스 이름(ARN)으로 치환한 후 명령을 실행할 수 있습니다.

```
copy catdemo
from 's3://redshift-downloads/tickit/category_pipe.txt'
iam_role 'arn:aws:iam::<aws-account-id>:role/<role-name>'
region 'us-east-1';
```

다른 AWS 리전에서 데이터를 로드하기 위한 지침을 포함하여 COPY 명령으로 샘플 데이터를 로드하는 방법에 대한 전체 지침은 Amazon Redshift 시작 안내서의 Amazon S3에서 샘플 데이터 로드를 참조하세요.

table-name

COPY 명령을 실행할 대상 테이블의 이름입니다. 이 테이블은 사전에 데이터베이스에 존재해야 하며, 임시 테이블일 수도 있고, 영구 테이블일 수도 있습니다. COPY 명령은 새로운 입력 데이터를 테이블의 기존 행에 추가합니다.

FROM data-source

대상 테이블에 로드할 원본 테이블의 위치입니다. 일부 데이터 소스로 매니페스트 파일을 지정할 수 있습니다.

가장 공통적으로 사용되는 데이터 리포지토리는 Amazon S3 버킷입니다. 또한 클러스터가 Amazon EMR 클러스터와 Amazon EC2 인스턴스, 또는 SSH 연결을 사용하여 액세스할 수 있는 원격 호스트에 위치한 데이터 파일에서 데이터를 로드하거나, 혹은 DynamoDB 테이블에서 직접 데이터를 로드할 수도 있습니다.

- Amazon S3에서 COPY
- Amazon EMR에서 COPY
- 원격 호스트(SSH)에서 COPY 지원
- Amazon DynamoDB에서 COPY

권한 부여

클러스터가 다른 AWS 리소스에 액세스하기 위한 인증 및 권한 부여에 사용할 방법을 나타내는 절입니다. COPY 명령을 실행하려면 Amazon S3, Amazon EMR, Amazon DynamoDB, Amazon EC2의 데이터를 포함해 다른 AWS 리소스의 데이터에도 액세스할 수 있도록 권한 부여가 필요합니다. 이러한 권한은 클러스터에 연결되어 있는 IAM 역할을 참조하거나, 혹은 IAM 사용자의 액세스 키 ID와 보안 액세스 키를 입력하면 부여할 수 있습니다.

- 권한 부여 파라미터
- 역할 기반 액세스 제어
- 키 기반 액세스 제어

선택적 파라미터

또는 COPY에서 필드 데이터를 대상 테이블의 열에 매핑하는 방식을 지정하고, COPY 명령에서 원본 데이터를 제대로 읽고 구문 분석하도록 원본 데이터 속성을 정의하고, 로드 프로세스 중 COPY 명령으로 수행하는 작업을 관리할 수 있습니다.

- 열 매핑 옵션
- 데이터 형식 파라미터
- 데이터 변환 파라미터
- 데이터 로드 작업

열 매핑

기본적으로 COPY 명령은 데이터 파일의 필드 순서와 동일하게 필드 값을 대상 테이블의 열에 삽입합니다. 기본 열 순서가 유효하지 않은 경우에는 열 목록을 지정하거나 JSONPath 표현식을 사용하여 원본 데이터 필드를 대상 열로 매핑할 수 있습니다.

- Column List
- JSONPaths File

데이터 형식 파라미터

데이터는 텍스트 파일에서 고정 폭, 문자 구분, 쉼표 구분 값(CSV) 또는 JSON 형식으로, 혹은 Avro 파일에서 로드할 수 있습니다.

COPY 명령에서는 기본적으로 원본 데이터가 문자로 구분된 UTF-8 텍스트 파일 형식입니다. 기본 구분자는 파이프 문자(|)입니다. 원본 데이터가 다른 형식인 경우에는 다른 파라미터를 사용하여 데이터 형식을 지정합니다.

- FORMAT
- CSV
- DELIMITER
- FIXEDWIDTH
- SHAPEFILE
- AVRO
- JSON
- ENCRYPTED
- BZIP2
- GZIP
- LZOP
- PARQUET
- ORC
- ZSTD

데이터 변환 파라미터

COPY 명령은 테이블에 데이터를 로드할 때 원본 데이터의 문자열을 묵시적으로 대상 열의 데이터 형식으로 변환합니다. 기본 동작과 다른 변환을 지정해야 하거나, 혹은 기본 변환이 오류를 일으킬 때는 다음 파라미터를 지정하여 데이터 변환을 관리할 수 있습니다.

- ACCEPTANYDATE
- ACCEPTINVCHARS
- BLANKSASNULL
- DATEFORMAT
- EMPTYASNULL

- ENCODING
- ESCAPE
- EXPLICIT_IDS
- FILLRECORD
- IGNOREBLANKLINES
- IGNOREHEADER
- NULL AS
- REMOVEQUOTES
- ROUNDEC
- TIMEFORMAT
- TRIMBLANKS
- TRUNCATECOLUMNS

데이터 로드 작업

문제 해결을 위해, 혹은 로드 시간을 줄일 목적으로 다음 파라미터를 지정하여 로드 작업의 기본 동작을 관리합니다.

- COMPROWS
- COMPUPDATE
- IGNOREALLERRORS
- MAXERROR
- NOLOAD
- STATUPDATE

COPY 명령에 대한 사용 참고 사항 및 추가 리소스

COPY 명령의 사용 방법에 대한 자세한 내용은 아래 주제를 참조하십시오.

- 사용 노트
- 튜토리얼: Amazon S3에서 데이터 로드
- 데이터 로드에 대한 Amazon Redshift 모범 사례
- COPY 명령으로 테이블 로드

- Amazon S3에서 데이터 로드
- Amazon EMR에서 데이터 로드
- 원격 호스트에서 데이터 로드
- Amazon DynamoDB 테이블에서 데이터 로드
- 데이터 로드 문제 해결

COPY 명령 예시

다양한 소스에서 다양한 형식으로 다양한 COPY 옵션을 사용하여 COPY를 실행하는 방법을 보여주는 추가 예시는 COPY 예 섹션을 참조하세요.

COPY JOB

이 명령 사용에 대한 자세한 내용은 Amazon S3 버킷에서 자동으로 파일을 복사하기 위해 S3 이벤트 통합 만들기 섹션을 참조하세요.

데이터를 테이블로 로드하는 COPY 명령을 관리합니다. COPY JOB 명령은 COPY 명령의 확장이며 Amazon S3 버킷에서 데이터 로드를 자동화합니다. COPY 작업을 생성하면 Amazon Redshift는 지정된 경로에 새 Amazon S3 파일이 생성되는 시기를 감지한 다음 사용자 개입 없이 자동으로 로드합니다. 원래 COPY 명령에 사용된 것과 동일한 파라미터가 데이터를 로드할 때 사용됩니다. Amazon Redshift는 파일 이름을 기반으로 로드된 파일을 추적하여 한 번만 로드되었는지 확인합니다.

> **Note**
>
> 사용법, 파라미 및 권한을 비롯한 COPY 명령에 대한 자세한 내용은 COPY 단원을 참조하세요.

필수 권한

COPY JOB의 COPY 명령을 실행하려면 로드 중인 테이블의 INSERT 권한이 있어야 합니다.

COPY 명령으로 지정된 IAM 역할에는 로드할 데이터에 액세스할 수 있는 권한이 있어야 합니다. 자세한 내용은 COPY, UNLOAD 및 CREATE LIBRARY 작업을 위한 IAM 권한 단원을 참조하십시오.

구문

COPY 작업을 생성합니다. COPY 명령의 파라미터는 COPY 작업과 함께 저장됩니다.

트랜잭션 블록의 범위 내에서 COPY JOB CREATE를 실행할 수 없습니다.

```
COPY copy-command JOB CREATE job-name
[AUTO ON | OFF]
```

COPY 작업의 구성을 변경합니다.

```
COPY JOB ALTER job-name
[AUTO ON | OFF]
```

COPY 작업을 실행합니다. 저장된 COPY 명령 파라미터가 사용됩니다.

```
COPY JOB RUN job-name
```

모든 COPY 작업을 나열합니다.

```
COPY JOB LIST
```

COPY 작업의 세부 정보를 표시합니다.

```
COPY JOB SHOW job-name
```

COPY 작업을 삭제합니다.

트랜잭션 블록 범위 내에서 COPY JOB DROP을 실행할 수 없습니다.

```
COPY JOB DROP job-name
```

파라미터

copy-command

Amazon S3에서 Amazon Redshift로 데이터를 로드하는 COPY 명령입니다. 이 절에는 데이터를 로드할 때 사용되는 Amazon S3 버킷, 대상 테이블, IAM 역할 및 기타 파라미터를 정의하는 COPY 파라미터가 포함되어 있습니다. Amazon S3 데이터 로드에 대한 모든 COPY 명령 파라미터는 다음을 제외하고 지원됩니다.

- COPY JOB은 COPY 명령이 가리키는 폴더에 있는 기존 파일을 수집하지 않습니다. COPY JOB 생성 타임스탬프 이후에 생성된 파일만 수집합니다.

- MAXERROR 또는 IGNOREALLERRORS 옵션과 함께 COPY 명령을 지정할 수 없습니다.
- 매니페스트 파일을 지정할 수 없습니다. COPY JOB에는 새로 생성된 파일을 모니터링하기 위해 지정된 Amazon S3 위치가 필요합니다.
- 액세스 및 비밀 키와 같은 인증 유형으로 COPY 명령을 지정할 수 없습니다. 권한 부여에 IAM_ROLE 파라미터를 사용하는 COPY 명령만 지원됩니다. 자세한 내용은 권한 부여 파라미터 단원을 참조하십시오.
- COPY JOB은 클러스터와 연결된 기본 IAM 역할을 지원하지 않습니다. COPY 명령에서 IAM_ROLE을 지정해야 합니다.

자세한 내용은 Amazon S3에서 COPY 단원을 참조하십시오.

job-name

COPY 작업을 참조하는 데 사용되는 작업의 이름입니다.

[AUTO ON | OFF]

Amazon S3 데이터가 Amazon Redshift 테이블에 자동으로 로드되는지 여부를 나타내는 절입니다.

- ON일 때 Amazon Redshift는 새로 생성된 파일의 소스 Amazon S3 경로를 모니터링하고, 파일이 발견되면 작업 정의의 COPY 파라미터와 함께 COPY 명령이 실행됩니다. 이 값이 기본값입니다.
- OFF일 때 Amazon Redshift는 COPY JOB을 자동으로 실행하지 않습니다.

사용 노트

COPY 명령의 옵션은 런타임까지 유효성이 검사되지 않습니다. 예를 들어 잘못된 IAM_ROLE 또는 Amazon S3 데이터 소스로 인해 COPY JOB이 시작될 때 런타임 오류가 발생합니다.

클러스터가 일시 중지된 경우 COPY JOBS는 실행되지 않습니다.

로드된 COPY 명령 파일을 쿼리하고 오류를 로드하려면 STL_LOAD_COMMITS, STL_LOAD_ERRORS, STL_LOADERROR_DETAIL 단원을 참조하세요. 자세한 내용은 데이터가 올바로 로드되었는지 확인 단원을 참조하십시오.

예시

다음 예에서는 Amazon S3 버킷에서 데이터를 로드하기 위해 COPY 작업을 생성하는 방법을 보여줍니다.

```
COPY public.target_table
FROM 's3://amzn-s3-demo-bucket/staging-folder'
```

```
IAM_ROLE 'arn:aws:iam::123456789012:role/MyLoadRoleName'
JOB CREATE my_copy_job_name
AUTO ON;
```

COPY 파라미터 참조

COPY에는 다양한 상황에서 사용할 수 있는 많은 파라미터가 있습니다. 그러나 각 상황에서 모든 파라미터가 지원되는 것은 아닙니다. 예를 들어 ORC 또는 PARQUET 파일에서 로드하는 경우 지원되는 파라미터 수가 제한되어 있습니다. 자세한 내용은 열 기반 데이터 형식에서 COPY 명령 단원을 참조하십시오.

주제

- 데이터 소스
- 권한 부여 파라미터
- 열 매핑 옵션
- 데이터 형식 파라미터
- 파일 압축 파라미터
- 데이터 변환 파라미터
- 데이터 로드 작업
- 알파벳 순서의 파라미터 목록

데이터 소스

클러스터가 Amazon S3 버킷과 Amazon EMR 클러스터 또는 SSH 연결을 사용하여 액세스할 수 있는 원격 호스트에 위치한 텍스트 파일에서 데이터를 로드할 수 있습니다. 또한 DynamoDB 테이블에서도 데이터를 직접 로드할 수 있습니다.

어떤 원본이든지 상관없이 단일 입력 행의 최대 크기는 4MB입니다.

테이블에서 Amazon S3의 파일로 데이터를 내보내려면 UNLOAD 명령을 사용합니다.

주제

- Amazon S3에서 COPY
- Amazon EMR에서 COPY
- 원격 호스트(SSH)에서 COPY 지원
- Amazon DynamoDB에서 COPY

Amazon S3에서 COPY

1개 이상의 S3 버킷에 위치한 파일에서 데이터를 로드하려면 FROM 절을 사용하여 COPY가 Amazon S3의 파일을 찾는 방법을 지정합니다. 데이터 파일의 객체 경로를 FROM 절의 일부로 제공하거나 Amazon S3 객체 경로 목록이 포함된 매니페스트 파일의 위치를 제공할 수 있습니다. Amazon S3에서 COPY는 HTTPS 연결을 사용합니다. S3 IP 범위가 허용 목록에 추가되었는지 확인합니다. 필요한 S3 IP 범위에 대한 자세한 내용은 네트워크 격리를 참조하세요.

> **Important**
>
> 데이터 파일이 위치한 Amazon S3 버킷이 클러스터와 동일한 AWS 리전에 속하지 않을 때는 REGION 파라미터를 사용하여 데이터가 위치한 리전을 지정해야 합니다.

주제

- 구문
- 예시
- 선택적 파라미터
- 지원되지 않는 파라미터

구문

```
FROM { 's3://objectpath' | 's3://manifest_file' }
authorization
| MANIFEST
| ENCRYPTED
| REGION [AS] 'aws-region'
| optional-parameters
```

예시

다음은 객체 경로를 사용하여 Amazon S3에서 데이터를 로드하는 예입니다.

```
copy customer
from 's3://amzn-s3-demo-bucket/customer'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

다음은 매니페스트 파일을 사용하여 Amazon S3에서 데이터를 로드하는 예입니다.

```
copy customer
from 's3://amzn-s3-demo-bucket/cust.manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest;
```

파라미터

FROM

로드할 데이터 원본입니다. Amazon S3 파일의 인코딩에 대한 자세한 내용은 데이터 변환 파라미터 섹션을 참조하세요.

's3://copy_from_s3_objectpath'

데이터가 포함된 Amazon S3 객체의 경로를 지정합니다(예: `'s3://amzn-s3-demo-bucket/custdata.txt'`). s3://copy_from_s3_objectpath 파라미터는 단일 파일, 또는 동일한 키 접두사를 갖는 객체 및 폴더 집합을 참조할 수 있습니다. 예를 들어 `custdata.txt`는 `custdata.txt`,`custdata.txt.1`, `custdata.txt.2`, `custdata.txt.bak` 등 다수의 물리적 파일을 참조하는 키 접두사입니다. 키 접두사는 다수의 폴더도 참조할 수 있습니다. 예를 들어 `'s3://amzn-s3-demo-bucket/custfolder'`는 `custfolder`, `custfolder_1`, `custfolder_2` 폴더 등을 참조합니다. 키 접두사가 다수의 폴더를 참조하는 경우 해당 폴더의 모든 파일이 로드됩니다. 키 접두사가 폴더는 물론 파일과도 일치하는 경우(예: `custfolder.log`) COPY가 파일도 로드하려고 합니다. 이처럼 키 접두사로 인해 COPY 명령이 불필요한 파일까지 로드하려고 할 때는 매니페스트 파일을 사용하십시오. 자세한 내용은 다음 자료를 참조하십시오.copy_from_s3_manifest_file

> **Important**
>
> 데이터 파일이 위치한 S3 버킷이 클러스터와 동일한 AWS 리전에 속하지 않을 때는 REGION 파라미터를 사용하여 데이터가 위치한 리전을 지정해야 합니다.

자세한 내용은 Amazon S3에서 데이터 로드 단원을 참조하십시오.

's3://copy_from_s3_manifest_file'

로드할 데이터 파일을 나열하는 매니페스트 파일에 Amazon S3 객체 키를 지정합니다. 's3://copy_from_s3_manifest_file' 인수는 단일 파일을 명시적으로 참조해야 합니다(예: `'s3://amzn-s3-demo-bucket/manifest.txt'`). 키 접두사를 참조할 수는 없습니다.

매니페스트란 Amazon S3에서 로드할 개별 파일의 URL을 JSON 형식으로 나열한 텍스트 파일을 말합니다. URL에는 파일의 버킷 이름과 전체 객체 경로가 포함됩니다. 매니페스트에서 지정되는 파일이 서로 다른 버킷에 있을 수 있지만 모든 버킷은 Amazon Redshift 클러스터와 동일한 AWS 리전에 속해야 합니다. 파일이 두 번 나열되면 마찬가지로 두 번 로드됩니다. 다음은 3개의 파일을 로드하는 매니페스트의 JSON 형식을 나타낸 예입니다.

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket1/custdata.1","mandatory":true},
    {"url":"s3://amzn-s3-demo-bucket1/custdata.2","mandatory":true},
    {"url":"s3://amzn-s3-demo-bucket2/custdata.1","mandatory":false}
  ]
}
```

큰따옴표가 필요하며, 기울어진 따옴표나 "스마트" 따옴표가 아닌 단순한 따옴표(0x22)여야 합니다. 매니페스트의 각 항목은 옵션으로 `mandatory` 플래그를 추가할 수 있습니다. `mandatory`가 `true`로 설정된 경우 해당 항목에서 파일을 찾지 못하면 COPY 명령이 종료되고, 그렇지 않으면 COPY 명령이 계속 됩니다. `mandatory`의 기본값은 `false`입니다.

ORC 또는 Parquet 형식의 데이터 파일에서 불러올 경우 다음 예에 나와 있는 것처럼 `meta` 필드가 필요합니다.

```
{
   "entries":[
      {
         "url":"s3://amzn-s3-demo-bucket1/orc/2013-10-04-custdata",
         "mandatory":true,
         "meta":{
            "content_length":99
         }
      },
      {
         "url":"s3://amzn-s3-demo-bucket2/orc/2013-10-05-custdata",
         "mandatory":true,
         "meta":{
            "content_length":99
         }
      }
   ]
}
```

매니페스트 파일에서도 ENCRYPTED, GZIP, LZOP, BZIP2 또는 ZSTD 옵션이 지정되지만 암호화하거나 압축해서는 안 됩니다. 지정된 매니페스트 파일을 찾지 못하거나, 혹은 매니페스트 파일의 형식이 잘못되면 COPY 명령이 오류를 반환합니다.

매니페스트 파일을 사용할 때는 COPY 명령에서 MANIFEST 파라미터를 지정해야 합니다. MANIFEST 파라미터를 지정하지 않으면 COPY가 FROM에서 지정한 파일이 데이터 파일인 것으로 간주합니다.

자세한 내용은 Amazon S3에서 데이터 로드 단원을 참조하십시오.

권한 부여

COPY 명령을 실행하려면 Amazon S3, Amazon EMR, Amazon DynamoDB, Amazon EC2의 데이터를 포함해 다른 AWS 리소스의 데이터에도 액세스할 수 있도록 권한 부여가 필요합니다. 권한은 클러스터에 연결되는 AWS Identity and Access Management(IAM) 역할을 참조하거나(역할 기반 액세스 제어) 사용자의 액세스 자격 증명을 입력하면(키 기반 액세스 제어) 부여할 수 있습니다. 보안과 유연성을 높이려면 IAM 역할 기반 액세스 제어를 권장합니다. 자세한 내용은 권한 부여 파라미터 단원을 참조하십시오.

MANIFEST

Amazon S3에서 로드할 데이터 파일을 식별할 때 매니페스트를 사용하도록 지정합니다. MANIFEST 파라미터를 사용하는 경우에는 COPY 명령이 's3://copy_from_s3_manifest_file'에서 참조하는 매니페스트 파일에 나열된 파일에서 데이터를 로드합니다. 매니페스트 파일을 찾지 못하거나, 혹은 잘못된 형식일 때는 COPY가 중단됩니다. 자세한 내용은 매니페스트를 사용하여 데이터 파일 지정 단원을 참조하십시오.

Encrypted

고객 관리형 키를 사용하는 클라이언트 측 암호화로 Amazon S3의 입력 파일을 암호화하도록 지정하는 절입니다. 자세한 내용은 Amazon S3에서 암호화된 데이터 파일 로드 단원을 참조하십시오. 입력 파일이 Amazon S3 서버 측 암호화(SSE-KMS 또는 SSE-S3)로 암호화되어 있는 경우에는 ENCRYPTED를 지정하지 마십시오. COPY가 자동으로 서버 측 암호화 파일을 읽습니다.

ENCRYPTED 파라미터를 지정하는 경우에는 MASTER_SYMMETRIC_KEY 파라미터도 지정하거나, 혹은 **master_symmetric_key** 값을 CREDENTIALS 문자열에 추가해야 합니다.

암호화된 파일이 압축 형식일 때는 GZIP, LZOP, BZIP2 또는 ZSTD 파라미터를 추가하십시오.

ENCRYPTED 옵션을 지정하더라도 매니페스트 파일과 JSONPaths 파일을 암호화해서는 안 됩니다.

MASTER_SYMMETRIC_KEY 'root_key'

Amazon S3에서 데이터 파일을 암호화할 때 사용되는 루트 대칭 키입니다. MASTER_SYMMETRIC_KEY를 지정하면 ENCRYPTED 파라미터도 지정해야 합니다. MASTER_SYMMETRIC_KEY를 CREDENTIALS 파라미터와 함께 사용할 수는 없습니다. 자세한 내용은 Amazon S3에서 암호화된 데이터 파일 로드 단원을 참조하십시오.

암호화된 파일이 압축 형식일 때는 GZIP, LZOP, BZIP2 또는 ZSTD 파라미터를 추가하십시오.

REGION [AS] 'aws-region'

원본 데이터가 위치한 AWS 리전을 지정합니다. 데이터가 위치한 AWS 리소스가 Amazon Redshift 클러스터와 동일한 리전에 속하지 않을 때는 Amazon S3 버킷 또는 DynamoDB 테이블에서 COPY를 실행하려면 REGION이 필요합니다.

aws_region의 값은 Amazon Redshift 리전 및 엔드포인트 표에 나와 있는 리전과 일치해야 합니다.

REGION 파라미터를 지정하면 매니페스트 파일 또는 여러 Amazon S3 버킷을 포함한 모든 리소스가 지정한 리전에 위치해야 합니다.

> **Note**
>
> 리전 간 데이터를 전송하면 데이터가 있는 Amazon S3 버킷 또는 DynamoDB 테이블을 대상으로 추가 요금이 발생합니다. 요금에 대한 자세한 내용은 Amazon S3 요금 페이지의 Amazon S3에서 다른 AWS 리전으로 데이터 전송과 Amazon DynamoDB 요금 페이지의 데이터 전송을 참조하세요.

기본적으로 COPY의 경우 데이터가 Amazon Redshift 클러스터와 동일한 리전에 위치한다고 가정합니다.

선택적 파라미터

Amazon S3에서 COPY를 지원할 때는 다음 파라미터를 옵션으로 지정할 수 있습니다.

- 열 매핑 옵션
- 데이터 형식 파라미터
- 데이터 변환 파라미터
- 데이터 로드 작업

지원되지 않는 파라미터

Amazon S3에서 COPY를 지원할 때는 다음 파라미터를 사용할 수 없습니다.

- SSH
- READRATIO

Amazon EMR에서 COPY

COPY 명령을 사용하면 클러스터의 Hadoop 분산 파일 시스템(HDFS)에 고정 폭 파일, 문자로 구분된 파일, CSV 파일, JSON 파일 또는 Avro 파일 형식으로 텍스트 파일을 쓰도록 구성된 Amazon EMR 클러스터에서 병렬로 데이터를 로드할 수 있습니다.

주제

구문

```
FROM 'emr://emr_cluster_id/hdfs_filepath'
authorization
[ optional_parameters ]
```

예제

다음은 Amazon EMR 클러스터에서 데이터를 로드하는 예입니다.

```
copy sales
from 'emr://j-SAMPLE2B500FC/myoutput/part-*'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

파라미터

FROM

로드할 데이터 원본입니다.

'emr://emr_cluster_id/hdfs_file_path'

COPY 명령에서 데이터 파일을 참조하는 Amazon EMR 클러스터 및 HDFS 파일 경로의 고유 식별자입니다. HDFS 데이터 파일 이름에는 와일드카드 문자인 별표(*)와 물음표(?)가 포함되어서는 안 됩니다.

> **Note**
>
> Amazon EMR 클러스터는 COPY 작업이 완료될 때까지 계속 실행되어야 합니다. COPY 작업을 마치기 전에 HDFS 데이터 파일이 하나라도 변경되거나 삭제되면 예상하지 못한 결과가 나오거나 COPY 작업이 중단될 수 있습니다.

와일드카드 문자인 별표(*)와 물음표(?)를 hdfs_file_path 인수에 사용하여 다수의 파일을 로드하도록 지정할 수 있습니다. 예를 들어 `'emr://j-SAMPLE2B500FC/myoutput/part*'`는 파일 `part-0000`, `part-0001` 등을 식별합니다. 와일드카드 문자가 없는 파일 경로는 문자열 리터럴로 처리됩니다. 폴더 이름만 지정하면 COPY가 폴더의 모든 파일을 로드하려고 합니다.

> **Important**
>
> 와일드카드 문자를 사용하거나 폴더 이름만 사용하는 경우에는 불필요한 파일이 로드되지 않는지 확인하십시오. 예를 들어 일부 프로세스에서는 로그 파일이 출력 폴더로 로드되는 경우도 있습니다.

자세한 내용은 Amazon EMR에서 데이터 로드 단원을 참조하십시오.

권한 부여

COPY 명령을 실행하려면 Amazon S3, Amazon EMR, Amazon DynamoDB, Amazon EC2의 데이터를 포함해 다른 AWS 리소스의 데이터에도 액세스할 수 있도록 권한 부여가 필요합니다. 권한은 클러스터에 연결되는 AWS Identity and Access Management(IAM) 역할을 참조하거나(역할 기반 액세스 제어) 사용자의 액세스 자격 증명을 입력하면(키 기반 액세스 제어) 부여할 수 있습니다. 보안과 유연성을 높이려면 IAM 역할 기반 액세스 제어를 권장합니다. 자세한 내용은 권한 부여 파라미터 단원을 참조하십시오.

지원되는 파라미터

Amazon EMR에서 COPY를 지원할 때는 다음 파라미터를 옵션으로 지정할 수 있습니다.

- 열 매핑 옵션
- 데이터 형식 파라미터
- 데이터 변환 파라미터
- 데이터 로드 작업

지원되지 않는 파라미터

Amazon EMR에서 COPY를 지원할 때는 다음 파라미터를 사용할 수 없습니다.

- Encrypted
- MANIFEST
- REGION
- READRATIO
- SSH

원격 호스트(SSH)에서 COPY 지원

COPY 명령을 사용하면 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스나 다른 컴퓨터 같은 하나 이상의 원격 호스트에서 병렬로 파일을 로드할 수 있습니다. COPY는 SSH(Secure Shell)을 사용하여 원격 호스트에 연결하고 원격 호스트에서 명령을 실행해 텍스트 출력을 생성합니다. 원격 호스트는 EC2 Linux 인스턴스이거나 SSH 연결을 허용하도록 구성된 다른 Unix 또는 Linux 컴퓨터일 수 있습니다. Amazon Redshift는 여러 호스트에 연결할 수 있으며 각 호스트에 대해 여러 SSH 연결을 열 수 있습니다. Amazon Redshift는 각 연결을 통해 고유한 명령을 전송하여 호스트의 표준 출력에 대한 텍스트 출력을 생성합니다. 그러면 Amazon Redshift가 텍스트 파일을 읽을 때 텍스트 출력을 읽습니다.

FROM 절은 매니페스트 파일에 Amazon S3 객체 키를 지정할 때 사용합니다. 매니페스트 파일은 COPY가 SSH 연결을 열어 원격 명령을 실행할 때 사용할 정보를 제공합니다.

주제

- 구문
- 예시
- 파라미터
- 선택적 파라미터
- 지원되지 않는 파라미터

> **Important**
>
> 매니페스트 파일이 위치한 S3 버킷이 클러스터와 동일한 AWS 리전에 속하지 않을 때는 REGION 파라미터를 사용하여 버킷이 위치한 리전을 지정해야 합니다.

구문

```
FROM 's3://'ssh_manifest_file' }
authorization
SSH
| optional-parameters
```

예시

다음은 매니페스트 파일을 사용하여 원격 호스트에서 SSH 연결을 통해 데이터를 로드하는 예입니다.

```
copy sales
from 's3://amzn-s3-demo-bucket/ssh_manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
ssh;
```

파라미터

FROM

로드할 데이터 원본입니다.

's3://copy_from_ssh_manifest_file'

COPY 명령은 SSH를 사용하여 다수의 호스트에 연결할 뿐만 아니라 각 호스트마다 SSH 연결을 생성할 수 있습니다. 이렇게 생성된 각 호스트 연결을 통해 명령을 실행하고, 명령을 통해 출력되는 데이터를 병렬 방식으로 테이블에 로드합니다. s3://copy_from_ssh_manifest_file 인수는 매니페스트 파일에 Amazon S3 객체 키를 지정합니다. 매니페스트 파일은 COPY가 SSH 연결을 열어 원격 명령을 실행할 때 사용할 정보를 제공합니다.

s3://copy_from_ssh_manifest_file 인수는 명시적으로 단일 파일을 참조해야 합니다. 이때 파일은 키 접두사가 될 수 없습니다. 다음은 그 한 예입니다:

```
's3://amzn-s3-demo-bucket/ssh_manifest.txt'
```

매니페스트 파일은 Amazon Redshift가 호스트에 연결하는 데 사용하는 JSON 형식의 텍스트 파일입니다. 매니페스트 파일은 SSH 호스트 엔드포인트를 비롯해 호스트에서 데이터를 Amazon Redshift에 반환할 때 실행하는 명령을 지정합니다. 필요한 경우, 호스트 퍼블릭 키, 로그인 사용자 이름 및 각 항목의 필수 플래그를 포함시킬 수 있습니다. 다음은 SSH 연결 2개를 생성하는 매니페스트 파일 예입니다.

```
{
    "entries": [
     {"endpoint":"<ssh_endpoint_or_IP>",
           "command": "<remote_command>",
           "mandatory":true,
           "publickey": "<public_key>",
           "username": "<host_user_name>"},
     {"endpoint":"<ssh_endpoint_or_IP>",
           "command": "<remote_command>",
           "mandatory":true,
           "publickey": "<public_key>",
           "username": "<host_user_name>"}
     ]
}
```

매니페스트 파일에는 SSH 연결마다 "entries" 구문이 하나씩 포함됩니다. 단일 호스트에 다중 연결을, 혹은 다수의 호스트에 다중 연결을 생성할 수 있습니다. 표시된 대로 필드 이름과 값 모두에 큰따옴표가 필요하며, 기울어진 따옴표나 "스마트" 따옴표가 아닌 단순한 따옴표(0x22)여야 합니다. 큰따옴표가 필요하지 않은 유일한 값은 "mandatory" 필드의 부울 값인 true 또는 false입니다.

다음 표에서는 매니페스트 파일의 속성을 설명합니다.

엔드포인트

호스트의 URL 주소 또는 IP 주소(예: "ec2-111-222-333.compute-1.amazonaws.com" 또는 "198.51.100.0").

명령

호스트에서 gzip, lzop, bzip2 또는 zstd 형식으로 텍스트 출력 또는 이진수 출력을 생성할 때 실행하는 명령입니다. 명령은 사용자 "host_user_name"이 실행 권한을 갖고 있는 어떤 명령도 될 수 있습니다. 명령은 파일 인쇄처럼 간단할 수도 있고 데이터베이스 쿼리나 스크립트 시작이 될 수도 있습니다. 출력(텍스트 파일, gzip 이진 파일, lzop 이진 파일 또는 bzip2 이진 파일)은 Amazon Redshift COPY 명령이 수집할 수 있는 형식이어야 합니다. 자세한 내용은 입력 데이터 준비 단원을 참조하십시오.

publickey

(옵션) 호스트의 퍼블릭 키입니다. 퍼블릭 키를 입력하면 Amazon Redshift가 호스트를 식별하는 데 이 키를 사용합니다. 퍼블릭 키를 입력하지 않으면 Amazon Redshift가 호스트를 식별하지 않습니다. 예를 들어 원격 호스트의 퍼블릭 키가 `ssh-rsa AbcCbaxxx…Example root@amazon.com`인 경우에는 publickey 필드에 `"AbcCbaxxx…Example"` 텍스트를 입력하십시오.

mandatory

(옵션) 연결 시도가 실패할 경우 COPY 명령의 중단 여부를 나타내는 절입니다. 기본값은 `false`입니다. Amazon Redshift가 하나 이상 연결에 성공하지 못하면 COPY 명령이 중단됩니다.

사용자 이름

(옵션) 호스트 시스템에 로그온하고 원격 명령을 실행하기 위해 사용할 사용자 이름입니다. 사용자 로그인 이름은 권한이 부여된 호스트 키 파일에 Amazon Redshift 클러스터의 퍼블릭 키를 추가할 때 사용한 로그인 이름과 동일해야 합니다. 기본 위치는 `redshift`입니다.

매니페스트 파일 생성에 대한 자세한 내용은 데이터 로드 프로세스 섹션을 참조하세요.

원격 호스트에서 COPY를 지원할 때는 COPY 명령에서 SSH 파라미터를 지정해야 합니다. SSH 파라미터를 지정하지 않으면 COPY가 FROM에서 지정한 파일이 데이터 파일인 것으로 간주하여 중단되고 맙니다.

자동 압축을 사용하는 경우 COPY 명령은 데이터 읽기 작업을 2회 실행합니다. 즉, 원격 명령을 2회 실행합니다. 첫 번째 읽기 작업은 압축 분석을 위한 데이터 샘플을 제공하기 위한 것이며, 두 번째 읽기 작업에서 실제로 데이터를 로드합니다. 원격 명령 2회 실행이 문제를 일으킬 수 있는 경우에는 자동 압축을 비활성화해야 합니다. 자동 압축을 비활성화하려면 COMPUPDATE 파라미터를 OFF로 설정하여 COPY 명령을 실행합니다. 자세한 내용은 자동 압축을 사용하여 테이블 로드 단원을 참조하십시오.

SSH에서 COPY를 사용하는 자세한 절차는 원격 호스트에서 데이터 로드 섹션을 참조하세요.

권한 부여

COPY 명령을 실행하려면 Amazon S3, Amazon EMR, Amazon DynamoDB, Amazon EC2의 데이터를 포함해 다른 AWS 리소스의 데이터에도 액세스할 수 있도록 권한 부여가 필요합니다. 권한은 클러스터에 연결되는 AWS Identity and Access Management(IAM) 역할을 참조하거나(역할 기반 액세스 제어) 사용자의 액세스 자격 증명을 입력하면(키 기반 액세스 제어) 부여할 수 있습니다. 보

안과 유연성을 높이려면 IAM 역할 기반 액세스 제어를 권장합니다. 자세한 내용은 권한 부여 파라미터 단원을 참조하십시오.

SSH

원격 호스트에서 데이터를 로드할 때 SSH 프로토콜을 사용하도록 지정하는 절입니다. SSH를 지정하는 경우에는 s3://copy_from_ssh_manifest_file 인수를 사용하여 매니페스트 파일도 지정해야 합니다.

> **Note**
>
> SSH를 사용하여 원격 VPC에서 프라이빗 IP 주소를 사용하는 호스트에서 복사하는 경우 VPC는 향상된 VPC 라우팅을 사용하도록 설정해야 합니다. Enhanced VPC Routing에 대한 자세한 내용은 Amazon Redshift Enhanced VPC Routing 섹션을 참조하세요.

선택적 파라미터

SSH 연결을 통해 COPY를 사용할 때는 다음 파라미터를 옵션으로 지정할 수 있습니다.

- 열 매핑 옵션
- 데이터 형식 파라미터
- 데이터 변환 파라미터
- 데이터 로드 작업

지원되지 않는 파라미터

SSH에서 COPY를 사용할 때는 다음 파라미터를 사용할 수 없습니다.

- Encrypted
- MANIFEST
- READRATIO

Amazon DynamoDB에서 COPY

기존 DynamoDB 테이블에서 데이터를 로드할 때는 FROM 절을 사용하여 DynamoDB 테이블 이름을 지정합니다.

주제

> **Important**
>
> DynamoDB 테이블이 Amazon Redshift 클러스터와 동일한 리전에 속하지 않는 경우에는 REGION 파라미터를 사용하여 데이터가 위치한 리전을 지정해야 합니다.

구문

```
FROM 'dynamodb://table-name'
authorization
READRATIO ratio
| REGION [AS] 'aws_region'
| optional-parameters
```

예시

다음은 DynamoDB 테이블에서 데이터를 로드하는 예입니다.

```
copy favoritemovies from 'dynamodb://ProductCatalog'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
readratio 50;
```

파라미터

FROM

로드할 데이터 원본입니다.

'dynamodb://table-name'

데이터가 들어 있는 DynamoDB 테이블의 이름입니다(예: `'dynamodb://ProductCatalog'`). DynamoDB 속성과 Amazon Redshift 열의 매핑 방식에 대한 자세한 내용은 Amazon DynamoDB 테이블에서 데이터 로드 섹션을 참조하세요.

DynamoDB 테이블 이름은 AWS 계정에서 고유하여 AWS 액세스 자격 증명으로 식별됩니다.

권한 부여

COPY 명령을 실행하려면 Amazon S3, Amazon EMR, DynamoDB, Amazon EC2의 데이터를 포함해 다른 AWS 리소스의 데이터에도 액세스할 수 있도록 권한 부여가 필요합니다. 권한은 클러스터에 연결되는 AWS Identity and Access Management(IAM) 역할을 참조하거나(역할 기반 액세스 제어) 사용자의 액세스 자격 증명을 입력하면(키 기반 액세스 제어) 부여할 수 있습니다. 보안과 유연성을 높이려면 IAM 역할 기반 액세스 제어를 권장합니다. 자세한 내용은 권한 부여 파라미터 단원을 참조하십시오.

READRATIO [AS] ratio

DynamoDB 테이블에서 데이터 로드에 사용할 프로비저닝 처리량의 비율입니다. READRATIO는 DynamoDB에서 COPY에 필요합니다. Amazon S3에서 COPY를 지원할 때는 사용할 수 없습니다. 이 비율은 사용되지 않는 평균 프로비저닝 처리량 미만의 값으로 설정하는 것이 좋습니다. 유효한 값은 1~200의 정수입니다.

> **Important**
>
> READRATIO를 100 이상으로 설정하면 Amazon Redshift가 DynamoDB 테이블의 프로비저닝된 처리량을 완전히 소비하여 COPY 세션 도중 동일한 테이블에 대한 동시 읽기 작업 성능을 심각하게 떨어뜨릴 수 있습니다. 쓰기 트래픽은 영향을 받지 않습니다. 100보다 높은 값일 때는 Amazon Redshift가 테이블의 프로비저닝 처리량을 충족하지 못하는 드문 경우의 문제를 해결할 수 있습니다. DynamoDB에서 Amazon Redshift로 데이터를 지속적으로 로드하는 경우에는 DynamoDB 테이블을 시계열로 구성하여 COPY 작업에서 실시간 트래픽을 분리하는 것이 좋습니다.

선택적 파라미터

Amazon DynamoDB에서 COPY를 지원할 때는 다음 파라미터를 옵션으로 지정할 수 있습니다.

- 열 매핑 옵션
- 지원되는 데이터 변환 파라미터는 다음과 같습니다.
  - ACCEPTANYDATE
  - BLANKSASNULL
  - DATEFORMAT
  - EMPTYASNULL

  - ROUNDEC
  - TIMEFORMAT
  - TRIMBLANKS
  - TRUNCATECOLUMNS
- 데이터 로드 작업

지원되지 않는 파라미터

DynamoDB에서 COPY를 지원할 때는 다음 파라미터를 사용할 수 없습니다.

- 모든 데이터 형식 파라미터
- ESCAPE
- FILLRECORD
- IGNOREBLANKLINES
- IGNOREHEADER
- NULL
- REMOVEQUOTES
- ACCEPTINVCHARS
- MANIFEST
- Encrypted

권한 부여 파라미터

COPY 명령을 실행하려면 Amazon S3, Amazon EMR, Amazon DynamoDB, Amazon EC2의 데이터를 포함해 다른 AWS 리소스의 데이터에도 액세스할 수 있도록 권한 부여가 필요합니다. 클러스터에 연결되어 있는 AWS Identity and Access Management(IAM) 역할을 참조하여 이러한 권한을 부여할 수 있습니다(역할 기반 액세스 제어). Amazon S3에서 로드 데이터를 암호화할 수 있습니다.

아래 주제에서는 더욱 자세한 내용과 함께 인증 옵션의 예를 살펴보겠습니다.

- COPY, UNLOAD 및 CREATE LIBRARY 작업을 위한 IAM 권한
- 역할 기반 액세스 제어
- 키 기반 액세스 제어

COPY 명령에서는 다음 중 한 가지를 사용하여 권한을 부여합니다.

- IAM_ROLE 파라미터
- ACCESS_KEY_ID and SECRET_ACCESS_KEY 파라미터
- CREDENTIALS 절

IAM_ROLE { default | 'arn:aws:iam::*<AWS ##-id>*:role/*<role-name>*' }

기본 키워드를 사용하여 COPY 명령이 실행될 때 Amazon Redshift에서 기본값으로 설정되고 클러스터와 연결된 IAM 역할을 사용하도록 합니다.

클러스터가 인증 및 권한 부여에 사용하는 IAM 역할의 Amazon 리소스 이름(ARN)을 사용합니다. IAM_ROLE을 지정하면 ACCESS_KEY_ID 및 SECRET_ACCESS_KEY, SESSION_TOKEN 또는 CREDENTIALS는 사용할 수 없습니다.

다음은 IAM_ROLE 파라미터에 대한 구문을 나타낸 것입니다.

```
IAM_ROLE { default | 'arn:aws:iam::<AWS ##-id>:role/<role-name>' }
```

자세한 내용은 역할 기반 액세스 제어 단원을 참조하십시오.

ACCESS_KEY_ID 'access-key-id ' SECRET_ACCESS_KEY 'secret-access-key'

이 권한 부여 방법은 권장되지 않습니다.

> **Note**
>
> 일반 텍스트로 입력되는 액세스 자격 증명보다는 IAM_ROLE 파라미터를 지정하는 역할 기반 인증의 사용을 강력하게 권장합니다. 자세한 내용은 역할 기반 액세스 제어 단원을 참조하십시오.

SESSION_TOKEN 'temporary-token'

임시 액세스 자격 증명으로 사용되는 세션 토큰입니다. SESSION_TOKEN을 지정할 때도 ACCESS_KEY_ID와 SECRET_ACCESS_KEY를 사용하여 임시 액세스 키 자격 증명을 입력해야 합니다. SESSION_TOKEN을 지정하면 IAM_ROLE 또는 CREDENTIALS는 사용할 수 없습니다. 자세한 내용은 IAM 사용 설명서에서 임시 보안 자격 증명 섹션을 참조하세요.

> **Note**
>
> 임시 보안 자격 증명을 생성하는 것보다는 역할 기반 인증의 사용을 강력하게 권장합니다. IAM 역할을 사용하여 권한을 부여하면 Amazon Redshift가 각 세션마다 임시 사용자 자격 증명을 자동으로 생성합니다. 자세한 내용은 역할 기반 액세스 제어 단원을 참조하십시오.

다음은 ACCESS_KEY_ID 및 SECRET_ACCESS_KEY 파라미터를 사용한 SESSION_TOKEN 파라미터의 구문 예입니다.

```
ACCESS_KEY_ID '<access-key-id>'
SECRET_ACCESS_KEY '<secret-access-key>'
SESSION_TOKEN '<temporary-token>';
```

SESSION_TOKEN을 지정하면 CREDENTIALS 또는 IAM_ROLE은 사용할 수 없습니다.

[WITH] CREDENTIALS [AS] 'credentials-args'

클러스터가 데이터 파일 또는 매니페스트 파일이 위치한 다른 AWS 리소스에 액세스할 때 사용할 방법을 나타내는 절입니다. CREDENTIALS 파라미터는 IAM_ROLE 또는 ACCESS_KEY_ID 및 SECRET_ACCESS_KEY와 함께 사용할 수 없습니다.

> **Note**
>
> 유연성을 높이려면 CREDENTIALS 파라미터 대신 IAM_ROLE 파라미터를 사용하는 것이 좋습니다.

옵션으로 ENCRYPTED 파라미터를 사용하는 경우에는 credentials-args 문자열 역시 암호화 키의 역할을 합니다.

credentials-args 문자열은 대소문자를 구분하며 공백이 포함되어서는 안됩니다.

키워드 WITH와 AS는 옵션이며 무시해도 좋습니다.

role-based access control 또는 key-based access control를 지정할 수 있습니다. 어떤 액세스 제어를 지정하든 간에 IAM 역할 또는 사용자는 지정한 AWS 리소스에 액세스할 수 있는 권한이 필요합니다. 자세한 내용은 COPY, UNLOAD 및 CREATE LIBRARY 작업을 위한 IAM 권한 단원을 참조하십시오.

> **Note**
>
> AWS 자격 증명을 안전하게 지키고 민감한 데이터를 보호하려면 역할 기반 액세스 제어의 사용을 강력하게 권장합니다.

역할 기반 액세스 제어를 지정하려면 credentials-args 문자열을 다음과 같은 형식으로 입력합니다.

```
'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
```

임시 토큰 자격 증명을 사용하려면 임시 액세스 키 ID와 임시 보안 액세스 키, 그리고 임시 토큰을 입력해야 합니다. credentials-args 문자열의 형식은 다음과 같습니다.

```
CREDENTIALS
'aws_access_key_id=<temporary-access-key-id>;aws_secret_access_key=<temporary-
secret-access-key>;token=<temporary-token>'
```

자세한 내용은 임시 보안 자격 증명 단원을 참조하십시오.

ENCRYPTED 파라미터를 사용하는 경우 credentials-args 문자열의 형식은 다음과 같습니다. 여기에서 *<root-key>*는 파일을 암호화할 때 사용한 루트 키의 값입니다.

```
CREDENTIALS
'<credentials-args>;master_symmetric_key=<root-key>'
```

예를 들어 다음은 암호화 키와 함께 역할 기반 액세스 제어를 사용하는 COPY 명령입니다.

```
copy customer from 's3://amzn-s3-demo-bucket/mydata'
credentials
'aws_iam_role=arn:aws:iam::<account-id>:role/<role-name>;master_symmetric_key=<root-
key>'
```

다음 COPY 명령은 암호화 키를 사용한 역할 기반 액세스 제어를 보여줍니다.

```
copy customer from 's3://amzn-s3-demo-bucket/mydata'
credentials
'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-
name>;master_symmetric_key=<root-key>'
```

열 매핑 옵션

기본적으로 COPY 명령은 데이터 파일의 필드 순서와 동일하게 값을 대상 테이블의 열에 삽입합니다. 기본 열 순서가 유효하지 않은 경우에는 열 목록을 지정하거나 JSONPath 표현식을 사용하여 원본 데이터 필드를 대상 열로 매핑할 수 있습니다.

- Column List
- JSONPaths File

열 목록

열 이름이 쉼표로 구분된 목록을 지정하여 원본 데이터 필드를 특정 대상 열로 로드할 수 있습니다. COPY 문에서는 열의 순서가 정해져 있지 않지만 Amazon S3 버킷과 같은 플랫 파일에서 로드할 때는 원본 데이터의 순서와 일치해야 합니다.

Amazon DynamoDB 테이블에서 로드할 때는 순서가 중요하지 않습니다. COPY 명령은 DynamoDB 테이블에서 가져오는 항목의 속성 이름을 Amazon Redshift 테이블의 열 이름과 일치시킵니다. 자세한 내용은 Amazon DynamoDB 테이블에서 데이터 로드 섹션을 참조하세요.

열 목록의 형식은 다음과 같습니다.

```
COPY tablename (column1 [,column2, ...])
```

대상 테이블의 열이 열 목록에서 빠져있으면 COPY가 대상 열의 DEFAULT 표현식을 로드합니다.

대상 열에 기본값이 없으면 COPY가 NULL을 로드하려고 합니다.

NOT NULL로 정의된 열에 COPY가 NULL을 할당하려고 하면 COPY 명령이 실패합니다.

IDENTITY 열이 열 목록에 포함되어 있으면 EXPLICIT_IDS 역시 지정해야 합니다. 반대로 IDENTITY 열이 없으면 EXPLICIT_IDS를 지정할 수 없습니다. 열 목록을 지정하지 않으면 COPY 명령이 마치 열 목록이 순서에 따라 완전하게 지정되어 있는 것처럼, 그리고 EXPLICIT_IDS도 지정하지 않은 것처럼 IDENTITY 열을 생략하여 실행됩니다.

열이 GENERATED BY DEFAULT AS IDENTITY로 정의된 경우 복사할 수 있습니다. 값이 새로 생성되거나 입력한 값으로 업데이트됩니다. EXPLICIT_IDS 옵션은 필요하지 않습니다. COPY는 자격 증명하이 워터마크를 업데이트하지 않습니다. 자세한 내용은 GENERATED BY DEFAULT AS IDENTITY 단원을 참조하십시오.

JSONPaths 파일

데이터 파일에서 데이터를 JSON 또는 Avro 형식으로 로드할 때는 COPY 명령이 JSON 또는 Avro 원본 데이터의 데이터 요소를 대상 테이블 열에 자동으로 매핑합니다. 이는 Avro 스키마의 필드 이름을 대상 테이블 또는 열 목록의 열 이름과 일치시키는 방식으로 수행됩니다.

경우에 따라 열 이름과 필드 이름이 일치하지 않거나 데이터 계층의 더 깊은 수준에 매핑해야 합니다. 이러한 경우 JSONPaths 파일을 사용하여 JSON 또는 Avro 데이터 요소를 열에 명시적으로 매핑할 수 있습니다.

자세한 내용은 JSONPaths 파일 단원을 참조하십시오.

데이터 형식 파라미터

COPY 명령에서는 기본적으로 원본 데이터가 문자로 구분된 UTF-8 텍스트 형식입니다. 기본 구분자는 파이프 문자(|)입니다. 원본 데이터가 다른 형식인 경우에는 다른 파라미터를 사용하여 데이터 형식을 지정합니다:

- FORMAT
- CSV
- DELIMITER
- FIXEDWIDTH
- SHAPEFILE
- AVRO
- JSON
- PARQUET
- ORC

COPY는 표준 데이터 형식 외에도 Amazon S3에서 COPY를 실행할 때 다음과 같은 열 기반 데이터 형식을 지원합니다.

- ORC
- PARQUET

열 기반 형식에서의 COPY는 지원되지만 특정 제한이 따릅니다. 자세한 내용은 열 기반 데이터 형식에서 COPY 명령 단원을 참조하십시오.

데이터 형식 파라미터

FORMAT [AS]

(선택 사항) 데이터 형식 키워드를 식별합니다. FORMAT 인수는 다음과 같습니다.

CSV [QUOTE [AS] 'quote_character']

입력 데이터에서 CSV 형식을 사용할 수 있습니다. 구분자와 줄 바꿈 문자, 그리고 캐리지 리턴을 자동으로 이스케이프하려면 QUOTE 파라미터에서 지정한 문자로 묶습니다. 기본 인용 부호는 큰따옴표(")입니다. 필드 안에서 인용 부호를 사용할 때는 추가 인용 부호로 이스케이프 처리해야 합니다. 예를 들어 인용 부호로 큰따옴표를 사용한다고 가정할 때 문자열 `A "quoted" word`는 입력 파일에 문자열 `"A ""quoted"" word"`로 삽입해야 합니다. CSV 파라미터를 사용할 때는 기본 구분자가 쉼표(,)입니다. DELIMITER 파라미터를 사용하여 다른 구분자를 지정할 수도 있습니다.

필드가 인용 부호로 묶이면 구분자와 인용 부호 사이의 공백은 무시됩니다. 구분자가 탭과 같은 공백 문자라면 구분자를 공백으로 처리하지 않습니다.

CSV는 FIXEDWIDTH, REMOVEQUOTES 또는 ESCAPE와 함께 사용할 수 없습니다.

QUOTE [AS] 'quote_character'

선택 사항. CSV 파라미터 사용 시 인용 부호로 사용할 문자를 지정합니다. 기본 문자는 큰 따옴표(")입니다. QUOTE 파라미터를 사용하여 큰따옴표 이외의 다른 인용 부호를 정의하는 경우 필드 내에서 큰따옴표를 이스케이프 처리할 필요는 없습니다. QUOTE 파라미터는 CSV 파라미터하고만 사용할 수 있습니다. AS 키워드는 옵션입니다.

DELIMITER [AS] ['delimiter_char']

파이프 문자(`|`), 쉼표(`,`), 탭(`\t`) 또는 `|~|` 기호 등의 여러 문자와 같이 입력 파일의 필드를 구분하는 데 사용되는 문자를 지정합니다. 인쇄할 수 없는 문자가 지원됩니다. 문자를 UTF-8 코드 단위로 8진수로 표시할 수도 있습니다. 8진수로는 '\ddd' 형식을 사용하는데, 여기서 'd'는 8진수(0~7)를 나타냅니다. 기본 구분자는 CSV 파라미터를 사용하지 않는 한 파이프 문자(`|`)입니다. CSV 파라미터를 사용할 경우에는 쉼표(`,`)가 기본 구분자입니다. AS 키워드는 옵션입니다. DELIMITER는 FIXEDWIDTH와 함께 사용할 수 없습니다.

FIXEDWIDTH 'fixedwidth_spec'

각 열이 구분자로 구분되지 않고, 폭의 길이가 고정되어 있는 파일에서 데이터를 로드합니다. fixedwidth_spec은 열의 레이블과 폭을 사용자 지정하는 문자열입니다. 열 레이블은 사용자 선택에 따라 문자열 또는 정수가 될 수 있습니다. 열 레이블은 열 이름과 관계가 없습니다. 레이블/폭 페어의 순서는 테이블 열의 순서와 정확히 일치해야 합니다. FIXEDWIDTH는 CSV 또는 DELIMITER와 함께 사용할 수 없습니다. Amazon Redshift에서는 CHAR 및 VARCHAR 열의 길이가 바이트로 표

현됩니다. 따라서 로드할 파일을 준비할 때는 지정하는 열의 폭이 멀티바이트 문자의 이진 길이를 수용할 정도로 충분한지 확인해야 합니다. 자세한 내용은 문자 형식 단원을 참조하십시오.

fixedwidth_spec의 형식은 다음과 같습니다.

```
'colLabel1:colWidth1,colLabel:colWidth2, ...'
```

SHAPEFILE [SIMPLIFY [AUTO] ['tolerance']]

입력 데이터에서 SHAPEFILE 형식을 사용할 수 있습니다. 기본적으로 shapefile의 첫 번째 열은 GEOMETRY 또는 IDENTITY 열입니다. 모든 후속 열은 shapefile에 지정된 순서를 따릅니다.

FIXEDWIDTH, REMOVEQUOTES 또는 ESCAPE와 함께 SHAPEFILE을 사용할 수 없습니다.

COPY FROM SHAPEFILE과 함께 GEOGRAPHY 객체를 사용하려면 먼저 GEOMETRY 열로 수집한 다음 객체를 GEOGRAPHY 객체로 캐스팅합니다.

SIMPLIFY [tolerance]

(옵션) Ramer-Douglas-Peucker 알고리즘과 주어진 허용치를 사용하여 수집 프로세스 동안 모든 지오메트리를 단순화합니다.

SIMPLIFY AUTO [tolerance]

(옵션) 최대 지오메트리 크기보다 큰 지오메트리만 단순화합니다. 이 단순화는 Ramer-Douglas-Peucker 알고리즘과 자동으로 계산된 허용치(지정된 허용치를 초과하지 않는 경우)를 사용합니다. 이 알고리즘은 지정된 허용치 내에서 객체를 저장할 크기를 계산합니다. 허용치 값은 옵션입니다.

shapefile 로드의 예는 Amazon Redshift에 shapefile 로드 섹션을 참조하세요.

AVRO [AS] 'avro_option'

원본 데이터를 Avro 형식으로 지정합니다.

Avro 형식은 다음과 같은 서비스 및 프로토콜을 통해 COPY에 지원됩니다.

- Amazon S3
- Amazon EMR
- 원격 호스트(SSH)

Avro는 DynamoDB에서 COPY에 대해 지원되지 않습니다.

Avro는 데이터 직렬화 프로토콜입니다. Avro 원본 파일에는 데이터 구조를 정의하는 스키마가 저장되어 있습니다. Avro 스키마 형식은 record가 되어야 합니다. COPY에서는 기본 비압축 코덱과

deflate 및 snappy 압축 코덱을 사용하여 생성된 Avro 파일이 허용됩니다. Avro에 대한 자세한 내용은 Apache Avro에서 확인할 수 있습니다.

avro_option의 유효 값은 다음과 같습니다.

- 'auto'
- 'auto ignorecase'
- 's3://*jsonpaths_file*'

기본값은 'auto'입니다.

COPY는 Avro 원본 데이터의 데이터 요소를 대상 테이블의 열에 자동으로 매핑합니다. 이는 Avro 스키마의 필드 이름을 대상 테이블의 열 이름과 일치시키는 방식으로 수행됩니다. 일치는 'auto'의 경우 대/소문자를 구분하고 'auto ignorecase'의 경우 대/소문자를 구분하지 않습니다.

Amazon Redshift 테이블의 열 이름은 항상 소문자이기 때문에 'auto' 옵션을 사용할 때는 일치하는 필드 이름 역시 소문자가 되어야 합니다. 필드 이름이 모두 소문자가 아닌 경우 'auto ignorecase' 옵션을 사용할 수 있습니다. 기본 'auto' 인수를 사용하면 COPY는 구조에서 첫 번째 수준의 필드 또는 외부 필드만 인식합니다.

열 이름을 Avro 필드 이름에 명시적으로 매핑하려면 JSONPaths 파일를 사용합니다.

기본적으로 COPY는 대상 테이블의 모든 열을 Avro 필드 이름과 일치시키려고 합니다. 열의 하위 집합까지 로드하려면 옵션으로 열 목록을 지정할 수도 있습니다. 대상 테이블의 열이 열 목록에서 빠져있으면 COPY가 대상 열의 DEFAULT 표현식을 로드합니다. 대상 열에 기본값이 없으면 COPY가 NULL을 로드하려고 합니다. 열 목록에 임의의 열이 포함되어 있다고 할 때 COPY가 Avro 데이터에서 일치하는 필드를 찾지 못할 경우에는 COPY가 해당 열에 NULL을 로드하려고 합니다.

NOT NULL로 정의된 열에 COPY가 NULL을 할당하려고 하면 COPY 명령이 실패합니다.

Avro 스키마

Avro 원본 데이터 파일에는 데이터 구조를 정의하는 스키마가 저장되어 있습니다. COPY는 Avro 원본 데이터 파일에 저장된 스키마를 읽어서 데이터 요소를 대상 테이블 열에 매핑합니다. 다음은 Avro 스키마의 예입니다.

```
{
    "name": "person",
    "type": "record",
    "fields": [
        {"name": "id", "type": "int"},
```

```
        {"name": "guid", "type": "string"},
        {"name": "name", "type": "string"},
        {"name": "address", "type": "string"}]
}
```

Avro 스키마는 JSON 형식으로 정의됩니다. 최상위 JSON 객체에는 이름, 즉 키가 포함된 이름-값 페어 3개인 "name", "type" 및 "fields"가 있습니다.

"fields" 키는 데이터 구조에서 각 필드의 이름과 데이터 형식을 정의하는 객체 배열과 쌍을 이룹니다. 기본적으로 COPY는 필드 이름을 열 이름과 자동으로 일치시킵니다. 열 이름은 항상 소문자이므로 'auto ignorecase' 옵션을 지정하지 않는 한 일치하는 필드 이름도 소문자여야 합니다. 열 이름과 일치하지 않는 필드 이름은 모두 무시됩니다. 순서는 중요하지 않습니다. 위의 예에서는 COPY가 열 이름 id, guid, name 및 address에 매핑하고 있습니다.

기본값인 'auto' 인수를 사용하면 COPY가 첫 번째 레벨의 객체만 열과 일치시킵니다. 스키마에서 더욱 깊숙한 레벨까지 매핑하려면, 혹은 필드 이름과 열 이름이 일치하지 않는 경우에는 JSONPaths 파일을 사용하여 매핑을 정의할 수 있습니다. 자세한 내용은 JSONPaths 파일 단원을 참조하십시오.

키와 연결된 값이 바이트, 배열, 레코드, 맵 또는 링크 같이 복잡한 Avro 데이터 형식인 경우에는 COPY가 이 값을 문자열로 로드합니다. 여기서 문자열은 데이터의 JSON 표현입니다. 열거형(enum)인 Avro 데이터 형식은 COPY에서 문자열로 로드되며, 이때 열거되는 내용은 형식 이름입니다. 예시는 JSON 형식의 COPY 지원을 확인하세요.

스키마와 파일 메타데이터가 저장되는 Avro 파일 헤더의 최대 크기는 1MB입니다.

단일 Avro 데이터 블록의 최대 크기는 4MB입니다. 이는 최대 행 크기와는 완전히 다릅니다. 단일 Avro 데이터 블록의 최대 크기를 초과하면 이에 따른 행 크기가 최댓값인 4MB보다 작더라도 COPY 명령이 중단됩니다.

Amazon Redshift가 행 크기를 계산할 때는 내부적으로 파이프 문자(|)를 두 번 계산합니다. 따라서 입력 데이터에 파이프 문자가 다수 포함되어 있으면 데이터 블록이 4MB 미만이더라도 행 크기가 4MB를 초과할 수도 있습니다.

JSON [AS] 'json_option'

원본 데이터가 JSON 형식입니다.

JSON 형식은 다음과 같은 서비스 및 프로토콜을 통해 COPY에 지원됩니다.

- Amazon S3
- Amazon EMR에서 COPY

- SSH 연결을 통한 COPY

JSON은 DynamoDB를 통한 COPY에서는 지원되지 않습니다.

json_option의 유효 값은 다음과 같습니다.

- `'auto'`
- `'auto ignorecase'`
- `'s3://jsonpaths_file'`
- `'noshred'`

기본값은 `'auto'`입니다. Amazon Redshift는 JSON 문서를 로드하는 동안 JSON 구조의 속성을 여러 열로 나누지 않습니다.

기본적으로 COPY는 대상 테이블의 모든 열을 JSON 필드 이름 키와 일치시키려고 합니다. 열의 하위 집합까지 로드하려면 옵션으로 열 목록을 지정할 수도 있습니다. JSON 필드 이름 키가 모두 소문자가 아닌 경우에도 `'auto ignorecase'` 옵션이나 JSONPaths 파일를 사용하여 열 이름을 명시적으로 JSON 필드 이름 키로 매핑할 수 있습니다.

대상 테이블의 열이 열 목록에서 빠져있으면 COPY가 대상 열의 DEFAULT 표현식을 로드합니다. 대상 열에 기본값이 없으면 COPY가 NULL을 로드하려고 합니다. 열 목록에 임의의 열이 포함되어 있다고 할 때 COPY가 JSON 데이터에서 일치하는 필드를 찾지 못할 경우에는 COPY가 해당 열에 NULL을 로드하려고 합니다.

NOT NULL로 정의된 열에 COPY가 NULL을 할당하려고 하면 COPY 명령이 실패합니다.

COPY는 JSON 원본 데이터의 데이터 요소를 대상 테이블의 열에 매핑합니다. 이는 원본 이름-값 페어의 객체 키 또는 이름을 대상 테이블의 열 이름과 일치시키는 방식으로 수행됩니다.

각 json_option 값에 대한 다음 세부 정보를 참조하세요.

'auto'

이 옵션을 사용하면 일치에서 대/소문자를 구분합니다. Amazon Redshift 테이블의 열 이름은 항상 소문자이기 때문에 `'auto'` 옵션을 사용할 때는 일치하는 JSON 필드 이름 역시 소문자가 되어야 합니다.

'auto ignorecase'

이 옵션을 사용하면 일치에서 대/소문자를 구분하지 않습니다. Amazon Redshift 테이블의 열 이름은 항상 소문자이기 때문에 `'auto ignorecase'` 옵션을 사용할 때는 해당 JSON 필드 이름은 소문자, 대문자 또는 대/소문자 혼합일 수 있습니다.

's3://jsonpaths_file'

이 옵션을 사용하면 COPY가 여기서 지정하는 JSONPaths 파일을 사용하여 JSON 원본 데이터의 데이터 요소를 대상 테이블의 열에 매핑합니다. *s3://jsonpaths_file* 인수는 단일 파일을 명시적으로 참조하는 Amazon S3 객체 키여야 합니다. 예를 들면 `'s3://amzn-s3-demo-bucket/jsonpaths.txt'`입니다. 인수는 키 접두사가 될 수 없습니다. JSONPaths 파일 사용에 대한 자세한 내용은 the section called “JSONPaths 파일” 섹션을 참조하세요.

경우에 따라 `jsonpaths_file`로 지정된 파일은 데이터 파일에 대해 `copy_from_s3_objectpath`로 지정된 경로와 동일한 접두사를 갖습니다. 그러면 COPY는 JSONPaths 파일을 데이터 파일로 읽고 오류를 반환합니다. 예를 들어 데이터 파일이 객체 경로 `s3://amzn-s3-demo-bucket/my_data.json`을 사용하고 JSONPaths 파일이 `s3://amzn-s3-demo-bucket/my_data.jsonpaths`라고 가정합니다. 이 경우 COPY는 `my_data.jsonpaths`를 데이터 파일로 로드하려고 합니다.

'noshred'

이 옵션을 사용하면 Amazon Redshift가 JSON 문서를 로드하는 동안 JSON 구조의 속성을 여러 열로 나누지 않습니다.

JSON 데이터 파일

JSON 데이터 파일에는 객체 또는 배열 집합이 저장됩니다. COPY는 저장되어 있는 JSON 객체 또는 배열을 각각 대상 테이블의 행 하나로 로드합니다. 행으로 로드되는 객체 또는 배열은 각각 독립된 루트 레벨 구조이어야 합니다. 즉 다른 JSON 구조의 멤버가 되어서는 안됩니다.

JSON 객체는 중괄호({ })로 시작해서 끝나며 이름-값 페어 집합이 순서에 상관없이 포함되어 있습니다. 쌍을 이루는 이름과 값은 각각 콜론으로 구분되며, 페어는 서로 쉼표로 구분됩니다. 기본적으로 이름-값 페어에서 객체 키 또는 이름은 테이블의 해당 열 이름과 일치해야 합니다. Amazon Redshift 테이블의 열 이름은 항상 소문자이기 때문에 일치하는 JSON 필드 이름 키 역시 소문자가 되어야 합니다. 열 이름과 JSON 키가 일치하지 않을 때는 the section called “JSONPaths 파일”을 사용하여 명시적으로 열을 키로 매핑하십시오.

JSON 객체의 순서는 중요하지 않습니다. 열 이름과 일치하지 않는 이름은 모두 무시됩니다. 다음은 간단한 JSON 객체의 구조를 나타낸 예입니다.

```
{
  "column1": "value1",
  "column2": value2,
  "notacolumn" : "ignore this value"
```

```
}
```

JSON 배열은 대괄호([ ])로 시작해서 끝나며 쉼표로 구분된 값 집합이 순서에 따라 포함되어 있습니다. 데이터 파일이 배열을 사용하는 경우에는 JSONPaths 파일을 지정하여 값과 열을 일치시켜야 합니다. 다음은 간단한 JSON 배열의 구조를 나타낸 예입니다.

```
["value1", value2]
```

JSON은 올바른 형식을 따라야 합니다. 예를 들어 객체나 배열은 공백을 제외하고 쉼표나 기타 다른 문자로 구분할 수 없습니다. 문자열은 큰따옴표로 묶어야 합니다. 인용 부호는 기울어진 인용 부호나 "스마트" 인용 부호가 아닌 단순 인용 부호(0x22)가 되어야 합니다.

중괄호 또는 대괄호를 포함하여 단일 JSON 객체 및 배열의 최대 크기는 4MB입니다. 이는 최대 행 크기와는 완전히 다릅니다. 단일 JSON 객체 또는 배열의 최대 크기를 초과하면 이에 따른 행 크기가 최댓값인 4MB보다 작더라도 COPY 명령이 중단됩니다.

Amazon Redshift가 행 크기를 계산할 때는 내부적으로 파이프 문자(|)를 두 번 계산합니다. 따라서 입력 데이터에 파이프 문자가 다수 포함되어 있으면 객체 크기가 4MB 미만이더라도 행 크기가 4MB를 초과할 수도 있습니다.

COPY는 줄 바꿈 문자로 \n을, 그리고 탭 문자로 \t를 로드합니다. 백슬래시를 로드하려면 백슬래시(\\)로 이스케이프하십시오.

COPY는 지정한 JSON 원본에서 올바른 형식의 유효 JSON 객체 또는 배열을 검색합니다. COPY가 사용 가능한 JSON 구조를 찾기 전에 또는 유효한 JSON 객체 또는 배열 사이에서 공백이 아닌 문자를 발견하면 COPY는 각 인스턴스에 대해 오류를 반환합니다. 이러한 오류는 MAXERROR 오류로 가산되며, 오류 수가 MAXERROR와 같거나 이를 초과하면 COPY는 중단됩니다.

Amazon Redshift는 STL_LOAD_ERRORS 시스템 테이블의 행에 오류를 일일이 기록합니다. LINE_NUMBER 열에는 오류를 일으킨 JSON 객체의 마지막 라인이 기록됩니다.

IGNOREHEADER를 지정하면 COPY가 지정한 만큼 JSON 데이터의 라인 수를 무시합니다. JSON 데이터의 줄 바꿈 문자도 항상 IGNOREHEADER 계산에 포함됩니다.

COPY는 기본적으로 빈 문자열을 빈 필드의 형태로 로드합니다. 이때 EMPTYASNULL을 지정하면 COPY가 CHAR 및 VARCHAR 필드의 빈 문자열을 NULL로 로드합니다. INT 같이 다른 데이터 형식의 빈 문자열은 항상 NULL로 로드됩니다.

다음 옵션은 JSON에서 지원되지 않습니다.

- CSV
- DELIMITER
- ESCAPE
- FILLRECORD
- FIXEDWIDTH
- IGNOREBLANKLINES
- NULL AS
- READRATIO
- REMOVEQUOTES

자세한 내용은 JSON 형식의 COPY 지원 단원을 참조하십시오. JSON 데이터 구조에 대한 자세한 내용은 www.json.org에서 확인할 수 있습니다.

JSONPaths 파일

JSON 형식 또는 Avro 원본 데이터에서 로드하는 경우에는 기본적으로 COPY가 원본 데이터의 첫 번째 수준 데이터 요소를 대상 테이블의 열에 매핑합니다. 이는 이름-값 페어의 각 이름 또는 객체 키를 대상 테이블의 열 이름과 일치시키는 방식으로 수행됩니다.

열 이름과 객체 키가 일치하지 않는 경우, 혹은 데이터 계층에서 더욱 깊숙한 레벨까지 매핑하려면 JSONPaths 파일을 사용하여 JSON 또는 Avro 데이터 요소를 명시적으로 열까지 매핑할 수 있습니다. JSONPaths 파일은 대상 테이블 또는 열 목록의 열 순서를 일치시켜 JSON 데이터 요소를 열로 매핑합니다.

JSONPaths 파일에는 배열이 아닌 단일 JSON 객체만 저장되어야 합니다. JSON 객체는 이름-값 페어입니다. 이름-값 페어에서 이름에 해당하는 객체 키는 `"jsonpaths"`가 되어야 합니다. 이름-값 페어에서 값은 JSONPath 표현식의 배열입니다. JSONPath 표현식은 각각 JSON 데이터 계층 또는 Avro 스키마의 단일 요소를 참조합니다. 이는 XPath 표현식이 XML 문서의 요소를 참조하는 방식과 비슷합니다. 자세한 내용은 JSONPath 표현식 단원을 참조하십시오.

JSONPaths 파일을 사용하려면 COPY 명령에 JSON 또는 AVRO 키워드를 추가합니다. 다음 형식을 사용하여 JSONPaths 파일의 S3 버킷 이름과 객체 경로를 지정합니다.

```
COPY tablename
FROM 'data_source'
CREDENTIALS 'credentials-args'
FORMAT AS { AVRO | JSON } 's3://jsonpaths_file';
```

s3://jsonpaths_file 값은 's3://amzn-s3-demo-bucket/jsonpaths.txt'와 같은 단일 파일을 명시적으로 참조하는 Amazon S3 객체 키여야 합니다. 키 접두사일 수 없습니다.

경우에 따라 경우에 따라 Amazon S3에서 로드하는 경우 jsonpaths_file로 지정된 파일은 데이터 파일에 대해 copy_from_s3_objectpath로 지정된 경로와 동일한 접두사를 갖습니다. 그러면 COPY는 JSONPaths 파일을 데이터 파일로 읽고 오류를 반환합니다. 예를 들어 데이터 파일이 객체 경로 s3://amzn-s3-demo-bucket/my_data.json을 사용하고 JSONPaths 파일이 s3://amzn-s3-demo-bucket/my_data.jsonpaths라고 가정합니다. 이 경우 COPY는 my_data.jsonpaths를 데이터 파일로 로드하려고 합니다.

키 이름이 "jsonpaths"가 아닌 다른 문자열이라면 COPY 명령이 오류를 반환하지는 않습니다. 하지만 jsonpaths_file을 무시하고 'auto' 인수를 사용합니다.

다음 중 한 가지라도 발생하면 COPY 명령이 중단됩니다.

- JSON 형식이 잘못된 경우
- JSON 객체가 2개 이상인 경우
- 공백을 제외한 문자가 객체 외부에 존재하는 경우
- 배열 요소가 빈 문자열이거나, 혹은 문자열이 아닌 경우

JSONPaths 파일에는 MAXERROR가 적용되지 않습니다.

ENCRYPTED 옵션을 지정하더라도 JSONPaths 파일을 암호화해서는 안 됩니다.

자세한 내용은 JSON 형식의 COPY 지원 단원을 참조하십시오.

JSONPath 표현식

JSONPaths 파일은 JSONPath 표현식을 사용하여 데이터 필드를 대상 열로 매핑합니다. 각 JSONPath 표현식은 Amazon Redshift 대상 테이블의 열 하나에 해당합니다. JSONPath 배열 요소의 순서는 대상 테이블의 열 순서와, 혹은 열 목록을 사용하는 경우 열 목록의 열 순서와 일치해야 합니다.

표시된 대로 필드 이름과 값 모두에 큰따옴표가 필요하며, 기울어진 따옴표나 "스마트" 따옴표가 아닌 단순한 따옴표(0x22)여야 합니다.

JSONPath 표현식에서 참조하는 객체 요소가 JSON 데이터에서 발견되지 않으면 COPY가 NULL 값을 로드하려고 합니다. 참조 객체의 형식이 잘못되면 COPY가 로드 오류를 반환합니다.

JSONPath 표현식에서 참조하는 배열 요소가 JSON 또는 Avro 데이터에서 발견되지 않으면 COPY가 다음 오류와 함께 중단됩니다. Invalid JSONPath format: Not an array or index out

of range. 이때 원본 데이터에 존재하지 않는 배열 요소는 JSONPaths에서 모두 제거한 후 원본 데이터의 배열이 올바른 형식인지 확인하십시오.

JSONPath 표현식은 대괄호 또는 점 표기법을 사용할 수 있지만 표기법을 혼용할 수는 없습니다. 다음은 대괄호 표기법을 사용한 JSONPath 표현식 예입니다.

```
{
    "jsonpaths": [
        "$['venuename']",
        "$['venuecity']",
        "$['venuestate']",
        "$['venueseats']"
    ]
}
```

다음은 점 표기법을 사용한 JSONPath 표현식 예입니다.

```
{
    "jsonpaths": [
        "$.venuename",
        "$.venuecity",
        "$.venuestate",
        "$.venueseats"
    ]
}
```

Amazon Redshift COPY 구문 컨텍스트에서 JSONPath 표현식은 JSON 또는 Avro 계층적 데이터 구조의 단일 이름 요소에 대한 명시적 경로를 지정해야 합니다. Amazon Redshift는 모호한 경로 또는 여러 이름 요소로 해석될 수 있는 와일드카드 문자 또는 필터 표현식과 같은 JSONPath 요소를 지원하지 않습니다.

자세한 내용은 JSON 형식의 COPY 지원 단원을 참조하십시오.

Avro 데이터의 JSONPaths 사용

다음은 다중 레벨의 Avro 스키마를 나타낸 예입니다.

```
{
    "name": "person",
    "type": "record",
    "fields": [
```

```
        {"name": "id", "type": "int"},
        {"name": "guid", "type": "string"},
        {"name": "isActive", "type": "boolean"},
        {"name": "age", "type": "int"},
        {"name": "name", "type": "string"},
        {"name": "address", "type": "string"},
        {"name": "latitude", "type": "double"},
        {"name": "longitude", "type": "double"},
        {
            "name": "tags",
            "type": {
                        "type" : "array",
                        "name" : "inner_tags",
                        "items" : "string"
                    }
        },
        {
            "name": "friends",
            "type": {
                        "type" : "array",
                        "name" : "inner_friends",
                        "items" : {
                                    "name" : "friends_record",
                                    "type" : "record",
                                    "fields" : [
                                                {"name" : "id", "type" : "int"},
                                                {"name" : "name", "type" : "string"}
                                               ]
                                  }
                    }
        },
        {"name": "randomArrayItem", "type": "string"}
    ]
}
```

다음은 AvroPath 표현식을 사용하여 위의 스키마를 참조하는 JSONPaths 파일의 예입니다.

```
{
    "jsonpaths": [
        "$.id",
        "$.guid",
        "$.address",
        "$.friends[0].id"
```

```
    ]
}
```

위의 JSONPaths 예에는 다음과 같은 요소가 포함되어 있습니다.

jsonpaths

AvroPath 표현식이 포함된 JSON 객체의 이름입니다.

[…]

경로 요소가 포함된 JSON 배열을 묶는 대괄호입니다.

$

Avro 스키마에서 "fields" 배열에 해당하는 루트 요소를 참조하는 달러 기호입니다.

"$.id",

AvroPath 표현식의 대상입니다. 위 예에서 대상은 "fields" 배열에서 이름으로 "id"가 포함된 요소입니다. 표현식은 쉼표로 구분됩니다.

"$.friends[0].id"

대괄호는 배열 인덱스를 나타냅니다. JSONPath 표현식은 0부터 시작되는 인덱싱을 사용하기 때문에 이 표현식에서는 "friends" 배열에서 이름으로 "id"가 포함된 첫 번째 요소를 참조합니다.

Avro 스키마 구문에서는 안쪽 필드를 사용하여 레코드 및 배열 데이터 형식의 구조를 정의해야 합니다. AvroPath 표현식에서는 안쪽 필드가 무시됩니다. 예를 들어 "friends" 필드는 "inner_friends"라는 이름의 배열을 정의하고, 이 배열은 "friends_record"라는 이름의 레코드를 정의합니다. "id" 필드를 참조하는 AvroPath 표현식은 추가되는 필드를 무시하고 대상 필드를 직접 참조할 수 있습니다. 다음은 "friends" 배열에 속한 두 필드를 참조하는 AvroPath 표현식입니다.

```
"$.friends[0].id"
"$.friends[0].name"
```

열 기반 데이터 형식 파라미터

COPY는 표준 데이터 형식 외에도 Amazon S3에서 COPY를 실행할 때 다음과 같은 열 기반 데이터 형식을 지원합니다. 열 기반 형식에 대한 COPY는 지원되지만 특정 제한이 따릅니다. 자세한 내용은 열 기반 데이터 형식에서 COPY 명령 단원을 참조하십시오.

ORC

ORC(Optimized Row Columnar) 파일 형식을 사용하는 파일에서 데이터를 불러옵니다.

PARQUET

Parquet 파일 형식을 사용하는 파일에서 데이터를 불러옵니다.

파일 압축 파라미터

다음 파라미터를 지정하여 압축된 데이터 파일로부터 로드할 수 있습니다.

파일 압축 파라미터

bzip2

입력 파일이 bzip2 압축 형식(.bz2 파일)을 따르도록 지정하는 값입니다. COPY 작업은 데이터를 로드할 때 각 압축 파일을 읽고 데이터 압축을 풉니다.

GZIP

입력 파일이 gzip 압축 형식(.gz 파일)을 따르도록 지정하는 값입니다. COPY 작업은 데이터를 로드할 때 각 압축 파일을 읽고 데이터 압축을 풉니다.

LZOP

입력 파일이 lzop 압축 형식(.lzo 파일)을 따르도록 지정하는 값입니다. COPY 작업은 데이터를 로드할 때 각 압축 파일을 읽고 데이터 압축을 풉니다.

> **Note**
>
> COPY는 lzop --filter 옵션을 사용한 압축 파일을 지원하지 않습니다.

ZSTD

입력 파일이 Zstandard 압축 형식(.zst 파일)을 따르도록 지정하는 값입니다. COPY 작업은 데이터를 로드할 때 각 압축 파일을 읽고 데이터 압축을 풉니다.

> **Note**
>
> ZSTD는 Amazon S3에서 COPY에서만 지원됩니다.

데이터 변환 파라미터

COPY 명령은 테이블에 데이터를 로드할 때 원본 데이터의 문자열을 묵시적으로 대상 열의 데이터 형식으로 변환합니다. 기본 동작과 다른 변환을 지정해야 하거나, 혹은 기본 변환이 오류를 일으킬 때는 다음 파라미터를 지정하여 데이터 변환을 관리할 수 있습니다. 이러한 파라미터 구문에 대한 자세한 내용은 COPY 구문을 참조하세요.

- ACCEPTANYDATE
- ACCEPTINVCHARS
- BLANKSASNULL
- DATEFORMAT
- EMPTYASNULL
- ENCODING
- ESCAPE
- EXPLICIT_IDS
- FILLRECORD
- IGNOREBLANKLINES
- IGNOREHEADER
- NULL AS
- REMOVEQUOTES
- ROUNDEC
- TIMEFORMAT
- TRIMBLANKS
- TRUNCATECOLUMNS

데이터 변환 파라미터

ACCEPTANYDATE

`00/00/00 00:00:00` 같이 잘못된 형식을 포함하여 모든 데이터 형식을 오류 없이 로드할 수 있도록 허용합니다. 이 파라미터는 TIMESTAMP 및 DATE 열에만 적용됩니다. ACCEPTANYDATE는 항상 DATEFORMAT 파라미터와 함께 사용합니다. 데이터의 날짜 형식이 DATEFORMAT 명세와 일치하지 않는 경우에는 Amazon Redshift가 NULL 값을 해당 필드에 삽입합니다.

ACCEPTINVCHARS [AS] ['replacement_char']

데이터에 잘못된 UTF-8 문자가 포함되어 있어도 VARCHAR 열에 데이터를 로드할 수 있습니다. ACCEPTINVCHARS를 지정하면 COPY가 잘못된 UTF-8 문자를 각각 replacement_char에서 지정하는 문자로 구성된, 동일한 길이의 문자열로 변경합니다. 예를 들어 변경 문자가 '^'라면 잘못된 3바이트 문자는 '^^^'로 변경됩니다.

변경 문자는 NULL을 제외한 모든 ASCII 문자가 될 수 있습니다. 기본 문자는 물음표(?)입니다. 잘못된 UTF-8 문자에 대한 자세한 내용은 멀티바이트 문자 로드 오류 섹션을 참조하세요.

COPY는 잘못된 UTF-8 문자가 포함된 행의 수를 반환한 후 해당하는 행마다 STL_REPLACEMENTS 시스템 테이블에 항목을 추가합니다. 이때 각 노드 조각에서 항목이 추가되는 행의 최대 수는 100개입니다. 이후 추가되는 잘못된 UTF-8 문자 역시 변경되지만 이러한 문자의 변경 이벤트는 기록되지 않습니다.

ACCEPTINVCHARS를 지정하지 않으면 COPY가 잘못된 UTF-8 문자를 발견할 때마다 오류를 반환합니다.

ACCEPTINVCHARS는 VARCHAR 열에서만 유효합니다.

BLANKSASNULL

공백 문자로만 구성된 빈 필드를 NULL로 로드합니다. 이 옵션은 CHAR 및 VARCHAR 열에만 적용됩니다. INT 같이 다른 데이터 형식의 빈 필드는 항상 NULL로 로드됩니다. 예를 들어 공백 문자 3개가 다른 문자 없이 연이어 포함된 문자열은 NULL로 로드됩니다. 이 옵션을 사용하지 않으면 기본 동작으로 공백 문자를 있는 그대로 로드합니다.

DATEFORMAT [AS] {'dateformat_string' | 'auto' }

인수를 지정하지 않을 경우 기본값은 'YYYY-MM-DD'입니다. 예를 들어 이를 대신할 수 있는 유효 형식은 'MM-DD-YYYY'입니다.

COPY 명령이 날짜 또는 시간 값의 형식을 인식하지 못하거나, 혹은 날짜 또는 시간 값이 다른 형식인 경우에는 DATEFORMAT 또는 TIMEFORMAT 파라미터에 'auto' 인수를 사용하십시오. 'auto' 인수는 DATEFORMAT 및 TIMEFORMAT 문자열 사용 시 지원되지 않는 몇 가지 형식을 인식합니다. 'auto'' 키워드는 대/소문자를 구분합니다. 자세한 내용은 DATEFORMAT 및 TIMEFORMAT 옵션의 자동 인식 사용 단원을 참조하십시오.

날짜 형식에 시간 정보(시, 분, 초)가 포함될 수는 있지만 이 정보는 무시됩니다. AS 키워드는 옵션입니다. 자세한 내용은 DATEFORMAT 및 TIMEFORMAT 문자열 단원을 참조하십시오.

EMPTYASNULL

Amazon Redshift가 비어있는 CHAR 및 VARCHAR 필드를 NULL로 로드하도록 지정합니다. INT 같이 다른 데이터 형식의 빈 필드는 항상 NULL로 로드됩니다. 빈 필드는 2개의 구분자가 사이에 아무런 문자도 없이 연이어 데이터에 포함되어 있을 때 발생합니다. EMPTYASNULL과 NULL AS '' (empty string)는 동작이 동일합니다.

ENCODING [AS] file_encoding

로드 데이터의 인코딩 유형을 지정합니다. COPY 명령은 로딩 도중 데이터를 지정된 인코딩에서 UTF-8로 변환합니다.

file_encoding의 유효 값은 다음과 같습니다.

- UTF8
- UTF16
- UTF16LE
- UTF16BE
- ISO88591

기본값은 UTF8입니다.

원본 파일 이름은 UTF-8 인코딩을 사용해야 합니다.

다음 파일은 로드 데이터로 다른 인코딩이 지정되어 있더라도 UTF-8 인코딩을 사용해야 합니다.

- 매니페스트 파일
- JSONPaths 파일

다음 파라미터와 함께 입력되는 인수 문자열은 UTF-8을 사용해야 합니다.

- FIXEDWIDTH 'fixedwidth_spec'
- ACCEPTINVCHARS 'replacement_char'
- DATEFORMAT 'dateformat_string'
- TIMEFORMAT 'timeformat_string'
- NULL AS 'null_string'

고정 폭 데이터 파일은 UTF-8 인코딩을 사용해야 합니다. 필드 폭은 바이트 수가 아니라 문자 수를 기준으로 하기 때문입니다.

로드 데이터는 모두 지정된 인코딩을 사용해야 합니다. COPY가 다른 인코딩을 만나면 파일을 건너뛰고 오류를 반환합니다.

UTF16을 지정한 경우에는 데이터에 바이트 순서 표식(BOM)이 있어야 합니다. UTF-16 데이터가 리틀 엔디안(LE)인지, 빅 엔디안(BE)인지 알고 있다면 BOM 유무에 상관없이 UTF16LE 또는 UTF16BE를 사용할 수 있습니다.

ISO-8859-1 인코딩을 사용하려면 ISO88591을 지정합니다. 자세한 내용은 Wikipedia의 ISO/IEC 8859-1을 참조하세요.

ESCAPE

이 파라미터를 지정하면 입력 데이터의 백슬래시 문자(\)가 이스케이프 문자로 처리됩니다. 그러면 백슬래시 문자 바로 뒤에 있는 문자가 일반적으로 특정 용도로 사용되는 문자라고 해도 현재 열 값의 일부로 테이블에 로드됩니다. 예를 들어 이 파라미터를 사용하여 구분자 문자, 인용 부호, 줄 바꿈 문자 또는 이스케이프 문자 자체를 이스케이프 처리할 수 있습니다. 단, 이러한 문자가 모두 열 값에서 유효해야 합니다.

ESCAPE 파라미터를 REMOVEQUOTES 파라미터와 함께 지정하면 그 외에 삭제될 수도 있는 인용 부호(' 또는 ")를 이스케이프 처리하여 유지할 수 있습니다. 기본 NULL 문자열인 \N은 있는 그대로 유효하지만 입력 데이터에서는 \\N로 이스케이프 처리할 수도 있습니다. NULL AS 파라미터를 사용해 대체 NULL 문자열을 지정하지만 않는다면 \N과 \\N은 동일한 결과를 산출합니다.

> **Note**
>
> 제어 문자 0x00(NUL)은 이스케이프 처리할 수 없기 때문에 입력 데이터에서 삭제하거나, 혹은 변환해야 합니다. 이 문자는 레코드 끝(EOR) 마커로 처리되어 나머지 레코드는 모두 잘립니다.

FIXEDWIDTH 로드 시 ESCAPE 파라미터를 사용할 수 없으며, 이스케이프 문자 자체를 지정하는 것도 안 됩니다. 이스케이프 문자는 항상 백슬래시 문자입니다. 또한 입력 데이터에는 적절한 자리에 이스케이프 문자가 포함되어야 합니다.

여기 몇 가지 입력 데이터를 비롯해 ESCAPE 파라미터를 지정했을 때 로드되는 데이터의 예가 있습니다. 행 4의 결과는 REMOVEQUOTES 파라미터 역시 지정된다는 것을 가정합니다. 입력 데이터는 다음과 같이 파이프로 구분된 필드 2개로 구성됩니다.

```
1|The quick brown fox\[newline]
jumped over the lazy dog.
2| A\\B\\C
3| A \| B \| C
4| 'A Midsummer Night\'s Dream'
```

열 2로 로드되는 데이터는 다음과 같습니다.

```
The quick brown fox
jumped over the lazy dog.
A\B\C
A|B|C
A Midsummer Night's Dream
```

> **Note**
>
> 로드 시 입력 데이터에 대한 이스케이프 문자의 적용 여부는 사용자에게 책임이 있습니다. 이러한 요건의 한 가지 예외라면 이전에 ESCAPE 파라미터를 사용해 언로드되었던 데이터를 다시 로드할 때입니다. 이때는 데이터에 이미 필요한 이스케이프 문자가 포함되어 있기 때문입니다.

ESCAPE 파라미터는 8진수, 16진수, Unicode 또는 기타 이스케이프 시퀀스 표기법을 해석하지 않습니다. 예를 들어 원본 데이터에 8진수 라인 피드 값(\012)이 포함되어 있을 때 ESCAPE 파라미터를 사용해 이 데이터를 로드하려고 하면 Amazon Redshift가 값 012를 테이블로 로드하는 동시에 이 값을 이스케이프 처리되는 라인 피드로 해석하지 않습니다.

Microsoft Windows 플랫폼에서 작성된 데이터의 줄 바꿈 문자를 이스케이프 처리하려면 캐리지 리턴용 하나와 라인 피드용 하나, 총 2개의 이스케이프 문자를 사용해야 할 수도 있습니다. 그 밖에 파일을 로드하기 전에 dos2unix 유틸리티 등을 사용해 캐리지 리턴을 삭제하는 방법도 있습니다.

EXPLICIT_IDS

원본 데이터 파일에서 명시적인 값을 사용해 자동 생성된 값을 재정의하려면 IDENTITY 열이 포함된 테이블에서 EXPLICIT_IDS를 사용하십시오. 명령에 열 목록을 추가하는 경우에는 해당 목록에 이 파라미터를 사용할 IDENTITY 열도 포함되어야 합니다. EXPLICIT_IDS 값의 데이터 형식은 CREATE TABLE 정의에서 지정한 IDENTITY 형식과 일치해야 합니다.

EXPLICIT_IDS 옵션을 사용하여 테이블에 대해 COPY 명령을 실행할 때, Amazon Redshift는 테이블에서 IDENTITY 열의 고유성을 확인하지 않습니다.

열이 GENERATED BY DEFAULT AS IDENTITY로 정의된 경우 복사할 수 있습니다. 값이 새로 생성되거나 입력한 값으로 업데이트됩니다. EXPLICIT_IDS 옵션은 필요하지 않습니다. COPY는 자격 증명 하이 워터마크를 업데이트하지 않습니다.

EXPLICIT_ID를 사용하는 COPY 명령의 예는 IDENTITY 열의 명시적인 값을 사용한 VENUE 로드을 참조하십시오.

FILLRECORD

서로 인접한 열들이 일부 레코드 끝에서 누락되었을 때도 데이터 파일의 로드를 허용합니다. 누락된 열은 NULL로 로드됩니다. 텍스트 및 CSV 형식의 경우 누락된 열이 VARCHAR 열이면 NULL 대신 길이가 0인 문자열이 로드됩니다. 텍스트 및 CSV에서 VARCHAR 열로 NULL을 로드하려면 EMPTYASNULL 키워드를 지정합니다. NULL 치환은 열 정의가 NULL을 허용할 때만 유효합니다.

예를 들어 테이블 정의에 NULL을 허용하는 CHAR 열이 4개 포함되어 있고, 레코드에 포함된 값이 `apple, orange, banana, mango`라고 가정한다면 COPY 명령은 `apple, orange` 값만 포함된 레코드를 로드하여 채울 수 있습니다. 이때 누락된 CHAR 값은 NULL 값으로 로드됩니다.

IGNOREBLANKLINES

데이터 파일에서 라인 피드만 포함된 빈 라인을 무시하고 로드하지 않습니다.

IGNOREHEADER [AS] number_rows

지정하는 number_rows를 파일 헤더로 처리하고 로드하지 않습니다. IGNOREHEADER는 병렬 로드에서 모든 파일의 헤더를 건너뛸 때 사용됩니다.

NULL AS 'null_string'

null_string과 일치하는 필드를 NULL로 로드합니다. 여기에서 null_string은 어떤 문자열이든 가능합니다. 데이터에 NUL(UTF-8 0000) 또는 이진 0(0x000)으로도 불리는 null 종결자가 포함되어 있으면 COPY는 이를 다른 문자로 취급합니다. 예를 들어 '1' || NUL || '2'를 포함하는 레코드는 길이가 3바이트인 문자열로 복사됩니다. 필드에 NUL만 포함된 경우 NULL AS로 `'\0'` 또는 `'\000'`을 지정(예: `NULL AS '\0'` 또는 `NULL AS '\000'`)하여 null 종결자를 NULL로 바꿀 수 있습니다. 필드에 NUL로 끝나는 문자열이 있을 때 NULL AS를 지정하면 끝에서 NUL과 함께 문자열이 삽입됩니다. null_string 값에 '\n'(줄 바꿈)을 사용하지 않습니다. Amazon Redshift는 줄 구분 기호로 사용하기 위해 '\n'을 예약합니다. 기본 null_string은 `'\N'`입니다.

> **Note**
>
> NULL을 NOT NULL로 정의된 열에 로드하려고 하면 COPY 명령이 실패합니다.

REMOVEQUOTES

입력 데이터의 문자열에서 묶고 있는 인용 부호를 제거합니다. 인용 부호 안의 문자는 구분자를 포함하여 모두 유지됩니다. 문자열에 선행하는 작은 또는 큰 따옴표만 있고 후행하는 인용 부호가 없는 경우에는 COPY 명령이 해당 행을 로드하지 못하고 오류를 반환합니다. 다음 표는 인용 부호로 묶인 문자열과 이 옵션으로 로드되는 값의 몇 가지 예를 보여줍니다.

입력 문자열	REMOVEQUOTES 옵션으로 로드되는 값
"The delimiter is a pipe (|) character"	The delimiter is a pipe (|) character
'Black'	검은색
"흰색"	흰색
파란색'	파란색'
'파란색	값이 로드되지 않음: 오류 조건
"파란색	값이 로드되지 않음: 오류 조건
' ' 'black' ' '	' 'black' '
' '	<공백>

ROUNDEC

입력 값의 소수점 자릿수가 열 값의 소수점 자릿수보다 큰 경우 숫자 값을 반올림합니다. 기본적으로 COPY는 열 값의 소수점 자릿수에 맞출 필요가 있을 때 값을 자릅니다. 예를 들어 `20.259` 값을 DECIMAL(8,2) 열에 로드한다고 가정할 때 기본적으로 COPY가 이 값을 `20.25`로 자릅니다. 하지만 ROUNDEC를 지정하면 COPY가 값을 `20.26`으로 반올림합니다. INSERT 명령은 열 값의 소수점 자릿수와 일치시켜야 할 때마다 항상 값을 반올림합니다. 따라서 ROUNDEC 파라미터가 지정된 COPY 명령은 INSERT 명령과 똑같이 동작합니다.

TIMEFORMAT [AS] {'timeformat_string' | 'auto' | 'epochsecs' | 'epochmillisecs' }

시간 형식을 지정합니다. TIMEFORMAT을 지정하지 않았을 때 기본 형식은 TIMESTAMP 열의 경우 `YYYY-MM-DD HH:MI:SS`이고, 그리고 TIMESTAMPTZ 열의 경우 `YYYY-MM-DD HH:MI:SSOF`입니다. 여기에서 `OF`는 협정 세계시(UTC)의 오프셋을 말합니다. timeformat_string에는 시간대 지정자를 추가할 수 없습니다. 기본 형식과 다른 형식의 TIMESTAMPTZ 데이터를 로드하려면 'auto'를 지정해야 합니다. 자세한 내용은 DATEFORMAT 및 TIMEFORMAT 옵션의 자동 인식 사용 섹션을 참조하세요. timeformat_string에 대한 자세한 내용은 DATEFORMAT 및 TIMEFORMAT 문자열 섹션을 참조하세요.

`'auto'` 인수는 DATEFORMAT 및 TIMEFORMAT 문자열 사용 시 지원되지 않는 몇 가지 형식을 인식합니다. COPY 명령이 날짜 또는 시간 값의 형식을 인식하지 못하거나, 혹은 날짜 및 시간 값이 서로 다른 형식인 경우에는 DATEFORMAT 또는 TIMEFORMAT 파라미터에 `'auto'` 인수를 사용

하십시오. 자세한 내용은 DATEFORMAT 및 TIMEFORMAT 옵션의 자동 인식 사용 단원을 참조하십시오.

원본 데이터가 epoch 시간, 즉 1970년 1월 1일 00:00:00 UTC 이후의 시간(초 또는 밀리초)으로 표현되는 경우에는 'epochsecs' 또는 'epochmillisecs'를 지정합니다.

'auto', 'epochsecs' 및 'epochmillisecs' 키워드는 대/소문자를 구분합니다.

AS 키워드는 옵션입니다.

TRIMBLANKS

VARCHAR 문자열에서 후행 공백 문자를 제거합니다. 이 파라미터는 VARCHAR 데이터 형식의 열에만 적용됩니다.

TRUNCATECOLUMNS

열의 데이터를 열 명세에 따라 적합한 수의 문자로 자릅니다. VARCHAR 또는 CHAR 데이터 형식의 열에만 적용되며, 행의 크기는 4MB 이하입니다.

데이터 로드 작업

문제 해결을 위해, 혹은 로드 시간을 줄일 목적으로 다음 파라미터를 지정하여 로드 작업의 기본 동작을 관리합니다.

- COMPROWS
- COMPUPDATE
- IGNOREALLERRORS
- MAXERROR
- NOLOAD
- STATUPDATE

파라미터

COMPROWS numrows

압축 분석 시 샘플 크기로 사용할 행의 수를 지정합니다. 분석은 각 데이터 조각의 행에서 실행됩니다. 예를 들어 COMPROWS 1000000(1,000,000)을 지정했을 때 시스템에 총 4개의 조각이 있는 경우에는 조각당 250,000개까지만 행을 읽고 분석합니다.

COMPROWS가 지정되지 않은 경우 샘플 크기는 기본적으로 조각당 100,000개입니다. 기본값인 조각당 100,000개의 행보다 낮은 COMPROWS 값은 자동으로 기본값으로 업그레이드됩니다. 하지만 로드되는 데이터의 크기가 유의적인 샘플을 산출할 정도로 충분하지 않은 경우에는 자동 압축이 일어나지 않습니다.

COMPROWS 숫자가 입력 파일의 행 개수보다 큰 값이면 COPY 명령이 가능한 모든 행에 대한 압축 분석을 계속 진행하고 실행합니다. 이 인수의 허용 범위는 1000과 2147483647(2,147,483,647) 사이의 수입니다.

COMPUPDATE [PRESET | { ON | TRUE } | { OFF | FALSE }],

COPY 실행 시 압축 인코딩의 자동 적용 여부를 제어합니다.

COMPUPDATE가 PRESET인 경우 열에 이미 RAW 이외의 다른 인코딩이 있어도 대상 테이블이 비어 있으면 COPY 명령이 각 열에 대한 압축 인코딩을 선택합니다. 현재 지정된 열 인코딩을 바꿀 수 있습니다. 각 열에 대한 인코딩은 열 데이터 형식을 기준으로 합니다. 데이터가 샘플링되지 않습니다. Amazon Redshift가 다음과 같이 압축 인코딩을 자동으로 할당합니다.

- 정렬 키로 정의된 열은 RAW 압축이 할당됩니다.
- BOOLEAN, REAL 또는 DOUBLE PRECISION 데이터 형식으로 정의된 열은 RAW 압축이 할당됩니다.
- SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIMESTAMP 또는 TIMESTAMPTZ로 정의된 열에는 AZ64 압축이 할당됩니다.
- CHAR 또는 VARCHAR로 정의된 열에는 LZO 압축이 할당됩니다.

COMPUPDATE가 생략된 경우 COPY 명령은 대상 테이블이 비어 있고 임의의 열에 대한 인코딩(RAW 이외)을 지정하지 않은 경우에만 각 열에 대한 압축 인코딩을 선택합니다. 각 열에 대한 인코딩은 Amazon Redshift에 의해 결정됩니다. 데이터가 샘플링되지 않습니다.

COMPUPDATE가 ON(또는 TRUE)이거나 옵션 없이 COMPUPDATE가 지정된 경우 테이블 열에 이미 RAW 외에 다른 인코딩이 있어도 테이블이 비어 있으면 COPY 명령이 자동 압축을 적용합니다. 현재 지정된 열 인코딩을 바꿀 수 있습니다. 각 열에 대한 인코딩은 샘플 데이터 분석을 기반으로 합니다. 자세한 내용은 자동 압축을 사용하여 테이블 로드 단원을 참조하십시오.

COMPUPDATE가 OFF(또는 FALSE)일 때는 자동 압축이 비활성화됩니다. 열 인코딩은 변경되지 않습니다.

압축을 분석하는 시스템 테이블에 대한 자세한 내용은 STL_ANALYZE_COMPRESSION 섹션을 참조하세요.

IGNOREALLERRORS

이 옵션을 지정하여 로드 작업 중에 발생하는 모든 오류를 무시할 수 있습니다.

MAXERROR 옵션을 지정하면 IGNOREALLERRORS 옵션을 지정할 수 없습니다. ORC 및 Parquet를 포함한 열 형식에 대해서는 IGNOREALLERRORS 옵션을 지정할 수 없습니다.

MAXERROR [AS] error_count

데이터 로드 시 error_count를 오류 수 이상으로 반환하면 로드가 중단됩니다. 반환되는 오류 수가 더 적을 때는 로드를 계속하면서 INFO 메시지를 반환하여 로드할 수 없는 행의 수를 알려줍니다. 이 파라미터는 형식 오류 또는 기타 데이터 불일치로 인해 일부 행을 테이블에 로드하지 못할 때 중단 없이 로드를 계속할 수 있도록 지정합니다.

첫 번째 오류 발생 시 바로 로드를 중단하려면 이 값을 0 또는 1로 설정하십시오. AS 키워드는 옵션입니다. MAXERROR 기본 값은 0이며 최대 100000까지 설정할 수 있습니다.

Amazon Redshift의 병렬 특성 때문에 실제로 보고되는 오류 수는 지정한 MAXERROR 값보다 클 수 있습니다. Amazon Redshift 클러스터 노드에서 MAXERROR 값이 초과된 것을 감지하면 각 노드가 발생하는 모든 오류를 보고합니다.

NOLOAD

실제로 데이터를 로드하지 않고 데이터 파일의 유효성을 검사합니다. NOLOAD 파라미터를 사용하면 실제로 데이터를 로드하기 전에 데이터 파일의 로드에 따른 잠재적 오류 여부를 확인할 수 있습니다. COPY를 NOLOAD 파라미터와 함께 실행하면 파일의 구문만 분석하기 때문에 데이터 로드보다 속도가 훨씬 빠릅니다.

STATUPDATE [{ ON | TRUE } | { OFF | FALSE }]

COPY 명령을 성공적으로 마치면 옵티마이저 통계를 자동 계산하여 업데이트합니다. 기본적으로 처음부터 빈 테이블이라면 STATUPDATE 파라미터를 사용하지 않아도 통계가 자동 업데이트됩니다.

데이터를 비어있지 않은 테이블로 수집하면서 테이블 크기의 변동이 심할 때마다 ANALYZE 명령을 실행하거나, 혹은 STATUPDATE ON 인수를 사용하여 통계를 업데이트하는 것이 좋습니다.

STATUPDATE ON(또는 TRUE)일 때는 테이블이 처음부터 비어있는지 상관없이 통계가 자동 업데이트됩니다. STATUPDATE를 사용할 경우 현재 사용자가 테이블 소유자이거나 수퍼유저이어야 합니다. 하지만 STATUPDATE를 지정하지 않는 경우에는 사용자에게 INSERT 권한만 있으면 됩니다.

STATUPDATE OFF(또는 FALSE)일 때는 통계가 업데이트되지 않습니다.

자세한 내용은 테이블 분석 섹션을 참조하세요.

알파벳 순서의 파라미터 목록

다음은 COPY 명령 파라미터 각각에 대한 설명 링크가 알파벳 순으로 정렬되어 있는 목록입니다.

- ACCEPTANYDATE
- ACCEPTINVCHARS
- ACCESS_KEY_ID and SECRET_ACCESS_KEY
- AVRO
- BLANKSASNULL
- BZIP2
- COMPROWS
- COMPUPDATE
- CREDENTIALS
- CSV
- DATEFORMAT
- DELIMITER
- EMPTYASNULL
- ENCODING
- ENCRYPTED
- ESCAPE
- EXPLICIT_IDS
- FILLRECORD
- FIXEDWIDTH
- FORMAT
- FROM
- GZIP
- IAM_ROLE

- IGNOREALLERRORS
- IGNOREBLANKLINES
- IGNOREHEADER
- JSON
- LZOP
- MANIFEST
- MASTER_SYMMETRIC_KEY
- MAXERROR
- NOLOAD
- NULL AS
- READRATIO
- REGION
- REMOVEQUOTES
- ROUNDEC
- SESSION_TOKEN
- SHAPEFILE
- SSH
- STATUPDATE
- TIMEFORMAT
- SESSION_TOKEN
- TRIMBLANKS
- TRUNCATECOLUMNS
- ZSTD

사용 노트

주제

- 다른 AWS 리소스에 대한 액세스 권한
- Amazon S3 액세스 포인트 별칭과 함께 COPY 사용

- Amazon S3에서 멀티바이트 데이터 로드
- GEOMETRY 또는 GEOGRAPHY 데이터 유형의 열 로드
- HLLSKETCH 데이터 형식 로드
- VARBYTE 데이터 유형의 열 로드
- 다수의 파일을 읽어올 때 발생하는 오류
- JSON 형식의 COPY 지원
- 열 기반 데이터 형식에서 COPY 명령
- DATEFORMAT 및 TIMEFORMAT 문자열
- DATEFORMAT 및 TIMEFORMAT 옵션의 자동 인식 사용

다른 AWS 리소스에 대한 액세스 권한

Amazon S3, Amazon DynamoDB, Amazon EMR, Amazon EC2 등의 다른 AWS 리소스와 클러스터 사이에 데이터를 이동시키려면 클러스터에 리소스에 대한 액세스 권한을 비롯해 필요한 작업 권한이 있어야 합니다. 예를 들어 Amazon S3에서 데이터를 로드하려면 COPY에 버킷에 대한 LIST 액세스 권한과 버킷 객체에 대한 GET 액세스 권한이 있어야 합니다. 최소 권한에 대한 자세한 내용은 COPY, UNLOAD 및 CREATE LIBRARY 작업을 위한 IAM 권한 섹션을 참조하세요.

리소스에 대한 액세스 권한을 얻기 위해서는 먼저 클러스터의 인증이 필요합니다. 인증 방법은 다음 중 한 가지를 선택할 수 있습니다.

- 역할 기반 액세스 제어 – 역할 기반 액세스 제어에서는 사용자가 클러스터가 인증 및 권한 부여에 사용할 AWS Identity and Access Management(IAM) 역할을 지정합니다. AWS 자격 증명과 민감한 데이터를 보호하려면 역할 기반 인증의 사용을 강력하게 권장합니다.
- 키 기반 액세스 제어 – 키 기반 액세스 제어에서는 사용자의 AWS 액세스 자격 증명(액세스 키 ID와 비밀 액세스 키)을 일반 텍스트로 입력합니다.

역할 기반 액세스 제어

역할 기반 액세스 제어를 통해 클러스터는 IAM 역할을 임시로 맡습니다. 그런 다음 역할에 부여되는 권한에 따라 클러스터가 필요한 AWS 리소스에 액세스할 수 있습니다.

AWS에서 자격 증명이 할 수 있는 것과 없는 것을 결정하는 권한 정책을 갖춘 AWS 자격 증명이라는 점에서 IAM 역할 생성은 사용자에게 권한을 제공하는 것과 유사합니다. 하지만 역할은 사용자 한 명과 독자적으로 연결되는 것이 아니기 때문에 필요한 모든 객체에게 전달할 수 있습니다. 또한 역할과 연결되

는 자격 증명(암호 또는 액세스 키)도 전혀 없습니다. 다만 역할이 클러스터와 연결되면 액세스 키가 동적으로 생성되어 클러스터에게 제공됩니다.

역할 기반 액세스 제어는 AWS 자격 증명을 보호하는 것은 물론이고 AWS 리소스와 민감한 사용자 데이터에 대한 액세스를 더욱 안전하게, 그리고 세분화하여 제어할 수 있다는 점에서 더욱 바람직합니다.

역할 기반 인증의 이점은 다음과 같습니다.

- AWS 표준 IAM 도구를 사용하여 IAM 역할을 정의한 후 다수의 클러스터와 연결할 수 있습니다. 역할 액세스 정책을 수정할 경우에는 변경 사항이 역할을 사용하는 모든 클러스터에 자동 적용됩니다.
- 특정 클러스터 및 데이터베이스 사용자에게 원하는 AWS 리소스와 작업에 대한 액세스 권한을 부여하는 IAM 정책을 세분화하여 정의할 수 있습니다.
- 클러스터가 실행 시간에 임시 세션 자격 증명을 가져온 후 작업이 끝날 때까지 필요에 따라 자격 증명을 새롭게 변경합니다. 키 기반 임시 자격 증명을 사용하는 경우에는 작업을 마치기 전에 임시 자격 증명이 만료되면 작업이 중단됩니다.
- 액세스 키 ID와 보안 액세스 키 ID는 저장되지도 않고 SQL 코드로 전송되지도 않습니다.

역할 기반 액세스 제어를 사용하려면 먼저 Amazon Redshift 서비스 역할 유형을 사용하여 IAM 역할을 생성한 후 클러스터에 연결해야 합니다. 이 역할에는 최소한 COPY, UNLOAD 및 CREATE LIBRARY 작업을 위한 IAM 권한에 나열된 권한이 있어야 합니다. IAM 역할을 생성하여 클러스터에 연결하는 단계는 Amazon Redshift 관리 가이드의 사용자를 대신하여 기타 AWS 서비스에 액세스하도록 Amazon Redshift에 권한 부여 섹션을 참조하세요.

Amazon Redshift 관리 콘솔, CLI 또는 API를 사용하여 역할을 클러스터에 추가하거나, 클러스터와 연결된 역할을 확인할 수 있습니다. 자세한 내용은 Amazon Redshift 관리 가이드의 클러스터와 IAM 역할 연결 섹션을 참조하세요.

IAM 역할을 생성하면 IAM이 생성된 역할의 Amazon 리소스 이름(ARN)을 반환합니다. IAM 역할을 지정하려면 IAM_ROLE 파라미터 또는 CREDENTIALS 파라미터에 이 역할 ARN을 입력해야 합니다.

예를 들어 클러스터에 다음과 같은 역할이 연결되어 있다고 가정하겠습니다.

```
"IamRoleArn": "arn:aws:iam::0123456789012:role/MyRedshiftRole"
```

다음은 위의 예에서 Amazon S3에 대한 인증 및 액세스를 위해 IAM_ROLE 파라미터에 ARN을 사용하는 COPY 명령 예입니다.

```
copy customer from 's3://amzn-s3-demo-bucket/mydata'
```

```
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

다음은 CREDENTIALS 파라미터를 사용하여 IAM 역할을 지정하는 COPY 명령 예입니다.

```
copy customer from 's3://amzn-s3-demo-bucket/mydata'
credentials
'aws_iam_role=arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

또한 슈퍼 사용자는 데이터베이스 사용자 및 그룹에 ASSUMEROLE 권한을 부여하여 COPY 작업을 위한 역할에 대한 액세스 권한을 제공할 수 있습니다. 자세한 내용은 GRANT을 참조하세요.

키 기반 액세스 제어

키 기반 액세스 제어의 경우 데이터가 포함된 AWS 리소스에 액세스할 권한이 부여된 IAM 사용자에게 액세스 키 ID와 비밀 액세스 키를 제공합니다. ACCESS_KEY_ID and SECRET_ACCESS_KEY 파라미터를 함께, 혹은 CREDENTIALS 파라미터를 사용할 수 있습니다.

> **Note**
>
> 액세스 키 ID와 비밀 액세스 키를 일반 텍스트 형태로 입력하기보다는 IAM 역할을 인증에 사용하는 것을 강력 권장합니다. 키 기반 액세스 제어를 선택하는 경우에는 절대 AWS 계정(루트) 자격 증명을 사용하지 마세요. 항상 IAM 사용자를 먼저 생성한 후 해당 사용자의 액세스 키 ID와 비밀 액세스 키를 입력합니다. IAM 사용자를 생성하는 단계는 AWS 계정의 IAM 사용자 생성을 참조하세요.

ACCESS_KEY_ID와 SECRET_ACCESS_KEY를 사용하여 인증하려면 다음 예와 같이 *<access-key-id>* 및 *<secret-access-key>*에 권한이 부여된 사용자의 액세스 키 ID와 전체 보안 액세스 키를 입력합니다.

```
ACCESS_KEY_ID '<access-key-id>'
SECRET_ACCESS_KEY '<secret-access-key>';
```

CREDENTIALS 파라미터를 사용하여 인증하려면 다음 예와 같이 *<access-key-id>* 및 *<secret-access-key>*에 권한이 부여된 사용자의 액세스 키 ID와 전체 보안 액세스 키를 입력합니다.

```
CREDENTIALS
'aws_access_key_id=<access-key-id>;aws_secret_access_key=<secret-access-key>';
```

IAM 사용자는 최소한 COPY, UNLOAD 및 CREATE LIBRARY 작업을 위한 IAM 권한에 나열된 권한이 있어야 합니다.

임시 보안 자격 증명

키 기반 액세스 제어를 사용하는 경우에는 임시 보안 자격 증명을 통해 데이터에 대한 사용자 액세스를 추가로 제한할 수 있습니다. 역할 기반 인증은 임시 자격 증명을 자동으로 사용합니다.

> **Note**
>
> 임시 자격 증명을 생성하여 액세스 키 ID와 보안 액세스 키를 일반 텍스트 형태로 입력하기보다는 role-based access control를 사용하는 것이 더욱 바람직합니다. 역할 기반 액세스 제어는 자동으로 임시 자격 증명을 사용합니다.

임시 보안 자격 증명은 유효 기간이 짧고 만료 후 재사용이 불가능하기 때문에 보안을 강화하는 효과가 있습니다. 토큰으로 생성된 액세스 키 ID와 비밀 액세스 키는 토큰 없이 사용할 수 없으며, 이러한 임시 보안 자격 증명이 발급된 사용자는 자격 증명이 만료될 때까지만 리소스에 액세스할 수 있습니다.

사용자에게 리소스에 대한 임시 액세스 권한을 부여할 때는 AWS STS(AWS Security Token Service) API 작업을 호출합니다. AWS STS API 작업은 보안 토큰, 액세스 키 ID 및 보안 액세스 키로 구성된 임시 보안 자격 증명을 반환합니다. 임시 보안 자격 증명은 리소스에 대한 일시적으로 액세스가 필요한 사용자에게 발급합니다. 이러한 사용자들은 기존 IAM 사용자일 수도 있고, 혹은 AWS를 사용하지 않는 사용자일 수도 있습니다. 임시 보안 자격 증명 생성에 대한 자세한 내용은 IAM User Guide의 Using Temporary Security Credentials 섹션을 참조하세요.

ACCESS_KEY_ID and SECRET_ACCESS_KEY 파라미터를 SESSION_TOKEN 파라미터 또는 CREDENTIALS 파라미터와 함께 사용할 수 있습니다. 또한 토큰과 함께 제공되는 액세스 키 ID와 보안 액세스 키를 입력해야 합니다.

ACCESS_KEY_ID, SECRET_ACCESS_KEY 및 SESSION_TOKEN을 사용하여 인증하려면 다음 예와 같이 *<temporary-access-key-id>*, *<temporary-secret-access-key>*, *<temporary-token>*에 해당하는 자격 증명을 입력합니다.

```
ACCESS_KEY_ID '<temporary-access-key-id>'
SECRET_ACCESS_KEY '<temporary-secret-access-key>'
SESSION_TOKEN '<temporary-token>';
```

CREDENTIALS를 사용하여 인증하려면 다음 예와 같이 자격 증명 문자열에 session_token=*<temporary-token>*을 입력합니다.

```
CREDENTIALS
'aws_access_key_id=<temporary-access-key-id>;aws_secret_access_key=<temporary-secret-access-key>;session_token=<temporary-token>';
```

다음은 임시 보안 자격 증명을 사용하는 COPY 명령 예입니다.

```
copy table-name
from 's3://objectpath'
access_key_id '<temporary-access-key-id>'
secret_access_key '<temporary-secret-access-key>'
session_token '<temporary-token>';
```

다음은 임시 자격 증명과 파일 암호화를 사용해 LISTING 테이블에 로드하는 예입니다.

```
copy listing
from 's3://amzn-s3-demo-bucket/data/listings_pipe.txt'
access_key_id '<temporary-access-key-id>'
secret_access_key '<temporary-secret-access-key>'
session_token '<temporary-token>'
master_symmetric_key '<root-key>'
encrypted;
```

다음은 CREDENTIALS 파라미터에서 임시 자격 증명과 파일 암호화를 사용해 LISTING 테이블에 로드하는 예입니다.

```
copy listing
from 's3://amzn-s3-demo-bucket/data/listings_pipe.txt'
credentials
'aws_access_key_id=<temporary-access-key-id>;aws_secret_access_key=<temporary-secret-access-key>;session_token=<temporary-token>;master_symmetric_key=<root-key>'
encrypted;
```

Important

임시 보안 자격 증명은 COPY 또는 UNLOAD 작업이 완전히 끝날 때까지 유효해야 합니다. 작업 도중 임시 보안 자격 증명이 만료되면 명령 중단과 함께 트랜잭션이 롤백됩니다. 예를 들어 임시 보안 자격 증명이 15분 후 만료되는데 필요한 COPY 작업 시간이 1시간이라면 COPY 작업이 완료 이전에 중단됩니다. 하지만 역할 기반 액세스를 사용하는 경우에는 작업을 마치기 전에 임시 보안 자격 증명이 새롭게 자동 업데이트됩니다.

COPY, UNLOAD 및 CREATE LIBRARY 작업을 위한 IAM 권한

CREDENTIALS 파라미터에서 참조하는 IAM 역할 또는 사용자는 최소한 다음 권한을 가지고 있어야 합니다.

- Amazon S3에서 COPY의 경우 Amazon S3 버킷에 대한 LIST 및 로드 중인 Amazon S3 객체와 매니페스트 파일(사용 중인 경우)에 대한 GET 권한.
- JSON 형식 데이터가 있는 Amazon S3, Amazon EMR 및 원격 호스트(SSH)에서 COPY를 지원하는 경우 Amazon S3의 JSONPaths 파일(사용하는 경우)에 대한 LIST 및 GET 권한.
- DynamoDB에서 COPY의 경우 로드 중인 DynamoDB 테이블에 대한 SCAN 및 DESCRIBE 권한.
- Amazon EMR 클러스터에서 COPY의 경우 Amazon EMR 클러스터에 대한 `ListInstances` 작업 권한.
- Amazon S3로 UNLOAD의 경우 데이터 파일을 언로드 중인 Amazon S3 버킷에 대한 GET, LIST 및 PUT 권한.
- Amazon S3에서 CREATE LIBRARY의 경우 Amazon S3 버킷에 대한 LIST 및 가져오고 있는 Amazon S3 객체에 대한 GET 권한.

> **Note**
>
> COPY, UNLOAD 또는 CREATE LIBRARY 명령 실행 시 오류 메시지로 `S3ServiceException: Access Denied`가 수신되면 클러스터에 Amazon S3에 대한 액세스 권한이 없는 것입니다.

IAM 권한은 클러스터, 사용자 또는 사용자가 속한 그룹에 연결되어 있는 IAM 역할에 IAM 정책을 연결하여 관리할 수 있습니다. 예를 들어 `AmazonS3ReadOnlyAccess` 관리형 정책은 Amazon S3 리소스에 LIST 및 GET 권한을 부여합니다. IAM 정책에 대한 자세한 내용은 IAM User Guide의 Managing IAM Policies 섹션을 참조하세요.

Amazon S3 액세스 포인트 별칭과 함께 COPY 사용

COPY는 Amazon S3 액세스 포인트 별칭을 지원합니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 액세스 포인트에 버킷 스타일 별칭 사용을 참조하세요.

Amazon S3에서 멀티바이트 데이터 로드

데이터에 중국어나 키릴 문자 같이 ASCII가 아닌 멀티바이트 문자가 포함되어 있는 경우에는 VARCHAR 열에 데이터를 로드해야 합니다. VARCHAR 데이터 형식은 4바이트 UTF-8 문자를 지원하

지만 CHAR 데이터 형식에서는 1바이트 ASCII 문자만 허용되기 때문입니다. Amazon Redshift 테이블에 5바이트 이상의 문자는 로드할 수 없습니다. 자세한 내용은 멀티바이트 문자 단원을 참조하십시오.

GEOMETRY 또는 GEOGRAPHY 데이터 유형의 열 로드

CSV 파일과 같이 문자로 구분된 텍스트 파일의 데이터에서 GEOMETRY 또는 GEOGRAPHY 열로 COPY가 가능합니다. 데이터는 잘 알려진 이진 형식(WKB 또는 EWKB) 또는 잘 알려진 텍스트 형식(WKT 또는 EWKT)의 16진수 형식이어야 하며 COPY 명령에 대한 단일 입력 행의 최대 크기에 맞아야 합니다. 자세한 내용은 COPY 단원을 참조하십시오.

shapefile에서 로드하는 방법에 대한 자세한 내용은 Amazon Redshift에 shapefile 로드 섹션을 참조하세요.

GEOMETRY 또는 GEOGRAPHY 데이터 유형에 대한 자세한 내용은 Amazon Redshift에서 공간 데이터 쿼리섹션을 참조하세요.

HLLSKETCH 데이터 형식 로드

HLL 스케치는 Amazon Redshift에서 지원하는 희소 또는 밀집 형식으로만 복사할 수 있습니다. HyperLogLog 스케치에서 COPY 명령을 사용하려면 밀접 HyperLogLog 스케치에 Base64 형식을 사용하고 희소 HyperLogLog 스케치에 JSON 형식을 사용합니다. 자세한 내용은 HyperLogLog 함수 단원을 참조하십시오.

다음 예에서는 CREATE TABLE 및 COPY를 사용하여 CSV 파일의 데이터를 테이블로 가져옵니다. 먼저 이 예에서는 CREATE TABLE을 사용하여 테이블 t1을 생성합니다.

```
CREATE TABLE t1 (sketch hllsketch, a bigint);
```

그런 다음 COPY를 사용하여 CSV 파일의 데이터를 테이블 t1로 가져옵니다.

```
COPY t1 FROM s3://amzn-s3-demo-bucket/unload/' IAM_ROLE
 'arn:aws:iam::0123456789012:role/MyRedshiftRole' NULL AS 'null' CSV;
```

VARBYTE 데이터 유형의 열 로드

CSV, Parquet, ORC 형식의 파일에서 데이터를 로드할 수 있습니다. CSV의 경우 데이터는 파일에서 VARBYTE 데이터의 16진수 표현으로 로드됩니다. FIXEDWIDTH 옵션을 사용하여 VARBYTE 데이터를 로드할 수 없습니다. COPY의 ADDQUOTES 또는 REMOVEQUOTES 옵션은 지원되지 않습니다. VARBYTE 열은 파티션 열로 사용할 수 없습니다.

다수의 파일을 읽어올 때 발생하는 오류

COPY 명령은 원자성을 띤 트랜잭션으로 처리됩니다. 다시 말해서 COPY 명령이 다수의 파일에서 데이터를 읽어올 때도 전체 프로세스가 하나의 트랜잭션으로 처리됩니다. COPY가 파일을 읽어오다 오류가 발생하면 프로세스 시간 제한에 이를 때까지(statement_timeout 참조) 자동으로 재시도합니다. 또는 Amazon S3에서 장시간(15~30분) 데이터를 다운로드할 수 없는 경우에도 마찬가지입니다. 따라서 각 파일마다 반드시 단 한 번 로드됩니다. COPY 명령이 실패하면 전체 트랜잭션이 취소되고 모든 변경 사항이 롤백됩니다. 로드 오류 처리에 대한 자세한 내용은 데이터 로드 문제 해결 섹션을 참조하세요.

COPY 명령이 성공적으로 시작되면 클라이언트가 분리되는 등 세션이 종료되더라도 중단되지 않습니다. 하지만 COPY 명령이 세션 종료로 인해 아직 완료되지 않은 BEGIN … END 트랜잭션 블록 내에서 정체되면 COPY를 포함해 전체 트랜잭션이 롤백됩니다. 버전 관리에 대한 자세한 내용은 BEGIN 단원을 참조하십시오.

JSON 형식의 COPY 지원

JSON 데이터 구조는 객체 또는 배열 집합으로 구성됩니다. JSON 객체는 중괄호로 시작해서 끝나며 이름-값 페어 집합이 순서에 상관없이 포함되어 있습니다. 각 이름과 값은 각각 콜론으로 구분되며, 페어는 서로 쉼표로 구분됩니다. 이름은 큰 따옴표로 묶이는 문자열입니다. 인용 부호는 기울어진 따옴표나 "스마트" 따옴표가 아닌 단순한 따옴표(0x22)여야 합니다.

JSON 배열은 대괄호로 시작해서 끝나며 쉼표로 구분된 값 집합이 순서에 따라 포함되어 있습니다. 값은 큰 따옴표로 묶이는 문자열, 숫자, 부울(true 또는 false), NULL, JSON 객체 또는 배열이 될 수 있습니다.

JSON 객체와 배열은 중첩이 가능하여 계층적인 데이터 구조를 이룹니다. 다음은 유효한 객체 2개가 포함된 JSON 데이터 구조의 예입니다.

```
{
    "id": 1006410,
    "title": "Amazon Redshift Database Developer Guide"
}
{
    "id": 100540,
    "name": "Amazon Simple Storage Service User Guide"
}
```

다음은 두 개의 JSON 배열과 동일한 데이터를 보여 줍니다.

```
[
```

```
    1006410,
    "Amazon Redshift Database Developer Guide"
]
[
    100540,
    "Amazon Simple Storage Service User Guide"
]
```

JSON에 대한 COPY 옵션

JSON 형식 데이터와 함께 COPY를 사용할 경우 다음 옵션을 지정할 수 있습니다.

- `'auto'` - COPY가 JSON 파일에서 필드를 자동으로 로드합니다.
- `'auto ignorecase'` - COPY가 필드 이름의 대/소문자를 무시하면서 JSON 파일에서 필드를 자동으로 로드합니다.
- `s3://jsonpaths_file` - COPY가 JSONPaths 파일을 사용하여 JSON 원본 데이터를 구문 분석합니다. JSONPaths 파일은 이름 `"jsonpaths"`가 JSONPath 표현식 배열과 쌍을 이루는 단일 JSON 객체가 포함된 텍스트 파일입니다. 이때 이름이 `"jsonpaths"`가 아닌 다른 문자열이면 COPY는 JSONPaths 파일 대신 `'auto'` 인수를 사용합니다.

`'auto'`, `'auto ignorecase'` 또는 JSONPaths 파일을 사용하여 데이터를 로드하는 방식과 JSON 객체 또는 배열을 사용하여 데이터를 로드하는 방법을 나타낸 예는 JSON에서 복사 예제 섹션을 참조하세요.

JSONPath 옵션

Amazon Redshift COPY 구문에서 JSONPath 표현식은 대괄호 또는 점 표기법을 사용하여 JSON 계층적 데이터 구조의 단일 이름 요소에 대한 명시적 경로를 지정합니다. Amazon Redshift는 모호한 경로 또는 여러 이름 요소로 해석될 수 있는 와일드카드 문자 또는 필터 표현식과 같은 JSONPath 요소를 지원하지 않습니다. 결과적으로 Amazon Redshift는 복잡한 다단 데이터 구조의 구문을 분석할 수 없습니다.

다음은 대괄호 표기법을 사용하여 JSONPath 표현식을 작성한 JSONPaths 파일의 예입니다. 달러 기호($)는 루트 레벨 구조를 나타냅니다.

```
{
    "jsonpaths": [
       "$['id']",
       "$['store']['book']['title']",
```

```
 "$['location'][0]"
    ]
}
```

위 예에서 `$['location'][0]`은 배열에서 첫 번째 요소를 참조합니다. JSON은 0부터 시작되는 배열 인덱싱을 사용합니다. 배열 인덱스는 0보다 크거나 같은 양의 정수가 되어야 합니다.

다음은 위의 JSONPaths 파일에 점 표기법을 사용한 예입니다.

```
{
    "jsonpaths": [
       "$.id",
       "$.store.book.title",
 "$.location[0]"
    ]
}
```

jsonpaths 배열에서 대괄호 표기법과 점 표기법을 혼용할 수는 없습니다. 다만 배열 요소를 참조할 때는 대괄호 표기법과 점 표기법 모두에서 대괄호를 사용할 수 있습니다.

점 표기법을 사용할 경우에는 JSONPath 표현식에 다음 문자가 포함되어서는 안 됩니다.

- 작은 직선형 따옴표(')
- 마침표 또는 점(.)
- 배열 요소를 참조하기 위한 경우를 제외한 대괄호([ ])

JSONPath 표현식이 참조하는 이름-값 페어의 값이 객체 또는 배열이면 중괄호나 대괄호를 포함하여 전체 객체 또는 배열이 문자열로 로드됩니다. 예를 들어 JSON 데이터에 다음과 같은 객체가 포함되어 있다고 가정하겠습니다.

```
{
    "id": 0,
    "guid": "84512477-fa49-456b-b407-581d0d851c3c",
    "isActive": true,
    "tags": [
        "nisi",
        "culpa",
        "ad",
        "amet",
        "voluptate",
```

```
        "reprehenderit",
        "veniam"
    ],
    "friends": [
        {
            "id": 0,
            "name": "Martha Rivera"
        },
        {
            "id": 1,
            "name": "Renaldo"
        }
    ]
}
```

그러면 JSONPath 표현식 `$['tags']`가 다음 값을 반환합니다.

```
"["nisi","culpa","ad","amet","voluptate","reprehenderit","veniam"]"
```

그러면 JSONPath 표현식 `$['friends'][1]`가 다음 값을 반환합니다.

```
"{"id": 1,"name": "Renaldo"}"
```

`jsonpaths` 배열의 JSONPath 표현식은 각각 Amazon Redshift 대상 테이블의 열 1개에 해당합니다. 따라서 `jsonpaths` 배열 요소의 순서는 대상 테이블의 열 순서와, 혹은 열 목록을 사용하는 경우 열 목록의 열 순서와 일치해야 합니다.

`'auto'` 인수 또는 JSONPaths 파일을 사용하여 데이터를 로드하는 방법과 JSON 객체 또는 배열을 사용하여 데이터를 로드하는 방법을 보여주는 예는 JSON에서 복사 예제 섹션을 참조하세요.

여러 JSON 파일을 복사하는 방법에 대한 자세한 내용은 매니페스트를 사용하여 데이터 파일 지정 섹션을 참조하세요.

JSON의 이스케이프 문자

COPY는 줄 바꿈 문자로 `\n`을, 그리고 탭 문자로 `\t`를 로드합니다. 백슬래시를 로드하려면 백슬래시(`\\`)로 이스케이프하십시오.

예를 들어 `escape.json` 버킷에 저장된 `s3://amzn-s3-demo-bucket/json/`이라는 이름의 파일에 다음과 같은 JSON이 있다고 가정하겠습니다.

```
{
  "backslash": "This is a backslash: \\",
  "newline": "This sentence\n is on two lines.",
  "tab": "This sentence \t contains a tab."
}
```

다음 명령을 실행하여 ESCAPES 테이블을 생성하고 JSON을 로드합니다.

```
create table escapes (backslash varchar(25), newline varchar(35), tab varchar(35));

copy escapes from 's3://amzn-s3-demo-bucket/json/escape.json'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as json 'auto';
```

ESCAPES 테이블에 대한 쿼리를 실행하여 결과를 확인합니다.

```
select * from escapes;

       backslash        |      newline      |               tab
------------------------+-------------------+----------------------------------
 This is a backslash: \ | This sentence     | This sentence    contains a tab.
                        :  is on two lines.
(1 row)
```

숫자 정밀도 상실

JSON 형식의 데이터 파일에서 숫자 데이터 형식으로 정의된 열로 숫자를 로드할 경우 정밀도가 상실될 수 있습니다. 일부 부동 소수점 값이 컴퓨터 시스템에 정확하게 표시되지 않습니다. 따라서 JSON 파일에서 복사하는 데이터가 예상한 대로 반올림되지 않을 수 있습니다. 정밀도 상실을 방지하려면 다음 대안 중 하나를 사용하는 것이 좋습니다.

- 값을 큰 따옴표로 묶어서 숫자를 문자열로 표시합니다.
- ROUNDEC를 사용하여 숫자를 자르지 않고 반올림합니다.
- JSON 또는 Avro 파일 대신 CSV, 문자로 구분된 형식 또는 고정 너비 텍스트 파일을 사용합니다.

열 기반 데이터 형식에서 COPY 명령

COPY는 Amazon S3에서 다음과 같은 열 기반 형식의 데이터를 불러올 수 있습니다.

- ORC

- PARQUET

열 데이터 형식에서 COPY를 사용하는 예는 COPY 예 섹션을 참조하세요.

COPY는 열 기반 형식 데이터를 지원하지만 다음과 같은 고려 사항이 있습니다.

- Amazon S3 버킷이 Amazon Redshift 데이터베이스와 동일한 AWS 리전에 있어야 합니다.
- VPC 엔드포인트를 통해 Amazon S3 데이터에 액세스하려면 Amazon Redshift 관리 가이드의 Enhanced VPC Routing과 함께 Amazon Redshift Spectrum 사용에 설명된 대로 IAM 정책과 IAM 역할을 사용하여 액세스를 설정합니다.
- COPY는 압축 인코딩을 자동으로 적용하지 않습니다.
- 다음과 같은 COPY 파라미터만 지원됩니다.
  - ORC 또는 Parquet 파일에서 복사하는 경우 ACCEPTINVCHARS
  - FILLRECORD
  - FROM
  - IAM_ROLE
  - CREDENTIALS
  - STATUPDATE
  - MANIFEST
  - EXPLICIT_IDS
- COPY 명령을 실행하여 불러오는 중 오류가 발생하면 명령이 실패합니다. 열 기반 데이터 유형의 경우 ACCEPTANYDATE 및 MAXERROR가 지원되지 않습니다.
- 오류 메시지는 SQL 클라이언트로 전송됩니다. 일부 오류는 STL_LOAD_ERRORS 및 STL_ERROR에 기록됩니다.
- COPY 명령은 열 기반 데이터 파일의 열 순서와 동일하게 대상 테이블의 열에 값을 삽입합니다. 대상 테이블의 열 수와 데이터 파일의 열 수는 일치해야 합니다.
- COPY 작업에 대해 지정한 파일에 다음 확장자 중 하나가 포함되어 있는 경우 파라미터를 추가하지 않고 데이터의 압축을 해제할 수 있습니다.
  - `.gz`
  - `.snappy`
  - `.bz2`
- Parquet 및 ORC 파일 형식에서 COPY는 Redshift Spectrum과 버킷 액세스를 사용합니다. 이러한 형식에 COPY를 사용하려면 Amazon S3의 미리 서명된 URL 사용을 차단하는 IAM 정책이 없어야

합니다. Amazon Redshift에서 생성한 미리 서명된 URL은 1시간 동안 유효하므로 Amazon Redshift가 충분한 시간을 갖고 Amazon S3 버킷에서 모든 파일을 로드할 수 있습니다. 열 기반 데이터 형식에서 COPY로 스캔한 각 파일에 대해 미리 서명된 고유한 URL이 생성됩니다. `s3:signatureAge` 작업이 포함된 버킷 정책의 경우 값을 최소 3,600,000밀리초로 설정해야 합니다. 자세한 내용은 Enhanced VPC Routing과 함께 Amazon Redshift Spectrum 사용 섹션을 참조하세요.

- REGION 파라미터는 열 데이터 형식의 COPY에서는 지원되지 않습니다. Amazon S3 버킷과 데이터베이스가 동일한 AWS 리전에 있더라도 PARQUET 기반 COPY에는 REGION 인수가 지원되지 않음 등의 오류가 발생할 수 있습니다.

DATEFORMAT 및 TIMEFORMAT 문자열

COPY 명령은 DATEFORMAT 및 TIMEFORMAT 옵션을 사용하여 소스 데이터의 날짜 및 시간 값을 구문 분석합니다. DATEFORMAT 및 TIMEFORMAT은 소스 데이터의 날짜 및 시간 값 형식과 일치해야 하는 형식이 지정된 문자열입니다. 예를 들어 날짜 값이 `Jan-01-1999`인 소스 데이터를 로드하는 COPY 명령에는 다음 DATEFORMAT 문자열이 포함되어야 합니다.

```
COPY ...
            DATEFORMAT AS 'MON-DD-YYYY'
```

COPY 데이터 변환 관리에 대한 자세한 내용은 데이터 변환 파라미터를 참조하세요.

DATEFORMAT 및 TIMEFORMAT 문자열은 날짜/시간 구분 기호(예: '-', '/' 또는 ':')와 다음 테이블의 날짜 부분 및 시간 부분 형식을 포함할 수 있습니다.

> **Note**
>
> 날짜 또는 시간 값의 형식을 다음 날짜 부분 및 시간 부분과 일치시킬 수 없거나 서로 다른 형식을 사용하는 날짜 및 시간 값이 있는 경우 DATEFORMAT 또는 TIMEFORMAT 파라미터와 함께 `'auto'` 인수를 사용하세요. `'auto'` 인수는 DATEFORMAT 또는 TIMEFORMAT 문자열 사용 시 지원되지 않는 몇 가지 형식을 인식합니다. 자세한 내용은 DATEFORMAT 및 TIMEFORMAT 옵션의 자동 인식 사용 단원을 참조하십시오.

날짜 부분 또는 시간 부분	의미
YY	세기를 제외한 연도

날짜 부분 또는 시간 부분	의미
YYYY	세기를 포함한 연도
MM	숫자 형태의 월
MON	이름 형태의 월(약어 또는 전체 이름)
DD	숫자 형태의 월중 일
HH 또는 HH24	시간(24시간 방식) **Note** SQL 함수의 DATETIME 형식 문자열에서 HH는 HH12와 동일합니다. 그러나 COPY의 DATEFORMAT 및 TIMEFORMAT 문자열에서 HH는 HH24와 동일합니다.
HH12	시간(12시간 방식)
MI	분
SS	초
AM(오전) 또는 PM(오후)	오전/오후 표시자 (12시 방식일 때)

기본 날짜 형식은 YYYY-MM-DD입니다. 시간대(TIMESTAMP)를 제외한 기본 타임스탬프 형식은 YYYY-MM-DD HH:MI:SS입니다. 시간대가 포함된 기본 타임스탬프(TIMESTAMPTZ) 형식은 YYYY-MM-DD HH:MI:SSOF입니다. 여기서 OF는 UTC의 오프셋입니다(예: -8:00). timeformat_string에 시간대 지정자(TZ, tz 또는 OF)를 포함할 수 없습니다. 초(SS) 필드는 마이크로초 수준의 세부 정보까지 소수 초를 지원합니다. 기본 형식과 다른 형식으로 TIMESTAMPTZ 데이터를 로드하려면 'auto'를 지정합니다.

다음은 소스 데이터에서 볼 수 있는 몇 가지 샘플 날짜 또는 시간과 그에 해당하는 DATEFORMAT 또는 TIMEFORMAT 문자열입니다.

소스 데이터 날짜 또는 시간의 예	DATEFORMAT 또는 TIMEFORMAT 구문
03/31/2003	DATEFORMAT AS 'MM/DD/YYYY'
2003년 3월 31일	DATEFORMAT AS 'MON DD, YYYY'
03.31.2003 18:45:05 03.31.2003 18:45:05.123456	TIMEFORMAT AS 'MM.DD.YYYY HH:MI:SS'

예제

TIMEFORMAT 사용 예는 타임스탬프 또는 데이트스탬프 로드을 참조하십시오.

DATEFORMAT 및 TIMEFORMAT 옵션의 자동 인식 사용

DATEFORMAT 또는 TIMEFORMAT 파라미터의 인수로 'auto'를 지정하면 Amazon Redshift가 원본 데이터의 날짜 형식 또는 시간 형식을 자동으로 인식하여 변환합니다. 다음은 그 한 예입니다.

```
copy favoritemovies from 'dynamodb://ProductCatalog'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
dateformat 'auto';
```

DATEFORMAT 및 TIMEFORMAT 파라미터에서 'auto' 인수를 사용하면 COPY가 DATEFORMAT 및 TIMEFORMAT 문자열 단원의 표에 나열된 날짜 및 시간 형식을 인식하여 변환합니다. 또한 'auto' 인수는 DATEFORMAT 및 TIMEFORMAT 문자열 사용 시 아래와 같이 지원되지 않는 몇 가지 형식을 인식합니다.

형식	유효한 입력 문자열의 예
ISO 8601	2019-02-11T05:09:12.195Z
Julian	J2451187
BC	Jan-08-95 BC

형식	유효한 입력 문자열의 예
YYYYMMDD HHMISS	19960108 040809
YYMMDD HHMISS	960108 040809
YYYY.DDD	1996.008
YYYY-MM-DD HH:MI:SS.SSS	1996-01-08 04:05:06.789
DD Mon HH:MI:SS YYYY TZ	17 Dec 07:37:16 1997 PST
MM/DD/YYYY HH:MI:SS.SS TZ	12/17/1997 07:37:16.00 PST
YYYY-MM-DD HH:MI:SS+/-TZ	1997-12-17 07:37:16-08
DD.MM.YYYY HH:MI:SS TZ	12.17.1997 07:37:16.00 PST

자동 인식은 epochsec 및 epochmillisec을 지원하지 않습니다.

날짜 또는 타임스탬프 값의 자동 변환 여부를 테스트하려면 CAST 함수를 사용하여 문자열을 날짜 또는 타임스탬프 값으로 변환해볼 수 있습니다. 예를 들어 다음은 타임스탬프 값 `'J2345678 04:05:06.789'`를 테스트하는 명령입니다.

```
create table formattest (test char(21));
insert into formattest values('J2345678 04:05:06.789');
select test, cast(test as timestamp) as timestamp, cast(test as date) as date from
 formattest;

        test          |      timestamp      | date
----------------------+---------------------+------------
J2345678 04:05:06.789   1710-02-23 04:05:06 1710-02-23
```

DATE 열의 원본 데이터에 시간 정보가 포함되면 시간 구성요소는 잘립니다. TIMESTAMP 열의 원본 데이터에서 시간 정보가 빠지면 시간 구성요소로 00:00:00이 사용됩니다.

COPY 예

> **Note**
>
> 여기에서 설명하는 예는 가독성을 위해 줄 바꿈이 포함되었습니다. 실제 credentials-args 문자열에서는 줄 바꿈이나 공백을 입력하지 마십시오.

주제

DynamoDB 테이블에서 FAVORITEMOVIES 로드

AWS SDK에는 Movies라는 DynamoDB 테이블을 생성하는 간단한 예가 포함되어 있습니다. 이러한 예제는 DynamoDB 시작하기를 참조하십시오. 다음은 DynamoDB 테이블의 데이터를 Amazon Redshift MOVIES 테이블에 로드하는 예입니다. Amazon Redshift 테이블은 사전에 데이터베이스에 존재해야 하며,

```
copy favoritemovies from 'dynamodb://Movies'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
readratio 50;
```

Amazon S3 버킷에서 LISTING 로드

다음 예는 Amazon S3 버킷에서 LISTING을 로드합니다. COPY 명령은 `/data/listing/` 폴더에 위치한 모든 파일을 로드합니다.

```
copy listing
from 's3://amzn-s3-demo-bucket/data/listing/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

Amazon EMR 클러스터에서 LISTING 로드

다음은 Amazon EMR 클러스터의 lzop 압축 파일에서 탭으로 구분된 데이터를 SALES 테이블과 함께 로드하는 예입니다. COPY는 `myoutput/` 폴더에서 `part-`로 시작하는 모든 파일을 로드합니다.

```
copy sales
from 'emr://j-SAMPLE2B500FC/myoutput/part-*'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '\t' lzop;
```

다음은 Amazon EMR 클러스터에서 JSON 형식 데이터를 SALES 테이블과 함께 로드하는 예입니다. COPY는 `myoutput/json/` 폴더에서 모든 파일을 로드합니다.

```
copy sales
from 'emr://j-SAMPLE2B500FC/myoutput/json/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
JSON 's3://amzn-s3-demo-bucket/jsonpaths.txt';
```

매니페스트를 사용하여 데이터 파일 지정

매니페스트를 사용하여 COPY 명령이 Amazon S3에서 필요한 파일을 모두, 즉 필요한 파일만 로드할 수 있습니다. 그 밖에 동일한 접두사를 공유하지 않는 다른 버킷이나 파일에서 다수의 파일을 로드해야 할 때도 매니페스트를 사용할 수 있습니다.

예를 들어 custdata1.txt, custdata2.txt, custdata3.txt 등 파일 3개를 로드해야 하는 경우 다음 명령을 사용하여 접두사를 지정함으로써 amzn-s3-demo-bucket에서 custdata로 시작하는 파일을 모두 로드할 수 있습니다.

```
copy category
from 's3://amzn-s3-demo-bucket/custdata'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

실수로 파일이 2개만 존재하는 경우에는 COPY가 해당 파일 2개만 로드하고 명령을 성공적으로 마치지만 결과적으로는 불완전 데이터 로드가 됩니다. 불필요하지만 동일한 접두사를 사용하는 파일(예: custdata.backup 파일)이 버킷에 있다면 COPY가 해당 파일까지 로드하므로 결국 원하지 않는 데이터가 로드됩니다.

필요한 파일은 모두 로드하고 원하지 않는 파일은 로드되지 않도록 하려면 매니페스트 파일을 사용합니다. 매니페스트란 COPY 명령에서 처리할 파일이 나열된 JSON 형식의 파일을 말합니다. 예를 들어 다음 매니페스트는 위의 예에서 파일 3개를 로드합니다.

```
{
   "entries":[
      {
         "url":"s3://amzn-s3-demo-bucket/custdata.1",
         "mandatory":true
      },
      {
         "url":"s3://amzn-s3-demo-bucket/custdata.2",
         "mandatory":true
      },
      {
         "url":"s3://amzn-s3-demo-bucket/custdata.3",
         "mandatory":true
      }
   ]
}
```

옵션인 mandatory 플래그는 파일이 없을 때 COPY의 종료 여부를 나타냅니다. 기본값은 false입니다. 필수 설정에 상관없이 파일이 검색되지 않으면 COPY가 종료됩니다. 이 예에서 파일 중 하나라도 검색되지 않으면 COPY가 오류를 반환합니다. 키 접두사만 지정하여 선택될 수도 있는 불필요한 파일(custdata.backup 등)은 매니페스트에 없기 때문에 무시됩니다.

ORC 또는 Parquet 형식의 데이터 파일에서 불러올 경우 다음 예에 나와 있는 것처럼 meta 필드가 필요합니다.

```
{
   "entries":[
      {
         "url":"s3://amzn-s3-demo-bucket1/orc/2013-10-04-custdata",
         "mandatory":true,
         "meta":{
            "content_length":99
         }
      },
      {
         "url":"s3://amzn-s3-demo-bucket2/orc/2013-10-05-custdata",
         "mandatory":true,
         "meta":{
            "content_length":99
         }
      }
   ]
}
```

다음 예는 cust.manifest라는 매니페스트를 사용합니다.

```
copy customer
from 's3://amzn-s3-demo-bucket/cust.manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as orc
manifest;
```

매니페스트를 이용하면 다른 버킷, 다른 리전이나 접두사가 다른 파일에서 파일을 불러올 수 있습니다. 다음은 이름이 데이트스탬프로 시작하는 파일의 데이터를 로드하는 JSON 예입니다.

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket/2013-10-04-custdata.txt","mandatory":true},
```

```
    {"url":"s3://amzn-s3-demo-bucket/2013-10-05-custdata.txt","mandatory":true},
    {"url":"s3://amzn-s3-demo-bucket/2013-10-06-custdata.txt","mandatory":true},
    {"url":"s3://amzn-s3-demo-bucket/2013-10-07-custdata.txt","mandatory":true}
  ]
}
```

매니페스트는 버킷이 클러스터와 동일한 AWS 리전에 속하는 한 다른 버킷에 있는 파일을 나열할 수 있습니다.

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket1/custdata1.txt","mandatory":false},
    {"url":"s3://amzn-s3-demo-bucket2/custdata1.txt","mandatory":false},
    {"url":"s3://amzn-s3-demo-bucket2/custdata2.txt","mandatory":false}
  ]
}
```

파이프로 구분된 파일(기본 구분자)에서 LISTING 로드

다음은 별도의 옵션 지정 없이 입력 파일에 기본 구분자인 파이프 문자('|')가 포함된, 매우 단순한 예입니다.

```
copy listing
from 's3://amzn-s3-demo-bucket/data/listings_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

Parquet 형식의 열 기반 데이터를 사용한 LISTING 로드

다음은 Amazon S3에서 parquet이라는 데이터를 로드하는 예입니다.

```
copy listing
from 's3://amzn-s3-demo-bucket/data/listings/parquet/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as parquet;
```

ORC 형식의 열 기반 데이터를 사용한 LISTING 로드

다음은 Amazon S3에서 orc라는 데이터를 로드하는 예입니다.

```
copy listing
```

```
from 's3://amzn-s3-demo-bucket/data/listings/orc/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as orc;
```

옵션을 사용한 EVENT 로드

다음은 파이프로 구분된 데이터를 EVENT 테이블에 로드하면서 아래 규칙을 적용하는 예입니다.

- 인용 부호 쌍을 사용하여 문자열을 묶는 경우 인용 부호가 삭제됩니다.
- 빈 문자열과 공백이 포함된 문자열 모두 NULL 값으로 로드됩니다.
- 오류가 6회 이상 반환되면 로드가 실패합니다.
- 타임스탬프 값은 지정된 형식을 따라야 합니다. 예를 들어 유효한 타임스탬프는 2008-09-26 05:43:12입니다.

```
copy event
from 's3://amzn-s3-demo-bucket/data/allevents_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
removequotes
emptyasnull
blanksasnull
maxerror 5
delimiter '|'
timeformat 'YYYY-MM-DD HH:MI:SS';
```

고정 폭 데이터 파일에서 VENUE 로드

```
copy venue
from 's3://amzn-s3-demo-bucket/data/venue_fw.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
fixedwidth 'venueid:3,venuename:25,venuecity:12,venuestate:2,venueseats:6';
```

위의 예는 데이터 파일이 샘플 데이터와 동일한 형식이라는 가정을 전제로 합니다. 아래 샘플에서는 모든 열이 명세에서 기록한 것과 동일한 폭을 갖도록 공백이 자리 표시자의 역할을 합니다.

```
1  Toyota Park              Bridgeview  IL0
2  Columbus Crew Stadium    Columbus    OH0
3  RFK Stadium              Washington  DC0
4  CommunityAmerica BallparkKansas City KS0
5  Gillette Stadium         Foxborough  MA68756
```

CSV 파일에서 CATEGORY 로드

다음과 같이 표에 기재된 값을 CATEGORY에 로드한다고 가정하겠습니다.

catid	catgroup	catname	catdesc
12	Shows	Musicals	Musical theatre
13	Shows	Plays	All "non-musical" theatre
14	Shows	Opera	All opera, light, and "rock" opera
15	Concerts	Classical	All symphony, concerto, and choir concerts

다음 예에서는 필드 값을 쉼표로 구분하여 텍스트 파일의 내용을 보여 줍니다.

```
12,Shows,Musicals,Musical theatre
13,Shows,Plays,All "non-musical" theatre
14,Shows,Opera,All opera, light, and "rock" opera
15,Concerts,Classical,All symphony, concerto, and choir concerts
```

쉼표로 구분된 입력을 지정하는 DELIMITER 파라미터를 사용하여 파일을 로드하면 일부 입력 필드에 쉼표가 포함되어 있기 때문에 COPY 명령이 실패합니다. 이러한 문제는 CSV 파라미터를 사용하고 쉼표가 포함된 필드를 인용 부호로 묶으면 피할 수 있습니다. 인용 부호가 다른 인용 부호로 묶인 문자열 내에 들어가는 경우에는 인용 부호를 이중으로 사용하여 이스케이프 처리해야 합니다. 기본 인용 부호가 큰따옴표이므로 큰따옴표를 추가하여 큰따옴표를 각각 이스케이프 처리해야 합니다. 새 입력 파일의 모양은 다음과 같습니다.

```
12,Shows,Musicals,Musical theatre
13,Shows,Plays,"All ""non-musical"" theatre"
14,Shows,Opera,"All opera, light, and ""rock"" opera"
15,Concerts,Classical,"All symphony, concerto, and choir concerts"
```

파일 이름이 `category_csv.txt`인 경우 다음 COPY 명령을 사용하여 파일을 로드할 수 있습니다.

```
copy category
from 's3://amzn-s3-demo-bucket/data/category_csv.txt'
```

```
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
csv;
```

그 밖에 입력 파일에서 큰따옴표의 이스케이프 처리를 피하려면 QUOTE AS 파라미터를 사용해 다른 인용 부호를 지정하는 방법도 있습니다. 예를 들어 다음 버전의 `category_csv.txt`는 '%'를 인용 부호로 사용합니다.

```
12,Shows,Musicals,Musical theatre
13,Shows,Plays,%All "non-musical" theatre%
14,Shows,Opera,%All opera, light, and "rock" opera%
15,Concerts,Classical,%All symphony, concerto, and choir concerts%
```

다음은 QUOTE AS를 사용하여 `category_csv.txt`를 로드하는 COPY 명령입니다.

```
copy category
from 's3://amzn-s3-demo-bucket/data/category_csv.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
csv quote as '%';
```

IDENTITY 열의 명시적인 값을 사용한 VENUE 로드

다음 예는 VENUE 테이블을 생성하면서 1개 이상의 열(`venueid` 열 등)을 IDENTITY 열로 지정했다는 가정을 전제로 합니다. 이 명령은 IDENTITY 열의 자동 생성 값에 대한 기본적인 IDENTITY 동작을 재정의하고 venue.txt 파일에서 명시적인 값을 로드합니다. Amazon Redshift는 EXLICIT_IDS 옵션을 사용할 때 테이블에 중복된 IDENTITY 값이 로드되었는지 확인하지 않습니다.

```
copy venue
from 's3://amzn-s3-demo-bucket/data/venue.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
explicit_ids;
```

파이프로 구분된 GZIP파일에서 TIME 로드

다음은 파이프로 구분된 GZIP 파일에서 TIME 테이블을 로드하는 예입니다.

```
copy time
from 's3://amzn-s3-demo-bucket/data/timerows.gz'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
gzip
```

```
delimiter '|';
```

타임스탬프 또는 데이트스탬프 로드

다음은 형식이 지정된 타임스탬프로 데이터를 로드하는 예입니다.

> **Note**
>
> HH:MI:SS 형식의 TIMEFORMAT 역시 SS를 넘어 마이크로초의 정밀도까지 소수 초를 지원합니다. 이번 예에서 사용하는 time.txt 파일에는 2009-01-12 14:15:57.119568 행 하나가 포함되어 있습니다.

```
copy timestamp1
from 's3://amzn-s3-demo-bucket/data/time.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
timeformat 'YYYY-MM-DD HH:MI:SS';
```

이 파일의 구조는 다음과 같습니다.

```
select * from timestamp1;
c1
----------------------------
2009-01-12 14:15:57.119568
(1 row)
```

파일에서 기본값을 사용한 데이터 로드

다음은 TICKIT 데이터베이스에서 변형된 VENUE 테이블을 사용하는 예입니다. 이때 VENUE_NEW 테이블을 아래와 같은 문으로 정의한다고 가정합니다.

```
create table venue_new(
venueid smallint not null,
venuename varchar(100) not null,
venuecity varchar(30),
venuestate char(2),
venueseats integer not null default '1000');
```

아래 예와 같이 venue_noseats.txt 데이터 파일에 VENUESEATS 열의 값이 포함되어 있지 않다고 가정합니다.

```
1|Toyota Park|Bridgeview|IL|
2|Columbus Crew Stadium|Columbus|OH|
3|RFK Stadium|Washington|DC|
4|CommunityAmerica Ballpark|Kansas City|KS|
5|Gillette Stadium|Foxborough|MA|
6|New York Giants Stadium|East Rutherford|NJ|
7|BMO Field|Toronto|ON|
8|The Home Depot Center|Carson|CA|
9|Dick's Sporting Goods Park|Commerce City|CO|
10|Pizza Hut Park|Frisco|TX|
```

다음은 데이터 파일에서 테이블을 성공적으로 로드한 후 누락된 열에 DEFAULT 값('1000')을 적용하는 COPY 문입니다.

```
copy venue_new(venueid, venuename, venuecity, venuestate)
from 's3://amzn-s3-demo-bucket/data/venue_noseats.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|';
```

그 결과 로드되는 테이블은 다음과 같습니다.

```
select * from venue_new order by venueid;
venueid |         venuename          |    venuecity    | venuestate | venueseats
---------+----------------------------+-----------------+------------+------------
1 | Toyota Park                | Bridgeview      | IL         |       1000
2 | Columbus Crew Stadium      | Columbus        | OH         |       1000
3 | RFK Stadium                | Washington      | DC         |       1000
4 | CommunityAmerica Ballpark  | Kansas City     | KS         |       1000
5 | Gillette Stadium           | Foxborough      | MA         |       1000
6 | New York Giants Stadium    | East Rutherford | NJ         |       1000
7 | BMO Field                  | Toronto         | ON         |       1000
8 | The Home Depot Center      | Carson          | CA         |       1000
9 | Dick's Sporting Goods Park | Commerce City   | CO         |       1000
10 | Pizza Hut Park             | Frisco          | TX         |       1000
(10 rows)
```

다음 예에서는 파일에 VENUESEATS 데이터가 포함되지 않았을 뿐만 아니라 VENUENAME 데이터도 포함되지 않았다고 가정합니다.

```
1||Bridgeview|IL|
2||Columbus|OH|
```

```
3||Washington|DC|
4||Kansas City|KS|
5||Foxborough|MA|
6||East Rutherford|NJ|
7||Toronto|ON|
8||Carson|CA|
9||Commerce City|CO|
10||Frisco|TX|
```

동일한 테이블 정의를 사용하는 경우 VENUENAME에 대한 DEFAULT 값이 지정되지 않았고 VENUENAME이 NOT NULL 열이므로 다음 COPY 문이 실패합니다.

```
copy venue(venueid, venuecity, venuestate)
from 's3://amzn-s3-demo-bucket/data/venue_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|';
```

이제 IDENTITY 열을 사용하는 다른 형태의 VENUE 테이블이 있다고 가정합니다.

```
create table venue_identity(
venueid int identity(1,1),
venuename varchar(100) not null,
venuecity varchar(30),
venuestate char(2),
venueseats integer not null default '1000');
```

앞의 예와 마찬가지로 원본 파일에 VENUESEATS 열에 해당하는 값이 없다고 가정하면 다음 COPY 문에서 사전 정의한 IDENTITY 데이터 값을 자동 생성하지 않고 포함하는 테이블을 성공적으로 로드합니다.

```
copy venue(venueid, venuename, venuecity, venuestate)
from 's3://amzn-s3-demo-bucket/data/venue_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|' explicit_ids;
```

다음 문은 IDENTITY 열 없이(VENUEID가 열 목록에서 누락됨) EXPLICIT_IDS 파라미터를 추가했기 때문에 중단됩니다.

```
copy venue(venuename, venuecity, venuestate)
from 's3://amzn-s3-demo-bucket/data/venue_pipe.txt'
```

```
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|' explicit_ids;
```

다음 문은 EXPLICIT_IDS 파라미터를 추가하지 않았기 때문에 중단됩니다.

```
copy venue(venueid, venuename, venuecity, venuestate)
from 's3://amzn-s3-demo-bucket/data/venue_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|';
```

ESCAPE 옵션을 사용한 데이터 COPY 작업

다음은 구분자 문자(여기에서는 파이프 문자)와 일치하는 문자의 로드 방법을 나타낸 예입니다. 먼저 입력 파일에서 로드할 파이프 문자(|)가 모두 백슬래시 문자(\)를 사용해 이스케이프 처리되어 있는지 확인합니다. 그런 다음 ESCAPE 파라미터를 사용해 파일을 로드합니다.

```
$ more redshiftinfo.txt
1|public\|event\|dwuser
2|public\|sales\|dwuser

create table redshiftinfo(infoid int,tableinfo varchar(50));

copy redshiftinfo from 's3://amzn-s3-demo-bucket/data/redshiftinfo.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|' escape;

select * from redshiftinfo order by 1;
infoid |       tableinfo
-------+--------------------
1      | public|event|dwuser
2      | public|sales|dwuser
(2 rows)
```

ESCAPE 파라미터를 사용하지 않으면 이 COPY 문은 `Extra column(s) found` 오류와 함께 중단됩니다.

> **Important**
>
> COPY를 ESCAPE 파라미터와 함께 사용하여 데이터를 로드하는 경우 역수 출력 파일을 생성하도록 UNLOAD 명령과 함께 ESCAPE 파라미터도 지정해야 합니다. 마찬가지로 ESCAPE 파

라미터를 사용하여 UNLOAD하는 경우 동일한 데이터를 COPY할 때 ESCAPE를 사용해야 합니다.

JSON에서 복사 예제

다음 예에서는 다음 데이터와 함께 CATEGORY 테이블을 로드합니다.

CATID	CATGROUP	CATNAME	CATDESC
1	스포츠	MLB	Major League Baseball
2	스포츠	NHL	National Hockey League
3	스포츠	NFL	National Football League
4	스포츠	NBA	National Basketball Association
5	Concerts	Classical	All symphony, concerto, and choir concerts

주제

- 'auto' 옵션을 사용하여 JSON 데이터에서 로드
- 'auto ignorecase' 옵션을 사용하여 JSON 데이터에서 로드
- JSONPaths 파일을 사용하여 JSON 데이터에서 로드
- JSONPaths 파일을 사용하여 JSON 배열에서 로드

'auto' 옵션을 사용하여 JSON 데이터에서 로드

JSON 데이터에서 `'auto'` 옵션을 사용해 로드하려면 JSON 데이터가 객체 집합으로 구성되어야 합니다. 키 이름이 열 이름과 일치해야 하지만 순서는 중요하지 않습니다. 다음은 `category_object_auto.json`이라는 이름의 파일 내용을 나타낸 것입니다.

```
{
    "catdesc": "Major League Baseball",
    "catid": 1,
    "catgroup": "Sports",
    "catname": "MLB"
```

```
}
{
    "catgroup": "Sports",
    "catid": 2,
    "catname": "NHL",
    "catdesc": "National Hockey League"
}{
    "catid": 3,
    "catname": "NFL",
    "catgroup": "Sports",
    "catdesc": "National Football League"
}
{
    "bogus": "Bogus Sports LLC",
    "catid": 4,
    "catgroup": "Sports",
    "catname": "NBA",
    "catdesc": "National Basketball Association"
}
{
    "catid": 5,
    "catgroup": "Shows",
    "catname": "Musicals",
    "catdesc": "All symphony, concerto, and choir concerts"
}
```

위의 예에 있는 JSON 데이터 파일에서 로드하려면 다음 COPY 명령을 실행합니다.

```
copy category
from 's3://amzn-s3-demo-bucket/category_object_auto.json'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
json 'auto';
```

'auto ignorecase' 옵션을 사용하여 JSON 데이터에서 로드

JSON 데이터에서 `'auto ignorecase'` 옵션을 사용해 로드하려면 JSON 데이터가 객체 집합으로 구성되어야 합니다. 키 이름의 대/소문자는 열 이름과 일치하지 않아도 되며 순서는 중요하지 않습니다. 다음은 `category_object_auto-ignorecase.json`이라는 이름의 파일 내용을 나타낸 것입니다.

```
{
    "CatDesc": "Major League Baseball",
```

```
    "CatID": 1,
    "CatGroup": "Sports",
    "CatName": "MLB"
}
{
    "CatGroup": "Sports",
    "CatID": 2,
    "CatName": "NHL",
    "CatDesc": "National Hockey League"
}{
    "CatID": 3,
    "CatName": "NFL",
    "CatGroup": "Sports",
    "CatDesc": "National Football League"
}
{
    "bogus": "Bogus Sports LLC",
    "CatID": 4,
    "CatGroup": "Sports",
    "CatName": "NBA",
    "CatDesc": "National Basketball Association"
}
{
    "CatID": 5,
    "CatGroup": "Shows",
    "CatName": "Musicals",
    "CatDesc": "All symphony, concerto, and choir concerts"
}
```

위의 예에 있는 JSON 데이터 파일에서 로드하려면 다음 COPY 명령을 실행합니다.

```
copy category
from 's3://amzn-s3-demo-bucket/category_object_auto ignorecase.json'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
json 'auto ignorecase';
```

JSONPaths 파일을 사용하여 JSON 데이터에서 로드

JSON 데이터 객체가 열 이름과 정확히 일치하지 않으면 JSONPaths 파일을 사용하여 JSON 요소를 열로 매핑할 수 있습니다. JSON 원본 데이터에서 순서는 중요하지 않습니다. 단, JSONPaths 파일 표현식의 순서는 열 순서와 일치해야 합니다. 다음과 같이 `category_object_paths.json`이라는 이름의 데이터 파일이 있는 경우

```
{
    "one": 1,
    "two": "Sports",
    "three": "MLB",
    "four": "Major League Baseball"
}
{
    "three": "NHL",
    "four": "National Hockey League",
    "one": 2,
    "two": "Sports"
}
{
    "two": "Sports",
    "three": "NFL",
    "one": 3,
    "four": "National Football League"
}
{
    "one": 4,
    "two": "Sports",
    "three": "NBA",
    "four": "National Basketball Association"
}
{
    "one": 6,
    "two": "Shows",
    "three": "Musicals",
    "four": "All symphony, concerto, and choir concerts"
}
```

이름이 `category_jsonpath.json`인 다음 JSONPaths 파일이 원본 데이터를 테이블 열로 매핑합니다.

```
{
    "jsonpaths": [
        "$['one']",
        "$['two']",
        "$['three']",
        "$['four']"
    ]
}
```

위의 예에 있는 JSON 데이터 파일에서 로드하려면 다음 COPY 명령을 실행합니다.

```
copy category
from 's3://amzn-s3-demo-bucket/category_object_paths.json'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
json 's3://amzn-s3-demo-bucket/category_jsonpath.json';
```

JSONPaths 파일을 사용하여 JSON 배열에서 로드

배열 집합으로 구성된 JSON 데이터에서 로드하려면 JSONPaths 파일을 사용하여 배열 요소를 열로 매핑해야 합니다. 다음과 같이 `category_array_data.json`이라는 이름의 데이터 파일이 있는 경우

```
[1,"Sports","MLB","Major League Baseball"]
[2,"Sports","NHL","National Hockey League"]
[3,"Sports","NFL","National Football League"]
[4,"Sports","NBA","National Basketball Association"]
[5,"Concerts","Classical","All symphony, concerto, and choir concerts"]
```

이름이 `category_array_jsonpath.json`인 다음 JSONPaths 파일이 원본 데이터를 테이블 열로 매핑합니다.

```
{
    "jsonpaths": [
        "$[0]",
        "$[1]",
        "$[2]",
        "$[3]"
    ]
}
```

위의 예에 있는 JSON 데이터 파일에서 로드하려면 다음 COPY 명령을 실행합니다.

```
copy category
from 's3://amzn-s3-demo-bucket/category_array_data.json'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
json 's3://amzn-s3-demo-bucket/category_array_jsonpath.json';
```

Avro에서 복사 예제

다음 예에서는 다음 데이터와 함께 CATEGORY 테이블을 로드합니다.

CATID	CATGROUP	CATNAME	CATDESC
1	스포츠	MLB	Major League Baseball
2	스포츠	NHL	National Hockey League
3	스포츠	NFL	National Football League
4	스포츠	NBA	National Basketball Association
5	Concerts	Classical	All symphony, concerto, and choir concerts

주제

- 'auto' 옵션을 사용하여 Avro 데이터에서 로드
- 'auto ignorecase' 옵션을 사용하여 Avro 데이터에서 로드
- JSONPaths 파일을 사용하여 Avro 데이터에서 로드

'auto' 옵션을 사용하여 Avro 데이터에서 로드

Avro 데이터에서 `'auto'` 인수를 사용하여 로드하려면 Avro 스키마의 필드 이름이 열 이름과 일치해야 합니다. `'auto'` 인수를 사용할 때 순서는 중요하지 않습니다. 다음은 이름이 `category_auto.avro`인 파일의 스키마를 나타낸 것입니다.

```
{
    "name": "category",
    "type": "record",
    "fields": [
        {"name": "catid", "type": "int"},
        {"name": "catdesc", "type": "string"},
        {"name": "catname", "type": "string"},
        {"name": "catgroup", "type": "string"},
}
```

Avro 파일의 데이터는 이진 형식이기 때문에 사람이 읽을 수 없습니다. 다음은 `category_auto.avro` 파일의 데이터를 JSON으로 표현한 것입니다.

```
{
```

```
   "catid": 1,
   "catdesc": "Major League Baseball",
   "catname": "MLB",
   "catgroup": "Sports"
}
{
   "catid": 2,
   "catdesc": "National Hockey League",
   "catname": "NHL",
   "catgroup": "Sports"
}
{
   "catid": 3,
   "catdesc": "National Basketball Association",
   "catname": "NBA",
   "catgroup": "Sports"
}
{
   "catid": 4,
   "catdesc": "All symphony, concerto, and choir concerts",
   "catname": "Classical",
   "catgroup": "Concerts"
}
```

위의 예에 있는 Avro 데이터 파일에서 로드하려면 다음 COPY 명령을 실행합니다.

```
copy category
from 's3://amzn-s3-demo-bucket/category_auto.avro'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as avro 'auto';
```

'auto ignorecase' 옵션을 사용하여 Avro 데이터에서 로드

Avro 데이터에서 `'auto ignorecase'` 인수를 사용하여 로드하려는 경우 Avro 스키마에서 필드 이름의 대/소문자는 열 이름의 대/소문자와 일치하지 않아도 됩니다. `'auto ignorecase'` 인수를 사용할 때 순서는 중요하지 않습니다. 다음은 이름이 `category_auto-ignorecase.avro`인 파일의 스키마를 나타낸 것입니다.

```
{
    "name": "category",
    "type": "record",
    "fields": [
```

```
        {"name": "CatID", "type": "int"},
        {"name": "CatDesc", "type": "string"},
        {"name": "CatName", "type": "string"},
        {"name": "CatGroup", "type": "string"},
}
```

Avro 파일의 데이터는 이진 형식이기 때문에 사람이 읽을 수 없습니다. 다음은 `category_auto-ignorecase.avro` 파일의 데이터를 JSON으로 표현한 것입니다.

```
{
   "CatID": 1,
   "CatDesc": "Major League Baseball",
   "CatName": "MLB",
   "CatGroup": "Sports"
}
{
   "CatID": 2,
   "CatDesc": "National Hockey League",
   "CatName": "NHL",
   "CatGroup": "Sports"
}
{
   "CatID": 3,
   "CatDesc": "National Basketball Association",
   "CatName": "NBA",
   "CatGroup": "Sports"
}
{
   "CatID": 4,
   "CatDesc": "All symphony, concerto, and choir concerts",
   "CatName": "Classical",
   "CatGroup": "Concerts"
}
```

위의 예에 있는 Avro 데이터 파일에서 로드하려면 다음 COPY 명령을 실행합니다.

```
copy category
from 's3://amzn-s3-demo-bucket/category_auto-ignorecase.avro'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as avro 'auto ignorecase';
```

JSONPaths 파일을 사용하여 Avro 데이터에서 로드

Avro 스키마의 필드 이름이 열 이름과 정확히 일치하지 않으면 JSONPaths 파일을 사용하여 스키마 요소를 열로 매핑할 수 있습니다. JSONPaths 파일 표현식의 순서는 열 순서와 일치해야 합니다.

앞의 예와 동일한 데이터를 포함하지만 스키마는 다음과 같은 이름이 `category_paths.avro`인 데이터 파일이 있는 경우

```
{
    "name": "category",
    "type": "record",
    "fields": [
        {"name": "id", "type": "int"},
        {"name": "desc", "type": "string"},
        {"name": "name", "type": "string"},
        {"name": "group", "type": "string"},
        {"name": "region", "type": "string"}
     ]
}
```

이름이 `category_path.avropath`인 다음 JSONPaths 파일이 원본 데이터를 테이블 열로 매핑합니다.

```
{
    "jsonpaths": [
        "$['id']",
        "$['group']",
        "$['name']",
        "$['desc']"
    ]
}
```

위의 예에 있는 Avro 데이터 파일에서 로드하려면 다음 COPY 명령을 실행합니다.

```
copy category
from 's3://amzn-s3-demo-bucket/category_object_paths.avro'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format avro 's3://amzn-s3-demo-bucket/category_path.avropath ';
```

ESCAPE 옵션과 함께 COPY에 사용할 파일 준비

다음 예에서는 ESCAPE 파라미터와 함께 COPY 명령을 사용하여 데이터를 Amazon Redshift 테이블에 가져오기 전에 줄 바꿈 문자를 "이스케이프" 처리하도록 데이터를 준비하는 방법을 설명합니다. 줄 바꿈 문자를 구분할 데이터를 준비하지 않으면 줄 바꿈 문자가 대개 레코드 구분자로 사용되므로 COPY 명령을 실행할 때 Amazon Redshift에서 오류를 로드합니다.

예를 들어 외부 테이블에 Amazon Redshift 테이블로 복사할 파일 또는 열이 있는 경우 파일 또는 열에 XML 형식의 내용이나 이와 비슷한 데이터가 포함되어 있으면 내용의 일부인 줄 바꿈 문자(\n)가 모두 백슬래시 문자(\)로 이스케이프 처리되어 있는지 확인해야 합니다.

줄 바꿈 문자가 포함된 파일 또는 테이블은 비교적 식별하기 용이한 패턴을 제공합니다. 줄 바꿈 문자가 항상 > 문자 뒤에 나오고, 그 사이에 잠재적으로 공백(' ' 또는 탭)이 포함될 가능성이 매우 높기 때문입니다. 다음과 같이 nlTest1.txt라는 이름의 텍스트 파일 예를 참조하십시오.

```
$ cat nlTest1.txt
<xml start>
<newline characters provide>
<line breaks at the end of each>
<line in content>
</xml>|1000
<xml>
</xml>|2000
```

다음 예에서는 텍스트 처리 유틸리티를 실행하여 원본 파일을 전처리한 후 필요한 자리에 이스케이프 문자를 삽입할 수 있습니다. (| 문자는 원래 Amazon Redshift 테이블에 복사할 때 열 데이터의 구분자로 사용하기 위한 것입니다.)

```
$ sed -e ':a;N;$!ba;s/>[[:space:]]*\n/>\\\n/g' nlTest1.txt > nlTest2.txt
```

마찬가지로 Perl을 사용하여 비슷한 작업을 실행할 수 있습니다.

```
cat nlTest1.txt | perl -p -e 's/>\s*\n/>\\\n/g' > nlTest2.txt
```

먼저 nlTest2.txt 파일의 데이터를 Amazon Redshift로 로드할 수 있도록에서 2열 테이블을 생성했습니다. 첫 번째 열인 c1은 nlTest2.txt 파일에서 XML 형식의 내용을 저장할 문자 열입니다. 두 번째 열인 c2에는 동일한 파일에서 로드되는 정수 값이 저장됩니다.

sed 명령을 실행한 후에는 ESCAPE 파라미터를 사용하여 nlTest2.txt 파일의 데이터를 Amazon Redshift 테이블로 정확하게 로드할 수 있습니다.

> **Note**
>
> COPY 명령에 ESCAPE 파라미터를 추가하면 백슬래시 문자(줄 바꿈 문자 포함)를 포함하여 다수의 특수 문자가 이스케이프 처리됩니다.

```
copy t2 from 's3://amzn-s3-demo-bucket/data/nlTest2.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
escape
delimiter as '|';

select * from t2 order by 2;

c1           |  c2
-------------+------
<xml start>
<newline characters provide>
<line breaks at the end of each>
<line in content>
</xml>
| 1000
<xml>
</xml>       | 2000
(2 rows)
```

유사한 방법으로 외부 데이터베이스에서 내보내는 데이터 파일도 준비할 수 있습니다. Oracle 데이터베이스를 예로 들면, 테이블에서 Amazon Redshift로 복사할 열마다 REPLACE 함수를 사용하면 됩니다.

```
SELECT c1, REPLACE(c2, \n',\\n' ) as c2 from my_table_with_xml
```

그 밖에 일반적으로 대용량 데이터를 처리하는 데이터베이스 내보내기 및 ETL(Extract, Transform, Load) 도구들도 대부분 이스케이프 및 구분자 문자를 지정하는 옵션이 있습니다.

Amazon Redshift에 shapefile 로드

다음 예에서는 COPY를 사용하여 Esri shapefile을 로드하는 방법을 보여줍니다. shapefile 로드에 대한 자세한 내용은 Amazon Redshift에 shapefile 로드 섹션을 참조하세요.

shapefile 로드

다음 단계에서는 COPY 명령을 사용하여 Amazon S3에서 OpenStreetMap 데이터를 수집하는 방법을 보여줍니다. 이 예에서는 Geofabrik 다운로드 사이트의 Norway shapefile 아카이브가 해당 AWS 리전의 프라이빗 Amazon S3 버킷에 업로드되었다고 가정합니다. `.shp`, `.shx` 및 `.dbf` 파일은 동일한 Amazon S3 접두사와 파일 이름을 공유해야 합니다.

단순화 없이 데이터 수집

다음 명령은 단순화 없이 최대 지오메트리 크기에 맞는 테이블을 생성하고 데이터를 수집합니다. 선호하는 GIS 소프트웨어에서 `gis_osm_natural_free_1.shp`를 열고 이 계층의 열을 검사합니다. 기본적으로 IDENTITY 또는 GEOMETRY 열이 첫 번째 열입니다. GEOMETRY 열이 첫 번째 열이면 다음과 같이 테이블을 생성할 수 있습니다.

```
CREATE TABLE norway_natural (
   wkb_geometry GEOMETRY,
   osm_id BIGINT,
   code INT,
   fclass VARCHAR,
   name VARCHAR);
```

또는 IDENTITY 열이 첫 번째 열이면 다음과 같이 테이블을 생성할 수 있습니다.

```
CREATE TABLE norway_natural_with_id (
   fid INT IDENTITY(1,1),
   wkb_geometry GEOMETRY,
   osm_id BIGINT,
   code INT,
   fclass VARCHAR,
   name VARCHAR);
```

이제 COPY를 사용하여 데이터를 수집할 수 있습니다.

```
COPY norway_natural FROM 's3://bucket_name/shapefiles/norway/
gis_osm_natural_free_1.shp'
FORMAT SHAPEFILE
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';
INFO: Load into table 'norway_natural' completed, 83891 record(s) loaded successfully
```

또는 다음과 같이 데이터를 수집할 수 있습니다.

```
COPY norway_natural_with_id FROM 's3://bucket_name/shapefiles/norway/
gis_osm_natural_free_1.shp'
FORMAT SHAPEFILE
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';
INFO: Load into table 'norway_natural_with_id' completed, 83891 record(s) loaded
 successfully.
```

단순화와 함께 데이터 수집

다음 명령은 단순화 없이 최대 지오메트리 크기에 맞지 않는 테이블을 생성하고 데이터 수집을 시도합니다. gis_osm_water_a_free_1.shp shapefile을 검사하고 다음과 같이 적절한 테이블을 생성합니다.

```
CREATE TABLE norway_water (
   wkb_geometry GEOMETRY,
   osm_id BIGINT,
   code INT,
   fclass VARCHAR,
   name VARCHAR);
```

COPY 명령이 실행되면 오류가 발생합니다.

```
COPY norway_water FROM 's3://bucket_name/shapefiles/norway/gis_osm_water_a_free_1.shp'
FORMAT SHAPEFILE
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';
ERROR:  Load into table 'norway_water' failed.  Check 'stl_load_errors' system table
 for details.
```

STL_LOAD_ERRORS 쿼리 결과 지오메트리가 너무 큽니다.

```
SELECT line_number, btrim(colname), btrim(err_reason) FROM stl_load_errors WHERE query
 = pg_last_copy_id();
 line_number |    btrim     |                                  btrim
-------------+--------------
+-------------------------------------------------------------------------
     1184705 | wkb_geometry | Geometry size: 1513736 is larger than maximum supported
 size: 1048447
```

이를 극복하기 위해SIMPLIFY AUTO 파라미터가 COPY 명령에 추가되어 지오메트리를 단순화합니다.

```
COPY norway_water FROM 's3://bucket_name/shapefiles/norway/gis_osm_water_a_free_1.shp'
FORMAT SHAPEFILE
SIMPLIFY AUTO
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';

INFO:  Load into table 'norway_water' completed, 1989196 record(s) loaded successfully.
```

단순화된 행과 지오메트리를 보려면 SVL_SPATIAL_SIMPLIFY를 쿼리합니다.

```
SELECT * FROM svl_spatial_simplify WHERE query = pg_last_copy_id();
 query | line_number | maximum_tolerance | initial_size | simplified | final_size |
 final_tolerance
-------+-------------+-------------------+--------------+------------+------------
+----------------------
    20 |     1184704 |                -1 |      1513736 | t          |    1008808 |
 1.276386653895e-05
    20 |     1664115 |                -1 |      1233456 | t          |    1023584 |
 6.11707814796635e-06
```

자동 계산된 것보다 낮은 허용치로 SIMPLIFY AUTO max_tolerance를 사용하면 수집 오류가 발생할 수 있습니다. 이 경우 MAXERROR를 사용하여 오류를 무시합니다.

```
COPY norway_water FROM 's3://bucket_name/shapefiles/norway/gis_osm_water_a_free_1.shp'
FORMAT SHAPEFILE
SIMPLIFY AUTO 1.1E-05
MAXERROR 2
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';

INFO:  Load into table 'norway_water' completed, 1989195 record(s) loaded successfully.
INFO:  Load into table 'norway_water' completed, 1 record(s) could not be loaded.
 Check 'stl_load_errors' system table for details.
```

SVL_SPATIAL_SIMPLIFY를 다시 쿼리하여 COPY가 로드하지 않은 레코드를 식별합니다.

```
SELECT * FROM svl_spatial_simplify WHERE query = pg_last_copy_id();
 query | line_number | maximum_tolerance | initial_size | simplified | final_size |
 final_tolerance
-------+-------------+-------------------+--------------+------------+------------
+-----------------
    29 |     1184704 |           1.1e-05 |      1513736 | f          |          0 |
           0
```

```
    29 |     1664115 |           1.1e-05 |      1233456 | t           |     794432 |
     1.1e-05
```

이 예에서는 첫 번째 레코드가 맞지 않아 simplified 열이 false로 표시됩니다. 두 번째 레코드가 지정된 허용치 내에서 로드되었습니다. 그러나 최종 크기는 최대 허용치를 지정하지 않고 자동 계산된 허용치를 사용하는 것보다 큽니다.

압축된 shapefile에서 로드

Amazon Redshift COPY는 압축된 shapefile에서 데이터 수집을 지원합니다. 모든 shapefile 구성 요소에는 동일한 Amazon S3 접두사와 동일한 압축 접미사가 있어야 합니다. 예를 들어 이전 예의 데이터를 로드하려고 한다고 가정합니다. 이 경우 파일 gis_osm_water_a_free_1.shp.gz, gis_osm_water_a_free_1.dbf.gz, 및 gis_osm_water_a_free_1.shx.gz는 동일한 Amazon S3 디렉터리를 공유해야 합니다. COPY 명령에는 GZIP 옵션이 필요하고 FROM 절은 다음과 같이 올바른 압축 파일을 지정해야 합니다.

```
COPY norway_natural FROM 's3://bucket_name/shapefiles/norway/compressed/
gis_osm_natural_free_1.shp.gz'
FORMAT SHAPEFILE
GZIP
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';
INFO:  Load into table 'norway_natural' completed, 83891 record(s) loaded successfully.
```

다른 열 순서로 테이블에 데이터 로드

첫 번째 열이 GEOMETRY가 아닌 테이블이 있는 경우 열 매핑을 사용하여 열을 대상 테이블에 매핑할 수 있습니다. 예를 들어 osm_id가 첫 번째 열로 지정된 테이블을 생성합니다.

```
CREATE TABLE norway_natural_order (
   osm_id BIGINT,
   wkb_geometry GEOMETRY,
   code INT,
   fclass VARCHAR,
   name VARCHAR);
```

그런 다음 열 매핑을 사용하여 shapefile을 수집합니다.

```
COPY norway_natural_order(wkb_geometry, osm_id, code, fclass, name)
FROM 's3://bucket_name/shapefiles/norway/gis_osm_natural_free_1.shp'
```

```
FORMAT SHAPEFILE
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';
INFO:  Load into table 'norway_natural_order' completed, 83891 record(s) loaded
 successfully.
```

지오그래피 열이 있는 테이블에 데이터 로드

GEOGRAPHY 열이 있는 테이블이 있는 경우 먼저 열GEOMETRY 로 수집한 다음 객체를 GEOGRAPHY 객체로 캐스팅합니다. 예를 들어, shapefile을 GEOMETRY 열에 복사한 후 테이블을 변경하여 GEOGRAPHY 데이터 유형의 열을 추가합니다.

```
ALTER TABLE norway_natural ADD COLUMN wkb_geography GEOGRAPHY;
```

그런 다음 지오메트리를 지오그래피로 변환합니다.

```
UPDATE norway_natural SET wkb_geography = wkb_geometry::geography;
```

선택적으로 GEOMETRY 열을 삭제할 수 있습니다.

```
ALTER TABLE norway_natural DROP COLUMN wkb_geometry;
```

NOLOAD 옵션을 사용한 CPPY 명령

실제로 데이터를 로드하기 전에 데이터 파일을 확인하려면 COPY 명령과 함께 NOLOAD 옵션을 사용합니다. Amazon Redshift Redshift는 입력 파일을 구문 분석하여 발생한 오류를 표시합니다. 다음 예제에서는 NOLOAD 옵션을 사용하며 실제로 테이블에 행이 로드되지 않습니다.

```
COPY public.zipcode1
FROM 's3://amzn-s3-demo-bucket/mydata/zipcode.csv'
DELIMITER ';'
IGNOREHEADER 1 REGION 'us-east-1'
NOLOAD
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/myRedshiftRole';

Warnings:
Load into table 'zipcode1' completed, 0 record(s) loaded successfully.
```

멀티바이트 구분 기호와 ENCODING 옵션을 포함한 COPY 명령

다음 예제는 멀티바이트 데이터가 포함된 Amazon S3 파일에서 LATIN1을 로드합니다. COPY 명령은 구분 기호를 8진수 형식(\302\246\303\254)으로 지정하여 ISO-8859-1로 인코딩된 입력 파일의 필드를 구분합니다. UTF-8에서 동일한 구분 기호를 지정하려면 `DELIMITER '¦ì'`를 지정합니다.

```
COPY latin1
FROM 's3://amzn-s3-demo-bucket/multibyte/myfile'
IAM_ROLE 'arn:aws:iam::123456789012:role/myRedshiftRole'
DELIMITER '\302\246\303\254'
ENCODING ISO88591
```

데이터베이스 생성

새 레벨을 생성합니다.

데이터베이스를 생성하려면 슈퍼 사용자이거나 CREATEDB 권한이 있어야 합니다. 제로 ETL 통합과 연결된 데이터베이스를 생성하려면 수퍼유저이거나 CREATEDB 권한과 CREATEDB 권한이 모두 있어야 합니다.

트랜잭션 블록(BEGIN ... END) 내에서는 CREATE DATABASE를 실행할 수 없습니다. 버전 관리에 대한 자세한 내용은 직렬화 가능 격리 단원을 참조하십시오.

구문

```
CREATE DATABASE database_name
[ { [ WITH ]
    [ OWNER [=] db_owner ]
    [ CONNECTION LIMIT { limit | UNLIMITED } ]
    [ COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE } ]
    [ ISOLATION LEVEL { SERIALIZABLE | SNAPSHOT } ]
  }
  | { [ WITH PERMISSIONS ] FROM DATASHARE datashare_name ] OF [ ACCOUNT account_id ]
 NAMESPACE namespace_guid }
  | { FROM { { ARN '<arn>' } { WITH DATA CATALOG SCHEMA '<schema>' | WITH NO DATA
 CATALOG SCHEMA } }
             | { INTEGRATION '<integration_id>' [ DATABASE '<source_database>' ] [SET
 {REFRESH_INTERVAL <interval>} ] } }
  | { IAM_ROLE  {default | 'SESSION' | 'arn:aws:iam::<account-id>:role/<role-name>' } }
]
```

파라미터

database_name

새 데이터베이스의 이름입니다. 유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요.

WITH

선택적 키워드입니다.

OWNER

데이터베이스 소유자를 지정합니다.

=

선택적 문자입니다.

db_owner

데이터베이스 소유자의 사용자 이름입니다.

CONNECTION LIMIT { limit | UNLIMITED }

사용자가 동시에 열어놓을 수 있는 데이터베이스 연결의 최대 개수입니다. 슈퍼 사용자에 대해서는 이 제한이 적용되지 않습니다. 최대 동시 연결 수를 허용하려면 UNLIMITED 키워드를 사용하십시오. 각 사용자에 대한 연결 개수 제한이 적용될 수도 있습니다. 자세한 내용은 사용자 생성 단원을 참조하십시오. 기본값은 UNLIMITED입니다. 현재 연결을 보려면 STV_SESSIONS 시스템 뷰를 쿼리하십시오.

> **Note**
>
> 사용자 및 데이터베이스 연결 제한이 모두 적용되는 경우 사용되지 않는 연결 슬롯은 사용자가 연결 시도 시 양쪽 제한 범위 내에서 모두 사용 가능해야 합니다.

COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE }

문자열 검색 또는 비교가 CASE_SENSITIVE인지, CASE_INSENSITIVE인지를 지정하는 절입니다. 기본값은 CASE_SENSITIVE입니다.

ISOLATION LEVEL { SERIALIZABLE | SNAPSHOT }

데이터베이스에 대해 쿼리가 실행될 때 사용되는 격리 수준을 지정하는 절입니다.

- SERIALIZABLE 격리 - 동시 트랜잭션에 대한 완전한 직렬화 기능을 제공합니다. 자세한 내용은 직렬화 가능 격리 단원을 참조하십시오.
- SNAPSHOT 격리 - 업데이트 및 삭제 충돌로부터 보호하는 격리 수준을 제공합니다. 프로비저닝된 클러스터 또는 서버리스 네임스페이스에서 생성된 데이터베이스의 기본값입니다.

다음과 같이 데이터베이스에서 실행 중인 동시성 모델을 볼 수 있습니다.

- STV_DB_ISOLATION_LEVEL 보기를 쿼리합니다. 자세한 내용은 STV_DB_ISOLATION_LEVEL 단원을 참조하십시오.

```
SELECT * FROM stv_db_isolation_level;
```

- PG_DATABASE_INFO 보기를 쿼리합니다.

```
SELECT datname, datconfig FROM pg_database_info;
```

  데이터베이스당 격리 수준이 `concurrency_model` 키 옆에 나타납니다. 값 `1`은 SNAPSHOT을 나타냅니다. 값 `2`는 SERIALIZABLE을 나타냅니다.

Amazon Redshift 데이터베이스에서 SERIALIZABLE 및 SNAPSHOT 격리는 모두 직렬화 가능한 격리 수준 유형입니다. 즉, SQL 표준에 따라 더티 읽기, 비반복 읽기, 팬텀 읽기가 방지됩니다. 두 격리 수준 모두 트랜잭션이 시작할 때 존재했던 데이터의 스냅샷에서 트랜잭션이 작동하고 다른 트랜잭션은 해당 스냅샷을 변경할 수 없도록 보장합니다. 그러나 SNAPSHOT 격리는 서로 다른 테이블 행에서 쓰기 스큐 삽입 및 업데이트를 방지하지 않기 때문에 완전한 직렬화 기능을 제공하지 않습니다.

다음 시나리오에서는 SNAPSHOT 격리 수준을 사용한 쓰기 스큐 업데이트를 보여줍니다. `Numbers` 테이블에는 `0` 및 `1` 값을 포함하는 `digits` 열이 있습니다. 각 사용자의 UPDATE 문은 다른 사용자와 겹치지 않습니다. 그러나 `0` 및 `1` 값은 스왑됩니다. 실행 중인 SQL은 이 타임라인에 따라 다음과 같은 결과가 나타납니다.

Time	사용자 1 작업	사용자 2 작업
1	BEGIN:	
2		BEGIN:
3	SELECT *	

Time	사용자 1 작업	사용자 2 작업
	FROM Numbers `digits - ----- 0 1`	
4		SELECT * FROM Numbers; `digits ------ 0 1`
5	UPDATE Numbers SET digits=0 WHERE digits=1;	
6	SELECT * FROM Numbers `digits - ----- 0 0`	

Time	사용자 1 작업	사용자 2 작업
7	COMMIT	
8		Update Numbers SET digits=1 WHERE digits=0;
9		SELECT * FROM Numbers; digits ------ 1 1
10		COMMIT;
11	SELECT * FROM Numbers; digits - ----- 1 0	
12		SELECT * FROM Numbers; digits ------ 1 0

직렬화 가능 격리를 사용하여 동일한 시나리오를 실행하는 경우 Amazon Redshift 직렬화 가능 위반으로 인해 사용자 2를 종료하고 오류 1023을 반환합니다. 자세한 내용은 직렬화 가능 격리 오류

수정 방법 단원을 참조하십시오. 이 경우 사용자 1만 성공적으로 커밋할 수 있습니다. 모든 워크로드에 직렬화 가능 격리가 필요한 것은 아니며, 이 경우 스냅샷 격리는 데이터베이스의 대상 격리 수준으로 충분합니다.

FROM ARN '<ARN>'

데이터베이스를 생성하는 데 사용할 AWS Glue 데이터베이스 ARN입니다.

{ DATA CATALOG SCHEMA '<schema>' | WITH NO DATA CATALOG SCHEMA }

> **Note**
>
> 이 파라미터는 CREATE DATABASE 명령에 FROM ARN 파라미터도 사용하는 경우에만 적용할 수 있습니다.

AWS Glue Data Catalog의 객체에 액세스하는 데 도움이 되는 스키마를 사용하여 데이터베이스를 생성할지 여부를 지정합니다.

FROM INTEGRATION '<integration_id>' [DATABASE '<source_database>'] [SET {REFRESH_INTERVAL <interval>}]

제로 ETL 통합 식별자를 사용하여 데이터베이스를 생성할지 여부를 지정합니다. SVV_INTEGRATION 시스템 뷰에서 `integration_id`를 검색할 수 있습니다. Aurora PostgreSQL 제로 ETL 통합의 경우 SVV_INTEGRATION에서 검색할 수도 있는 `source_database` 이름을 지정해야 합니다. SET REFRESH_INTERVAL 절은 제로 ETL 소스에서 대상 데이터베이스로 데이터를 새로고침하기 위해 대략적인 시간 간격을 초 단위로 설정합니다. 소스 유형이 Aurora MySQL, Aurora PostgreSQL 또는 RDS for MySQL인 제로 ETL 통합의 경우 값을 0~432,000초(5일)로 설정할 수 있습니다. Amazon DynamoDB 제로 ETL 통합의 경우 값을 900~432,000초(15분~5일)로 설정할 수 있습니다. 소스 유형이 Aurora MySQL, Aurora PostgreSQL 또는 RDS for MySQL인 제로 ETL 통합의 경우 기본 `interval`은 0초입니다. Amazon DynamoDB 제로 ETL 통합의 경우 기본 `interval`은 900초(15분)입니다.

예시는 제로 ETL 통합 결과를 받을 데이터베이스 생성을 확인하세요. 제로 ETL 통합을 사용하여 데이터베이스를 생성하는 방법에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 Amazon Redshift에서 대상 데이터베이스 생성을 참조하세요.

IAM_ROLE { default | 'SESSION' | 'arn:aws:iam::*<AWS ##-id>*:role/*<role-name>*' }

> **Note**
>
> 이 파라미터는 CREATE DATABASE 명령에 FROM ARN 파라미터도 사용하는 경우에만 적용할 수 있습니다.

CREATE DATABASE 명령을 실행할 때 클러스터와 연결된 IAM 역할을 지정하면 데이터베이스에서 쿼리를 실행할 때 Amazon Redshift가 해당 역할의 보안 인증을 사용합니다.

`default` 키워드를 지정한다는 것은 기본값으로 설정되어 클러스터와 연결된 IAM 역할을 사용한다는 의미입니다.

페더레이션형 ID를 사용하여 Amazon Redshift 클러스터에 연결하고 이 명령을 사용하여 생성된 외부 스키마에서 테이블에 액세스하는 경우에 `'SESSION'`을 사용합니다. 페더레이션 ID 사용의 예를 보려면 페더레이션형 ID 구성 방법이 설명된 페더레이션형 ID를 사용하여 로컬 리소스 및 Amazon Redshift Spectrum 외부 테이블에 대한 Amazon Redshift 액세스 관리 섹션을 참조하세요.

클러스터가 인증 및 권한 부여에 사용하는 IAM 역할의 Amazon 리소스 이름(ARN)을 사용합니다. 최소 IAM 역할은 액세스되는 Amazon S3 버킷에서 LIST 작업을 수행하고 버킷에 포함된 Amazon S3 객체에 대한 GET 작업을 수행할 수 있는 권한이 있어야 합니다. 데이터 공유를 위해 AWS Glue Data Catalog를 사용하여 데이터베이스를 생성할 때 IAM_ROLE을 사용하는 방법에 대해 자세히 알아보려면 소비자로서 레이크 포메이션 관리형 데이터 공유로 작업을 참조하세요.

다음은 단일 ARN에 대한 IAM_ROLE 파라미터의 구문을 보여줍니다.

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-name>'
```

역할을 함께 묶어 클러스터가 다른 계정에 속한 다른 IAM 역할을 수임하도록 할 수 있습니다. 최대 10개의 역할을 함께 묶을 수 있습니다. 자세한 내용은 Amazon Redshift Spectrum에서 IAM 역할 연결 단원을 참조하십시오.

이 IAM 역할에 다음과 유사한 IAM 권한 정책을 연결합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "AccessSecret",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecretVersionIds"
            ],
            "Resource": "arn:aws:secretsmanager:us-west-2:123456789012:secret:my-
rds-secret-VNenFy"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword",
                "secretsmanager:ListSecrets"
            ],
            "Resource": "*"
        }
    ]
}
```

연합 쿼리에 사용할 IAM 역할을 생성하는 단계는 연합 쿼리 사용을 위해 비밀 및 IAM 역할 생성 섹션을 참조하세요.

> **Note**
>
> 연결된 역할 목록에 공백을 포함하지 마십시오.

다음은 세 역할을 함께 묶기 위한 구문을 나타낸 것입니다.

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-1-name>,arn:aws:iam::<aws-
account-id>:role/<role-2-name>,arn:aws:iam::<aws-account-id>:role/<role-3-name>'
```

datashare와 함께 CREATE DATABASE를 사용하기 위한 구문

다음 구문에서는 동일한 AWS 계정 내에서 데이터를 공유하기 위해 datashare에서 데이터베이스를 생성하는 데 사용되는 CREATE DATABASE 명령을 설명합니다.

```
CREATE DATABASE database_name
[ [ WITH PERMISSIONS ] FROM DATASHARE datashare_name ] OF [ ACCOUNT account_id ]
 NAMESPACE namespace_guid
```

다음 구문에서는 AWS 계정 간에 데이터를 공유하기 위해 datashare에서 데이터베이스를 생성하는 데 사용되는 CREATE DATABASE 명령을 설명합니다.

```
CREATE DATABASE database_name
[ [ WITH PERMISSIONS ] FROM DATASHARE datashare_name ] OF ACCOUNT account_id
 NAMESPACE namespace_guid
```

datashare와 함께 CREATE DATABASE를 사용하기 위한 파라미터

FROM DATASHARE

datashare가 있는 위치를 나타내는 키워드입니다.

datashare_name

소비자 데이터베이스가 생성되는 datashare의 이름입니다.

WITH PERMISSIONS

개별 데이터베이스 객체에 액세스하려면 데이터 공유에서 만든 데이터베이스에 객체 수준 권한이 필요하다는 것을 지정합니다. 이 절을 사용하지 않으면 데이터베이스에 대한 USAGE 권한이 부여된 사용자 또는 역할은 데이터베이스의 모든 데이터베이스 객체에 대한 액세스 권한을 자동으로 갖게 됩니다.

NAMESPACE namespace_guid

datashare가 속한 생산자 네임스페이스를 지정하는 값입니다.

ACCOUNT account_id

datashare가 속한 생산자 계정을 지정하는 값입니다.

datashare를 위한 CREATE DATABASE 사용 노트

데이터베이스 수퍼유저는 CREATE DATABASE를 사용하여 AWS 계정 내의 datashare에서 데이터베이스를 생성할 때 NAMESPACE 옵션을 지정합니다. ACCOUNT 옵션은 선택 사항입니다. CREATE DATABASE를 사용하여 AWS 여러 계정의 데이터 공유에서 데이터베이스를 생성하는 경우 프로듀서의 ACCOUNT 및 NAMESPACE 옵션을 모두 지정합니다.

소비자 클러스터에서 하나의 datashare에 대해 하나의 소비자 데이터베이스만 생성할 수 있습니다. 동일한 datashare를 참조하는 여러 소비자 데이터베이스를 만들 수 없습니다.

AWS Glue Data Catalog의 CREATE DATABASE

AWS Glue 데이터베이스 ARN을 사용하여 데이터베이스를 생성하려면 CREATE DATABASE 명령에 ARN을 지정합니다.

```
CREATE DATABASE sampledb FROM ARN <glue-database-arn> WITH NO DATA CATALOG SCHEMA;
```

선택적으로 IAM_ROLE 파라미터에 값을 제공할 수도 있습니다. 파라미터 및 허용되는 값에 대한 자세한 내용은 파라미터를 참조하세요.

다음은 IAM 역할을 사용하여 ARN에서 데이터베이스를 만드는 방법을 보여주는 예제입니다.

```
CREATE DATABASE sampledb FROM ARN <glue-database-arn> WITH NO DATA CATALOG SCHEMA
 IAM_ROLE <iam-role-arn>
```

```
CREATE DATABASE sampledb FROM ARN <glue-database-arn> WITH NO DATA CATALOG SCHEMA
 IAM_ROLE default;
```

DATA CATALOG SCHEMA를 사용하여 데이터베이스를 생성할 수도 있습니다.

```
CREATE DATABASE sampledb FROM ARN <glue-database-arn> WITH DATA CATALOG SCHEMA
 <sample_schema> IAM_ROLE default;
```

제로 ETL 통합 결과를 받을 데이터베이스 생성

제로 ETL 통합 ID를 사용하여 데이터베이스를 생성하려면 CREATE DATABASE 명령에 `integration_id`를 지정합니다.

```
CREATE DATABASE destination_db_name FROM INTEGRATION 'integration_id';
```

예를 들어, 먼저 SVV_INTEGRATION에서 통합 ID를 검색합니다.

```
SELECT integration_id FROM SVV_INTEGRATION;
```

그런 다음 검색된 통합 ID 중 하나를 사용하여 제로 ETL 통합을 받는 데이터베이스를 생성합니다.

```
CREATE DATABASE sampledb FROM INTEGRATION 'a1b2c3d4-5678-90ab-cdef-EXAMPLE11111';
```

제로 ETL 통합 소스 데이터베이스가 필요한 경우 예를 들어 지정합니다.

```
CREATE DATABASE sampledb FROM INTEGRATION 'a1b2c3d4-5678-90ab-cdef-EXAMPLE11111'
 DATABASE 'sourcedb';
```

데이터베이스의 새로고침 간격을 설정할 수도 있습니다. 예를 들어 제로 ETL 통합 소스의 데이터에 대해 새로고침 간격을 7,200초로 설정하려면 다음과 같이 합니다.

```
CREATE DATABASE myacct_mysql FROM INTEGRATION 'a1b2c3d4-5678-90ab-cdef-EXAMPLE11111'
 SET REFRESH_INTERVAL 7200;
```

SVV_INTEGRATION 카탈로그 뷰에서 integration_id, target_database, source, refresh_interval 등의 제로 ETL 통합에 관한 정보를 쿼리합니다.

```
SELECT * FROM svv_integration;
```

CREATE DATABASE의 제한 사항

Amazon Redshift는 데이터베이스에 대해 다음과 같은 제한 사항을 적용합니다.

- 클러스터당 사용자 정의 데이터베이스 최대 개수는 60개입니다.
- 데이터베이스 이름은 최대 127바이트입니다.
- 데이터베이스 이름은 예약어가 될 수 없습니다.

데이터베이스 데이터 정렬

데이터 정렬은 데이터베이스 엔진이 SQL의 문자 형식 데이터를 비교하고 정렬하는 방법을 정의하는 일련의 규칙입니다. 대/소문자를 구분하지 않는 데이터 정렬은 가장 일반적으로 사용되는 데이터 정렬입니다. Amazon Redshift는 대/소문자를 구분하지 않는 데이터 정렬을 사용하여 다른 데이터 웨어하우스 시스템에서 쉽게 마이그레이션할 수 있습니다. 대/소문자를 구분하지 않는 데이터 정렬을 기본적으로 지원하는 Amazon Redshift는 배포 키, 정렬 키 또는 범위 제한 스캔과 같은 중요한 조정 또는 최적화 방법을 계속 사용합니다.

COLLATE 절은 데이터베이스의 모든 CHAR 및 VARCHAR 열에 대한 기본 데이터 정렬을 지정합니다. CASE_INSENSITIVE가 지정되면 모든 CHAR 또는 VARCHAR 열은 대/소문자를 구분하지 않는 데이터 정렬을 사용합니다. 데이터 정렬에 대한 자세한 내용은 콜레이션 시퀀스 섹션을 참조하세요.

대/소문자를 구분하지 않는 열에 삽입되거나 수집된 데이터는 원래 대/소문자를 유지합니다. 그러나 정렬 및 그룹화를 포함한 모든 비교 기반 문자열 연산은 대/소문자를 구분하지 않습니다. LIKE 조건자, 유사 및 정규식 함수와 같은 패턴 일치 연산도 대/소문자를 구분하지 않습니다.

다음 SQL 연산은 적용 가능한 데이터 정렬 의미 체계를 지원합니다.

- 비교 연산자: =, <>, <, <=, >, >=.
- LIKE 연산자
- ORDER BY 절
- GROUP BY 절
- MIN 및 MAX 및 LISTAGG와 같은 문자열 비교를 사용하는 집계 함수
- PARTITION BY 절 및 ORDER BY 절과 같은 윈도 함수
- 스칼라 함수 greatest() and least(), STRPOS(), REGEXP_COUNT(), REGEXP_REPLACE(), REGEXP_INSTR(), REGEXP_SUBSTR()
- Distinct 절
- UNION, INTERSECT 및 EXCEPT
- IN LIST

Amazon Redshift Spectrum 및 Aurora PostgreSQL 연합 쿼리를 포함한 외부 쿼리의 경우 VARCHAR 또는 CHAR 열의 데이터 정렬은 현재 데이터베이스 수준 데이터 정렬과 동일합니다.

다음 예에서는 Amazon Redshift Spectrum 테이블을 쿼리합니다.

```
SELECT ci_varchar FROM spectrum.test_collation
WHERE ci_varchar = 'AMAZON';

ci_varchar
----------
amazon
Amazon
AMAZON
AmaZon
(4 rows)
```

데이터베이스 데이터 정렬을 사용하여 테이블을 생성하는 방법에 대한 자세한 내용은 CREATE TABLE 섹션을 참조하세요.

COLLATE 함수에 대한 자세한 내용은 COLLATE 함수 섹션을 참조하세요.

데이터베이스 데이터 정렬 제한 사항

다음은 Amazon Redshift에서 데이터베이스 데이터 정렬 작업 시 제한 사항입니다.

- PG 카탈로그 테이블과 Amazon Redshift 시스템 테이블을 포함한 모든 시스템 테이블 또는 뷰는 대/소문자를 구분합니다.
- 소비자 데이터베이스와 생산자 데이터베이스의 데이터베이스 수준 데이터 정렬이 다른 경우 Amazon Redshift는 데이터베이스 간 쿼리와 클러스터 간 쿼리를 지원하지 않습니다.
- Amazon Redshift는 리더 노드 전용 쿼리에서 대/소문자를 구분하지 않는 데이터 정렬을 지원하지 않습니다.

  다음 예에서는 지원되지 않는 대/소문자를 구분하지 않는 쿼리와 Amazon Redshift에서 전송하는 오류를 보여줍니다.

```
SELECT collate(usename, 'case_insensitive') FROM pg_user;
ERROR:  Case insensitive collation is not supported in leader node only query.
```

- Amazon Redshift는 비교, 함수, 조인 또는 설정 연산과 같이 대/소문자를 구분하는 열과 대/소문자를 구분하지 않는 열 간의 상호 작용을 지원하지 않습니다.

  다음 예에서는 대/소문자를 구분하는 열과 대/소문자를 구분하지 않는 열이 상호 작용할 때의 오류를 보여줍니다.

```
CREATE TABLE test
  (ci_col varchar(10) COLLATE case_insensitive,
   cs_col varchar(10) COLLATE case_sensitive,
   cint int,
   cbigint bigint);
```

```
SELECT ci_col = cs_col FROM test;
ERROR:  Query with different collations is not supported yet.
```

```
SELECT concat(ci_col, cs_col) FROM test;
ERROR:  Query with different collations is not supported yet.
```

```
SELECT ci_col FROM test UNION SELECT cs_col FROM test;
ERROR:  Query with different collations is not supported yet.
```

```
SELECT * FROM test a, test b WHERE a.ci_col = b.cs_col;
ERROR:  Query with different collations is not supported yet.
```

```
Select Coalesce(ci_col, cs_col) from test;
ERROR:  Query with different collations is not supported yet.
```

```
Select case when cint > 0 then ci_col else cs_col end from test;
ERROR:  Query with different collations is not supported yet.
```

- Amazon Redshift는 SUPER 데이터 형식에 대한 데이터 정렬을 지원하지 않습니다. 대/소문자를 구분하지 않는 데이터베이스에 SUPER 열을 생성하고 SUPER 열과 대/소문자를 구분하지 않는 열 간의 상호 작용은 지원되지 않습니다.

  다음 예에서는 대/소문자를 구분하지 않는 데이터베이스에서 데이터 형식으로 SUPER를 사용하여 테이블을 생성합니다.

  ```
  CREATE TABLE super_table (a super);
  ERROR: SUPER column is not supported in case insensitive database.
  ```

  다음 예에서는 SUPER 데이터와 비교하여 대/소문자를 구분하지 않는 문자열로 데이터를 쿼리합니다.

  ```
  CREATE TABLE test_super_collation
  (s super, c varchar(10) COLLATE case_insensitive, i int);
  ```

  ```
  SELECT s = c FROM test_super_collation;
  ERROR:  Coercing from case insensitive string to SUPER is not supported.
  ```

이러한 쿼리가 작동하도록 하려면 COLLATE 함수를 사용하여 한 열의 데이터 정렬을 다른 열과 일치하도록 변환합니다. 자세한 내용은 COLLATE 함수 단원을 참조하십시오.

예시

데이터베이스 생성

다음 예에서는 TICKIT로 명명된 데이터베이스를 생성하고 사용자 DWUSER에게 소유권을 부여합니다.

```
create database tickit
with owner dwuser;
```

데이터베이스에 대한 세부 정보를 보려면 PG_DATABASE_INFO 카탈로그 테이블을 쿼리합니다.

```
select datname, datdba, datconnlimit
from pg_database_info
where datdba > 1;

 datname     | datdba | datconnlimit
-------------+--------+-------------
 admin       |    100 | UNLIMITED
 reports     |    100 | 100
 tickit      |    100 | 100
```

다음 예에서는 SNAPSHOT 격리 수준의 **sampledb** 데이터베이스를 생성합니다.

```
CREATE DATABASE sampledb ISOLATION LEVEL SNAPSHOT;
```

다음 예에서는 datashare salesshare에서 데이터베이스 sales_db를 생성합니다.

```
CREATE DATABASE sales_db FROM DATASHARE salesshare OF NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

데이터베이스 데이터 정렬 예

대/소문자를 구분하지 않는 데이터베이스 생성

다음 예에서는 sampledb 데이터베이스를 생성하고, T1 테이블을 생성하고, 데이터를 T1 테이블에 삽입합니다.

```
create database sampledb collate case_insensitive;
```

SQL 클라이언트를 사용하여 방금 만든 새 데이터베이스에 연결합니다. Amazon Redshift 쿼리 에디터 v2를 사용하려면 에디터에서 sampledb를 선택합니다. RSQL을 사용하려면 다음과 같은 명령을 사용합니다.

```
\connect sampledb;
```

```
CREATE TABLE T1 (
  col1 Varchar(20) distkey sortkey
);
```

```
INSERT INTO T1 VALUES ('bob'), ('john'), ('Mary'), ('JOHN'), ('Bob');
```

그런 다음 쿼리에서 John을 포함하는 결과를 찾습니다.

```
SELECT * FROM T1 WHERE col1 = 'John';

 col1
 ------
 john
 JOHN
(2 row)
```

대/소문자를 구분하지 않는 순서

다음 예에서는 테이블 T1을 사용하여 대/소문자를 구분하지 않는 순서를 보여줍니다. Bob과bob 또는 John과 john의 순서는 대/소문자를 구분하지 않는 열에서 동일하기 때문에 비결정적입니다.

```
SELECT * FROM T1 ORDER BY 1;

 col1
 ------
 bob
 Bob
 JOHN
 john
 Mary
(5 rows)
```

마찬가지로 다음 예에서는 GROUP BY 절을 사용하여 대/소문자를 구분하지 않는 순서를 보여줍니다. Bob과 bob은 동일하며 같은 그룹에 속합니다. 어느 것이 결과에 나타나는지는 비결정적입니다.

```
SELECT col1, count(*) FROM T1 GROUP BY 1;

 col1 | count
 -----+------
 Mary |  1
```

```
 bob  |  2
 JOHN |  2
(3 rows)
```

대/소문자를 구분하지 않는 열에서 윈도 함수로 쿼리

다음 예에서는 대/소문자를 구분하지 않는 열에서 윈도 함수를 쿼리합니다.

```
SELECT col1, rank() over (ORDER BY col1) FROM T1;

 col1 | rank
 -----+------
 bob  |   1
 Bob  |   1
 john |   3
 JOHN |   3
 Mary |   5
(5 rows)
```

DISTINCT 키워드로 쿼리

다음 예에서는 DISTINCT 키워드로 T1 테이블을 쿼리합니다.

```
SELECT DISTINCT col1 FROM T1;

 col1
 ------
 bob
 Mary
 john
(3 rows)
```

UNION 절로 쿼리

다음 예에서는 테이블 T1 및 T2의 UNION 결과를 보여줍니다.

```
CREATE TABLE T2 AS SELECT * FROM T1;
```

```
SELECT col1 FROM T1 UNION SELECT col1 FROM T2;
```

```
 col1
 ------
 john
 bob
 Mary
(3 rows)
```

CREATE DATASHARE

현재 데이터베이스에서 새 datashare를 생성합니다. 이 datashare의 소유자는 CREATE DATASHARE 명령의 발급자입니다.

Amazon Redshift는 각 datashare를 단일 Amazon Redshift 데이터베이스와 연결합니다. 연결된 데이터베이스의 객체만 datashare에 추가할 수 있습니다. 동일한 Amazon Redshift 데이터베이스에 여러 datashare를 생성할 수 있습니다.

datashare에 대한 자세한 내용은 Amazon Redshift에서 데이터 공유 섹션을 참조하세요.

datashare에 대한 정보를 보려면 SHOW DATASHARES를 사용합니다.

필수 권한

CREATE DATASHARE에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- CREATE DATASHARE 권한이 있는 사용자
- 데이터베이스 소유자

구문

```
CREATE DATASHARE datashare_name
[[SET] PUBLICACCESSIBLE [=] TRUE | FALSE ];
```

파라미터

datashare_name

datashare의 이름입니다. datashare 이름은 클러스터 네임스페이스에서 고유해야 합니다.

[[SET] PUBLICACCESSIBLE]

공개적으로 액세스할 수 있는 클러스터와 datashare를 공유할 수 있는지 여부를 지정하는 절입니다.

SET PUBLICACCESSIBLE의 기본값은 FALSE입니다.

사용 노트

기본적으로 datashare의 소유자는 공유만 소유하고 공유 내의 객체는 소유하지 않습니다.

슈퍼 사용자와 데이터베이스 소유자만 CREATE DATASHARE를 사용하고 ALTER 권한을 다른 사용자 또는 그룹에 위임할 수 있습니다.

예시

다음 예에서는 datashare salesshare를 생성합니다.

```
CREATE DATASHARE salesshare;
```

다음 예에서는 AWS Data Exchange에서 관리하는 datashare demoshare를 생성합니다.

```
CREATE DATASHARE demoshare SET PUBLICACCESSIBLE TRUE, MANAGEDBY ADX;
```

CREATE EXTERNAL FUNCTION

Amazon Redshift에 대해 AWS Lambda를 기반으로 스칼라 사용자 정의 함수(UDF)를 생성합니다. Lambda 사용자 정의 함수에 대한 자세한 내용은 Scalar Lambda UDF 섹션을 참조하세요.

필수 권한

CREATE EXTERNAL FUNCTION에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- CREATE [OR REPLACE] EXTERNAL FUNCTION 권한이 있는 사용자

구문

```
CREATE [ OR REPLACE ] EXTERNAL FUNCTION external_fn_name ( [data_type] [, ...] )
```

```
RETURNS data_type
{ VOLATILE | STABLE }
LAMBDA 'lambda_fn_name'
IAM_ROLE { default | 'arn:aws:iam::<AWS ##-id>:role/<role-name>'
RETRY_TIMEOUT milliseconds
MAX_BATCH_ROWS count
MAX_BATCH_SIZE size [ KB | MB ];
```

파라미터

OR REPLACE

이름 및 입력 인수 데이터 형식이 같은 함수 또는 서명이 같은 함수인 경우 이런 함수가 이미 하나 존재하므로 기존 함수를 대체하도록 지정하는 절입니다. 함수는 똑같은 데이터 형식 집합을 정의하는 새 함수로만 바꿀 수 있습니다. 수퍼유저만이 함수를 바꿀 수 있습니다.

기존 함수와 이름은 같지만 서명이 다른 함수를 정의하면 새 함수가 생성됩니다. 즉, 함수 이름이 오버로드됩니다. 자세한 내용은 함수 이름 오버로드 단원을 참조하십시오.

external_fn_name

외부 함수의 이름입니다. 스키마 이름을 지정하는 경우(예: myschema.myfunction) 함수는 지정된 스키마를 사용하여 생성됩니다. 그렇지 않으면, 함수가 현재 스키마로 생성됩니다. 유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요.

모든 UDF 이름에 f_를 접두사로 사용하는 것이 좋습니다. Amazon Redshift는 UDF 이름용으로 f_ 접두사를 예약합니다. f_ 접두사를 사용하면 UDF 이름이 현재 또는 미래에 Amazon Redshift의 기본 제공 SQL 함수 이름과 충돌하지 않도록 할 수 있습니다. 자세한 내용은 UDF 이름 충돌 방지 단원을 참조하십시오.

data_type

입력 인수의 데이터 형식입니다. 자세한 내용은 데이터 타입 단원을 참조하십시오.

RETURNS data_type

함수에 의해 반환되는 값의 데이터 형식입니다. RETURNS 데이터 형식은 임의의 표준 Amazon Redshift 데이터 형식일 수 있습니다. 자세한 내용은 Python UDF 데이터 형식 단원을 참조하십시오.

VOLATILE | STABLE

쿼리 최적화 프로그램에 함수의 휘발성에 대해 알립니다.

최상으로 최적화하기 위해 함수에 대해 유효한 가장 엄격한 휘발성 범주로 함수에 레이블을 지정합니다. 가장 덜 엄격한 범주부터 시작해서 엄격성의 순으로 휘발성 범주를 나열하면 다음과 같습니다.

- VOLATILE
- STABLE

VOLATILE

인수가 같을 경우, 함수는 단일 명령문의 행에 대해서도 연속적인 호출에 대해 상이한 결과를 반환할 수 있습니다. 쿼리 옵티마이저는 휘발성 함수의 동작에 대해 가정을 할 수 없습니다. 휘발성 함수를 사용하는 쿼리는 모든 입력에 대해 함수를 재평가해야 합니다.

STABLE

인수가 동일하면 함수는 단일 문 내에서 처리되는 연속적인 호출에서 동일한 결과를 반환하도록 보장됩니다. 이 함수는 서로 다른 명령문에서 호출될 경우 서로 다른 결과를 반환할 수 있습니다. 이 범주를 사용하면 옵티마이저가 단일 문 내에서 함수가 호출되는 횟수를 줄일 수 있습니다.

선택한 엄격도가 함수에 유효하지 않은 경우 옵티마이저가 이 엄격도에 따라 일부 호출을 건너뛸 수 있다는 점에 유의하세요. 이로 인해 잘못된 결과 집합이 생성될 수 있습니다.

IMMUTABLE 절은 현재 Lambda UDF에서 지원되지 않습니다.

LAMBDA 'lambda_fn_name'

Amazon Redshift에서 호출하는 함수의 이름입니다.

AWS Lambda 함수를 생성하는 단계는 AWS Lambda 개발자 안내서의 콘솔로 Lambda 함수 생성 섹션을 참조하세요.

Lambda 함수에 필요한 권한에 대한 자세한 내용은 AWS Lambda 개발자 안내서의 AWS Lambda 권한 섹션을 참조하세요.

IAM_ROLE { default | 'arn:aws:iam::*<AWS ##-id>*:role/*<role-name>*'

기본 키워드를 사용하여 CREATE EXTERNAL FUNCTION 명령이 실행될 때 Amazon Redshift에서 기본값으로 설정되고 클러스터와 연결된 IAM 역할을 사용하도록 합니다.

클러스터가 인증 및 권한 부여에 사용하는 IAM 역할의 Amazon 리소스 이름(ARN)을 사용합니다. CREATE EXTERNAL FUNCTION 명령은 이 IAM 역할을 통해 Lambda 함수를 호출할 권한이 있습니다. 클러스터에 연결된 Lambda 함수를 호출할 수 있는 권한이 있는 기존 IAM 역할이 있는 경우

역할의 ARN을 대체할 수 있습니다. 자세한 내용은 Lambda UDF에 대한 권한 부여 파라미터 구성 단원을 참조하십시오.

다음은 IAM_ROLE 파라미터에 대한 구문을 나타낸 것입니다.

```
IAM_ROLE 'arn:aws:iam::aws-account-id:role/role-name'
```

RETRY_TIMEOUT milliseconds

Amazon Redshift가 재시도 백오프 지연에 사용하는 총 시간(밀리초)입니다.

실패한 쿼리에 대해 즉시 재시도하는 대신 Amazon Redshift는 백오프를 수행하고 재시도 사이에 일정 시간 동안 기다립니다. 그런 다음 Amazon Redshift는 모든 지연의 합계가 지정한 RETRY_TIMEOUT 값과 같거나 초과할 때까지 실패한 쿼리를 다시 실행하도록 요청을 재시도합니다. 기본값은 20,000밀리초입니다.

Lambda 함수가 호출되면 Amazon Redshift는 `TooManyRequestsException`, `EC2ThrottledException`, `ServiceException` 등의 오류를 수신하는 쿼리에 대해 재시도합니다.

RETRY_TIMEOUT 파라미터를 0밀리초로 설정하여 Lambda UDF에 대한 재시도를 방지할 수 있습니다.

MAX_BATCH_ROWS count

Amazon Redshift가 단일 람다 호출에 대한 단일 배치 요청으로 보내는 최대 행 수입니다.

이 파라미터의 최소값은 1입니다. 최대값은 INT_MAX 또는 2,147,483,647입니다.

이 파라미터는 선택 사항입니다. 기본값은 INT_MAX 또는 2,147,483,647입니다.

MAX_BATCH_SIZE size[KB|MB]

Amazon Redshift가 단일 람다 호출에 대한 단일 배치 요청으로 보내는 데이터 페이로드의 최대 크기입니다.

이 파라미터의 최소값은 1KB입니다. 최대값은 5MB입니다.

이 파라미터의 기본값은 5MB입니다.

KB 및 MB는 선택 사항입니다. 측정 단위를 설정하지 않으면 Amazon Redshift는 기본적으로 KB를 사용합니다.

사용 노트

Lambda UDF를 생성할 때 다음 사항을 고려하세요.

- 입력 인수에 대한 Lambda 함수 호출 순서는 고정되거나 보장되지 않습니다. 클러스터 구성에 따라 쿼리를 실행하는 인스턴스마다 다를 수 있습니다.
- 함수는 각 입력 인수에 한 번만 적용된다는 보장이 없습니다. Amazon Redshift와 AWS Lambda 간의 상호 작용으로 인해 동일한 입력으로 반복적인 호출이 발생할 수 있습니다.

예시

다음은 스칼라 Lambda 사용자 정의 함수(UDF)를 사용하는 예입니다.

Node.js Lambda 함수를 사용하는 스칼라 Lambda UDF 예

다음 예에서는 2개의 정수를 입력 인수로 사용하는 `exfunc_sum`이라는 외부 함수를 생성합니다. 이 함수는 정수 출력으로 합계를 반환합니다. 호출할 Lambda 함수의 이름은 `lambda_sum`입니다. 이 Lambda 함수에 사용되는 언어는 Node.js 12.x입니다. IAM 역할을 지정해야 합니다. 이 예에서는 `'arn:aws:iam::123456789012:user/johndoe'`를 IAM 역할로 사용합니다.

```
CREATE EXTERNAL FUNCTION exfunc_sum(INT,INT)
RETURNS INT
VOLATILE
LAMBDA 'lambda_sum'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-Exfunc-Test';
```

Lambda 함수는 요청 페이로드를 받아 각 행을 반복합니다. 단일 행의 모든 값은 응답 배열에 저장되는 해당 행의 합계를 계산하기 위해 추가됩니다. 결과 배열의 행 수는 요청 페이로드에서 수신된 행 수와 유사합니다.

JSON 응답 페이로드는 'results' 필드에 결과 데이터가 있어야 외부 기능에서 인식됩니다. Lambda 함수로 전송된 요청의 arguments 필드에는 데이터 페이로드가 포함되어 있습니다. 배치 요청의 경우 데이터 페이로드에 여러 행이 있을 수 있습니다. 다음 Lambda 함수는 요청 데이터 페이로드의 모든 행을 반복합니다. 또한 단일 행 내의 모든 값을 개별적으로 반복합니다.

```
exports.handler = async (event) => {
    // The 'arguments' field in the request sent to the Lambda function contains the
 data payload.
    var t1 = event['arguments'];
```

```
    // 'len(t1)' represents the number of rows in the request payload.
    // The number of results in the response payload should be the same as the number
 of rows received.
    const resp = new Array(t1.length);

    // Iterating over all the rows in the request payload.
    for (const [i, x] of t1.entries())
    {
        var sum = 0;
        // Iterating over all the values in a single row.
        for (const y of x) {
            sum = sum + y;
        }
        resp[i] = sum;
    }
    // The 'results' field should contain the results of the lambda call.
    const response = {
        results: resp
    };
    return JSON.stringify(response);
};
```

다음 예에서는 리터럴 값을 사용하여 외부 함수를 호출합니다.

```
select exfunc_sum(1,2);
exfunc_sum
------------
 3
(1 row)
```

다음 예에서는 정수 데이터 형식의 두 열 c1 및 c2가 있는 t_sum이라는 테이블을 생성하고 두 행의 데이터를 삽입합니다. 그런 다음 이 테이블의 열 이름을 전달하여 외부 함수를 호출합니다. 두 테이블 행은 단일 Lambda 호출로 요청 페이로드의 배치 요청으로 전송됩니다.

```
CREATE TABLE t_sum(c1 int, c2 int);
INSERT INTO t_sum VALUES (4,5), (6,7);
SELECT exfunc_sum(c1,c2) FROM t_sum;
 exfunc_sum
---------------
 9
 13
```

```
(2 rows)
```

RETRY_TIMEOUT 속성을 사용하는 Scalar Lambda UDF 예

다음 섹션에서 Lambda UDF에서 RETRY_TIMEOUT 속성을 사용하는 방법의 예를 찾아볼 수 있습니다.

AWS Lambda 함수에는 각 함수에 대해 설정할 수 있는 동시성 제한이 있습니다. 동시성 한도에 대한 자세한 내용은 AWS Lambda개발자 가이드의 Lambda 함수에 대한 동시성 관리 및 AWS 컴퓨팅 블로그의 게시물 AWS Lambda 함수 동시성 관리를 참조하세요.

Lambda UDF에서 제공하는 요청 수가 동시성 제한을 초과하면 새 요청에서 `TooManyRequestsException` 오류를 수신합니다. ambda UDF는 Lambda 함수로 전송된 요청 간의 모든 지연 합계가 설정한 RETRY_TIMEOUT 값과 같거나 초과할 때까지 이 오류에 대해 재시도합니다. 기본 RETRY_TIMEOUT 값은 20,000밀리초입니다.

다음 예시에서는 `exfunc_sleep_3`이라는 Lambda 함수를 생성합니다. 이 함수는 요청 페이로드를 받아 각 행을 반복하고 입력을 대문자로 변환합니다. 그런 다음 3초 동안 휴면하고 결과를 반환합니다. 이 Lambda 함수에 사용되는 언어는 Python 3.8입니다.

결과 배열의 행 수는 요청 페이로드에서 수신된 행 수와 유사합니다. JSON 응답 페이로드는 `results` 필드에 결과 데이터가 있어야 외부 기능에서 인식됩니다. Lambda 함수로 전송된 요청의 `arguments` 필드에는 데이터 페이로드가 포함되어 있습니다. 배치 요청의 경우 데이터 페이로드에 여러 행이 나타날 수 있습니다.

이 함수의 동시성 제한은 RETRY_TIMEOUT 속성의 사용을 보여주기 위해 예약된 동시성에서 특별히 1로 설정됩니다. 속성이 1로 설정되면 Lambda 함수는 한 번에 하나의 요청만 처리할 수 있습니다.

```
import json
import time
def lambda_handler(event, context):
    t1 = event['arguments']
    # 'len(t1)' represents the number of rows in the request payload.
    # The number of results in the response payload should be the same as the number of
 rows received.
    resp = [None]*len(t1)

    # Iterating over all rows in the request payload.
    for i, x in enumerate(t1):
        # Iterating over all the values in a single row.
```

```
        for j, y in enumerate(x):
            resp[i] = y.upper()

    time.sleep(3)
    ret = dict()
    ret['results'] = resp
    ret_json = json.dumps(ret)
    return ret_json
```

다음 두 가지 추가 예에서는 RETRY_TIMEOUT 속성을 보여줍니다. 각 예에서 단일 Lambda UDF를 호출합니다. Lambda UDF를 호출하는 동안 각 예에서는 동일한 SQL 쿼리를 실행하여 동시에 2개의 동시 데이터베이스 세션에서 Lambda UDF를 호출합니다. Lambda UDF를 호출하는 첫 번째 쿼리가 UDF에 의해 제공되는 경우 두 번째 쿼리는 `TooManyRequestsException` 오류를 수신합니다. 이 결과는 UDF에서 예약된 동시성을 1로 특별히 설정했기 때문에 발생합니다. Lambda 함수에 대해 예약된 동시성을 설정하는 방법에 대한 자세한 내용은 예약된 동시성 구성 섹션을 참조하세요.

다음 첫 번째 예에서는 Lambda UDF의 RETRY_TIMEOUT 속성을 0밀리초로 설정합니다. Lambda 요청이 Lambda 함수에서 예외를 수신하면 Amazon Redshift는 재시도하지 않습니다. RETRY_TIMEOUT 특성이 0으로 설정되어 있기 때문에 이러한 결과가 발생합니다.

```
CREATE OR REPLACE EXTERNAL FUNCTION exfunc_upper(varchar)
RETURNS varchar
VOLATILE
LAMBDA 'exfunc_sleep_3'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-Exfunc-Test'
RETRY_TIMEOUT 0;
```

RETRY_TIMEOUT을 0으로 설정하면 별도의 데이터베이스 세션에서 다음 두 쿼리를 실행하여 다른 결과를 볼 수 있습니다.

Lambda UDF를 사용하는 첫 번째 SQL 쿼리가 성공적으로 실행됩니다.

```
select exfunc_upper('Varchar');
 exfunc_upper
 --------------
 VARCHAR
(1 row)
```

별도의 데이터베이스 세션에서 동시에 실행되는 두 번째 쿼리는 `TooManyRequestsException` 오류를 수신합니다.

```
select exfunc_upper('Varchar');
ERROR:  Rate Exceeded.; Exception: TooManyRequestsException; ShouldRetry: 1
DETAIL:
-----------------------------------------------
error:  Rate Exceeded.; Exception: TooManyRequestsException; ShouldRetry: 1
code:      32103
context:query:     0
location:  exfunc_client.cpp:102
process:   padbmaster [pid=26384]
-----------------------------------------------
```

다음 두 번째 예에서는 Lambda UDF의 RETRY_TIMEOUT 속성을 3,000밀리초로 설정합니다. 두 번째 쿼리가 동시에 실행되더라도 총 지연 시간이 3,000밀리초가 될 때까지 Lambda UDF가 재시도합니다. 따라서 두 쿼리가 모두 성공적으로 실행됩니다.

```
CREATE OR REPLACE EXTERNAL FUNCTION exfunc_upper(varchar)
RETURNS varchar
VOLATILE
LAMBDA 'exfunc_sleep_3'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-Exfunc-Test'
RETRY_TIMEOUT 3000;
```

RETRY_TIMEOUT을 3,000밀리초로 설정하면 별도의 데이터베이스 세션에서 다음 두 쿼리를 실행하여 같은 결과를 볼 수 있습니다.

Lambda UDF를 실행하는 첫 번째 SQL 쿼리가 성공적으로 실행됩니다.

```
select exfunc_upper('Varchar');
 exfunc_upper
 --------------
 VARCHAR
(1 row)
```

두 번째 쿼리가 동시에 실행되고 총 지연 시간이 3,000밀리초가 될 때까지 Lambda UDF가 재시도합니다.

```
select exfunc_upper('Varchar');
 exfunc_upper
--------------
 VARCHAR
```

```
(1 row)
```

Python Lambda 함수를 사용하는 스칼라 Lambda UDF 예

다음 예에서는 exfunc_multiplication이라는 외부 함수를 생성하고 숫자를 곱하고 정수를 반환합니다. 이 예에서는 Lambda 응답에 성공 및 error_msg 필드를 통합합니다. 곱셈 결과에 정수 오버플로가 있는 경우 성공 필드가 false로 설정되고 error_msg 메시지가 Integer multiplication overflow로 설정됩니다. exfunc_multiplication 함수는 3개의 정수를 입력 인수로 사용하고 합계를 정수 출력으로 반환합니다.

호출되는 Lambda 함수의 이름은 lambda_multiplication입니다. 이 Lambda 함수에 사용되는 언어는 Python 3.8입니다. IAM 역할을 지정해야 합니다.

```
CREATE EXTERNAL FUNCTION exfunc_multiplication(int, int, int)
RETURNS INT
VOLATILE
LAMBDA 'lambda_multiplication'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-Exfunc-Test';
```

Lambda 함수는 요청 페이로드를 받아 각 행을 반복합니다. 단일 행의 모든 값을 곱하여 응답 목록에 저장되는 해당 행의 결과를 계산합니다. 이 예에서는 기본적으로 true로 설정된 부울 성공 값을 사용합니다. 행의 곱셈 결과에 정수 오버플로가 있으면 성공 값은 false로 설정됩니다. 그런 다음 반복 루프가 중단됩니다.

응답 페이로드를 생성하는 동안 성공 값이 false인 경우 다음 Lambda 함수는 페이로드에 error_msg 필드를 추가합니다. 또한 오류 메시지를 Integer multiplication overflow로 설정합니다. 성공 값이 true이면 결과 필드에 결과 데이터가 추가됩니다. 결과 배열(있는 경우)의 행 수는 요청 페이로드에서 수신된 행 수와 유사합니다.

Lambda 함수로 전송된 요청의 arguments 필드에는 데이터 페이로드가 포함되어 있습니다. 배치 요청의 경우 데이터 페이로드에 여러 행이 있을 수 있습니다. 다음 Lambda 함수는 요청 데이터 페이로드의 모든 행을 반복하고 단일 행 내의 모든 값을 개별적으로 반복합니다.

```
import json
def lambda_handler(event, context):
    t1 = event['arguments']
    # 'len(t1)' represents the number of rows in the request payload.
    # The number of results in the response payload should be the same as the number of
 rows received.
    resp = [None]*len(t1)
```

```
    # By default success is set to 'True'.
    success = True
    # Iterating over all rows in the request payload.
    for i, x in enumerate(t1):
        mul = 1
        # Iterating over all the values in a single row.
        for j, y in enumerate(x):
            mul = mul*y

        # Check integer overflow.
        if (mul >= 9223372036854775807 or mul <= -9223372036854775808):
            success = False
            break
        else:
            resp[i] = mul
    ret = dict()
    ret['success'] = success
    if not success:
        ret['error_msg'] = "Integer multiplication overflow"
    else:
        ret['results'] = resp
    ret_json = json.dumps(ret)

    return ret_json
```

다음 예에서는 리터럴 값을 사용하여 외부 함수를 호출합니다.

```
SELECT exfunc_multiplication(8, 9, 2);
  exfunc_multiplication
--------------------------
          144
(1 row)
```

다음 예에서는 3개(c1, c2, c3)의 정수 데이터 형식 열이 있는 t_multi라는 테이블을 생성합니다. 이 테이블의 열 이름을 전달하여 외부 함수를 호출합니다. 정수 오버플로가 오류 전파 방식을 표시하는 방식으로 데이터가 삽입됩니다.

```
CREATE TABLE t_multi (c1 int, c2 int, c3 int);
INSERT INTO t_multi VALUES (2147483647, 2147483647, 4);
SELECT exfunc_multiplication(c1, c2, c3) FROM t_multi;
DETAIL:
```

```
-----------------------------------------------
error:  Integer multiplication overflow
code:      32004context:
context:
query:     38
location:  exfunc_data.cpp:276
process:   query2_16_38 [pid=30494]
-----------------------------------------------
```

CREATE EXTERNAL MODEL

주제

CREATE EXTERNAL MODEL의 사전 조건

CREATE EXTERNAL MODEL 스테이트먼트를 사용하기 전에 먼저 Amazon Redshift 기계 학습 사용을 위한 클러스터 설정의 사전 조건을 충족해야 합니다. 다음은 사전 조건을 개괄적으로 요약한 것입니다.

- AWS 관리 콘솔 또는 AWS 명령줄 인터페이스(AWS CLI)를 사용하여 Amazon Redshift 클러스터를 생성합니다.
- 클러스터를 생성하는 동안 AWS Identity and Access Management(IAM) 정책을 연결합니다.
- Amazon Redshift와 Amazon Bedrock이 다른 서비스와 상호 작용하는 역할을 맡도록 허용하려면 IAM 역할에 적절한 신뢰 정책을 추가합니다.
- Amazon Bedrock 콘솔에서 사용하려는 특정 LLM에 대한 액세스를 활성화합니다.
- (선택 사항) 소량의 데이터라도 `Too many requests, please wait before trying again`과 같이 Amazon Bedrock에서 발생하는 스로틀링 예외가 있는 경우 Amazon Bedrock 계정의 Service Quotas에서 할당량을 확인합니다. 적용된 계정 수준 할당량이 사용 중인 모델의 InvokeModel 요청에 대한 AWS 기본 할당량 값과 적어도 동일한지 확인합니다.

IAM 역할, 신뢰 정책 및 기타 사전 조건에 대한 자세한 내용은 Amazon Redshift 기계 학습 사용을 위한 클러스터 설정 섹션을 참조하세요.

필수 권한

CREATE EXTERNAL MODEL에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- CREATE MODEL 권한이 있는 사용자
- GRANT CREATE MODEL 권한이 있는 역할

비용 관리

Amazon Redshift ML은 기존 클러스터 리소스를 사용하여 예측 모델을 생성하므로 추가 비용을 지불하지 않아도 됩니다. 그러나 선택한 모델에 따라 Amazon Bedrock 사용에 대해 AWS 요금이 부과됩니다. 자세한 내용은 Amazon Redshift 기계 학습 사용 비용을 참조하세요.

CREATE EXTERNAL MODEL 구문

다음은 CREATE EXTERNAL MODEL 스테이트먼트의 전체 구문입니다.

```
CREATE EXTERNAL MODEL model_name
FUNCTION function_name
IAM_ROLE {default/'arn:aws:iam::<account-id>:role/<role-name>'}
MODEL_TYPE BEDROCK
SETTINGS (
   MODEL_ID model_id
   [, PROMPT 'prompt prefix']
   [, SUFFIX 'prompt suffix']
   [, REQUEST_TYPE {RAW|UNIFIED}]
   [, RESPONSE_TYPE {VARCHAR|SUPER}]
);
```

CREATE EXTERNAL MODEL 명령은 콘텐츠를 만드는 데 사용하는 추론 함수를 생성합니다.

다음은 RAW의 REQUEST_TYPE을 사용하여 CREATE EXTERNAL MODEL을 생성하는 추론 함수의 구문입니다.

```
SELECT inference_function_name(request_super)
```

```
[FROM table];
```

다음은 UNIFIED의 REQUEST_TYPE을 사용하여 CREATE EXTERNAL MODEL을 생성하는 추론 함수의 구문입니다.

```
SELECT inference_function_name(input_text, [, inference_config [,
 additional_model_request_fields]])
[FROM table];
```

추론 함수 사용 방법에 관한 내용은 Amazon Redshift ML과 Amazon Bedrock의 통합용 외부 모델 사용 섹션을 참조하시기 바랍니다.

CREATE EXTERNAL MODEL 파라미터 및 설정

이 섹션에서는 CREATE EXTERNAL MODEL 명령의 파라미터와 설정에 대해 설명합니다.

주제

- CREATE EXTERNAL MODEL 파라미터
- CREATE EXTERNAL MODEL 설정

CREATE EXTERNAL MODEL 파라미터

model_name

외부 모델의 이름입니다. 스키마의 모델 이름은 고유해야 합니다.

FUNCTION function_name (data_type [,...])

CREATE EXTERNAL MODEL에서 생성되는 추론 함수의 이름입니다. 추론 함수를 사용하여 Amazon Bedrock에 요청을 보내고 ML에서 생성된 텍스트를 검색합니다.

IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }

Amazon Redshift가 Amazon Bedrock에 액세스하는 데 사용하는 IAM 역할입니다. IAM 역할에 대한 상세 정보는 Amazon Redshift ML과 Amazon Bedrock의 통합을 위한 IAM 역할 생성 또는 업데이트 단원을 참조하십시오.

MODEL_TYPE BEDROCK

모델 유형을 지정합니다. 유일한 유효 값은 BEDROCK입니다.

SETTINGS (MODEL_ID model_id [,...])

외부 모델 설정을 지정합니다. 자세한 내용은 다음 섹션을 참조하시기 바랍니다.

CREATE EXTERNAL MODEL 설정

MODEL_ID model_id

외부 모델의 식별자입니다(예: `anthropic.claude-v2`). Amazon Bedrock 모델 ID에 관한 자세한 내용은 Amazon Bedrock 모델 ID를 참조하시기 바랍니다.

PROMPT '프롬프트 접두사'

Amazon Redshift가 모든 추론 요청의 시작 부분에 추가하는 정적 프롬프트를 지정합니다. `UNIFIED`의 `REQUEST_TYPE`에서만 지원됩니다.

SUFFIX '프롬프트 접미사'

Amazon Redshift가 모든 추론 요청의 끝에 추가하는 정적 프롬프트를 지정합니다. `UNIFIED`의 `REQUEST_TYPE`에서만 지원됩니다.

REQUEST_TYPE { RAW | UNIFIED }

Amazon Bedrock으로 전송된 요청의 형식을 지정합니다. 유효한 값은 다음과 같습니다.

- RAW: 추론 함수는 입력을 단일 슈퍼 값으로 간주하고 항상 슈퍼 값을 반환합니다. 슈퍼 값의 형식은 선택한 Amazon Bedrock 모델에 따라 다릅니다. 슈퍼는 여러 알고리즘을 결합하여 향상된 단일 예측을 생성하는 예측 모델입니다.
- UNIFIED: 추론 함수는 통합 API를 사용합니다. 모든 모델은 Amazon Bedrock과 통합되고 일관된 인터페이스를 갖추고 있습니다. 이는 메시지를 지원하는 모든 모델에 적용됩니다. 이 값이 기본값입니다.

  자세한 내용은 Amazon Bedrock API 설명서에 나와 있는 Converse API 설명서를 참조하시기 바랍니다.

RESPONSE_TYPE { VARCHAR | SUPER }

응답의 형식을 지정합니다. `REQUEST_TYPE`이 `RAW`인 경우 `RESPONSE_TYPE`이 필요하며 유일한 유효한 값은 `SUPER`입니다. 다른 모든 `REQUEST  TYPE` 값의 경우 기본값은 `VARCHAR`이며 `RESPONSE_TYPE`은 선택 사항입니다. 유효한 값은 다음과 같습니다.

- VARCHAR: Amazon Redshift는 모델에서 생성된 텍스트 응답만 반환합니다.

- SUPER: Amazon Redshift는 모델에서 생성된 전체 응답 JSON을 슈퍼로 반환합니다. 여기에는 텍스트 응답과 함께 중단 이유, 모델 입력 및 출력 토큰 사용과 같은 정보가 포함됩니다. 슈퍼는 여러 알고리즘을 결합하여 향상된 단일 예측을 생성하는 예측 모델입니다.

CREATE EXTERNAL MODEL 추론 함수 파라미터

이 섹션에서는 `CREATE EXTERNAL MODEL` 명령으로 생성되는 추론 함수의 유효한 파라미터를 설명합니다.

`RAW`의 `REQUEST_TYPE`에 대한 CREATE EXTERNAL MODEL 추론 함수 파라미터

`RAW`의 `REQUEST_TYPE`로 생성된 추론 함수에는 슈퍼 입력 인수가 하나 있으며 항상 슈퍼 데이터 유형을 반환합니다. 입력 슈퍼의 구문은 Amazon Bedrock에서 선택한 특정 모델의 요청 구문을 따릅니다.

`UNIFIED`의 `REQUEST_TYPE`에 대한 CREATE EXTERNAL MODEL 추론 함수 파라미터

input_text

Amazon Redshift가 Amazon Bedrock에 보내는 텍스트입니다.

inference_config

Amazon Redshift가 Amazon Bedrock에 보내는 선택 사항 파라미터가 포함된 슈퍼 값입니다. 여기에는 다음이 포함될 수 있습니다.

- maxTokens
- stopSequences
- temperature
- topP

이러한 파라미터는 모두 선택 사항이며 대/소문자를 구분합니다. 이러한 파라미터에 관한 자세한 내용은 Amazon Bedrock API 참조의 InferenceConfiguration을 참조하시기 바랍니다.

CREATE EXTERNAL SCHEMA

현재 데이터베이스에서 새 외부 스키마를 생성합니다. 이 외부 스키마를 사용하여 Amazon RDS for PostgreSQL 또는 Amazon Aurora PostgreSQL 호환 버전 데이터베이스에 연결할 수 있습니다. 또한 AWS Glue, Athena와 같은 외부 데이터 카탈로그의 데이터베이스 또는 Amazon EMR과 같은 Apache Hive 메타스토어의 데이터베이스를 참조하는 외부 스키마를 생성할 수 있습니다.

이 스키마의 소유자는 CREATE EXTERNAL SCHEMA 명령의 발행자입니다. 외부 스키마의 소유권을 이전하려면 ALTER SCHEMA를 사용해 소유자를 변경합니다. 다른 사용자 또는 사용자 그룹에 스키마에 대한 액세스를 허용하려면 GRANT 명령을 사용합니다.

외부 테이블에서 권한에 대한 GRANT 또는 REVOKE 명령을 사용할 수 없습니다. 대신에, 외부 스키마에 대한 권한을 허용하거나 취소합니다.

> **Note**
>
> 현재 Amazon Athena 데이터 카탈로그에 Redshift Spectrum 외부 테이블이 있다면 Athena 데이터 카탈로그를 AWS Glue Data Catalog로 마이그레이션할 수 있습니다. AWS Glue Data Catalog를 Redshift Spectrum과 함께 사용하려면 AWS Identity and Access Management(IAM) 정책을 변경해야 할 수 있습니다. 자세한 내용은 Athena User Guide의 Upgrading to the AWS Glue Data Catalog 섹션을 참조하세요.

외부 스키마에 대한 세부 정보를 보려면 SVV_EXTERNAL_SCHEMAS 시스템 뷰를 쿼리하십시오.

구문

다음 구문은 외부 데이터 카탈로그를 사용하여 데이터를 참조하는 데 사용되는 CREATE EXTERNAL SCHEMA 명령에 대해 설명합니다. 자세한 내용은 Amazon Redshift Spectrum 단원을 참조하십시오.

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] local_schema_name
FROM [ [ DATA CATALOG ] | HIVE METASTORE | POSTGRES | MYSQL | KINESIS | MSK | REDSHIFT
 | KAFKA ]
[ DATABASE 'database_name' ]
[ SCHEMA 'schema_name' ]
[ REGION 'aws-region' ]
[ IAM_ROLE [ default | 'SESSION' | 'arn:aws:iam::<AWS ##-id>:role/<role-name>' ] ]
[ AUTHENTICATION [ none | iam | mtls] ]
[ AUTHENTICATION_ARN 'acm-certificate-arn' | SECRET_ARN 'ssm-secret- arn' ]
[ URI ['hive_metastore_uri' [ PORT port_number ] | 'hostname' [ PORT port_number ] |
 'Kafka bootstrap URL'] ]
[ CLUSTER_ARN 'arn:aws:kafka:<region>:<AWS ##-id>:cluster/msk/<cluster uuid>' ]
[ CATALOG_ROLE [ 'SESSION' | 'catalog-role-arn-string' ] ]
[ CREATE EXTERNAL DATABASE IF NOT EXISTS ]
[ CATALOG_ID 'Amazon Web Services account ID containing Glue or Lake Formation
 database' ]
```

다음 구문은 RDS POSTGRES 또는 Aurora PostgreSQL에 대한 연합 쿼리로 데이터를 참조하는 데 사용되는 CREATE EXTERNAL SCHEMA 명령을 설명합니다. Kinesis Data Streams와 같은 스트리밍 소스를 참조하는 외부 스키마를 생성할 수도 있습니다. 자세한 내용은 Amazon Redshift에서 연합 쿼리를 사용하여 데이터 쿼리 단원을 참조하십시오.

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] local_schema_name
FROM POSTGRES
DATABASE 'federated_database_name' [SCHEMA 'schema_name']
URI 'hostname' [ PORT port_number ]
IAM_ROLE [ default | 'arn:aws:iam::<AWS ##-id>:role/<role-name>' ]
SECRET_ARN 'ssm-secret-arn'
```

다음 구문은 RDS MySQL 또는 Aurora MySQL에 대한 연합 쿼리로 데이터를 참조하는 데 사용되는 CREATE EXTERNAL SCHEMA 명령을 설명합니다. 자세한 내용은 Amazon Redshift에서 연합 쿼리를 사용하여 데이터 쿼리 단원을 참조하십시오.

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] local_schema_name
FROM MYSQL
DATABASE 'federated_database_name'
URI 'hostname' [ PORT port_number ]
IAM_ROLE [ default | 'arn:aws:iam::<AWS ##-id>:role/<role-name>' ]
SECRET_ARN 'ssm-secret-arn'
```

다음 구문은 Kinesis 스트림 데이터를 참조하는 데 사용되는 CREATE EXTERNAL SCHEMA 명령에 대해 설명합니다. 자세한 내용은 구체화된 뷰로 스트리밍 모으기 단원을 참조하십시오.

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] schema_name
FROM KINESIS
IAM_ROLE [ default | 'arn:aws:iam::<AWS ##-id>:role/<role-name>' ]
```

다음 구문은 Amazon Managed Streaming for Apache Kafka 또는 Confluent Cloud 클러스터와 해당 수집 대상 주제를 참조하는 데 사용되는 CREATE EXTERNAL SCHEMA 명령을 설명합니다. 연결하려면 브로커 URI을 제공하면 됩니다. 자세한 내용은 구체화된 뷰로 스트리밍 모으기 단원을 참조하십시오.

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] schema_name
FROM KAFKA
[ IAM_ROLE [ default | 'arn:aws:iam::<AWS ##-id>:role/<role-name>' ] ]
URI 'Kafka bootstrap URI'
AUTHENTICATION [ none | iam | mtls ]
[ AUTHENTICATION_ARN 'acm-certificate-arn' | SECRET_ARN 'ssm-secret- arn' ];
```

다음 구문은 데이터베이스 간 쿼리로 데이터를 참조하는 데 사용되는 CREATE EXTERNAL SCHEMA 명령을 설명합니다.

```
CREATE EXTERNAL SCHEMA local_schema_name
FROM  REDSHIFT
DATABASE 'redshift_database_name' SCHEMA 'redshift_schema_name'
```

파라미터

IF NOT EXISTS

지정된 스키마가 이미 존재하는 경우 오류 메시지와 함께 종료하는 대신, 명령이 아무 것도 변경하지 않고 스키마가 존재한다는 메시지를 반환함을 나타내는 절입니다. 이 절은 스크립트 작성 시 유용하므로, CREATE EXTERNAL SCHEMA가 이미 존재하는 스키마의 생성을 시도하는 경우에는 스크립트가 실패하지 않습니다.

local_schema_name

새 외부 스키마의 이름입니다. 유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요.

FROM [DATA CATALOG] | HIVE METASTORE | POSTGRES | MYSQL | KINESIS | MSK | REDSHIFT

외부 데이터베이스가 있는 위치를 나타내는 키워드입니다.

DATA CATALOG는 외부 데이터베이스가 Athena 데이터 카탈로그 또는 AWS Glue Data Catalog에 정의되어 있음을 나타냅니다.

외부 데이터베이스가 다양한 AWS 리전에서 외부 Data Catalog에 정의되어 있는 경우 REGION 파라미터가 필요합니다. DATA CATALOG는 기본값입니다.

HIVE METASTORE는 외부 데이터베이스가 Apache Hive 메타스토어에 정의되어 있음을 나타냅니다. HIVE METASTORE가 지정되는 경우 URI가 필요합니다.

POSTGRES는 외부 데이터베이스가 RDS PostgreSQL 또는 Aurora PostgreSQL에 정의되어 있음을 나타냅니다.

MYSQL은 외부 데이터베이스가 RDS MySQL 또는 Aurora MySQL에 정의되어 있음을 나타냅니다.

KINESIS는 데이터 원본이 Kinesis Data Streams의 스트림임을 나타냅니다.

MSK는 데이터 소스가 Amazon MSK 프로비저닝 또는 서버리스 클러스터임을 나타냅니다.

KAFKA는 데이터 소스가 Kafka 클러스터임을 나타냅니다. Amazon MSK와 Confluent Cloud 모두에 이 키워드를 사용할 수 있습니다.

FROM REDSHIFT

데이터베이스가 Amazon Redshift에 있음을 나타내는 키워드입니다.

DATABASE 'redshift_database_name' SCHEMA 'redshift_schema_name'

Amazon Redshift 데이터베이스의 이름입니다.

redshift_schema_name은 Amazon Redshift의 스키마를 나타냅니다. 기본 redshift_schema_name은 `public`입니다.

DATABASE 'federated_database_name'

키워드는 지원되는 PostgreSQL 또는 MySQL 데이터베이스 엔진에서 외부 데이터베이스의 이름을 나타냅니다.

[SCHEMA 'schema_name']

schema_name은 지원되는 PostgreSQL 데이터베이스 엔진의 스키마를 나타냅니다. 기본 schema_name은 `public`입니다.

지원되는 MySQL 데이터베이스 엔진에 대한 연합 쿼리를 설정할 때 SCHEMA를 지정할 수 없습니다.

REGION 'aws-region'

외부 데이터베이스가 Athena 데이터 카탈로그 또는 AWS Glue Data Catalog에 정의되어 있는 경우 데이터베이스가 있는 AWS 리전입니다. 이 파라미터는 데이터베이스가 외부 Data Catalog에 정의되어 있는 경우에 필요합니다.

URI ['hive_metastore_uri' [PORT port_number] | 'hostname' [PORT port_number] | 'Kafka bootstrap URI']

지원되는 PostgreSQL 또는 MySQL 데이터베이스 엔진의 호스트 이름 URI와 port_number입니다. hostname은 복제본 세트의 헤드 노드입니다. 엔드포인트는 Amazon Redshift 클러스터에서 연결(라우팅)할 수 있어야 합니다. 기본 PostgreSQL 포트 번호는 5432입니다. 기본 MySQL 포트 번호는 3306입니다.

> **Note**
>
> 지원되는 PostgreSQL 또는 MySQL 데이터베이스 엔진은 Amazon Redshift 및 RDS url-rsPostgreSQL 또는 Aurora PostgreSQL을 연결하는 보안 그룹을 포함하는 Amazon Redshift 클러스터와 동일한 VPC에 있어야 합니다. 또한, Enhanced VPC Routing을 사용

하여 VPC 간 사용 사례를 구성할 수 있습니다. 자세한 내용은 Redshift 관리형 VPC 엔드포인트를 참조하세요.

메타스토어 URI 지정

데이터베이스가 Hive 메타스토어에 있는 경우 메타스토어의 URI를 지정하고 선택적으로 포트 번호를 지정합니다. 기본 포트 번호는 9083입니다.

URI에는 프로토콜 사양("http://")이 포함되어 있지 않습니다. 유효한 URI의 예: `uri '172.10.10.10'`.

스트리밍 수집용 브로커 URI 지정

부트스트랩 브로커 URI를 포함하면 Amazon MSK 또는 Confluent Cloud 클러스터에 연결하고 스트리밍 데이터를 수신할 수 있습니다. 자세한 내용과 예시를 확인하려면 Amazon Managed Streaming for Apache Kafka에서 스트리밍 수집 시작을 참조하세요.

IAM_ROLE [default | 'SESSION' | 'arn:aws:iam::*<AWS ##-id>*:role/*<role-name>*']

기본 키워드를 사용하여 CREATE EXTERNAL SCHEMA 명령이 실행될 때 Amazon Redshift에서 기본값으로 설정되고 클러스터와 연결된 IAM 역할을 사용하도록 합니다.

페더레이션형 ID를 사용하여 Amazon Redshift 클러스터에 연결하고 이 명령을 사용하여 생성된 외부 스키마에서 테이블에 액세스하는 경우에 `'SESSION'`을 사용합니다. 자세한 내용은 페더레이션형 ID 구성 방법이 설명된 페더레이션형 ID를 사용하여 로컬 리소스 및 Amazon Redshift Spectrum 외부 테이블에 대한 Amazon Redshift 액세스 관리 섹션을 참조하세요. ARN 대신 `'SESSION'`을 사용하는 이 구성은 `DATA CATALOG`를 사용하여 스키마를 생성한 경우에만 사용할 수 있다는 점에 유의하세요.

클러스터가 인증 및 권한 부여에 사용하는 IAM 역할의 Amazon 리소스 이름(ARN)을 사용합니다. 최소 IAM 역할은 액세스되는 Amazon S3 버킷에서 LIST 작업을 수행하고 버킷에 포함된 Amazon S3 객체에 대한 GET 작업을 수행할 수 있는 권한이 있어야 합니다.

다음은 단일 ARN에 대한 IAM_ROLE 파라미터의 구문을 보여줍니다.

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-name>'
```

역할을 함께 묶어 클러스터가 다른 계정에 속한 다른 IAM 역할을 수임하도록 할 수 있습니다. 최대 10개의 역할을 함께 묶을 수 있습니다. 역할 연쇄의 예는 Amazon Redshift Spectrum에서 IAM 역할 연결를 참조하세요.

이 IAM 역할에 다음과 유사한 IAM 권한 정책을 연결합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessSecret",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecretVersionIds"
            ],
            "Resource": "arn:aws:secretsmanager:us-west-2:123456789012:secret:my-
rds-secret-VNenFy"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword",
                "secretsmanager:ListSecrets"
            ],
            "Resource": "*"
        }
    ]
}
```

연합 쿼리에 사용할 IAM 역할을 생성하는 단계는 연합 쿼리 사용을 위해 비밀 및 IAM 역할 생성 섹션을 참조하세요.

> **Note**
>
> 연결된 역할 목록에 공백을 포함하지 마십시오.

다음은 세 역할을 함께 묶기 위한 구문을 나타낸 것입니다.

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-1-name>,arn:aws:iam::<aws-
account-id>:role/<role-2-name>,arn:aws:iam::<aws-account-id>:role/<role-3-name>'
```

SECRET_ARN 'ssm-secret-arn'

AWS Secrets Manager를 사용하여 생성한 지원되는 PostgreSQL 또는 MySQL 데이터베이스 엔진 암호의 Amazon 리소스 이름(ARN)입니다. 보안 암호에 대한 ARN을 생성하고 가져오는 방법에 대한 자세한 내용은 AWS Secrets Manager User Guide의 Creating a Basic Secret 및 Retrieving the Secret Value Secret 섹션을 참조하세요.

CATALOG_ROLE ['SESSION' | catalog-role-arn-string]

`'SESSION'`은 데이터 카탈로그에 대한 인증 및 권한 부여를 위해 페더레이션형 ID를 사용하여 Amazon Redshift 클러스터에 연결하는 데 사용합니다. 페더레이션형 ID 단계를 완료하는 방법에 대한 자세한 내용은 페더레이션형 ID를 사용하여 로컬 리소스 및 Amazon Redshift Spectrum 외부 테이블에 대한 Amazon Redshift 액세스 관리를 참조하세요. `'SESSION'` 역할은 스키마가 DATA CATALOG에서 생성된 경우에만 사용할 수 있다는 점에 유의하세요.

클러스터가 데이터 카탈로그에 대한 인증 및 권한 부여에 사용하는 IAM 역할의 Amazon 리소스 이름(ARN)을 사용합니다.

CATALOG_ROLE을 지정하지 않으면 Amazon Redshift는 지정된 IAM_ROLE을 사용합니다. 카탈로그 역할은 AWS Glue 또는 Athena에 있는 Data Catalog에 액세스할 수 있는 권한이 있어야 합니다. 자세한 내용은 Amazon Redshift Spectrum에 대한 IAM 정책 단원을 참조하십시오.

다음은 단일 ARN에 대한 CATALOG_ROLE 파라미터의 구문을 보여줍니다.

```
CATALOG_ROLE 'arn:aws:iam::<aws-account-id>:role/<catalog-role>'
```

역할을 함께 묶어 클러스터가 다른 계정에 속한 다른 IAM 역할을 수임하도록 할 수 있습니다. 최대 10개의 역할을 함께 묶을 수 있습니다. 자세한 내용은 Amazon Redshift Spectrum에서 IAM 역할 연결 단원을 참조하십시오.

> **Note**
>
> 함께 묶은 역할 목록에 공백을 포함하면 안 됩니다.

다음은 세 역할을 함께 묶기 위한 구문을 나타낸 것입니다.

```
CATALOG_ROLE 'arn:aws:iam::<aws-account-id>:role/<catalog-role-1-
name>,arn:aws:iam::<aws-account-id>:role/<catalog-role-2-name>,arn:aws:iam::<aws-
account-id>:role/<catalog-role-3-name>'
```

CREATE EXTERNAL DATABASE IF NOT EXISTS

지정된 외부 데이터베이스가 존재하지 않는 경우 DATABASE 인수로 지정된 이름을 가진 외부 데이터베이스를 생성하는 절입니다. 지정된 외부 데이터베이스가 존재하는 경우 이 명령을 실행해도 아무런 변화가 없습니다. 이 경우에는 명령 실행 시 오류 메시지와 함께 종료되지 않고 외부 데이터베이스가 존재한다는 메시지가 반환됩니다.

> **Note**
>
> CREATE EXTERNAL DATABASE IF NOT EXISTS를 HIVE METASTORE와 함께 사용할 수 없습니다.
> Data Catalog가 AWS Lake Formation에 사용되는 상태에서 CREATE EXTERNAL DATABASE IF NOT EXISTS를 사용하려면 Data Catalog에 대한 CREATE_DATABASE 권한이 필요합니다.

CATALOG_ID 'Glue 또는 Lake Formation 데이터베이스가 포함된 Amazon Web Services 계정 ID'

데이터 카탈로그 데이터베이스가 저장되는 계정 ID입니다.

CATALOG_ID는 다음 중 하나를 설정하여 데이터 카탈로그에 대한 인증 및 권한 부여를 위해 페더레이션형 ID를 사용하여 Amazon Redshift 클러스터 또는 Amazon Redshift Serverless에 연결하려는 경우에만 지정할 수 있습니다.

- CATALOG_ROLE~'SESSION'
- IAM_ROLE을 'SESSION'으로, 'CATALOG_ROLE'을 기본값으로 설정

페더레이션형 ID 단계를 완료하는 방법에 대한 자세한 내용은 [페더레이션형 ID를 사용하여 로컬 리소스 및 Amazon Redshift Spectrum 외부 테이블에 대한 Amazon Redshift 액세스 관리]를 참조하세요.

인증

스트리밍 수집에 대해 정의된 인증 유형입니다. 인증 유형이 있는 스트리밍 수집은 Amazon Managed Streaming for Apache Kafka와 함께 작동합니다. AUTHENTICATION 유형은 다음과 같습니다.

- none - 필요한 인증이 없다고 지정합니다. 이는 Apache Kafka에서 TLS를 사용하는 MSK 또는 일반 텍스트에 대한 인증되지 않은 액세스에 해당합니다.

- iam - IAM 인증을 지정합니다. 이 항목을 선택할 때는 IAM 역할에 IAM 인증 권한이 있는지 확인해야 합니다. 외부 스키마 정의에 대한 자세한 내용은 Apache Kafka 소스에서 스트리밍 수집 시작하기을 참조하세요.
- mtls - 클라이언트와 서버 간의 인증을 용이하게 하여 상호 전송 계층 보안을 통해 안전한 통신을 제공하도록 지정합니다. 이 경우 클라이언트는 Redshift이고 서버는 Amazon MSK입니다. mTLS를 사용하여 스트리밍 수집 구성에 관한 자세한 내용은 Apache Kafka 소스에서 Redshift 스트리밍 수집을 위한 mTLS 인증 섹션을 참조하시기 바랍니다.

AUTHENTICATION_ARN

Amazon Redshift가 Amazon MSK에서 mtls 인증 시 사용하는 AWS Certificate Manager 인증서의 ARN입니다. ARN은 발급된 인증서를 선택하면 ACM 콘솔에서 사용할 수 있습니다.

CLUSTER_ARN

스트리밍 수집의 경우, CLUSTER_ARN은 스트리밍 출처인 Amazon Managed Streaming for Apache Kafka 클러스터의 클러스터 식별자입니다. CLUSTER_ARN을 사용할 때는 `kafka:GetBootstrapBrokers` 권한이 포함된 IAM 역할 정책이 필요합니다. 이 옵션은 이전 버전과의 호환성을 위해 제공됩니다. 현재 부트스트랩 브로커 URI 옵션을 사용하여 Amazon Managed Streaming for Apache Kafka 클러스터에 연결하는 것이 좋습니다. 자세한 내용은 스트리밍 수집을 참조하세요.

사용 노트

Athena Data Catalog 사용 시 제한 사항은 AWS 일반 참조의 Athena 제한 사항을 참조하세요.

AWS Glue Data Catalog 사용 시 제한 사항은 AWS 일반 참조의 AWS Glue 제한 사항을 참조하세요.

Hive 메타스토어에는 이러한 제한이 적용되지 않습니다.

데이터베이스당 스키마 수는 최대 9,900개입니다. 자세한 내용은 Amazon Redshift 관리 가이드의 할당량 및 제한 섹션을 참조하세요.

스키마의 등록을 취소하려면 DROP SCHEMA 명령을 사용합니다.

외부 스키마에 대한 세부 정보를 보려면 다음 시스템 보기를 쿼리합니다.

- SVV_EXTERNAL_SCHEMAS
- SVV_EXTERNAL_TABLES

- SVV_EXTERNAL_COLUMNS

예시

다음 예에서는 미국 서부(오레곤)에서 sampledb로 명명된 데이터 카탈로그에 있는 데이터베이스를 사용하여 외부 스키마를 생성합니다. 이 예시를 Athena 또는 AWS Glue 데이터 카탈로그와 함께 사용하세요.

```
create external schema spectrum_schema
from data catalog
database 'sampledb'
region 'us-west-2'
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole';
```

다음 예에서는 외부 스키마를 생성하고 spectrum_db로 명명된 새로운 외부 데이터베이스를 생성합니다.

```
create external schema spectrum_schema
from data catalog
database 'spectrum_db'
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole'
create external database if not exists;
```

다음 예에서는 hive_db로 명명된 Hive 메타스토어 데이터베이스를 사용하여 외부 스키마를 생성합니다.

```
create external schema hive_schema
from hive metastore
database 'hive_db'
uri '172.10.10.10' port 99
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole';
```

다음 예에서는 Amazon S3 액세스에 myS3Role 역할을 사용하고 데이터 카탈로그 액세스에 myAthenaRole을 사용하기 위해 역할을 함께 묶습니다. 자세한 내용은 Amazon Redshift Spectrum에서 IAM 역할 연결 단원을 참조하십시오.

```
create external schema spectrum_schema
from data catalog
database 'spectrum_db'
```

```
iam_role 'arn:aws:iam::123456789012:role/myRedshiftRole,arn:aws:iam::123456789012:role/
myS3Role'
catalog_role 'arn:aws:iam::123456789012:role/myAthenaRole'
create external database if not exists;
```

다음 예에서는 외부 Aurora PostgreSQL 데이터베이스를 참조하는 외부 스키마를 생성합니다.

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] myRedshiftSchema
FROM POSTGRES
DATABASE 'my_aurora_db' SCHEMA 'my_aurora_schema'
URI 'endpoint to aurora hostname' PORT 5432
IAM_ROLE 'arn:aws:iam::123456789012:role/MyAuroraRole'
SECRET_ARN 'arn:aws:secretsmanager:us-east-2:123456789012:secret:development/
MyTestDatabase-AbCdEf'
```

다음 예에서는 소비자 클러스터에서 가져온 sales_db를 참조하는 외부 스키마를 생성합니다.

```
CREATE EXTERNAL SCHEMA sales_schema FROM REDSHIFT DATABASE 'sales_db' SCHEMA 'public';
```

다음 예에서는 외부 Aurora MySQL 데이터베이스를 참조하는 외부 스키마를 생성합니다.

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] myRedshiftSchema
FROM MYSQL
DATABASE 'my_aurora_db'
URI 'endpoint to aurora hostname'
IAM_ROLE 'arn:aws:iam::123456789012:role/MyAuroraRole'
SECRET_ARN 'arn:aws:secretsmanager:us-east-2:123456789012:secret:development/
MyTestDatabase-AbCdEf'
```

CREATE EXTERNAL TABLE

지정된 스키마에서 새 외부 테이블을 생성합니다. 모든 외부 테이블은 외부 스키마에 생성해야 합니다. 외부 스키마와 외부 테이블에 대해서는 검색 경로가 지원되지 않습니다. 자세한 내용은 CREATE EXTERNAL SCHEMA 단원을 참조하십시오.

CREATE EXTERNAL TABLE 명령을 사용하여 생성되는 외부 테이블 외에도 Amazon Redshift는 AWS Glue 또는 AWS Lake Formation 카탈로그나 Apache Hive 메타스토어에 정의된 외부 테이블을 참조할 수 있습니다. CREATE EXTERNAL SCHEMA 명령을 사용하면 외부 카탈로그에 정의된 외부 데이터베이스를 등록하고 Amazon Redshift에서 외부 테이블을 사용하도록 할 수 있습니다. AWS

Glue 또는 AWS Lake Formation 카탈로그 또는 Hive 메타스토어에 외부 테이블이 존재하는 경우 CREATE EXTERNAL TABLE을 사용하여 테이블을 생성하지 않아도 됩니다. 외부 테이블을 보려면 SVV_EXTERNAL_TABLES 시스템 뷰를 쿼리하십시오.

CREATE EXTERNAL TABLE AS 명령을 실행하여 쿼리의 열 정의를 기반으로 외부 테이블을 생성하고 해당 쿼리의 결과를 Amazon S3에 작성할 수 있습니다. 결과는 Apache Parquet 또는 구분된 텍스트 형식입니다. 외부 테이블에 파티션 키가 있는 경우 Amazon Redshift는 해당 파티션 키에 따라 새 파일을 분할하고 새 파티션을 외부 카탈로그에 자동으로 등록합니다. CREATE EXTERNAL TABLE AS에 대한 자세한 내용은 사용 노트 섹션을 참조하세요.

다른 Amazon Redshift 테이블에 사용하는 것과 같은 SELECT 구문을 사용하여 외부 테이블을 쿼리할 수 있습니다. INSERT 구문을 사용하여 Amazon S3의 외부 테이블 위치에 새 파일을 작성할 수도 있습니다. 자세한 내용은 INSERT(외부 테이블) 단원을 참조하십시오.

외부 테이블로 보기를 새로 만들려면 CREATE VIEW 문에 WITH NO SCHEMA BINDING 절을 포함시키십시오.

트랜잭션(BEGIN … END) 내에서 CREATE EXTERNAL TABLE을 실행할 수 없습니다. 버전 관리에 대한 자세한 내용은 직렬화 가능 격리 단원을 참조하십시오.

필수 권한

외부 테이블을 생성하려면 외부 스키마의 소유자이거나 수퍼유저여야 합니다. 외부 스키마의 소유권을 이전하려면 ALTER SCHEMA를 사용해 소유자를 변경합니다. 외부 테이블에 대한 액세스는 외부 스키마에 대한 액세스에 의해 제어됩니다. 외부 테이블에 대한 권한을 GRANT 또는 REVOKE할 수 없습니다. 대신에, 외부 스키마에 대한 USAGE를 허용하거나 취소합니다.

사용 노트에는 외부 테이블의 특정 권한에 대한 추가 정보가 있습니다.

구문

```
CREATE EXTERNAL TABLE
external_schema.table_name
(column_name data_type [, …] )
[ PARTITIONED BY (col_name data_type [, … ] )]
[ { ROW FORMAT DELIMITED row_format |
  ROW FORMAT SERDE 'serde_name'
  [ WITH SERDEPROPERTIES ( 'property_name' = 'property_value' [, ...] ) ] } ]
STORED AS file_format
LOCATION { 's3://bucket/folder/' | 's3://bucket/manifest_file' }
```

```
[ TABLE PROPERTIES ( 'property_name'='property_value' [, ...] ) ]
```

다음은 CREATE EXTERNAL TABLE AS의 구문입니다.

```
CREATE EXTERNAL TABLE
external_schema.table_name
[ PARTITIONED BY (col_name [, … ] ) ]
[ ROW FORMAT DELIMITED row_format ]
STORED AS file_format
LOCATION { 's3://bucket/folder/' }
[ TABLE PROPERTIES ( 'property_name'='property_value' [, ...] ) ]
 AS
 { select_statement }
```

파라미터

external_schema.table_name

생성한 후 외부 스키마 이름으로 정규화할 테이블의 이름입니다. 외부 테이블은 외부 스키마에 생성해야 합니다. 자세한 내용은 CREATE EXTERNAL SCHEMA 단원을 참조하십시오.

테이블 이름의 최대 길이는 127바이트이며, 이보다 긴 이름은 127바이트까지 표시되고 나머지는 잘립니다. 최대 4바이트까지 UTF-8 멀티바이트 문자를 사용할 수 있습니다. Amazon Redshift는 사용자 정의 임시 테이블과 쿼리 처리 또는 시스템 유지 관리 중에 Amazon Redshift에서 생성한 임시 테이블을 포함하여 클러스터당 테이블을 9,900개로 제한합니다. 선택적으로, 데이터베이스 이름으로 테이블 이름을 정규화할 수 있습니다. 다음 예에서는 데이터베이스 이름이 `spectrum_db`이고, 외부 스키마 이름은 `spectrum_schema`이며, 테이블 이름은 `test`입니다.

```
create external table spectrum_db.spectrum_schema.test (c1 int)
stored as parquet
location 's3://amzn-s3-demo-bucket/myfolder/';
```

지정된 데이터베이스 또는 스키마가 존재하지 않는 경우 테이블이 생성되지 않으며, 이 명령문은 오류를 반환합니다. 시스템 데이터베이스 `template0`, `template1`, `padb_harvest` 또는 `sys:internal`에서 테이블 또는 뷰를 생성할 수 없습니다.

테이블 이름은 지정된 스키마에 대한 고유한 이름이어야 합니다.

유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요.

(column_name data_type)

생성되는 각 열의 이름과 데이터 형식입니다.

열 이름의 최대 길이는 127바이트이며, 이보다 긴 이름은 127바이트까지 표시되고 나머지는 잘립니다. 최대 4바이트까지 UTF-8 멀티바이트 문자를 사용할 수 있습니다. 열 이름 "$path" 또는 "$size"는 지정할 수 없습니다. 유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요.

기본적으로 Amazon Redshift에서는 가상 열 $path 및 $size가 있는 외부 테이블을 생성합니다. spectrum_enable_pseudo_columns 구성 파라미터를 false로 설정하여 세션에 대해 가상 열 생성을 비활성화할 수 있습니다. 자세한 내용은 가상 열 단원을 참조하십시오.

가상 열이 활성화되어 있으면 단일 테이블에서 정의할 수 있는 최대 열 개수는 1,598개입니다. 가상 열이 활성화되어 있지 않으면 테이블 하나에서 정의할 수 있는 최대 열 개수는 1,600개입니다.

"넓은 테이블"을 생성할 경우에는 로드 및 쿼리 처리 중에 열의 목록이 중간 결과에 대한 행 너비 경계를 초과하지 않도록 주의하십시오. 자세한 내용은 사용 노트 단원을 참조하십시오.

CREATE EXTERNAL TABLE AS 명령의 경우 열이 쿼리에서 파생되기 때문에 열 목록이 필요하지 않습니다.

data_type

다음 모듈을 지원합니다.데이터 타입

- SMALLINT (INT2)
- INTEGER (INT, INT4)
- BIGINT (INT8)
- DECIMAL (NUMERIC)
- REAL (FLOAT4)
- DOUBLE PRECISION (FLOAT8)
- BOOLEAN (BOOL)
- CHAR (CHARACTER)
- VARCHAR (CHARACTER VARYING)
- VARBYTE(문자 가변) - Parquet 및 ORC 데이터 파일과 함께 사용할 수 있으며 비분할 테이블에만 사용할 수 있습니다.

- DATE - 텍스트, Parquet 또는 ORC 데이터 파일과 함께 또는 파티션 열로만 사용할 수 있습니다.
- TIMESTAMP

DATE의 경우 다음과 같은 형식을 사용할 수 있습니다. 숫자를 사용하여 표시되는 월 값의 경우 다음 형식이 지원됩니다.

- `mm-dd-yyyy` 예: `05-01-2017`. 이 값이 기본값입니다.
- `yyyy-mm-dd` - 여기서 연도는 2자리 이상의 숫자로 표시됩니다. 예: `2017-05-01`.

세 글자 약어를 사용하여 표시되는 월 값의 경우 다음 형식이 지원됩니다.

- `mmm-dd-yyyy` 예: `may-01-2017`. 이 값이 기본값입니다.
- `dd-mmm-yyyy` - 여기서 연도는 2자리 이상의 숫자로 표시됩니다. 예: `01-may-2017`.
- `yyyy-mmm-dd` - 여기서 연도는 2자리 이상의 숫자로 표시됩니다. 예: `2017-may-01`.

지속적으로 100보다 작은 연도 값의 경우 연도는 다음 방식으로 계산됩니다.

- 연도가 70보다 작은 경우 연도는 연도에 2000을 더한 값으로 계산됩니다. 예를 들어 `mm-dd-yyyy` 형식의 날짜 05-01-17은 `05-01-2017`로 변환됩니다.
- 연도가 100보다 작고 69보다 큰 경우 연도는 연도에 1900을 더한 값으로 계산됩니다. 예를 들어 `mm-dd-yyyy` 형식의 날짜 05-01-89는 `05-01-1989`로 변환됩니다.
- 2자리로 표시되는 연도 값의 경우 앞에 0을 추가하여 연도를 4자리로 표시합니다.

타임스탬프 값 `yyyy-mm-dd HH:mm:ss.SSSSSS`에서 볼 수 있듯이, 텍스트 파일의 타임스탬프 값은 `2017-05-01 11:30:59.000000` 형식이어야 합니다.

VARCHAR 열의 길이 단위는 문자 수가 아니라 바이트로 정의됩니다. 예를 들어 VARCHAR(12) 열에는 단일 바이트 문자 12개, 2바이트 문자 6개가 포함될 수 있습니다. 외부 테이블을 쿼리하는 경우 오류를 반환하지 않고 정의된 열 크기에 맞춰 결과가 잘립니다. 자세한 내용은 스토리지 및 범위 단원을 참조하십시오.

최상의 성능을 위해 데이터에 맞춰 가장 작은 열 크기를 지정하는 것이 좋습니다. 열의 값에 대한 최대 크기(바이트)를 찾으려면 OCTET_LENGTH 함수를 사용하십시오. 다음 예에서는 이메일 열 값의 최대 크기를 반환합니다.

```
select max(octet_length(email)) from users;

max
```

```
---
 62
```

PARTITIONED BY (col_name data_type [, …])

하나 이상의 파티션 열로 분할된 테이블을 정의하는 절입니다. 지정된 각각의 조합에 별개의 데이터 디렉터리가 사용되어 상황에 따라 쿼리 성능을 개선할 수 있습니다. 분할된 열은 테이블 데이터 자체 내에는 존재하지 않습니다. 테이블 열과 동일한 col_name에 대한 값을 사용하는 경우 오류가 발생합니다.

분할된 테이블을 생성한 후 ALTER TABLE ... ADD PARTITION 문을 사용하여 테이블을 변경하여 새 파티션을 외부 카탈로그에 등록합니다. 파티션을 추가할 때 Amazon S3에서 파티션 데이터가 들어 있는 하위 폴더의 위치를 정의합니다.

예를 들어, 테이블 `spectrum.lineitem_part`가 `PARTITIONED BY (l_shipdate date)`로 정의되어 있는 경우 다음 ALTER TABLE 명령을 실행하여 파티션을 추가합니다.

```
ALTER TABLE spectrum.lineitem_part ADD PARTITION (l_shipdate='1992-01-29')
LOCATION 's3://spectrum-public/lineitem_partition/l_shipdate=1992-01-29';
```

CREATE EXTERNAL TABLE AS를 사용하는 경우 ALTER TABLE...ADD PARTITION을 실행할 필요가 없습니다. Amazon Redshift는 외부 카탈로그에 새 파티션을 자동으로 등록합니다. 또한 Amazon Redshift는 테이블에 정의된 파티션 키를 기반으로 Amazon S3의 파티션에 해당 데이터를 자동으로 씁니다.

파티션을 보려면 SVV_EXTERNAL_PARTITIONS 시스템 뷰를 쿼리하십시오.

> **Note**
>
> CREATE EXTERNAL TABLE AS 명령의 경우 이 열은 쿼리에서 파생되므로 파티션 열의 데이터 형식을 지정할 필요가 없습니다.

ROW FORMAT DELIMITED rowformat

기본 데이터의 형식을 지정하는 절입니다. rowformat에 가능한 값은 다음과 같습니다.

- LINES TERMINATED BY 'delimiter'
- FIELDS TERMINATED BY 'delimiter'

'delimiter'에 단일 ASCII 문자를 지정합니다. 8진수 문자를 사용해 인쇄되지 않는 ASCII 문자를 '*ddd*' 형식으로 지정할 수 있습니다. 여기서 *d*는 최대 '\177'까지의 8진수(0~7)입니다. 다음은 8진수 문자를 사용하여 BEL(bell) 문자를 지정하는 예입니다.

```
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\007'
```

ROW FORMAT이 생략된 경우 기본 형식은 '\A(제목 시작)'로 종료되는 구분된 필드와 '\n(줄 바꿈)'으로 종료되는 행입니다.

ROW FORMAT SERDE 'serde_name', [WITH SERDEPROPERTIES ('property_name' = 'property_value' [, ...])]

기본 데이터의 SERDE 형식을 지정하는 절입니다.

'serde_name'

SerDe의 이름입니다. 다음 형식을 지정할 수 있습니다.

- org.apache.hadoop.hive.serde2.RegexSerDe
- com.amazonaws.glue.serde.GrokSerDe
- org.apache.hadoop.hive.serde2.OpenCSVSerde

  이 파라미터는 OpenCSVSerde에 대해 다음 SerDe 속성을 지원합니다.

  ```
  'wholeFile' = 'true'
  ```

  `wholeFile` 속성을 `true`로 설정하여 OpenCSV 요청에 대해 따옴표로 묶인 문자열 내의 줄 바꿈 문자(\in)를 적절하게 구문 분석합니다.
- org.openx.data.jsonserde.JsonSerDe
  - JSON SERDE는 또한 Ion 파일을 지원합니다.
  - JSON은 올바른 형식을 따라야 합니다.
  - Ion 및 JSON 타임스탬프에서는 ISO8601 형식을 사용해야 합니다.
  - 이 파라미터는 JsonSerDe에 대해 다음 SerDe 속성을 지원합니다.

    ```
    'strip.outer.array'='true'
    ```

    마치 배열 내에 여러 JSON 레코드가 포함되어 있는 것처럼 대괄호([…])로 둘러싸인 매우 큰 배열 하나가 포함된 Ion/JSON 파일을 처리합니다.

- com.amazon.ionhiveserde.IonHiveSerDe

  Amazon ION 형식은 데이터 유형 외에도 텍스트 및 이진 형식을 제공합니다. ION 형식의 데이터를 참조하는 외부 테이블의 경우 외부 테이블의 각 열을 ION 형식 데이터의 해당 요소에 매핑합니다. 자세한 내용은 Amazon Ion을 참조하세요. 또한 입력 및 출력 형식을 지정해야 합니다.

WITH SERDEPROPERTIES ('property_name' = 'property_value' [, ...])]

경우에 따라 쉼표로 구분된 속성 값 및 이름을 지정합니다.

ROW FORMAT이 생략된 경우 기본 형식은 '\A(제목 시작)'로 종료되는 구분된 필드와 '\n(줄 바꿈)'으로 종료되는 행입니다.

STORED AS file_format

데이터 파일의 파일 형식입니다.

유효한 형식은 다음과 같습니다.

- PARQUET
- RCFILE(LazyBinaryColumnarSerDe가 아니라 ColumnarSerDe만 사용하는 데이터의 경우)
- SEQUENCEFILE
- TEXTFILE(JSON 파일을 포함한 텍스트 파일의 경우)
- ORC
- AVRO
- INPUTFORMAT 'input_format_classname' OUTPUTFORMAT 'output_format_classname'

CREATE EXTERNAL TABLE AS 명령은 두 가지 파일 형식, 즉 TEXTFILE 및 PARQUET를 지원합니다.

INPUTFORMAT과 OUTPUTFORMAT에 대해 다음 예에 나온 것처럼 클래스 이름을 지정합니다.

```
'org.apache.hadoop.mapred.TextInputFormat'
```

LOCATION { 's3://bucket/folder/' | 's3://bucket/manifest_file'}

데이터 파일이 포함된 Amazon S3 버킷 또는 폴더, 혹은 Amazon S3 객체 경로 목록이 포함된 매니페스트 파일까지의 경로. 버킷은 Amazon Redshift 클러스터와 동일한 AWS 리전에 있어야 합니다. 지원되는 AWS 리전 목록은 Amazon Redshift Spectrum 제한 사항 섹션을 참조하세요.

경로가 버킷 또는 폴더를 지정하는 경우(예: 's3://amzn-s3-demo-bucket/custdata/') Redshift Spectrum은 지정된 버킷 또는 폴더와 모든 하위 폴더에서 파일을 스캔합니다. Redshift Spectrum은 마침표나 밑줄로 시작되는 숨겨진 파일과 파일을 무시합니다.

경로가 매니페스트 파일을 지정하는 경우 's3://bucket/manifest_file' 인수는 단일 파일을 명시적으로 참조해야 합니다(예: 's3://amzn-s3-demo-bucket/manifest.txt'). 키 접두사를 참조할 수는 없습니다.

매니페스트란 Amazon S3에서 로드할 개별 파일의 URL과 파일 크기(단위: 바이트)를 JSON 형식으로 나열한 텍스트 파일을 말합니다. URL에는 파일의 버킷 이름과 전체 객체 경로가 포함됩니다. 매니페스트에서 지정되는 파일이 서로 다른 버킷에 있을 수 있지만 모든 버킷은 Amazon Redshift 클러스터와 동일한 AWS 리전에 속해야 합니다. 파일이 두 번 나열되면 마찬가지로 두 번 로드됩니다. 다음은 3개의 파일을 로드하는 매니페스트의 JSON 형식을 나타낸 예입니다.

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket1/custdata.1", "meta": { "content_length":
 5956875 } },
    {"url":"s3://amzn-s3-demo-bucket1/custdata.2", "meta": { "content_length":
 5997091 } },
    {"url":"s3://amzn-s3-demo-bucket2/custdata.1", "meta": { "content_length":
 5978675 } }
  ]
}
```

특정 파일의 포함을 필수로 설정할 수 있습니다. 이렇게 하려면 매니페스트의 파일 수준에서 mandatory 옵션을 포함시킵니다. 누락된 필수 파일이 있는 외부 테이블을 쿼리하면 SELECT 문이 실패합니다. 외부 테이블 정의에 포함된 모든 파일이 있는지 확인합니다. 모든 파일이 없으면 찾을 수 없는 첫 번째 필수 파일을 표시하는 오류가 나타납니다. 다음은 mandatory 옵션이 true로 설정된 매니페스트의 JSON 형식을 나타낸 예입니다.

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket1/custdata.1", "mandatory":true, "meta":
 { "content_length": 5956875 } },
    {"url":"s3://amzn-s3-demo-bucket1/custdata.2", "mandatory":false, "meta":
 { "content_length": 5997091 } },
    {"url":"s3://amzn-s3-demo-bucket2/custdata.1", "meta": { "content_length":
 5978675 } }
  ]
```

```
}
```

UNLOAD를 사용하여 만든 파일을 참조하기 위해 MANIFEST 파라미터와 함께 UNLOAD를 사용하여 만든 매니페스트를 사용할 수 있습니다. 이 매니페스트 파일은 Amazon S3에서 COPY의 매니페스트 파일과 호환되지만 사용하는 키는 다릅니다. 사용되지 않는 키는 무시됩니다.

TABLE PROPERTIES ('property_name'='property_value' [, ...])

테이블 속성의 테이블 정의를 설정하는 절입니다.

> **Note**
>
> 테이블 속성은 대/소문자를 구분합니다.

'compression_type'='value'

파일 이름에 확장자가 포함되지 않은 경우에 사용할 압축 유형을 설정하는 속성입니다. 이 속성을 설정하고 파일 확장자가 존재하는 경우 확장자를 무시하고 속성이 설정한 값을 사용합니다. 압축 유형에 유효한 값은 다음과 같습니다.

- bzip2
- gzip
- 없음
- gzip

'data_cleansing_enabled'='true / false’

이 속성은 테이블에 대해 데이터 처리가 켜져 있는지 여부를 설정합니다. 'data_cleansing_enabled'가 true로 설정되어 있으면 테이블에 대한 데이터 처리가 켜져 있는 것입니다. 'data_cleansing_enabled'가 false로 설정되어 있으면 테이블에 대한 데이터 처리가 꺼져 있는 것입니다. 이 속성으로 제어되는 테이블 수준 데이터 처리 속성 목록은 다음과 같습니다.

- column_count_mismatch_handling
- invalid_char_handling
- numeric_overflow_handling
- replacement_char
- surplus_char_handling

예시는 데이터 처리 예 섹션을 참조하세요.

'invalid_char_handling'='value'

쿼리 결과에 잘못된 UTF-8 문자 값이 포함된 경우 수행할 작업을 지정합니다. 다음 작업을 지정할 수 있습니다.

DISABLED

잘못된 문자 처리를 수행하지 않습니다.

FAIL

잘못된 UTF-8 값이 포함된 데이터를 반환하는 쿼리를 취소합니다.

SET_TO_NULL

잘못된 UTF-8 값을 null로 바꿉니다.

DROP_ROW

행의 각 값을 null로 바꿉니다.

REPLACE

잘못된 문자를 `replacement_char`을 사용하여 지정한 대체 문자열로 바꿉니다.

'replacement_char'='character’

`invalid_char_handling`을 `REPLACE`로 설정할 경우 사용할 대체 문자를 지정합니다.

'numeric_overflow_handling'='value’

ORC 데이터에 열 정의(예: SMALLINT 또는 int16)보다 큰 정수(예: BIGINT 또는 int64)가 포함된 경우 수행할 작업을 지정합니다. 다음 작업을 지정할 수 있습니다.

DISABLED

잘못된 문자 처리가 꺼져 있습니다.

FAIL

데이터에 잘못된 문자가 포함된 경우 쿼리를 취소합니다.

SET_TO_NULL

잘못된 문자를 null로 설정합니다.

DROP_ROW

행의 각 값을 null로 설정합니다.

'surplus_bytes_handling'='value'

VARBYTE 데이터를 포함하는 열에 대해 정의된 데이터 형식의 길이를 초과하는 데이터 로드를 처리하는 방법을 지정합니다. 기본적으로 Redshift Spectrum에서는 열의 너비를 초과하는 데이터에 대해 값을 null로 설정합니다.

쿼리가 데이터 형식의 길이를 초과하는 데이터를 반환하는 경우 수행할 다음 작업을 지정할 수 있습니다.

SET_TO_NULL

열 너비를 초과하는 데이터를 null로 바꿉니다.

DISABLED

잉여 바이트 처리를 수행하지 않습니다.

FAIL

열 너비를 초과하는 데이터를 반환하는 쿼리를 취소합니다.

DROP_ROW

열 너비를 초과하는 데이터가 포함된 모든 행을 삭제합니다.

TRUNCATE

열에 정의된 최대 문자 수를 초과하는 문자를 제거합니다.

'surplus_char_handling'='value'

VARCHAR, CHAR 또는 문자열 데이터를 포함하는 열에 대해 정의된 데이터 형식의 길이를 초과하는 데이터 로드를 처리하는 방법을 지정합니다. 기본적으로 Redshift Spectrum에서는 열의 너비를 초과하는 데이터에 대해 값을 null로 설정합니다.

쿼리가 열 너비를 초과하는 데이터를 반환하는 경우 수행할 다음 작업을 지정할 수 있습니다.

SET_TO_NULL

열 너비를 초과하는 데이터를 null로 바꿉니다.

DISABLED

잉여 문자 처리를 수행하지 않습니다.

FAIL

열 너비를 초과하는 데이터를 반환하는 쿼리를 취소합니다.

DROP_ROW

행의 각 값을 null로 바꿉니다.

TRUNCATE

열에 정의된 최대 문자 수를 초과하는 문자를 제거합니다.

'column_count_mismatch_handling'='value’

파일에 포함된 행 값이 외부 테이블 정의에 지정된 열 수보다 적거나 많은지 식별합니다. 이 속성은 압축되지 않은 텍스트 파일 형식에만 사용할 수 있습니다. 다음 작업을 지정할 수 있습니다.

DISABLED

열 개수 불일치 처리가 꺼져 있습니다.

FAIL

열 개수 불일치가 감지되면 쿼리에 실패합니다.

SET_TO_NULL

누락된 값을 NULL로 채우고 각 행의 추가 값을 무시합니다.

DROP_ROW

열 수 불일치 오류가 포함된 모든 행을 스캔에서 삭제합니다.

'numRows'='row_count'

테이블 정의를 위해 numRows 값을 설정하는 속성입니다. 외부 테이블의 통계를 명시적으로 업데이트하려면 테이블의 크기를 나타내도록 numRows 속성을 설정합니다. Amazon Redshift는 쿼리 옵티마이저가 쿼리 계획을 생성하는 데 사용하는 테이블 통계를 생성하기 위해 외부 테이블을 분석하지 않습니다. 따라서 외부 테이블에 대한 테이블 통계가 준비되어 있지 않으면 Amazon Redshift가 외부 테이블이 더 큰 테이블이고 로컬 테이블이 더 작은 테이블이라는 가정에 기초하여 쿼리 실행 계획을 생성합니다.

'skip.header.line.count'='line_count'

각 원본 파일의 시작 부분에서 건너 뛸 행 개수를 설정하는 속성입니다.

'serialization.null.format'=' '

필드에 제공된 텍스트와 정확히 일치하는 항목이 있는 경우 Spectrum을 지정하는 속성은 NULL 값을 반환해야 합니다.

'orc.schema.resolution'='mapping_type'

ORC 데이터 형식을 사용하는 테이블의 열 매핑 유형을 설정하는 속성입니다. 모든 데이터 형식의 경우 이 속성은 무시됩니다.

열 매핑 유형에 유효한 값은 다음과 같습니다.

- name
- position

orc.schema.resolution 속성을 생략하면 열이 기본적으로 이름별로 매핑됩니다. orc.schema.resolution을 'name' 또는 'position' 이외 값으로 설정하면 열은 위치별로 매핑됩니다. 열 매핑에 대한 자세한 내용은 외부 테이블 열을 ORC 열에 매핑 섹션을 참조하세요.

> **Note**
>
> COPY 명령은 반드시 위치를 기준으로 ORC 데이터 파일에 매핑됩니다. orc.schema.resolution 테이블 속성은 COPY 명령 동작에 영향을 주지 않습니다.

'write.parallel'='on / off'

CREATE EXTERNAL TABLE AS가 병렬 방식으로 데이터를 작성해야 하는지 여부를 설정하는 속성입니다. 기본적으로 CREATE EXTERNAL TABLE AS는 클러스터의 조각 수에 따라 다수의 파일에 병렬 방식으로 데이터를 작성합니다. 기본 옵션은 on입니다. 'write.parallel'이 꺼짐으로 설정되면 CREATE EXTERNAL TABLE AS는 Amazon S3의 하나 이상의 데이터 파일에 순차적으로 작성합니다. 이 테이블 속성은 동일한 외부 테이블의 모든 후속 INSERT 문에도 적용됩니다.

'write.maxfilesize.mb'='size'

CREATE EXTERNAL TABLE AS에 의해 Amazon S3에 작성된 각 파일의 최대 크기(MB)를 설정하는 속성입니다. 크기는 5에서 6200 사이의 유효한 정수여야 합니다. 기본 최대 파일 크기는 6,200MB입니다. 이 테이블 속성은 동일한 외부 테이블의 모든 후속 INSERT 문에도 적용됩니다.

'write.kms.key.id'='value'

AWS Key Management Service 키를 지정하여 Amazon S3 객체에 대해 서버 측 암호화(SSE)를 사용할 수 있습니다. 여기서 값은 다음 중 하나입니다.

- `auto` - Amazon S3 버킷에 저장된 기본 AWS KMS 키를 사용합니다.
- 데이터를 암호화하기 위해 지정하는 kms-key입니다.

select_statement

쿼리를 정의하여 외부 테이블에 하나 이상의 행을 삽입하는 문입니다. 쿼리가 생성하는 모든 행은 테이블 정의에 따라 텍스트 또는 Parquet 형식으로 Amazon S3에 작성됩니다.

예시

예제 모음은 예시에서 확인할 수 있습니다.

사용 노트

이 주제에는 CREATE EXTERNAL TABLE에 대한 사용 참고 사항이 포함되어 있습니다. PG_TABLE_DEF, STV_TBL_PERM, PG_CLASS 또는 information_schema와 같은 표준 Amazon Redshift 테이블에 사용하는 것과 동일한 리소스를 사용하여 Amazon Redshift Spectrum 테이블에 대한 세부 정보를 볼 수 없습니다. 비즈니스 인텔리전스 또는 분석 도구가 Redshift Spectrum 외부 테이블을 인식하지 못하는 경우 애플리케이션이 SVV_EXTERNAL_TABLES 및 SVV_EXTERNAL_COLUMNS를 쿼리하도록 구성합니다.

CREATE EXTERNAL TABLE AS

경우에 따라 AWS Glue Data Catalog, AWS Lake Formation 외부 카탈로그 또는 Apache Hive 메타스토어에 대해 CREATE EXTERNAL TABLE AS 명령을 실행할 수 있습니다. 이러한 경우 AWS Identity and Access Management(IAM) 역할을 사용하여 외부 스키마를 생성합니다. 이 IAM 역할에는 Amazon S3에 대한 읽기 및 쓰기 권한이 모두 있어야 합니다.

Lake Formation 카탈로그를 사용하는 경우 IAM 역할에 카탈로그에서 테이블을 생성할 수 있는 권한이 있어야 합니다. 이 경우 대상 Amazon S3 경로에 대한 데이터 레이크 위치 권한도 있어야 합니다. 이 IAM 역할은 새 AWS Lake Formation 테이블의 소유자가 됩니다.

파일 이름이 고유한지 확인하기 위해 Amazon Redshift에서는 기본적으로 Amazon S3에 업로드된 각 파일의 이름에 다음 형식을 사용합니다.

*<date>*_*<time>*_*<microseconds>*_*<query_id>*_*<slice-number>*_part_*<part-number>*.*<format>*.

예를 들면, `20200303_004509_810669_1007_0001_part_00.parquet`입니다.

CREATE EXTERNAL TABLE AS 명령을 실행할 때 다음 사항을 고려하십시오.

- Amazon S3 위치는 비어 있어야 합니다.
- Amazon Redshift는 STORED AS 절을 사용할 때만 PARQUET 및 TEXTFILE 형식을 지원합니다.

- 열 정의 목록을 정의할 필요가 없습니다. 새 외부 테이블의 열 이름과 열 데이터 형식은 SELECT 쿼리에서 직접 파생됩니다.
- PARTITIONED BY 절에서 파티션 열의 데이터 형식을 정의할 필요가 없습니다. 파티션 키를 지정하는 경우 SELECT 쿼리 결과에 이 열의 이름이 있어야 합니다. 여러 파티션 열이 있는 경우 SELECT 쿼리의 순서는 중요하지 않습니다. Amazon Redshift는 PARTITIONED BY 절에 정의된 순서를 사용하여 외부 테이블을 생성합니다.
- Amazon Redshift는 파티션 키 값에 따라 출력 파일을 파티션 폴더로 자동으로 분할합니다. 기본적으로 Amazon Redshift는 출력 파일에서 파티션 열을 제거합니다.
- LINES TERMINATED BY 'delimiter' 절은 지원되지 않습니다.
- ROW FORMAT SERDE 'serde_name' 절은 지원되지 않습니다.
- 매니페스트 파일의 사용은 지원되지 않습니다. 따라서 Amazon S3에서 매니페스트 파일에 LOCATION 절을 정의할 수 없습니다.
- Amazon Redshift는 명령 끝에서 'numRows' 테이블 속성을 자동으로 업데이트합니다.
- 'compression_type' 테이블 속성은 PARQUET 파일 형식에 대해 'none' 또는 'snappy'만 사용할 수 있습니다.
- Amazon Redshift는 외부 SELECT 쿼리에서 LIMIT 절을 허용하지 않습니다. 대신 중첩된 LIMIT 절을 사용할 수 있습니다.
- STL_UNLOAD_LOG를 사용하여 각 CREATE EXTERNAL TABLE AS 작업에 의해 Amazon S3에 작성된 파일을 추적할 수 있습니다.

외부 테이블 생성 및 쿼리 권한

외부 테이블을 생성하려면 외부 스키마의 소유자 또는 수퍼유저인지 확인하십시오. 외부 스키마의 소유권을 이전하려면 ALTER SCHEMA를 사용합니다. 다음 예에서는`spectrum_schema` 스키마의 소유자를 `newowner`로 바꿉니다.

```
alter schema spectrum_schema owner to newowner;
```

Redshift Spectrum 쿼리를 실행하는 데 필요한 권한은 다음과 같습니다.

- 스키마에 대한 사용 권한
- 현재 데이터베이스에서 임시 테이블을 생성할 수 있는 권한

다음 예에서는 `spectrum_schema` 스키마에 대한 사용 권한을 `spectrumusers` 사용자 그룹에 부여합니다.

```
grant usage on schema spectrum_schema to group spectrumusers;
```

다음 예에서는 spectrumdb 데이터베이스에 대한 임시 권한을 spectrumusers 사용자 그룹에 부여합니다.

```
grant temp on database spectrumdb to group spectrumusers;
```

가상 열

기본적으로 Amazon Redshift에서는 가상 열 $path 및 $size가 있는 외부 테이블을 생성합니다. 이러한 열을 선택하면 Amazon S3에 있는 데이터 파일의 경로와 쿼리에서 반환한 각 열에 대한 데이터 파일의 크기를 확인할 수 있습니다. $path 및 $size 열 이름은 큰 따옴표로 구분해야 합니다. SELECT * 절은 가상 열을 반환하지 않습니다. 다음 예에서처럼 쿼리에 $path 및 $size 열을 명시적으로 포함해야 합니다.

```
select "$path", "$size"
from spectrum.sales_part
where saledate = '2008-12-01';
```

spectrum_enable_pseudo_columns 구성 파라미터를 false로 설정하여 세션에 대해 가상 열 생성을 비활성화할 수 있습니다.

> **Important**
>
> $size 또는 $path를 선택하면 Redshift Spectrum이 Amazon S3의 데이터 파일을 스캔해 결과 집합의 크기를 결정하기 때문에 요금이 발생합니다. 자세한 내용은 Amazon Redshift Pricing(Amazon Redshift 요금) 섹션을 참조하세요.

데이터 처리 옵션 설정

테이블 파라미터를 설정하여 외부 테이블에서 쿼리되는 다음을 비롯한 데이터의 입력 처리를 지정할 수 있습니다.

- VARCHAR, CHAR 및 문자열 데이터를 포함하는 열의 잉여 문자. 자세한 내용은 외부 테이블 속성 surplus_char_handling 섹션을 참조하세요.
- VARCHAR, CHAR 및 문자열 데이터를 포함하는 열의 잘못된 문자. 자세한 내용은 외부 테이블 속성 invalid_char_handling 섹션을 참조하세요.

- 외부 테이블 속성 invalid_char_handling에 REPLACE를 지정하는 경우 사용할 대체 문자.
- 정수 및 십진수 데이터가 포함된 열에서 캐스트 오버플로우 처리 자세한 내용은 외부 테이블 속성 numeric_overflow_handling 섹션을 참조하세요.
- Surplus_bytes_handling은 varbyte 데이터를 포함하는 열의 잉여 바이트에 대한 입력 처리를 지정합니다. 자세한 내용은 외부 테이블 속성 surplus_bytes_handling 섹션을 참조하세요.

예시

다음 예에서는 spectrum이라는 Amazon Redshift 외부 스키마에서 SALES로 명명된 테이블을 생성합니다. 데이터는 탭으로 구분된 텍스트 파일입니다. TABLE PROPERTIES 절은 numRows 속성을 170,000개 행으로 설정합니다.

CREATE EXTERNAL TABLE을 실행할 때 사용하는 자격 증명에 따라 구성해야 하는 IAM 권한이 있을 수 있습니다. 가장 좋은 방법은 권한 정책을 IAM 역할에 연결한 다음 필요에 따라 사용자 및 그룹에 할당하는 것입니다. 자세한 내용은 Amazon Redshift의 Identity and Access Management를 참조하세요.

```
create external table spectrum.sales(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
saledate date,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
row format delimited
fields terminated by '\t'
stored as textfile
location 's3://redshift-downloads/tickit/spectrum/sales/'
table properties ('numRows'='170000');
```

다음 예에서는 JsonSerDe를 사용하여 JSON 형식의 데이터를 참조하는 테이블을 생성합니다.

```
create external table spectrum.cloudtrail_json (
event_version int,
event_id bigint,
event_time timestamp,
event_type varchar(10),
```

```
awsregion varchar(20),
event_name varchar(max),
event_source varchar(max),
requesttime timestamp,
useragent varchar(max),
recipientaccountid bigint)
row format serde 'org.openx.data.jsonserde.JsonSerDe'
with serdeproperties (
'dots.in.keys' = 'true',
'mapping.requesttime' = 'requesttimestamp'
) location 's3://amzn-s3-demo-bucket/json/cloudtrail';
```

다음 CREATE EXTERNAL TABLE AS 예는 분할되지 않은 외부 테이블을 생성합니다. 그런 다음 대상 Amazon S3 위치에 Apache Parquet로 SELECT 쿼리의 결과를 작성합니다.

```
CREATE EXTERNAL TABLE spectrum.lineitem
STORED AS parquet
LOCATION 'S3://amzn-s3-demo-bucket/cetas/lineitem/'
AS SELECT * FROM local_lineitem;
```

다음 예에서는 분할된 외부 테이블을 생성하고 SELECT 쿼리에 파티션 열을 포함합니다.

```
CREATE EXTERNAL TABLE spectrum.partitioned_lineitem
PARTITIONED BY (l_shipdate, l_shipmode)
STORED AS parquet
LOCATION 'S3://amzn-s3-demo-bucket/cetas/partitioned_lineitem/'
AS SELECT l_orderkey, l_shipmode, l_shipdate, l_partkey FROM local_table;
```

외부 데이터 카탈로그에 있는 기존 데이터베이스의 목록을 보려면 SVV_EXTERNAL_DATABASES 시스템 뷰를 쿼리하십시오.

```
select eskind,databasename,esoptions from svv_external_databases order by databasename;
```

```
eskind | databasename | esoptions
-------+--------------
+-------------------------------------------------------------------------------
     1 | default      | {"REGION":"us-
west-2","IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
     1 | sampledb     | {"REGION":"us-
west-2","IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
```

```
     1 | spectrumdb   | {"REGION":"us-
west-2","IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
```

외부 테이블의 세부 정보를 보려면 SVV_EXTERNAL_TABLES 및 SVV_EXTERNAL_COLUMNS 시스템 뷰를 쿼리하십시오.

다음 예에서는 SVV_EXTERNAL_TABLES 뷰를 쿼리합니다.

```
select schemaname, tablename, location from svv_external_tables;
```

```
schemaname | tablename            | location
-----------+----------------------
+-----------------------------------------------------
spectrum   | sales                | s3://redshift-downloads/tickit/spectrum/sales
spectrum   | sales_part           | s3://redshift-downloads/tickit/spectrum/
sales_partition
```

다음 예에서는 SVV_EXTERNAL_COLUMNS 뷰를 쿼리합니다.

```
select * from svv_external_columns where schemaname like 'spectrum%' and tablename
 ='sales';
```

```
schemaname | tablename | columnname | external_type | columnnum | part_key
-----------+-----------+------------+---------------+-----------+---------
spectrum   | sales     | salesid    | int           |         1 |        0
spectrum   | sales     | listid     | int           |         2 |        0
spectrum   | sales     | sellerid   | int           |         3 |        0
spectrum   | sales     | buyerid    | int           |         4 |        0
spectrum   | sales     | eventid    | int           |         5 |        0
spectrum   | sales     | saledate   | date          |         6 |        0
spectrum   | sales     | qtysold    | smallint      |         7 |        0
spectrum   | sales     | pricepaid  | decimal(8,2)  |         8 |        0
spectrum   | sales     | commission | decimal(8,2)  |         9 |        0
spectrum   | sales     | saletime   | timestamp     |        10 |        0
```

테이블 파티션을 보려면 다음 쿼리를 사용하십시오.

```
select schemaname, tablename, values, location
from svv_external_partitions
where tablename = 'sales_part';
```

```
schemaname | tablename  | values         | location
-----------+------------+----------------
+-------------------------------------------------------------------------
spectrum   | sales_part | ["2008-01-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-01
spectrum   | sales_part | ["2008-02-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-02
spectrum   | sales_part | ["2008-03-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-03
spectrum   | sales_part | ["2008-04-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-04
spectrum   | sales_part | ["2008-05-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-05
spectrum   | sales_part | ["2008-06-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-06
spectrum   | sales_part | ["2008-07-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-07
spectrum   | sales_part | ["2008-08-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-08
spectrum   | sales_part | ["2008-09-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-09
spectrum   | sales_part | ["2008-10-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-10
spectrum   | sales_part | ["2008-11-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-11
spectrum   | sales_part | ["2008-12-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-12
```

다음은 외부 테이블의 관련 데이터 파일의 총 크기를 반환하는 예입니다.

```
select distinct "$path", "$size"
   from spectrum.sales_part;

 $path                                                                      | $size
----------------------------------------------------------------------------+-------
s3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-01/ |  1616
s3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-02/ |  1444
s3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-02/ |  1444
```

파티셔닝 예

날짜 기준으로 파티셔닝된 외부 테이블을 만들려면 다음 명령을 실행합니다.

```
create external table spectrum.sales_part(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
partitioned by (saledate date)
row format delimited
fields terminated by '|'
stored as textfile
location 's3://redshift-downloads/tickit/spectrum/sales_partition/'
table properties ('numRows'='170000');
```

파티션을 추가하려면 다음 ALTER TABLE 명령을 실행합니다.

```
alter table spectrum.sales_part
add if not exists partition (saledate='2008-01-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-01/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-02-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-02/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-03-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-03/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-04-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-04/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-05-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-05/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-06-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-06/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-07-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-07/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-08-01')
```

```
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-08/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-09-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-09/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-10-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-10/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-11-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-11/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-12-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-12/';
```

파티셔닝된 테이블에서 데이터를 선택하려면 다음 쿼리를 실행합니다.

```
select top 10 spectrum.sales_part.eventid, sum(spectrum.sales_part.pricepaid)
from spectrum.sales_part, event
where spectrum.sales_part.eventid = event.eventid
  and spectrum.sales_part.pricepaid > 30
  and saledate = '2008-12-01'
group by spectrum.sales_part.eventid
order by 2 desc;
```

```
eventid | sum
--------+---------
    914 | 36173.00
   5478 | 27303.00
   5061 | 26383.00
   4406 | 26252.00
   5324 | 24015.00
   1829 | 23911.00
   3601 | 23616.00
   3665 | 23214.00
   6069 | 22869.00
   5638 | 22551.00
```

외부 테이블 파티션을 보려면 SVV_EXTERNAL_PARTITIONS 시스템 뷰를 쿼리하십시오.

```
select schemaname, tablename, values, location from svv_external_partitions
where tablename = 'sales_part';
```

```
schemaname | tablename  | values         | location
-----------+------------+----------------
+-----------------------------------------------
spectrum   | sales_part | ["2008-01-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-01
spectrum   | sales_part | ["2008-02-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-02
spectrum   | sales_part | ["2008-03-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-03
spectrum   | sales_part | ["2008-04-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-04
spectrum   | sales_part | ["2008-05-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-05
spectrum   | sales_part | ["2008-06-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-06
spectrum   | sales_part | ["2008-07-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-07
spectrum   | sales_part | ["2008-08-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-08
spectrum   | sales_part | ["2008-09-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-09
spectrum   | sales_part | ["2008-10-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-10
spectrum   | sales_part | ["2008-11-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-11
spectrum   | sales_part | ["2008-12-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-12
```

행 형식 예

다음은 AVRO 형식으로 저장된 데이터 파일에 대해 ROW FORMAT SERDE 파라미터를 지정하는 예입니다.

```
create external table spectrum.sales(salesid int, listid int, sellerid int,
 buyerid int, eventid int, dateid int, qtysold int, pricepaid decimal(8,2), comment
 VARCHAR(255))
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
WITH SERDEPROPERTIES ('avro.schema.literal'='{\"namespace\": \"dory.sample\",\"name\":
 \"dory_avro\",\"type\": \"record\", \"fields\": [{\"name\":\"salesid\", \"type\":\"int
\"},
{\"name\":\"listid\", \"type\":\"int\"},
{\"name\":\"sellerid\", \"type\":\"int\"},
{\"name\":\"buyerid\", \"type\":\"int\"},
```

```
{\"name\":\"eventid\",\"type\":\"int\"},
{\"name\":\"dateid\",\"type\":\"int\"},
{\"name\":\"qtysold\",\"type\":\"int\"},
{\"name\":\"pricepaid\", \"type\": {\"type\": \"bytes\", \"logicalType\": \"decimal\",
 \"precision\": 8, \"scale\": 2}}, {\"name\":\"comment\",\"type\":\"string\"}]}')
STORED AS AVRO
location 's3://amzn-s3-demo-bucket/avro/sales' ;
```

다음은 RegEx를 사용하여 ROW FORMAT SERDE 파라미터를 지정하는 예입니다.

```
create external table spectrum.types(
cbigint bigint,
cbigint_null bigint,
cint int,
cint_null int)
row format serde 'org.apache.hadoop.hive.serde2.RegexSerDe'
with serdeproperties ('input.regex'='([^\\x01]+)\\x01([^\\x01]+)\\x01([^\\x01]+)\
\x01([^\\x01]+)')
stored as textfile
location 's3://amzn-s3-demo-bucket/regex/types';
```

다음은 Grok를 사용하여 ROW FORMAT SERDE 파라미터를 지정하는 예입니다.

```
create external table spectrum.grok_log(
timestamp varchar(255),
pid varchar(255),
loglevel varchar(255),
progname varchar(255),
message varchar(255))
row format serde 'com.amazonaws.glue.serde.GrokSerDe'
with serdeproperties ('input.format'='[DFEWI], \\[%{TIMESTAMP_ISO8601:timestamp} #
%{POSINT:pid:int}\\] *(?<loglevel>:DEBUG|FATAL|ERROR|WARN|INFO) -- +%{DATA:progname}:
 %{GREEDYDATA:message}')
stored as textfile
location 's3://DOC-EXAMPLE-BUCKET/grok/logs';
```

다음은 S3 버킷에서 Amazon S3 서버 액세스 로그를 정의하는 예입니다. Redshift Spectrum을 사용하여 Amazon S3 액세스 로그를 쿼리할 수 있습니다.

```
CREATE EXTERNAL TABLE spectrum.mybucket_s3_logs(
bucketowner varchar(255),
```

```
bucket varchar(255),
requestdatetime varchar(2000),
remoteip varchar(255),
requester varchar(255),
requested varchar(255),
operation varchar(255),
key varchar(255),
requesturi_operation varchar(255),
requesturi_key varchar(255),
requesturi_httpprotoversion varchar(255),
httpstatus varchar(255),
errorcode varchar(255),
bytessent bigint,
objectsize bigint,
totaltime varchar(255),
turnaroundtime varchar(255),
referrer varchar(255),
useragent varchar(255),
versionid varchar(255)
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
'input.regex' = '([^ ]*) ([^ ]*) \\[(.*?)\\] ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*)
 \"([^ ]*)\\s*([^ ]*)\\s*([^ ]*)\" (- |[^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*)
 ([^ ]*) (\"[^\"]*\") ([^ ]*).*$')
LOCATION 's3://amzn-s3-demo-bucket/s3logs’;
```

다음은 ION 형식 데이터에 대해 ROW FORMAT SERDE 파라미터를 지정하는 예입니다.

```
CREATE EXTERNAL TABLE tbl_name (columns)
ROW FORMAT SERDE 'com.amazon.ionhiveserde.IonHiveSerDe'
STORED AS
INPUTFORMAT 'com.amazon.ionhiveserde.formats.IonInputFormat'
OUTPUTFORMAT 'com.amazon.ionhiveserde.formats.IonOutputFormat'
LOCATION 's3://amzn-s3-demo-bucket/prefix'
```

데이터 처리 예

다음 예제에서는 spi_global_rankings.csv 파일에 액세스합니다. 이 예에서 보여주는 것처럼 `spi_global_rankings.csv` 파일을 Amazon S3 버킷에 업로드할 수 있습니다.

다음 예에서는 외부 스키마 schema_spectrum_uddh 및 spectrum_db_uddh 데이터베이스를 생성합니다. aws-account-id에 AWS 계정 ID, role-name에 Redshift Spectrum 역할 이름을 입력합니다.

```
create external schema schema_spectrum_uddh
from data catalog
database 'spectrum_db_uddh'
iam_role 'arn:aws:iam::aws-account-id:role/role-name'
create external database if not exists;
```

다음 예에서는 외부 스키마 schema_spectrum_uddh에서 외부 테이블 soccer_league를 생성합니다.

```
CREATE EXTERNAL TABLE schema_spectrum_uddh.soccer_league
(
  league_rank smallint,
  prev_rank   smallint,
  club_name   varchar(15),
  league_name varchar(20),
  league_off  decimal(6,2),
  league_def  decimal(6,2),
  league_spi  decimal(6,2),
  league_nspi integer
)
ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n\l'
stored as textfile
LOCATION 's3://spectrum-uddh/league/'
table properties ('skip.header.line.count'='1');
```

soccer_league 테이블에 포함된 행의 수를 확인합니다.

```
select count(*) from schema_spectrum_uddh.soccer_league;
```

행 수가 표시됩니다.

```
count
645
```

다음 쿼리는 상위 10개 클럽을 표시합니다. 클럽 Barcelona의 문자열에 잘못된 문자가 있으므로 이름에 NULL이 표시됩니다.

```
select league_rank,club_name,league_name,league_nspi
from schema_spectrum_uddh.soccer_league
where league_rank between 1 and 10;
```

```
league_rank club_name league_name   league_nspi
1  Manchester City Barclays Premier Lea  34595
2  Bayern Munich German Bundesliga  34151
3  Liverpool Barclays Premier Lea  33223
4  Chelsea  Barclays Premier Lea  32808
5  Ajax  Dutch Eredivisie  32790
6  Atletico  Madrid Spanish Primera Divi 31517
7  Real Madrid Spanish Primera Divi  31469
8  NULL         Spanish Primera Divi            31321
9  RB Leipzig German Bundesliga  31014
10  Paris Saint-Ger French Ligue 1   30929
```

다음 예제에서는 invalid_char_handling, replacement_char, data_cleansing_enabled 외부 테이블 속성을 지정하여 예기치 않은 문자의 대체 문자로 물음표(?)를 삽입하도록 soccer_league 테이블을 변경합니다.

```
alter  table schema_spectrum_uddh.soccer_league
set table properties
 ('invalid_char_handling'='REPLACE','replacement_char'='?','data_cleansing_enabled'='true');
```

다음 예제에서는 soccer_league 테이블에서 순위가 1~10위인 팀을 쿼리합니다.

```
select league_rank,club_name,league_name,league_nspi
from schema_spectrum_uddh.soccer_league
where league_rank between 1 and 10;
```

테이블 속성이 변경되어 결과에 상위 10개 클럽과 Barcelona 클럽의 8번째 행을 대체하는 물음표(?) 문자가 표시됩니다.

```
league_rank club_name league_name  league_nspi
1  Manchester City Barclays Premier Lea 34595
2  Bayern Munich German Bundesliga 34151
3  Liverpool Barclays Premier Lea 33223
```

```
4  Chelsea  Barclays Premier Lea 32808
5  Ajax  Dutch Eredivisie 32790
6  Atletico Madrid Spanish Primera Divi 31517
7  Real Madrid Spanish Primera Divi 31469
8  Barcel?na Spanish Primera Divi 31321
9  RB Leipzig German Bundesliga 31014
10  Paris Saint-Ger French Ligue 1  30929
```

다음 예제에서는 invalid_char_handling 외부 테이블 속성을 지정하여 예기치 않은 문자가 있는 행을 삭제하도록 soccer_league 테이블을 변경합니다.

```
alter table schema_spectrum_uddh.soccer_league
set table properties
 ('invalid_char_handling'='DROP_ROW','data_cleansing_enabled'='true');
```

다음 예제에서는 soccer_league 테이블에서 순위가 1~10위인 팀을 쿼리합니다.

```
select league_rank,club_name,league_name,league_nspi
from schema_spectrum_uddh.soccer_league
where league_rank between 1 and 10;
```

결과에는 상위 클럽이 Barcelona 클럽의 8번째 행은 포함하지 않고 표시됩니다.

```
league_rank   club_name         league_name            league_nspi
1             Manchester City   Barclays Premier Lea   34595
2             Bayern Munich     German Bundesliga      34151
3             Liverpool         Barclays Premier Lea   33223
4             Chelsea           Barclays Premier Lea   32808
5             Ajax              Dutch Eredivisie       32790
6             Atletico Madrid   Spanish Primera Divi   31517
7             Real Madrid       Spanish Primera Divi   31469
9             RB Leipzig        German Bundesliga      31014
10            Paris Saint-Ger   French Ligue 1         30929
```

CREATE EXTERNAL VIEW

데이터 카탈로그 미리 보기 기능은 다음 리전에서만 사용 가능합니다.

- 미국 동부(오하이오)(us-east-2)
- 미국 동부(버지니아 북부)(us-east-1)

- 미국 서부(캘리포니아 북부)(us-west-1)
- 아시아 태평양(도쿄)(ap-northeast-1)
- 유럽(아일랜드)(eu-west-1)
- 유럽(스톡홀름)(eu-north-1)

데이터 카탈로그에서 뷰를 생성합니다. 데이터 카탈로그 뷰는 Amazon Athena와 Amazon EMR 등의 다른 SQL 엔진에서 작동하는 단일 뷰 스키마입니다. 원하는 엔진에서 뷰를 쿼리할 수 있습니다. 데이터 카탈로그 뷰에 대한 자세한 내용은 데이터 카탈로그 뷰 생성을 참조하세요.

구문

```
CREATE EXTERNAL VIEW schema_name.view_name [ IF NOT EXISTS ]
{catalog_name.schema_name.view_name | awsdatacatalog.dbname.view_name |
 external_schema_name.view_name}
AS query_definition;
```

파라미터

schema_name.view_name

AWS Glue 데이터베이스에 연결된 스키마이며, 뷰 이름이 뒤따릅니다.

PROTECTED

query_definition 내의 쿼리를 성공적으로 완료할 수 있는 경우에만 CREATE EXTERNAL VIEW 명령을 완료하도록 지정합니다.

IF NOT EXISTS

뷰가 아직 존재하지 않는 경우 뷰를 생성합니다.

catalog_name.schema_name.view_name | awsdatacatalog.dbname.view_name | external_schema_name.view_name

뷰를 생성할 때 사용하는 스키마의 표기법입니다. 직접 만든 Glue 데이터베이스인 AWS Glue Data Catalog 또는 직접 만든 외부 스키마를 사용하도록 지정할 수 있습니다. 자세한 내용은 CREATE DATABASE 및 CREATE EXTERNAL SCHEMA를 참조하세요.

query_definition

뷰를 변경하기 위해 Amazon Redshift가 실행하는 SQL 쿼리의 정의입니다.

예시

다음 예시에서는 sample_schema.glue_data_catalog_view라는 데이터 카탈로그 뷰를 생성합니다.

```
CREATE EXTERNAL PROTECTED VIEW sample_schema.glue_data_catalog_view IF NOT EXISTS
AS SELECT * FROM sample_database.remote_table "remote-table-name";
```

CREATE FUNCTION

SQL SELECT 절 또는 Python 프로그램을 이용해 스칼라 사용자 정의 함수(UDF)를 새로 만듭니다.

자세한 정보와 지침은 Amazon Redshift의 사용자 정의 함수 섹션을 참조하세요.

필수 권한

CREATE OR REPLACE FUNCTION을 실행하려면 다음 방법 중 하나에 대한 권한이 있어야 합니다.

- CREATE FUNCTION의 경우
  - 슈퍼 사용자는 신뢰할 수 있는 언어와 신뢰할 수 없는 언어를 사용하여 함수를 만들 수 있습니다.
  - CREATE [OR REPLACE] FUNCTION 권한이 있는 사용자는 신뢰할 수 있는 언어로 함수를 만들 수 있습니다.
- REPLACE FUNCTION의 경우
  - 수퍼유저
  - CREATE [OR REPLACE] FUNCTION 권한이 있는 사용자
  - 함수 소유자

구문

```
CREATE [ OR REPLACE ] FUNCTION f_function_name
( { [py_arg_name  py_arg_data_type |
sql_arg_data_type } [ , ... ] ] )
RETURNS data_type
{ VOLATILE | STABLE | IMMUTABLE }
AS $$
  { python_program | SELECT_clause }
$$ LANGUAGE { plpythonu | sql }
```

파라미터

OR REPLACE

이름 및 입력 인수 데이터 형식이 같은 함수 또는 서명이 같은 함수인 경우 이런 함수가 이미 하나 존재하므로 기존 함수가 대체됨을 지정합니다. 함수는 똑같은 데이터 형식 집합을 정의하는 새 함수로만 바꿀 수 있습니다. 수퍼유저만이 함수를 바꿀 수 있습니다.

기존 함수와 이름은 같지만 서명이 다른 함수를 정의하면 새 함수가 생성됩니다. 즉, 함수 이름이 오버로드됩니다. 자세한 내용은 함수 이름 오버로드 단원을 참조하십시오.

f_function_name

함수의 이름입니다. 스키마 이름을 지정하는 경우(예: `myschema.myfunction`), 함수는 지정된 스키마를 사용하여 생성됩니다. 그렇지 않으면, 함수가 현재 스키마로 생성됩니다. 유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요.

모든 UDF 이름에 `f_`를 접두사로 사용하는 것이 좋습니다. Amazon Redshift는 UDF 이름에 대해 `f_` 접두사를 예약하므로, `f_` 접두사를 사용하면 UDF 이름이 기존 또는 향후의 Amazon Redshift 내장 SQL 함수 이름과 충돌하지 않을 것입니다. 자세한 내용은 UDF 이름 충돌 방지 단원을 참조하십시오.

입력 인수에 대한 데이터 형식이 서로 다른 경우 함수 이름이 동일한 함수를 2개 이상 정의할 수 있습니다. 즉, 함수 이름이 오버로드됩니다. 자세한 내용은 함수 이름 오버로드 단원을 참조하십시오.

py_arg_name py_arg_data_type | sql_arg_data_type

Python UDF에 대해서는 입력 인수 이름과 데이터 형식의 목록입니다. SQL UDF에 대해서는 인수 이름이 없는 데이터 형식의 목록입니다. Python UDF에서는 인수 이름을 이용한 인수를 참조하십시오. SQL UDF에서는 인수 목록에서 인수의 순서를 토대로 $1, $2, ... 등을 이용한 인수를 참조하십시오.

SQL UDF의 경우, 입력 및 반환 데이터 형식은 모든 표준 Amazon Redshift 데이터 형식일 수 있습니다. Python UDF의 경우, 입력 및 반환 데이터 형식은 SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE 또는 TIMESTAMP일 수 있습니다. 또한 Python 사용자 정의 함수(UDF)는 ANYELEMENT 데이터 형식을 지원합니다. 이는 런타임에 제공되는 해당 인수의 데이터 형식을 기준으로 표준 데이터 형식으로 자동 변환됩니다. 여러 인수가 ANYELEMENT를 사용하는 경우 이들 인수는 모두 목록에서 첫 번째 ANYELEMENT 인수를 기준으로 런타임에 동일한 데이터 형식으로 확인됩니다. 자세한 내용은 Python UDF 데이터 형식 및 데이터 타입 섹션을 참조하세요.

인수를 최대 32개까지 지정할 수 있습니다.

RETURNS data_type

함수에 의해 반환되는 값의 데이터 형식입니다. RETURNS 데이터 형식은 임의의 표준 Amazon Redshift 데이터 형식일 수 있습니다. 뿐만 아니라, Python UDF는 런타임에 제공되는 인수를 기준으로 표준 데이터 형식으로 자동 변환되는 ANYELEMENT의 데이터 형식을 사용할 수 있습니다. 반환 데이터 형식에 대해 ANYELEMENT를 지정하는 경우 하나 이상의 인수가 ANYELEMENT를 사용해야 합니다. 실제 반환 데이터 형식은 함수가 호출될 때 ANYELEMENT 인수에 대해 제공되는 데이터 형식과 일치합니다. 자세한 내용은 Python UDF 데이터 형식 단원을 참조하십시오.

VOLATILE | STABLE | IMMUTABLE

쿼리 최적화 프로그램에 함수의 휘발성에 대해 알립니다.

함수에 대해 유효한 가장 엄격한 휘발성 범주로 함수에 레이블을 지정하면 최상으로 최적화할 수 있습니다. 그러나 범주가 너무 엄격하면 최적화 프로그램이 일부 호출을 잘못 건너뛰어 잘못된 결과 집합을 초래할 위험이 있습니다. 가장 덜 엄격한 범주부터 시작해서 엄격성의 순으로 휘발성 범주를 나열하면 다음과 같습니다.

- VOLATILE
- STABLE
- IMMUTABLE

VOLATILE

인수가 같을 경우, 함수는 단일 명령문의 행에 대해서도 연속적인 호출에 대해 상이한 결과를 반환할 수 있습니다. 쿼리 최적화 프로그램은 휘발성 함수의 동작에 대해 어떤 가정도 할 수 없으므로, 휘발성 함수를 사용하는 쿼리가 모든 입력 행에 대해 함수를 재평가해야 합니다.

STABLE

인수가 같을 경우, 함수는 단일 명령문 내에서 처리되는 모든 행에 대해 동일한 결과를 반환하도록 보장됩니다. 이 함수는 서로 다른 명령문에서 호출될 경우 서로 다른 결과를 반환할 수 있습니다. 최적화 프로그램은 이 범주를 사용하여 단일 명령문 내에서 함수의 여러 호출을 명령문에 대한 단일 호출로 최적화할 수 있습니다.

IMMUTABLE

인수가 같을 경우, 함수는 항상 동일한 결과를 계속 반환합니다. 쿼리가 일정한 인수로 IMMUTABLE 함수를 호출할 때는 최적화 프로그램이 함수를 미리 평가합니다.

AS $$ statement $$

실행할 문을 둘러싼 구문입니다. 리터럴 키워드 AS $$ 및 $$가 필요합니다.

Amazon Redshift에서는 $ 인용이라는 형식을 사용하여 함수의 문을 묶어야 합니다. 묶음 기호 내에 있는 것은 모두 정확히 그대로 전달됩니다. 문자열의 내용은 문자 그대로 작성되므로 특수 문자를 이스케이프할 필요가 없습니다.

$ 인용 사용 시, 달러 기호 쌍($$)을 사용하면 다음 예에 표시된 것처럼 실행할 문의 시작과 끝을 나타낼 수 있습니다.

```
$$ my statement $$
```

각 쌍의 달러 기호들 사이에서 선택적으로 문을 식별하는 데 도움이 되는 문자열을 지정할 수 있습니다. 사용하는 문자열은 묶음 쌍의 시작과 끝에서 모두 동일해야 합니다. 이 문자열은 대/소문자를 구분하고 달러 기호를 포함할 수 없는 경우를 제외하면 따옴표가 없는 식별자와 똑같은 제약 조건을 따릅니다. 다음 예에서는 `test` 문자열을 사용합니다.

```
$test$ my statement $test$
```

달러 인용에 대한 자세한 내용은 PostgreSQL 설명서의 어휘 구조에서 “달러 기호로 인용된 문자열 상수”를 참조하십시오.

python_program

값을 반환하는 유효하고 실행 가능한 Python 프로그램입니다. 함수와 함께 전달하는 문은 Python 웹사이트의 Python 코드 스타일 가이드에 지정된 들여쓰기 요구 사항을 준수해야 합니다. 자세한 내용은 UDF에 대한 Python 언어 지원 단원을 참조하십시오.

SQL 절

SQL SELECT 절.

SELECT 절은 다음 유형의 절을 포함할 수 없습니다.

- FROM
- INTO
- WHERE
- GROUP BY
- ORDER BY
- LIMIT

LANGUAGE { plpythonu | sql }

Python에는 `plpythonu`를 지정합니다. SQL에는 `sql`을 지정합니다. SQL 또는 plpythonu에 대한 언어에 사용 권한이 필요합니다. 자세한 내용은 UDF 보안 및 권한 단원을 참조하십시오.

사용 노트

중첩 함수

SQL UDF 안에서 다른 SQL 사용자 정의 함수(UDF)를 호출할 수 있습니다. CREATE FUNCTION 명령을 실행할 때 중첩 함수가 있어야 합니다. Amazon Redshift는 UDF에 대한 종속성을 추적하지 않으므로 중첩 함수를 삭제해도 Amazon Redshift는 오류를 반환하지 않습니다. 그러나 중첩 함수가 존재하지 않으면 UDF가 실패합니다. 예를 들어 다음 함수는 SELECT 절에 `f_sql_greater` 함수를 호출합니다.

```
create function f_sql_commission (float, float )
  returns float
stable
as $$
  select f_sql_greater ($1, $2)
$$ language sql;
```

UDF 보안 및 권한

UDF를 새로 만들려면 SQL 또는 plpythonu(Python)에 대한 언어에 사용 권한이 필요합니다. 기본적으로 USAGE ON LANGUAGE SQL은 PUBLIC에 허용됩니다. 그러나 특정 사용자 또는 그룹에 USAGE ON LANGUAGE PLPYTHONU 권한을 명시적으로 허용해야 합니다.

SQL에 대한 사용을 취소하려면 먼저 PUBLIC에서의 사용을 취소해야 합니다. 그런 다음 SQL UDF 생성이 허용된 사용자 또는 그룹에게만 SQL에 대한 사용 권한을 허용합니다. 다음은 PUBLIC에서의 SQL 사용을 취소한 후 사용자 그룹 `udf_devs`에 사용을 허용하는 예시입니다.

```
revoke usage on language sql from PUBLIC;
grant usage on language sql to group udf_devs;
```

UDF를 실행하려면 각 함수마다 실행 권한이 필요합니다. 기본적으로 새로운 UDF에 대한 실행 권한은 PUBLIC에 허용됩니다. 사용을 제한하려면 함수에 대해 PUBLIC에서 실행 권한을 취소합니다. 그런 다음 개인 혹은 그룹에 권한을 허용합니다.

다음은 PUBLIC에서 `f_py_greater` 함수에 대한 실행 권한을 취소한 다음 사용자 그룹 `udf_devs`에 사용 권한을 부여하는 예입니다.

```
revoke execute on function f_py_greater(a float, b float) from PUBLIC;
grant execute on function f_py_greater(a float, b float) to group udf_devs;
```

기본적으로 수퍼유저는 모든 권한을 갖습니다.

자세한 내용은 GRANT 및 REVOKE 섹션을 참조하세요.

예시

스칼라 Python UDF 예시

다음 예에서는 두 정수를 비교하여 더 큰 값을 반환하는 Python UDF를 생성합니다.

```
create function f_py_greater (a float, b float)
  returns float
stable
as $$
  if a > b:
    return a
  return b
$$ language plpythonu;
```

다음 예에서는 SALES 테이블을 쿼리하고 COMMISSION 또는 PRICEPAID의 20% 중 큰 값을 반환하는 새 `f_py_greater` 함수를 호출합니다.

```
select f_py_greater (commission, pricepaid*0.20) from sales;
```

스칼라 SQL UDF 예시

다음은 2개의 수를 비교하여 더 큰 값을 반환하는 함수의 생성 예입니다.

```
create function f_sql_greater (float, float)
  returns float
stable
as $$
  select case when $1 > $2 then $1
    else $2
  end
$$ language sql;
```

다음은 새로운 `f_sql_greater` 함수를 호출하여 SALES 테이블에 대한 쿼리를 실행한 후 COMMISSION 또는 PRICEPAID의 20% 중에서 더 큰 값을 반환하는 쿼리입니다.

```
select f_sql_greater (commission, pricepaid*0.20) from sales;
```

create group

새로운 사용자 그룹을 정의합니다. 수퍼유저만이 그룹을 생성할 수 있습니다.

구문

```
CREATE GROUP group_name
[ [ WITH ] [ USER username ] [, ...] ]
```

파라미터

group_name

새로운 사용자 그룹의 이름입니다. 2개의 밑줄로 시작하는 그룹 이름은 Amazon Redshift 내부 용도로 예약되어 있습니다. 유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요.

WITH

CREATE GROUP을 위한 추가 파라미터를 표시하는 선택적 구문입니다.

USER

한 명 이상의 사용자를 그룹에 추가합니다.

사용자 이름

그룹에 추가할 사용자의 이름입니다.

예시

다음 예에서는 두 사용자 ADMIN1 및 ADMIN2가 있고 ADMIN_GROUP으로 명명된 사용자 그룹을 생성합니다.

```
create group admin_group with user admin1, admin2;
```

CREATE IDENTITY PROVIDER

새 자격 증명 공급자를 정의합니다. 슈퍼 사용자만 자격 증명 공급자를 생성할 수 있습니다.

구문

```
CREATE IDENTITY PROVIDER identity_provider_name TYPE type_name
NAMESPACE namespace_name
```

```
[PARAMETERS parameter_string]
[APPLICATION_ARN arn]
[IAM_ROLE iam_role]
[AUTO_CREATE_ROLES
    [ TRUE [ { INCLUDE | EXCLUDE } GROUPS LIKE filter_pattern] |
      FALSE
    ]
  ];
```

파라미터

identity_provider_name

새 자격 증명 공급자 이름입니다. 유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요.

type_name

인터페이스할 자격 증명 공급자입니다. Azure는 현재 유일하게 지원되는 자격 증명 공급자입니다.

namespace_name

네임스페이스입니다. 자격 증명 공급자 디렉터리에 대한 고유한 약식 식별자입니다.

parameter_string

자격 증명 공급자에 필요한 파라미터와 값이 포함된 올바른 형식의 JSON 객체를 포함하는 문자열입니다.

arn

IAM Identity Center 관리형 애플리케이션의 Amazon 리소스 이름(ARN)입니다. 이 파라미터는 ID 제공업체 유형이 AWSIDC인 경우에만 적용됩니다.

iam_role

IAM Identity Center에 대한 연결 권한을 제공하는 IAM 역할입니다. 이 파라미터는 ID 제공업체 유형이 AWSIDC인 경우에만 적용됩니다.

auto_create_roles

역할 자동 생성 기능을 활성화하거나 비활성화합니다. SQL에서 옵션이 제공되지 않으면 기본값은 FALSE이고, 값 없이 옵션이 제공되는 경우 기본값은 TRUE입니다.

그룹을 포함하려면 INCLUDE를 지정합니다. 기본값은 비어 있습니다. 즉, AUTO_CREATES_ROLES가 켜져 있는 경우 모든 그룹이 포함됩니다.

그룹을 제외하려면 EXCLUDE를 지정합니다. 기본값은 비어 있습니다. 즉, AUTO_CREATES_ROLES가 켜져 있는 경우 그룹을 제외하지 않습니다.

예시

다음 예에서는 TYPE azure를 통해 oauth_standard라는 자격 증명 공급자를 생성하여 Microsoft Azure Active Directory(AD)와의 통신을 설정합니다.

```
CREATE IDENTITY PROVIDER oauth_standard TYPE azure
NAMESPACE 'aad'
PARAMETERS '{"issuer":"https://sts.windows.net/2sdfdsf-d475-420d-b5ac-667adad7c702/",
"client_id":"87f4aa26-78b7-410e-bf29-57b39929ef9a",
"client_secret":"BUAH~ewrqewrqwerUUY^%tHe1oNZShoiU7",
"audience":["https://analysis.windows.net/powerbi/connector/AmazonRedshift"]
}'
```

IAM Identity Center의 관리형 애플리케이션을 기존의 프로비저닝된 클러스터 또는 Amazon Redshift Serverless 작업 그룹에 연결할 수 있습니다. 이렇게 하면 IAM Identity Center를 통해 Redshift 데이터베이스에 대한 액세스를 관리할 수 있습니다. 이렇게 하려면 다음 샘플과 같은 SQL 명령을 실행합니다. 데이터베이스 관리자여야 합니다.

```
CREATE IDENTITY PROVIDER "redshift-idc-app" TYPE AWSIDC
NAMESPACE 'awsidc'
APPLICATION_ARN 'arn:aws:sso::123456789012:application/ssoins-12345f67fe123d4/apl-
a0b0a12dc123b1a4'
IAM_ROLE 'arn:aws:iam::123456789012:role/MyRedshiftRole';
```

이 경우 연결할 관리형 애플리케이션은 애플리케이션 ARN으로 식별됩니다. SELECT * FROM SVV_IDENTITY_PROVIDERS;를 실행하여 이를 찾을 수 있습니다.

추가 예를 포함하여 CREATE IDENTITY PROVIDER 사용에 대한 자세한 내용은 Amazon Redshift 용 기본 ID 제공업체(IdP) 페더레이션을 참조하세요. Redshift에서 IAM Identity Center에 대한 연결을 설정하는 방법에 대한 자세한 내용은 Redshift를 IAM Identity Center와 연결하여 사용자에게 Single Sign-On 경험을 제공합니다를 참조하세요.

CREATE LIBRARY

사용자가 CREATE FUNCTION 명령으로 UDF(사용자 정의 함수)를 만들 때 사용자가 포함할 수 있도록 제공되는 Python 라이브러리를 설치합니다. 사용자가 설치한 라이브러리의 총 크기는 100MB를 초과할 수 없습니다.

CREATE LIBRARY를 트랜잭션 블록(BEGIN … END) 내에서 실행할 수 없습니다. 버전 관리에 대한 자세한 내용은 직렬화 가능 격리 단원을 참조하십시오.

Amazon Redshift는 Python 버전 2.7을 지원합니다. 자세한 내용은 www.python.org를 참조하세요.

자세한 내용은 예제: 사용자 지정 Python 라이브러리 모듈 가져오기 단원을 참조하십시오.

필수 권한

CREATE LIBRARY에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- CREATE LIBRARY 권한이 있거나 지정된 언어의 권한이 있는 사용자

구문

```
CREATE [ OR REPLACE ] LIBRARY library_name LANGUAGE plpythonu
FROM
{ 'https://file_url'
| 's3://bucketname/file_name'
authorization
  [ REGION [AS] 'aws_region']
  IAM_ROLE { default | ‘arn:aws:iam::<AWS ##-id>:role/<role-name>’ }
}
```

파라미터

OR REPLACE

이름이 같은 라이브러리인 경우 이런 라이브러리가 이미 하나 존재하므로 기존 라이브러리가 대체됨을 지정합니다. REPLACE는 즉시 커밋합니다. 라이브러리에 종속된 UDF가 동시에 실행 중인 경우 UDF는 트랜잭션 내에서 실행 중이더라도 실패하거나 예기치 않은 결과를 반환할 수 있습니다. 소유자 또는 수퍼유저만이 라이브러리를 바꿀 수 있습니다.

library_name

설치할 라이브러리의 이름입니다. Python Standard Library 모듈 또는 Amazon Redshift에 사전 설치되어 있는 Python 모듈과 동일한 이름의 모듈이 포함되어 있는 라이브러리는 생성할 수 없습니다. 기존의 사용자 설치 라이브러리가 설치할 라이브러리와 같은 Python 패키지를 사용하는 경우 기존 라이브러리를 삭제한 후 새 라이브러리를 설치해야 합니다. 자세한 내용은 UDF에 대한 Python 언어 지원 단원을 참조하십시오.

LANGUAGE plpythonu

사용할 언어입니다. Python(plpythonu)이 유일하게 지원되는 언어입니다. Amazon Redshift는 Python 버전 2.7을 지원합니다. 자세한 내용은 www.python.org를 참조하세요.

FROM

라이브러리 파일의 위치. Amazon S3 버킷과 객체 이름을 지정하거나 공용 웹사이트에서 파일을 다운로드하기 위한 URL을 지정할 수 있습니다. 라이브러리는 `.zip` 파일의 형식으로 압축해야 합니다. 자세한 내용은 Python 설명서의 Building and Installing Python Modules 섹션을 참조하세요.

https://file_url

공용 웹사이트에서 파일을 다운로드하기 위한 URL입니다. 이 URL은 최대 3개의 리디렉션을 포함할 수 있습니다. 다음은 파일 URL의 예입니다.

```
'https://www.example.com/pylib.zip'
```

s3://버킷 이름/파일 이름

라이브러리 파일이 들어 있는 단일 Amazon S3 객체의 경로입니다. 다음은 Amazon S3 객체 경로의 예입니다.

```
's3://amzn-s3-demo-bucket/my-pylib.zip'
```

Amazon S3 버킷을 지정하는 경우 파일 다운로드 권한을 가진 AWS 사용자를 위한 자격 증명도 제공해야 합니다.

> **Important**
>
> Amazon S3 버킷이 Amazon Redshift 클러스터와 동일한 AWS 리전에 없는 경우에는 REGION 옵션을 사용하여 데이터가 위치한 AWS 리전을 지정해야 합니다. aws_region의 값은 COPY 명령에 대한 REGION 파라미터 설명에서 표에 나와 있는 AWS 리전과 일치해야 합니다.

권한 부여

클러스터가 라이브러리 파일이 있는 Amazon S3 버킷에 액세스하기 위한 인증 및 권한 부여에 사용할 방법을 나타내는 절입니다. 클러스터에 LIST 및 GET 작업으로 Amazon S3에 액세스할 권한이 있어야 합니다.

권한 부여를 위한 구문은 COPY 명령 권한 부여에 대한 것과 동일합니다. 자세한 내용은 권한 부여 파라미터 단원을 참조하십시오.

```
IAM_ROLE { default | 'arn:aws:iam::<AWS ##-id>:role/<role-name>'
```

기본 키워드를 사용하여 CREATE LIBRARY 명령이 실행될 때 Amazon Redshift에서 기본값으로 설정되고 클러스터와 연결된 IAM 역할을 사용하도록 합니다.

클러스터가 인증 및 권한 부여에 사용하는 IAM 역할의 Amazon 리소스 이름(ARN)을 사용합니다. IAM_ROLE을 지정하면 ACCESS_KEY_ID 및 SECRET_ACCESS_KEY, SESSION_TOKEN 또는 CREDENTIALS는 사용할 수 없습니다.

선택적으로, Amazon S3 버킷이 서버 측 암호화를 사용하는 경우 credentials-args 문자열에 암호화 키를 제공합니다. 임시 보안 자격 증명을 사용하는 경우 credentials-args 문자열에 임시 토큰을 제공합니다.

자세한 내용은 임시 보안 자격 증명 단원을 참조하십시오.

REGION [AS] aws_region

Amazon S3 버킷이 위치한 AWS 리전입니다. Amazon S3 버킷이 Amazon Redshift 클러스터와 같은 AWS 리전에 있지 않을 때는 REGION이 필요합니다. aws_region의 값은 COPY 명령에 대한 REGION 파라미터 설명에서 표에 나와 있는 AWS 리전과 일치해야 합니다.

기본적으로, CREATE LIBRARY에서는 Amazon S3 버킷이 Amazon Redshift 클러스터와 같은 AWS 리전에 위치한 것으로 가정합니다.

예시

다음 두 예에서는 `urlparse3-1.0.3.zip`이라는 파일에 패키지된urlparse Python 모듈을 설치합니다.

다음 명령으로 미국 동부 리전에 위치한 Amazon S3 버킷으로 업로드된 패키지에서 `f_urlparse`로 명명된 UDF 라이브러리를 설치합니다.

```
create library f_urlparse
language plpythonu
from 's3://amzn-s3-demo-bucket/urlparse3-1.0.3.zip'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
region as 'us-east-1';
```

다음 예에서는 라이브러리 파일에서 f_urlparse로 명명된 라이브러리를 웹사이트에 설치합니다.

```
create library f_urlparse
language plpythonu
from 'https://example.com/packages/urlparse3-1.0.3.zip';
```

마스킹 정책 생성

지정된 형식의 데이터를 난독화하는 새 동적 데이터 마스킹 정책을 만듭니다. 동적 데이터 마스킹에 대한 자세한 내용은 동적 데이터 마스킹(동적 데이터 마스킹(미리 보기)을 참조하세요.

sys:secadmin 역할이 부여된 수퍼유저와 사용자 또는 역할은 마스킹 정책을 생성할 수 있습니다.

구문

```
CREATE MASKING POLICY
   policy_name [IF NOT EXISTS]
   WITH (input_columns)
   USING (masking_expression);
```

파라미터

policy_name

마스킹 정책의 이름입니다. 마스킹 정책은 데이터베이스에 이미 있는 다른 마스킹 정책과 이름이 같을 수 없습니다.

input_columns

형식(col1 형식, col2 형식...)의 열 이름 튜플입니다.

열 이름은 마스킹 표현식의 입력으로 사용됩니다. 열 이름은 마스킹되는 열의 이름과 일치하지 않아도 되지만 입력 및 출력 데이터 유형은 일치해야 합니다.

masking_expression

대상 열을 변환하는 데 사용되는 SQL 표현식입니다. 문자열 조작 함수와 같은 데이터 조작 함수를 사용하거나 SQL, Python 또는 AWS Lambda로 작성된 사용자 정의 함수와 함께 작성할 수 있습니다. 출력이 여러 개인 정책을 마스킹하기 위해 열 표현식의 튜플을 포함할 수 있습니다. 상수를 마스킹 표현식으로 사용하는 경우 입력 유형과 일치하는 유형으로 상수를 명시적으로 변환해야 합니다.

마스킹 표현식에서 사용하는 모든 사용자 정의 함수에 대한 USAGE 권한이 있어야 합니다.

CREATE MATERIALIZED VIEW

하나 이상의 Amazon Redshift 테이블을 기반으로 구체화된 뷰를 생성합니다. Spectrum 또는 통합 쿼리를 사용하여 생성된 외부 테이블을 기반으로 구체화된 뷰를 만들 수도 있습니다. Spectrum에 대한 자세한 내용은 Amazon Redshift Spectrum 섹션을 참조하세요. 연합 쿼리에 대한 자세한 내용은 Amazon Redshift에서 연합 쿼리를 사용하여 데이터 쿼리 섹션을 참조하세요.

구문

```
CREATE MATERIALIZED VIEW mv_name
[ BACKUP { YES | NO } ]
[ table_attributes ]
[ AUTO REFRESH { YES | NO } ]
AS query
```

파라미터

BACKUP

자동 및 수동 클러스터 스냅샷에 구체화된 뷰를 포함해야 할지 여부를 지정하는 절입니다.

중요한 데이터를 포함하지 않는 구체화된 뷰의 경우 BACKUP NO를 지정하여 스냅샷을 생성하고 스냅샷으로부터 복원할 때의 처리 시간을 절약하고 Amazon Simple Storage Service의 스토리지 공간을 줄입니다. BACKUP NO 설정은 클러스터 내에 있는 다른 노드로의 데이터 자동 복제에 아무런 영향도 미치지 않으므로, 노드 장애가 발생할 경우 BACKUP NO가 지정된 구체화된 뷰가 복원됩니다. 기본값은 BACKUP YES입니다.

table_attributes

다음을 포함하여 구체화된 보기의 데이터가 분산되는 방식을 지정하는 절입니다.

- 구체화된 보기의 분산 스타일입니다(`DISTSTYLE { EVEN | ALL | KEY }` 형식). 이 절을 생략한 경우 배포 스타일은 `EVEN`입니다. 자세한 내용은 분산 스타일 단원을 참조하십시오.
- 구체화된 보기의 분산 키입니다(`DISTKEY ( distkey_identifier )` 형식). 자세한 내용은 분산 스타일 지정 단원을 참조하십시오.
- 구체화된 보기의 정렬 키입니다(`SORTKEY ( column_name [, ...] )` 형식). 자세한 내용은 정렬 키 단원을 참조하십시오.

AS query

구체화된 뷰와 해당 콘텐츠를 정의하는 유효한 SELECT 문입니다. 쿼리의 결과 집합은 구체화된 보기의 열과 행을 정의합니다. 구체화된 뷰 생성 시 제한 사항에 대한 자세한 내용은 제한 사항 섹션을 참조하세요.

또한 쿼리에 사용되는 특정 SQL 언어 구조는 구체화된 보기를 증분 새로 고침으로 할지 아니면 전체 새로 고침으로 할지를 결정합니다. 새로 고침 방법에 대한 자세한 내용은 REFRESH MATERIALIZED VIEW 섹션을 참조하세요. 증분 새로 고침의 제한 사항에 대한 자세한 내용은 증분 새로 고침에 대한 제한 사항 섹션을 참조하세요.

쿼리에 증분 새로 고침을 지원하지 않는 SQL 명령이 포함된 경우 Amazon Redshift는 구체화된 뷰가 전체 새로 고침 사용을 알리는 메시지를 표시합니다. SQL 클라이언트 애플리케이션에 따라 메시지가 표시될 수도 있고 표시되지 않을 수도 있습니다. 구체화된 보기에서 사용되는 새로 고침 유형을 보려면 STV_MV_INFO의 `state` 열을 확인합니다.

AUTO REFRESH

구체화된 뷰가 기본 테이블의 최신 변경 사항으로 자동으로 새로 고쳐져야 하는지 여부를 정의하는 절입니다. 기본 값은 NO입니다. 자세한 내용은 구체화된 뷰 새로 고침 단원을 참조하십시오.

사용 노트

구체화된 보기를 생성하려면 다음 권한이 있어야 합니다.

- 스키마에 대한 CREATE 권한
- 구체화된 뷰를 생성하기 위한 기본 테이블에 대한 테이블 수준 또는 열 수준 SELECT 권한입니다. 특정 열에 대한 열 수준 권한이 있는 경우 해당 열에 대해서만 구체화된 뷰를 생성할 수 있습니다.

데이터 공유의 구체화된 뷰에 대한 증분 새로 고침

Amazon Redshift는 기본 테이블이 공유될 때 소비자 데이터 공유의 구체화된 뷰에 대한 자동 및 증분 새로 고침을 지원합니다. 증분 새로 고침은 Amazon Redshift가 이전 새로 고침 이후에 발생한 기본 테이블의 변경 사항을 식별하고 구체화된 뷰의 해당 레코드만 업데이트하는 작업입니다. 이는 전체 새로 고침보다 더 빠르게 실행되며 워크로드 성능을 개선합니다. 증분 새로 고침을 활용하기 위해 구체화된 뷰 정의를 변경할 필요는 없습니다.

구체화된 뷰에서 증분 새로 고침을 활용할 때는 다음과 같은 몇 가지 제한 사항이 있습니다.

- 구체화된 뷰는 로컬 또는 원격 데이터베이스 하나만 참조해야 합니다.

- 증분 새로 고침은 새 구체화된 뷰에서만 사용할 수 있습니다. 따라서 증분 새로 고침이 발생하려면 기존의 구체화된 뷰를 삭제하고 다시 생성해야 합니다.

데이터 공유에서 구체화된 뷰를 생성하는 방법에 대한 자세한 내용은 여러 쿼리 예제가 Amazon Redshift 데이터 공유에서 보기 작업을 참조하세요.

구체화된 보기 또는 기본 테이블에 대한 DDL 업데이트

Amazon Redshift에서 구체화된 뷰를 사용하는 경우 구체화된 뷰 또는 기본 테이블에 대한 데이터 정의 언어(DDL) 업데이트와 관련된 다음 사용 노트를 따릅니다.

- 기본 테이블을 참조하는 구체화된 보기에 영향을 주지 않고 기본 테이블에 열을 추가할 수 있습니다.
- 일부 작업은 구체화된 보기를 전혀 새로 고칠 수 없는 상태로 둘 수 있습니다. 예를 들어, 열 이름 바꾸기 또는 삭제, 열 유형 변경 및 스키마 이름 변경 등의 작업이 그렇습니다. 이러한 구체화된 보기는 쿼리할 수 있지만 새로 고칠 수는 없습니다. 이 경우 구체화된 보기를 삭제하고 다시 생성해야 합니다.
- 일반적으로 구체화된 보기의 정의(해당 SQL 문)는 변경할 수 없습니다.
- 구체화된 보기의 이름을 바꿀 수 없습니다.

제한 사항

다음 중 하나를 참조 또는 포함하는 구체화된 보기를 정의할 수 없습니다.

- 표준 뷰 또는 시스템 테이블 및 뷰.
- 임시 테이블.
- 사용자 정의 함수.
- ORDER BY, LIMIT 또는 OFFSET 절
- 기본 테이블에 대한 지연 바인딩 참조. 다시 말해, 구체화된 보기의 SQL 쿼리 정의에서 참조된 기본 테이블 또는 관련 열이 존재해야 하며 유효해야 합니다.
- 리더 노드 전용 함수: CURRENT_SCHEMA, CURRENT_SCHEMAS, HAS_DATABASE_PRIVILEGE, HAS_SCHEMA_PRIVILEGE, HAS_TABLE_PRIVILEGE.
- 구체화된 뷰 정의에 변경 가능한 함수 또는 외부 스키마가 포함된 경우 AUTO REFRESH YES 옵션을 사용할 수 없습니다. 또한 다른 구체화된 뷰에서 구체화된 뷰를 정의할 때도 사용할 수 없습니다.
- 구체화된 뷰에서 ANALYZE를 수동으로 실행할 필요가 없습니다. 이는 현재 AUTO ANALYZE를 통해서만 이루어집니다. 자세한 내용은 테이블 분석 단원을 참조하십시오.

- RLS 보호 또는 DDM 보호 테이블

예시

다음 예에서는 조인 및 집계되는 3개의 기본 테이블에서 구체화된 뷰를 생성합니다. 각 행은 판매된 티켓 수가 있는 범주를 나타냅니다. tickets_mv 구체화된 보기를 쿼리할 때 tickets_mv 구체화된 보기에서 미리 계산된 데이터에 직접 액세스합니다.

```
CREATE MATERIALIZED VIEW tickets_mv AS
    select   catgroup,
    sum(qtysold) as sold
    from     category c, event e, sales s
    where    c.catid = e.catid
    and      e.eventid = s.eventid
    group by catgroup;
```

다음 예제에서는 이전 예제와 유사한 구체화된 뷰를 생성하고 집계 함수 MAX()를 사용합니다.

```
CREATE MATERIALIZED VIEW tickets_mv_max AS
    select   catgroup,
    max(qtysold) as sold
    from     category c, event e, sales s
    where    c.catid = e.catid
    and      e.eventid = s.eventid
    group by catgroup;

SELECT name, state FROM STV_MV_INFO;
```

다음 예에서는 UNION ALL 절을 사용하여 Amazon Redshift `public_sales` 테이블과 Redshift Spectrum `spectrum.sales` 테이블을 조인하여 구체화된 뷰 `mv_sales_vw`를 생성합니다. Amazon Redshift Spectrum의 CREATE EXTERNAL TABLE 명령에 대한 자세한 내용은 CREATE EXTERNAL TABLE 섹션을 참조하세요. Redshift Spectrum 외부 테이블은 Amazon S3의 데이터를 참조합니다.

```
CREATE MATERIALIZED VIEW mv_sales_vw as
select salesid, qtysold, pricepaid, commission, saletime from public.sales
union all
select salesid, qtysold, pricepaid, commission, saletime from spectrum.sales
```

다음 예에서는 연합 쿼리 외부 테이블을 기반으로 구체화된 보기 `mv_fq`를 생성합니다. 연합 쿼리에 대한 자세한 내용은 CREATE EXTERNAL SCHEMA 섹션을 참조하세요.

```
CREATE MATERIALIZED VIEW mv_fq as select firstname, lastname from apg.mv_fq_example;

select firstname, lastname from mv_fq;
 firstname | lastname
-----------+----------
 John      | Day
 Jane      | Doe
(2 rows)
```

다음 예에서는 구체화된 보기의 정의를 보여줍니다.

```
SELECT pg_catalog.pg_get_viewdef('mv_sales_vw'::regclass::oid, true);

pg_get_viewdef
-------------------------------------------------
create materialized view mv_sales_vw as select a from t;
```

다음 샘플은 구체화된 뷰 정의에서 AUTO REFRESH를 설정하는 방법을 보여주고 DISTSTYLE도 지정합니다. 먼저 간단한 기본 테이블을 생성합니다.

```
CREATE TABLE baseball_table (ball int, bat int);
```

그런 다음 구체화된 뷰를 생성합니다.

```
CREATE MATERIALIZED VIEW mv_baseball DISTSTYLE ALL AUTO REFRESH YES AS SELECT ball AS
 baseball FROM baseball_table;
```

이제 mv_baseball 구체화된 뷰를 쿼리할 수 있습니다. 구체화된 뷰에 대해 AUTO REFRESH가 켜져 있는지 확인하려면 STV_MV_INFO을 참조하십시오.

다음 샘플은 다른 데이터베이스의 소스 테이블을 참조하는 구체화된 뷰를 생성합니다. 소스 테이블인 database_A를 포함하는 데이터베이스가 database_B에서 생성한 구체화된 뷰와 동일한 클러스터 또는 작업 그룹에 있다고 가정합니다. (샘플을 자체 데이터베이스로 대체할 수 있습니다.) 먼저 cities라는 database_A에 cityname 열이 있는 테이블을 생성합니다. 열의 데이터 유형을 VARCHAR로 설정합니다. 소스 테이블을 생성한 후 database_B에서 다음 명령을 실행하여 소스가 cities 테이블인 구체화된 뷰를 생성합니다. FROM 절에 소스 테이블의 데이터베이스와 스키마를 지정해야 합니다.

```
CREATE MATERIALIZED VIEW cities_mv AS
SELECT  cityname
```

```
FROM    database_A.public.cities;
```

생성한 구체화된 뷰를 쿼리합니다. 쿼리는 원본 소스가 database_A의 cities 테이블인 레코드를 검색합니다.

```
select * from cities_mv;
```

SELECT 명령문을 실행하면 cities_mv가 레코드를 반환합니다. REFRESH 명령문을 실행할 때만 소스 테이블에서 레코드가 새로 고쳐집니다. 또한 구체화된 뷰에서는 레코드를 직접 업데이트할 수 없습니다. 구체화된 뷰의 데이터 새로 고침에 대한 자세한 내용은 REFRESH MATERIALIZED VIEW 섹션을 참조하세요.

구체화된 보기 개요와 구체화된 보기를 새로 고치고 삭제하는 데 사용되는 SQL 명령에 대한 자세한 내용은 다음 주제를 참조하십시오.

- Amazon Redshift의 구체화된 뷰
- REFRESH MATERIALIZED VIEW
- DROP MATERIALIZED VIEW

CREATE MODEL

주제

- 사전 조건
- 필수 권한
- 비용 관리
- 전체 CREATE MODEL
- 파라미터
- 사용 노트
- 사용 사례

사전 조건

CREATE MODEL 문을 사용하기 전에 Amazon Redshift 기계 학습 사용을 위한 클러스터 설정의 사전 조건을 충족해야 합니다. 다음은 사전 조건을 개괄적으로 요약한 것입니다.

- AWS 관리 콘솔 또는 AWS 명령줄 인터페이스(AWS CLI)를 사용하여 Amazon Redshift 클러스터를 생성합니다.
- 클러스터를 생성하는 동안 AWS Identity and Access Management(IAM) 정책을 연결합니다.
- Amazon Redshift와 SageMaker AI가 다른 서비스와 상호 작용하는 역할을 맡도록 허용하려면 IAM 역할에 적절한 신뢰 정책을 추가합니다.

IAM 역할, 신뢰 정책 및 기타 사전 조건에 대한 자세한 내용은 Amazon Redshift 기계 학습 사용을 위한 클러스터 설정 섹션을 참조하세요.

다음으로 CREATE MODEL 문에 대한 여러 사용 사례를 찾아볼 수 있습니다.

- 단순 CREATE MODEL
- 사용자 안내에 따라 CREATE MODEL
- AUTO OFF로 CREATE XGBoost 모델
- 기존 보유 모델 사용(BYOM) - 로컬 추론
- 기존 보유 모델 사용(BYOM) - 원격 추론
- K-MEANS를 사용한 CREATE MODEL
- 전체 CREATE MODEL

필수 권한

CREATE MODEL에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- CREATE MODEL 권한이 있는 사용자
- GRANT CREATE MODEL 권한이 있는 역할

비용 관리

Amazon Redshift ML은 기존 클러스터 리소스를 사용하여 예측 모델을 생성하므로 추가 비용을 지불하지 않아도 됩니다. 그러나 클러스터 크기를 조정해야 하거나 모델을 훈련하려는 경우 추가 비용이 발생할 수 있습니다. Amazon Redshift AI 기계 학습은 모델 훈련에 Amazon SageMaker를 사용하며, 이 경우 추가 관련 비용이 발생합니다. 훈련에 소요되는 최대 시간을 제한하거나 모델 훈련에 사용되는 훈련 예제의 수를 제한하는 등 추가 비용을 관리하는 방법이 있습니다. 자세한 내용은 Amazon Redshift 기계 학습 사용 비용을 참조하세요.

전체 CREATE MODEL

다음은 전체 CREATE MODEL 구문의 기본 옵션을 요약한 것입니다.

전체 CREATE MODEL 구문

다음은 CREATE MODEL 문의 전체 구문입니다.

> **Important**
>
> CREATE MODEL 문을 사용하여 모델을 생성할 때 다음 구문의 키워드 순서를 따릅니다.

```
CREATE MODEL model_name
FROM { table_name | ( select_statement )  | 'job_name' }
[ TARGET column_name ]
FUNCTION function_name [ ( data_type [, ...] ) ]
[ RETURNS data_type ]
  -- supported only for BYOM
[ SAGEMAKER 'endpoint_name'[:'model_name']]
  -- supported only for BYOM remote inference
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
[ AUTO ON / OFF ]
  -- default is AUTO ON
[ MODEL_TYPE { XGBOOST | MLP | LINEAR_LEARNER | KMEANS | FORECAST } ]
  -- not required for non AUTO OFF case, default is the list of all supported types
  -- required for AUTO OFF
[ PROBLEM_TYPE ( REGRESSION | BINARY_CLASSIFICATION | MULTICLASS_CLASSIFICATION ) ]
  -- not supported when AUTO OFF
[ OBJECTIVE ( 'MSE' | 'Accuracy' | 'F1' | 'F1_Macro' | 'AUC' |
             'reg:squarederror' | 'reg:squaredlogerror'| 'reg:logistic'|
             'reg:pseudohubererror' | 'reg:tweedie' | 'binary:logistic' |
 'binary:hinge',
             'multi:softmax' | 'RMSE' | 'WAPE' | 'MAPE' | 'MASE' |
 'AverageWeightedQuantileLoss' ) ]
  -- for AUTO ON: first 5 are valid
  -- for AUTO OFF: 6-13 are valid
  -- for FORECAST: 14-18 are valid
[ PREPROCESSORS 'string' ]
  -- required for AUTO OFF, when it has to be 'none'
  -- optional for AUTO ON
[ HYPERPARAMETERS { DEFAULT | DEFAULT EXCEPT ( Key 'value' (,...) ) } ]
  -- support XGBoost hyperparameters, except OBJECTIVE
```

```
  -- required and only allowed for AUTO OFF
  -- default NUM_ROUND is 100
  -- NUM_CLASS is required if objective is multi:softmax (only possible for AUTO OFF)
 [ SETTINGS (
   S3_BUCKET 'amzn-s3-demo-bucket',  |
     -- required
  TAGS 'string', |
     -- optional
  KMS_KEY_ID 'kms_string', |
     -- optional
  S3_GARBAGE_COLLECT on / off, |
     -- optional, defualt is on.
  MAX_CELLS integer, |
     -- optional, default is 1,000,000
  MAX_RUNTIME integer (, ...) |
     -- optional, default is 5400 (1.5 hours)
  HORIZON integer, |
     -- required if creating a forecast model
  FREQUENCY integer, |
     -- required if creating a forecast model
  PERCENTILES string, |
     -- optional if creating a forecast model
  MAX_BATCH_ROWS integer -- optional for BYOM remote inference
     ) ]
```

파라미터

model_name

모델의 이름입니다. 스키마의 모델 이름은 고유해야 합니다.

FROM { table_name | (select_query) | 'job_name'}

table_name 또는 훈련 데이터를 지정하는 쿼리입니다. 시스템의 기존 테이블이거나 괄호로 묶인 Amazon Redshift 호환 SELECT 쿼리, 즉 ()일 수 있습니다. 쿼리 결과에는 2개 이상의 열이 있어야 합니다.

TARGET column_name

예측 대상이 되는 열의 이름입니다. FROM 절에 열이 있어야 합니다.

FUNCTION function_name (data_type [, ...])

생성할 함수의 이름과 입력 인수의 데이터 형식입니다. 데이터베이스에 있는 스키마의 스키마 이름을 함수 이름 대신 입력할 수 있습니다.

RETURNS data_type

모델의 함수에서 반환할 데이터 형식입니다. 반환된 SUPER 데이터 형식은 원격 추론이 있는 BYOM에만 적용됩니다.

SAGEMAKER 'endpoint_name'[:'model_name']

Amazon SageMaker AI 엔드포인트의 이름입니다. 엔드포인트 이름이 다중 모델 엔드포인트를 가리키는 경우 사용할 모델의 이름을 추가합니다. 엔드포인트는 Amazon Redshift 클러스터와 동일한 AWS 리전에서 호스팅되어야 합니다.

IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>'}

기본 키워드를 사용하여 CREATE MODEL 명령이 실행될 때 Amazon Redshift에서 기본값으로 설정되고 클러스터와 연결된 IAM 역할을 사용하도록 합니다. 또는 IAM 역할의 ARN을 지정하여 해당 역할을 사용할 수도 있습니다.

[AUTO ON / OFF]

전처리기, 알고리즘 및 하이퍼파라미터 선택의 CREATE MODEL 자동 검색을 켜거나 끕니다. 예측 모델을 생성할 때 on을 지정하면 Amazon Forecast가 데이터 세트의 각 시계열에 최적의 알고리즘 조합을 적용하는 AutoPredictor를 사용한다는 뜻입니다.

MODEL_TYPE { XGBOOST | MLP | LINEAR_LEARNER | KMEANS | FORECAST }

(옵션) 모델 유형을 지정합니다. XGBoost, 다층 퍼셉트론(MLP), KMEANS 또는 Linear Learner와 같은 특정 모델 유형의 모델을 훈련할지 여부를 지정할 수 있습니다. 이 모델은 모두 Amazon SageMaker AI Autopilot에서 지원하는 알고리즘입니다. 파라미터를 지정하지 않으면 지원되는 모든 모델 유형이 훈련 중 최상의 모델을 찾기 위해 검색됩니다. 또한 Redshift ML에서 예측 모델을 생성하여 정확한 시계열 예측을 생성할 수도 있습니다.

PROBLEM_TYPE (REGRESSION | BINARY_CLASSIFICATION | MULTICLASS_CLASSIFICATION)

(옵션) 문제 유형을 지정합니다. 문제 유형을 알고 있는 경우 해당 특정 모델 유형의 최상의 모델만 검색하도록 Amazon Redshift를 제한할 수 있습니다. 이 파라미터를 지정하지 않으면 데이터를 기반으로 훈련 중 문제 유형이 검색됩니다.

OBJECTIVE ('MSE' | 'Accuracy' | 'F1' | 'F1Macro' | 'AUC' | 'reg:squarederror' | 'reg:squaredlogerror' | 'reg:logistic' | 'reg:pseudohubererror' | 'reg:tweedie' | 'binary:logistic' | 'binary:hinge' | 'multi:softmax' | 'RMSE' | 'WAPE' | 'MAPE' | 'MASE' | 'AverageWeightedQuantileLoss')

(옵션) 기계 학습 시스템의 예측 품질을 측정하는 데 사용되는 목표 지표의 이름을 지정합니다. 이 지표는 데이터의 모델 파라미터 값에 대한 최상의 추정치를 제공하기 위해 훈련 중에 최적화됩니다. 지표를 명시적으로 지정하지 않은 경우 기본 동작은 MSE(회귀 분석의 경우), F1(이진 분류의 경

우), 정확도(다중 클래스 분류의 경우)를 자동으로 사용하는 것입니다. 목표에 대한 자세한 내용은 Amazon SageMaker AI API Reference의 AutoMLJobObjective 및 XGBOOST 설명서의 Learning task parameters를 참조하세요 RMSE, WAPE, MAPE, MASE 및 AverageWeightedQuantileLoss 값은 예측 모델에만 적용할 수 있습니다. 자세한 내용은 CreateAutoPredictor API 작업을 참조하세요.

PREPROCESSORS 'string'

(옵션) 특정 열 집합에 대한 전처리기의 특정 조합을 지정합니다. 형식은 columnSet의 목록과 각 열 집합에 적용할 적절한 변환입니다. Amazon Redshift는 특정 변환기 목록의 모든 변환기를 해당 ColumnSet의 모든 열에 적용합니다. 예를 들어 Imputer가 있는 OneHotEncoder를 열 t1 및 t2에 적용하려면 다음 샘플 명령을 사용합니다.

```
CREATE MODEL customer_churn
FROM customer_data
TARGET 'Churn'
FUNCTION predict_churn
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
PROBLEM_TYPE BINARY_CLASSIFICATION
OBJECTIVE 'F1'
PREPROCESSORS '[
...
  {"ColumnSet": [
      "t1",
      "t2"
    ],
    "Transformers": [
      "OneHotEncoder",
      "Imputer"
    ]
  },
  {"ColumnSet": [
      "t3"
    ],
    "Transformers": [
      "OneHotEncoder"
    ]
  },
  {"ColumnSet": [
      "temp"
    ],
    "Transformers": [
      "Imputer",
      "NumericPassthrough"
```

```
    ]
  }
]'
SETTINGS (
  S3_BUCKET 'amzn-s3-demo-bucket'
)
```

HYPERPARAMETERS { DEFAULT | DEFAULT EXCEPT (key ‘value’ (,..)) }

기본 XGBoost 파라미터가 사용되는지 아니면 사용자 지정 값으로 재정의되는지를 지정합니다. 작은따옴표로 값을 묶어야 합니다. 다음은 XGBoost에 대한 파라미터와 해당 기본값의 예입니다.

파라미터 이름	파라미터값	기본값	참고
num_class	Integer	다중 클래스 분류에 필요합니다.	N/A
num_round	Integer	100	N/A
tree_method	String	자동	N/A
max_depth	Integer	6	[0 , 10]
min_child_weight	Float	1	MinValue: 0, MaxValue: 120
subsample	Float	1	MinValue: 0.5, MaxValue: 1
gamma	Float	0	MinValue: 0, MaxValue: 5
alpha	Float	0	MinValue: 0, MaxValue: 1000
eta	Float	0.3	MinValue: 0.1, MaxValue: 0.5

파라미터 이름	파라미터값	기본값	참고
colsample_byleve	Float	1	MinValue: 0.1, MaxValue: 1
colsample_bynode	Float	1	MinValue: 0.1, MaxValue: 1
colsample_bytree	Float	1	MinValue: 0.5, MaxValue: 1
lambda	Float	1	MinValue: 0, MaxValue: 1000
max_delta_step	Integer	0	[0, 10]

SETTINGS (S3_BUCKET 'amzn-s3-demo-bucket', | TAGS 'string', | KMS_KEY_ID 'kms_string' , | S3_GARBAGE_COLLECT on / off, | MAX_CELLS integer , | MAX_RUNTIME (,...) , | HORIZON integer, | FREQUENCY forecast_frequency, | PERCENTILES array of strings)

S3_BUCKET 절은 중간 결과를 저장하는 데 사용되는 Amazon S3 위치를 지정합니다.

(선택 사항) TAGS 파라미터는 쉼표로 구분된 키-값 쌍의 목록으로, Amazon SageMaker AI에서 생성한 리소스에 태그를 지정하는 데 사용할 수 있으며 Amazon Forecast에 사용할 수 있습니다. 태그는 리소스를 구성하고 비용을 할당하는 데 도움이 됩니다 쌍의 값은 선택 사항이므로 `key=value` 형식을 사용하거나 키만 생성하여 태그를 만들 수 있습니다. Amazon Redshift의 태그에 대한 자세한 내용은 태그 개요를 참조하세요.

(선택 사항) KMS_KEY_ID는 Amazon Redshift가 AWS KMS 키로 서버 측 암호화를 사용하여 저장 데이터를 보호할지 여부를 지정합니다. 전송 중인 데이터는 보안 소켓 계층(SSL)으로 보호됩니다.

(선택 사항) S3_GARBAGE_COLLECT { ON | OFF }는 Amazon Redshift가 모델 훈련에 사용되는 결과 데이터 세트에 대해 가비지 수집을 수행할지 여부를 지정합니다. OFF로 설정하면 모델 훈련에 사용된 결과 데이터 집합과 모델이 Amazon S3에 남아 다른 용도로 사용할 수 있습니다. ON으로 설정하면 훈련이 완료된 후 Amazon Redshift가 Amazon S3에서 아티팩트를 삭제합니다. 기본값은 ON입니다.

(선택 사항) MAX_CELLS는 훈련 데이터의 셀 수를 지정합니다. 이 값은 레코드 수(훈련 쿼리 또는 테이블)에 열 수를 곱한 값입니다. 기본값은 1,000,000입니다.

(선택 사항) MAX_RUNTIME은 훈련할 최대 시간을 지정합니다. 데이터 집합 크기에 따라 훈련 작업이 더 빨리 완료되는 경우가 많습니다. 이는 훈련에 소요되는 최대 시간을 지정합니다. 기본값은 5,400(90분)입니다.

HORIZON은 예측 모델이 반환할 수 있는 최대 예측 수를 지정합니다. 모델을 학습시킨 후에는 이 정수를 변경할 수 없습니다. 이 파라미터는 예측 모델을 학습하는 경우 필요합니다.

FREQUENCY는 예측을 얼마나 세분화된 시간 단위로 설정할지를 지정합니다. 사용 가능한 옵션은 `Y | M | W | D | H | 30min | 15min | 10min | 5min | 1min`입니다. 이 파라미터는 예측 모델을 학습하는 경우 필요합니다.

(선택 사항) PERCENTILES는 예측기를 훈련하는 데 사용되는 예측 유형을 지정하는 쉼표로 구분된 문자열입니다. 예측 유형은 0.01에서 0.99까지의 사분위수(0.01 이상 증분)일 수 있습니다. 평균을 사용하여 평균 예측을 지정할 수도 있습니다. 최대 5개의 예측 유형을 지정할 수 있습니다.

MAX_BATCH_ROWS 정수

(선택) Amazon Redshift가 단일 SageMaker AI 간접 호출에 대한 단일 배치 요청으로 보내는 최대 행 수입니다. 원격 추론 기능이 있는 BYOM에만 지원됩니다. 이 파라미터의 최소값은 1입니다. 최대값은 `INT_MAX` 또는 2,147,483,647입니다. 이 파라미터는 입력 및 반환 데이터 형식이 모두 SUPER인 경우에만 필요합니다. 기본값은 `INT_MAX` 또는 2,147,483,647입니다.

사용 노트

CREATE MODEL을 사용할 때는 다음 사항을 고려하세요.

- CREATE MODEL 문은 비동기 모드에서 작동하며 Amazon S3로 훈련 데이터를 내보낼 때 반환됩니다. Amazon SageMaker AI의 나머지 훈련 단계는 백그라운드에서 진행됩니다. 훈련이 진행되는 동안 해당 추론 함수가 표시되지만 실행할 수는 없습니다. STV_ML_MODEL_INFO를 쿼리하여 훈련 상태를 볼 수 있습니다.
- 훈련은 기본적으로 자동 모델에서 백그라운드로 최대 90분 동안 실행될 수 있으며 확장될 수 있습니다. 훈련을 취소하려면 DROP MODEL 명령을 실행하기만 하면 됩니다.
- 모델을 생성하는 데 사용되는 Amazon Redshift 클러스터와 훈련 데이터 및 모델 아티팩트를 준비하는 데 사용되는 Amazon S3 버킷은 동일한 AWS 리전에 있어야 합니다.
- 모델 훈련 중 Amazon Redshift 및 SageMaker AI는 사용자가 제공하는 Amazon S3 버킷에 중간 아티팩트를 저장합니다. 기본적으로 Amazon Redshift는 CREATE MODEL 작업이 끝날 때 가비지 수

집을 수행합니다. Amazon Redshift는 Amazon S3에서 해당 객체를 제거합니다. Amazon S3에 이러한 아티팩트를 유지하려면 S3_GARBAGE COLLECT OFF 옵션을 설정합니다.

- FROM 절에 제공된 훈련 데이터에서 최소 500개의 행을 사용해야 합니다.
- CREATE MODEL 문을 사용할 때 FROM { table_name | (select_query) } 절에 특성(입력) 열을 최대 256개까지만 지정할 수 있습니다.
- AUTO ON의 경우 훈련 집합으로 사용할 수 있는 열 형식은 SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE, BOOLEAN, CHAR, VARCHAR, DATE, TIME, TIMETZ, TIMESTAMP 및 TIMESTAMPTZ입니다. AUTO OFF의 경우 훈련 집합으로 사용할 수 있는 열 형식은 SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE 및 BOOLEAN입니다.
- DECIMAL, DATE, TIME, TIMETZ, TIMESTAMP, TIMESTAMPTZ, GEOMETRY, GEOGRAPHY, HLLSKETCH, SUPER 또는 VARBYTE는 대상 열 형식으로 사용할 수 없습니다.
- 모델 정확도를 높이려면 다음 중 하나를 수행합니다.
  - FROM 절에 훈련 데이터를 지정할 때 CREATE MODEL 명령에 가능한 한 많은 관련 열을 추가합니다.
  - MAX_RUNTIME 및 MAX_CELLS에 더 큰 값을 사용합니다. 이 파라미터의 값이 클수록 모델 학습 비용이 증가합니다.
- 훈련 데이터가 계산되어 Amazon S3 버킷으로 내보내는 즉시 CREATE MODEL 문 실행이 돌아옵니다. 그 이후에는 SHOW MODEL 명령을 사용하여 훈련 상태를 확인할 수 있습니다. 백그라운드에서 훈련 중인 모델이 실패하면 SHOW MODEL을 사용하여 오류를 확인할 수 있습니다. 실패한 모델은 재시도할 수 없습니다. DROP MODEL을 사용하여 실패한 모델을 제거하고 새 모델을 다시 생성합니다. SHOW MODEL에 대한 자세한 내용은 SHOW MODEL 섹션을 참조하세요.
- 로컬 BYOM은 Amazon Redshift 기계 학습에서 BYOM이 아닌 경우에 지원하는 것과 동일한 종류의 모델을 지원합니다. Amazon Redshift는 일반 XGBoost(XGBoost 버전 1.0 이상 사용), 프리프로세서가 없는 KMEANS 모델 및 Amazon SageMaker AI Autopilot에서 훈련된 XGBOOST/MLP/Linear Learner 모델을 지원합니다. Amazon SageMaker AI Neo에서도 지원하는 Autopilot이 지정한 전처리기로 후자를 지원합니다.
- Amazon Redshift 클러스터에서 가상 프라이빗 클라우드(VPC)에 향상된 라우팅을 사용하는 경우 클러스터가 있는 VPC에 대해 Amazon S3 VPC 엔드포인트와 SageMaker AI VPC 엔드포인트를 생성해야 합니다. 이렇게 하면 CREATE MODEL을 수행하는 동안 이러한 서비스 간에 VPC를 통해 트래픽을 실행할 수 있습니다. 자세한 내용은 SageMaker AI Clarify Job Amazon VPC Subnets and Security Groups을 참조하세요.

사용 사례

다음 사용 사례는 필요에 맞게 CREATE MODEL을 사용하는 방법을 보여줍니다.

단순 CREATE MODEL

다음은 CREATE MODEL 구문의 기본 옵션을 요약한 것입니다.

단순 CREATE MODEL 구문

```
CREATE MODEL model_name
FROM { table_name | ( select_query ) }
TARGET column_name
FUNCTION prediction_function_name
IAM_ROLE { default }
SETTINGS (
  S3_BUCKET 'amzn-s3-demo-bucket',
  [ MAX_CELLS integer ]
)
```

단순 CREATE MODEL 파라미터

model_name

모델의 이름입니다. 스키마의 모델 이름은 고유해야 합니다.

FROM { table_name | (select_query) }

table_name 또는 훈련 데이터를 지정하는 쿼리입니다. 시스템의 기존 테이블이거나 괄호로 묶인 Amazon Redshift 호환 SELECT 쿼리, 즉 ()일 수 있습니다. 쿼리 결과에는 2개 이상의 열이 있어야 합니다.

TARGET column_name

예측 대상이 되는 열의 이름입니다. FROM 절에 열이 있어야 합니다.

FUNCTION prediction_function_name

CREATE MODEL에서 생성하고 이 모델을 사용하여 예측하는 데 사용할 Amazon Redshift 기계 학습 함수의 이름을 지정하는 값입니다. 이 함수는 모델 객체와 동일한 스키마에 생성되며 오버로드될 수 있습니다.

Amazon Redshift 기계 학습은 회귀 및 분류를 위한 XGBoost(Xtreme Gradient Boosted tree) 모델과 같은 모델을 지원합니다.

IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }

기본 키워드를 사용하여 CREAT MODEL 명령이 실행될 때 Amazon Redshift에서 기본값으로 설정되고 클러스터와 연결된 IAM 역할을 사용하도록 합니다. 또는 IAM 역할의 ARN을 지정하여 해당 역할을 사용할 수도 있습니다.

S3_BUCKET 'amzn-s3-demo-bucket'

이전에 만든 Amazon S3 버킷의 이름은 Amazon Redshift와 SageMaker AI 간에 훈련 데이터와 아티팩트를 공유하는 데 사용됩니다. Amazon Redshift는 훈련 데이터를 언로드하기 전에 이 버킷에 하위 폴더를 생성합니다. 훈련이 완료되면 Amazon Redshift는 생성된 하위 폴더와 해당 콘텐츠를 삭제합니다.

MAX_CELLS 정수

FROM 절에서 내보낼 최대 셀 수입니다. 기본값은 1,000,000입니다.

셀 수는 훈련 데이터의 행 수(FROM 절 테이블 또는 쿼리에 의해 생성됨)에 열 수를 곱한 값입니다. 훈련 데이터의 셀 수가 max_cells 파라미터에 지정된 것보다 많은 경우 CREATE MODEL은 FROM 절 훈련 데이터를 다운샘플링하여 훈련 집합의 크기를 MAX_CELLS 미만으로 줄입니다. 더 큰 훈련 데이터 집합을 허용하면 정확도를 높일 수 있지만 모델을 훈련하는 데 더 많은 시간과 비용이 더 많이 들 수 있습니다.

Amazon Redshift 사용 비용에 대한 자세한 내용은 Amazon Redshift 기계 학습 사용 비용 섹션을 참조하세요.

다양한 셀 번호와 관련된 비용 및 무료 평가판 세부 정보에 대한 자세한 내용은 Amazon Redshift 요금 섹션을 참조하세요.

사용자 안내에 따라 CREATE MODEL

다음으로 단순 CREATE MODEL에 설명된 옵션 외에 CREATE MODEL에 대한 옵션에 대한 설명을 찾아볼 수 있습니다.

기본적으로 CREATE MODEL은 특정 데이터 집합에 대한 최적의 전처리 및 모델 조합을 검색합니다. 추가 제어를 원하거나 모델에 대한 추가 도메인 지식(예: 문제 유형 또는 목표)을 도입할 수 있습니다. 고객 이탈 시나리오에서 "고객이 활동하고 있지 않음"이라는 결과가 드물다면 F1 목표가 정확도 목표보다 선호되는 경우가 많습니다. 높은 정확도 모델은 항상 "고객이 활동하고 있음"이라고 예측할 수 있기 때문에 정확도는 높지만 비즈니스 가치는 거의 없습니다. F1 목표에 대한 자세한 내용은 Amazon SageMaker AI API Reference의 AutoMLJobObjective 섹션을 참조하세요.

그런 다음 CREATE MODEL은 목표와 같은 지정된 측면에 대한 제안을 따릅니다. 동시에 CREATE MODEL은 최고의 전처리기와 최고의 하이퍼파라미터를 자동으로 검색합니다.

사용자 안내 구문으로 CREATE MODEL

CREATE MODEL은 지정할 수 있는 측면과 Amazon Redshift가 자동으로 검색하는 측면에서 더 많은 유연성을 제공합니다.

```
CREATE MODEL model_name
FROM { table_name | ( select_statement ) }
TARGET column_name
FUNCTION function_name
IAM_ROLE { default }
[ MODEL_TYPE { XGBOOST | MLP | LINEAR_LEARNER} ]
[ PROBLEM_TYPE ( REGRESSION | BINARY_CLASSIFICATION | MULTICLASS_CLASSIFICATION ) ]
[ OBJECTIVE ( 'MSE' | 'Accuracy' | 'F1' | 'F1Macro' | 'AUC') ]
SETTINGS (
  S3_BUCKET 'amzn-s3-demo-bucket', |
  S3_GARBAGE_COLLECT { ON | OFF }, |
  KMS_KEY_ID 'kms_key_id', |
  MAX_CELLS integer, |
  MAX_RUNTIME integer (, ...)
)
```

사용자 안내 파라미터로 CREATE MODEL

MODEL_TYPE { XGBOOST | MLP | LINEAR_LEARNER }

(옵션) 모델 유형을 지정합니다. XGBoost, 다층 퍼셉트론(MLP) 또는 Linear Learner와 같은 특정 모델 유형의 모델을 교육할지 여부를 지정할 수 있습니다. 이 모델은 모두 Amazon SageMaker AI Autopilot에서 지원하는 알고리즘입니다. 파라미터를 지정하지 않으면 지원되는 모든 모델 유형이 훈련 중 최상의 모델을 찾기 위해 검색됩니다.

PROBLEM_TYPE (REGRESSION | BINARY_CLASSIFICATION | MULTICLASS_CLASSIFICATION)

(옵션) 문제 유형을 지정합니다. 문제 유형을 알고 있는 경우 해당 특정 모델 유형의 최상의 모델만 검색하도록 Amazon Redshift를 제한할 수 있습니다. 이 파라미터를 지정하지 않으면 데이터를 기반으로 훈련 중 문제 유형이 검색됩니다.

OBJECTIVE ('MSE' | 'Accuracy' | 'F1' | 'F1Macro' | 'AUC')

(옵션) 기계 학습 시스템의 예측 품질을 측정하는 데 사용되는 목표 지표의 이름을 지정합니다. 이 지표는 데이터의 모델 파라미터 값에 대한 최상의 추정치를 제공하기 위해 훈련 중에 최적화됩니다. 지표를 명시적으로 지정하지 않은 경우 기본 동작은 MSE(회귀 분석의 경우), F1(이진 분류의 경우), 정확도(다중 클래스 분류의 경우)를 자동으로 사용하는 것입니다. 목표에 대한 자세한 내용은 Amazon SageMaker AI API Reference의 AutoMLJobObjective 섹션을 참조하세요.

MAX_CELLS 정수

(옵션) 훈련 데이터의 셀 수를 지정합니다. 이 값은 레코드 수(훈련 쿼리 또는 테이블)에 열 수를 곱한 값입니다. 기본값은 1,000,000입니다.

MAX_RUNTIME 정수

(옵션) 훈련할 최대 시간을 지정합니다. 데이터 집합 크기에 따라 훈련 작업이 더 빨리 완료되는 경우가 많습니다. 이는 훈련에 소요되는 최대 시간을 지정합니다. 기본값은 5,400(90분)입니다.

S3_GARBAGE_COLLECT { ON | OFF }

(옵션) Amazon Redshift가 모델 훈련에 사용되는 결과 데이터 집합에 대해 가비지 수집을 수행할지 여부를 지정합니다. OFF로 설정하면 모델 훈련에 사용된 결과 데이터 집합과 모델이 Amazon S3에 남아 다른 용도로 사용할 수 있습니다. ON으로 설정하면 훈련이 완료된 후 Amazon Redshift가 Amazon S3에서 아티팩트를 삭제합니다. 기본값은 ON입니다.

KMS_KEY_ID 'kms_key_id'

(옵션) Amazon Redshift가 AWS KMS 키로 서버 측 암호화를 사용하여 저장된 데이터를 보호할지 여부를 지정합니다. 전송 중인 데이터는 보안 소켓 계층(SSL)으로 보호됩니다.

PREPROCESSORS 'string'

(옵션) 특정 열 집합에 대한 전처리기의 특정 조합을 지정합니다. 형식은 columnSet의 목록과 각 열 집합에 적용할 적절한 변환입니다. Amazon Redshift는 특정 변환기 목록의 모든 변환기를 해당 ColumnSet의 모든 열에 적용합니다. 예를 들어 Imputer가 있는 OneHotEncoder를 열 t1 및 t2에 적용하려면 다음 샘플 명령을 사용합니다.

```
CREATE MODEL customer_churn
FROM customer_data
TARGET 'Churn'
FUNCTION predict_churn
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
PROBLEM_TYPE BINARY_CLASSIFICATION
OBJECTIVE 'F1'
PREPROCESSORS '[
```

```
...
{"ColumnSet": [
    "t1",
    "t2"
  ],
  "Transformers": [
    "OneHotEncoder",
    "Imputer"
  ]
},
{"ColumnSet": [
    "t3"
  ],
  "Transformers": [
    "OneHotEncoder"
  ]
},
{"ColumnSet": [
    "temp"
  ],
  "Transformers": [
    "Imputer",
    "NumericPassthrough"
  ]
}
]'
SETTINGS (
S3_BUCKET 'amzn-s3-demo-bucket'
)
```

Amazon Redshift는 다음 변환기를 지원합니다.

- OneHotEncoder - 일반적으로 이산 값을 하나의 0이 아닌 값이 있는 이진 벡터로 인코딩하는 데 사용됩니다. 이 변환기는 많은 기계 학습 모델에 적합합니다.
- OrdinalEncoder – 이산 값을 단일 정수로 인코딩합니다. 이 변환기는 MLP 및 Linear Learner와 같은 특정 기계 학습 모델에 적합합니다.
- NumericPassthrough – 입력을 있는 그대로 모델에 전달합니다.
- Imputer – 누락된 값과 NaN(Not a Number) 값을 채웁니다.
- ImputerWithIndicator – 누락된 값과 NaN 값을 채웁니다. 또한 이 변환기는 누락되어 채워진 값이 있는지 여부에 대한 표시기를 생성합니다.

- Normalizer – 많은 기계 학습 알고리즘의 성능을 향상시킬 수 있는 값을 정규화합니다.
- DateTimeVectorizer - 기계 학습 모델에서 사용할 수 있는 날짜/시간 데이터 형식의 열을 나타내는 벡터 임베딩을 생성합니다.
- PCA - 가능한 한 많은 정보를 유지하면서 특성 수를 줄이기 위해 데이터를 저차원 공간에 표시합니다.
- StandardScaler – 평균을 제거하고 단위 분산에 맞게 조정하여 특성을 표준화합니다.
- MinMax – 각 특성을 지정된 범위로 확장하여 특성을 변환합니다.

Amazon Redshift 기계 학습은 훈련된 변환기를 저장하고 예측 쿼리의 일부로 자동 적용합니다. 모델에서 예측을 생성할 때는 변환기를 지정할 필요가 없습니다.

AUTO OFF로 CREATE XGBoost 모델

AUTO OFF CREATE MODEL은 일반적으로 기본 CREATE MODEL과 다른 목표를 가지고 있습니다.

원하는 모델 유형과 이러한 모델을 훈련할때 사용할 하이퍼파라미터를 이미 알고 있는 고급 사용자는 AUTO OFF와 함께 CREATE MODEL을 사용하여 전처리기 및 하이퍼파라미터의 CREATE MODEL 자동 검색을 해제할 수 있습니다. 이를 위해서는 모델 유형을 명시적으로 지정합니다. XGBoost는 현재 AUTO가 OFF로 설정된 경우 지원되는 유일한 모델 유형입니다. 하이퍼파라미터를 지정할 수 있습니다. Amazon Redshift는 지정한 하이퍼파라미터에 대해 기본값을 사용합니다.

AUTO OFF 구문이 있는 CREATE XGBoost 모델

```
CREATE MODEL model_name
FROM { table_name | (select_statement ) }
TARGET column_name
FUNCTION function_name
IAM_ROLE { default }
AUTO OFF
MODEL_TYPE XGBOOST
OBJECTIVE { 'reg:squarederror' | 'reg:squaredlogerror' | 'reg:logistic' |
            'reg:pseudohubererror' | 'reg:tweedie' | 'binary:logistic' | 'binary:hinge'
 |
            'multi:softmax' | 'rank:pairwise' | 'rank:ndcg' }
HYPERPARAMETERS DEFAULT EXCEPT (
    NUM_ROUND '10',
    ETA '0.2',
    NUM_CLASS '10',
    (, ...)
)
```

```
PREPROCESSORS 'none'
SETTINGS (
  S3_BUCKET 'amzn-s3-demo-bucket', |
  S3_GARBAGE_COLLECT { ON | OFF }, |
  KMS_KEY_ID 'kms_key_id', |
  MAX_CELLS integer, |
  MAX_RUNTIME integer (, ...)
)
```

AUTO OFF 파라미터로 CREATE XGBoost 모델

AUTO OFF

전처리기, 알고리즘 및 하이퍼파라미터 선택의 CREATE MODEL 자동 검색을 해제합니다.

MODEL_TYPE XGBOOST

XGBOOST를 사용하여 모델을 훈련하도록 지정합니다.

OBJECTIVE str

알고리즘에서 인식하는 목표를 지정합니다. Amazon Redshift는 reg:squarederror, reg:squaredlogerror, reg:logistic, reg:pseudohubererror, reg:tweedie, binary:logistic, binary:hinge, multi:softmax를 지원합니다. 이러한 목표에 대한 자세한 내용은 XGBoost 설명서의 Learning task parameters 섹션을 참조하세요.

HYPERPARAMETERS { DEFAULT | DEFAULT EXCEPT (key ‘value’ (,..)) }

기본 XGBoost 파라미터가 사용되는지 아니면 사용자 지정 값으로 재정의되는지를 지정합니다. 작은따옴표로 값을 묶어야 합니다. 다음은 XGBoost에 대한 파라미터와 해당 기본값의 예입니다.

파라미터 이름	파라미터값	기본값	참고
num_class	Integer	다중 클래스 분류에 필요합니다.	N/A

파라미터 이름	파라미터값	기본값	참고
num_round	Integer	100	N/A
tree_method	String	자동	N/A
max_depth	Integer	6	[0 , 10]
min_child_weight	Float	1	MinValue: 0, MaxValue: 120
subsample	Float	1	MinValue: 0.5, MaxValue: 1
gamma	Float	0	MinValue: 0, MaxValue: 5
alpha	Float	0	MinValue: 0, MaxValue: 1000
eta	Float	0.3	MinValue: 0.1, MaxValue: 0.5
colsample_byleve	Float	1	MinValue: 0.1, MaxValue: 1
colsample_bynode	Float	1	MinValue: 0.1, MaxValue: 1
colsample_bytree	Float	1	MinValue: 0.5, MaxValue: 1
lambda	Float	1	MinValue: 0, MaxValue: 1000
max_delta_step	Integer	0	[0, 10]

다음 예에서는 XGBoost용 데이터를 준비합니다.

```
DROP TABLE IF EXISTS abalone_xgb;

CREATE TABLE abalone_xgb (
```

```
length_val float,
diameter float,
height float,
whole_weight float,
shucked_weight float,
viscera_weight float,
shell_weight float,
rings int,
record_number int);

COPY abalone_xgb
FROM 's3://redshift-downloads/redshift-ml/abalone_xg/'
REGION 'us-east-1'
IAM_ROLE default
IGNOREHEADER 1 CSV;
```

다음 예에서는 MODEL_TYPE, OBJECTIVE 및 PREPROCESSORS와 같은 고급 옵션이 지정된 XGBoost 모델을 생성합니다.

```
DROP MODEL abalone_xgboost_multi_predict_age;

CREATE MODEL abalone_xgboost_multi_predict_age
FROM ( SELECT length_val,
              diameter,
              height,
              whole_weight,
              shucked_weight,
              viscera_weight,
              shell_weight,
              rings
   FROM abalone_xgb WHERE record_number < 2500 )
TARGET rings FUNCTION ml_fn_abalone_xgboost_multi_predict_age
IAM_ROLE default
AUTO OFF
MODEL_TYPE XGBOOST
OBJECTIVE 'multi:softmax'
PREPROCESSORS 'none'
HYPERPARAMETERS DEFAULT EXCEPT (NUM_ROUND '100', NUM_CLASS '30')
SETTINGS (S3_BUCKET 'amzn-s3-demo-bucket');
```

다음 예에서는 추론 쿼리를 사용하여 레코드 번호가 2,500보다 큰 물고기의 나이를 예측합니다. 위의 명령에서 생성된 ml_fn_abalone_xgboost_multi_predict_age 함수가 사용됩니다.

```
select ml_fn_abalone_xgboost_multi_predict_age(length_val,
                                               diameter,
                                               height,
                                               whole_weight,
                                               shucked_weight,
                                               viscera_weight,
                                               shell_weight)+1.5 as age
from abalone_xgb where record_number > 2500;
```

기존 보유 모델 사용(BYOM) - 로컬 추론

Amazon Redshift 기계 학습은 로컬 추론에서 기존 보유 모델 사용(BYOM)을 지원합니다.

다음은 BYOM용 CREATE MODEL 구문에 대한 옵션을 요약한 것입니다. Amazon Redshift에서 로컬로 데이터베이스 내 추론을 위해 Amazon SageMaker AI와 함께 Amazon Redshift 외부에서 훈련된 모델을 사용할 수 있습니다.

로컬 추론을 위한 CREATE MODEL 구문

다음은 로컬 추론을 위한 CREATE MODEL 구문에 대한 설명입니다.

```
CREATE MODEL model_name
FROM ('job_name' | 's3_path' )
FUNCTION function_name ( data_type [, ...] )
RETURNS data_type
IAM_ROLE { default }
[ SETTINGS (
  S3_BUCKET 'amzn-s3-demo-bucket', | --required
  KMS_KEY_ID 'kms_string') --optional
];
```

Amazon Redshift는 현재 BYOM용으로 사전 훈련된 XGBoost, MLP 및 Linear Learner 모델만 지원합니다. 이 경로를 사용하여 로컬 추론을 위해 Amazon SageMaker AI에서 직접 훈련된 SageMaker AI Autopilot과 모델을 가져올 수 있습니다.

로컬 추론을 위한 CREATE MODEL 파라미터

model_name

모델의 이름입니다. 스키마의 모델 이름은 고유해야 합니다.

FROM ('job_name' | 's3_path')

job_name은 Amazon SageMaker AI 작업 이름을 입력으로 사용합니다. 작업 이름은 Amazon SageMaker AI 훈련 작업 이름 또는 Amazon SageMaker AI Autopilot 작업 이름일 수 있습니다. 작업은 Amazon Redshift 클러스터를 소유한 동일한 AWS 계정에서 생성되어야 합니다. 작업 이름을 찾으려면 Amazon SageMaker AI를 시작합니다. 훈련(Training) 드롭다운 메뉴에서 훈련 작업(Training jobs)을 선택합니다.

's3_path'는 모델을 생성할 때 사용할 .tar.gz 모델 아티팩트 파일의 S3 위치를 지정합니다.

FUNCTION function_name (data_type [, ...])

생성할 함수의 이름과 입력 인수의 데이터 형식입니다. 스키마 이름을 제공할 수 있습니다.

RETURNS data_type

함수에 의해 반환되는 값의 데이터 형식입니다.

IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>'}

기본 키워드를 사용하여 CREATE MODEL 명령이 실행될 때 Amazon Redshift에서 기본값으로 설정되고 클러스터와 연결된 IAM 역할을 사용하도록 합니다.

클러스터가 인증 및 권한 부여에 사용하는 IAM 역할의 Amazon 리소스 이름(ARN)을 사용합니다.

SETTINGS (S3_BUCKET 'amzn-s3-demo-bucket', | KMS_KEY_ID 'kms_string')

S3_BUCKET 절은 중간 결과를 저장하는 데 사용되는 Amazon S3 위치를 지정합니다.

(옵션) KMS_KEY_ID 절은 Amazon Redshift가 AWS KMS 키로 서버 측 암호화를 사용하여 저장된 데이터를 보호할지 여부를 지정합니다. 전송 중인 데이터는 보안 소켓 계층(SSL)으로 보호됩니다.

자세한 내용은 사용자 안내에 따라 CREATE MODEL 단원을 참조하십시오.

로컬 추론을 위한 CREATE MODEL 예

다음 예에서는 Amazon Redshift 외부의 Amazon SageMaker AI에서 이전에 훈련된 모델을 생성합니다. 모델 유형은 로컬 추론을 위해 Amazon Redshift 기계 학습에서 지원되므로 다음 CREATE MODEL은 Amazon Redshift에서 로컬로 사용할 수 있는 함수를 생성합니다. SageMaker AI 훈련 작업 이름을 제공할 수 있습니다.

```
CREATE MODEL customer_churn
FROM 'training-job-customer-churn-v4'
```

```
FUNCTION customer_churn_predict (varchar, int, float, float)
RETURNS int
IAM_ROLE default
SETTINGS (S3_BUCKET 'amzn-s3-demo-bucket');
```

모델이 생성된 후 지정된 인수 형식과 함께 customer_churn_predict 함수를 사용하여 예측할 수 있습니다.

기존 보유 모델 사용(BYOM) - 원격 추론

또한 Amazon Redshift 기계 학습은 원격 추론에서도 기존 보유 모델 사용(BYOM)을 지원합니다.

다음은 BYOM용 CREATE MODEL 구문에 대한 옵션을 요약한 것입니다.

원격 추론을 위한 CREATE MODEL 구문

다음은 원격 추론을 위한 CREATE MODEL 구문에 대한 설명입니다.

```
CREATE MODEL model_name
FUNCTION function_name ( data_type [, ...] )
RETURNS data_type
SAGEMAKER 'endpoint_name'['model_name']
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
[SETTINGS (MAX_BATCH_ROWS integer)];
```

원격 추론을 위한 CREATE MODEL 파라미터

model_name

모델의 이름입니다. 스키마의 모델 이름은 고유해야 합니다.

FUNCTION fn_name ([data_type] [, ...])

함수의 이름과 입력 인수의 데이터 형식입니다. 지원되는 모든 데이터 유형을 보려면 데이터 유형을 참조하세요. Geography, geometry 및 hllsketch는 지원되지 않습니다.

myschema.myfunction과 같이 두 부분으로 구성된 표기법을 사용하여 스키마 내에 함수 이름을 제공할 수도 있습니다.

RETURNS data_type

함수에 의해 반환되는 값의 데이터 형식입니다. 지원되는 모든 데이터 유형을 보려면 데이터 유형을 참조하세요. Geography, geometry 및 hllsketch는 지원되지 않습니다.

SAGEMAKER 'endpoint_name'[:'model_name']

Amazon SageMaker AI 엔드포인트의 이름입니다. 엔드포인트 이름이 다중 모델 엔드포인트를 가리키는 경우 사용할 모델의 이름을 추가합니다. 엔드포인트는 Amazon Redshift 클러스터와 동일한 AWS 리전 및 AWS 계정에서 호스팅되어야 합니다. 엔드포인트를 찾으려면 Amazon SageMaker AI를 시작합니다. 추론(Inference) 드롭다운 메뉴에서 엔드포인트(Endpoints)를 선택합니다.

IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>'}

기본 키워드를 사용하여 CREATE MODEL 명령이 실행될 때 Amazon Redshift에서 기본값으로 설정되고 클러스터와 연결된 IAM 역할을 사용하도록 합니다. 또는 IAM 역할의 ARN을 지정하여 해당 역할을 사용할 수도 있습니다.

MAX_BATCH_ROWS 정수

Amazon Redshift가 단일 SageMaker AI 간접 호출에 대한 단일 배치 요청으로 보내는 최대 행 수입니다. 원격 추론 기능이 있는 BYOM에만 지원됩니다. 배치의 실제 행 수는 입력 크기에 따라 다르지만 이 값보다 작거나 같아야 합니다. 이 파라미터의 최소값은 1입니다. 최대값은 `INT_MAX` 또는 2,147,483,647입니다. 이 파라미터는 입력 및 반환 데이터 형식이 모두 `SUPER`인 경우에만 필요합니다. 기본값은 `INT_MAX` 또는 2,147,483,647입니다.

모델이 SageMaker AI 엔드포인트에 배포되면 SageMaker AI는 Amazon Redshift에 모델 정보를 생성합니다. 그런 다음 외부 기능을 통해 추론을 수행합니다. SHOW MODEL 명령을 사용하여 Amazon Redshift 클러스터의 모델 정보를 볼 수 있습니다.

원격 추론을 위한 CREATE MODEL 사용 노트

원격 추론을 위해 CREATE MODEL을 사용하기 전에 다음 사항을 고려하세요.

- 엔드포인트는 Amazon Redshift 클러스터를 소유한 동일한 AWS 계정에서 호스팅되어야 합니다.
- Amazon SageMaker AI 엔드포인트에 Amazon Redshift의 추론 직접 호출을 수용할 수 있는 충분한 리소스가 있는지 또는 Amazon SageMaker AI 엔드포인트가 자동으로 확장될 수 있는지 확인합니다.
- `SUPER` 데이터 형식을 입력으로 사용하지 않는 경우 모델은 쉼표로 구분된 값(CSV) 형식의 입력만 허용하며, 이는 SageMaker AI의 `text/CSV` 콘텐츠 유형에 해당합니다.
- `SUPER` 데이터 형식을 입력으로 사용하지 않는 경우 모델의 출력은 함수를 만들 때 지정한 유형의 단일 값입니다. 출력은 쉼표로 구분된 값(CSV) 형식의 `text/CSV` 콘텐츠 유형을 통해 SageMaker AI에서 이루어집니다. `VARCHAR` 데이터 형식은 따옴표로 묶을 수 없고 새 줄을 포함할 수 없으며 각 출력은 새 줄에 있어야 합니다.

- 모델은 null을 빈 문자열로 수락합니다.
- 입력 데이터 형식이 SUPER인 경우 하나의 입력 인수만 지원됩니다.
- 입력 데이터 형식이 SUPER인 경우 반환되는 데이터 형식도 SUPER여야 합니다.
- 입력 및 반환된 데이터 형식이 모두 SUPER인 경우 MAX_BATCH_ROWS가 필요합니다.
- 입력 데이터 형식이 SUPER인 경우 엔드포인트 호출의 콘텐츠 유형은 MAX_BATCH_ROWS가 `application/json`인 경우 1, 그 외의 모든 경우에는 `application/jsonlines`입니다.
- 반환 데이터 형식이 SUPER인 경우 엔드포인트 호출의 수락 유형은 MAX_BATCH_ROWS가 `application/json`인 경우 1, 그 외의 모든 경우에는 `application/jsonlines`입니다.

원격 추론을 위한 CREATE MODEL 예

다음 예에서는 SageMaker AI 엔드포인트를 사용하여 예측하는 모델을 생성합니다. 예측을 수행하고 CREATE MODEL 명령에서 해당 이름을 지정하기 위해 엔드포인트가 실행 중인지 확인합니다.

```
CREATE MODEL remote_customer_churn
FUNCTION remote_fn_customer_churn_predict (varchar, int, float, float)
RETURNS int
SAGEMAKER 'customer-churn-endpoint'
IAM_ROLE default;
```

다음 예는 대규모 언어 모델 모델(LLM)을 사용하여 원격 추론으로 BYOM을 생성하는 예제입니다. Amazon SageMaker AI Jumpstart에서 호스팅되는 LLM은 `application/json` 콘텐츠 유형을 수락하고 반환하며 간접 호출당 단일 JSON을 지원합니다. 입력 및 반환 데이터 형식은 SUPER여야 하며 MAX_BATCH_ROWS는 1로 설정해야 합니다.

```
CREATE MODEL sample_super_data_model
FUNCTION sample_super_data_model_predict(super)
RETURNS super
SAGEMAKER 'sample_super_data_model_endpoint'
IAM_ROLE default
SETTINGS (MAX_BATCH_ROWS 1);
```

K-MEANS를 사용한 CREATE MODEL

Amazon Redshift는 레이블이 지정되지 않은 데이터를 그룹화하는 K-Means 알고리즘을 지원합니다. 이 알고리즘은 데이터에서 그룹을 검색하려는 클러스터링 문제를 해결합니다. 분류되지 않은 데이터는 유사점과 차이점에 따라 그룹화되고 분할됩니다.

K-MEANS 구문을 사용한 CREATE MODEL

```
CREATE MODEL model_name
FROM { table_name | ( select_statement ) }
FUNCTION function_name
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
AUTO OFF
MODEL_TYPE KMEANS
PREPROCESSORS 'string'
HYPERPARAMETERS DEFAULT EXCEPT ( K 'val' [, ...] )
SETTINGS (
  S3_BUCKET 'amzn-s3-demo-bucket',
  KMS_KEY_ID 'kms_string', |
    -- optional
  S3_GARBAGE_COLLECT on / off, |
    -- optional
  MAX_CELLS integer, |
    -- optional
  MAX_RUNTIME integer
    -- optional);
```

K-MEANS 파라미터를 사용한 CREATE MODEL

AUTO OFF

전처리기, 알고리즘 및 하이퍼파라미터 선택의 CREATE MODEL 자동 검색을 해제합니다.

MODEL_TYPE KMEANS

KMEANS를 사용하여 모델을 훈련하도록 지정합니다.

PREPROCESSORS 'string'

특정 열 집합에 대한 프로프로세서의 특정 조합을 지정합니다. 형식은 columnSet의 목록과 각 열 집합에 적용할 적절한 변환입니다. Amazon Redshift는 3개의 K-Means 프리프로세서, 즉 StandardScaler, MinMax 및 NumericPassthrough를 지원합니다. K-Means에 대한 사전 처리를 적용하지 않으려면 변환기로 명시적으로 NumericPassthrough를 선택합니다. 지원되는 변환기에 대한 자세한 내용은 사용자 안내 파라미터로 CREATE MODEL 섹션을 참조하세요.

K-Means 알고리즘은 유클리드 거리를 사용하여 유사성을 계산합니다. 데이터 사전 처리는 모델의 특성이 동일한 규모로 유지되고 신뢰할 수 있는 결과를 생성하도록 보장합니다.

HYPERPARAMETERS DEFAULT EXCEPT (K 'val' [, ...])

K-Means 파라미터의 사용 여부를 지정합니다. K-Means 알고리즘을 사용할 때는 K 파라미터를 지정해야 합니다. 자세한 내용은 Amazon SageMaker AI Developer Guide의 K-Means Hyperparameters를 참조하세요.

다음 예에서는 K-Means용 데이터를 준비합니다.

```
CREATE MODEL customers_clusters
FROM customers
FUNCTION customers_cluster
IAM_ROLE default
AUTO OFF
MODEL_TYPE KMEANS
PREPROCESSORS '[
{
  "ColumnSet": [ "*" ],
  "Transformers": [ "NumericPassthrough" ]
}
]'
HYPERPARAMETERS DEFAULT EXCEPT ( K '5' )
SETTINGS (S3_BUCKET 'amzn-s3-demo-bucket');

select customer_id, customers_cluster(...) from customers;
customer_id | customers_cluster
--------------------
12345            1
12346            2
12347            4
12348
```

예측이 포함된 CREATE MODEL

Redshift ML의 예측 모델은 Amazon Forecast를 사용하여 정확한 시계열 예측을 생성합니다. 이렇게 하면 일정 기간 동안의 과거 데이터를 사용하여 향후 이벤트를 예측할 수 있습니다. Amazon Forecast의 일반적인 사용 사례에는 소매 제품 데이터를 사용하여 재고 가격 책정 방법을 결정하고, 제조 수량 데이터를 사용하여 주문할 품목의 양을 예측하고, 웹 트래픽 데이터를 사용하여 웹 서버에 수신될 수 있는 트래픽 양을 예측하는 것이 포함됩니다.

Amazon Forecast의 할당량 한도는 Amazon Redshift 예측 모델에 적용됩니다. 예를 들어 최대 예측 수는 100개이지만 조정 가능합니다. 예측 모델을 삭제해도 Amazon Forecast의 관련 리소스가 자동으로 삭제되지는 않습니다. Redshift 클러스터를 삭제하면 관련 모델도 모두 삭제됩니다.

Forecast 모델은 현재 다음 리전에서만 사용할 수 있습니다.

- 미국 동부(오하이오)(us-east-2)
- 미국 동부(버지니아 북부)(us-east-1)
- 미국 서부(오레곤)(us-west-2)
- 아시아 태평양(뭄바이)(ap-south-1)
- 아시아 태평양(서울)(ap-northeast-2)
- 아시아 태평양(싱가포르)(ap-southeast-1)
- 아시아 태평양(시드니)(ap-southeast-2)
- 아시아 태평양(도쿄)(ap-northeast-1)
- 유럽(프랑크푸르트)(eu-central-1)
- 유럽(아일랜드)(eu-west-1)

예측 구문이 포함된 CREATE MODEL

```
CREATE [ OR REPLACE ] MODEL forecast_model_name
FROM { table_name | ( select_query ) }
TARGET column_name
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>'}
AUTO ON
MODEL_TYPE FORECAST
SETTINGS (
  S3_BUCKET 'amzn-s3-demo-bucket',
  HORIZON integer,
  FREQUENCY forecast_frequency
  [PERCENTILES '0.1', '0.5', '0.9']
  )
```

예측 파라미터가 포함된 CREATE MODEL

forecast_model_name

모델의 이름입니다. 모델 이름은 고유해야 합니다.

FROM { table_name | (select_query) }

table_name 또는 훈련 데이터를 지정하는 쿼리입니다. 이는 시스템의 기존 테이블이거나 괄호로 묶인 Amazon Redshift 호환 SELECT 쿼리일 수 있습니다. 테이블 또는 쿼리 결과에는 다음과 같이 3개 이상의 열이 있어야 합니다. (1) 시계열 이름을 지정하는 varchar 열. 각 데이터 세트에는 여러 시계열이 있음, (2) 날짜/시간 열, (3) 예측할 대상 열. 이 대상 열은 int 또는 float여야 함. 3개 이상의 열이 있는 데이터 세트를 제공하는 경우 Amazon Redshift는 모든 추가 열이 관련 시계열의 일부라고 가정합니다. 참고로 관련 시계열은 int 또는 float 유형이어야 합니다. 관련 시계열에 대한 자세한 내용은 관련 시계열 데이터 세트 사용을 참조하세요.

TARGET column_name

예측 대상이 되는 열의 이름입니다. FROM 절에 열이 있어야 합니다.

IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }

기본 키워드를 사용하여 CREAT MODEL 명령이 실행될 때 Amazon Redshift에서 기본값으로 설정되고 클러스터와 연결된 IAM 역할을 사용하도록 합니다. 또는 IAM 역할의 ARN을 지정하여 해당 역할을 사용할 수도 있습니다.

AUTO ON

알고리즘 및 하이퍼파라미터 선택의 CREATE MODEL 자동 검색을 켭니다. 예측 모델을 생성할 때 on을 지정하면 Amazon Forecast가 데이터 세트의 각 시계열에 최적의 알고리즘 조합을 적용하는 Forecast AutoPredictor를 사용한다는 뜻입니다.

MODEL_TYPE FORECAST

FORECAST를 사용하여 모델을 훈련하도록 지정합니다.

S3_BUCKET 'amzn-s3-demo-bucket'

이전에 생성한 Amazon Simple Storage Service 버킷의 이름으로, Amazon Redshift와 Amazon Forecast 간에 훈련 데이터와 아티팩트를 공유하는 데 사용됩니다. Amazon Redshift는 훈련 데이터를 언로드하기 전에 이 버킷에 하위 폴더를 생성합니다. 훈련이 완료되면 Amazon Redshift는 생성된 하위 폴더와 해당 콘텐츠를 삭제합니다.

HORIZON 정수

예측 모델이 반환할 수 있는 최대 예측 수입니다. 모델을 학습시킨 후에는 이 정수를 변경할 수 없습니다.

FREQUENCY forecast_frequency

예측을 얼마나 세분화하여 설정할지를 지정합니다. 사용 가능한 옵션은 `Y | M | W | D | H | 30min | 15min | 10min | 5min | 1min`입니다. 예측 모델을 학습하는 경우 필요합니다.

PERCENTILES 문자열

예측기를 훈련하는 데 사용되는 예측 유형을 지정하는 쉼표로 구분된 문자열입니다. 예측 유형은 0.01에서 0.99까지의 사분위수(0.01 이상 증분)일 수 있습니다. 평균을 사용하여 평균 예측을 지정할 수도 있습니다. 최대 5개의 예측 유형을 지정할 수 있습니다.

다음 예는 간단한 예측 모델을 만드는 방법을 보여줍니다.

```
CREATE MODEL forecast_example
FROM forecast_electricity_
TARGET target
IAM_ROLE 'arn:aws:iam::<account-id>:role/<role-name>'
AUTO ON
MODEL_TYPE FORECAST
SETTINGS (S3_BUCKET 'amzn-s3-demo-bucket',
          HORIZON 24,
          FREQUENCY 'H',
          PERCENTILES '0.25,0.50,0.75,mean',
          S3_GARBAGE_COLLECT OFF);
```

예측 모델을 생성한 후 예측 데이터가 포함된 새 테이블을 생성할 수 있습니다.

```
CREATE TABLE forecast_model_results as SELECT Forecast(forecast_example)
```

그런 다음 새 테이블을 쿼리하여 예측을 얻을 수 있습니다.

```
SELECT * FROM forecast_model_results
```

CREATE PROCEDURE

새 저장 프로시저를 생성하거나 현재 데이터베이스의 기존 프로시저를 바꿉니다.

자세한 정보와 지침은 Amazon Redshift에서 저장 프로시저 생성 섹션을 참조하세요.

필수 권한

CREATE OR REPLACE PROCEDURE를 실행하려면 다음 방법 중 하나에 대한 권한이 있어야 합니다.

- CREATE PROCEDURE의 경우

- 수퍼유저
- 저장 프로시저가 생성된 스키마에 대한 CREATE 및 USAGE 권한이 있는 사용자

- REPLACE PROCEDURE의 경우
  - 수퍼유저
  - 프로시저 소유자

구문

```
CREATE [ OR REPLACE ] PROCEDURE sp_procedure_name
  ( [ [ argname ] [ argmode ] argtype [, ...] ] )
[ NONATOMIC ]
AS $$
  procedure_body
$$ LANGUAGE plpgsql
[ { SECURITY INVOKER | SECURITY DEFINER } ]
[ SET configuration_parameter { TO value | = value } ]
```

파라미터

OR REPLACE

이름 및 입력 인수 데이터 형식이 같은 프로시저 또는 서명이 같은 프로시저인 경우 이런 프로시저가 이미 하나 존재하므로 기존 프로시저가 대체됨을 지정하는 절입니다. 프로시저는 똑같은 데이터 형식 집합을 정의하는 새 프로시저로만 바꿀 수 있습니다.

기존 프로시저와 이름이 동일하지만 서명이 다른 프로시저를 정의하는 경우, 새 프로시저를 생성합니다. 다시 말해 프로시저 이름이 오버로드됩니다. 자세한 내용은 프로시저 이름 오버로드 단원을 참조하십시오.

sp_procedure_name

프로시저의 이름입니다. 스키마 이름을 지정하는 경우(예: **myschema.myprocedure**), 프로시저는 지정된 스키마에서 생성됩니다. 그렇지 않으면, 프로시저가 현재 스키마에서 생성됩니다. 유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요.

모든 저장 프로시저 이름에 sp_를 접두사로 사용하는 것이 좋습니다. Amazon Redshift는 저장 프로시저 이름용으로 sp_ 접두사를 예약합니다. sp_ 접두사를 사용하여 저장 프로시저 이름이 기존 또는 향수 Amazon Redshift 기본 제공 저장 프로시저 또는 함수 이름과 충돌하지 않도록 할 수 있습니다. 자세한 내용은 저장 프로시저 명명 단원을 참조하십시오.

입력 인수에 대한 데이터 형식 또는 서명이 서로 다른 경우 같은 이름을 가진 프로시저를 2개 이상 정의할 수 있습니다. 다시 말해 이 경우 프로시저 이름이 오버로드됩니다. 자세한 내용은 프로시저 이름 오버로드 섹션을 참조하세요.

[argname] [argmode] argtype

인수 이름, 인수 모드 및 데이터 형식의 목록입니다. 데이터 형식만 필수입니다. 이름과 모드는 선택 사항이고, 위치를 전환할 수 있습니다.

인수 모드는 IN, OUT 또는 INOUT일 수 있습니다. 기본값은 IN입니다.

OUT 및 INOUT 인수를 사용하여 프로시저 호출에서 값을 하나 이상 반환할 수 있습니다. OUT 또는 INOUT 인수가 있는 경우, 프로시저 호출이 n 열이 포함된 결과 행 하나를 반환합니다. 여기에서 n은 OUT 또는 INOUT 인수의 총 수입니다.

INOUT 인수는 동시에 입력 및 출력 인수입니다. 입력 인수에는 IN 인수와 INOUT 인수가 모두 포함되며, 출력 인수에는 OUT 인수와 INOUT 인수가 모두 포함됩니다.

OUT 인수는 CALL 문의 일부로 지정되지 않습니다. 저장 프로시저 CALL 문에서 INOUT 인수를 지정합니다. INOUT 인수는 중첩 호출에서 값을 전달 및 반환하고 `refcursor`를 반환할 때 유용할 수 있습니다. `refcursor` 형식에 대한 자세한 내용은 커서 섹션을 참조하세요.

인수 데이터 형식은 모든 표준 Amazon Redshift 데이터 형식일 수 있습니다. 또한 인수 데이터 형식은 `refcursor`일 수 있습니다.

최대 32개의 입력 인수와 32개의 출력 인수를 지정할 수 있습니다.

AS $$ procedure_body $$

실행할 프로시저를 둘러싼 구문입니다. 리터럴 키워드 AS $$ 및 $$가 필요합니다.

Amazon Redshift에서는 $ 인용이라는 형식을 사용하여 프로시저의 문을 묶어야 합니다. 묶음 기호 내에 있는 것은 모두 정확히 그대로 전달됩니다. 문자열의 내용은 문자 그대로 작성되므로 특수 문자를 이스케이프할 필요가 없습니다.

$ 인용 사용 시, 달러 기호 쌍($$)을 사용하면 다음 예에 표시된 것처럼 실행할 문의 시작과 끝을 나타낼 수 있습니다.

```
$$ my statement $$
```

각 쌍의 달러 기호들 사이에서 선택적으로 문을 식별하는 데 도움이 되는 문자열을 지정할 수 있습니다. 사용하는 문자열은 묶음 쌍의 시작과 끝에서 모두 동일해야 합니다. 이 문자열은 대/소문자를

구분하고 달러 기호를 포함할 수 없는 경우를 제외하면 따옴표가 없는 식별자와 똑같은 제약 조건을 따릅니다. 다음 예에서는 문자열 테스트를 사용합니다.

```
$test$ my statement $test$
```

이 구문은 중첩 달러 인용에도 유용합니다. 달러 인용에 대한 자세한 내용은 PostgreSQL 설명서의 어휘 구조에서 "달러 기호로 인용된 문자열 상수"를 참조하십시오.

procedure_body

유효한 PL/pgSQL 문 세트입니다. PL/pgSQL 문은 루프 및 조건 표현식을 비롯한 프로시저 구문으로 SQL 명령을 보완하여 논리 흐름을 제어합니다. COPY, UNLOAD, INSERT 등의 데이터 수정 언어(DML)와 CREATE TABLE 등의 데이터 정의 언어(DDL)를 포함한 대부분의 SQL 명령을 프로시저 본문에서 사용할 수 있습니다. 자세한 내용은 PL/pgSQL 언어 참조 단원을 참조하십시오.

LANGUAGE plpgsql

언어 값입니다. `plpgsql`를 지정합니다. `plpgsql`을 사용하려면 언어에 대한 사용 권한이 필요합니다. 자세한 내용은 GRANT 단원을 참조하십시오.

NONATOMIC

NONATOMIC 트랜잭션 모드에서 저장 프로시저를 생성합니다. NONATOMIC 모드는 프로시저 내에서 문을 자동으로 커밋합니다. 또한 NONATOMIC 프로시저 내에서 오류가 발생하여 예외 블록에서 처리되는 경우 오류가 다시 발생하지 않습니다. 자세한 내용은 트랜잭션 관리 및 RAISE 단원을 참조하세요.

저장 프로시저를 `NONATOMIC`으로 정의할 때 다음 사항을 고려하세요.

- 저장 프로시저 호출을 중첩할 때는 모든 프로시저를 동일한 트랜잭션 모드에서 생성해야 합니다.
- NONATOMIC 모드에서 프로시저를 생성할 때는 `SECURITY DEFINER` 옵션과 `SET configuration_parameter` 옵션이 지원되지 않습니다.
- 암시적 커밋이 처리되면 (명시적으로 또는 암시적으로) 열린 모든 커서가 자동으로 닫힙니다. 따라서 커서 루프를 시작하기 전에 명시적 트랜잭션을 열어 루프 반복 내의 SQL이 암시적으로 커밋되지 않도록 해야 합니다.

SECURITY INVOKER | SECURITY DEFINER

`NONATOMIC`이 지정된 경우 `SECURITY DEFINER` 옵션이 지원되지 않습니다.

프로시저의 보안 모드는 실행 시간에 프로시저의 액세스 권한을 결정합니다. 프로시저는 기본 데이터베이스 객체에 액세스할 권한이 있어야 합니다.

SECURITY INVOKER 모드에서 프로시저는 프로시저를 호출하는 사용자의 권한을 사용합니다. 사용자는 기본 데이터베이스 객체에 대한 명시적 권한이 있어야 합니다. 기본값은 SECURITY INVOKER입니다.

SECURITY DEFINER 모드에서, 프로시저는 프로시저 소유자의 권한을 사용합니다. 프로시저 소유자는 런타임에 프로시저를 소유하는 사용자로 정의되며, 프로시저를 처음 정의한 사용자일 필요는 없습니다. 프로시저를 호출하는 사용자는 프로시저에 대한 권한을 실행해야 하지만, 기본 객체에 대한 권한은 필요하지 않습니다.

SET configuration_parameter { TO value | = value }

NONATOMIC이 지정된 경우 이 옵션이 지원되지 않습니다.

SET 절은 프로시저가 입력될 때 지정된 configuration_parameter가 지정된 값으로 설정되게 합니다. 그런 다음 이 절은 프로시저가 종료되면 configuration_parameter를 이전 값으로 복원합니다.

사용 노트

SECURITY DEFINER 옵션을 사용하여 저장 프로시저를 생성한 경우, 저장 프로시저 내에서 CURRENT_USER 함수를 호출하면 Amazon Redshift는 저장 프로시저 소유자의 사용자 이름을 반환합니다.

예시

> **Note**
>
> 이러한 예제를 실행할 때 다음과 비슷한 오류가 발생하는 경우
>
> ```
> ERROR: 42601: [Amazon](500310) unterminated dollar-quoted string at or near "$$
> ```
>
> Amazon Redshift의 저장 프로시저 개요 섹션을 참조하세요.

다음 예제에서는 입력 파라미터가 두 개 있는 프로시저를 생성합니다.

```
CREATE OR REPLACE PROCEDURE test_sp1(f1 int, f2 varchar(20))
AS $$
DECLARE
  min_val int;
```

```
BEGIN
  DROP TABLE IF EXISTS tmp_tbl;
  CREATE TEMP TABLE tmp_tbl(id int);
  INSERT INTO tmp_tbl values (f1),(10001),(10002);
  SELECT INTO min_val MIN(id) FROM tmp_tbl;
  RAISE INFO 'min_val = %, f2 = %', min_val, f2;
END;
$$ LANGUAGE plpgsql;
```

> **Note**
>
> 저장 프로시저를 작성할 때는 민감한 값을 보호하기 위한 모범 사례를 참조하는 것이 좋습니다.
> 민감한 정보를 저장 프로시저 로직 내에 하드 코딩하지 마십시오. 예를 들어, 저장 프로시저 본문의 CREATE USER 문에 사용자 암호를 할당하지 마십시오. 하드 코딩된 값이 카탈로그 테이블에 스키마 메타 데이터로 기록될 수 있기 때문에 보안 위험이 따릅니다. 암호와 같은 민감한 값은 파라미터를 사용하여 저장 프로시저에 인수로 전달하십시오.
> 저장된 프로시저에 대한 자세한 내용은 프로시저 생성 및 Amazon Redshift에서 저장 프로시저 생성을 참조하십시오. 카탈로그 테이블에 대한 자세한 내용은 시스템 카탈로그 테이블을 참조하십시오.

다음 예제에서는 IN 파라미터 한 개, OUT 파라미터 한 개, INOUT 파라미터 한 개가 있는 프로시저를 생성합니다.

```
CREATE OR REPLACE PROCEDURE test_sp2(f1 IN int, f2 INOUT varchar(256), out_var OUT
 varchar(256))
AS $$
DECLARE
  loop_var int;
BEGIN
  IF f1 is null OR f2 is null THEN
    RAISE EXCEPTION 'input cannot be null';
  END IF;
  DROP TABLE if exists my_etl;
  CREATE TEMP TABLE my_etl(a int, b varchar);
    FOR loop_var IN 1..f1 LOOP
        insert into my_etl values (loop_var, f2);
        f2 := f2 || '+' || f2;
    END LOOP;
  SELECT INTO out_var count(*) from my_etl;
```

```
END;
$$ LANGUAGE plpgsql;
```

다음 예제에서는 SECURITY DEFINER 파라미터를 사용하는 프로시저를 생성합니다. 이 프로시저는 프로시저를 소유한 사용자의 권한을 사용하여 실행됩니다.

```
CREATE OR REPLACE PROCEDURE sp_get_current_user_definer()
AS $$
DECLARE curr_user varchar(250);
BEGIN
  SELECT current_user INTO curr_user;
  RAISE INFO '%', curr_user;
END;
$$ LANGUAGE plpgsql
SECURITY DEFINER;
```

다음 예제에서는 SECURITY INVOKER 파라미터를 사용하는 프로시저를 생성합니다. 이 프로시저는 프로시저를 실행하는 사용자의 권한을 사용하여 실행됩니다.

```
CREATE OR REPLACE PROCEDURE sp_get_current_user_invoker()
AS $$
DECLARE curr_user varchar(250);
BEGIN
  SELECT current_user INTO curr_user;
  RAISE INFO '%', curr_user;
END;
$$ LANGUAGE plpgsql
SECURITY INVOKER;
```

CREATE RLS POLICY

데이터베이스 객체에 대한 세분화된 액세스를 제공하는 새 행 수준 보안 정책을 생성합니다.

sys:secadmin 역할이 부여된 수퍼유저와 사용자 또는 역할은 정책을 생성할 수 있습니다.

구문

```
CREATE RLS POLICY policy_name
[ WITH (column_name data_type [, ...]) [ [AS] relation_alias ] ]
```

```
USING ( using_predicate_exp )
```

파라미터

policy_name

정책의 이름입니다.

WITH (column_name data_type [, ...])

정책이 연결된 테이블의 열에 참조되는 column_name과 data_type을 지정합니다.

연결된 테이블 열을 RLS 정책에서 참조하지 않는 경우에만 WITH 절을 생략할 수 있습니다.

AS relation_alias

RLS 정책이 연결될 테이블의 별칭을 선택적으로 지정합니다.

USING (using_predicate_exp)

쿼리의 WHERE 절에 적용되는 필터를 지정합니다. Amazon Redshift는 쿼리 수준 사용자 조건자를 지정하기 전에 정책 조건자를 적용합니다. 예를 들어 **`current_user = 'joe' and price > 10`**은 가격이 10 USD보다 큰 레코드만 Joe에게 표시하도록 제한합니다.

사용 노트

CREATE RLS POLICY 문과 관련한 작업을 수행할 때는 다음을 준수하세요.

- Amazon Redshift는 쿼리의 WHERE 절에 포함될 수 있는 필터를 지원합니다.
- 테이블에 연결되는 모든 정책은 동일한 테이블 별칭을 사용하여 생성되어야 합니다.
- 조회 테이블에 대한 SELECT 권한은 필요하지 않습니다. 정책을 생성하면 Amazon Redshift가 각 정책의 조회 테이블에 대한 SELECT 권한을 부여합니다. 조회 테이블은 정책 정의 내에 사용되는 테이블 객체입니다.
- Amazon Redshift 행 수준 보안은 정책 정의 내에서 카탈로그 테이블, 교차 데이터베이스 관계, 외부 테이블, 일반 보기, 늦은 바인딩 보기, RLS 정책이 설정된 테이블, 임시 테이블 등의 객체 유형을 지원하지 않습니다.

예시

다음 SQL 문은 CREATE RLS POLICY 예의 테이블, 사용자 및 역할을 생성합니다.

```
-- Create users and roles reference in the policy statements.
CREATE ROLE analyst;

CREATE ROLE consumer;

CREATE USER bob WITH PASSWORD 'Name_is_bob_1';

CREATE USER alice WITH PASSWORD 'Name_is_alice_1';

CREATE USER joe WITH PASSWORD 'Name_is_joe_1';

GRANT ROLE sys:secadmin TO bob;

GRANT ROLE analyst TO alice;

GRANT ROLE consumer TO joe;

GRANT ALL ON TABLE tickit_category_redshift TO PUBLIC;
```

다음 예에서는 policy_concerts라는 정책을 만듭니다.

```
CREATE RLS POLICY policy_concerts
WITH (catgroup VARCHAR(10))
USING (catgroup = 'Concerts');
```

역할 생성

권한 모음인 새 사용자 지정 역할을 생성합니다. Amazon Redshift 시스템 정의 역할 목록은 the section called “Amazon Redshift 시스템 정의 역할” 섹션을 참조하세요. SVV_ROLES를 쿼리하여 클러스터 또는 작업 그룹에 현재 생성된 역할을 확인합니다.

생성할 수 있는 역할 수에는 할당량이 있습니다. 자세한 내용은 Amazon Redshift 관리 가이드의 Amazon Redshift의 할당량 및 제한 섹션을 참조하세요.

필수 권한

CREATE ROLE에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- CREATE ROLE 권한이 있는 사용자

구문

```
CREATE ROLE role_name
[ EXTERNALID external_id ]
```

파라미터

role_name

역할의 이름. 역할의 이름은 고유해야 하며 사용자 이름과 같을 수 없습니다. 역할 이름은 예약어가 될 수 없습니다.

CREATE ROLE 권한이 있는 슈퍼 사용자 또는 일반 사용자는 역할을 생성할 수 있습니다. 슈퍼 사용자가 아닌 사용자이지만 WITH GRANT OPTION과 ALTER 권한 역할에 USAGE가 부여되었다면 누구에게나 이 역할을 부여할 수 있습니다.

EXTERNALID external_id

자격 증명 공급자와 연결된 역할의 식별자입니다. 자세한 내용은 Native identity provider (IdP) federation for Amazon Redshift(Amazon Redshift용 네이티브 자격 증명 공급자(IdP) 페더레이션)를 참조하세요.

예시

다음 예에서는 `sample_role1` 역할을 생성합니다.

```
CREATE ROLE sample_role1;
```

다음 예에서는 자격 증명 공급자와 연결된 외부 ID를 사용하여 역할 `sample_role1`을 생성합니다.

```
CREATE ROLE sample_role1 EXTERNALID "ABC123";
```

CREATE SCHEMA

현재 데이터베이스에 대한 새 스키마를 정의합니다.

필수 권한

CREATE SCHEMA에 필요한 권한은 다음과 같습니다.

- 수퍼유저

- CREATE SCHEMA 권한이 있는 사용자

구문

```
CREATE SCHEMA [ IF NOT EXISTS ] schema_name [ AUTHORIZATION username ]
           [ QUOTA {quota [MB | GB | TB] | UNLIMITED} ] [ schema_element [ ... ]

CREATE SCHEMA AUTHORIZATION username[ QUOTA {quota [MB | GB | TB] | UNLIMITED} ]
 [ schema_element [ ... ] ]
```

파라미터

IF NOT EXISTS

지정된 스키마가 이미 존재하는 경우 오류 메시지와 함께 종료하는 대신, 명령이 아무 것도 변경하지 않고 스키마가 존재한다는 메시지를 반환함을 나타내는 절입니다.

이 절은 스크립트 작성 시 유용하므로, CREATE SCHEMA가 이미 존재하는 스키마의 생성을 시도하는 경우에는 스크립트가 실패하지 않습니다.

schema_name

새 스키마의 이름입니다. 스키마 이름은 PUBLIC일 수 없습니다. 유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요.

> **Note**
>
> 똑같은 이름의 객체가 스키마 이름 없이 참조될 때 이런 객체의 우선 순위는 search_path 구성 파라미터에 있는 스키마의 목록에 따라 결정됩니다.

AUTHORIZATION

지정된 사용자에게 소유권을 부여하는 절입니다.

사용자 이름

스키마 소유자의 이름입니다.

schema_element

스키마 내에서 생성할 하나 이상의 객체에 대한 정의입니다.

QUOTA

지정된 스키마에서 사용할 수 있는 최대 디스크 공간 양입니다. 이 공간은 총제적 디스크 사용량입니다. 여기에는 모든 영구 테이블, 지정된 스키마 아래의 구체화된 보기 및 각 컴퓨팅 노드에서 ALL 배포가 있는 모든 테이블의 중복 복사본이 포함됩니다. 임시 네임스페이스 또는 스키마의 일부로 생성된 임시 테이블은 스키마 할당량에서 고려되지 않습니다.

구성된 스키마 할당량을 보려면 SVV_SCHEMA_QUOTA_STATE 섹션을 참조하세요.

스키마 할당량이 초과된 레코드를 보려면 STL_SCHEMA_QUOTA_VIOLATIONS 섹션을 참조하세요.

Amazon Redshift는 선택한 값을 메가바이트로 변환합니다. 값을 지정하지 않으면 기가바이트가 기본 측정 단위입니다.

스키마 할당량을 설정하고 변경하려면 데이터베이스 수퍼유저여야 합니다. 수퍼유저는 아니어도 CREATE SCHEMA 권한이 있는 사용자는 정의된 할당량으로 스키마를 생성할 수 있습니다. 할당량을 정의하지 않고 스키마를 생성하면 스키마는 무제한 할당량을 가집니다. 스키마에서 사용하는 현재 값 이하로 할당량을 설정하면 Amazon Redshift에서는 디스크 공간을 확보할 때까지 추가 수집을 허용하지 않습니다. DELETE 문은 테이블에서 데이터를 삭제하고 VACUUM이 실행될 때만 디스크 공간이 확보됩니다.

Amazon Redshift는 트랜잭션을 커밋하기 전에 각 트랜잭션에 할당량 위반이 있는지 확인합니다. Amazon Redshift는 설정된 할당량에 대해 수정된 각 스키마의 크기(스키마의 모든 테이블이 사용하는 디스크 공간)를 확인합니다. 트랜잭션이 끝날 때 할당량 위반 검사가 발생하기 때문에 크기 제한은 커밋되기 전에 트랜잭션 내에서 일시적으로 할당량을 초과할 수 있습니다. 트랜잭션이 할당량을 초과하면 Amazon Redshift는 트랜잭션을 중단하고 후속 수집 작업을 금지하며 디스크 공간을 확보할 때까지 모든 변경 사항을 되돌립니다. 백그라운드 VACUUM 및 내부 정리로 인해 취소된 트랜잭션 후 스키마를 검사할 때까지 스키마가 가득 차지 않을 수 있습니다.

예외적으로 Amazon Redshift는 할당량 위반을 무시하고 특정 경우에 트랜잭션을 커밋합니다. Amazon Redshift는 동일한 트랜잭션에 INSERT 또는 COPY 수집 문이 없는 다음 문 중 하나 이상으로만 구성된 트랜잭션에 대해 이 작업을 수행합니다.

- DELETE
- TRUNCATE
- VACUUM
- DROP TABLE
- 전체 스키마에서 다른 비전체 스키마로 데이터를 이동할 때만 ALTER TABLE APPEND

UNLIMITED

Amazon Redshift는 스키마의 총 크기 증가에 제한을 두지 않습니다.

Limits

Amazon Redshift에서는 스키마에 대해 다음과 같은 제한 사항을 둡니다.

- 데이터베이스당 스키마 수는 최대 9,900개입니다.

예시

다음 예제에서는 US_SALES로 명명된 스키마를 생성하고 사용자 DWUSER에게 소유권을 부여합니다.

```
create schema us_sales authorization dwuser;
```

다음 예제에서는 US_SALES로 명명된 스키마를 생성하고 사용자 DWUSER에게 소유권을 부여하며 할당량을 50GB로 설정합니다.

```
create schema us_sales authorization dwuser QUOTA 50 GB;
```

새로운 스키마를 보려면 다음과 같이 PG_NAMESPACE 카탈로그 테이블을 쿼리하십시오.

```
select nspname as schema, usename as owner
from pg_namespace, pg_user
where pg_namespace.nspowner = pg_user.usesysid
and pg_user.usename ='dwuser';

   schema |  owner
----------+----------
 us_sales | dwuser
(1 row)
```

다음 예제에서는 US_SALES 스키마를 생성하거나 이 스키마가 이미 존재하는 경우에는 아무 것도 하지 않고 메시지를 반환합니다.

```
create schema if not exists us_sales;
```

CREATE TABLE

현재 데이터베이스에서 새 테이블을 생성합니다. 열 목록을 정의합니다. 각 열에는 고유한 유형의 데이터가 들어 있습니다. 테이블의 소유자는 CREATE TABLE 명령의 발행자입니다.

필수 권한

CREATE TABLE에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- CREATE TABLE 권한이 있는 사용자

구문

```
CREATE [ [LOCAL ] { TEMPORARY | TEMP } ] TABLE
[ IF NOT EXISTS ] table_name
( { column_name data_type [column_attributes] [ column_constraints ]
  | table_constraints
  | LIKE parent_table [ { INCLUDING | EXCLUDING } DEFAULTS ] }
  [, ... ]  )
[ BACKUP { YES | NO } ]
[table_attributes]

where column_attributes are:
  [ DEFAULT default_expr ]
  [ IDENTITY ( seed, step ) ]
  [ GENERATED BY DEFAULT AS IDENTITY ( seed, step ) ]
  [ ENCODE encoding ]
  [ DISTKEY ]
  [ SORTKEY ]
  [ COLLATE CASE_SENSITIVE | COLLATE CASE_INSENSITIVE  ]

and column_constraints are:
  [ { NOT NULL | NULL } ]
  [ { UNIQUE  |  PRIMARY KEY } ]
  [ REFERENCES reftable [ ( refcolumn ) ] ]

and table_constraints  are:
  [ UNIQUE ( column_name [, ... ] ) ]
  [ PRIMARY KEY ( column_name [, ... ] )  ]
  [ FOREIGN KEY (column_name [, ... ] ) REFERENCES reftable [ ( refcolumn ) ]
```

```
and table_attributes are:
  [ DISTSTYLE { AUTO | EVEN | KEY | ALL } ]
  [ DISTKEY ( column_name ) ]
  [ [COMPOUND | INTERLEAVED ] SORTKEY ( column_name [,...]) |  [ SORTKEY AUTO ] ]
  [ ENCODE AUTO ]
```

파라미터

LOCAL

선택 사항. 이 키워드는 문에서 허용되지만, Amazon Redshift에는 아무런 효과도 없습니다.

TEMPORARY | TEMP

현재 세션 내에서만 보이는 임시 테이블을 생성하는 키워드입니다. 이 테이블은 자신이 생성된 세션이 끝날 때 자동으로 삭제됩니다. 임시 테이블의 이름은 영구 테이블의 이름과 같을 수 있습니다. 임시 테이블은 세션별로 별개의 스키마에서 생성됩니다. (이 스키마의 이름을 지정할 수는 없습니다.) 이 임시 스키마는 검색 경로의 첫 번째 스키마가 되므로, 영구 테이블에 액세스하려고 스키마 이름으로 테이블 이름을 정규화하지 않는 한 임시 테이블이 영구 테이블보다 우선하게 됩니다. 스키마와 우선 순위에 대한 자세한 내용은 search_path 섹션을 참조하세요.

> **Note**
>
> 기본적으로, 데이터베이스 사용자는 PUBLIC 그룹에서 자동 멤버십으로 임시 테이블을 생성할 권한이 있습니다. 어떤 사용자에 대해 이 권한을 거부하려면 PUBLIC 그룹에서 TEMP 권한을 취소한 다음, 특정 사용자 또는 사용자 그룹에게만 TEMP 권한을 명시적으로 허용하십시오.

IF NOT EXISTS

지정된 테이블이 이미 존재하는 경우 오류 메시지와 함께 중지하는 대신, 명령이 아무 것도 변경하지 않고 테이블이 존재한다는 메시지를 반환함을 나타내는 절입니다. 참고로, 기존 테이블은 사용자가 생성하려고 했던 테이블과는 전혀 다를 수도 있습니다. 테이블 이름만 비교됩니다.

이 절은 스크립트 작성 시 유용하므로, CREATE TABLE이 이미 존재하는 테이블의 생성을 시도하는 경우에는 스크립트가 실패하지 않습니다.

table_name

생성할 테이블의 이름입니다.

> **Important**
>
> '#'으로 시작하는 테이블 이름을 지정하면 테이블이 임시 테이블로 생성됩니다. 다음은 예제입니다.
>
> ```
> create table #newtable (id int);
> ```
>
> '#' 기호를 사용하여 테이블을 참조할 수도 있습니다. 예시:
>
> ```
> select * from #newtable;
> ```

테이블 이름의 최대 길이는 127바이트이며, 이보다 긴 이름은 127바이트까지 표시되고 나머지는 잘립니다. 최대 4바이트까지 UTF-8 멀티바이트 문자를 사용할 수 있습니다. Amazon Redshift는 사용자 정의 임시 테이블과 쿼리 처리 또는 시스템 유지 관리 중에 Amazon Redshift에서 생성한 임시 테이블을 포함하여 노드 유형별로 클러스터당 테이블 수 할당량을 적용합니다. 선택적으로, 데이터베이스 및 스키마 이름으로 테이블 이름을 정규화할 수 있습니다. 다음 예에서는 데이터베이스 이름이 `tickit`이고, 스키마 이름은 `public`이며, 테이블 이름은 `test`입니다.

```
create table tickit.public.test (c1 int);
```

데이터베이스 또는 스키마가 존재하지 않는 경우 테이블이 생성되지 않으며, 이 명령문은 오류를 반환합니다. 시스템 데이터베이스 `template0`, `template1`, `padb_harvest` 또는 `sys:internal`에서 테이블 또는 뷰를 생성할 수 없습니다.

스키마 이름이 주어지는 경우 새 테이블은 그 스키마에서 생성됩니다(생성자가 해당 스키마에 액세스할 수 있다고 가정). 테이블 이름은 그 스키마에 대한 고유한 이름이어야 합니다. 아무런 스키마도 지정되지 않으면 현재 데이터베이스 스키마를 사용하여 테이블이 생성됩니다. 임시 테이블을 생성할 경우, 임시 테이블이 특수한 스키마에 존재하므로 스키마 이름을 지정할 수 없습니다.

같은 이름을 가진 여러 임시 테이블이 각각 별도의 세션에서 생성되는 경우에는 각기 다른 스키마에 할당되므로, 이런 테이블은 같은 데이터베이스에 동시에 존재할 수 있습니다. 유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요.

column_name

새 테이블에 생성할 열의 이름입니다. 열 이름의 최대 길이는 127바이트이며, 이보다 긴 이름은 127바이트까지 표시되고 나머지는 잘립니다. 최대 4바이트까지 UTF-8 멀티바이트 문자를 사용할 수 있습니다. 단일 테이블에서 정의할 수 있는 최대 열 수는 1,600개입니다. 유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요.

> **Note**
>
> "넓은 테이블"을 생성할 경우에는 로드 및 쿼리 처리 중에 열의 목록이 중간 결과에 대한 행 너비 경계를 초과하지 않도록 주의하십시오. 자세한 내용은 사용 노트 단원을 참조하십시오.

data_type

생성되는 열의 데이터 형식입니다. CHAR 및 VARCHAR 열의 경우 최대 길이를 선언하는 대신 MAX 키워드를 사용할 수 있습니다. MAX는 최대 길이를 CHAR의 경우 4,096바이트, VARCHAR의 경우 65,535바이트로 설정합니다. GEOMETRY 객체의 최대 크기는 1,048,447바이트입니다.

Amazon Redshift에서 지원하는 데이터 형식에 대한 자세한 내용은 데이터 타입 섹션을 참조하세요.

DEFAULT default_expr

열의 기본 데이터 값을 할당하는 절입니다. default_expr의 데이터 형식은 열의 데이터 형식과 일치해야 합니다. DEFAULT 값은 변수가 없는 표현식이어야 합니다. 하위 쿼리, 현재 테이블에 있는 다른 열과의 상호 참조, 사용자 정의 함수는 허용되지 않습니다.

default_expr 표현식은 열의 값을 지정하지 않는 INSERT 작업에 사용됩니다. 기본값이 지정되지 않은 경우 열의 기본값은 Null입니다.

열 목록이 정의된 COPY 작업에서 DEFAULT 값을 가진 열을 생략하는 경우 COPY 명령은 default_expr의 값을 삽입합니다.

IDENTITY(seed, step)

열이 IDENTITY 열임을 지정하는 절입니다. IDENTITY 열에는 자동 생성되는 고유한 값이 있습니다. IDENTITY 열의 데이터 형식은 INT 또는 BIGINT여야 합니다.

INSERT 또는 INSERT INTO [tablename] VALUES() 문을 사용하여 행을 추가할 때 이런 값은 seed로 지정되는 값으로 시작하고 단계로 지정되는 수만큼 증가합니다.

`INSERT INTO [tablename] SELECT * FROM` 또는 COPY 문을 사용하여 테이블을 로드하면 데이터가 병렬로 로드되고 노드 조각으로 배포됩니다. 자격 증명 값을 생성할 때는 자격 증명 값이 고유할 수 있도록 Amazon Redshift가 다수의 값을 건너뜁니다. 자격 증명 값은 고유하지만 순서는 소스 파일의 순서와 일치하지 않을 수 있습니다.

GENERATED BY DEFAULT AS IDENTITY(seed, step)

열이 기본 IDENTITY 열임을 지정하고, 사용자가 열에 고유한 값을 자동으로 할당할 수 있는 절. IDENTITY 열의 데이터 형식은 INT 또는 BIGINT여야 합니다. 값 없이 행을 추가할 때 이런 값은 seed로 지정되는 값으로 시작하고 단계로 지정되는 수만큼 증가합니다. 값이 생성되는 과정에 대한 자세한 내용은 IDENTITY를 참조하십시오.

또한 INSERT, UPDATE 또는 COPY 중 EXPLICIT_IDS 없이 값을 제공할 수 있습니다. Amazon Redshift는 시스템 생성 값을 사용하는 대신 해당 값을 사용하여 자격 증명 열에 삽입합니다. 이 값은 seed보다 작은 중복 값 또는 단계 값 사이의 값일 수 있습니다. Amazon Redshift는 열에서 값의 고유성을 확인하지 않습니다. 값을 입력해도 다음 시스템 생성 값에는 영향을 미치지 않습니다.

> **Note**
>
> 열에 고유성이 필요한 경우 중복 값을 추가하지 마십시오. 대신, seed보다 작거나 단계 값 사이의 고유한 값을 추가합니다.

기본 자격 증명 열의 경우, 다음에 유의하십시오.

- 기본 자격 증명 열은 NOT NULL입니다. NULL을 삽입할 수 없습니다.
- 생성된 값을 기본 자격 증명 열에 삽입하려면 키워드 DEFAULT를 사용하십시오.

```
INSERT INTO tablename (identity-column-name) VALUES (DEFAULT);
```

- 기본 자격 증명 열의 값을 재정의해도 다음에 생성되는 값에는 영향을 미치지 않습니다.
- ALTER TABLE ADD COLUMN 문으로 기본 자격 증명 열을 추가할 수 없습니다.
- ALTER TABLE APPEND 문으로 기본 자격 증명 열을 추가할 수 있습니다.

ENCODE encoding

열에 대한 압축 인코딩입니다. ENCODE AUTO는 테이블의 기본값입니다. Amazon Redshift는 테이블의 모든 열에 대한 압축 인코딩을 자동으로 관리하지 않습니다. 테이블의 열에 압축 인코딩을 지정하면 테이블이 더 이상 ENCODE AUTO로 설정되지 않습니다. Amazon Redshift는 더 이상 테이블의 모든 열에 대한 압축 인코딩을 자동으로 관리하지 않습니다. 테이블에 대해 ENCODE

AUTO 옵션을 지정하여 Amazon Redshift에서 테이블의 모든 열에 대한 압축 인코딩을 자동으로 관리하도록 할 수 있습니다.

Amazon Redshift는 다음과 같이 압축 인코딩을 지정하지 않은 열에 초기 압축 인코딩을 자동으로 할당합니다.

- 임시 테이블의 모든 열은 기본적으로 RAW 압축으로 할당됩니다.
- 정렬 키로 정의된 열은 RAW 압축이 할당됩니다.
- BOOLEAN, REAL, DOUBLE PRECISION, GEOMETRY 또는 GEOGRAPHY 데이터 유형으로 정의된 열은 RAW 압축이 할당됩니다.
- SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIME, TIMETZ, TIMESTAMP 또는 TIMESTAMPTZ로 정의된 열에는 AZ64 압축이 할당됩니다.
- CHAR, VARCHAR 또는 VARBYTE로 정의된 열에는 LZO 압축이 할당됩니다.

> **Note**
>
> 열을 압축하지 않으려면 RAW 인코딩을 명시적으로 지정하십시오.

다음 모듈을 지원합니다.compression encodings (p. 59)

- AZ64
- BYTEDICT
- 델타
- 델타
- LZO
- LZO
- LZO
- LZO
- RAW(압축 없음)
- RUNLENGTH
- TEXT255
- TEXT32K
- ZSTD

DISTKEY

해당 열이 테이블에 대한 배포 키임을 지정하는 키워드입니다. 테이블에서는 한 개의 열만 배포 키일 수 있습니다. DISTKEY 키워드를 열 이름 뒤에 사용하거나 DISTKEY (column_name) 구문을 사용하여 테이블 정의의 일부로 사용할 수 있습니다. 둘 중 어떤 방법을 사용하든 그 효과는 똑같습니다. 자세한 내용은 이 주제의 뒷부분에 나오는 DISTSTYLE 파라미터를 참조하십시오.

배포 키 열의 데이터 형식은 BOOLEAN, REAL, DOUBLE PRECISION, SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIME, TIMETZ, TIMESTAMP 또는 TIMESTAMPTZ, CHAR 또는 VARCHAR일 수 있습니다.

SORTKEY

해당 열이 테이블에 대한 정렬 키임을 지정하는 키워드입니다. 데이터가 테이블로 로드될 때 데이터는 정렬 키로 지정되는 하나 이상의 열을 기준으로 정렬됩니다. 열 이름 뒤에 SORTKEY 키워드를 사용하여 단일 열 정렬 키를 지정하거나, SORTKEY (column_name [, ...]) 구문을 사용하여 하나 이상의 열을 테이블의 정렬 키 열로 지정할 수 있습니다. 이 구문을 사용하면 복합 정렬 키만 생성됩니다.

테이블당 최대 400개의 SORTKEY 열을 정의할 수 있습니다.

정렬 키 열의 데이터 형식은 BOOLEAN, REAL, DOUBLE PRECISION, SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIME, TIMETZ, TIMESTAMP 또는 TIMESTAMPTZ, CHAR 또는 VARCHAR일 수 있습니다.

COLLATE CASE_SENSITIVE | COLLATE CASE_INSENSITIVE

열의 문자열 검색 또는 비교가 CASE_SENSITIVE인지, CASE_INSENSITIVE인지를 지정하는 절입니다. 기본값은 데이터베이스의 현재 대/소문자 구분 구성과 동일합니다.

데이터베이스 데이터 정렬 정보를 찾으려면 다음 명령을 사용합니다.

```
SELECT db_collation();

db_collation
----------------
 case_sensitive
(1 row)
```

NOT NULL | NULL

NOT NULL은 열이 null 값을 포함하도록 허용되지 않도록 지정합니다. 기본값인 NULL은 열이 null 값을 허용하도록 지정합니다. IDENTITY 열은 기본적으로 NOT NULL로 선언됩니다.

UNIQUE

해당 열이 고유한 값만 포함할 수 있음을 지정하는 키워드입니다. 고유한 테이블 제약 조건의 동작은 여러 열에 적용할 수 있는 추가적인 기능과 함께, 열 제약 조건에 대한 동작과 동일합니다. 고유한 테이블 제약 조건을 정의하려면 UNIQUE (column_name [, ...]) 구문을 사용하십시오.

> **Important**
>
> 고유 제약 조건은 정보를 제공하기 위한 것으로, 시스템에서 이를 강제 적용하지는 않습니다.

PRIMARY KEY

해당 열이 테이블에 대한 기본 키임을 지정하는 키워드입니다. 열 정의를 사용하면 한 개의 열만 기본 키로 정의할 수 있습니다. 다중 열 기본 키로 테이블 제약 조건을 정의하려면 PRIMARY KEY (column_name [, ...]) 구문을 사용하십시오.

열을 기본 키로 식별하면 스키마의 설계에 대한 메타데이터가 제공됩니다. 기본 키는 다른 테이블들이 이 열 집합을 행의 고유 식별자로 사용할 수 있음을 암시합니다. 테이블에 대해 한 개의 기본 키를 열 제약 조건이나 테이블 조약 조건으로 지정할 수 있습니다. 기본 키 제약 조건은 같은 테이블에 대해 정의된 고유 제약 조건에 의해 명명되는 다른 열 집합과는 상이한 열 집합을 명명해야 합니다.

PRIMARY KEY 열도 NOT NULL로 정의됩니다.

> **Important**
>
> 기본 키 제약 조건은 정보 제공만을 목적으로 합니다. 기본 키 제약 조건은 시스템에서 강제 적용되지 않지만, 플래너는 이 제약 조건을 사용합니다.

References reftable [(refcolumn)]

외래 키 제약 조건을 지정하는 절로, 해당 열은 참조되는 테이블의 어떤 행에서 참조되는 열에 있는 값과 일치하는 값만 포함해야 함을 암시합니다. 참조되는 열은 참조되는 테이블에서 고유 키 또는 기본 키 제약 조건의 열이어야 합니다.

> **Important**
>
> 외래 키 제약 조건은 정보 제공만을 목적으로 합니다. 기본 키 제약 조건은 시스템에서 강제 적용되지 않지만, 플래너는 이 제약 조건을 사용합니다.

LIKE parent_table [{ INCLUDING | EXCLUDING } DEFAULTS]

새 테이블이 열 이름, 데이터 형식 및 NOT NULL 제약 조건을 자동으로 복사하는 기존 원본 테이블을 지정하는 절입니다. 새 테이블과 상위 테이블이 분리되고, 상위 테이블에 대한 변경 사항이 새 테이블에 적용되지 않습니다. 복사된 열 정의에 대한 기본 표현식은 INCLUDING DEFAULTS가 지정된 경우에만 복사됩니다. 기본 표현식을 제외하는 것이 기본 동작이므로, 새 테이블의 모든 열에 null 기본값이 있습니다.

LIKE 옵션으로 생성되는 테이블은 기본 키 및 외래 키 제약 조건을 상속하지 않습니다. 배포 스타일, 정렬 키, BACKUP 및 NULL 속성은 LIKE 테이블에 상속되지만 CREATE TABLE ...에서 명시적으로 설정할 수 없습니다. LIKE 문에서 이런 속성을 명시적으로 설정할 수 없습니다.

BACKUP { YES | NO }

자동 및 수동 클러스터 스냅샷에 테이블을 포함해야 할지 여부를 지정하는 절입니다.

중요한 데이터를 포함하지 않는 스테이징 테이블과 같은 테이블의 경우 BACKUP NO를 지정하여 스냅샷을 생성하고 스냅샷으로부터 복원할 때의 처리 시간을 절약하고 Amazon Simple Storage Service의 스토리지 공간을 줄입니다. BACKUP NO 설정은 클러스터 내에 있는 다른 노드로의 데이터 자동 복제에 아무런 영향도 미치지 않으므로, 노드 장애가 발생할 경우 BACKUP NO가 지정된 테이블이 복원됩니다. 기본값은 BACKUP YES입니다.

DISTSTYLE { AUTO | EVEN | KEY | ALL }

전체 테이블의 데이터 배포 스타일을 정의하는 키워드입니다. Amazon Redshift는 테이블에 대해 지정된 배포 스타일에 따라 컴퓨팅 노드에 테이블의 행을 배포합니다. 기본값은 AUTO입니다.

테이블에 대해 선택하는 분산 스타일이 데이터베이스의 전체 성능에 영향을 미칩니다. 자세한 내용은 쿼리 최적화를 위한 데이터 배포 단원을 참조하십시오. 가능한 분산 스타일은 다음과 같습니다.

- AUTO: Amazon Redshift에서 테이블 데이터를 기반으로 최적의 배포 스타일을 할당합니다. 예를 들어 AUTO 배포 스타일을 지정하면 Amazon Redshift는 처음에 작은 테이블에 ALL 배포 스타일을 할당합니다. 테이블이 커지면 Amazon Redshift는 배포 스타일을 KEY로 변경하여 기본 키(또는 복합 기본 키의 열)를 DISTKEY로 선택할 수 있습니다. 테이블이 커지고 DISTKEY로 적합한 열이 없으면 Amazon Redshift는 배포 스타일을 EVEN으로 변경합니다. 배포 스타일 변경은 사용자 쿼리에 미치는 영향을 최소화하면서 백그라운드에서 이루어집니다.

테이블에 적용된 분산 스타일을 보려면 PG_CLASS 시스템 카탈로그 테이블을 쿼리합니다. 자세한 내용은 분산 스타일 보기 단원을 참조하십시오.

- EVEN: 테이블의 데이터가 클러스터의 노드들에 걸쳐 라운드 로빈 배포 방식으로 균등하게 분포됩니다. 행 ID는 배포의 결정에 사용되며, 대략적으로 같은 수의 행이 각 노드로 배포됩니다.
- KEY: 데이터가 DISTKEY 열에 있는 값을 기준으로 배포됩니다. 조인 테이블의 조인 열을 배포 키로 설정하면 두 테이블 모두의 조인 행이 컴퓨팅 노드에 배치됩니다. 데이터가 배치되면 최적화 프로그램이 조인을 더 효율적으로 수행할 수 있습니다. DISTSTYLE KEY를 지정하는 경우 테이블에 대해, 또는 열 정의의 일부로서 DISTKEY 열의 이름을 지정해야 합니다. 자세한 내용은 이 주제의 앞부분에 나오는 DISTKEY 파라미터를 참조하십시오.
- ALL: 전체 테이블의 복사본이 모든 노드로 배포됩니다. 이 분산 스타일은 임의의 조인에 필요한 모든 행을 모든 노드에서 사용할 수 있도록 보장하지만, 스토리지 요구량이 몇 배로 늘고 테이블의 로드 시간과 유지 관리 시간도 증가합니다. ALL 배포는 KEY 배포가 적절하지 않은 특정 차원 테이블과 함께 사용할 때 실행 시간을 개선할 수 있지만, 성능 개선 정도를 유지 관리 비용과 비교 검토해야 합니다.

DISTKEY (column_name)

해당 열을 테이블에 대한 배포 키로 사용해야 함을 지정하는 제약 조건입니다. DISTKEY 키워드를 열 이름 뒤에 사용하거나 DISTKEY (column_name) 구문을 사용하여 테이블 정의의 일부로 사용할 수 있습니다. 둘 중 어떤 방법을 사용하든 그 효과는 똑같습니다. 자세한 내용은 이 주제의 앞부분에 나오는 DISTSTYLE 파라미터를 참조하십시오.

[COMPOUND | INTERLEAVED] SORTKEY (column_name [,...]) | [SORTKEY AUTO]

테이블에 대해 하나 이상의 정렬 키를 지정합니다. 데이터가 테이블로 로드될 때 데이터는 정렬 키로 지정되는 열을 기준으로 정렬됩니다. 열 이름 뒤에 SORTKEY 키워드를 사용하여 단일 열 정렬 키를 지정하거나, SORTKEY (column_name [, ...]) 구문을 사용하여 하나 이상의 열을 테이블의 정렬 키 열로 지정할 수 있습니다.

COMPOUND 또는 INTERLEAVED 정렬 스타일을 선택적으로 지정할 수 있습니다. 열과 함께 SORTKEY를 지정하는 경우 기본값은 COMPOUND입니다. 자세한 내용은 정렬 키 단원을 참조하십시오.

정렬 키 옵션을 지정하지 않는 경우 기본값은 AUTO입니다.

테이블당 최대 400개의 COMPOUND SORTKEY 열 또는 8개의 INTERLEAVED SORTKEY 열을 정의할 수 있습니다.

AUTO

Amazon Redshift에서 테이블 데이터를 기반으로 최적의 정렬 키를 할당하도록 지정합니다. 예를 들어 AUTO 정렬 키가 지정된 경우 Amazon Redshift는 처음에 테이블에 정렬 키를 할당하지 않습니다. 정렬 키가 쿼리 성능을 향상시킬 것이라고 판단되면 Amazon Redshift는 테이블의 정렬 키를 변경할 수 있습니다. 테이블의 실제 정렬은 자동 테이블 정렬에 의해 수행됩니다. 자세한 내용은 자동 테이블 정렬 단원을 참조하십시오.

Amazon Redshift는 기존 정렬 또는 배포 키가 있는 테이블을 수정하지 않습니다. 한 가지 예외를 제외하고 테이블에 JOIN에서 사용된 적이 없는 배포 키가 있는 경우 Amazon Redshift에서 더 나은 키가 있다고 판단하면 키가 변경될 수 있습니다.

테이블의 정렬 키를 보려면 SVV_TABLE_INFO 시스템 카탈로그 뷰를 쿼리합니다. 자세한 내용은 SVV_TABLE_INFO 단원을 참조하십시오. 테이블에 대한 Amazon Redshift Advisor 권장 사항을 보려면 SVV_ALTER_TABLE_RECOMMENDATIONS 시스템 카탈로그 뷰를 쿼리합니다. 자세한 내용은 SVV_ALTER_TABLE_RECOMMENDATIONS 단원을 참조하십시오. Amazon Redshift에서 수행한 작업을 보려면 SVL_AUTO_WORKER_ACTION 시스템 카탈로그 뷰를 쿼리합니다. 자세한 내용은 SVL_AUTO_WORKER_ACTION 단원을 참조하십시오.

COMPOUND

목록에 표시되는 모든 열로 구성된 복합 키를 사용하여 데이터가 나열되는 순서대로 정렬됨을 지정합니다. 복합 정렬 키는 쿼리가 정렬 열의 순서에 따라 행을 스캔할 때 가장 유용합니다. 쿼리가 보조 정렬 열에 의존할 때는 복합 키로 정렬함으로써 얻는 성능상의 이점이 감소합니다. 테이블당 최대 400개의 COMPOUND SORTKEY 열을 정의할 수 있습니다.

INTERLEAVED

데이터가 인터리브 정렬 키를 사용하여 정렬됨을 지정합니다. 인터리브 정렬 키에 대해 최대 8개의 열을 지정할 수 있습니다.

인터리브 정렬에서는 정렬 키에서 각 열이나 열의 하위 집합에 똑같은 가중치를 부여하므로, 쿼리가 정렬 키에 있는 열의 순서에 종속되지 않습니다. 쿼리가 하나 이상의 보조 정렬 열을 사용하는 경우 인터리브 정렬은 쿼리 성능을 크게 높여줍니다. 인터리브 정렬에는 데이터 로드와 vacuum 작업에 약간의 오버헤드 비용이 수반됩니다.

> **Important**
>
> 자격 증명 열, 날짜, 타임스탬프처럼 단순 증가하는 속성이 있는 열에 인터리브 정렬 키를 쓰지 마십시오.

ENCODE AUTO

Amazon Redshift에서 테이블의 모든 열에 대한 인코딩 형식을 자동으로 조정하여 쿼리 성능을 최적화할 수 있도록 합니다. ENCODE AUTO는 테이블 생성 시 지정한 초기 인코딩 형식을 유지합니다. 그런 다음 새 인코딩 형식이 쿼리 성능을 향상시킬 수 있다고 판단되면 Amazon Redshift가 테이블 열의 인코딩 형식을 변경할 수 있습니다. 테이블의 열에 인코딩 형식을 지정하지 않으면 ENCODE AUTO가 기본값입니다.

UNIQUE (column_name [,...])

테이블에서 하나 이상의 열로 구성된 그룹은 고유한 값만 포함할 수 있음을 지정하는 제약 조건입니다. 고유한 테이블 제약 조건의 동작은 여러 열에 적용할 수 있는 추가적인 기능과 함께, 열 제약 조건에 대한 동작과 동일합니다. 고유한 제약 조건이라는 맥락에서, null 값은 같은 값으로 간주되지 않습니다. 각각의 고유 테이블 제약 조건은 테이블에 대해 정의된 다른 고유 제약 조건이나 기본 키 제약 조건에 의해 명명되는 열의 집합과는 상이한 열 집합을 명명해야 합니다.

> **Important**
>
> 고유 제약 조건은 정보를 제공하기 위한 것으로, 시스템에서 이를 강제 적용하지는 않습니다.

PRIMARY KEY (column_name [,...])

테이블의 한 열 또는 여러 개의 열이 null이 아닌 고유한(중복되지 않는) 값만 포함할 수 있음을 지정하는 제약 조건입니다. 열 집합을 기본 키로 식별하면 스키마의 설계에 대한 메타데이터도 제공됩니다. 기본 키는 다른 테이블들이 이 열 집합을 행의 고유 식별자로 사용할 수 있음을 암시합니다. 테이블에 대해 한 개의 기본 키를 단일 열 제약 조건이나 테이블 조약 조건으로 지정할 수 있습니다. 기본 키 제약 조건은 같은 테이블에 대해 정의된 고유 제약 조건에 의해 명명되는 다른 열 집합과는 상이한 열 집합을 명명해야 합니다.

> **Important**
>
> 기본 키 제약 조건은 정보 제공만을 목적으로 합니다. 기본 키 제약 조건은 시스템에서 강제 적용되지 않지만, 플래너는 이 제약 조건을 사용합니다.

FOREIGN KEY (column_name [, ...]) REFERENCES reftable [(refcolumn)]

외래 키 제약 조건을 지정하는 제약 조건으로, 새 테이블에서 하나 이상의 열로 구성된 그룹이 참조되는 테이블의 어떤 행에서 참조되는 열에 있는 값과 일치하는 값만 포함해야 하도록 요구합니다. refcolumn이 생략된 경우에는 reftable의 기본 키가 사용됩니다. 참조되는 열은 참조되는 테이블에서 고유 키 또는 기본 키 제약 조건의 열이어야 합니다.

> **Important**
>
> 외래 키 제약 조건은 정보 제공만을 목적으로 합니다. 기본 키 제약 조건은 시스템에서 강제 적용되지 않지만, 플래너는 이 제약 조건을 사용합니다.

사용 노트

고유성, 프라이머리 키 및 외래 키 제약 조건은 참고용일 뿐 표를 채울 때 Amazon Redshift에 적용되지는 않습니다. 예를 들어 종속성이 있는 테이블에 데이터를 삽입하면 제약 조건을 위반하더라도 삽입이 성공할 수 있습니다. 그래도 기본 키와 외래 키는 계획 힌트로 사용되며, ETL 프로세스 또는 애플리케이션의 다른 프로세스가 무결성을 적용하는 경우에는 선언되어야 합니다. 종속성이 있는 테이블을 삭제하는 방법에 대한 내용은 DROP TABLE 섹션을 참조하세요.

제한 및 할당량

테이블을 생성할 경우 다음 제한 사항을 고려하십시오.

- 노드 유형별로 클러스터의 최대 테이블 수가 제한되어 있습니다. 자세한 내용은 Amazon Redshift 관리 가이드의 제한 섹션을 참조하세요.
- 테이블 이름으로 입력할 수 있는 최대 문자 수는 127자입니다.
- 단일 테이블에서 정의할 수 있는 최대 열 수는 1,600개입니다.
- 단일 테이블에서 정의할 수 있는 최대 SORTKEY 열 수는 400개입니다.

열 수준 설정 및 테이블 수준 설정의 요약

여러 속성과 설정은 열 수준 또는 테이블 수준에서 설정될 수 있습니다. 어떤 경우에는 열 수준 또는 테이블 수준에서 속성이나 제약 조건을 설정하는 효과가 동일합니다. 다른 경우에는 다른 결과를 낳습니다.

다음 목록에 열 수준 설정 및 테이블 수준 설정이 요약되어 있습니다.

DISTKEY

열 수준이나 테이블 수준 중 어떤 수준에서 설정되든 실제로는 차이가 없습니다.

DISTKEY가 설정된 경우 열 수준 또는 테이블 수준 중 하나에서 DISTSTYLE이 KEY로 설정되거나 전혀 설정되지 않아야 합니다. DISTSTYLE은 테이블 수준에서만 설정될 수 있습니다.

SORTKEY

열 수준에서 설정되는 경우 SORTKEY는 단일 열이어야 합니다. SORTKEY가 테이블 수준에서 설정된 경우 하나 이상의 열이 복합 또는 인터리브 합성 정렬 키를 구성할 수 있습니다.

COLLATE CASE_SENSITIVE | COLLATE CASE_INSENSITIVE

Amazon Redshift는 열에 대한 대/소문자 구분 구성 변경을 지원하지 않습니다. 테이블에 새 열을 추가하면 Amazon Redshift는 대/소문자 구분에 기본값을 사용합니다. Amazon Redshift는 새 열을 추가할 때 COLLATE 키워드를 지원하지 않습니다.

데이터베이스 데이터 정렬을 사용하여 데이터베이스를 생성하는 방법에 대한 자세한 내용은 데이터베이스 생성 섹션을 참조하세요.

COLLATE 함수에 대한 자세한 내용은 COLLATE 함수 섹션을 참조하세요.

UNIQUE

열 수준에서는 하나 이상의 키가 UNIQUE로 설정될 수 있고, UNIQUE 제약 조건이 각각의 열에 개별적으로 적용됩니다. UNIQUE가 테이블 수준에서 설정된 경우 하나 이상의 열이 복합 UNIQUE 제약 조건을 구성할 수 있습니다.

PRIMARY KEY

열 수준에서 설정되는 경우 PRIMARY KEY는 단일 열이어야 합니다. PRIMARY KEY가 테이블 수준에서 설정된 경우 하나 이상의 열이 복합 기본 키를 구성할 수 있습니다.

FOREIGN KEY

FOREIGN KEY가 열 수준이나 테이블 수준 중 어떤 수준에서 설정되든 실제로는 차이가 없습니다. 열 수준에서, 구문은 간단히 REFERENCES reftable [(refcolumn)]입니다.

수신 데이터의 배포

수신 데이터의 해시 배포 구성표가 대상 테이블의 해시 배포 구성표와 일치하는 경우, 데이터 로드 시 데이터를 물리적으로 배포할 필요는 실제로 없습니다. 예를 들어, 새 테이블에 대해 배포 키가 설정되고 같은 키 열에 배포된 다른 테이블에서 데이터가 삽입되고 있는 경우, 같은 노드와 조각을 사용하여

데이터가 적절히 로드됩니다. 하지만 원본 테이블과 대상 테이블이 모두 EVEN 배포로 설정되어 있는 경우 데이터는 대상 테이블로 다시 배포됩니다.

넓은 테이블

폭이 매우 넓은 테이블을 만들 수는 있겠지만 그런 테이블에서는 INSERT 또는 SELECT 문과 같은 쿼리 처리를 수행하지 못할 수도 있습니다. CHAR과 같이 열 너비가 고정된 테이블의 최대 너비는 64KB - 1(또는 65535바이트)입니다. 테이블에 VARCHAR 열이 포함되어 있는 경우, VARCHARS 열이 선언된 전체 너비를 계산된 쿼리 처리 제한에 산입하지 않기 때문에 테이블은 오류를 반환하지 않고 더 큰 값으로 선언된 너비를 가질 수 있습니다. VARCHAR 열을 포함한 유효 쿼리 처리 제한은 요소의 수에 따라 달라집니다.

테이블이 너무 넓어 삽입 또는 선택 작업을 수행할 수 없는 경우 다음 오류가 발생합니다.

```
ERROR:  8001
DETAIL:  The combined length of columns processed in the SQL statement
exceeded the query-processing limit of 65535 characters (pid:7627)
```

예시

CREATE TABLE 명령을 사용하는 방법을 보여주는 예제는 예시 주제를 참조하세요.

예시

다음 예에서는 Amazon Redshift CREATE TABLE 문에 있는 다양한 열 및 테이블 속성을 보여줍니다. 파라미터 정의를 비롯한 CREATE TABLE에 대한 자세한 내용은 CREATE TABLE 섹션을 참조하세요.

대부분의 예제는 TICKIT 샘플 데이터 세트의 테이블과 데이터를 사용합니다. 자세한 내용은 샘플 데이터베이스를 참조하세요.

CREATE TABLE 명령에서 데이터베이스 이름 및 스키마 이름을 테이블 이름 앞에 붙일 수 있습니다. 예를 들면 다음과 같습니다. `dev_database.public.sales` 데이터베이스 이름은 연결된 데이터베이스여야 합니다. 다른 데이터베이스에서 데이터베이스 객체를 만들려는 모든 시도는 실패하고 잘못된 작업 오류가 발생합니다.

배포 키, 복합 정렬 키 및 압축을 사용하여 테이블 생성

다음 예에서는 여러 열에 대해 정의된 압축을 사용하여 TICKIT 데이터베이스에서 SALES 테이블을 생성합니다. LISTID는 배포 키로 선언되고, LISTID 및 SELLERID는 다중 열 복합 정렬 키로 선언됩니다. 테이블에 대한 기본 키 및 외래 키 제약 조건 역시 정의됩니다. 제약 조건이 없는 경우 예제에서 테이블을 만들기 전에 외래 키가 참조하는 각 열에 UNIQUE 제약 조건을 추가해야 할 수 있습니다.

```
create table sales(
salesid integer not null,
listid integer not null,
sellerid integer not null,
buyerid integer not null,
eventid integer not null encode mostly16,
dateid smallint not null,
qtysold smallint not null encode mostly8,
pricepaid decimal(8,2) encode delta32k,
commission decimal(8,2) encode delta32k,
saletime timestamp,
primary key(salesid),
foreign key(listid) references listing(listid),
foreign key(sellerid) references users(userid),
foreign key(buyerid) references users(userid),
foreign key(dateid) references date(dateid))
distkey(listid)
compound sortkey(listid,sellerid);
```

결과는 다음과 같습니다.

```
schemaname | tablename | column     | type                          | encoding | distkey
 | sortkey | notnull
-----------+-----------+------------+-------------------------------+----------+---------
+---------+--------
public     | sales     | salesid    | integer                       | lzo      | false
 |       0 | true
public     | sales     | listid     | integer                       | none     | true
 |       1 | true
public     | sales     | sellerid   | integer                       | none     | false
 |       2 | true
public     | sales     | buyerid    | integer                       | lzo      | false
 |       0 | true
public     | sales     | eventid    | integer                       | mostly16 | false
 |       0 | true
public     | sales     | dateid     | smallint                      | lzo      | false
 |       0 | true
public     | sales     | qtysold    | smallint                      | mostly8  | false
 |       0 | true
public     | sales     | pricepaid  | numeric(8,2)                  | delta32k | false
 |       0 | false
public     | sales     | commission | numeric(8,2)                  | delta32k | false
 |       0 | false
```

```
public     | sales     | saletime   | timestamp without time zone | lzo      | false
 |       0 | false
```

다음 예에서는 대/소문자를 구분하지 않는 col1 열이 있는 테이블 t1을 생성합니다.

```
create table T1 (
  col1 Varchar(20) collate case_insensitive
 );

insert into T1 values ('bob'), ('john'), ('Tom'), ('JOHN'), ('Bob');
```

테이블을 쿼리합니다.

```
select * from T1 where col1 = 'John';

col1
------
 john
 JOHN
(2 rows)
```

인터리브 정렬 키를 사용하여 테이블 생성

다음 예에서는 인터리브 정렬 키를 사용하여 CUSTOMER 테이블을 생성합니다.

```
create table customer_interleaved (
  c_custkey      integer        not null,
  c_name         varchar(25)    not null,
  c_address      varchar(25)    not null,
  c_city         varchar(10)    not null,
  c_nation       varchar(15)    not null,
  c_region       varchar(12)    not null,
  c_phone        varchar(15)    not null,
  c_mktsegment      varchar(10)    not null)
diststyle all
interleaved sortkey (c_custkey, c_city, c_mktsegment);
```

IF NOT EXISTS를 사용하여 테이블 생성

다음은 CITIES 테이블을 생성하거나, 이 스키마가 이미 존재하는 경우에는 아무 것도 하지 않고 메시지를 반환하는 예입니다.

```
create table if not exists cities(
cityid integer not null,
city varchar(100) not null,
state char(2) not null);
```

ALL 배포로 테이블 생성

다음 예에서는 ALL 배포로 VENUE 테이블을 생성합니다.

```
create table venue(
venueid smallint not null,
venuename varchar(100),
venuecity varchar(30),
venuestate char(2),
venueseats integer,
primary key(venueid))
diststyle all;
```

EVEN 배포로 테이블 생성

다음 예에서는 3개의 열이 있는, MYEVENT라는 테이블을 생성합니다.

```
create table myevent(
eventid int,
eventname varchar(200),
eventcity varchar(30))
diststyle even;
```

이 테이블은 균등하게 배포되고 정렬되지 않습니다. 테이블에 선언된 DISTKEY 또는 SORTKEY 열이 없습니다.

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'myevent';

  column   |          type          | encoding | distkey | sortkey
-----------+------------------------+----------+---------+---------
 eventid   | integer                | lzo      | f       |       0
 eventname | character varying(200) | lzo      | f       |       0
 eventcity | character varying(30)  | lzo      | f       |       0
(3 rows)
```

다른 테이블과 같은 임시 테이블 생성

다음 예에서는 EVENT 테이블에서 열을 상속하는, TEMPEVENT라는 임시 테이블을 생성합니다.

```
create temp table tempevent(like event);
```

이 테이블은 상위 테이블의 DISTKEY 및 SORTKEY 속성도 상속합니다.

```
select "column", type, encoding, distkey, sortkey
 from pg_table_def where tablename = 'tempevent';

  column   |            type             | encoding | distkey | sortkey
-----------+-----------------------------+----------+---------+---------
 eventid   | integer                     | none     | t       |       1
 venueid   | smallint                    | none     | f       |       0
 catid     | smallint                    | none     | f       |       0
 dateid    | smallint                    | none     | f       |       0
 eventname | character varying(200)      | lzo      | f       |       0
 starttime | timestamp without time zone | bytedict | f       |       0
(6 rows)
```

IDENTITY 열을 포함한 테이블 생성

다음 예에서는 VENUEID로 명명된 IDENTITY 열을 가진, VENUE_IDENT라는 테이블을 생성합니다. 이 열은 0으로 시작되고 각 레코드마다 1씩 증가합니다. VENUEID는 테이블의 기본 키로도 선언됩니다.

```
create table venue_ident(venueid bigint identity(0, 1),
venuename varchar(100),
venuecity varchar(30),
venuestate char(2),
venueseats integer,
primary key(venueid));
```

기본 IDENTITY 열을 포함한 테이블 생성

다음 예제에서는 t1이라는 테이블을 생성합니다. 이 테이블에는 hist_id라는 IDENTITY 열과 base_id라는 기본 IDENTITY 열이 있습니다.

```
CREATE TABLE t1(
  hist_id BIGINT IDENTITY NOT NULL, /* Cannot be overridden */
  base_id BIGINT GENERATED BY DEFAULT AS IDENTITY NOT NULL, /* Can be overridden */
```

```
  business_key varchar(10) ,
  some_field varchar(10)
);
```

테이블에 행을 삽입하면 hist_id 및 base_id 값이 모두 생성됨을 알 수 있습니다.

```
INSERT INTO T1 (business_key, some_field) values ('A','MM');
```

```
SELECT * FROM t1;

 hist_id | base_id | business_key | some_field
---------+---------+--------------+------------
       1 |       1 | A            | MM
```

두 번째 행을 삽입하면 base_id의 기본 값이 생성됨을 알 수 있습니다.

```
INSERT INTO T1 (base_id, business_key, some_field) values (DEFAULT, 'B','MNOP');
```

```
SELECT * FROM t1;

 hist_id | base_id | business_key | some_field
---------+---------+--------------+------------
       1 |       1 | A            | MM
       2 |       2 | B            | MNOP
```

세 번째 행을 삽입하면 base_id의 값이 고유할 필요가 없음을 알 수 있습니다.

```
INSERT INTO T1 (base_id, business_key, some_field) values (2,'B','MNNN');
```

```
SELECT * FROM t1;

 hist_id | base_id | business_key | some_field
---------+---------+--------------+------------
       1 |       1 | A            | MM
       2 |       2 | B            | MNOP
       3 |       2 | B            | MNNN
```

DEFAULT 열 값을 포함한 테이블 생성

다음 예에서는 각 열에 대한 기본값을 선언하는 CATEGORYDEF 테이블을 생성합니다.

```
create table categorydef(
catid smallint not null default 0,
catgroup varchar(10) default 'Special',
catname varchar(10) default 'Other',
catdesc varchar(50) default 'Special events',
primary key(catid));

insert into categorydef values(default,default,default,default);
```

```
select * from categorydef;

 catid | catgroup | catname |    catdesc
-------+----------+---------+----------------
     0 | Special  | Other   | Special events
(1 row)
```

DISTSTYLE, DISTKEY 및 SORTKEY 옵션

다음 예에서는 DISTKEY, SORTKEY 및 DISTSTYLE 옵션의 작동 방식을 보여줍니다. 이 예에서, COL1은 분산 키이므로 분산 스타일이 KEY로 설정되거나 설정되지 않아야 합니다. 기본적으로, 이 테이블에는 정렬 키가 없으므로 테이블이 정렬되지 않습니다.

```
create table t1(col1 int distkey, col2 int) diststyle key;
```

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 't1';

column |  type   | encoding | distkey | sortkey
-------+---------+----------+---------+---------
col1   | integer | az64     | t       | 0
col2   | integer | az64     | f       | 0
```

다음 예에서는 동일한 열이 배포 키와 정렬 키로 정의되어 있습니다. 이때도 분산 스타일은 KEY로 설정되거나 설정되지 않아야 합니다.

```
create table t2(col1 int distkey sortkey, col2 int);
```

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 't2';
```

```
column |  type   | encoding | distkey | sortkey
-------+---------+----------+---------+---------
col1   | integer | none     | t       | 1
col2   | integer | az64     | f       | 0
```

다음 예에서는 분산 키로 설정되는 열이 없고 COL2는 정렬 키로 설정되고 분산 스타일은 ALL로 설정됩니다.

```
create table t3(col1 int, col2 int sortkey) diststyle all;
```

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 't3';

Column |  Type   | Encoding | DistKey | SortKey
-------+---------+----------+---------+--------
col1   | integer | az64     | f       | 0
col2   | integer | none     | f       | 1
```

다음 예에서는 배포 키가 EVEN으로 설정되고 정렬 키는 명시적으로 정의되지 않으므로, 테이블이 균등하게 배포되지만 정렬되지 않습니다.

```
create table t4(col1 int, col2 int) diststyle even;
```

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 't4';

             column |  type   |encoding | distkey | sortkey
--------+---------+---------+---------+--------
col1    | integer | az64    | f       | 0
col2    | integer | az64    | f       | 0
```

ENCODE AUTO 옵션을 사용하여 테이블 생성

다음 예에서는 자동 압축 인코딩을 사용하여 테이블 t1을 생성합니다. 열에 대한 인코딩 형식을 지정하지 않으면 ENCODE AUTO가 기본값입니다.

```
create table t1(c0 int, c1 varchar);
```

다음 예에서는 ENCODE AUTO를 지정하여 자동 압축 인코딩으로 테이블 t2를 생성합니다.

```
create table t2(c0 int, c1 varchar) encode auto;
```

다음 예에서는 ENCODE AUTO를 지정하여 자동 압축 인코딩으로 테이블 t3를 생성합니다. 열 c0은 DELTA의 초기 인코딩 형식으로 정의됩니다. 다른 인코딩이 더 나은 쿼리 성능을 제공하는 경우 Amazon Redshift에서 인코딩을 변경할 수 있습니다.

```
create table t3(c0 int encode delta, c1 varchar) encode auto;
```

다음 예에서는 ENCODE AUTO를 지정하여 자동 압축 인코딩으로 테이블 t4를 생성합니다. 열 c0은 초기 인코딩 DELTA로 정의되고, 열 c1은 LZO의 초기 인코딩으로 정의된다. 다른 인코딩이 더 나은 쿼리 성능을 제공하는 경우 Amazon Redshift에서 이러한 인코딩을 변경할 수 있습니다.

```
create table t4(c0 int encode delta, c1 varchar encode lzo) encode auto;
```

CREATE TABLE AS

주제

쿼리를 기반으로 새 테이블을 만듭니다. 이 테이블의 소유자는 명령을 발행하는 사용자입니다.

명령에서 쿼리에 의해 정의된 데이터와 함께 새 테이블이 로드됩니다. 테이블 열의 이름과 데이터 형식은 쿼리의 출력 열과 연결되어 있습니다. CREATE TABLE AS(CTAS) 명령을 실행하면 새 테이블이 생성되고 새 테이블을 로드하는 쿼리가 평가됩니다.

구문

```
CREATE [ [ LOCAL ] { TEMPORARY | TEMP } ]
TABLE table_name
[ ( column_name [, ... ] ) ]
[ BACKUP { YES | NO } ]
[ table_attributes ]
AS query

where table_attributes are:
```

```
[ DISTSTYLE { AUTO | EVEN | ALL | KEY } ]
[ DISTKEY( distkey_identifier ) ]
[ [ COMPOUND | INTERLEAVED ] SORTKEY( column_name [, ...] ) ]
```

파라미터

LOCAL

이 선택적 키워드는 문에서 허용되지만, Amazon Redshift에는 아무런 효과도 없습니다.

TEMPORARY | TEMP

임시 테이블을 만듭니다. 임시 테이블은 자신이 생성된 세션이 끝날 때 자동으로 삭제됩니다.

table_name

생성할 테이블의 이름입니다.

> **Important**
>
> '#'으로 시작하는 테이블 이름을 지정하면 테이블이 임시 테이블로 생성됩니다. 예:
>
> ```
> create table #newtable (id) as select * from oldtable;
> ```

최대 테이블 이름 길이는 127바이트이며, 이보다 긴 이름은 127바이트까지 표시되고 나머지는 잘립니다. Amazon Redshift는 노드 유형별로 클러스터당 테이블 수 할당량을 적용합니다. 다음 테이블에서 보듯이, 데이터베이스 및 스키마 이름으로 테이블 이름을 정규화할 수 있습니다.

```
create table tickit.public.test (c1) as select * from oldtable;
```

이 예에서는 `tickit`이 데이터베이스 이름이고, `public`이 스키마 이름입니다. 데이터베이스 또는 스키마가 존재하지 않는 경우 테이블이 생성되지 않으며, 이 명령문은 오류를 반환합니다.

스키마 이름이 주어지는 경우 새 테이블은 그 스키마에서 생성됩니다(생성자가 해당 스키마에 액세스할 수 있다고 가정). 테이블 이름은 그 스키마에 대한 고유한 이름이어야 합니다. 아무런 스키마도 지정되지 않으면 현재 데이터베이스 스키마를 사용하여 테이블이 생성됩니다. 임시 테이블을 생성할 경우, 임시 테이블이 특수한 스키마에 존재하므로 스키마 이름을 지정할 수 없습니다.

같은 이름을 가진 여러 임시 테이블이 각각 별도의 세션에서 생성되는 경우에는 같은 데이터베이스에 동시에 존재하도록 허용됩니다. 이러한 테이블은 서로 다른 스키마에 할당됩니다.

column_name

새 테이블에 있는 열의 이름입니다. 열 이름을 입력하지 않은 경우에는 쿼리의 출력 열 이름에서 열 이름을 따옵니다. 기본 열 이름은 표현식에 사용됩니다. 유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요.

BACKUP { YES | NO }

자동 및 수동 클러스터 스냅샷에 테이블을 포함해야 할지 여부를 지정하는 절입니다.

중요한 데이터를 포함하지 않는 스테이징 테이블과 같은 테이블의 경우 BACKUP NO를 지정하여 스냅샷을 생성하고 스냅샷으로부터 복원할 때의 처리 시간을 절약하고 Amazon Simple Storage Service의 스토리지 공간을 줄입니다. BACKUP NO 설정은 클러스터 내에 있는 다른 노드로의 데이터 자동 복제에 아무런 영향도 미치지 않으므로, 노드 장애가 발생할 경우 BACKUP NO가 지정된 테이블이 복원됩니다. 기본값은 BACKUP YES입니다.

DISTSTYLE { AUTO | EVEN | KEY | ALL }

전체 테이블의 데이터 배포 스타일을 정의하는 키워드입니다. Amazon Redshift는 테이블에 대해 지정된 배포 스타일에 따라 컴퓨팅 노드에 테이블의 행을 배포합니다. 기본값은 DISTSTYLE AUTO입니다.

테이블에 대해 선택하는 분산 스타일이 데이터베이스의 전체 성능에 영향을 미칩니다. 자세한 내용은 쿼리 최적화를 위한 데이터 배포 단원을 참조하십시오.

- AUTO: Amazon Redshift에서 테이블 데이터를 기반으로 최적의 배포 스타일을 할당합니다. 테이블에 적용된 분산 스타일을 보려면 PG_CLASS 시스템 카탈로그 테이블을 쿼리합니다. 자세한 내용은 분산 스타일 보기 단원을 참조하십시오.
- EVEN: 테이블의 데이터가 클러스터의 노드들에 걸쳐 라운드 로빈 배포 방식으로 균등하게 분포됩니다. 행 ID는 배포의 결정에 사용되며, 대략적으로 같은 수의 행이 각 노드로 배포됩니다. 이것이 기본 배포 방법입니다.
- KEY: 데이터가 DISTKEY 열에 있는 값을 기준으로 배포됩니다. 조인 테이블의 조인 열을 배포 키로 설정하면 두 테이블 모두의 조인 행이 컴퓨팅 노드에 배치됩니다. 데이터가 배치되면 최적화 프로그램이 조인을 더 효율적으로 수행할 수 있습니다. DISTSTYLE KEY를 지정하는 경우 DISTKEY 열의 이름을 지정해야 합니다.
- ALL: 전체 테이블의 복사본이 모든 노드로 배포됩니다. 이 분산 스타일은 임의의 조인에 필요한 모든 행을 모든 노드에서 사용할 수 있도록 보장하지만, 스토리지 요구량이 몇 배로 늘고 테이블의 로드 시간과 유지 관리 시간도 증가합니다. ALL 배포는 KEY 배포가 적절하지 않은 특정 차원 테이블과 함께 사용할 때 실행 시간을 개선할 수 있지만, 성능 개선 정도를 유지 관리 비용과 비교 검토해야 합니다.

DISTKEY (column)

배포 키에 대해 열 이름이나 위치 번호를 지정합니다. 테이블에 대한 선택적 열 목록 또는 쿼리의 선택 목록 중 하나에 지정된 이름을 사용합니다. 또는 첫 번째로 선택한 열이 1, 두 번째로 선택한 열이 2 등과 같이 이어지는 경우에는 위치 번호를 사용합니다. 테이블에서는 한 개의 열만 배포 키일 수 있습니다.

- 어떤 열을 DISTKEY 열로 선언하는 경우 DISTSTYLE이 KEY로 설정되거나 전혀 설정되지 않아야 합니다.
- 열을 DISTKEY 열로 선언하지 않은 경우 DISTSTYLE을 EVEN으로 설정할 수 있습니다.
- DISTKEY 또는 DISTSTYLE을 지정하지 않으면 CTAS가 SELECT 절의 쿼리 계획을 기반으로 새 테이블의 분산 스타일을 결정합니다. 자세한 내용은 열 및 테이블 속성의 상속 단원을 참조하십시오.

배포 키 및 정렬 키와 동일한 열을 정의할 수 있습니다. 이 접근 방식은 문제가 되는 열이 쿼리에서 조인 열일 때 조인을 가속화하는 경향이 있습니다.

[COMPOUND | INTERLEAVED] SORTKEY (column_name [, ...])

테이블에 대해 하나 이상의 정렬 키를 지정합니다. 데이터가 테이블로 로드될 때 데이터는 정렬 키로 지정되는 열을 기준으로 정렬됩니다.

COMPOUND 또는 INTERLEAVED 정렬 스타일을 선택적으로 지정할 수 있습니다. 기본 키워드는 COMPOUND입니다. 자세한 내용은 정렬 키 단원을 참조하십시오.

테이블당 최대 400개의 COMPOUND SORTKEY 열 또는 8개의 INTERLEAVED SORTKEY 열을 정의할 수 있습니다.

SORTKEY를 지정하지 않으면 CTAS가 SELECT 절의 쿼리 계획을 기반으로 새 테이블의 정렬 키를 결정합니다. 자세한 내용은 열 및 테이블 속성의 상속 단원을 참조하십시오.

COMPOUND

목록에 표시되는 모든 열로 구성된 복합 키를 사용하여 데이터가 나열되는 순서대로 정렬됨을 지정합니다. 복합 정렬 키는 쿼리가 정렬 열의 순서에 따라 행을 스캔할 때 가장 유용합니다. 쿼리가 보조 정렬 열에 의존할 때는 복합 키로 정렬함으로써 얻는 성능상의 이점이 감소합니다. 테이블당 최대 400개의 COMPOUND SORTKEY 열을 정의할 수 있습니다.

INTERLEAVED

데이터가 인터리브 정렬 키를 사용하여 정렬됨을 지정합니다. 인터리브 정렬 키에 대해 최대 8개의 열을 지정할 수 있습니다.

인터리브 정렬에서는 정렬 키에서 각 열이나 열의 하위 집합에 똑같은 가중치를 부여하므로, 쿼리가 정렬 키에 있는 열의 순서에 종속되지 않습니다. 쿼리가 하나 이상의 보조 정렬 열을 사용하는 경우 인터리브 정렬은 쿼리 성능을 크게 높여줍니다. 인터리브 정렬에는 데이터 로드와 vacuum 작업에 약간의 오버헤드 비용이 수반됩니다.

AS query

Amazon Redshift가 지원하는 쿼리(SELECT 문)입니다.

CTAS 사용 시 주의 사항

Limits

Amazon Redshift는 노드 유형별로 클러스터당 테이블 수 할당량을 적용합니다.

테이블 이름으로 입력할 수 있는 최대 문자 수는 127자입니다.

단일 테이블에서 정의할 수 있는 최대 열 수는 1,600개입니다.

열 및 테이블 속성의 상속

CREATE TABLE AS(CTAS) 테이블은 자신이 생성된 테이블로부터 제약 조건, 자격 증명 열, 기본 열 값 또는 기본 키를 상속하지 않습니다.

CTAS 테이블에 대한 열 압축 인코딩을 지정할 수 없습니다. Amazon Redshift가 다음과 같이 압축 인코딩을 자동으로 할당합니다.

- 정렬 키로 정의된 열은 RAW 압축이 할당됩니다.
- BOOLEAN, REAL, DOUBLE PRECISION, GEOMETRY 또는 GEOGRAPHY 데이터 유형으로 정의된 열은 RAW 압축이 할당됩니다.
- SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIME, TIMETZ, TIMESTAMP 또는 TIMESTAMPTZ로 정의된 열에는 AZ64 압축이 할당됩니다.
- CHAR, VARCHAR 또는 VARBYTE로 정의된 열에는 LZO 압축이 할당됩니다.

자세한 내용은 압축 인코딩 및 데이터 타입 단원을 참조하세요.

열 인코딩을 명시적으로 할당하려면 CREATE TABLE을 사용하세요.

CTAS가 SELECT 절의 쿼리 계획을 기반으로 새 테이블의 분산 스타일과 정렬 키를 결정합니다.

조인, 집계, order by 절 또는 limit 절을 포함하는 쿼리와 같은 복잡한 쿼리인 경우, CTAS는 쿼리 계획을 기반으로 최적의 분산 스타일과 정렬 키를 선택하기 위해 최대한 시도합니다.

> **Note**
>
> 큰 데이터 집합 또는 복잡한 쿼리로 최상의 성능을 얻으려면 일반적인 데이터 집합을 사용하여 테스트하는 것이 좋습니다.

쿼리 최적화 프로그램이 데이터 정렬과 배포를 위해 선택하는 열이 있을 경우 어떤 열을 선택할지 살펴보는 쿼리 계획을 검사함으로써 CTAS가 선택할 배포 키와 정렬 키를 예측할 수 있는 경우가 종종 있습니다. 쿼리 계획의 최상위 노드가 단일 테이블(XN Seq Scan)의 간단한 순차적 스캔인 경우에는 CTAS가 일반적으로 원본 테이블의 분산 스타일과 정렬 키를 사용합니다. 쿼리 계획의 최상위 노드가 다른 순차적 스캔(예: XN Limit, XN Sort, XN HashAggregate 등)인 경우, CTAS는 쿼리 계획을 기반으로 최적의 분산 스타일과 정렬 키를 선택하기 위해 최대한 시도합니다.

예를 들어, 다음 유형의 SELECT 절을 사용하여 5개의 테이블을 만든다고 가정해봅시다.

- 간단한 select 문
- limit 절
- LISTID를 사용하는 order by 절
- QTYSOLD를 사용하는 order by 절
- group by 절을 사용하는 SUM 집계 함수

다음 예에서는 각 CTAS 문에 대한 쿼리 계획을 보여 줍니다.

```
explain create table sales1_simple as select listid, dateid, qtysold from sales;
                           QUERY PLAN
-----------------------------------------------------------------
 XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=8)
(1 row)


explain create table sales2_limit as select listid, dateid, qtysold from sales limit
 100;
                              QUERY PLAN
----------------------------------------------------------------------
 XN Limit  (cost=0.00..1.00 rows=100 width=8)
   ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=8)
```

```
(2 rows)


explain create table sales3_orderbylistid as select listid, dateid, qtysold from sales
 order by listid;
                              QUERY PLAN
----------------------------------------------------------------------
 XN Sort  (cost=1000000016724.67..1000000017155.81 rows=172456 width=8)
   Sort Key: listid
   ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=8)
(3 rows)


explain create table sales4_orderbyqty as select listid, dateid, qtysold from sales
 order by qtysold;
                              QUERY PLAN
----------------------------------------------------------------------
 XN Sort  (cost=1000000016724.67..1000000017155.81 rows=172456 width=8)
   Sort Key: qtysold
   ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=8)
(3 rows)


explain create table sales5_groupby as select listid, dateid, sum(qtysold) from sales
 group by listid, dateid;
                              QUERY PLAN
----------------------------------------------------------------------
 XN HashAggregate  (cost=3017.98..3226.75 rows=83509 width=8)
   ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=8)
(2 rows)
```

각 테이블에 대한 배포 키와 정렬 키를 보려면 다음에 표시된 것처럼 PG_TABLE_DEF 시스템 카탈로그 테이블을 쿼리합니다.

```
select * from pg_table_def where tablename like 'sales%';

      tablename       |   column   | distkey | sortkey
----------------------+------------+---------+---------
 sales                | salesid    | f       |       0
 sales                | listid     | t       |       0
 sales                | sellerid   | f       |       0
 sales                | buyerid    | f       |       0
 sales                | eventid    | f       |       0
```

```
 sales                | dateid     | f       |       1
 sales                | qtysold    | f       |       0
 sales                | pricepaid  | f       |       0
 sales                | commission | f       |       0
 sales                | saletime   | f       |       0
 sales1_simple        | listid     | t       |       0
 sales1_simple        | dateid     | f       |       1
 sales1_simple        | qtysold    | f       |       0
 sales2_limit         | listid     | f       |       0
 sales2_limit         | dateid     | f       |       0
 sales2_limit         | qtysold    | f       |       0
 sales3_orderbylistid | listid     | t       |       1
 sales3_orderbylistid | dateid     | f       |       0
 sales3_orderbylistid | qtysold    | f       |       0
 sales4_orderbyqty    | listid     | t       |       0
 sales4_orderbyqty    | dateid     | f       |       0
 sales4_orderbyqty    | qtysold    | f       |       1
 sales5_groupby       | listid     | f       |       0
 sales5_groupby       | dateid     | f       |       0
 sales5_groupby       | sum        | f       |       0
```

다음 표에 결과가 요약되어 있습니다. 단순성을 기하기 위해 설명 계획에서 비용, 행 및 너비 세부 정보는 생략합니다.

표	CTAS SELECT 문	설명 계획 최상위 노드	배포 키	정렬 키
S1_SIMPLE	`select listid, dateid, qtysold from sales`	`XN Seq Scan on sales ...`	LISTID	DATEID
S2_LIMIT	`select listid, dateid, qtysold from sales limit 100`	`XN Limit ...`	없음(EVEN)	없음
S3_ORDER_BY_LISTID	`select listid, dateid, qtysold from sales order by listid`	`XN Sort ...` `Sort Key: listid`	LISTID	LISTID

표	CTAS SELECT 문	설명 계획 최상위 노드	배포 키	정렬 키
S4_ORDER_BY_QTY	`select listid, dateid, qtysold from sales order by qtysold`	`XN Sort ...` `Sort Key: qtysold`	LISTID	QTYSOLD
S5_GROUP_BY	`select listid, dateid, sum(qtysold) from sales group by listid, dateid`	`XN HashAggregate ...`	없음(EVEN)	없음

CTAS 문에서 분산 스타일과 정렬 키를 명시적으로 지정할 수 있습니다. 예를 들어, 다음 문은 EVEN 분산을 사용하여 테이블을 생성하고 SALESID를 정렬 키로 지정합니다.

```
create table sales_disteven
diststyle even
sortkey (salesid)
as
select eventid, venueid, dateid, eventname
from event;
```

압축 인코딩

ENCODE AUTO는 테이블의 기본값으로 사용됩니다. Amazon Redshift는 테이블의 모든 열에 대한 압축 인코딩을 자동으로 관리하지 않습니다.

수신 데이터의 배포

수신 데이터의 해시 배포 구성표가 대상 테이블의 해시 배포 구성표와 일치하는 경우, 데이터 로드 시 데이터를 물리적으로 배포할 필요는 실제로 없습니다. 예를 들어, 새 테이블에 대해 배포 키가 설정되고 같은 키 열에 배포된 다른 테이블에서 데이터가 삽입되고 있는 경우, 같은 노드와 조각을 사용하여 데이터가 적절히 로드됩니다. 하지만 원본 테이블과 대상 테이블이 모두 EVEN 배포로 설정되어 있는 경우 데이터는 대상 테이블로 다시 배포됩니다.

자동 ANALYZE 작업

Amazon Redshift는 CTAS 명령으로 생성하는 테이블을 자동으로 분석합니다. 이 테이블들이 처음 생성될 때는 테이블에서 ANALYZE 명령을 실행할 필요가 없습니다. 테이블을 수정하는 경우, 다른 테이블과 같은 방법으로 분석해야 합니다.

CTAS 예

다음 예에서는 EVENT 테이블에 대해 EVENT_BACKUP이라는 테이블을 생성합니다.

```
create table event_backup as select * from event;
```

결과 테이블은 EVENT 테이블에서 배포 키와 정렬 키를 상속합니다.

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'event_backup';

column    | type                        | encoding | distkey | sortkey
----------+-----------------------------+----------+---------+--------
catid     | smallint                    | none     | false   |       0
dateid    | smallint                    | none     | false   |       1
eventid   | integer                     | none     | true    |       0
eventname | character varying(200)      | none     | false   |       0
starttime | timestamp without time zone | none     | false   |       0
venueid   | smallint                    | none     | false   |       0
```

다음 명령은 EVENT 테이블에서 4개의 열을 선택하여 EVENTDISTSORT라는 새 테이블을 생성합니다. 새 테이블은 EVENTID를 기준으로 배포되고 EVENTID 및 DATEID를 기준으로 정렬됩니다.

```
create table eventdistsort
distkey (1)
sortkey (1,3)
as
select eventid, venueid, dateid, eventname
from event;
```

결과는 다음과 같습니다.

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'eventdistsort';
```

```
column   |          type          | encoding | distkey | sortkey
---------+------------------------+----------+---------+-------
eventid   | integer                | none     | t       | 1
venueid   | smallint               | none     | f       | 0
dateid    | smallint               | none     | f       | 2
eventname | character varying(200)| none     | f       | 0
```

배포 키 및 정렬 키에 대한 열 이름을 사용하여 같은 테이블을 정확하게 생성할 수 있습니다. 예:

```
create table eventdistsort1
distkey (eventid)
sortkey (eventid, dateid)
as
select eventid, venueid, dateid, eventname
from event;
```

다음 문은 테이블에 균등 배포를 적용하지만 명시적인 정렬 키를 정의하지 않습니다.

```
create table eventdisteven
diststyle even
as
select eventid, venueid, dateid, eventname
from event;
```

새 테이블에 대해 EVEN 배포가 지정되어 있으므로 테이블이 EVENT 테이블(EVENTID)에서 정렬 키를 상속하지 않습니다. 새 테이블에는 정렬 키와 배포 키가 없습니다.

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'eventdisteven';

column    |          type          | encoding | distkey | sortkey
----------+------------------------+----------+---------+---------
eventid   | integer                | none     | f       | 0
venueid   | smallint               | none     | f       | 0
dateid    | smallint               | none     | f       | 0
eventname | character varying(200) | none     | f       | 0
```

다음 문은 균등 배포를 적용하고 정렬 키를 정의합니다.

```
create table eventdistevensort diststyle even sortkey (venueid)
as select eventid, venueid, dateid, eventname from event;
```

결과 테이블에 정렬 키는 있지만 배포 키는 없습니다.

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'eventdistevensort';

column    |          type          | encoding | distkey | sortkey
----------+------------------------+----------+---------+-------
eventid   | integer                | none     | f       | 0
venueid   | smallint               | none     | f       | 1
dateid    | smallint               | none     | f       | 0
eventname | character varying(200) | none     | f       | 0
```

다음 문은 EVENTID 열에 정렬되어 있는 수신 데이터에서 다른 키 열에 EVENT 테이블을 다시 배포하고 SORTKEY 열은 정의하지 않으므로, 테이블이 정렬되지 않습니다.

```
create table venuedistevent distkey(venueid)
as select * from event;
```

결과는 다음과 같습니다.

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'venuedistevent';

 column   |            type             | encoding | distkey | sortkey
----------+-----------------------------+----------+---------+-------
eventid   | integer                     | none     | f       | 0
venueid   | smallint                    | none     | t       | 0
catid     | smallint                    | none     | f       | 0
dateid    | smallint                    | none     | f       | 0
eventname | character varying(200)      | none     | f       | 0
starttime | timestamp without time zone | none     | f       | 0
```

사용자 생성

새 데이터베이스 사용자를 생성합니다. 데이터베이스 사용자는 권한 및 역할에 따라 데이터베이스에서 데이터를 검색하고, 명령을 실행하고, 기타 작업을 수행할 수 있습니다. 이 명령을 실행하려면 데이터베이스 슈퍼 사용자여야 합니다.

필수 권한

CREATE USER에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- CREATE USER 권한이 있는 사용자

구문

```
CREATE USER name [ WITH ]
PASSWORD { 'password' | 'md5hash' | 'sha256hash' | DISABLE }
[ option [ ... ] ]

where option can be:

CREATEDB | NOCREATEDB
| CREATEUSER | NOCREATEUSER
| SYSLOG ACCESS { RESTRICTED | UNRESTRICTED }
| IN GROUP groupname [, ... ]
| VALID UNTIL 'abstime'
| CONNECTION LIMIT { limit | UNLIMITED }
| SESSION TIMEOUT limit
| EXTERNALID external_id
```

파라미터

이름

생성할 사용자의 이름입니다. 사용자 이름은 PUBLIC일 수 없습니다. 유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요.

WITH

선택적 키워드입니다. WITH는 Amazon Redshift에서 무시됩니다.

PASSWORD { 'password' | 'md5hash' | 'sha256hash' | DISABLE }

사용자의 암호를 설정합니다.

기본적으로, 암호가 비활성화 상태가 아닌 이상 사용자는 암호를 변경할 수 있습니다. 사용자의 암호를 비활성화하려면 DISABLE을 지정하십시오. 사용자 암호를 비활성화하면 시스템에서 해당 암호가 삭제되고 사용자는 임시 AWS Identity and Access Management(IAM) 사용자 자격 증명만 이용해 로그인할 수 있습니다. 자세한 내용은 IAM 인증을 이용한 데이터베이스 사용자 자격 증명 생성을 참조하십시오. 수퍼유저만이 암호를 활성화 또는 비활성화할 수 있습니다. 수퍼유저의 암호를 비활성화할 수는 없습니다. 암호를 활성화하려면 ALTER USER 단원을 실행하고 암호를 지정하십시오.

암호를 일반 텍스트, MD5 해시 문자열 또는 SHA256 해시 문자열로 지정할 수 있습니다.

> **Note**
>
> AWS Management Console, AWS CLI 또는 Amazon Redshift API를 사용하여 새 클러스터를 시작할 때 초기 데이터베이스 사용자를 위한 일반 텍스트 암호를 입력해야 합니다. 이후에 ALTER USER를 사용하여 암호를 변경할 수 있습니다.

일반 텍스트의 경우, 암호는 다음 제약 조건을 충족해야 합니다.

- 8~64자 사이의 길이여야 합니다.
- 최소한 대문자 1개, 소문자 1개 및 숫자 1개를 포함해야 합니다.
- '(작은따옴표), "(큰따옴표), \, / 또는 @을 제외하고 모든 ASCII 문자(ASCII 코드 33~126)를 사용할 수 있습니다.

CREATE USER 암호 파라미터를 일반 텍스트로 전달하는 대신 사용할 수 있는 더 안전한 방법으로, 암호와 사용자 이름을 포함하는 문자열의 MD5 해시를 지정할 수 있습니다.

> **Note**
>
> MD5 해시 문자열을 지정하면 CREATE USER 명령이 유효한 MD5 해시 문자열이 있는지 확인하지만 그 문자열에서 암호 부분의 유효성을 검사하지는 않습니다. 이 경우에는 빈 문자열과 같은 암호를 생성할 수 있는데, 이 암호로 데이터베이스에 로그인할 수는 없습니다.

MD5 암호를 지정하려면 다음 단계를 따르십시오.

1. 암호와 사용자 이름을 연결합니다.

   예를 들어, 암호가 `ez`이고 사용자가 `user1`인 경우 연결된 문자열은 `ezuser1`입니다.

2. 연결된 문자열을 32자의 MD5 해시 문자열로 변환합니다. MD5 유틸리티를 사용하여 해시 문자열을 생성할 수 있습니다. 다음 예에서는 Amazon Redshift MD5 함수 및 연결 연산자(||)를 사용하여 32자의 MD5 해시 문자열을 반환합니다.

```
select md5('ez' || 'user1');

md5
----------------------------------
```

```
153c434b4b77c89e6b94f12c5393af5b
```

3. MD5 해시 문자열 앞에 'md5'를 연결하고 연결된 문자열을 md5hash 인수로 제공합니다.

```
create user user1 password 'md5153c434b4b77c89e6b94f12c5393af5b';
```

4. 로그인 자격 증명을 사용하여 데이터베이스에 로그인합니다.

   이 예의 경우 암호 user1를 사용하여 ez로 로그인합니다.

또 다른 안전한 대안은 암호 문자열의 SHA-256 해시를 지정하는 것입니다. 또는 사용자 고유의 유효한 SHA-256 다이제스트 및 다이제스트를 생성하는 데 사용된 256비트 솔트를 제공할 수 있습니다.

- 다이제스트 – 해싱 함수의 출력입니다.
- 솔트 – 해시 함수 출력의 패턴을 줄이는 데 도움이 되도록 암호와 결합되는 무작위로 생성된 데이터입니다.

```
'sha256|Mypassword'
```

```
'sha256|digest|256-bit-salt'
```

다음 예에서 Amazon Redshift는 솔트를 생성하고 관리합니다.

```
CREATE USER admin PASSWORD 'sha256|Mypassword1';
```

다음 예에서는 유효한 SHA-256 다이제스트 및 다이제스트를 생성하는 데 사용된 256비트 솔트가 제공됩니다.

비밀번호를 지정하고 자체 솔트로 해시하려면 다음 단계를 따릅니다.

1. 256비트 솔트를 생성합니다. 16진수 문자열 생성기를 사용하여 64자 길이의 문자열을 생성하면 솔트를 확보할 수 있습니다. 이 예제에서 솔트는 c721bff5d9042cf541ff7b9d48fa8a6e545c19a763e3710151f9513038b0f6c6입니다.
2. FROM_HEX 함수를 사용하여 솔트를 바이너리로 변환합니다. SHA2 함수에는 솔트의 바이너리 표현이 필요하기 때문입니다. 다음 문을 참조하십시오.

```
SELECT
 FROM_HEX('c721bff5d9042cf541ff7b9d48fa8a6e545c19a763e3710151f9513038b0f6c6');
```

3. CONCAT 함수를 사용하여 솔트를 비밀번호에 추가하십시오. 이 예제의 암호는 Mypassword1입니다. 다음 문을 참조하십시오.

```
SELECT
 CONCAT('Mypassword1',FROM_HEX('c721bff5d9042cf541ff7b9d48fa8a6e545c19a763e3710151f951303
```

4. SHA2 함수를 사용하여 암호와 솔트 조합으로 다이제스트를 만들 수 있습니다. 다음 문을 참조하십시오.

```
SELECT
 SHA2(CONCAT('Mypassword1',FROM_HEX('c721bff5d9042cf541ff7b9d48fa8a6e545c19a763e3710151f95
 0);
```

5. 이전 단계의 다이제스트와 솔트를 사용하여 사용자를 생성합니다. 다음 문을 참조하십시오.

```
CREATE USER admin PASSWORD 'sha256|
821708135fcc42eb3afda85286dee0ed15c2c461d000291609f77eb113073ec2|
c721bff5d9042cf541ff7b9d48fa8a6e545c19a763e3710151f9513038b0f6c6';
```

6. 로그인 자격 증명을 사용하여 데이터베이스에 로그인합니다.

   이 예의 경우 암호 admin를 사용하여 Mypassword1로 로그인합니다.

해싱 함수를 지정하지 않고 암호를 일반 텍스트로 설정하면 사용자 이름을 솔트로 사용하여 MD5 다이제스트가 생성됩니다.

CREATEDB | NOCREATEDB

CREATEDB 옵션을 사용하여 신규 사용자가 새 데이터베이스를 만들 수 있습니다. 기본값은 NOCREATEDB입니다.

CREATEUSER | NOCREATEUSER

CREATEUSER 옵션으로 CREATE USER를 포함한 모든 데이터베이스 권한을 가진 수퍼유저를 생성합니다. 기본값은 NOCREATEUSER입니다. 자세한 내용은 superuser 단원을 참조하십시오.

SYSLOG ACCESS { RESTRICTED | UNRESTRICTED }

Amazon Redshift 시스템 테이블 및 뷰에 대한 사용자의 액세스 수준을 지정합니다.

SYSLOG ACCESS RESTRICTED 권한이 있는 일반 사용자는 사용자 가시성 시스템 테이블 및 뷰에서 해당 사용자가 생성한 행만 볼 수 있습니다. 기본값은 RESTRICTED입니다.

SYSLOG ACCESS UNRESTRICTED 권한이 있는 일반 사용자는 사용자 가시성 시스템 테이블 및 뷰에서 다른 사용자가 생성한 행을 포함한 모든 행을 볼 수 있습니다. UNRESTRICTED는 수퍼유저 가시성 테이블에 대한 일반 사용자 액세스를 부여하지 않습니다. 수퍼유저만 수퍼유저 가시성 테이블을 볼 수 있습니다.

> **Note**
>
> 사용자에게 시스템 테이블에 대한 무제한 액세스를 제공하면 다른 사용자가 생성한 데이터에 대한 가시성이 사용자에게 제공됩니다. 예를 들어, STL_QUERY 및 STL_QUERYTEXT에는 INSERT, UPDATE 및 DELETE 문의 전체 텍스트가 포함되며, 여기에는 사용자가 생성한 민감한 데이터가 포함될 수 있습니다.

SVV_TRANSACTIONS의 모든 행은 모든 사용자에게 표시됩니다.

자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

IN GROUP groupname

사용자가 속한 기존 그룹의 이름을 지정합니다. 여러 개의 그룹 이름이 목록에 표시될 수 있습니다.

VALID UNTIL abstime

VALID UNTIL 옵션은 그 시간 이후로는 사용자 암호가 더 이상 유효하지 않게 되는 절대 시간을 설정합니다. 기본적으로 암호에는 제한 시간이 없습니다.

CONNECTION LIMIT { limit | UNLIMITED }

사용자가 동시에 열어놓을 수 있는 데이터베이스 연결의 최대 개수입니다. 슈퍼 사용자에 대해서는 이 제한이 적용되지 않습니다. 최대 동시 연결 수를 허용하려면 UNLIMITED 키워드를 사용하십시오. 각 데이터베이스에 대한 연결 개수 제한이 적용될 수도 있습니다. 자세한 내용은 데이터베이스 생성 단원을 참조하십시오. 기본값은 UNLIMITED입니다. 현재 연결을 보려면 STV_SESSIONS 시스템 뷰를 쿼리하십시오.

> **Note**
>
> 사용자 및 데이터베이스 연결 제한이 모두 적용되는 경우 사용되지 않는 연결 슬롯은 사용자가 연결 시도 시 양쪽 제한 범위 내에서 모두 사용 가능해야 합니다.

SESSION TIMEOUT limit

세션이 비활성 또는 유휴 상태로 유지되는 최대 시간(초)입니다. 범위는 60초(1분)~1,728,000초(20일)입니다. 사용자에 대해 세션 시간 제한이 설정되지 않은 경우 클러스터 설정이 적용됩니다. 자세한 내용은 Amazon Redshift 관리 가이드의 Amazon Redshift의 할당량 및 제한 섹션을 참조하세요.

세션 시간 제한을 설정하면 새 세션에만 적용됩니다.

시작 시간, 사용자 이름 및 세션 시간 제한을 포함하여 활성 사용자 세션에 대한 정보를 보려면 STV_SESSIONS 시스템 뷰를 쿼리합니다. 사용자 세션 기록에 대한 정보를 보려면 STL_SESSIONS 뷰를 쿼리합니다. 세션 시간 제한 값을 포함하여 데이터베이스 사용자에 대한 정보를 검색하려면 SVL_USER_INFO 뷰를 쿼리합니다.

EXTERNALID external_id

자격 증명 공급자와 연결된 사용자의 식별자입니다. 사용자는 암호를 비활성화해야 합니다. 자세한 내용은 Amazon Redshift용 네이티브 자격 증명 공급자(IdP) 페더레이션을 참조하세요.

사용 노트

기본적으로, 모든 사용자는 PUBLIC 스키마에 대해 CREATE 및 USAGE 권한이 있습니다. 사용자가 데이터베이스의 PUBLIC 스키마에 객체를 만들도록 허용하지 않으려면 REVOKE 명령을 사용하여 그 권한을 제거하십시오.

IAM 인증을 이용해 데이터베이스 사용자 자격 증명을 만들 때는 임시 자격 증명만을 이용해 로그인할 수 있는 수퍼유저를 만들 수 있습니다. 수퍼유저의 암호를 비활성화할 수는 없지만 임의로 생성되는 MD5 해시 문자열을 이용해 알 수 없는 암호를 만들 수는 있습니다.

```
create user iam_superuser password 'md5A1234567890123456780123456789012' createuser;
```

큰따옴표로 묶여 있는 사용자 이름의 대/소문자는 enable_case_sensitive_identifier 구성 옵션 설정과 관계없이 항상 유지됩니다. 자세한 내용은 enable_case_sensitive_identifier 단원을 참조하십시오.

예시

다음 명령은 이름이 dbuser이고 암호가 "abcD1234"이며 데이터베이스 생성 권한이 있고 연결 제한이 30인 사용자를 생성합니다.

```
create user dbuser with password 'abcD1234' createdb connection limit 30;
```

PG_USER_INFO 카탈로그 테이블을 쿼리하여 데이터베이스 사용자에 대한 세부 정보를 봅니다.

```
select * from pg_user_info;

 usename   | usesysid | usecreatedb | usesuper | usecatupd | passwd   | valuntil |
 useconfig | useconnlimit
-----------+----------+-------------+----------+-----------+----------+----------
+-----------+-------------
 rdsdb     |        1 | true        | true     | true      | ******** | infinity |
      |
 adminuser |      100 | true        | true     | false     | ******** |          |
      | UNLIMITED
 dbuser    |      102 | true        | false    | false     | ******** |          |
      | 30
```

다음 예에서는 계정 암호가 2017년 6월 10일까지 유효합니다.

```
create user dbuser with password 'abcD1234' valid until '2017-06-10';
```

다음 예에서는 특수 문자를 포함하고 대/소문자를 구분하는 암호를 가진 사용자를 생성합니다.

```
create user newman with password '@AbC4321!';
```

MD5 암호에 백래시('\')를 사용하려면 원본 문자열에 있는 백래시로 백래시를 이스케이프하십시오. 다음 예에서는 이름이 `slashpass`이고 암호가 단일 백래시('\')인 사용자를 생성합니다.

```
select md5('\\'||'slashpass');

md5
----------------------------------
0c983d1a624280812631c5389e60d48c
```

md5 비밀번호로 사용자를 생성합니다.

```
create user slashpass password 'md50c983d1a624280812631c5389e60d48c';
```

다음 예에서는 유휴 세션 시간 제한이 120초로 설정된 `dbuser`라는 사용자를 생성합니다.

```
CREATE USER dbuser password 'abcD1234' SESSION TIMEOUT 120;
```

다음 예에서는 이름이 bob인 사용자를 생성합니다. 네임스페이스는 myco_aad입니다. 이것은 예시입니다. 명령을 성공적으로 실행하려면 등록된 ID 공급업체가 있어야 합니다. 자세한 내용은 Native identity provider (IdP) federation for Amazon Redshift(Amazon Redshift용 네이티브 자격 증명 공급자(IdP) 페더레이션)를 참조하세요.

```
CREATE USER myco_aad:bob EXTERNALID "ABC123" PASSWORD DISABLE;
```

CREATE VIEW

데이터베이스에 뷰를 생성합니다. 뷰는 물리적으로 구체화되는 것은 아닙니다. 뷰를 정의하는 쿼리는 뷰가 쿼리에서 참조될 때마다 실행됩니다. 외부 테이블로 보기를 새로 만들려면 WITH NO SCHEMA BINDING 절을 포함시키십시오.

표준 뷰를 생성하려면 기본 테이블 또는 기본 뷰에 대한 액세스 권한이 있어야 합니다. 표준 보기를 쿼리하려면 보기 자체에 대한 선택 권한이 필요하지만, 기본 테이블에 대한 선택 권한은 필요하지 않습니다. 다른 스키마의 테이블이나 뷰를 참조하는 뷰를 생성하거나 구체화된 뷰를 참조하는 뷰를 생성하는 경우 사용 권한이 필요합니다. 지연 바인딩 뷰를 쿼리하려면 지연 바인딩 뷰 자체에 대한 몇 가지 권한이 필요합니다. 또한 Late Binding 뷰의 소유자에게 참조 객체(테이블, 뷰, 사용자 정의 함수 등)에 대한 선택 권한이 있는지 확인해야 합니다. 지연 바인딩 뷰에 대한 자세한 내용은 사용 노트 섹션을 참조하세요.

필수 권한

CREATE VIEW를 사용하려면 다음 권한 중 하나가 필요합니다.

- CREATE [OR REPLACE] VIEW를 사용하여 뷰를 만들려면 다음을 수행하세요.
  - 수퍼유저
  - CREATE [REPLACE] VIEW 권한이 있는 사용자
- CREATE OR REPLACE VIEW를 사용하여 기존 뷰를 바꾸려면 다음을 수행하세요.
  - 수퍼유저
  - CREATE [OR REPLACE] VIEW 권한이 있는 사용자
  - 보기 소유자

사용자가 사용자 정의 함수가 포함된 뷰에 액세스하려는 경우 해당 함수에 대한 EXECUTE 권한이 있어야 합니다.

구문

```
CREATE [ OR REPLACE ] VIEW name [ ( column_name [, ...] ) ] AS query
[ WITH NO SCHEMA BINDING ]
```

파라미터

OR REPLACE

같은 이름의 뷰가 이미 있는 경우 뷰가 대체됩니다. 같은 열 이름과 데이터 형식을 사용하여 같은 열 집합을 생성하는 새 쿼리로만 뷰를 바꿀 수 있습니다. CREATE OR REPLACE VIEW는 작업 완료 시까지 읽기 및 쓰기를 위한 뷰를 잠급니다.

뷰가 교체될 때 소유권과 부여된 권한 등의 다른 속성은 유지됩니다.

이름

결과의 이름. 스키마 이름이 지정되어 있는 경우(예: `myschema.myview`), 뷰는 지정된 스키마를 사용하여 생성됩니다. 그렇지 않으면, 뷰가 현재 스키마로 생성됩니다. 보기 이름은 같은 스키마에 있는 다른 보기 또는 테이블의 이름과는 달라야 합니다.

'#'으로 시작하는 보기 이름을 지정하면 보기가 현재 세션에서만 보이는 임시 보기로 생성됩니다.

유효한 이름에 대한 자세한 내용은 이름 및 식별자 섹션을 참조하세요. 시스템 데이터베이스 template0, template1, padb_harvest 또는 sys:internal에서 테이블 또는 뷰를 생성할 수 없습니다.

column_name

뷰에서 열에 사용할 이름의 선택적 목록입니다. 열 이름이 지정되지 않은 경우에는 쿼리에서 파생됩니다. 단일 뷰에서 정의할 수 있는 최대 열 수는 1,600개입니다.

query

테이블로 평가되는 (SELECT 문 형식의) 쿼리입니다. 이 테이블은 뷰에 있는 열과 행을 정의합니다.

WITH NO SCHEMA BINDING

보기가 테이블, 사용자 정의 함수 같은 기본 데이터베이스 객체에 바인딩되지 않도록 지정하는 절입니다. 결과적으로 보기와 보기가 참조하는 객체 사이에 종속성이 없습니다. 참조 객체가 존재하지 않는 경우에도 보기를 새로 만들 수 있습니다. 종속성이 없기 때문에 뷰에 영향을 주지 않고서 참조 객체를 중단하거나 변경할 수 있습니다. Amazon Redshift는 뷰가 쿼리될 때까지 종속성을 확인하지 않습니다. 지연 바인딩 뷰에 대한 자세한 내용을 보려면 PG_GET_LATE_BINDING_VIEW_COLS 함수를 실행합니다.

WITH NO SCHEMA BINDING 절을 포함할 때는 SELECT 문에서 참조한 테이블 및 보기를 스키마 이름으로 정규화해야 합니다. 참조 테이블이 존재하지 않더라도 보기를 새로 만들 때는 스키마가 반드시 존재해야 합니다. 예를 들어 다음과 같은 문은 오류를 반환합니다.

```
create view myevent as select eventname from event
with no schema binding;
```

다음 문은 성공적으로 실행됩니다.

```
create view myevent as select eventname from public.event
with no schema binding;
```

> **Note**
>
> 뷰를 업데이트하거나 뷰에 삽입하거나 뷰에서 삭제할 수 없습니다.

사용 노트

Late Binding 보기

Late Binding 보기는 자신이 쿼리될 때까지 테이블 및 기타 보기와 같은 기본 데이터베이스 객체를 확인하지 않습니다. 따라서 보기를 삭제한 다음 다시 생성하지 않고 기본 객체를 수정 또는 삭제할 수 있습니다. 기본 객체를 삭제하는 경우 Late Binding 보기에 대한 쿼리가 실패합니다. Late Binding 보기에 대한 쿼리가 기본 객체에 존재하지 않는 열을 참조하는 경우 쿼리가 실패합니다.

Late Binding 보기의 기본 테이블 또는 보기를 삭제한 후 다시 생성하면 기본 액세스 권한을 가진 새 객체가 생성됩니다. 뷰를 쿼리할 사용자에게 기본 객체에 대한 권한을 부여해야 할 수 있습니다.

Late Binding 보기를 생성하려면 WITH NO SCHEMA BINDING 절을 포함시키십시오. 다음 예에서는 스키마 바인딩 없이 보기를 새로 만듭니다.

```
create view event_vw as select * from public.event
with no schema binding;
```

```
select * from event_vw limit 1;
```

```
eventid | venueid | catid | dateid | eventname     | starttime
--------+---------+-------+--------+---------------+--------------------
      2 |     306 |     8 |   2114 | Boris Godunov | 2008-10-15 20:00:00
```

다음은 Late Binding 보기를 다시 생성하지 않고 기본 테이블을 변경할 수 있음을 보여주는 예입니다.

```
alter table event rename column eventname to title;
```

```
select * from event_vw limit 1;

eventid | venueid | catid | dateid | title         | starttime
--------+---------+-------+--------+---------------+--------------------
      2 |     306 |     8 |   2114 | Boris Godunov | 2008-10-15 20:00:00
```

Amazon Redshift Spectrum 외부 테이블은 Late Binding 뷰에서만 참조할 수 있습니다. Late Binding 뷰의 애플리케이션 하나가 Amazon Redshift 및 Redshift Spectrum 테이블을 둘 다 쿼리합니다. 예를 들어 UNLOAD 명령을 사용하여 오래된 데이터를 Amazon S3에 보관할 수 있습니다. 그런 다음 Amazon S3의 데이터를 참조하는 Redshift Spectrum 외부 테이블을 생성하고, 두 테이블을 쿼리하는 뷰를 생성합니다. 다음은 UNION ALL 절을 사용하여 Amazon Redshift SALES 테이블과 Redshift Spectrum SPECTRUM.SALES 테이블을 조인하는 예입니다.

```
create view sales_vw as
select * from public.sales
union all
select * from spectrum.sales
with no schema binding;
```

SPECTRUM.SALES 테이블을 비롯한 Redshift Spectrum 외부 테이블 생성에 대한 자세한 내용은 Amazon Redshift Spectrum 시작하기 섹션을 참조하세요.

> **Important**
>
> 지연 바인딩 뷰에서 표준 뷰를 만드는 경우 표준 뷰의 정의에는 지연 바인딩 뷰의 소유자를 비롯하여 표준 뷰가 만들어질 당시 지연 바인딩 뷰의 정의가 포함됩니다. 기본 지연 바인딩 뷰를 변경하는 경우 표준 뷰를 다시 만들 때까지 표준 뷰에서 이러한 변경 사항이 사용되지 않습니다. 따라서 표준 뷰가 쿼리될 때 항상 지연 바인딩 뷰의 정의와 지연 바인딩 뷰의 소유자를 사용하여 해당 표준 뷰를 만들 때 권한을 확인합니다.

지연 바인딩 뷰의 최신 정의를 참조하도록 표준 뷰를 업데이트하려면 표준 뷰를 만들 때 사용한 초기 뷰 정의와 함께 CREATE OR REPLACE VIEW를 실행하세요.

지연 바인딩 뷰에서 표준 뷰를 만드는 다음 예시를 참조하세요.

```
create view sales_vw_lbv as
select * from public.sales
with no schema binding;

show view sales_vw_lbv;
                            Show View DDL statement
--------------------------------------------------------------------------------
 create view sales_vw_lbv as select * from public.sales with no schema binding;
(1 row)

create view sales_vw as
select * from sales_vw_lbv;

show view sales_vw;
                                                  Show View DDL statement
--------------------------------------------------------------------------------------------------------------
 SELECT sales_vw_lbv.price, sales_vw_lbv."region" FROM (SELECT sales.price,
 sales."region" FROM sales) sales_vw_lbv;
(1 row)
```

참고로, 표준 뷰의 DDL 문에 표시된 지연 바인딩 뷰는 표준 뷰를 만들 때 정의되며 이후에 지연 바인딩 뷰를 변경해도 업데이트되지 않습니다.

예시

예제 명령은 TICKIT 데이터베이스라는 샘플 객체 및 데이터 세트를 사용합니다. 자세한 내용은 샘플 데이터베이스를 참조하세요.

다음 명령은 EVENT라는 테이블에서 myevent라는 뷰를 생성합니다.

```
create view myevent as select eventname from event
where eventname = 'LeAnn Rimes';
```

다음 명령은 USERS라는 테이블에서 myuser라는 뷰를 생성합니다.

```
create view myuser as select lastname from users;
```

다음 명령은 USERS라는 테이블에서 myuser라는 뷰를 생성하거나 대체합니다.

```
create or replace view myuser as select lastname from users;
```

다음 예에서는 스키마 바인딩 없이 보기를 새로 만듭니다.

```
create view myevent as select eventname from public.event
with no schema binding;
```

DEALLOCATE

준비된 문을 할당 취소합니다.

구문

```
DEALLOCATE [PREPARE] plan_name
```

파라미터

PREPARE

이 키워드는 선택 사항으로서 무시됩니다.

plan_name

할당 취소할 준비된 문의 이름입니다.

사용 관련 참고 사항

DEALLOCATE는 이전에 준비된 SQL 문의 할당 취소에 사용됩니다. 준비된 문을 명시적으로 할당 취소하지 않으면 현재 세션 종료 시 할당 취소됩니다. 준비된 문에 대한 자세한 내용은 PREPARE 섹션을 참조하세요.

참고

EXECUTE, PREPARE

DECLARE

새로운 커서를 정의합니다. 더 큰 쿼리의 결과 집합에서 한 번에 몇 개의 행을 검색하려면 커서를 사용하십시오.

커서의 첫 행을 가져올 때, 필요한 경우 메모리나 디스크에서 리더 노드에 전체 결과 집합이 구체화됩니다. 큰 결과 집합을 가진 커서를 사용하면 성능에 나쁜 영향을 미칠 가능성이 있으므로, 가급적이면 다른 접근 방식을 사용하는 것이 좋습니다. 자세한 내용은 커서 사용 시 성능 고려사항 단원을 참조하십시오.

트랜잭션 블록 내에 커서를 선언해야 합니다. 세션당 한 번에 한 개의 커서만 열 수 있습니다.

자세한 내용은 FETCH, CLOSE 섹션을 참조하세요.

구문

```
DECLARE cursor_name CURSOR FOR query
```

파라미터

cursor_name

새 커서의 이름입니다.

query

커서를 채우는 SELECT 문입니다.

DECLARE CURSOR 사용 시 주의 사항

클라이언트 애플리케이션이 ODBC 연결을 사용하고 쿼리가 메모리에 비해 너무 큰 결과 집합을 생성하는 경우, 커서를 사용하여 결과 집합을 클라이언트 애플리케이션으로 스트리밍할 수 있습니다. 커서를 사용할 때, 리더 노드에서 전체 결과 집합이 구체화된 다음에 클라이언트가 결과를 증분 방식으로 가져올 수 있습니다.

> **Note**
>
> Microsoft Windows용 ODBC에서 커서를 사용하려면 Amazon Redshift에 사용하는 ODBC DSN에서 [선언/가져오기 사용(Use Declare/Fetch)] 옵션을 사용합니다. 라운드 트립을 최소화하려면 다중 노드 클러스터에 대해 ODBC DSN 옵션 대화 상자에서 캐시 크기 필드를 사용하여 ODBC 캐시 크기를 4,000 이상으로 설정하는 것이 좋습니다. 단일 노드 클러스터에서 Cache Size를 1,000으로 설정합니다.

커서를 사용하면 성능에 나쁜 영향을 미칠 가능성이 있으므로, 가급적이면 다른 접근 방식을 사용하는 것이 좋습니다. 자세한 내용은 커서 사용 시 성능 고려사항 단원을 참조하십시오.

Amazon Redshift 커서는 다음 제한 사항과 함께 지원됩니다.

- 세션당 한 번에 한 개의 커서만 열 수 있습니다.
- 트랜잭션 (BEGIN … END) 내에서 커서를 사용해야 합니다.
- 모든 커서에 대한 최대 누적 결과 집합 크기는 클러스터 노드 유형을 기준으로 제한됩니다. 더 큰 결과 집합이 필요하면 XL 또는 8XL 노드 구성으로 크기를 조정할 수 있습니다.

  자세한 내용은 커서 제약 조건 단원을 참조하십시오.

커서 제약 조건

커서의 첫 행을 가져올 때, 리더 노드에 전체 결과 집합이 구체화됩니다. 결과 집합이 메모리에 적합하지 않을 경우 필요에 따라 디스크에 기록됩니다. Amazon Redshift는 리더 노드의 무결성을 보호하려고 클러스터의 노드 유형을 기준으로 모든 커서 결과 집합의 크기에 제약 조건을 적용합니다.

다음 표에는 각 클러스터 노드 유형에 대한 최대 전체 결과 집합 크기가 나와 있습니다. 최대 결과 집합 크기의 단위는 메가바이트입니다.

노드 유형	클러스터당 최대 결과 집합 크기(MB)
DC2 Large 다중 노드	192,000
DC2 Large 단일 노드	8,000
DC2 8XL 다중 노드	3,200,000
RA3 16XL 다중 노드	14,400,000
RA3 4XL 다중 노드	3,200,000
RA3 XLPLUS 다중 노드	1,000,000
RA3 XLPLUS 단일 노드	64,000
RA3 LARGE 다중 노드	240,000
RA3 LARGE 단일 노드	8,000

노드 유형	클러스터당 최대 결과 집합 크기(MB)
Amazon Redshift Serverless	150,000

클러스터에 대한 활성 커서 구성을 보려면 수퍼유저로서 STV_CURSOR_CONFIGURATION 시스템 테이블을 쿼리하십시오. 활성 커서의 상태를 보려면 STV_ACTIVE_CURSORS 시스템 테이블을 쿼리하십시오. 사용자에게는 자신의 커서에 대한 행만 보이지만, 수퍼유저는 모든 커서를 볼 수 있습니다.

커서 사용 시 성능 고려사항

커서는 리더 노드의 전체 결과 집합을 구체화한 후 결과를 클라이언트로 반환하기 시작하므로, 매우 큰 결과 집합을 가진 커서를 사용하면 성능에 나쁜 영향을 미칠 수 있습니다. 결과 집합이 매우 클 때는 커서를 사용하지 않는 것이 좋습니다. 애플리케이션에서 ODBC 연결을 사용할 때와 같이, 경우에 따라서는 커서가 유일하게 실행 가능한 해결책일 수도 있습니다. 가능하면 다음과 같은 대안을 사용하는 것이 좋습니다.

- UNLOAD를 사용하여 큰 테이블을 내보냅니다. UNLOAD 사용 시, 컴퓨팅 노드가 병렬로 작동하여 Amazon Simple Storage Service에 있는 데이터 파일로 데이터를 직접 전송합니다. 자세한 내용은 Amazon Redshift에서 데이터 언로드 단원을 참조하십시오.
- 클라이언트 애플리케이션에서 JDBC 가져오기 크기 파라미터를 설정합니다. JDBC 연결을 사용하는데 클라이언트 측 메모리 부족 오류가 발생할 경우, JDBC 가져오기 크기 파라미터를 설정하여 클라이언트가 더 작은 배치에서 결과 집합을 검색하도록 할 수 있습니다. 자세한 내용은 JDBC Fetch Size 파라미터 설정 단원을 참조하십시오.

DECLARE CURSOR 예제

다음 예에서는 LOLLAPALOOZA로 명명된 커서를 선언하여 롤라팔루자(LOLLAPALOOZA) 행사를 위한 판매 정보를 선택한 후 커서를 사용하여 결과 집합에서 행을 가져옵니다.

```
-- Begin a transaction

begin;

-- Declare a cursor

declare lollapalooza cursor for
select eventname, starttime, pricepaid/qtysold as costperticket, qtysold
from sales, event
```

```
where sales.eventid = event.eventid
and eventname='Lollapalooza';

-- Fetch the first 5 rows in the cursor lollapalooza:

fetch forward 5 from lollapalooza;

  eventname   |      starttime      | costperticket | qtysold
--------------+---------------------+---------------+---------
 Lollapalooza | 2008-05-01 19:00:00 |   92.00000000 |       3
 Lollapalooza | 2008-11-15 15:00:00 |  222.00000000 |       2
 Lollapalooza | 2008-04-17 15:00:00 |  239.00000000 |       3
 Lollapalooza | 2008-04-17 15:00:00 |  239.00000000 |       4
 Lollapalooza | 2008-04-17 15:00:00 |  239.00000000 |       1
(5 rows)

-- Fetch the next row:

fetch next from lollapalooza;

  eventname   |      starttime      | costperticket | qtysold
--------------+---------------------+---------------+---------
 Lollapalooza | 2008-10-06 14:00:00 |  114.00000000 |       2

-- Close the cursor and end the transaction:

close lollapalooza;
commit;
```

다음 예제는 테이블의 모든 결과를 포함하는 refcursor를 반복합니다.

```
CREATE TABLE tbl_1 (a int, b int);
INSERT INTO tbl_1 values (1, 2),(3, 4);

CREATE OR REPLACE PROCEDURE sp_cursor_loop() AS $$
DECLARE
    target record;
    curs1 cursor for select * from tbl_1;
BEGIN
    OPEN curs1;
    LOOP
        fetch curs1 into target;
        exit when not found;
```

```
        RAISE INFO 'a %', target.a;
    END LOOP;
    CLOSE curs1;
END;
$$ LANGUAGE plpgsql;

CALL sp_cursor_loop();

SELECT message
   from svl_stored_proc_messages
   where querytxt like 'CALL sp_cursor_loop()%';

  message
----------
      a 1
      a 3
```

DELETE

행을 테이블에서 삭제합니다.

> **Note**
>
> 단일 SQL 문의 최대 크기는 16MB입니다.

구문

```
[ WITH [RECURSIVE] common_table_expression [, common_table_expression , ...] ]
DELETE [ FROM ] { table_name | materialized_view_name }
    [ { USING } table_name, ... ]
    [ WHERE condition ]
```

파라미터

WITH 절

하나 이상의 common-table-expressions를 지정하는 절(옵션)입니다. WITH 절 섹션을 참조하세요.

FROM

USING 절이 지정되어 있을 때를 제외하면 FROM 키워드는 선택 사항입니다. `delete from event;` 및 `delete event;` 문은 똑같이 EVENT 테이블에서 모든 행을 제거하는 작업을 수행합니다.

> **Note**
>
> 테이블에서 모든 행을 삭제하려면 테이블을 TRUNCATE합니다. TRUNCATE는 DELETE보다 훨씬 더 효율적이며 VACUUM 및 ANALYZE가 필요하지 않습니다. 하지만 TRUNCATE는 이 명령이 실행되는 트랜잭션을 커밋합니다.

table_name

임시 또는 영구 테이블입니다. 테이블의 소유자 또는 테이블에 대한 DELETE 권한을 가진 사용자만이 테이블에서 행을 삭제할 수 있습니다.

대규모 테이블에서 정규화되지 않은 빠른 삭제 작업을 하려면 TRUNCATE 명령의 사용을 고려하십시오(TRUNCATE 참조).

> **Note**
>
> 테이블에서 많은 수의 행을 삭제한 후
>
> - 테이블을 완전히 비워 스토리지 공간을 회수하고 행을 다시 정렬합니다.
> - 테이블을 분석하여 쿼리 플래너에 대한 통계를 업데이트합니다.

materialized_view_name

구체화된 뷰 DELETE 문은 구체화된 뷰로 스트리밍 모으기에서 사용되는 구체화된 뷰에서 작동합니다. 구체화된 뷰의 소유자 또는 구체화된 뷰에 대한 DELETE 권한을 가진 사용자만 구체화된 뷰에서 행을 삭제할 수 있습니다.

사용자에게 RLS 무시 권한이 부여되지 않은 행 수준 보안(RLS) 정책을 사용하는 스트리밍 수집을 위해 구체화된 뷰에서 DELETE를 실행할 수 없습니다. 여기에는 예외가 있습니다. DELETE를 수행하는 사용자에게 IGNORE RLS가 부여되면 성공적으로 실행됩니다. 자세한 내용은 RLS 정책 소유권 및 관리를 참조하세요.

USING table_name, ...

USING 키워드는 WHERE 절 조건에서 추가적인 테이블이 참조될 때 테이블 목록을 소개하는 데 사용됩니다. 예를 들어, 다음 문은 EVENT 및 SALES 테이블에 대한 조인 조건을 충족시키는 EVENT 테이블에서 모든 행을 삭제합니다. SALES 테이블은 FROM 목록에 명시적으로 명명되어야 합니다.

```
delete from event using sales where event.eventid=sales.eventid;
```

USING 절에서 대상 테이블 이름을 반복하는 경우 DELETE 작업은 자가 조인을 실행합니다. 같은 쿼리를 쓰는 방법의 대안으로서, WHERE 절에서 USING 구문 대신 하위 쿼리를 사용할 수 있습니다.

WHERE condition

행 삭제를 조건과 일치하는 행으로 제한하는 선택적인 절입니다. 예를 들어, 조건은 열에 대한 제한, 조인 조건 또는 쿼리의 결과를 바탕으로 하는 조건일 수 있습니다. 쿼리는 DELETE 명령의 대상이 아닌 테이블을 참조할 수 있습니다. 예:

```
delete from t1
where col1 in(select col2 from t2);
```

아무런 조건도 지정되지 않은 경우 테이블에 있는 모든 행이 삭제됩니다.

예시

CATEGORY 테이블에서 모든 행을 삭제합니다.

```
delete from category;
```

CATEGORY 테이블에서 CATID 값이 0~9 사이의 값인 행을 삭제합니다.

```
delete from category
where catid between 0 and 9;
```

SALES 테이블에 자신의 SELLERID 값이 존재하지 않는 LISTING 테이블에서 행을 삭제합니다.

```
delete from listing
where listing.sellerid not in(select sales.sellerid from sales);
```

다음 두 쿼리는 모두 EVENT 테이블에 대한 조인과 CATID 열에 대한 추가적인 제한 사항을 바탕으로 CATEGORY 테이블에서 한 개의 행을 삭제합니다.

```
delete from category
using event
where event.catid=category.catid and category.catid=9;
```

```
delete from category
where catid in
(select category.catid from category, event
where category.catid=event.catid and category.catid=9);
```

다음 쿼리는 mv_cities 구체화된 뷰에서 모든 행을 삭제합니다. 이 예의 구체화된 뷰 이름은 샘플입니다.

```
delete from mv_cities;
```

DESC DATASHARE

ALTER DATASHARE를 사용하여 추가된 datashare 내의 데이터베이스 객체 목록을 표시합니다. Amazon Redshift는 테이블, 뷰 및 함수의 이름, 데이터베이스, 스키마 및 형식을 표시합니다.

데이터 공유 객체에 대한 추가 정보는 시스템 보기를 사용하여 찾을 수 있습니다. 자세한 내용은 SVV_DATASHARE_OBJECTS 및 SVV_DATASHARES를 참조하세요.

구문

```
DESC DATASHARE datashare_name [ OF [ ACCOUNT account_id ] NAMESPACE namespace_guid ]
```

파라미터

datashare_name

datashare의 이름입니다.

NAMESPACE namespace_guid

datashare에서 사용하는 네임스페이스를 지정하는 값입니다. 소비자 클러스터 관리자로 DESC DATAHSARE를 실행할 때 NAMESPACE 파라미터를 지정하여 인바운드 datashare를 봅니다.

ACCOUNT account_id

datashare가 속한 계정을 지정하는 값입니다.

사용 관련 참고 사항

소비자 계정 관리자는 DESC DATASHARE를 실행하여 AWS 계정 내의 인바운드 datashare를 볼 때 NAMESPACE 옵션을 지정합니다. DESC DATASHARE를 실행하여 AWS 계정 간 인바운드 datashare를 볼 때 ACCOUNT 및 NAMESPACE 옵션을 지정합니다.

예시

다음 예에서는 생산자 클러스터의 아웃바운드 datashare에 대한 정보를 표시합니다.

```
DESC DATASHARE salesshare;

producer_account |          producer_namespace           | share_type  | share_name   |
 object_type |        object_name           |  include_new
-----------------+--------------------------------------+-------------+--------------
+-------------+------------------------------+--------------
 123456789012    | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d  | OUTBOUND    |  salesshare  |
 TABLE       | public.tickit_sales_redshift |
 123456789012    | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d  | OUTBOUND    |  salesshare  |
 SCHEMA      | public                       |   t
```

다음 예에서는 소비자 클러스터의 인바운드 datashare에 대한 정보를 표시합니다.

```
DESC DATASHARE salesshare of ACCOUNT '123456789012' NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';

 producer_account |          producer_namespace          | share_type | share_name |
 object_type |         object_name          |  include_new
-----------------+--------------------------------------+------------+------------
+-------------+------------------------------+--------------
 123456789012     | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare |
 table       | public.tickit_sales_redshift |
 123456789012     | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare |
 schema      | public                       |
(2 rows)
```

DESC IDENTITY PROVIDER

자격 증명 공급자에 대한 정보를 표시합니다. 슈퍼 사용자만 자격 증명 공급자를 설명할 수 있습니다.

구문

```
DESC IDENTITY PROVIDER identity_provider_name
```

파라미터

identity_provider_name

자격 증명 공급자 이름입니다.

예제

다음 예에서는 자격 증명 공급자에 대한 정보를 표시합니다.

```
DESC IDENTITY PROVIDER azure_idp;
```

샘플 출력은 다음과 같습니다.

```
  uid   |   name    | type  |             instanceid              | namespc |

                                        params

                          | enabled
--------+-----------+-------+-------------------------------------+---------
+-----------------------------------------------------------------------------------------------------------
+---------
 126692 | azure_idp | azure | e40d4bb2-7670-44ae-bfb8-5db013221d73 | aad     |
 {"issuer":"https://login.microsoftonline.com/e40d4bb2-7670-44ae-bfb8-5db013221d73/
v2.0", "client_id":"871c010f-5e61-4fb1-83ac-98610a7e9110", "client_secret":'',
 "audience":["https://analysis.windows.net/powerbi/connector/AmazonRedshift", "https://
analysis.windows.net/powerbi/connector/AWSRDS"]} | t
(1 row)
```

마스킹 정책 분리

이미 연결되어 있던 동적 데이터 마스킹 정책을 열에서 분리합니다. 동적 데이터 마스킹에 대한 자세한 내용은 동적 데이터 마스킹(동적 데이터 마스킹(미리 보기)을 참조하세요.

sys:secadmin 역할이 부여된 수퍼유저와 사용자 또는 역할은 마스킹 정책을 분리할 수 있습니다.

구문

```
DETACH MASKING POLICY policy_name
   ON { table_name }
   ( output_column_names )
   FROM { user_name | ROLE role_name | PUBLIC };
```

파라미터

policy_name

분리할 마스킹 정책의 이름입니다.

table_name

마스킹 정책을 분리할 테이블의 이름입니다.

output_column_names

마스킹 정책이 연결된 열의 이름입니다.

user_name

마스킹 정책이 연결된 사용자의 이름입니다.

단일 DETACH MASKING POLICY 문에서 user_name, role_name 및 PUBLIC 중 하나만 설정할 수 있습니다.

role_name

마스킹 정책이 연결된 역할의 이름입니다.

단일 DETACH MASKING POLICY 문에서 user_name, role_name 및 PUBLIC 중 하나만 설정할 수 있습니다.

PUBLIC

정책이 테이블의 모든 사용자에게 연결되었음을 표시합니다.

단일 DETACH MASKING POLICY 문에서 user_name, role_name 및 PUBLIC 중 하나만 설정할 수 있습니다.

DETACH RLS POLICY

테이블에 대한 행 수준 보안 정책을 하나 이상의 사용자 또는 역할에서 분리합니다.

`sys:secadmin` 역할이 부여된 수퍼유저와 사용자 또는 역할은 정책을 분리할 수 있습니다.

구문

```
DETACH RLS POLICY policy_name ON [TABLE] table_name [, ...]
FROM { user_name | ROLE role_name | PUBLIC } [, ...]
```

파라미터

policy_name

정책의 이름입니다.

ON [TABLE] table_name [, ...]

행 수준 보안 정책이 분리되는 테이블 또는 보기입니다.

FROM { user_name | ROLE role_name | PUBLIC} [, ...]

정책이 하나 이상의 지정된 사용자 또는 역할에서 분리되는지 여부를 지정합니다.

사용 노트

DETACH RLS POLICY 문과 관련한 작업을 수행할 때는 다음을 준수하세요.

- 관계, 사용자, 역할 또는 퍼블릭 객체에서 정책을 분리할 수 있습니다.

예시

다음 예에서는 테이블에 대한 정책을 역할에서 분리합니다.

```
DETACH RLS POLICY policy_concerts ON tickit_category_redshift FROM ROLE analyst, ROLE
 dbadmin;
```

DROP DATABASE

데이터베이스를 삭제합니다.

트랜잭션 블록(BEGIN ... END) 내에서는 DROP DATABASE를 실행할 수 없습니다. 버전 관리에 대한 자세한 내용은 직렬화 가능 격리 단원을 참조하십시오.

구문

```
DROP DATABASE database_name
```

파라미터

database_name

삭제될 데이터베이스의 이름입니다. dev, padb_harvest, template0, template1 또는 sys:internal 데이터베이스를 삭제할 수 없고, 현재 데이터베이스도 삭제할 수 없습니다.

외부 데이터베이스를 삭제하려면 외부 스키마를 삭제합니다. 자세한 내용은 DROP SCHEMA 단원을 참조하십시오.

DROP DATABASE 사용 참고 사항

DROP DATABASE 문을 사용할 때 다음 사항을 고려하세요.

- 일반적으로 DROP DATABASE 문을 사용하여 AWS Data Exchange datashare가 포함된 데이터베이스를 삭제하지 않는 것이 좋습니다. 그렇게 하면 datashare에 대한 액세스 권한이 있는 AWS 계정가 액세스 권한을 상실합니다. 이러한 유형의 변경을 수행하면 AWS Data Exchange의 데이터 제품 조건을 위반할 수 있습니다.

  다음 예에서는 AWS Data Exchange datashare가 포함된 데이터베이스가 삭제될 경우 오류를 보여줍니다.

  ```
  DROP DATABASE test_db;
  ERROR:  Drop of database test_db that contains ADX-managed datashare(s)
   requires session variable datashare_break_glass_session_var to be set to value
   'ce8d280c10ad41'
  ```

  데이터베이스 삭제를 허용하려면 다음 변수를 설정하고 DROP DATABASE 문을 다시 실행합니다.

  ```
  SET datashare_break_glass_session_var to 'ce8d280c10ad41';
  ```

  ```
  DROP DATABASE test_db;
  ```

이 경우 Amazon Redshift는 임의의 일회성 값을 생성하여 AWS Data Exchange datashare가 포함된 데이터베이스에 대해 DROP DATABASE를 허용하도록 세션 변수를 설정합니다.

예시

다음 예에서는 TICKIT_TEST라는 데이터베이스를 삭제합니다.

```
drop database tickit_test;
```

DROP DATASHARE

datashare를 삭제합니다. 이 명령은 되돌릴 수 없습니다.

슈퍼 사용자 또는 datashare 소유자만 datashare를 삭제할 수 있습니다.

필수 권한

DROP DATASHARE에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- DROP DATASHARE 권한이 있는 사용자
- Datashare 소유자

구문

```
DROP DATASHARE datashare_name;
```

파라미터

datashare_name

삭제될 datashare의 이름입니다.

DROP DATASHARE 사용 참고 사항

DROP DATASHARE 문을 사용할 때 다음 사항을 고려하세요.

- 일반적으로 DROP DATASHARE 문을 사용하여 AWS Data Exchange datashare를 삭제하지 않는 것이 좋습니다. 그렇게 하면 datashare에 대한 액세스 권한이 있는 AWS 계정가 액세스 권한을 상실합니다. 이러한 유형의 변경을 수행하면 AWS Data Exchange의 데이터 제품 조건을 위반할 수 있습니다.

  다음 예에서는 AWS Data Exchange datashare가 삭제될 경우 오류를 보여줍니다.

```
DROP DATASHARE salesshare;
ERROR:  Drop of ADX-managed datashare salesshare requires session variable
 datashare_break_glass_session_var to be set to value '620c871f890c49'
```

  AWS Data Exchange datashare 삭제를 허용하려면 다음 변수를 설정하고 DROP DATASHARE 문을 다시 실행합니다.

```
SET datashare_break_glass_session_var to '620c871f890c49';
```

```
DROP DATASHARE salesshare;
```

  이 경우 Amazon Redshift는 임의의 일회성 값을 생성하여 AWS Data Exchange datashare에 대해 DROP DATASHARE를 허용하도록 세션 변수를 설정합니다.

예시

다음 예에서는 `salesshare`라는 datashare를 삭제합니다.

```
DROP DATASHARE salesshare;
```

DROP EXTERNAL VIEW

데이터베이스에서 외부 뷰를 삭제합니다. 외부 뷰를 삭제하면 뷰와 연결된 모든 SQL 엔진(예: Amazon Athena 및 Amazon EMR Spark)에서 해당 뷰가 제거됩니다. 이 명령은 되돌릴 수 없습니다. Data Catalog 뷰에 대한 자세한 내용은 AWS Glue Data Catalog 뷰를 참조하세요.

구문

```
DROP EXTERNAL VIEW schema_name.view_name [ IF EXISTS ]
{catalog_name.schema_name.view_name | awsdatacatalog.dbname.view_name |
 external_schema_name.view_name}
```

파라미터

schema_name.view_name

AWS Glue 데이터베이스에 연결된 스키마이며, 뷰 이름이 뒤따릅니다.

IF EXISTS

뷰가 있는 경우에만 뷰를 삭제합니다.

catalog_name.schema_name.view_name | awsdatacatalog.dbname.view_name | external_schema_name.view_name

뷰를 삭제할 때 사용하는 스키마의 표기법입니다. 직접 만든 Glue 데이터베이스인 AWS Glue Data Catalog 또는 직접 만든 외부 스키마를 사용하도록 지정할 수 있습니다. 자세한 내용은 CREATE DATABASE 및 CREATE EXTERNAL SCHEMA를 참조하세요.

query_definition

뷰를 변경하기 위해 Amazon Redshift가 실행하는 SQL 쿼리의 정의입니다.

예시

다음 예시에서는 sample_schema.glue_data_catalog_view라는 데이터 카탈로그 뷰를 삭제합니다.

```
DROP EXTERNAL VIEW sample_schema.glue_data_catalog_view IF EXISTS
```

DROP FUNCTION

데이터베이스에서 UDF(사용자 정의 함수)를 제거합니다. 이름은 같지만 서명은 다른 함수가 여러 개 존재할 수 있으므로 함수의 서명 또는 인수 데이터 형식의 목록을 지정해야 합니다. Amazon Redshift 내장 함수를 삭제할 수 없습니다.

이 명령은 되돌릴 수 없습니다.

필수 권한

DROP FUNCTION에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- DROP FUNCTION 권한이 있는 사용자
- 함수 소유자

구문

```
DROP FUNCTION name
( [arg_name] arg_type   [, ...] )
[ CASCADE | RESTRICT ]
```

파라미터

이름

제거할 기능의 이름입니다.

arg_name

입력 인수의 이름입니다. 함수의 자격 증명을 확인하는 데는 인수 데이터 형식만 있으면 되므로 DROP FUNCTION은 인수 이름을 무시합니다.

arg_type

입력 인수의 데이터 형식입니다. 최대 32가지 데이터 형식을 가진, 쉼표로 구분된 목록을 제공할 수 있습니다.

CASCADE

보기 같은 함수에 의존하는 객체를 자동으로 삭제하도록 지정하는 키워드입니다.

함수에 종속되지 않는 보기를 새로 만들려면 보기 정의에 WITH NO SCHEMA BINDING 절을 포함시키십시오. 자세한 내용은 CREATE VIEW 단원을 참조하십시오.

RESTRICT

객체가 해당 함수에 의존하는 경우 함수를 삭제하지 않고 메시지를 반환하도록 지정하는 키워드입니다. 이 동작이 기본값입니다.

예시

다음 예에서는 `f_sqrt`라는 함수를 삭제합니다.

```
drop function f_sqrt(int);
```

종속 항목이 있는 함수를 제거하려면 다음 예와 같이 CASCADE 옵션을 사용하십시오.

```
drop function f_sqrt(int)cascade;
```

DROP GROUP

사용자 그룹을 삭제합니다. 이 명령은 되돌릴 수 없습니다. 이 명령은 그룹에 있는 개별 사용자를 삭제하지 않습니다.

개별 사용자를 삭제하려면 DROP USER 섹션을 참조하세요.

구문

```
DROP GROUP name
```

파라미터

이름

삭제할 사용자 그룹의 이름입니다.

예제

다음 예에서는 guests 사용자 그룹을 삭제합니다.

```
DROP GROUP guests;
```

객체에 대해 어떤 권한을 가지고 있는 그룹은 삭제할 수 없습니다. 그런 그룹을 삭제하려고 하면 다음 오류를 수신하게 됩니다.

```
ERROR: group "guests" can't be dropped because the group has a privilege on some object
```

그룹이 객체에 대한 권한을 갖고 있을 경우 권한을 취소한 후 그룹을 삭제합니다. guests 그룹에 대한 권한을 가진 객체를 찾으려면 다음 예제를 사용합니다. 예제에 사용된 메타데이터 뷰에 대한 자세한 내용은 SVV_RELATION_PRIVILEGES를 참조하세요.

```
SELECT DISTINCT namespace_name, relation_name, identity_name, identity_type
FROM svv_relation_privileges
WHERE identity_type='group' AND identity_name='guests';

+----------------+---------------+---------------+---------------+
| namespace_name | relation_name | identity_name | identity_type |
```

```
+----------------+---------------+---------------+---------------+
| public         | table1        | guests        | group         |
+----------------+---------------+---------------+---------------+
| public         | table2        | guests        | group         |
+----------------+---------------+---------------+---------------+
```

다음 예에서는 public 사용자 그룹에서 guests 스키마에 있는 모든 테이블에 대한 모든 권한을 취소한 후 그룹을 삭제합니다.

```
REVOKE ALL ON ALL TABLES IN SCHEMA public FROM GROUP guests;
DROP GROUP guests;
```

DROP IDENTITY PROVIDER

자격 증명 공급자를 삭제합니다. 이 명령은 되돌릴 수 없습니다. 슈퍼 사용자만 자격 증명 공급자를 삭제할 수 있습니다.

구문

```
DROP IDENTITY PROVIDER identity_provider_name [ CASCADE ]
```

파라미터

identity_provider_name

삭제할 자격 증명 공급자의 이름입니다.

CASCADE

자격 증명 공급자를 삭제할 때 해당 자격 증명 공급자에 연결된 사용자 및 역할을 삭제합니다.

예제

다음 예에서는 oauth_provider 자격 증명 공급자를 삭제합니다.

```
DROP IDENTITY PROVIDER oauth_provider;
```

자격 증명 공급자를 삭제하면 일부 사용자가 로그인하지 못하거나 자격 증명 공급자를 사용하도록 구성된 클라이언트 도구를 사용할 수 없습니다.

DROP LIBRARY

데이터베이스에서 사용자 지정 Python 라이브러리를 제거합니다. 라이브러리 소유자 또는 수퍼유저만 이 라이브러리를 삭제할 수 있습니다.

DROP LIBRARY를 트랜잭션 블록(BEGIN … END) 내에서 실행할 수 없습니다. 버전 관리에 대한 자세한 내용은 직렬화 가능 격리 단원을 참조하십시오.

이 명령은 되돌릴 수 없습니다. DROP LIBRARY 명령은 즉시 커밋합니다. 라이브러리에 종속된 UDF가 동시에 실행 중인 경우 UDF는 트랜잭션 내에서 실행 중이더라도 실패할 수 있습니다.

자세한 내용은 CREATE LIBRARY 단원을 참조하십시오.

필수 권한

DROPLIBRARY에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- DROP LIBRARY 권한이 있는 사용자
- 라이브러리 소유자

구문

```
DROP LIBRARY library_name
```

파라미터

library_name

라이브러리의 이름입니다.

마스킹 정책 삭제

모든 데이터베이스에서 동적 데이터 마스킹 정책을 삭제합니다. 아직 하나 이상의 테이블에 연결된 마스킹 정책을 삭제할 수 없습니다. 동적 데이터 마스킹에 대한 자세한 내용은 동적 데이터 마스킹(동적 데이터 마스킹(미리 보기)을 참조하세요.

sys:secadmin 역할이 부여된 수퍼유저와 사용자 또는 역할은 마스킹 정책을 삭제할 수 있습니다.

구문

```
DROP MASKING POLICY policy_name;
```

파라미터

policy_name

삭제될 마스킹 정책의 이름입니다.

DROP MODEL

데이터베이스에서 모델을 제거합니다. 모델 소유자나 슈퍼 사용자만 모델을 삭제할 수 있습니다.

DROP MODEL은 또한 이 모델에서 파생된 모든 연결된 예측 함수, 모델과 관련된 모든 Amazon Redshift 아티팩트 및 모델과 관련된 모든 Amazon S3 데이터를 삭제합니다. 모델이 Amazon SageMaker AI에서 계속 훈련되는 동안 DROP MODEL은 이러한 작업을 취소합니다.

이 명령은 되돌릴 수 없습니다. DROP MODEL 명령은 즉시 커밋됩니다.

필수 권한

DROP MODEL에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- DROP MODEL 권한이 있는 사용자
- 모델 소유자
- 스키마 소유자

구문

```
DROP MODEL [ IF EXISTS ] model_name
```

파라미터

IF EXISTS

지정된 스키마가 이미 존재하는 경우 명령이 아무 것도 변경하지 않고 스키마가 존재한다는 메시지를 반환함을 나타내는 절입니다.

model_name

모델의 이름입니다. 스키마의 모델 이름은 고유해야 합니다.

예시

다음 예에서는 demo_ml.customer_churn 모델을 삭제합니다.

```
DROP MODEL demo_ml.customer_churn
```

DROP MATERIALIZED VIEW

구체화된 보기를 제거합니다.

구체화된 뷰에 대한 자세한 내용은 Amazon Redshift의 구체화된 뷰 섹션을 참조하세요.

구문

```
DROP MATERIALIZED VIEW [ IF EXISTS ] mv_name [ CASCADE | RESTRICT ]
```

파라미터

IF EXISTS

명명된 구체화된 보기가 있는지 확인하기 위해 지정하는 절입니다. 구체화된 보기가 없으면 DROP MATERIALIZED VIEW 명령이 오류 메시지를 반환합니다. 이 절은 존재하지 않는 구체화된 보기를 삭제해도 스크립트가 실패하지 않도록 스크립팅할 때 유용합니다.

mv_name

삭제할 구체화된 보기의 이름입니다.

CASCADE

다른 뷰와 같이, 구체화된 뷰가 종속되는 객체를 자동으로 삭제함을 나타내는 절입니다.

RESTRICT

종속된 객체가 있는 경우 구체화된 뷰를 삭제하지 않음을 나타내는 절입니다. 이 값이 기본값입니다.

사용 관련 참고 사항

구체화된 보기의 소유자만 해당 보기에서 DROP MATERIALIZED VIEW를 사용할 수 있습니다. 수퍼유저 또는 특별히 DROP 권한이 부여된 사용자는 예외일 수 있습니다.

구체화된 뷰에 대한 drop 문을 작성하고 일치하는 이름을 가진 뷰가 있는 경우 DROP VIEW를 사용하도록 지시하는 오류가 발생합니다. DROP MATERIALIZED VIEW IF EXISTS를 사용하는 경우에도 오류가 발생합니다.

예제

다음 예제는 tickets_mv 구체화된 보기를 삭제합니다.

```
DROP MATERIALIZED VIEW tickets_mv;
```

DROP PROCEDURE

프로시저를 삭제합니다. 프로시저를 삭제하려면 프로시저 이름과 입력 인수 데이터 형식(서명)이 모두 필요합니다. OUT 인수를 포함하여 전체 인수 데이터 형식을 포함시킬 수도 있습니다. 프로시저의 서명을 찾으려면 SHOW PROCEDURE 명령을 사용합니다. 프로시저 서명에 대한 자세한 내용은 PG_PROC_INFO 섹션을 참조하세요.

필수 권한

DROP PROCEDURE에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- DROP PROCEDURE 권한이 있는 사용자
- 프로시저 소유자

구문

```
DROP PROCEDURE sp_name ( [ [ argname ] [ argmode ] argtype [, ...] ] )
```

파라미터

sp_name

제거할 프로시저의 이름입니다.

argname

입력 인수의 이름입니다. 프로시저의 자격 증명을 확인하는 데는 인수 데이터 형식만 있으면 되므로 DROP PROCEDURE는 인수 이름을 무시합니다.

argmode

인수 모드입니다. IN, OUT 또는 INOUT일 수 있습니다. OUT 인수는 저장 프로시저를 식별하는 데 사용되지 않으므로 선택 사항입니다.

argtype

입력 인수의 데이터 형식입니다. 지원되는 데이터 형식의 전체 목록은 데이터 타입 섹션을 참조하세요.

예시

다음 예제에서는 `quarterly_revenue`라는 저장 프로시저를 삭제합니다.

```
DROP PROCEDURE quarterly_revenue(volume INOUT bigint, at_price IN numeric,result OUT
 int);
```

DROP RLS POLICY

모든 데이터베이스의 전체 테이블에 대한 행 수준 보안 정책을 삭제합니다.

sys:secadmin 역할이 부여된 수퍼유저와 사용자 또는 역할은 정책을 삭제할 수 있습니다.

구문

```
DROP RLS POLICY [ IF EXISTS ] policy_name [ CASCADE | RESTRICT ]
```

파라미터

IF EXISTS

지정된 정책이 이미 존재하는지 여부를 나타내는 절입니다.

policy_name

정책의 이름입니다.

CASCADE

정책을 삭제하기 전에 연결된 모든 테이블에서 정책을 자동으로 분리하도록 지정하는 절입니다.

RESTRICT

정책이 일부 테이블에 연결되어 있는 경우 정책을 삭제하지 않도록 지정하는 절입니다. 이 값이 기본값입니다.

예시

다음 예에서는 행 수준 보안 정책을 삭제합니다.

```
DROP RLS POLICY policy_concerts;
```

DROP ROLE

데이터베이스에서 역할을 제거합니다. 역할을 만든 역할 소유자, WITH ADMIN 옵션이 있는 사용자 또는 슈퍼 사용자만 역할을 삭제할 수 있습니다.

사용자에게 부여된 역할 또는 이 역할에 종속된 다른 역할은 삭제할 수 없습니다.

필수 권한

DROP ROLE에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- 역할을 만들었거나 WITH ADMIN OPTION 권한이 있는 역할이 부여된 사용자인 역할 소유자.

구문

```
DROP ROLE role_name [ FORCE | RESTRICT ]
```

파라미터

role_name

역할의 이름.

[FORCE | RESTRICT]

기본 설정은 RESTRICT입니다. 다른 역할을 상속한 역할을 삭제하려고 하면 Amazon Redshift에서 오류가 발생합니다. 역할 할당이 있는 경우 FORCE를 사용하여 모든 역할 할당을 제거합니다.

예시

다음 예에서는 `sample_role` 역할을 제거합니다.

```
DROP ROLE sample_role FORCE;
```

다음 예에서는 기본 RESTRICT 옵션을 사용하여 사용자에게 부여된 sample_role1 역할을 삭제하려고 시도합니다.

```
CREATE ROLE sample_role1;
GRANT ROLE sample_role1 TO user1;
DROP ROLE sample_role1;
ERROR:  cannot drop this role since it has been granted on a user
```

사용자에게 부여된 sample_role1을 성공적으로 삭제하려면 FORCE 옵션을 사용합니다.

```
DROP ROLE sample_role1 FORCE;
```

다음 예에서는 기본 RESTRICT 옵션을 사용하여 종속된 다른 역할이 있는 sample_role1 역할을 삭제하려고 시도합니다.

```
CREATE ROLE sample_role1;
CREATE ROLE sample_role2;
GRANT ROLE sample_role1 TO sample_role2;
DROP ROLE sample_role2;
ERROR:  cannot drop this role since it depends on another role
```

종속된 다른 역할이 있는 sample_role2를 성공적으로 삭제하려면 FORCE 옵션을 사용합니다.

```
DROP ROLE sample_role2 FORCE;
```

DROP SCHEMA

스키마를 삭제합니다. 외부 스키마의 경우 스키마와 연관된 외부 데이터베이스도 삭제할 수 있습니다. 이 명령은 되돌릴 수 없습니다.

필수 권한

DROP SCHEMA에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- 스키마 소유자
- DROP SCHEMA 권한이 있는 사용자

구문

```
DROP SCHEMA [ IF EXISTS ] name [, ...]
[ DROP EXTERNAL DATABASE ]
[ CASCADE | RESTRICT ]
```

파라미터

IF EXISTS

지정된 스키마가 존재하지 않는 경우 오류 메시지와 함께 종료하는 대신, 명령이 아무 것도 변경하지 않고 스키마가 존재하지 않는다는 메시지를 반환함을 나타내는 절입니다.

이 절은 스크립트 작성 시 유용하므로, DROP SCHEMA가 존재하지 않는 스키마에 대해 실행되는 경우에는 스크립트가 실패하지 않습니다.

이름

삭제할 스키마의 이름입니다. 여러 스키마 이름을 쉼표로 구분하여 지정할 수 있습니다.

DROP EXTERNAL DATABASE

외부 스키마가 삭제되면 해당 외부 스키마와 연관된 외부 데이터베이스(있는 경우)를 삭제함을 나타내는 절입니다. 외부 데이터베이스가 없는 경우 이 명령은 외부 데이터베이스가 없다는 메시지를 반환합니다. 외부 스키마를 여러 개 삭제하면 지정된 스키마와 연관된 데이터베이스가 모두 삭제됩니다.

외부 데이터베이스에 테이블과 같은 종속 객체가 포함되어 있는 경우 종속 객체도 함께 삭제하려면 CASCADE 옵션을 포함합니다.

외부 데이터베이스를 삭제하는 경우 삭제하려는 데이터베이스와 연관된 기타 모든 외부 스키마에 대한 데이터베이스도 삭제됩니다. 해당 데이터베이스를 사용해 다른 외부 스키마에 정의된 테이블 역시 삭제됩니다.

DROP EXTERNAL DATABASE는 HIVE 메타스토어에 저장된 외부 데이터베이스는 지원하지 않습니다.

CASCADE

스키마에 있는 모든 객체를 자동으로 삭제함을 나타내는 키워드입니다. DROP EXTERNAL DATABASE가 지정되어 있으면 외부 데이터베이스의 모든 객체 역시 삭제됩니다.

RESTRICT

객체를 포함하고 있는 스키마 또는 외부 데이터베이스는 삭제하지 않음을 나타내는 키워드입니다. 이 동작이 기본값입니다.

예제

다음 예에서는 S_SALES로 명명된 스키마를 삭제합니다. 이 예에서는 객체를 포함하고 있는 스키마는 삭제되지 않도록 하는 안전 메커니즘으로서 RESTRICT를 사용합니다. 이 경우에는 스키마 객체를 삭제한 후에 스키마를 삭제해야 합니다.

```
drop schema s_sales restrict;
```

다음 예에서는 S_SALES로 명명된 스키마와 그 스키마에 종속되는 모든 객체를 삭제합니다.

```
drop schema s_sales cascade;
```

다음 예에서는 S_SALES 스키마가 존재하는 경우에는 이를 삭제하고 존재하지 않는 경우에는 아무 작업도 수행하지 않고 메시지를 반환합니다.

```
drop schema if exists s_sales;
```

다음 예에서는 외부 스키마 S_SPECTRUM과 이 스키마와 연관된 외부 데이터베이스를 삭제합니다. 이 예에서는 RESTRICT 사용합니다. 따라서 객체가 포함되어 있으면 스키마와 데이터베이스가 삭제되지 않습니다. 이 경우에는 스키마 및 데이터베이스를 삭제하기 전에 종속 객체를 삭제해야 합니다.

```
drop schema s_spectrum drop external database restrict;
```

다음 예에서는 종속 객체와 함께 여러 스키마와 이러한 스키마와 연관된 외부 데이터를 삭제합니다.

```
drop schema s_sales, s_profit, s_revenue drop external database cascade;
```

DROP TABLE

데이터베이스에서 테이블을 제거합니다.

테이블을 제거하지 않고 여러 행으로 구성된 테이블을 비우려면 DELETE 또는 TRUNCATE 명령을 사용하십시오.

DROP TABLE은 대상 테이블에 존재하는 제약 조건을 제거합니다. 단 한 번의 DROP TABLE 명령으로 여러 테이블을 제거할 수 있습니다.

트랜잭션(BEGIN … END) 내에서 외부 테이블을 포함한 DROP TABLE을 실행할 수 없습니다. 버전 관리에 대한 자세한 내용은 직렬화 가능 격리 단원을 참조하십시오.

그룹에 DROP 권한이 부여된 예를 보려면 GRANT 예시를 참조하세요.

필수 권한

DROP TABLE에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- DROP TABLE 권한이 있는 사용자
- 스키마에 대한 USAGE 권한이 있는 테이블 소유자

구문

```
DROP TABLE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

파라미터

IF EXISTS

지정된 테이블이 존재하지 않는 경우 오류 메시지와 함께 종료하는 대신, 명령이 아무 것도 변경하지 않고 테이블이 존재하지 않는다는 메시지를 반환함을 나타내는 절입니다.

이 절은 스크립트 작성 시 유용하므로, DROP TABLE이 존재하지 않는 테이블에 대해 실행되는 경우에는 스크립트가 실패하지 않습니다.

이름

삭제할 테이블의 이름입니다.

CASCADE

뷰와 같이, 테이블에 종속되는 객체를 자동으로 삭제함을 나타내는 절입니다.

보기, 테이블 같은 다른 데이터베이스 객체에 종속되지 않는 보기를 새로 만들려면 보기 정의에 WITH NO SCHEMA BINDING 절을 포함시키십시오. 자세한 내용은 CREATE VIEW 단원을 참조하십시오.

RESTRICT

테이블에 종속된 객체가 있는 경우 테이블을 삭제하지 않음을 나타내는 절입니다. 이 동작이 기본값입니다.

예시

종속 항목이 없는 테이블 삭제

다음 예에서는 종속 항목이 없는, FEEDBACK이라는 테이블을 생성하고 삭제합니다.

```
create table feedback(a int);

drop table feedback;
```

테이블의 열을 뷰 또는 다른 테이블에서 참조할 경우 Amazon Redshift는 다음과 같은 메시지를 표시합니다.

```
Invalid operation: cannot drop table feedback because other objects depend on it
```

두 테이블 동시 삭제

다음 명령 집합은 FEEDBACK 테이블과 BUYERS 테이블을 생성한 다음, 단일 명령으로 두 테이블을 모두 삭제합니다.

```
create table feedback(a int);

create table buyers(a int);

drop table feedback, buyers;
```

종속 항목이 있는 테이블 삭제

다음 절차에서는 CASCADE 스위치를 사용하여 FEEDBACK이라는 테이블을 삭제하는 방법을 보여줍니다.

먼저 CREATE TABLE 명령을 사용하여 FEEDBACK이라는 간단한 테이블을 만듭니다.

```
create table feedback(a int);
```

다음으로, CREATE VIEW 명령을 사용하여 테이블 FEEDBACK에 종속된 FEEDBACK_VIEW라는 뷰를 만듭니다.

```
create view feedback_view as select * from feedback;
```

다음 예에서는 테이블 FEEDBACK을 삭제하고 뷰 FEEDBACK_VIEW도 삭제합니다. FEEDBACK_VIEW가 테이블 FEEDBACK에 종속되기 때문입니다.

```
drop table feedback cascade;
```

테이블에 대한 종속 항목 보기

테이블에 대한 종속성을 반환하려면 다음 예시를 사용하세요. *my_schema* 및 *my_table*을 실제 스키마와 테이블로 바꾸세요.

```
SELECT dependent_ns.nspname as dependent_schema
, dependent_view.relname as dependent_view
, source_ns.nspname as source_schema
, source_table.relname as source_table
, pg_attribute.attname as column_name
FROM pg_depend
JOIN pg_rewrite ON pg_depend.objid = pg_rewrite.oid
JOIN pg_class as dependent_view ON pg_rewrite.ev_class = dependent_view.oid
JOIN pg_class as source_table ON pg_depend.refobjid = source_table.oid
JOIN pg_attribute ON pg_depend.refobjid = pg_attribute.attrelid
    AND pg_depend.refobjsubid = pg_attribute.attnum
JOIN pg_namespace dependent_ns ON dependent_ns.oid = dependent_view.relnamespace
JOIN pg_namespace source_ns ON source_ns.oid = source_table.relnamespace
WHERE
source_ns.nspname = 'my_schema'
AND source_table.relname = 'my_table'
AND pg_attribute.attnum > 0
ORDER BY 1,2
```

```
LIMIT 10;
```

*my_table*과 종속성을 삭제하려면 다음 예시를 사용하세요. 이 예시는 삭제된 테이블에 대한 모든 종속성도 반환합니다.

```
DROP TABLE my_table CASCADE;

SELECT dependent_ns.nspname as dependent_schema
, dependent_view.relname as dependent_view
, source_ns.nspname as source_schema
, source_table.relname as source_table
, pg_attribute.attname as column_name
FROM pg_depend
JOIN pg_rewrite ON pg_depend.objid = pg_rewrite.oid
JOIN pg_class as dependent_view ON pg_rewrite.ev_class = dependent_view.oid
JOIN pg_class as source_table ON pg_depend.refobjid = source_table.oid
JOIN pg_attribute ON pg_depend.refobjid = pg_attribute.attrelid
    AND pg_depend.refobjsubid = pg_attribute.attnum
JOIN pg_namespace dependent_ns ON dependent_ns.oid = dependent_view.relnamespace
JOIN pg_namespace source_ns ON source_ns.oid = source_table.relnamespace
WHERE
source_ns.nspname = 'my_schema'
AND source_table.relname = 'my_table'
AND pg_attribute.attnum > 0
ORDER BY 1,2
LIMIT 10;

+------------------+----------------+---------------+--------------+-------------+
| dependent_schema | dependent_view | source_schema | source_table | column_name |
+------------------+----------------+---------------+--------------+-------------+
```

IF EXISTS를 사용하여 테이블 삭제

다음 예에서는 FEEDBACK 테이블이 존재하는 경우에는 이를 삭제하고 존재하지 않는 경우에는 아무 작업도 수행하지 않고 메시지를 반환합니다.

```
drop table if exists feedback;
```

DROP USER

데이터베이스에서 사용자를 삭제합니다. 단 한 번의 DROP USER 명령으로 여러 사용자를 삭제할 수 있습니다. 이 명령을 실행하려면 데이터베이스 수퍼유저이거나 DROP USER 권한이 있어야 합니다.

구문

```
DROP USER [ IF EXISTS ] name [, ... ]
```

파라미터

IF EXISTS

지정된 사용자가 존재하지 않는 경우 오류 메시지와 함께 종료하는 대신, 명령이 아무 것도 변경하지 않고 사용자가 존재하지 않는다는 메시지를 반환함을 나타내는 절입니다.

이 절은 스크립트 작성 시 유용하므로, DROP USER가 존재하지 않는 사용자에 대해 실행되는 경우에는 스크립트가 실패하지 않습니다.

이름

제거할 사용자의 이름입니다. 제거할 각 사용자의 이름을 다음으로 제거할 이름과 구분하는 쉼표를 사용하여 여러 사용자를 지정할 수 있습니다.

사용 노트

일반적으로 이름이 `rdsdb`인 사용자 또는 보통 이름이 `awsuser` 또는 `admin`인 데이터베이스의 관리자 사용자는 삭제할 수 없습니다.

스키마, 데이터베이스, 테이블 또는 뷰와 같은 데이터베이스 객체를 소유하거나 데이터베이스, 테이블, 열 또는 그룹에 대한 권한이 있는 사용자는 삭제할 수 없습니다. 그런 사용자를 삭제하려고 하면 다음 오류 중 하나를 수신합니다.

```
ERROR: user "username" can't be dropped because the user owns some object [SQL
 State=55006]

ERROR: user "username" can't be dropped because the user has a privilege on some object
 [SQL State=55006]
```

데이터베이스 사용자가 소유한 객체를 찾는 방법에 대한 자세한 지침은 지식 센터에서 Amazon Redshift에서 '사용자를 삭제할 수 없습니다' 오류를 해결하려면 어떻게 해야 하나요?를 참조하세요.

> **Note**
>
> Amazon Redshift는 현재 데이터베이스만 확인한 후 사용자를 삭제합니다. 사용자가 데이터베이스 객체를 소유하고 있거나 다른 데이터베이스에 있는 객체에 대해 어떤 권한이든 있으면

DROP USER가 오류를 반환하지 않습니다. 다른 데이터베이스에 있는 객체를 소유한 사용자를 삭제하면 해당 객체의 소유자가 'unknown'으로 변경됩니다.

사용자가 객체를 소유한 경우에는 먼저 객체를 삭제하거나 그 소유권을 다른 사용자로 변경한 후 원래 사용자를 삭제합니다. 사용자가 객체에 대한 권한을 가지고 있을 경우 먼저 권한을 취소한 후 사용자를 삭제하십시오. 다음 예에서는 객체 삭제, 소유권 변경 및 권한 취소 후 사용자 삭제를 보여줍니다.

```
drop database dwdatabase;
alter schema dw owner to dwadmin;
revoke all on table dwtable from dwuser;
drop user dwuser;
```

예시

다음 예에서는 paulo라는 사용자를 삭제합니다.

```
drop user paulo;
```

다음 예시에서는 paulo와 martha라는 두 명의 사용자를 삭제합니다.

```
drop user paulo, martha;
```

다음 예에서는 사용자 paulo가 존재하는 경우에는 이를 삭제하고 존재하지 않는 경우에는 아무 작업도 수행하지 않고 메시지를 반환합니다.

```
drop user if exists paulo;
```

DROP VIEW

데이터베이스에서 뷰를 제거합니다. 단 한 번의 DROP VIEW 명령으로 여러 뷰를 삭제할 수 있습니다. 이 명령은 되돌릴 수 없습니다.

필수 권한

DROP VIEW에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- DROP VIEW 권한이 있는 사용자

- 보기 소유자

구문

```
DROP VIEW [ IF EXISTS ] name [, ... ] [ CASCADE | RESTRICT ]
```

파라미터

IF EXISTS

지정된 뷰가 존재하지 않는 경우 오류 메시지와 함께 종료하는 대신, 명령이 아무 것도 변경하지 않고 뷰가 존재하지 않는다는 메시지를 반환함을 나타내는 절입니다.

이 절은 스크립트 작성 시 유용하므로, DROP USER가 존재하지 않는 뷰에 대해 실행되는 경우에는 스크립트가 실패하지 않습니다.

이름

제거할 뷰의 이름입니다.

CASCADE

다른 뷰와 같이, 뷰에 종속되는 객체를 자동으로 삭제함을 나타내는 절입니다.

보기, 테이블 같은 다른 데이터베이스 객체에 종속되지 않는 보기를 새로 만들려면 보기 정의에 WITH NO SCHEMA BINDING 절을 포함시키십시오. 자세한 내용은 CREATE VIEW 단원을 참조하십시오.

CASCADE를 포함하고 삭제된 데이터베이스 객체 수가 10개 이상인 경우 데이터베이스 클라이언트가 요약 결과에 삭제된 객체를 모두 나열하지 않을 수도 있습니다. 이는 일반적으로 SQL 클라이언트 도구에 반환되는 결과에 대한 기본 제한이 있기 때문입니다.

RESTRICT

테이블에 종속된 객체가 있는 경우 뷰를 삭제하지 않음을 나타내는 절입니다. 이 동작이 기본값입니다.

예시

다음 예에서는 event라는 뷰를 삭제합니다.

```
drop view event;
```

종속 항목이 있는 뷰를 제거하려면 CASCADE 옵션을 사용하십시오. 예를 들어, EVENT라는 테이블로 시작해봅시다. 다음 예에서 보듯이, CREATE VIEW 명령을 사용하여 EVENT 테이블의 eventview 뷰를 생성합니다.

```
create view eventview as
select dateid, eventname, catid
from event where catid = 1;
```

이제 첫 번째 뷰 eventview를 기반으로 하는 myeventview라는 두 번째 뷰를 생성합니다.

```
create view myeventview as
select eventname, catid
from eventview where eventname <> ' ';
```

eventview 및 myeventview의 두 가지 뷰가 생성되었습니다.

myeventview 뷰는 eventview를 상위 뷰로 둔 하위 뷰입니다.

eventview 뷰를 삭제하기 위해 사용할 명령은 분명히 다음과 같습니다.

```
drop view eventview;
```

참고로, 이 경우에 이 명령을 실행하면 다음과 같은 오류가 발생합니다.

```
drop view eventview;
ERROR: can't drop view eventview because other objects depend on it
HINT: Use DROP ... CASCADE to drop the dependent objects too.
```

이 문제를 해결하려면 오류 메시지의 제안에 따라 다음 명령을 실행합니다.

```
drop view eventview cascade;
```

eventview 및 myeventview가 모두 올바로 삭제되었습니다.

다음 예에서는 eventview 뷰가 존재하는 경우에는 이를 삭제하고 존재하지 않는 경우에는 아무 작업도 수행하지 않고 메시지를 반환합니다.

```
drop view if exists eventview;
```

END

현재 트랜잭션을 커밋합니다. COMMIT 명령과 정확히 똑같은 기능을 수행합니다.

더 상세한 설명서는 COMMIT 설명서를 참조하십시오.

구문

```
END [ WORK | TRANSACTION ]
```

파라미터

Work

선택적 키워드입니다.

TRANSACTION

선택적 키워드: WORK와 TRANSACTION은 동의어입니다.

예시

다음 예에서는 모두 트랜잭션 블록을 종료하고 트랜잭션을 커밋합니다.

```
end;
```

```
end work;
```

```
end transaction;
```

이런 명령 중 어느 명령이든 실행된 후 Amazon Redshift는 트랜잭션 블록을 종료하고 변경 내용을 커밋합니다.

EXECUTE

이전에 준비된 문을 실행합니다.

구문

```
EXECUTE plan_name [ (parameter [, ...]) ]
```

파라미터

plan_name

실행할 준비된 문의 이름입니다.

파라미터

준비된 문에 대한 파라미터의 실제 값입니다. 이것은 준비된 문을 생성한 PREPARE 명령에서 이 명령 위치에 대해 지정된 데이터 형식과 호환 가능한 유형의 값을 산출하는 표현식이어야 합니다.

사용 노트

EXECUTE는 이전에 준비된 문의 실행에 사용됩니다. 준비된 문은 세션 기간 동안에만 존재하므로 준비된 문은 현재 세션의 앞부분에서 실행된 PREPARE 문에 의해 생성되었음에 틀림없습니다.

이전 PREPARE 문에서 일부 파라미터를 지정한 경우 호환 가능한 파라미터 집합은 EXECUTE 문으로 전달되어야 합니다. 그렇지 않으면 Amazon Redshift가 오류를 반환합니다. 함수와는 달리, 준비된 문은 지정된 파라미터의 유형이나 수를 기준으로 오버로드되지 않습니다. 준비된 문의 이름은 데이터베이스 세션 내에서 고유해야 합니다.

준비된 문에 대해 EXECUTE 명령이 실행될 때 Amazon Redshift는 (지정된 파라미터 값을 기반으로 성능을 개선하기 위해) 쿼리 실행 계획을 선택적으로 수정한 후 준비된 문을 실행할 수 있습니다. 또한, 준비된 문을 새로 실행할 때마다 Amazon Redshift는 EXECUTE 문과 함께 지정된 다양한 파라미터 값을 기반으로 쿼리 실행 계획을 다시 수정할 수 있습니다. Amazon Redshift가 주어진 EXECUTE 문에 대해 선택한 쿼리 실행 계획을 검사하려면 EXPLAIN 명령을 사용합니다.

준비된 문의 생성 및 사용에 대한 자세한 내용과 예는 PREPARE 섹션을 참조하세요.

다음 사항도 참조하세요.

DEALLOCATE, PREPARE

EXPLAIN

쿼리를 실행하지 않고 쿼리 문에 대한 실행 계획을 표시합니다. 쿼리 분석 워크플로에 대한 자세한 내용은 쿼리 분석 워크플로우 섹션을 참조하세요.

구문

```
EXPLAIN [ VERBOSE ] query
```

파라미터

상세 표시

단순한 요약 대신 전체 쿼리 계획을 표시합니다.

query

설명할 쿼리 문입니다. 쿼리는 SELECT, INSERT, CREATE TABLE AS, UPDATE 또는 DELETE 문일 수 있습니다.

사용 노트

EXPLAIN 성능은 때때로 임시 테이블 생성에 걸리는 시간의 영향을 받습니다. 예를 들어, 공통 하위 표현식 최적화를 사용하는 쿼리는 EXPLAIN 출력을 반환하기 위해 임시 테이블을 생성하고 분석해야 합니다. 쿼리 계획은 임시 테이블의 스키마와 통계에 따라 다릅니다. 따라서 이 유형의 쿼리에 대한 EXPLAIN 명령 실행 시간은 예상보다 길 수 있습니다.

다음 명령에만 EXPLAIN을 사용할 수 있습니다.

- SELECT
- SELECT INTO
- CREATE TABLE AS
- INSERT
- UPDATE
- DELETE

EXPLAIN 명령을 DDL(데이터 정의 언어) 또는 데이터베이스 작업 등의 다른 SQL 명령에 사용할 경우 실패하게 됩니다.

EXPLAIN 출력 상대 단위 비용은 Amazon Redshift에서 쿼리 계획을 선택하는 데 사용됩니다. Amazon Redshift는 다양한 리소스 추정치의 크기를 비교하여 계획을 결정합니다.

쿼리 계획 및 실행 단계

특정 Amazon Redshift 쿼리 문에 대한 실행 계획은 쿼리의 실행과 계산을 별개의 단계 순서와 테이블 작업으로 분리하며, 결국은 쿼리에 대한 최종 결과 집합을 생성하게 됩니다. 쿼리 계획에 대한 자세한 내용은 쿼리 처리 섹션을 참조하세요.

다음 표에는 Amazon Redshift에서 사용자가 실행을 위해 제출하는 쿼리를 위한 실행 계획을 개발하는 데 사용할 수 있는 단계가 요약되어 있습니다.

EXPLAIN 연산자	쿼리 실행 단계	설명
스캔:		
순차 스캔	스캔	Amazon Redshift 관계 스캔 또는 테이블 스캔 연산자 또는 단계입니다. 전체 테이블을 처음부터 끝까지 순차적으로 스캔합니다. 또한, WHERE 절과 함께 지정된 경우 모든 행에 대한 쿼리 제약 조건을 평가합니다(Filter). INSERT, UPDATE 및 DELETE 문의 실행에도 사용됩니다.
JOINS: Amazon Redshift는 조인되는 테이블의 물리적 설계, 조인에 필요한 데이터의 위치, 쿼리 자체의 특정 속성을 기반으로 여러 가지 조인 연산자를 사용합니다. 하위 쿼리 스캔 -- 하위 쿼리 스캔 및 추가는 UNION 쿼리 실행에 사용됩니다.		
중첩 루프	nloop	가장 덜 최적화된 조인으로, 주로 크로스 조인(조인 조건이 없는 데카르트 곱)과 일부 부등식 조인에 사용됩니다.
해시 조인	hjoin	내부 조인과 왼쪽 및 오른쪽 외부 조인에도 사용되며 일반적으로 중첩 루프 조인보다 빠릅니다. 해시 조인은 외부 테이블을 읽고 조인 열을 해시하고 내부 해시 테이블에서 일치하는 항목을 찾습니다. 단계가 디스크로 분산될 수 있습니다. (hjoin의 내부 입력은 디스크를 기반으로 할 수 있는 해시 단계입니다.)
병합 조인	mjoin	내부 조인과 외부 조인에도 사용됩니다(조인 열에서 배포 및 정렬이 모두 이루어지는 조인 테이

EXPLAIN 연산자	쿼리 실행 단계	설명
		블의 경우). 일반적으로 다른 비용 고려 사항은 포함하지 않은 가장 빠른 Amazon Redshift 조인 알고리즘입니다.
AGGREGATION: 집계 함수 및 GROUP BY 작업과 관련된 쿼리에 사용되는 연산자와 단계입니다.		
집계	aggr	스칼라 집계 함수를 위한 연산자/단계입니다.
HashAggregate	aggr	그룹화된 집계 함수를 위한 연산자/단계입니다. 해시 테이블이 디스크로 분산된 덕분에 디스크에서 작동할 수 있습니다.
GroupAggregate	aggr	force_hash_grouping 설정을 위한 Amazon Redshift 구성 설정이 해제된 경우 그룹화된 집계 쿼리를 위해 때때로 선택되는 연산자입니다.
SORT: 쿼리가 결과 집합을 정렬하거나 병합해야 할 때 사용되는 연산자와 단계입니다.		
정렬	정렬	Sort는 ORDER BY 절에 의해 지정된 정렬은 물론이고, UNION 및 조인 등의 기타 작업에 의해 지정된 정렬도 수행합니다. 디스크에서 작동할 수 있습니다.
병합	merge	병렬로 수행되는 작업에서 파생된, 즉시 정렬된 결과를 바탕으로 쿼리의 최종 정렬 결과를 생성합니다.
EXCEPT, INTERSECT 및 UNION 작업:		
SetOp Except [Distinct]	hjoin	EXCEPT 쿼리에 사용됩니다. 입력 해시가 디스크를 기반으로 할 수 있다는 사실의 이점을 바탕으로 디스크에서 작동할 수 있습니다.
Hash Intersect [Distinct]	hjoin	INTERSECT 쿼리에 사용됩니다. 입력 해시가 디스크를 기반으로 할 수 있다는 사실의 이점을 바탕으로 디스크에서 작동할 수 있습니다.

EXPLAIN 연산자	쿼리 실행 단계	설명
Append [All |Distinct]	저장	UNION 및 UNION ALL 쿼리를 구현하기 위해 Subquery Scan과 함께 사용되는 Append입니다. "save"의 이점을 바탕으로 디스크에서 작동할 수 있습니다.
기타:		
Hash	hash	내부 조인과 왼쪽 및 오른쪽 외부 조인도 사용됩니다(해시 조인에 대한 입력 제공). Hash 연산자는 조인의 내부 테이블에 대한 해시 테이블을 생성합니다. (내부 테이블은 일치 여부가 검사되는 테이블이며, 두 테이블이 조인된 경우에는 보통 둘 중 더 작은 테이블입니다.)
Limit	제한	LIMIT 절을 평가합니다.
Materialize	저장	중첩 루프 조인과 일부 병합 조인에 대한 입력을 위한 행을 구체화합니다. 디스크에서 작동할 수 있습니다.
--	parse	로드 중에 텍스트 입력 데이터를 구문 분석하는 데 사용됩니다.
--	project	열 및 컴퓨팅 표현식, 즉 프로젝트 데이터를 다시 배치하는 데 사용됩니다.
결과	--	어떤 테이블 액세스도 관련되지 않는 스칼라 함수를 실행합니다.
--	return	리더 또는 클라이언트로 행을 반환합니다.
Subplan	--	특정 하위 쿼리에 사용됩니다.
고유	고유	SELECT DISTINCT 및 UNION 쿼리에서 중복을 제거합니다.

EXPLAIN 연산자	쿼리 실행 단계	설명
창	Window	집계 및 순위 창 함수를 계산합니다. 디스크에서 작동할 수 있습니다.
네트워크 운영:		
네트워크(브로드캐스트)	bcast	브로드캐스트는 Join Explain 연산자와 단계의 속성이기도 합니다.
네트워크(배포)	dist	데이터 웨어하우스 클러스터에 의한 병렬 처리를 위한 컴퓨팅 노드로 행을 배포합니다.
네트워크(리더로 보내기)	return	추가적인 처리를 위해 리더로 결과를 다시 보냅니다.
DML 작업(데이터를 수정하는 연산자):		
삽입(결과 사용)	삽입	데이터를 삽입합니다.
삭제(스캔 + 필터)	삭제	데이터를 삭제합니다. 디스크에서 작동할 수 있습니다.
업데이트(스캔 + 필터)	delete, insert	삭제 및 삽입으로 구현됩니다.

RLS에 EXPLAIN 사용

쿼리에 행 수준 보안(RLS) 정책이 적용되는 테이블이 포함되어 있는 경우 EXPLAIN은 특별한 RLS SecureScan 노드를 표시합니다. 또한 Amazon Redshift는 STL_EXPLAIN 시스템 테이블에 동일한 노드 유형을 로깅합니다. EXPLAIN은 dim_tbl에 적용되는 RLS 조건자를 표시하지 않습니다. RLS SecureScan 노드 유형은 현재 사용자에게 보이지 않는 추가 작업이 실행 계획에 포함되어 있음을 나타내는 표시자로 사용됩니다.

다음 예는 RLS SecureScan 노드를 보여줍니다.

```
EXPLAIN
SELECT D.cint
FROM fact_tbl F INNER JOIN dim_tbl D ON F.k_dim = D.k
WHERE F.k_dim / 10 > 0;
                                QUERY PLAN
```

```
------------------------------------------------------------------------
 XN Hash Join DS_DIST_ALL_NONE  (cost=0.08..0.25 rows=1 width=4)
   Hash Cond: ("outer".k_dim = "inner"."k")
   ->  *XN* *RLS SecureScan f  (cost=0.00..0.14 rows=2 width=4)*
         Filter: ((k_dim / 10) > 0)
   ->  XN Hash  (cost=0.07..0.07 rows=2 width=8)
         ->  XN Seq Scan on dim_tbl d  (cost=0.00..0.07 rows=2 width=8)
               Filter: (("k" / 10) > 0)
```

RLS가 적용되는 전체 쿼리 계획을 살펴볼 수 있도록, Amazon Redshift는 EXPLAIN RLS 시스템 권한을 제공합니다. 이 권한이 부여된 사용자는 RLS 조건자까지 포함한 전체 쿼리 계획을 살펴볼 수 있습니다.

다음 예는 RLS SecureScan 노드 아래의 추가 Seq Scan에 RLS 정책 조건자(k_dim > 1)도 포함되어 있음을 보여줍니다.

```
EXPLAIN SELECT D.cint
FROM fact_tbl F INNER JOIN dim_tbl D ON F.k_dim = D.k
WHERE F.k_dim / 10 > 0;
                                  QUERY PLAN
------------------------------------------------------------------------------
 XN Hash Join DS_DIST_ALL_NONE  (cost=0.08..0.25 rows=1 width=4)
   Hash Cond: ("outer".k_dim = "inner"."k")
   *->  XN RLS SecureScan f  (cost=0.00..0.14 rows=2 width=4)
         Filter: ((k_dim / 10) > 0)*
         ->  *XN* *Seq Scan on fact_tbl rls_table  (cost=0.00..0.06 rows=5 width=8)
               Filter: (k_dim > 1)*
   ->  XN Hash  (cost=0.07..0.07 rows=2 width=8)
         ->  XN Seq Scan on dim_tbl d  (cost=0.00..0.07 rows=2 width=8)
               Filter: (("k" / 10) > 0)
```

사용자에게 EXPLAIN RLS 권한이 부여되지만, Amazon Redshift는 RLS 조건자를 포함한 전체 쿼리 계획을 STL_EXPLAIN 시스템 테이블에 로깅합니다. 이 권한이 부여되지 않은 상태에서 실행되는 쿼리는 RLS 내부 정보 없이 로깅됩니다. EXPLAIN RLS 권한을 부여하거나 제거해도 Amazon Redshift가 이전 쿼리와 관련하여 STL_EXPLANT에 로깅한 내용은 변경되지 않습니다.

AWS Lake Formation-RLS 보호 Redshift 관계

다음 예는 Lake Formation-RLS 관계를 보는 데 사용할 수 있는 LF SecureScan 노드를 보여줍니다.

```
EXPLAIN
```

```
SELECT *
FROM lf_db.public.t_share
WHERE a > 1;
QUERY PLAN
------------------------------------------------------------
XN LF SecureScan t_share  (cost=0.00..0.02 rows=2 width=11)
(2 rows)
```

예시

> **Note**
>
> 다음 예의 경우 샘플 출력은 Amazon Redshift 구성에 따라 다를 수 있습니다.

다음 예에서는 EVENT 및 VENUE 테이블에서 EVENTID, EVENTNAME, VENUEID 및 VENUENAME을 선택하는 쿼리를 위한 쿼리 계획을 반환합니다.

```
explain
select eventid, eventname, event.venueid, venuename
from event, venue
where event.venueid = venue.venueid;
```

```
                              QUERY PLAN
------------------------------------------------------------------------
XN Hash Join DS_DIST_OUTER  (cost=2.52..58653620.93 rows=8712 width=43)
Hash Cond: ("outer".venueid = "inner".venueid)
->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798 width=23)
->  XN Hash  (cost=2.02..2.02 rows=202 width=22)
->  XN Seq Scan on venue  (cost=0.00..2.02 rows=202 width=22)
(5 rows)
```

다음 예에서는 상세한 출력을 포함한 동일한 쿼리를 위한 쿼리 계획을 반환합니다.

```
explain verbose
select eventid, eventname, event.venueid, venuename
from event, venue
where event.venueid = venue.venueid;
```

```
                              QUERY PLAN
```

```
------------------------------------------------------------------------
{HASHJOIN
:startup_cost 2.52
:total_cost 58653620.93
:plan_rows 8712
:plan_width 43
:best_pathkeys <>
:dist_info DS_DIST_OUTER
:dist_info.dist_keys (
TARGETENTRY
{
VAR
:varno 2
:varattno 1
...

XN Hash Join DS_DIST_OUTER  (cost=2.52..58653620.93 rows=8712 width=43)
Hash Cond: ("outer".venueid = "inner".venueid)
->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798 width=23)
->  XN Hash  (cost=2.02..2.02 rows=202 width=22)
->  XN Seq Scan on venue  (cost=0.00..2.02 rows=202 width=22)
(519 rows)
```

다음 예에서는 CREATE TABLE AS(CTAS) 문을 위한 쿼리 계획을 반환합니다.

```
explain create table venue_nonulls as
select * from venue
where venueseats is not null;

QUERY PLAN
-------------------------------------------------------------
XN Seq Scan on venue  (cost=0.00..2.02 rows=187 width=45)
Filter: (venueseats IS NOT NULL)
(2 rows)
```

FETCH

커서를 사용하여 행을 가져옵니다. 커서 선언에 대한 자세한 내용은 DECLARE 섹션을 참조하세요.

FETCH는 커서 내에서 현재 위치를 기준으로 행을 가져옵니다. 생성되는 커서는 첫 번째 행 앞에 배치됩니다. FETCH 후, 커서는 마지막으로 가져온 행에 배치됩니다. FETCH ALL을 실행한 후와 같이, 사용 가능한 행의 끝에서 FETCH 실행이 끝나는 경우 커서는 마지막 행 다음에 배치된 상태로 남습니다.

FORWARD 0은 커서를 이동하지 않고 현재 행을 가져옵니다. 즉, 가장 최근에 가져온 행을 가져옵니다. 커서가 첫 번째 행 앞이나 마지막 행 뒤에 있는 경우에는 아무런 행도 반환되지 않습니다.

커서의 첫 행을 가져올 때, 필요한 경우 메모리나 디스크에서 리더 노드에 전체 결과 집합이 구체화됩니다. 큰 결과 집합을 가진 커서를 사용하면 성능에 나쁜 영향을 미칠 가능성이 있으므로, 가급적이면 다른 접근 방식을 사용하는 것이 좋습니다. 자세한 내용은 커서 사용 시 성능 고려사항 단원을 참조하십시오.

자세한 내용은 DECLARE, CLOSE 섹션을 참조하세요.

구문

```
FETCH [ NEXT | ALL | {FORWARD [ count | ALL ] } ] FROM cursor
```

파라미터

next

다음 행을 가져옵니다. 이 값이 기본값입니다.

ALL

나머지 행을 전부 가져옵니다. (FORWARD ALL과 동일합니다.) ALL은 단일 노드 클러스터에 대해 지원되지 않습니다.

FORWARD [count | ALL]

다음 count개의 행 또는 나머지 행 전부를 가져옵니다. FORWARD 0은 현재 행을 가져옵니다. 단일 노드 클러스터의 경우 count의 최댓값은 1000입니다. 단일 노드 클러스터에서는 FORWARD ALL이 지원되지 않습니다.

cursor

새 커서의 이름입니다.

FETCH 예

다음 예에서는 LOLLAPALOOZA로 명명된 커서를 선언하여 롤라팔루자(LOLLAPALOOZA) 행사를 위한 판매 정보를 선택한 후 커서를 사용하여 결과 집합에서 행을 가져옵니다.

```
-- Begin a transaction
```

```
begin;

-- Declare a cursor

declare lollapalooza cursor for
select eventname, starttime, pricepaid/qtysold as costperticket, qtysold
from sales, event
where sales.eventid = event.eventid
and eventname='Lollapalooza';

-- Fetch the first 5 rows in the cursor lollapalooza:

fetch forward 5 from lollapalooza;

  eventname   |      starttime      | costperticket | qtysold
--------------+---------------------+---------------+---------
 Lollapalooza | 2008-05-01 19:00:00 |   92.00000000 |       3
 Lollapalooza | 2008-11-15 15:00:00 |  222.00000000 |       2
 Lollapalooza | 2008-04-17 15:00:00 |  239.00000000 |       3
 Lollapalooza | 2008-04-17 15:00:00 |  239.00000000 |       4
 Lollapalooza | 2008-04-17 15:00:00 |  239.00000000 |       1
(5 rows)

-- Fetch the next row:

fetch next from lollapalooza;

  eventname   |      starttime      | costperticket | qtysold
--------------+---------------------+---------------+---------
 Lollapalooza | 2008-10-06 14:00:00 |  114.00000000 |       2

-- Close the cursor and end the transaction:

close lollapalooza;
commit;
```

GRANT

사용자 또는 역할에 대한 액세스 권한을 정의합니다.

권한에는 테이블 및 뷰의 데이터 읽기, 데이터 쓰기, 테이블 생성 및 테이블 삭제와 같은 액세스 옵션이 포함됩니다. 테이블, 데이터베이스, 스키마, 함수, 프로시저, 언어 또는 열에 대한 특정 권한을 부여하려면 이 명령을 사용합니다. 데이터베이스 객체에서 권한을 취소하려면 REVOKE 명령을 사용하세요.

권한에는 다음과 같은 데이터 공유 생산자 액세스 옵션도 포함됩니다.

- 소비자 네임스페이스 및 계정에 대한 데이터 공유 액세스 권한 부여
- 데이터 공유에 객체를 추가하거나 데이터 공유에서 객체를 제거하여 데이터 공유를 변경할 수 있는 권한 부여
- 데이터 공유에 소비자 네임스페이스를 추가하거나 데이터 공유에서 소비자 네임스페이스를 제거하여 데이터 공유를 공유할 수 있는 권한 부여

데이터 공유 소비자 액세스 옵션은 다음과 같습니다.

- 데이터 공유에서 생성된 데이터베이스 또는 해당 데이터베이스를 가리키는 외부 스키마에 대한 전체 액세스 권한을 사용자에게 부여
- 로컬 데이터베이스 객체에서와 마찬가지로 데이터 공유에서 생성된 데이터베이스에 대한 객체 수준 권한을 사용자에게 부여. 이 수준의 권한을 부여하려면 데이터 공유에서 데이터베이스를 만들 때 WITH PERMISSIONS 절을 사용해야 합니다. 자세한 내용은 데이터베이스 생성 단원을 참조하십시오.

데이터 공유 권한에 대한 자세한 내용은 클러스터 내 및 클러스터 간 데이터 공유 섹션을 참조하세요.

데이터베이스 권한을 관리하고 데이터와 관련하여 사용자가 수행할 수 있는 작업을 제어하는 역할을 부여할 수도 있습니다. 역할을 정의하고 사용자에게 역할을 할당하여 사용자가 수행할 수 있는 작업을 제한할 수 있습니다 예를 들어, CREATE TABLE 및 INSERT 명령만 사용하도록 제한할 수 있습니다. CREATE ROLE 명령에 대한 자세한 내용은 the section called “역할 생성” 섹션을 참조하세요. Amazon Redshift에는 사용자에게 특정 권한을 부여하는 데 사용할 수 있는 몇 가지 시스템 정의 역할이 있습니다. 자세한 내용은 the section called “Amazon Redshift 시스템 정의 역할” 단원을 참조하십시오.

ON SCHEMA 구문을 사용하는 데이터베이스 사용자 및 사용자 그룹에 외부 스키마에 대한 사용 권한만 부여하거나 취소할 수 있습니다. AWS Lake Formation에서 ON EXTERNAL SCHEMA를 사용하는 경우 AWS Identity and Access Management(IAM) 역할에 대한 권한의 GRANT 및 REVOKE만 허용됩니다. 권한 목록은 구문을 참조하세요.

저장 프로시저의 경우 부여할 수 있는 유일한 권한은 EXECUTE입니다.

트랜잭션 블록(BEGIN ... END) 내에서 GRANT(외부 리소스에 대해)를 실행할 수 없습니다. 버전 관리에 대한 자세한 내용은 직렬화 가능 격리 단원을 참조하십시오.

데이터베이스에 대해 사용자에게 부여된 권한을 확인하려면 HAS_DATABASE_PRIVILEGE를 사용합니다. 스키마에 대해 사용자에게 부여된 사용 권한을 확인하려면 HAS_SCHEMA_PRIVILEGE를 사용합니다. 테이블에 대해 사용자에게 부여된 사용 권한을 확인하려면 HAS_TABLE_PRIVILEGE를 사용합니다.

구문

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | DROP | REFERENCES | ALTER | TRUNCATE }
 [,...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...] | ALL TABLES IN SCHEMA schema_name [, ...] }
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { { CREATE | TEMPORARY | TEMP | ALTER } [,...] | ALL [ PRIVILEGES ] }
    ON DATABASE db_name [, ...]
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { { CREATE | USAGE | ALTER | DROP } [,...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON { FUNCTION function_name ( [ [ argname ] argtype [, ...] ] ) [, ...] | ALL
 FUNCTIONS IN SCHEMA schema_name [, ...] }
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON { PROCEDURE procedure_name ( [ [ argname ] argtype [, ...] ] ) [, ...] | ALL
 PROCEDURES IN SCHEMA schema_name [, ...] }
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT USAGE
    ON LANGUAGE language_name [, ...]
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { { ALTER | DROP} [,...] | ALL [ PRIVILEGES ] }
    ON COPY JOB job_name [,...]
```

```
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]
```

테이블에 대한 열 수준 권한 부여

다음은 Amazon Redshift 테이블 및 뷰에 대한 열 수준 권한에 대한 구문입니다.

```
GRANT { { SELECT | UPDATE } ( column_name [, ...] ) [, ...] | ALL [ PRIVILEGES ]
 ( column_name [,...] ) }
     ON { [ TABLE ] table_name [, ...] }

     TO { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
```

ASSUMEROLE 권한 부여

다음은 지정된 역할을 가진 사용자 및 그룹에 부여된 ASSUMEROLE 권한에 대한 구문입니다. ASSUMEROLE 권한을 사용하기 시작하려면 ASSUMEROLE 권한 부여에 대한 사용 노트 섹션을 참조하세요.

```
GRANT ASSUMEROLE
       ON { 'iam_role' [, ...] | default | ALL }
       TO { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
       FOR { ALL | COPY | UNLOAD | EXTERNAL FUNCTION | CREATE MODEL } [, ...]
```

Redshift Spectrum과 Lake Formation의 통합을 위한 권한 부여

다음은 Lake Formation과 Redshift Spectrum 통합을 위한 구문입니다.

```
GRANT { SELECT | ALL [ PRIVILEGES ] } ( column_list )
    ON EXTERNAL TABLE schema_name.table_name
    TO { IAM_ROLE iam_role } [, ...] [ WITH GRANT OPTION ]

GRANT { { SELECT | ALTER | DROP | DELETE | INSERT }  [, ...] | ALL [ PRIVILEGES ] }
    ON EXTERNAL TABLE schema_name.table_name [, ...]
    TO { { IAM_ROLE iam_role } [, ...] | PUBLIC } [ WITH GRANT OPTION ]

GRANT { { CREATE | ALTER | DROP }  [, ...] | ALL [ PRIVILEGES ] }
    ON EXTERNAL SCHEMA schema_name [, ...]
    TO { IAM_ROLE iam_role } [, ...] [ WITH GRANT OPTION ]
```

데이터 공유 권한 부여

생산자 측 데이터 공유 권한

다음은 GRANT를 사용하여 사용자 또는 역할에 ALTER 또는 SHARE 권한을 부여할 때 사용하는 구문입니다. 사용자는 ALTER 권한으로 데이터 공유를 변경하거나 SHARE 권한으로 소비자에게 사용 권한을 부여할 수 있습니다. ALTER 및 SHARE는 데이터 공유에서 사용자 및 역할에 부여할 수 있는 유일한 권한입니다.

```
GRANT { ALTER | SHARE } ON DATASHARE datashare_name
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]
```

다음은 Amazon Redshift에서 데이터 공유 사용 권한에 GRANT를 사용하는 구문입니다. USAGE 권한을 사용하여 소비자에게 데이터 공유에 대한 액세스 권한을 부여합니다. 사용자 또는 사용자 그룹에는 이 권한을 부여할 수 없습니다. 이 권한은 GRANT 문에 대한 WITH GRANT OPTION도 지원하지 않습니다. 이전에 데이터 공유에 대한 SHARE 권한이 부여된 사용자 또는 사용자 그룹만 이 유형의 GRANT 문을 실행할 수 있습니다.

```
GRANT USAGE
    ON DATASHARE datashare_name
    TO NAMESPACE 'namespaceGUID' | ACCOUNT 'accountnumber' [ VIA DATA CATALOG ]
```

다음은 Lake Formation 계정에 데이터 공유 사용 권한을 부여하는 방법의 예입니다.

```
GRANT USAGE ON DATASHARE salesshare TO ACCOUNT '123456789012' VIA DATA CATALOG;
```

소비자 측 데이터 공유 권한

다음은 datashare에서 생성된 특정 데이터베이스 또는 스키마에 대한 GRANT datashare 사용 권한에 대한 구문입니다.

소비자가 데이터 공유에서 생성된 데이터베이스에 액세스하는 데 필요한 추가 권한은 데이터 공유에서 데이터베이스를 생성하는 데 사용된 CREATE DATABASE 명령에서 WITH PERMISSIONS 절을 사용했는지에 따라 달라집니다. CREATE DATABASE 명령과 WITH PERMISSIONS에 대한 자세한 내용은 데이터베이스 생성 섹션을 참조하세요.

WITH PERMISSIONS 절을 사용하지 않고 생성된 데이터베이스

WITH PERMISSIONS 절을 사용하지 않은 데이터 공유에서 생성된 데이터베이스에 USAGE를 부여하면 공유 데이터베이스의 객체에 대해 별도로 권한을 부여할 필요가 없습니다. WITH PERMISSIONS

절을 사용하지 않은 데이터 공유에서 생성된 데이터베이스에 대해 사용 권한을 부여받은 엔터티는 데이터베이스의 모든 객체에 대한 액세스 권한을 자동으로 갖게 됩니다.

WITH PERMISSIONS 절을 사용하여 생성된 데이터베이스

WITH PERMISSIONS 절을 사용하여 데이터베이스가 생성된 데이터 공유에 USAGE를 부여할 때 로컬 데이터베이스 객체에 대한 권한을 부여하는 것과 마찬가지로 공유 데이터베이스의 데이터베이스 객체에 대한 관련 권한을 소비자 측 자격 증명에 부여해야 액세스할 수 있습니다. 데이터 공유에서 만든 데이터베이스의 객체에 권한을 부여하려면 세 부분으로 구성된 `database_name.schema_name.object_name` 구문을 사용합니다. 공유 데이터베이스 내의 공유 스키마를 가리키는 외부 스키마의 객체에 권한을 부여하려면 두 부분으로 구성된 `schema_name.object_name` 구문을 사용합니다.

```
GRANT USAGE ON { DATABASE shared_database_name [, ...] | SCHEMA shared_schema}
    TO { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
```

범위가 지정된 권한 부여

범위가 지정된 권한을 사용하면 데이터베이스 또는 스키마 내 특정 유형의 모든 객체에 대한 권한을 사용자 또는 역할에 부여할 수 있습니다. 범위가 지정된 권한이 있는 사용자와 역할은 데이터베이스 또는 스키마 내의 모든 현재 및 미래 객체에 대한 지정된 권한을 갖습니다.

SVV_DATABASE_PRIVILEGES에서 데이터베이스 수준 범위 지정 권한의 범위를 볼 수 있습니다. SVV_SCHEMA_PRIVILEGES에서 스키마 수준 범위 지정 권한의 범위를 볼 수 있습니다.

범위가 지정된 권한에 대한 자세한 내용은 범위가 지정된 권한 섹션을 참조하세요.

다음은 사용자 또는 역할에 범위가 지정된 권한을 부여할 때 사용하는 구문입니다.

```
GRANT { CREATE | USAGE | ALTER | DROP } [,...] | ALL [ PRIVILEGES ] }
FOR SCHEMAS IN
DATABASE db_name
TO { username [ WITH GRANT OPTION ] | ROLE role_name } [, ...]

GRANT
{ { SELECT | INSERT | UPDATE | DELETE | DROP | ALTER | TRUNCATE | REFERENCES }
 [, ...] } | ALL [PRIVILEGES] } }
FOR TABLES IN
{SCHEMA schema_name [DATABASE db_name ] | DATABASE db_name }
TO { username [ WITH GRANT OPTION ] | ROLE role_name} [, ...]
```

```
GRANT { EXECUTE | ALL [ PRIVILEGES ] }
FOR FUNCTIONS IN
{SCHEMA schema_name [DATABASE db_name ] | DATABASE db_name }
TO { username [ WITH GRANT OPTION ] | ROLE role_name | } [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
FOR PROCEDURES IN
{SCHEMA schema_name [DATABASE db_name ] | DATABASE db_name }
TO { username [ WITH GRANT OPTION ] | ROLE role_name | } [, ...]

GRANT USAGE
FOR LANGUAGES IN
{DATABASE db_name}
TO { username [ WITH GRANT OPTION ] | ROLE role_name } [, ...]

GRANT { { CREATE | ALTER | DROP} [,...] | ALL [ PRIVILEGES ] }
FOR COPY JOBS
IN DATABASE db_name
TO { username [ WITH GRANT OPTION ] | ROLE role_name } [, ...]
```

범위가 지정된 권한은 함수에 대한 권한과 프로시저에 대한 권한을 구별하지 않습니다. 예를 들어 다음 문은 Sales_schema 스키마의 함수와 프로시저 모두에 대한 EXECUTE 권한을 bob에게 부여합니다.

```
GRANT EXECUTE FOR FUNCTIONS IN SCHEMA Sales_schema TO bob;
```

기계 학습 권한 부여

다음은 Amazon Redshift의 기계 학습 모델 권한을 위한 구문입니다.

```
GRANT CREATE MODEL
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON MODEL model_name [, ...]

    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]
```

역할 권한 부여

다음은 Amazon Redshift에서 역할을 부여할 때 사용하는 구문입니다.

```
GRANT { ROLE role_name } [, ...] TO { { user_name [ WITH ADMIN OPTION ] } |
 ROLE role_name }[, ...]
```

다음은 Amazon Redshift에서 역할에 시스템 권한을 부여하기 위한 구문입니다. 사용자가 아닌 역할에만 권한을 부여할 수 있습니다.

```
GRANT
  {
    { CREATE USER | DROP USER | ALTER USER |
    CREATE SCHEMA | DROP SCHEMA |
    ALTER DEFAULT PRIVILEGES |
    ACCESS CATALOG | ACCESS SYSTEM TABLE
    CREATE TABLE | DROP TABLE | ALTER TABLE |
    CREATE OR REPLACE FUNCTION | CREATE OR REPLACE EXTERNAL FUNCTION |
    DROP FUNCTION |
    CREATE OR REPLACE PROCEDURE | DROP PROCEDURE |
    CREATE OR REPLACE VIEW | DROP VIEW |
    CREATE MODEL | DROP MODEL |
    CREATE DATASHARE | ALTER DATASHARE | DROP DATASHARE |
    CREATE LIBRARY | DROP LIBRARY |
    CREATE ROLE | DROP ROLE |
    TRUNCATE TABLE
    VACUUM | ANALYZE | CANCEL |
    IGNORE RLS | EXPLAIN RLS |
    EXPLAIN MASKING }[, ...]
  }
  | { ALL [ PRIVILEGES ] }
TO ROLE role_name [, ...]
```

행 수준 보안 정책 필터에 대한 설명 권한 부여

다음은 EXPLAIN 계획에서 쿼리의 행 수준 보안 정책 필터를 설명할 권한을 부여하는 구문입니다. REVOKE 문을 사용하여 이 권한을 취소할 수 있습니다.

```
GRANT EXPLAIN RLS TO ROLE rolename
```

다음은 쿼리에 대한 행 수준 보안 정책을 우회할 수 있는 권한을 부여하는 구문입니다.

```
GRANT IGNORE RLS TO ROLE rolename
```

정책 객체에 RLS 조회 테이블에 대한 권한 부여

다음은 지정된 행 수준 보안 정책에 권한을 부여하는 구문입니다.

```
GRANT SELECT ON [ TABLE ] table_name [, ...]
TO RLS POLICY policy_name [, ...]
```

파라미터

SELECT

SELECT 문을 사용하여 테이블 또는 뷰에서 데이터를 선택하는 권한을 부여합니다. UPDATE 또는 DELETE 작업을 위해 기존의 열 값을 참조하는 데도 SELECT 권한이 필요합니다.

INSERT

INSERT 문 또는 COPY 문을 사용하여 데이터를 테이블로 로드하는 권한을 부여합니다.

UPDATE

UPDATE 문을 사용하여 테이블 열을 업데이트하는 권한을 부여합니다. UPDATE 작업에서는 테이블 열을 참조하여 업데이트할 행을 결정하거나 열에 대한 값을 새로 계산해야 하므로, SELECT 권한도 필요합니다.

DELETE

테이블에서 데이터 행을 삭제하는 권한을 부여합니다. DELETE 작업에서는 테이블 열을 참조하여 삭제할 행을 결정해야 하므로, SELECT 권한도 필요합니다.

DROP

데이터베이스 객체에 따라, 사용자 또는 역할에 다음 권한을 부여합니다.

- 테이블의 경우 DROP은 테이블 또는 뷰를 삭제할 수 있는 권한을 부여합니다. 자세한 내용은 DROP TABLE 단원을 참조하십시오.
- 데이터베이스의 경우 DROP은 데이터베이스를 삭제할 수 있는 권한을 부여합니다. 자세한 내용은 DROP DATABASE 단원을 참조하십시오.
- 스키마의 경우 DROP은 스키마를 삭제할 수 있는 권한을 부여합니다. 자세한 내용은 DROP SCHEMA 단원을 참조하십시오.

REFERENCES

외래 키 제약 조건을 생성하는 권한을 부여합니다. 참조되는 테이블과 참조하는 테이블 모두에 대해 이 권한을 허용해야 하며, 그러지 않으면 사용자가 제약 조건을 생성할 수 없습니다.

ALTER

데이터베이스 객체에 따라, 사용자 또는 사용자 그룹에 다음 권한을 부여합니다.

- 테이블의 경우 ALTER는 테이블 또는 뷰를 변경할 수 있는 권한을 부여합니다. 자세한 내용은 ALTER TABLE 단원을 참조하십시오.
- 데이터베이스의 경우 ALTER는 데이터베이스를 변경할 수 있는 권한을 부여합니다. 자세한 내용은 ALTER DATABASE 단원을 참조하십시오.
- 스키마의 경우 ALTER는 스키마를 변경할 수 있는 권한을 부여합니다. 자세한 내용은 ALTER SCHEMA 단원을 참조하십시오.
- 외부 테이블의 경우 ALTER는 Lake Formation에 활성화된 AWS Glue Data Catalog의 테이블을 변경할 수 있는 권한을 부여합니다. 이 권한은 Lake Formation을 사용하는 경우에만 적용됩니다.

TRUNCATE

테이블을 자를 수 있는 권한을 부여합니다. 이 권한이 없으면 테이블 소유자 또는 수퍼유저만 테이블을 자를 수 있습니다. TRUNCATE 명령에 대한 자세한 내용은 the section called “TRUNCATE” 섹션을 참조하세요.

ALL [PRIVILEGES]

지정된 사용자 또는 역할에 사용 가능한 모든 권한을 한 번에 부여합니다. PRIVILEGES 키워드는 옵션입니다.

GRANT ALL ON SCHEMA는 외부 스키마에 대한 CREATE 권한을 부여하지 않습니다.

Lake Formation에 사용되는 AWS Glue Data Catalog의 테이블에 ALL 권한을 부여할 수 있습니다. 이 경우 개별 권한(예: SELECT, ALTER 등)이 Data Catalog에 기록됩니다.

> **Note**
>
> Amazon Redshift는 RULE 및 TRIGGER 권한을 지원하지 않습니다. 자세한 내용은 지원되지 않는 PostgreSQL 기능 섹션을 참조하세요.

ASSUMEROLE

지정된 역할을 가진 사용자, 역할 또는 그룹에 COPY, UNLOAD, EXTERNAL FUNCTION 및 CREATE MODEL 명령을 실행하는 권한을 부여합니다. 사용자, 역할 또는 그룹은 지정된 명령을 실행할 때 해당 역할을 맡습니다. ASSUMEROLE 권한을 사용하기 시작하려면 ASSUMEROLE 권한 부여에 대한 사용 노트 섹션을 참조하세요.

ON [TABLE] table_name

테이블 또는 뷰에 대해 지정된 권한을 부여합니다. TABLE 키워드는 옵션입니다. 하나의 문에 여러 개의 테이블과 뷰를 나열할 수 있습니다.

ON ALL TABLES IN SCHEMA schema_name

참조되는 스키마에 있는 모든 테이블과 뷰에 대해 지정된 권한을 부여합니다.

(column_name [,...]) ON TABLE table_name

Amazon Redshift 테이블 또는 뷰의 지정된 열에서 사용자, 그룹 또는 PUBLIC에 지정된 권한을 부여합니다.

(column_list) ON EXTERNAL TABLE schema_name.table_name

참조된 스키마에서 Lake Formation 테이블의 지정된 열에 있는 IAM 역할에 지정된 권한을 부여합니다.

ON EXTERNAL TABLE schema_name.table_name

참조된 스키마에서 지정된 Lake Formation 테이블의 IAM 역할에 지정된 권한을 부여합니다.

ON EXTERNAL SCHEMA schema_name

참조된 스키마에 있는 IAM 역할에 지정된 권한을 부여합니다.

ON iam_role

IAM 역할에 지정된 권한을 부여합니다.

TO username

권한을 받는 사용자를 나타냅니다.

TO IAM_ROLE iam_role

권한을 받는 IAM 역할을 나타냅니다.

WITH GRANT OPTION

권한을 받은 사용자가 동일한 권한을 다른 사용자에게 부여할 수 있음을 나타냅니다. 그룹 또는 PUBLIC에는 WITH GRANT OPTION이 허용될 수 없습니다.

ROLE role_name

역할에 권한을 부여합니다.

GROUP group_name

사용자 그룹에 권한을 부여합니다. 쉼표로 구분된 목록으로 여러 사용자 그룹을 지정할 수 있습니다.

PUBLIC

이후에 생성되는 사용자를 포함하여, 모든 사용자에게 지정된 권한을 부여합니다. PUBLIC은 모든 사용자를 항상 포함하는 그룹을 나타냅니다. 개별 사용자의 권한은 PUBLIC에 부여된 권한, 사용자가 속한 그룹에 부여된 권한, 사용자에게 개별적으로 부여된 모든 권한의 합입니다.

Lake Formation EXTERNAL TABLE에 PUBLIC을 부여하면 Lake Formation 모든 사람 그룹에 권한이 부여됩니다.

CREATE

데이터베이스 객체에 따라, 사용자 또는 사용자 그룹에 다음 권한을 부여합니다.

- 데이터베이스의 경우 CREATE를 통해 사용자가 데이터베이스 내에 스키마를 생성하도록 허용합니다.
- 스키마의 경우 CREATE를 통해 사용자가 스키마 내에 객체를 생성하도록 허용합니다. 객체의 이름을 바꾸려면 사용자가 CREATE 권한이 있고 이름을 바꿀 객체를 소유해야 합니다.
- Amazon Redshift Spectrum 외부 스키마에는 CREATE ON SCHEMA를 사용할 수 없습니다. 외부 스키마에서 외부 테이블 사용 권한을 부여하려면 액세스 권한이 필요한 사용자에게 USAGE ON SCHEMA를 부여합니다. 외부 스키마의 소유자 또는 슈퍼유저만 외부 스키마에 외부 테이블을 생성할 수 있습니다. 외부 스키마의 소유권을 이전하려면 ALTER SCHEMA를 사용해 소유자를 변경합니다.

TEMPORARY | TEMP

지정된 데이터베이스에서 임시 테이블을 생성할 권한을 부여합니다. Amazon Redshift Spectrum 쿼리를 실행하려면 데이터베이스 사용자가 데이터베이스에서 임시 테이블을 생성할 수 있는 권한을 보유해야 합니다.

> **Note**
>
> 기본적으로, 사용자는 PUBLIC 그룹에서 자동 멤버십으로 임시 테이블을 생성할 권한이 허용됩니다. 사용자가 임시 테이블을 만들 수 있는 권한을 제거하려면 PUBLIC 그룹에서 TEMP 권한을 취소합니다. 그런 다음 특정 사용자 또는 사용자 그룹에 임시 테이블을 만들 수 있는 권한을 명시적으로 부여합니다.

ON DATABASE db_name

데이터베이스에 대해 지정된 권한을 부여합니다.

객체

사용자가 특정 스키마의 객체에 액세스할 수 있도록, 해당 스키마에 대해 USAGE 권한을 부여합니다. 이러한 객체에 대한 특정 작업은 로컬 Amazon Redshift 스키마에 대해 별도로 부여되어야 합니다(예: 테이블에 대한 SELECT 또는 UPDATE 권한). 기본적으로 모든 사용자는 PUBLIC 스키마에서 CREATE 및 USAGE 권한을 갖습니다.

ON SCHEMA 구문을 사용하여 외부 스키마에 USAGE를 부여하면 외부 스키마의 객체에 대해 별도로 작업을 부여할 필요가 없습니다. 해당 카탈로그 권한은 외부 스키마 객체에 대한 세분화된 권한을 제어합니다.

ON SCHEMA schema_name

스키마에 대해 지정된 권한을 부여합니다.

Amazon Redshift Spectrum 외부 스키마에서 GRANT CREATE ON SCHEMA와 GRANT ALL ON SCHEMA의 CREATE 권한은 사용할 수 없습니다. 외부 스키마에서 외부 테이블 사용 권한을 부여하려면 액세스 권한이 필요한 사용자에게 USAGE ON SCHEMA를 부여합니다. 외부 스키마의 소유자 또는 슈퍼유저만 외부 스키마에 외부 테이블을 생성할 수 있습니다. 외부 스키마의 소유권을 이전하려면 ALTER SCHEMA를 사용해 소유자를 변경합니다.

EXECUTE ON ALL FUNCTIONS IN SCHEMA schema_name

참조되는 스키마에 있는 모든 함수에 대해 지정된 권한을 부여합니다.

Amazon Redshift는 pg_catalog 네임스페이스에 정의된 pg_proc 기본 제공 항목에 대해 GRANT 또는 REVOKE 문을 지원하지 않습니다.

EXECUTE ON PROCEDURE procedure_name

특정 저장 프로시저에 대해 EXECUTE 권한을 부여합니다. 저장 프로시저 이름은 오버로딩될 수 있기 때문에 해당 프로시저에 대한 인수 목록을 포함해야 합니다. 자세한 내용은 저장 프로시저 명명 단원을 참조하십시오.

EXECUTE ON ALL PROCEDURES IN SCHEMA schema_name

참조되는 스키마에 있는 모든 저장 프로시저에 대해 지정된 권한을 부여합니다.

USAGE ON LANGUAGE language_name

언어에 대한 USAGE 권한을 부여합니다.

CREATE FUNCTION 명령을 실행하여 사용자 정의 함수(UDF)를 생성하려면 USAGE ON LANGUAGE 권한이 필요합니다. 자세한 내용은 UDF 보안 및 권한 단원을 참조하십시오.

CREATE PROCEDURE 명령을 실행하여 저장 프로시저를 생성하려면 USAGE ON LANGUAGE 권한이 필요합니다. 자세한 내용은 저장 프로시저의 보안 및 권한 단원을 참조하십시오.

Python UDF에는 `plpythonu`를 사용합니다. SQL UDF에는 `sql`을 사용합니다. 저장 프로시저에는 `plpgsql`을 사용합니다.

ON COPY JOB job_name

복사 작업에 대해 지정된 권한을 부여합니다.

FOR { ALL | COPY | UNLOAD | EXTERNAL FUNCTION | CREATE MODEL } [, ...]

권한이 부여된 SQL 명령을 지정합니다. ALL을 지정하여 COPY, UNLOAD, EXTERNAL FUNCTION 및 CREATE MODEL 문에 대한 권한을 부여할 수 있습니다. 이 절은 ASSUMEROLE 권한을 부여하는 경우에만 적용됩니다.

ALTER

사용자에게 데이터 공유에서 객체를 추가 또는 제거하거나 속성 PUBLICACCESSIBLE을 설정할 수 있는 ALTER 권한을 부여합니다. 자세한 내용은 ALTER DATASHARE 단원을 참조하십시오.

SHARE

데이터 소비자를 데이터 공유에 추가할 수 있는 권한을 사용자 및 사용자 그룹에 부여합니다. 이 권한은 특정 소비자(계정 또는 네임스페이스)가 클러스터에서 데이터 공유에 액세스할 수 있도록 하는 데 필요합니다. 소비자는 GUID(Globally Unique Identifier)로 지정된 클러스터 네임스페이스가 같거나 다른 AWS 계정일 수도 있고 다를 수도 있습니다.

ON DATASHARE datashare_name

참조된 데이터 공유에 대해 지정된 권한을 부여합니다. 소비자 액세스 제어 세분화에 대한 자세한 내용은 Amazon Redshift에서 다양한 수준의 데이터 공유 단원을 참조하세요.

객체

USAGE가 소비자 계정 또는 동일한 계정 내의 네임스페이스에 부여된 경우 특정 소비자 계정 또는 계정 내의 네임스페이스는 읽기 전용 방식으로 datashare 및 datashare 객체에 액세스할 수 있습니다.

TO NAMESPACE 'clusternamespace GUID'

소비자가 데이터 공유에 대해 지정된 권한을 받을 수 있는 동일한 계정의 네임스페이스를 나타냅니다. 네임스페이스는 128비트 영숫자 GUID를 사용합니다.

TO ACCOUNT 'accountnumber' [VIA DATA CATALOG]

소비자가 데이터 공유에 대해 지정된 권한을 받을 수 있는 다른 계정의 번호를 나타냅니다. VIA DATA CATALOG'를 지정하면 Lake Formation 계정에 데이터 공유 사용 권한을 부여하고 있음을 나타냅니다. 이 파라미터를 생략하면 클러스터를 소유한 계정에 사용 권한을 부여한다는 의미입니다.

ON DATABASE shared_database_name> [, ...]

지정된 데이터 공유에서 생성된 지정된 데이터베이스에 대해 지정된 사용 권한을 부여합니다.

ON SCHEMA shared_schema

지정된 데이터 공유에서 생성된 지정된 스키마에 대해 지정된 권한을 부여합니다.

FOR { SCHEMAS | TABLES | FUNCTIONS | PROCEDURES | LANGUAGES | COPY JOBS} IN

권한을 부여할 데이터베이스 객체를 지정합니다. IN 다음의 파라미터는 부여된 권한의 범위를 정의합니다.

CREATE MODEL

특정 사용자 또는 사용자 그룹에 CREATE MODEL 권한을 부여합니다.

ON MODEL model_name

특정 모델에 대해 EXECUTE 권한을 부여합니다.

ACCESS CATALOG

역할이 액세스할 수 있는 객체의 관련 메타데이터를 볼 수 있는 권한을 부여합니다.

{ role } [, ...]

다른 역할, 사용자 또는 PUBLIC에 부여할 역할입니다.

PUBLIC은 모든 사용자를 항상 포함하는 그룹을 나타냅니다. 개별 사용자의 권한은 PUBLIC에 부여된 권한, 사용자가 속한 그룹에 부여된 권한, 사용자에게 개별적으로 부여된 모든 권한의 합입니다.

TO { { user_name [WITH ADMIN OPTION] } | role }[, ...]

WITH ADMIN OPTION, 다른 역할 또는 PUBLIC을 사용하여 지정된 사용자에게 지정된 역할을 부여합니다.

WITH ADMIN OPTION 절에서 모든 피부여자에게 부여된 모든 역할에 대한 관리 옵션을 제공합니다.

EXPLAIN RLS TO ROLE rolename

EXPLAIN 계획에서 쿼리의 행 수준 보안 정책 필터를 설명할 수 있는 권한을 역할에 부여합니다.

IGNORE RLS TO ROLE rolename

쿼리에 대한 행 수준 보안 정책을 우회할 수 있는 권한을 역할에 부여합니다.

사용 노트

GRANT 사용 노트에 대한 자세한 내용은 the section called “사용 노트” 섹션을 참조하세요.

예시

GRANT 사용 방법의 예는 the section called “예시” 섹션을 참조하세요.

사용 노트

객체에 대한 권한을 허용하려면 다음 조건 중 하나를 충족해야 합니다.

- 객체 소유자입니다.
- 수퍼유저입니다.
- 그 객체와 권한에 대해 허용된 권한이 있습니다.

예를 들어, 다음 명령을 실행하면 사용자 HR은 직원 테이블에서 SELECT 명령을 수행할 뿐 아니라 다른 사용자에 대해 같은 권한을 허용하고 취소할 수 있습니다.

```
grant select on table employees to HR with grant option;
```

HR은 SELECT 이외의 작업 권한이나 직원 이외의 테이블에 대한 권한을 허용할 수 없습니다.

또 다른 예로, 다음 명령을 실행하면 사용자 HR은 직원 테이블에서 ALTER 명령을 수행할 뿐 아니라 다른 사용자에 대해 같은 권한을 허용하고 취소할 수 있습니다.

```
grant ALTER on table employees to HR with grant option;
```

HR은 ALTER 이외의 작업 권한이나 직원 이외의 테이블에 대한 권한을 허용할 수 없습니다.

뷰에 대해 허용된 권한을 갖는다고 해서 기본 테이블에 대한 권한을 갖는다는 의미는 아닙니다. 마찬가지로, 스키마에 대해 허용된 권한을 갖는다고 해서 스키마에 있는 테이블에 대한 권한을 갖는다는 의미는 아닙니다. 대신에 기본 테이블에 대한 액세스 권한을 명시적으로 부여합니다.

AWS Lake Formation 테이블에 권한을 부여하려면 해당 테이블의 외부 스키마와 연결된 IAM 역할에 외부 테이블에 권한을 부여할 권한이 있어야 합니다. 다음 예제에서는 연결된 IAM 역할 myGrantor가 있는 외부 스키마를 생성합니다. IAM 역할 myGrantor는 다른 사람에게 권한을 부여할 권한이 있습니다. GRANT 명령은 외부 스키마와 연결된 IAM 역할 myGrantor의 권한을 사용하여 IAM 역할 myGrantee에 권한을 부여합니다.

```
create external schema mySchema
from data catalog
database 'spectrum_db'
iam_role 'arn:aws:iam::123456789012:role/myGrantor'
create external database if not exists;
```

```
grant select
on external table mySchema.mytable
to iam_role 'arn:aws:iam::123456789012:role/myGrantee';
```

IAM 역할에 GRANT ALL 권한을 부여하면 관련된 Lake Formation 사용 Data Catalog에서 개별 권한이 부여됩니다. 예를 들어 다음 GRANT ALL은 부여된 개별 권한(SELECT, ALTER, DROP, DELETE, INSERT)을 Lake Formation 콘솔에 표시합니다.

```
grant all
on external table mySchema.mytable
to iam_role 'arn:aws:iam::123456789012:role/myGrantee';
```

수퍼유저는 객체 권한을 설정하는 GRANT 및 REVOKE 명령과는 무관하게 모든 객체에 액세스할 수 있습니다.

열 수준 액세스 제어 사용 시 주의 사항

다음 사용 노트는 Amazon Redshift 테이블 및 뷰에 대한 열 수준 권한에 적용됩니다. 이러한 주의 사항은 테이블에 대해 설명합니다. 예외를 명시적으로 언급하지 않는 한 뷰에 동일한 주의 사항이 적용됩니다.

- Amazon Redshift 테이블의 경우 열 수준에서 SELECT 및 UPDATE 권한만 부여할 수 있습니다. Amazon Redshift 뷰의 경우 열 수준에서 SELECT 권한만 부여할 수 있습니다.
- ALL 키워드는 테이블의 열 수준 GRANT 컨텍스트에서 사용될 때 결합된 SELECT 및 UPDATE 권한의 동의어입니다.

- 테이블의 모든 열에 대해 SELECT 권한이 없는 경우 SELECT * 작업을 수행하면 액세스 권한이 있는 열만 반환됩니다. 뷰를 사용할 때 SELECT * 작업은 뷰의 모든 열에 액세스하려고 시도합니다. 모든 열에 액세스할 수 있는 권한이 없는 경우 이러한 쿼리는 권한 거부 오류와 함께 실패합니다.
- SELECT *는 다음과 같은 경우 액세스 가능한 열로만 확장되지 않습니다.
  - SELECT *를 사용하여 액세스 가능한 열만 포함하는 일반 뷰를 생성할 수 없습니다.
  - SELECT *를 사용하여 액세스 가능한 열만 포함하는 구체화된 뷰를 생성할 수 없습니다.
- 테이블 또는 뷰에 대해 SELECT 또는 UPDATE 권한이 있고 열을 추가하는 경우 테이블 또는 뷰 및 모든 열에 대해 동일한 권한이 여전히 있습니다.
- 테이블의 소유자 또는 수퍼유저만 열 수준 권한을 부여할 수 있습니다.
- 열 수준 권한에는 WITH GRANT OPTION 절이 지원되지 않습니다.
- 테이블 수준과 열 수준 모두에서 동일한 권한을 보유할 수 없습니다. 예를 들어 `data_scientist` 사용자는 `employee` 테이블에 대한 SELECT 권한과 `employee.department` 열에 대한 SELECT 권한을 모두 가질 수 없습니다. 테이블 및 테이블 내의 열에 동일한 권한을 부여할 때 다음 결과를 고려하십시오.
  - 사용자에게 테이블에 대한 테이블 수준 권한이 있는 경우 열 수준에서 동일한 권한을 부여해도 아무런 효과가 없습니다.
  - 사용자에게 테이블에 대한 테이블 수준 권한이 있는 경우 테이블의 하나 이상의 열에 대해 동일한 권한을 취소하면 오류가 반환됩니다. 대신 테이블 수준에서 권한을 취소합니다.
  - 사용자에게 열 수준 권한이 있는 경우 테이블 수준에서 동일한 권한을 부여하면 오류가 반환됩니다.
  - 사용자에게 열 수준 권한이 있는 경우 테이블 수준에서 동일한 권한을 취소하면 테이블의 모든 열에 대한 열 및 테이블 권한이 모두 취소됩니다.
- Late-Binding 보기에 대한 열 수준 권한은 부여할 수 없습니다.
- 구체화된 뷰를 생성하려면 기본 테이블에 대해 테이블 수준 SELECT 권한이 있어야 합니다. 특정 열에 대한 열 수준 권한이 있더라도 해당 열에 대해서만 구체화된 보기를 생성할 수 없습니다. 그러나 일반 보기와 마찬가지로 구체화된 보기의 열에 SELECT 권한을 부여할 수 있습니다.
- 열 수준 권한의 권한 부여를 조회하려면 PG_ATTRIBUTE_INFO 보기를 사용합니다.

ASSUMEROLE 권한 부여에 대한 사용 노트

다음 사용 노트는 Amazon Redshift에서 ASSUMEROLE 권한을 부여하는 데 적용됩니다.

ASSUMEROLE 권한을 사용하여 COPY, UNLOAD, EXTERNAL FUNCTION 또는 CREATE MODEL과 같은 명령에 대한 데이터베이스 사용자, 역할 또는 그룹의 IAM 역할 액세스 권한을 제어합니다. IAM

역할에 대해 사용자, 역할 또는 그룹에 ASSUMEROLE 권한을 부여한 후 해당 사용자, 역할 또는 그룹은 명령을 실행할 때 해당 역할을 맡을 수 있습니다. ASSUMEROLE 권한을 사용하면 필요에 따라 적절한 명령에 대한 액세스 권한을 부여할 수 있습니다.

데이터베이스 슈퍼유저만 사용자, 역할 및 그룹에 대한 ASSUMEROLE 권한을 부여하거나 취소할 수 있습니다. 슈퍼유저는 항상 ASSUMEROLE 권한을 유지합니다.

사용자, 역할 및 그룹에 ASSUMEROLE 권한을 사용하도록 설정하기 위해 슈퍼유저는 다음 두 가지 작업을 수행합니다.

- 클러스터에서 다음 문을 한 번 실행합니다.

```
revoke assumerole on all from public for all;
```

- 적절한 명령에 대해 사용자, 역할 및 그룹에 ASSUMEROLE 권한을 부여합니다.

ASSUMEROLE 권한을 부여할 때 ON 절에서 역할 체인을 지정할 수 있습니다. 쉼표를 사용하여 역할 체인에서 역할을 구분합니다(예: `Role1,Role2,Role3`). ASSUMEROLE 권한을 부여할 때 역할 체인을 지정한 경우 ASSUMEROLE 권한으로 부여된 작업을 수행할 때 역할 체인을 지정해야 합니다. ASSUMEROLE 권한으로 부여된 작업을 수행할 때 역할 체인 내에서 개별 역할을 지정할 수 없습니다. 예를 들어 사용자, 역할 또는 그룹에 역할 체인 `Role1,Role2,Role3`이 부여된 경우 작업을 수행하도록 `Role1`만 지정할 수 없습니다.

사용자가 COPY, UNLOAD, EXTERNAL FUNCTION 또는 CREATE MODEL 작업을 수행하려고 하고 ASSUMEROLE 권한이 부여되지 않은 경우 다음과 유사한 메시지가 나타납니다.

```
ERROR:  User awsuser does not have ASSUMEROLE permission on IAM role
 "arn:aws:iam::123456789012:role/RoleA" for COPY
```

ASSUMEROLE 권한을 통해 IAM 역할 및 명령에 대한 액세스 권한이 부여된 사용자를 나열하려면 HAS_ASSUMEROLE_PRIVILEGE 섹션을 참조하세요. 지정한 사용자에게 부여된 IAM 역할 및 명령 권한을 나열하려면 PG_GET_IAM_ROLE_BY_USER 섹션을 참조하세요. 지정한 IAM 역할에 대한 액세스 권한이 부여된 사용자, 역할 및 그룹을 나열하려면 PG_GET_GRANTEE_BY_IAM_ROLE 섹션을 참조하세요.

기계 학습 권한 부여에 대한 사용 노트

ML 함수와 관련된 권한을 직접 부여하거나 취소할 수 없습니다. ML 함수는 ML 모델에 속하며 권한은 모델을 통해 제어됩니다. 대신 ML 모델과 관련된 권한을 부여할 수 있습니다. 다음 예제는 모든 사용자

에게 customer_churn 모델과 연결된 ML 함수를 실행할 수 있는 권한을 부여하는 방법을 보여줍니다.

```
GRANT EXECUTE ON MODEL customer_churn TO PUBLIC;
```

ML 모델 customer_churn에 대한 모든 권한을 사용자에게 부여할 수도 있습니다.

```
GRANT ALL on MODEL customer_churn TO ml_user;
```

스키마에 ML 함수가 있는 경우 해당 ML 함수가 이미 GRANT EXECUTE ON MODEL을 통해 EXECUTE 권한을 부여받았더라도 ML 함수와 관련된 EXECUTE 권한 부여는 실패합니다. CREATE MODEL 명령을 사용할 때는 별도의 스키마를 사용하여 ML 함수를 별도의 스키마에 따로 보관하는 것이 좋습니다. 다음 예제에서는 이 작업을 수행하는 방법을 보여 줍니다.

```
CREATE MODEL ml_schema.customer_churn
FROM customer_data
TARGET churn
FUNCTION ml_schema.customer_churn_prediction
IAM_ROLE default
SETTINGS (
 S3_BUCKET 'amzn-s3-demo-bucket'
);
```

예시

다음 예에서는 사용자 fred에게 SALES 테이블에 대한 SELECT 권한을 허용합니다.

```
grant select on table sales to fred;
```

다음 예에서는 사용자 fred에게 QA_TICKIT 스키마에 있는 모든 테이블에 대한 SELECT 권한을 허용합니다.

```
grant select on all tables in schema qa_tickit to fred;
```

다음 예에서는 사용자 그룹 QA_USERS에게 스키마 QA_TICKIT에 대한 모든 스키마 권한을 허용합니다. 스키마 권한은 CREATE 및 USAGE입니다. USAGE는 스키마에 있는 객체에 대한 액세스 권한을 사용자에게 허용하지만, 해당 객체에 대한 INSERT 또는 SELECT 같은 권한은 허용하지 않습니다. 각 객체에 대한 권한을 개별적으로 부여합니다.

```
create group qa_users;
grant all on schema qa_tickit to group qa_users;
```

다음 예에서는 그룹 QA_USERS의 모든 사용자에게 QA_TICKIT 스키마에 있는 SALES 테이블에 대한 모든 권한을 허용합니다.

```
grant all on table qa_tickit.sales to group qa_users;
```

다음 예제에서는 그룹 QA_USERS 및 RO_USERS의 모든 사용자에게 QA_TICKIT 스키마에 있는 SALES 테이블에 대한 모든 권한을 부여합니다.

```
grant all on table qa_tickit.sales to group qa_users, group ro_users;
```

다음 예에서는 그룹 QA_USERS의 모든 사용자에게 QA_TICKIT 스키마에 있는 SALES 테이블에 대한 DROP 권한을 허용합니다.

```
grant drop on table qa_tickit.sales to group qa_users;>
```

다음 명령 시퀀스는 어떻게 스키마에 대한 액세스가 스키마에 있는 테이블에 대한 권한을 허용하지 않는지 보여줍니다.

```
create user schema_user in group qa_users password 'Abcd1234';
create schema qa_tickit;
create table qa_tickit.test (col1 int);
grant all on schema qa_tickit to schema_user;

set session authorization schema_user;
select current_user;


current_user
--------------
schema_user
(1 row)


select count(*) from qa_tickit.test;
```

```
ERROR: permission denied for relation test [SQL State=42501]


set session authorization dw_user;
grant select on table qa_tickit.test to schema_user;
set session authorization schema_user;
select count(*) from qa_tickit.test;


count
-------
0
(1 row)
```

다음 명령 시퀀스는 어떻게 뷰에 대한 액세스가 기본 테이블에 대한 액세스를 의미하는 것은 아닌지 보여줍니다. VIEW_USER라는 사용자에게는 VIEW_DATE에 대한 모든 권한이 허용되었지만, 이 사용자는 DATE 테이블에서 선택할 수 없습니다.

```
create user view_user password 'Abcd1234';
create view view_date as select * from date;
grant all on view_date to view_user;
set session authorization view_user;
select current_user;


current_user
--------------
view_user
(1 row)


select count(*) from view_date;


count
-------
365
(1 row)


select count(*) from date;
```

```
ERROR:  permission denied for relation date
```

다음 예에서는 user1 사용자에게 cust_profile 테이블의 cust_name 및 cust_phone 열에 대한 SELECT 권한을 부여합니다.

```
grant select(cust_name, cust_phone) on cust_profile to user1;
```

다음 예에서는 sales_group 그룹에 cust_profile 테이블의 cust_name 및 cust_phone 열에 대한 SELECT 권한과 cust_contact_preference 열에 대한 UPDATE 권한을 부여합니다.

```
grant select(cust_name, cust_phone), update(cust_contact_preference) on cust_profile to
 group sales_group;
```

다음 예에서는 ALL 키워드를 사용하여 cust_profile 테이블의 세 열에 대한 SELECT 및 UPDATE 권한을 모두 sales_admin 그룹에 부여하는 것을 보여 줍니다.

```
grant ALL(cust_name, cust_phone,cust_contact_preference) on cust_profile to group
 sales_admin;
```

다음 예에서는 cust_profile_vw 뷰의 cust_name 열에 대한 SELECT 권한을 user2 사용자에게 부여합니다.

```
grant select(cust_name) on cust_profile_vw to user2;
```

데이터 공유에 대한 액세스 권한 부여의 예

다음 예에서는 datashare에서 생성된 특정 데이터베이스 또는 스키마에 대한 GRANT datashare 사용 권한을 보여줍니다.

다음 예시에서는 생산자 측 관리자가 salesshare 데이터 공유에 대한 USAGE 권한을 지정된 네임스페이스에 부여합니다.

```
GRANT USAGE ON DATASHARE salesshare TO NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

다음 예시에서는 소비자 측 관리자가 sales_db에 대한 USAGE 권한을 Bob에게 부여합니다.

```
GRANT USAGE ON DATABASE sales_db TO Bob;
```

다음 예시에서는 소비자 측 관리자가 sales_schema 스키마에 대한 GRANT USAGE 권한을 Analyst_role 역할에 부여합니다. sales_schema는 sales_db를 가리키는 외부 스키마입니다.

```
GRANT USAGE ON SCHEMA sales_schema TO ROLE Analyst_role;
```

이때 Bob 및 Analyst_role은 sales_schema 및 sales_db에 있는 모든 데이터베이스 객체에 액세스할 수 있습니다.

다음 예시에서는 공유 데이터베이스의 객체에 추가 객체 수준 권한을 부여하는 방법을 보여줍니다. 이러한 추가 권한은 공유 데이터베이스를 만드는 데 사용된 CREATE DATABASE 명령에서 WITH PERMISSIONS 절을 사용한 경우에만 필요합니다. CREATE DATABASE 명령에서 WITH PERMISSIONS를 사용하지 않은 경우 공유 데이터베이스에 대한 USAGE를 부여하면 해당 데이터베이스의 모든 객체에 대한 전체 액세스 권한이 부여됩니다.

```
GRANT SELECT ON sales_db.sales_schema.tickit_sales_redshift to Bob;
```

범위가 지정된 권한 부여의 예

다음 예시에서는 Sales_db 데이터베이스의 모든 현재 및 미래 스키마에 대한 사용 권한을 Sales 역할에 부여합니다.

```
GRANT USAGE FOR SCHEMAS IN DATABASE Sales_db TO ROLE Sales;
```

다음 예시에서는 Sales_db 데이터베이스의 모든 현재 및 미래 테이블에 대한 SELECT 권한을 alice라는 사용자에게 부여하고, Sales_db에 있는 테이블에 대한 범위가 지정된 권한을 다른 사용자에게 부여할 수 있는 권한도 alice에게 부여합니다.

```
GRANT SELECT FOR TABLES IN DATABASE Sales_db TO alice WITH GRANT OPTION;
```

다음 예시에서는 Sales_schema 스키마의 함수에 대한 EXECUTE 권한을 bob이라는 사용자에게 부여합니다.

```
GRANT EXECUTE FOR FUNCTIONS IN SCHEMA Sales_schema TO bob;
```

다음 예시에서는 ShareDb 데이터베이스의 ShareSchema 스키마에 있는 모든 테이블에 대한 모든 권한을 Sales 역할에 부여합니다. 스키마를 지정할 때 두 부분으로 구성된 database.schema 형식을 사용하여 스키마의 데이터베이스를 지정할 수 있습니다.

```
GRANT ALL FOR TABLES IN SCHEMA ShareDb.ShareSchema TO ROLE Sales;
```

다음 예시는 이전 예시와 동일합니다. 두 부분으로 구성된 형식을 사용하는 대신 DATABASE 키워드를 사용하여 데이터베이스를 지정할 수 있습니다.

```
GRANT ALL FOR TABLES IN SCHEMA ShareSchema DATABASE ShareDb TO ROLE Sales;
```

ASSUMEROLE 권한 부여의 예

다음은 ASSUMEROLE 권한을 부여하는 예입니다.

다음 예에서는 슈퍼 사용자가 사용자 및 그룹에 대해 ASSUMEROLE 권한을 사용할 수 있도록 클러스터에서 한 번 실행하는 REVOKE 문을 보여줍니다. 그런 다음 슈퍼 사용자는 적절한 명령에 대해 사용자 및 그룹에 ASSUMEROLE 권한을 부여합니다. 사용자 및 그룹에 대한 ASSUMEROLE 권한 사용 설정에 대한 내용은 ASSUMEROLE 권한 부여에 대한 사용 노트 섹션을 참조하세요.

```
revoke assumerole on all from public for all;
```

다음 예에서는 COPY 작업을 수행할 IAM 역할 Redshift-S3-Read에 대한 ASSUMEROLE 권한을 사용자 reg_user1에게 부여합니다.

```
grant assumerole on 'arn:aws:iam::123456789012:role/Redshift-S3-Read'
to reg_user1 for copy;
```

다음 예에서는 UNLOAD 작업을 수행하기 위해 IAM 역할 체인 RoleA, RoleB에 대한 ASSUMEROLE 권한을 사용자 reg_user1에게 부여합니다.

```
grant assumerole
on 'arn:aws:iam::123456789012:role/RoleA,arn:aws:iam::210987654321:role/RoleB'
to reg_user1
for unload;
```

다음은 IAM 역할 체인 RoleA, RoleB를 사용하는 UNLOAD 명령의 예입니다.

```
unload ('select * from venue limit 10')
to 's3://companyb/redshift/venue_pipe_'
iam_role 'arn:aws:iam::123456789012:role/RoleA,arn:aws:iam::210987654321:role/RoleB';
```

다음 예에서는 외부 기능을 생성할 IAM 역할 Redshift-Exfunc에 대한 ASSUMEROLE 권한을 사용자 reg_user1에게 부여합니다.

```
grant assumerole on 'arn:aws:iam::123456789012:role/Redshift-Exfunc'
to reg_user1 for external function;
```

다음 예에서는 기계 학습 모델을 생성할 IAM 역할 Redshift-model에 대한 ASSUMEROLE 권한을 사용자 reg_user1에게 부여합니다.

```
grant assumerole on 'arn:aws:iam::123456789012:role/Redshift-ML'
to reg_user1 for create model;
```

ROLE 권한 부여의 예

다음은 sample_role1을 user1에 부여하는 예입니다.

```
CREATE ROLE sample_role1;
GRANT ROLE sample_role1 TO user1;
```

다음 예에서는 WITH ADMIN OPTION을 사용하여 sample_role1을 user1에 부여하고, user1에 대한 현재 세션을 설정하며, user1은 sample_role1을 user2에 부여합니다.

```
GRANT ROLE sample_role1 TO user1 WITH ADMIN OPTION;
SET SESSION AUTHORIZATION user1;
GRANT ROLE sample_role1 TO user2;
```

다음은 sample_role1을 sample_role2에 부여하는 예입니다.

```
GRANT ROLE sample_role1 TO ROLE sample_role2;
```

다음은 sample_role2를 sample_role3 및 sample_role4에 부여하는 예입니다. 그런 다음 sample_role3을 sample_role1에 부여하려고 시도합니다.

```
GRANT ROLE sample_role2 TO ROLE sample_role3;
GRANT ROLE sample_role3 TO ROLE sample_role2;
ERROR: cannot grant this role, a circular dependency was detected between these roles
```

다음 예에서는 CREATE USER 시스템 권한을 sample_role1에 부여합니다.

```
GRANT CREATE USER TO ROLE sample_role1;
```

다음은 sys:dba 시스템 정의 역할을 user1에 부여하는 예입니다.

```
GRANT ROLE sys:dba TO user1;
```

다음 예에서는 순환 종속성에서 sample_role3을 sample_role2에 부여하려고 시도합니다.

```
CREATE ROLE sample_role3;
GRANT ROLE sample_role2 TO ROLE sample_role3;
GRANT ROLE sample_role3 TO ROLE sample_role2; -- fail
ERROR:  cannot grant this role, a circular dependency was detected between these roles
```

INSERT

주제

테이블에 새 행을 삽입합니다. VALUES 구문을 포함한 단일 행, VALUES 구문을 포함한 다중 행, 또는 쿼리(INSERT INTO...SELECT)의 결과에 의해 정의되는 하나 이상의 행을 삽입할 수 있습니다.

> **Note**
>
> 대량의 데이터를 로드하려면 COPY 명령을 사용하는 것이 가장 좋습니다. 테이블을 채우는 데 개별적인 INSERT문을 사용하는 방식은 엄청나게 느릴 수 있습니다. 대안으로 데이터가 이미 다른 Amazon Redshift 데이터베이스 테이블에 존재하는 경우 성능을 개선하려면 INSERT INTO SELECT 또는 CREATE TABLE AS를 사용합니다. COPY 명령을 사용한 테이블 로드에 대한 자세한 내용은 Amazon Redshift에서 데이터 로드 섹션을 참조하세요.

> **Note**
>
> 단일 SQL 문의 최대 크기는 16MB입니다.

구문

```
INSERT INTO table_name [ ( column [, ...] ) ]
{DEFAULT VALUES |
VALUES ( { expression | DEFAULT } [, ...] )
[, ( { expression | DEFAULT } [, ...] )
[, ...] ] |
query }
```

파라미터

table_name

임시 또는 영구 테이블입니다. 테이블의 소유자 또는 테이블에 대한 INSERT 권한을 가진 사용자만이 행을 삽입할 수 있습니다. query 절을 사용하여 행을 삽입할 경우 쿼리에 이름이 지정된 테이블에 대한 SELECT 권한이 있어야 합니다.

> **Note**
>
> INSERT(외부 테이블)를 사용하여 외부 카탈로그의 기존 테이블에 SELECT 쿼리의 결과를 삽입합니다. 자세한 내용은 INSERT(외부 테이블) 단원을 참조하십시오.

column

테이블의 한 개 이상의 열에 값을 삽입할 수 있습니다. 임의의 순서대로 대상 열 이름을 나열할 수 있습니다. 열 목록을 지정하지 않은 경우 삽입되는 값은 CREATE TABLE 문에서 선언된 순서대로 테이블 열에 상응해야 합니다. 삽입되는 값의 개수가 테이블에 있는 열의 개수보다 작을 경우 처음 n개의 열이 로드됩니다.

INSERT 문에 (암시적으로 또는 명시적으로) 나열되지 않은 임의의 열로 선언된 기본값 또는 null 값이 로드됩니다.

DEFAULT VALUES

테이블 생성 시 테이블의 열에 기본값이 할당된 경우 이런 키워드를 사용하여 전적으로 기본값으로 구성되는 행을 삽입합니다. 어떤 열에도 기본값이 없는 경우에는 이러한 열에 null이 삽입됩니다. NOT NULL로 선언된 열이 있는 경우에는 INSERT 문이 오류를 반환합니다.

VALUES

각각 하나 이상의 값으로 구성된 하나 이상의 행을 삽입하려면 이 키워드를 사용하십시오. 각 행에 대한 VALUES 목록은 열 목록과 일치해야 합니다. 여러 행을 삽입하려면 표현식의 각 목록 사이에 쉼표를 구분 기호로 사용하십시오. VALUES 키워드를 반복하지 마십시오. 다중 행 INSERT 문에 대한 모든 VALUES 목록은 같은 수의 값을 포함해야 합니다.

expression

단일 값 또는 단일 값으로 계산되는 표현식입니다. 각각의 값은 자신이 삽입되는 열의 데이터 형식과 호환 가능해야 합니다. 가능한 경우 데이터 형식이 열의 선언된 데이터 형식과 일치하지 않는 값은 자동으로 호환 가능한 데이터 형식으로 변환됩니다. 예:

- 10진수 값 `1.1`은 INT 열에 1로 삽입됩니다.
- 10진수 값 `100.8976`은 DEC(5,2) 열에 `100.90`으로 삽입됩니다.

표현식에 형식 캐스팅 구문을 포함시킴으로써 값을 호환 가능한 데이터 형식으로 명시적으로 변환할 수 있습니다. 테이블 T1에서 열 COL1이 CHAR(3) 열인 경우를 예로 들면 다음과 같습니다.

```
insert into t1(col1) values('Incomplete'::char(3));
```

이 문은 값 `Inc`를 열에 삽입합니다.

단일 행 INSERT VALUES 문의 경우 스칼라 하위 쿼리를 표현식으로 사용할 수 있습니다. 하위 쿼리의 결과는 적절한 열에 삽입됩니다.

> **Note**
>
> 하위 쿼리는 다중 행 INSERT VALUES 문을 위한 표현식으로 지원되지 않습니다.

DEFAULT

테이블 생성 시 정의된 대로, 열의 기본값을 삽입하려면 이 키워드를 사용하십시오. 열에 대한 기본값이 존재하지 않으면 null이 삽입됩니다. NOT NULL 제약 조건이 있는 열이 CREATE TABLE 문에서 자신에게 할당된 명시적 기본값이 없는 경우에는 이 열에 기본값을 삽입할 수 없습니다.

query

임의의 쿼리를 정의하여 테이블에 하나 이상의 행을 삽입합니다. 쿼리가 생성하는 모든 행이 테이블에 삽입됩니다. 쿼리는 테이블에 있는 열과 호환되는 열 목록을 반환해야 하지만, 열 이름이 일치할 필요는 없습니다.

사용 노트

> **Note**
>
> 대량의 데이터를 로드하려면 COPY 명령을 사용하는 것이 가장 좋습니다. 테이블을 채우는 데 개별적인 INSERT문을 사용하는 방식은 엄청나게 느릴 수 있습니다. 대안으로 데이터가 이미 다른 Amazon Redshift 데이터베이스 테이블에 존재하는 경우 성능을 개선하려면 INSERT INTO SELECT 또는 CREATE TABLE AS를 사용합니다. COPY 명령을 사용한 테이블 로드에 대한 자세한 내용은 Amazon Redshift에서 데이터 로드 섹션을 참조하세요.

삽입되는 값의 데이터 형식은 CREATE TABLE 정의에 의해 지정된 데이터 형식과 일치해야 합니다.

테이블에 많은 수의 행을 새로 삽입한 후.

- 테이블을 완전히 비워 스토리지 공간을 회수하고 행을 다시 정렬합니다.
- 테이블을 분석하여 쿼리 플래너에 대한 통계를 업데이트합니다.

값이 DECIMAL 열에 삽입되고 지정된 규모를 초과할 때, 로드된 값은 적당히 반올림됩니다. 예를 들어, `20.259`라는 값이 DECIMAL(8,2) 열에 삽입되는 경우 저장되는 값은 `20.26`입니다.

GENERATED BY DEFAULT AS IDENTITY 열에 추가할 수 있습니다. GENERATED BY DEFAULT AS IDENTITY로 정의된 열을 직접 입력하는 값으로 업데이트할 수 있습니다. 자세한 내용은 GENERATED BY DEFAULT AS IDENTITY 단원을 참조하십시오.

INSERT 예

TICKIT 데이터베이스의 CATEGORY 테이블은 다음 행을 포함합니다.

```
 catid | catgroup |  catname  |                 catdesc
-------+----------+-----------+--------------------------------------------
     1 | Sports   | MLB       | Major League Baseball
     2 | Sports   | NHL       | National Hockey League
     3 | Sports   | NFL       | National Football League
     4 | Sports   | NBA       | National Basketball Association
     5 | Sports   | MLS       | Major League Soccer
     6 | Shows    | Musicals  | Musical theatre
     7 | Shows    | Plays     | All non-musical theatre
     8 | Shows    | Opera     | All opera and light opera
```

```
     9 | Concerts | Pop       | All rock and pop music concerts
    10 | Concerts | Jazz      | All jazz singers and bands
    11 | Concerts | Classical | All symphony, concerto, and choir concerts
(11 rows)
```

CATEGORY 테이블에 대한 유사한 스키마를 가진 CATEGORY_STAGE 테이블을 생성하되, 열에 대한 기본값을 정의합니다.

```
create table category_stage
(catid smallint default 0,
catgroup varchar(10) default 'General',
catname varchar(10) default 'General',
catdesc varchar(50) default 'General');
```

다음 INSERT 문은 CATEGORY 테이블에서 모든 행을 선택하여 CATEGORY_STAGE 테이블로 삽입합니다.

```
insert into category_stage
(select * from category);
```

쿼리 주변의 괄호는 선택 사항입니다.

이 명령은 각 열에 대해 순서대로 값이 지정된 CATEGORY_STAGE 테이블에 새 행을 삽입합니다.

```
insert into category_stage values
(12, 'Concerts', 'Comedy', 'All stand-up comedy performances');
```

특정 값과 기본값을 결합하는 새로운 행을 삽입할 수도 있습니다.

```
insert into category_stage values
(13, 'Concerts', 'Other', default);
```

다음 쿼리를 실행하여 삽입된 행을 반환합니다.

```
select * from category_stage
where catid in(12,13) order by 1;

 catid | catgroup | catname |             catdesc
-------+----------+---------+----------------------------------
```

```
    12 | Concerts | Comedy  | All stand-up comedy performances
    13 | Concerts | Other   | General
(2 rows)
```

다음 예에서는 몇 가지 다중 행 INSERT VALUES 문을 보여줍니다. 첫 번째 예에서는 두 행에 대한 특정 CATID 값과 다른 열의 기본값을 두 행에 모두 삽입합니다.

```
insert into category_stage values
(14, default, default, default),
(15, default, default, default);

select * from category_stage where catid in(14,15) order by 1;
 catid | catgroup | catname | catdesc
-------+----------+---------+---------
    14 | General  | General | General
    15 | General  | General | General
(2 rows)
```

그 다음 예에서는 특정한 값과 기본값이 다양하게 조합된 세 개의 행을 삽입합니다.

```
insert into category_stage values
(default, default, default, default),
(20, default, 'Country', default),
(21, 'Concerts', 'Rock', default);

select * from category_stage where catid in(0,20,21) order by 1;
 catid | catgroup | catname | catdesc
-------+----------+---------+---------
     0 | General  | General | General
    20 | General  | Country | General
    21 | Concerts | Rock    | General
(3 rows)
```

이 예에서 첫 번째 VALUES 집합은 단일 행 INSERT 문에 대한 DEFAULT VALUES를 지정하는 것과 똑같은 결과를 만들어냅니다.

다음 예에서는 테이블에 IDENTITY 열이 있을 때 INSERT의 동작을 보여줍니다. 먼저 CATEGORY 테이블의 새 버전을 만든 다음, CATEGORY에서 테이블에 행을 삽입합니다.

```
create table category_ident
(catid int identity not null,
```

```
catgroup varchar(10) default 'General',
catname varchar(10) default 'General',
catdesc varchar(50) default 'General');


insert into category_ident(catgroup,catname,catdesc)
select catgroup,catname,catdesc from category;
```

참고로, 특정한 정수 값을 CATID IDENTITY 열에 삽입할 수 없습니다. IDENTITY 열 값은 자동으로 생성됩니다.

다음 예에서는 다중 행 INSERT VALUES 문에 하위 쿼리를 표현식으로 사용할 수 없음을 보여줍니다.

```
insert into category(catid) values
((select max(catid)+1 from category)),
((select max(catid)+2 from category));

ERROR: can't use subqueries in multi-row VALUES
```

다음 예에서는 WITH SELECT 절을 사용하여 venue 테이블의 데이터로 채워진 임시 테이블에 삽입하는 방법을 보여줍니다. venue 테이블에 대한 자세한 내용은 샘플 데이터베이스 단원을 참조하세요.

먼저 임시 테이블 #venuetemp를 생성합니다.

```
CREATE TABLE #venuetemp AS SELECT * FROM venue;
```

#venuetemp 테이블의 행을 나열합니다.

```
SELECT * FROM #venuetemp ORDER BY venueid;

venueid | venuename                | venuecity  | venuestate| venueseats
--------+--------------------------+------------+-----------+------------
1       Toyota Park                Bridgeview   IL          0
2       Columbus Crew Stadium      Columbus     OH          0
3       RFK Stadium                Washington   DC          0
4       CommunityAmerica Ballpark  Kansas City  KS          0
5       Gillette Stadium           Foxborough   MA          68756
...
```

WITH SELECT 절을 사용하여 #venuetemp 테이블에 10개의 중복 행을 삽입합니다.

```
INSERT INTO #venuetemp (WITH venuecopy AS (SELECT * FROM venue) SELECT * FROM venuecopy
 ORDER BY 1 LIMIT 10);
```

#venuetemp 테이블의 행을 나열합니다.

```
SELECT * FROM #venuetemp ORDER BY venueid;

venueid | venuename                | venuecity   | venuestate| venueseats
--------+--------------------------+-------------+-----------+------------
1       | Toyota Park               Bridgeview    IL          0
1       | Toyota Park               Bridgeview    IL          0
2       | Columbus Crew Stadium     Columbus      OH          0
2       | Columbus Crew Stadium     Columbus      OH          0
3       | RFK Stadium               Washington    DC          0
3       | RFK Stadium               Washington    DC          0
4       | CommunityAmerica Ballpark  Kansas City  KS          0
4       | CommunityAmerica Ballpark  Kansas City  KS          0
5       | Gillette Stadium          Foxborough    MA          68756
5       | Gillette Stadium          Foxborough    MA          68756
...
```

INSERT(외부 테이블)

AWS Glue, AWS Lake Formation 또는 Apache Hive 메타스토어와 같은 외부 카탈로그의 기존 외부 테이블에 SELECT 쿼리의 결과를 삽입합니다. 외부 카탈로그 및 Amazon S3와 상호 작용하려면 CREATE EXTERNAL SCHEMA 명령에 사용된 것과 동일한 AWS Identity and Access Management(IAM) 역할을 사용합니다.

분할되지 않은 테이블의 경우 INSERT(외부 테이블) 명령은 지정된 테이블 속성과 파일 형식에 따라 테이블에 정의된 Amazon S3 위치에 데이터를 작성합니다.

분할된 테이블의 경우 INSERT(외부 테이블)는 테이블에 지정된 파티션 키에 따라 Amazon S3 위치에 데이터를 작성합니다. 또한 INSERT 작업이 완료된 후 외부 카탈로그에 새 파티션을 자동으로 등록합니다.

트랜잭션 블록(BEGIN ... END) 내에서는 INSERT(외부 테이블)를 실행할 수 없습니다. 버전 관리에 대한 자세한 내용은 직렬화 가능 격리 단원을 참조하십시오.

구문

```
INSERT INTO external_schema.table_name
```

```
{ select_statement }
```

파라미터

external_schema.table_name

기존 외부 스키마 및 삽입할 대상 외부 테이블의 이름입니다.

select_statement

쿼리를 정의하여 외부 테이블에 하나 이상의 행을 삽입하는 문입니다. 쿼리가 생성하는 모든 행은 테이블 정의에 따라 텍스트 또는 Parquet 형식으로 Amazon S3에 작성됩니다. 쿼리는 외부 테이블의 열 데이터 형식과 호환되는 열 목록을 반환해야 합니다. 그러나 열 이름은 일치하지 않아도 됩니다.

사용 노트

SELECT 쿼리의 열 수는 데이터 열 및 파티션 열의 합계와 같아야 합니다. 각 데이터 열의 위치 및 데이터 형식은 외부 테이블의 위치 및 데이터 형식과 일치해야 합니다. 파티션 열의 위치는 CREATE EXTERNAL TABLE 명령에 정의된 것과 동일한 순서로 SELECT 쿼리의 끝에 있어야 합니다. 열 이름은 일치하지 않아도 됩니다.

경우에 따라 AWS Glue 데이터 카탈로그 또는 Hive 메타스토어에 대해 INSERT(외부 테이블) 명령을 실행할 수 있습니다. AWS Glue의 경우 외부 스키마를 생성하는 데 사용되는 IAM 역할은 Amazon S3 및 AWS Glue에 대한 읽기 및 쓰기 권한이 모두 있어야 합니다. AWS Lake Formation 카탈로그를 사용하는 경우 이 IAM 역할이 새 Lake Formation 테이블의 소유자가 됩니다. 이 IAM 역할에는 최소한 다음 권한이 있어야 합니다.

- 외부 테이블에 대한 SELECT, INSERT, UPDATE 권한
- 외부 테이블의 Amazon S3 경로에 대한 데이터 위치 권한

파일 이름이 고유한지 확인하기 위해 Amazon Redshift에서는 기본적으로 Amazon S3에 업로드된 각 파일의 이름에 다음 형식을 사용합니다.

*<date>*_*<time>*_*<microseconds>*_*<query_id>*_*<slice-number>*_part_*<part-number>*.*<format>*.

예를 들면, 20200303_004509_810669_1007_0001_part_00.parquet입니다.

INSERT(외부 테이블) 명령을 실행할 때 다음 사항을 고려하십시오.

- PARQUET 또는 TEXTFILE이 아닌 다른 형식의 외부 테이블은 지원되지 않습니다.
- 이 명령은 'write.parallel', 'write.maxfilesize.mb', 'compression_type', 'serialization.null.format' 같은 기존 테이블 속성을 지원합니다. 이러한 값을 업데이트하려면 ALTER TABLE SET TABLE PROPERTIES 명령을 실행합니다.
- 'numRows' 테이블 속성은 자동으로 INSERT 작업의 끝으로 업데이트됩니다. CREATE EXTERNAL TABLE AS 작업에 의해 생성되지 않은 경우 테이블 속성을 정의하거나 테이블에 추가해야 합니다.
- LIMIT 절은 외부 SELECT 쿼리에서 지원되지 않습니다. 대신 중첩 LIMIT 절을 사용합니다.
- STL_UNLOAD_LOG 테이블을 사용하여 각 INSERT(외부 테이블) 작업에서 Amazon S3에 작성된 파일을 추적할 수 있습니다.
- Amazon Redshift는 INSERT(외부 테이블)에 대해 Amazon S3 Standard 암호화만 지원합니다.

INSERT(외부 테이블) 예

다음 예에서는 SELECT 문의 결과를 외부 테이블에 삽입합니다.

```
INSERT INTO spectrum.lineitem
SELECT * FROM local_lineitem;
```

다음 예에서는 정적 분할을 사용하여 분할된 외부 테이블에 SELECT 문의 결과를 삽입합니다. 파티션 열은 SELECT 문에서 하드 코딩됩니다. 파티션 열은 쿼리의 끝에 있어야 합니다.

```
INSERT INTO spectrum.customer
SELECT name, age, gender, 'May', 28 FROM local_customer;
```

다음 예에서는 동적 분할을 사용하여 분할된 외부 테이블에 SELECT 문의 결과를 삽입합니다. 파티션 열은 하드 코딩되지 않습니다. 데이터는 기존 파티션 폴더에 자동으로 추가되거나 새 파티션이 추가된 경우 새 폴더에 추가됩니다.

```
INSERT INTO spectrum.customer
SELECT name, age, gender, month, day FROM local_customer;
```

LOCK

데이터베이스 테이블에 대한 액세스를 제한합니다. 이 명령은 트랜잭션 블록 내에서 실행될 때만 의미가 있습니다.

LOCK 명령은 "ACCESS EXCLUSIVE" 모드에서 테이블 수준 잠금을 통해, 필요한 경우 서로 상충하는 잠금이 해제될 때까지 대기합니다. 이런 식으로 테이블을 명시적으로 잠그면 다른 트랜잭션 또는 세션에서 테이블에 대한 읽기 및 쓰기 작업을 시도할 때 이런 작업이 대기 상태로 전환됩니다. 한 사용자에 의해 생성되는 명시적 테이블 잠금은 일시적으로 다른 사용자가 그 테이블에서 데이터를 선택하거나 테이블로 데이터를 로드하지 못하게 막습니다. LOCK 명령을 포함한 트랜잭션이 완료되면 잠금이 해제됩니다.

쓰기 작업과 같이, 테이블을 참조하는 명령에 의해 덜 제한적인 테이블 잠금이 암시적으로 이루어집니다. 예를 들어, 한 사용자가 테이블을 업데이트하는 동안 다른 사용자가 테이블에서 데이터 읽기를 시도할 경우 이미 커밋된 데이터의 스냅샷이 읽히게 됩니다. (경우에 따라 쿼리가 직렬화 가능 격리 규칙을 위반한 경우에는 중지됩니다.) 동시 쓰기 작업 관리 섹션을 참조하세요.

DROP TABLE 및 TRUNCATE와 같은 일부 DDL 작업은 배타적 잠금을 생성합니다. 이런 작업으로 인해 데이터를 읽을 수 없습니다.

잠금 충돌이 발생하는 경우 Amazon Redshift는 충돌이 발생한 트랜잭션을 시작한 사용자에게 알리는 오류 메시지를 표시합니다. 잠금 충돌을 수신한 트랜잭션은 중지됩니다. 잠금 충돌이 발생할 때마다 Amazon Redshift가 STL_TR_CONFLICT 테이블에 항목을 작성합니다.

구문

```
LOCK [ TABLE ] table_name [, ...]
```

파라미터

TABLE

선택적 키워드입니다.

table_name

잠글 테이블의 이름입니다. 테이블 이름의 쉼표로 구분된 목록을 사용하여 두 개 이상의 테이블을 잠글 수 있습니다. 뷰를 잠글 수 없습니다.

예제

```
begin;

lock event, sales;
```

```
...
```

MERGE

소스 테이블의 행을 대상 테이블에 조건부로 병합합니다. 기존에 이는 여러 삽입, 업데이트 또는 삭제 문을 개별적으로 사용해야만 달성할 수 있습니다. MERGE를 통해 결합할 수 있는 작업에 대한 자세한 내용은 UPDATE, DELETE 및 INSERT를 참조하세요.

구문

```
MERGE INTO target_table
USING source_table [ [ AS ] alias ]
ON match_condition
[ WHEN MATCHED THEN { UPDATE SET col_name = { expr } [,...] | DELETE }
WHEN NOT MATCHED THEN INSERT [ ( col_name [,...] ) ] VALUES ( { expr } [, ...] ) |
REMOVE DUPLICATES ]
```

파라미터

target_table

MERGE 문이 병합되는 임시 또는 영구 테이블입니다.

source_table

target_table에 병합할 행을 제공하는 임시 또는 영구 테이블입니다. source_table은 스펙트럼 테이블일 수도 있습니다.

별칭

source_table의 임시 대체 이름입니다.

이 파라미터는 선택 사항입니다. AS로 시작하는 별칭도 선택 사항입니다.

match_condition

source_table의 행이 target_table의 행과 일치할 수 있는지 여부를 결정하는 데 사용되는 소스 테이블 열과 대상 테이블 열 사이에 동일한 술어를 지정합니다. 조건이 충족되면 MERGE는 해당 행에 대해 matching_clause를 실행합니다 그렇지 않으면 MERGE는 해당 행에 대해 not_matched_clause를 실행합니다.

WHEN MATCHED

소스 행과 대상 행 간의 일치 조건이 참으로 평가될 때 실행할 작업을 지정합니다. UPDATE 조치 또는 DELETE 조치를 지정할 수 있습니다.

UPDATE

target_table에서 일치하는 행을 업데이트합니다. 지정한 col_name의 값만 업데이트됩니다.

DELETE

target_table에서 일치하는 행을 삭제합니다.

WHEN NOT MATCHED

일치 조건이 거짓 또는 알 수 없음으로 평가될 때 실행할 작업을 지정합니다. 이 절에 대한 INSERT 삽입 조치만 지정할 수 있습니다.

INSERT

match_condition에 따라 target_table의 행과 일치하지 않는 source_table의 target_table 행에 삽입합니다. 임의의 순서대로 대상 col_name을 나열할 수 있습니다. col_name 값을 제공하지 않으면 기본 순서는 선언된 순서의 모든 테이블 열입니다.

col_name

수정하려는 하나 이상의 열 이름입니다. 대상 열을 지정할 때 테이블 이름을 포함하지 마세요.

expr

col_name의 새 값을 정의하는 표현식입니다.

REMOVE DUPLICATES

MERGE 명령이 단순 모드에서 실행되도록 지정합니다. 단순 모드의 요구 사항은 다음과 같습니다.

- target_table과 source_table은 열 수와 호환 가능한 열 유형이 같아야 합니다.
- MERGE 명령에서 WHEN 절과 UPDATE 및 INSERT 절을 생략해야 합니다.
- MERGE 명령에서 REMOVE DUPLICATES 절을 사용해야 합니다.

단순 모드에서 MERGE는 다음 작업을 수행합니다.

- target_table에서 source_table과 일치하는 행이 있으면 source_table의 값과 일치하도록 업데이트됩니다.
- source_table에서 target_table과 일치하지 않는 행은 target_table에 삽입됩니다.

- target_table의 여러 행이 source_table의 동일한 행과 일치하면 중복된 행이 제거됩니다. Amazon Redshift는 한 행을 유지하고 업데이트합니다. source_table의 행과 일치하지 않는 중복된 행은 변경되지 않습니다.

REMOVE DUPLICATES를 사용하면 WHEN MATCHED와 WHEN NOT MATCHED를 사용하는 것보다 성능이 향상됩니다. target_table과 source_table이 호환되고 target_table에 중복된 행을 보존할 필요가 없는 경우에는 REMOVE DUPLICATES를 사용하는 것이 좋습니다.

사용 노트

- MERGE 문을 실행하려면 source_table 및 target_table의 소유자이거나 해당 테이블에 대한 SELECT 권한이 있어야 합니다. 또한 MERGE 문에 포함된 작업에 따라 target_table에 대한 UPDATE, DELETE 및 INSERT 권한이 있어야 합니다.
- target_table은 시스템 테이블, 카탈로그 테이블 또는 외부 테이블일 수 없습니다.
- source_table 및 target_table은 동일한 테이블일 수 없습니다.
- MERGE 문에는 WITH 절을 사용할 수 없습니다.
- target_table의 행은 source_table의 여러 행과 일치할 수 없습니다.

다음 예제를 검토하십시오.

```
CREATE TABLE target (id INT, name CHAR(10));
CREATE TABLE source (id INT, name CHAR(10));

INSERT INTO target VALUES (1, 'Bob'), (2, 'John');
INSERT INTO source VALUES (1, 'Tony'), (1, 'Alice'), (3, 'Bill');

MERGE INTO target USING source ON target.id = source.id
WHEN MATCHED THEN UPDATE SET id = source.id, name = source.name
WHEN NOT MATCHED THEN INSERT VALUES (source.id, source.name);
ERROR: Found multiple matches to update the same tuple.

MERGE INTO target USING source ON target.id = source.id
WHEN MATCHED THEN DELETE
WHEN NOT MATCHED THEN INSERT VALUES (source.id, source.name);
ERROR: Found multiple matches to update the same tuple.
```

source 테이블에 ID 값이 1인 행이 여러 개 있기 때문에 두 MERGE 문 모두에서 작업이 실패합니다.

- match_condition 및 expr은 SUPER 형식 열을 부분적으로 참조할 수 없습니다. 예를 들어 SUPER 형식 객체가 배열이나 구조체인 경우 match_condition 또는 expr에 해당 열의 개별 요소를 사용할 수 없지만 전체 열을 사용할 수는 있습니다.

  다음 예제를 검토하십시오.

```
CREATE TABLE IF NOT EXISTS target (key INT, value SUPER);
CREATE TABLE IF NOT EXISTS source (key INT, value SUPER);

INSERT INTO target VALUES (1, JSON_PARSE('{"key": 88}'));
INSERT INTO source VALUES (1, ARRAY(1, 'John')), (2, ARRAY(2, 'Bill'));

MERGE INTO target USING source ON target.key = source.key
WHEN matched THEN UPDATE SET value = source.value[0]
WHEN NOT matched THEN INSERT VALUES (source.key, source.value[0]);
ERROR: Partial reference of SUPER column is not supported in MERGE statement.
```

  SUPER 유형형식에 대한 자세한 내용은 SUPER 형식을 참조하세요.
- source_table이 큰 경우 target_table과 source_table 모두의 조인 열을 분산 키로 정의하면 성능이 향상될 수 있습니다.
- REMOVE DUPLICATES 절을 사용하려면 target_table에 대한 SELECT, INSERT, DELETE 권한이 필요합니다.
- source_table은 뷰 또는 하위 쿼리일 수 있습니다. 다음은 source_table이 중복 행을 제거하는 하위 쿼리인 MERGE 문의 예제입니다.

```
MERGE INTO target
USING (SELECT id, name FROM source GROUP BY 1, 2) as my_source
ON target.id = my_source.id
WHEN MATCHED THEN UPDATE SET id = my_source.id, name = my_source.name
WHEN NOT MATCHED THEN INSERT VALUES (my_source.id, my_source.name);
```

예시

다음 예에서는 두 개의 테이블을 만든 다음 테이블에서 MERGE 작업을 실행하여 대상 테이블에서 일치하는 행을 업데이트하고 일치하지 않는 행을 삽입합니다. 그런 다음 소스 테이블에 다른 값을 삽입하고 다른 MERGE 작업을 실행합니다. 이번에는 일치하는 행을 삭제하고 소스 테이블에서 새 행을 삽입합니다.

먼저 소스 및 대상 테이블을 만들고 채웁니다.

```
CREATE TABLE target (id INT, name CHAR(10));
CREATE TABLE source (id INT, name CHAR(10));

INSERT INTO target VALUES (101, 'Bob'), (102, 'John'), (103, 'Susan');
INSERT INTO source VALUES (102, 'Tony'), (103, 'Alice'), (104, 'Bill');

SELECT * FROM target;
 id  |    name
-----+------------
 101 | Bob
 102 | John
 103 | Susan
(3 rows)

SELECT * FROM source;
 id  |    name
-----+------------
 102 | Tony
 103 | Alice
 104 | Bill
(3 rows)
```

그런 다음 소스 테이블을 대상 테이블에 병합하여 대상 테이블을 일치하는 행으로 업데이트하고 일치하는 행이 없는 소스 테이블의 행을 삽입합니다.

```
MERGE INTO target USING source ON target.id = source.id
WHEN MATCHED THEN UPDATE SET id = source.id, name = source.name
WHEN NOT MATCHED THEN INSERT VALUES (source.id, source.name);

SELECT * FROM target;
 id  |    name
-----+------------
 101 | Bob
 102 | Tony
 103 | Alice
 104 | Bill
(4 rows)
```

id 값이 102 및 103인 행은 대상 테이블의 name 값과 일치하도록 업데이트됩니다. 또한 id 값이 104이고 name 값이 Bill인 새 행이 대상 테이블에 삽입됩니다.

그런 다음 소스 테이블에 새 행을 삽입합니다.

```
INSERT INTO source VALUES (105, 'David');

SELECT * FROM source;
 id  |    name
-----+------------
 102 | Tony
 103 | Alice
 104 | Bill
 105 | David
(4 rows)
```

마지막으로 대상 테이블에서 일치하는 행을 삭제하고 일치하지 않는 행을 삽입하는 병합 작업을 실행합니다.

```
MERGE INTO target USING source ON target.id = source.id
WHEN MATCHED THEN DELETE
WHEN NOT MATCHED THEN INSERT VALUES (source.id, source.name);

SELECT * FROM target;
 id  |    name
-----+------------
 101 | Bob
 105 | David
(2 rows)
```

id 값이 102, 103 및 104인 행이 대상 테이블에서 삭제되고 id 값이 105이고 name 값이 David인 새 행이 대상 테이블에 삽입됩니다.

다음 예제는 REMOVE DUPLICATES 절을 사용하는 MERGE 명령의 단순화된 구문을 보여줍니다.

```
CREATE TABLE target (id INT, name CHAR(10));
CREATE TABLE source (id INT, name CHAR(10));

INSERT INTO target VALUES (30, 'Tony'), (11, 'Alice'), (23, 'Bill');
INSERT INTO source VALUES (23, 'David'), (22, 'Clarence');

MERGE INTO target USING source ON target.id = source.id REMOVE DUPLICATES;

SELECT * FROM target;
id | name
---+------------
30 | Tony
```

```
11 | Alice
23 | David
22 | Clarence
(4 rows)
```

다음 예제는 REMOVE DUPLICATES 절을 사용하여 source_table에 일치하는 행이 있는 경우 target_table에서 중복 행을 제거하는 MERGE 명령의 단순화된 구문을 보여줍니다.

```
CREATE TABLE target (id INT, name CHAR(10));
CREATE TABLE source (id INT, name CHAR(10));

INSERT INTO target VALUES (30, 'Tony'), (30, 'Daisy'), (11, 'Alice'), (23, 'Bill'),
 (23, 'Nikki');
INSERT INTO source VALUES (23, 'David'), (22, 'Clarence');

MERGE INTO target USING source ON target.id = source.id REMOVE DUPLICATES;

SELECT * FROM target;
id | name
---+------------
30 | Tony
30 | Daisy
11 | Alice
23 | David
22 | Clarence
(5 rows)
```

MERGE가 실행된 후에는 target_table에서 ID 값이 23인 행이 하나뿐입니다. source_table에 ID 값이 30인 행이 없었기 때문에 ID 값이 30인 두 개의 중복된 행은 target_table에 그대로 남아 있습니다.

다음 사항도 참조하세요.

INSERT, UPDATE, DELETE

PREPARE

실행을 위한 문을 준비합니다.

PREPARE는 준비된 문을 생성합니다. PREPARE 문이 실행될 때 지정된 문(SELECT, INSERT, UPDATE 또는 DELETE)이 구문 분석, 재작성 및 계획됩니다. 준비된 문에 대해 EXECUTE 명령이 실행될 때 Amazon Redshift는 (지정된 파라미터 값을 기반으로 성능을 개선하기 위해) 쿼리 실행 계획을 선택적으로 수정한 후 준비된 문을 실행할 수 있습니다.

구문

```
PREPARE plan_name [ (datatype [, ...] ) ] AS statement
```

파라미터

plan_name

이 특정한 준비된 문에 주어지는 임의의 이름입니다. 단일 세션 내에서 고유해야 하며 후속적으로 이전에 준비된 문을 실행하거나 할당 취소하는 데 사용됩니다.

DataType

준비된 문에 대한 파라미터의 데이터 형식입니다. 준비된 문 자체의 파라미터를 참조하려면 $1, $2 등을 사용하십시오.

statement

임의의 SELECT, INSERT, UPDATE 또는 DELETE 문입니다.

사용 노트

준비된 문은 파라미터를 취할 수 있습니다. 파라미터는 준비된 문이 실행될 때 문으로 대체되어 입력되는 값입니다. 준비된 문에 파라미터를 포함하려면 PREPARE 문에 데이터 형식 목록을 제공하고, 준비되는 문 자체에서 $1, $2의 표기법을 사용하여 위치에 따라 파라미터를 참조합니다. 문을 실행할 때 EXECUTE 문에서 이런 파라미터의 실제 값을 지정합니다. 자세한 내용은 EXECUTE 섹션을 참조하세요.

준비된 문은 현재 세션이 진행되는 기간 동안만 지속됩니다. 세션이 종료되면 준비된 문이 삭제되므로 다시 사용하려면 준비된 문을 다시 만들어야 합니다. 이는 곧 다중 동시 데이터베이스 클라이언트가 단 하나의 준비된 문을 사용할 수는 없다는 의미이기도 하지만, 각 클라이언트는 사용할 준비된 문을 스스로 만들 수 있습니다. DEALLOCATE 명령을 사용하여 준비된 문을 수동으로 제거할 수 있습니다.

준비된 문은 단일 세션을 사용하여 많은 수의 유사한 문을 실행할 때 최대의 성능상 이점이 있습니다. 언급한 바와 같이, 준비된 문을 새로 실행할 때마다 Amazon Redshift는 지정된 파라미터 값을 기반으로 성능을 개선하도록 쿼리 실행 계획을 수정할 수 있습니다. Amazon Redshift가 특정 EXECUTE 문에 대해 선택한 쿼리 실행 계획을 검사하려면 EXPLAIN 명령을 사용합니다.

Amazon Redshift가 쿼리 최적화를 위해 수집하는 통계와 쿼리 계획에 대한 자세한 내용은 ANALYZE 명령을 참조하세요.

예시

임시 테이블을 생성하고 INSERT 문을 준비한 후 실행합니다.

```
DROP TABLE IF EXISTS prep1;
CREATE TABLE prep1 (c1 int, c2 char(20));
PREPARE prep_insert_plan (int, char)
AS insert into prep1 values ($1, $2);
EXECUTE prep_insert_plan (1, 'one');
EXECUTE prep_insert_plan (2, 'two');
EXECUTE prep_insert_plan (3, 'three');
DEALLOCATE prep_insert_plan;
```

SELECT 문을 준비한 후 실행합니다.

```
PREPARE prep_select_plan (int)
AS select * from prep1 where c1 = $1;
EXECUTE prep_select_plan (2);
EXECUTE prep_select_plan (3);
DEALLOCATE prep_select_plan;
```

다음 사항도 참조하세요.

DEALLOCATE, EXECUTE

REFRESH MATERIALIZED VIEW

구체화된 보기를 새로 고칩니다.

구체화된 보기를 생성할 때 그 콘텐츠는 해당 시점에서 기본 데이터베이스 테이블의 상태를 반영합니다. 애플리케이션이 기본 테이블의 데이터를 변경하더라도 구체화된 보기의 데이터는 변경되지 않습니다.

구체화된 뷰에서 데이터를 업데이트하기 위해 언제든지 `REFRESH MATERIALIZED VIEW` 문을 사용할 수 있습니다. 이 문을 사용하면 Amazon Redshift가 기본 테이블 또는 테이블에서 발생한 변경 사항을 식별한 다음 해당 변경 사항을 구체화된 뷰에 적용합니다.

구체화된 뷰에 대한 자세한 내용은 Amazon Redshift의 구체화된 뷰 섹션을 참조하세요.

구문

```
REFRESH MATERIALIZED VIEW mv_name
```

파라미터

mv_name

새로 고칠 구체화된 보기의 이름입니다.

사용 노트

구체화된 보기의 소유자만 해당 구체화된 보기에 대한 REFRESH MATERIALIZED VIEW 작업을 수행할 수 있습니다. 또한 REFRESH MATERIALIZED VIEW를 성공적으로 실행하려면 소유자에게 기준 기본 테이블에 대한 SELECT 권한이 있어야 합니다.

REFRESH MATERIALIZED VIEW 명령은 자체 트랜잭션으로 실행됩니다. Amazon Redshift 트랜잭션 의미 체계에 따라 REFRESH 명령에 표시되는 기본 테이블의 데이터 또는 REFRESH 명령에 의한 변경 사항이 Amazon Redshift에서 실행 중인 다른 트랜잭션에 표시되는 시기를 결정합니다.

- 증분 구체화된 보기의 경우 REFRESH MATERIALIZED VIEW는 이미 커밋된 기본 테이블 행만 사용합니다. 따라서 동일한 트랜잭션에서 데이터 조작 언어(DML) 문 이후에 새로 고침 작업이 실행되면 해당 DML 문의 변경 내용이 새로 고침되지 않습니다.
- 전체 새로 고침한 구체화된 뷰의 경우 REFRESH MATERIALIZED VIEW를 실행하면 일반적인 Amazon Redshift 트랜잭션 의미 체계에 따라 새로 고침 트랜잭션에 모든 기본 테이블 행이 표시됩니다.
- 입력 인수 유형에 따라 Amazon Redshift는 DATE(타임스탬프), DATE_PART(날짜, 시간, 간격, 시간-tz), DATE_TRUNC(타임스탬프, 간격) 등 특정 입력 인수 유형을 사용하는 함수에 대한 구체화된 보기에 대한 증분 새로 고침을 계속 지원합니다.
- 증분 새로 고침은 기본 테이블이 데이터 공유에 있는 구체화된 뷰에서 지원됩니다.

Amazon Redshift의 일부 작업은 구체화된 뷰와 상호 작용합니다. 구체화된 보기를 정의하는 쿼리가 증분 새로 고침에 적합한 SQL 기능만 사용하더라도 이러한 작업 중 일부는 REFRESH MATERIALIZED VIEW 작업이 구체화된 보기를 완전히 다시 계산하도록 강제할 수 있습니다. 예:

- 구체화된 보기를 새로 고치지 않으면 백그라운드 vacuum 작업이 차단될 수 있습니다. 내부적으로 정의된 임계값 기간이 지나면 vacuum 작업을 실행할 수 있습니다. 이러한 vacuum 작업이 발생하면 다음에 새로 고칠 때 종속 구체화된 보기가 다시 계산되도록 표시됩니다(증분인 경우에도 해당). VACUUM에 대한 자세한 내용은 VACUUM 섹션을 참조하세요. 이벤트 및 상태 변경에 대한 자세한 내용은 STL_MV_STATE 섹션을 참조하세요.

- 일부 작업은 기본 테이블에서 사용자가 시작한 경우 다음에 REFRESH 작업이 실행될 때 구체화된 보기가 완전히 다시 계산됩니다. 이러한 작업의 예로는 수동으로 호출되는 VACUUM, 클래식 크기 조정, ALTER DISTKEY 작업, ALTER SORTKEY 작업 및 자르기 작업이 있습니다. 경우에 따라 자동 작업으로 인해 다음에 REFRESH 작업을 실행할 때 구체화된 뷰가 완전히 다시 계산될 수도 있습니다. 예를 들어 자동 진공 삭제 작업으로 인해 전체 재계산이 발생할 수 있습니다. 이벤트 및 상태 변경에 대한 자세한 내용은 STL_MV_STATE 섹션을 참조하세요.

데이터 공유의 구체화된 뷰에 대한 증분 새로 고침

Amazon Redshift는 기본 테이블이 공유될 때 소비자 데이터 공유의 구체화된 뷰에 대한 자동 및 증분 새로 고침을 지원합니다. 증분 새로 고침은 Amazon Redshift가 이전 새로 고침 이후에 발생한 기본 테이블의 변경 사항을 식별하고 구체화된 뷰의 해당 레코드만 업데이트하는 작업입니다. 이 동작에 대한 자세한 내용은 구체화된 뷰 생성을 참조하세요.

증분 새로 고침에 대한 제한 사항

Amazon Redshift는 다음 SQL 요소 중 하나를 사용하는 쿼리로 정의된 구체화된 뷰에 대해 증분 새로 고침을 지원하지 않습니다.

- OUTER JOIN(RIGHT, LEFT 또는 FULL).
- 세트 작업: UNION, INTERSECT, EXCEPT, MINUS.
- UNION ALL - 하위 쿼리에서 발생하고 집계 함수 또는 GROUP BY 절이 쿼리에 있는 경우.
- 집계 함수: MEDIAN, PERCENTILE_CONT, LISTAGG, STDDEV_SAMP, STDDEV_POP, APPROXIMATE COUNT, APPROXIMATE PERCENTILE 및 비트 단위 집계 함수.

  > **Note**
  >
  > COUNT, SUM, MIN, MAX 및 AVG 집계 함수가 지원됩니다.

- DISTINCT COUNT, DISTINCT SUM 등과 같은 DISTINCT 집계 함수.
- 창 함수.
- 일반 하위 표현식 최적화와 같은 쿼리 최적화를 위해 임시 테이블을 사용하는 쿼리입니다.
- 하위 쿼리
- 구체화된 뷰를 정의하는 쿼리에서 다음 형식을 참조하는 외부 테이블
  - Delta Lake
  - Hudi

위에 나열된 형식 외의 다른 형식을 사용하는 정의된 구체화된 뷰에 대한 증분 새로 고침이 지원됩니다. 자세한 내용은 Amazon Redshift Spectrum의 외부 데이터 레이크 테이블에 대한 구체화된 뷰 단원을 참조하십시오.

- 변경 가능한 함수 - 날짜-시간 함수, RANDOM 및 STABLE이 아닌 사용자 정의 함수 등.
- 제로 ETL 통합을 위한 증분 새로 고침과 관련된 제한 사항은 Amazon Redshift와 제로 ETL 통합을 사용할 때 고려할 사항을 참조하세요.

VACUUM과 같은 백그라운드 작업이 구체화된 뷰 새로 고침 작업에 미치는 영향을 포함하여 구체화된 뷰 제한에 대한 자세한 내용은 사용 노트 섹션을 참조하세요.

예시

다음 예제는 `tickets_mv` 구체화된 보기를 새로 고칩니다.

```
REFRESH MATERIALIZED VIEW tickets_mv;
```

reset

구성 파라미터의 값을 기본값으로 복원합니다.

지정된 단일 파라미터 또는 모든 파라미터를 한 번에 재설정할 수 있습니다. 파라미터를 특정 값으로 설정하려면 SET 명령을 사용합니다. 파라미터의 현재 값을 표시하려면 SET 명령을 사용합니다.

구문

```
RESET { parameter_name | ALL }
```

다음 문은 세션 컨텍스트 변수의 값을 NULL로 설정합니다.

```
RESET { variable_name | ALL }
```

파라미터

parameter_name

설정할 파라미터의 이름. 파라미터에 대한 추가 설명서는 서버 구성 수정 섹션을 참조하세요.

ALL

모든 세션 컨텍스트 변수를 포함하여 모든 런타임 파라미터를 재설정합니다.

variable

재설정할 변수의 이름입니다. RESET 값이 세션 컨텍스트 변수인 경우 Amazon Redshift는 이를 NULL로 설정합니다.

예시

다음 예에서는 query_group 파라미터를 기본값으로 재설정합니다.

```
reset query_group;
```

다음 예에서는 모든 런타임 파라미터를 기본값으로 재설정합니다.

```
reset all;
```

다음 예에서는 컨텍스트 변수를 재설정합니다.

```
RESET app_context.user_id;
```

REVOKE

사용자 또는 역할에서 테이블 생성, 삭제 또는 업데이트 권한과 같은 액세스 권한을 제거합니다.

ON SCHEMA 구문을 사용하는 데이터베이스 사용자 및 역할에 외부 스키마에 대한 사용 권한을 부여하거나 취소할 수 있습니다. AWS Lake Formation에서 ON EXTERNAL SCHEMA를 사용하는 경우 AWS Identity and Access Management(IAM) 역할에 대한 권한의 GRANT 및 REVOKE만 허용됩니다. 권한 목록은 구문을 참조하세요.

저장 프로시저의 경우 USAGE ON LANGUAGE plpgsql 권한은 기본적으로 PUBLIC에 허용됩니다. EXECUTE ON PROCEDURE 권한은 기본적으로 소유자 및 수퍼유저에게만 허용됩니다.

REVOKE 명령에서 제거하려는 권한을 지정합니다. 권한을 부여하려면 GRANT 명령을 사용합니다.

구문

```
REVOKE [ GRANT OPTION FOR ]
```

```
{ { SELECT | INSERT | UPDATE | DELETE | DROP | REFERENCES | ALTER | TRUNCATE } [,...] |
 ALL [ PRIVILEGES ] }
ON { [ TABLE ] table_name [, ...] | ALL TABLES IN SCHEMA schema_name [, ...] }
FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[ RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
{ { CREATE | TEMPORARY | TEMP | ALTER } [,...] | ALL [ PRIVILEGES ] }
ON DATABASE db_name [, ...]
FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[ RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
{ { CREATE | USAGE | ALTER | DROP } [,...] | ALL [ PRIVILEGES ] }
ON SCHEMA schema_name [, ...]
FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[ RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
EXECUTE
    ON FUNCTION function_name ( [ [ argname ] argtype [, ...] ] ) [, ...]
    FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[ RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
{ { EXECUTE } [,...] | ALL [ PRIVILEGES ] }
    ON PROCEDURE procedure_name ( [ [ argname ] argtype [, ...] ] ) [, ...]
    FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[ RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
USAGE
    ON LANGUAGE language_name [, ...]
    FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[ RESTRICT ]

REVOKE [GRANT OPTION FOR]
{ { ALTER | DROP} [,...] | ALL [ PRIVILEGES ] }
    ON COPY JOB job_name [,...]
    FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
```

테이블에 대한 열 수준 권한 취소

다음은 Amazon Redshift 테이블 및 뷰에 대한 열 수준 권한에 대한 구문입니다.

```
REVOKE { { SELECT | UPDATE } ( column_name [, ...] ) [, ...] | ALL [ PRIVILEGES ]
 ( column_name [,...] ) }
     ON { [ TABLE ] table_name [, ...] }
     FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
     [ RESTRICT ]
```

ASSUMEROLE 권한 취소

다음은 지정된 역할을 가진 사용자 및 그룹에서 ASSUMEROLE 권한을 취소하는 구문입니다.

```
REVOKE ASSUMEROLE
    ON { 'iam_role' [, ...]  | default | ALL }
    FROM { user_name | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
    FOR { ALL | COPY | UNLOAD | EXTERNAL FUNCTION | CREATE MODEL }
```

Lake Formation용 Redshift Spectrum에 대한 권한 취소

다음은 Lake Formation과 Redshift Spectrum 통합을 위한 구문입니다.

```
REVOKE [ GRANT OPTION FOR ]
{ SELECT | ALL [ PRIVILEGES ] } ( column_list )
    ON EXTERNAL TABLE schema_name.table_name
    FROM { IAM_ROLE iam_role } [, ...]

REVOKE [ GRANT OPTION FOR ]
{ { SELECT | ALTER | DROP | DELETE | INSERT }  [, ...] | ALL [ PRIVILEGES ] }
    ON EXTERNAL TABLE schema_name.table_name [, ...]
    FROM { { IAM_ROLE iam_role } [, ...] | PUBLIC }

REVOKE [ GRANT OPTION FOR ]
{ { CREATE | ALTER | DROP }  [, ...] | ALL [ PRIVILEGES ] }
    ON EXTERNAL SCHEMA schema_name [, ...]
    FROM { IAM_ROLE iam_role } [, ...]
```

데이터 공유 권한 취소

생산자 측 데이터 공유 권한

다음은 REVOKE를 사용하여 사용자 또는 역할에서 ALTER 또는 SHARE 권한을 제거할 때 사용하는 구문입니다. 권한이 취소된 사용자는 더 이상 데이터 공유를 변경하거나 소비자에게 사용 권한을 부여할 수 없습니다.

```
REVOKE { ALTER | SHARE } ON DATASHARE datashare_name
 FROM { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]
```

다음은 REVOKE를 사용하여 데이터 공유에 대한 소비자의 액세스 권한을 제거하는 구문입니다.

```
REVOKE USAGE
 ON DATASHARE datashare_name
 FROM NAMESPACE 'namespaceGUID' [, ...] | ACCOUNT 'accountnumber' [ VIA DATA CATALOG ]
 [, ...]
```

다음은 Lake Formation 계정에서 데이터 공유 사용 권한을 취소하는 방법의 예입니다.

```
REVOKE USAGE ON DATASHARE salesshare FROM ACCOUNT '123456789012' VIA DATA CATALOG;
```

소비자 측 데이터 공유 권한

다음은 datashare에서 생성된 특정 데이터베이스 또는 스키마에 대한 datashare 사용 권한에 대한 REVOKE 구문입니다. WITH PERMISSIONS 절을 사용하여 생성된 데이터베이스에서 사용 권한을 취소해도 기본 객체에 부여된 객체 수준 권한을 포함하여 사용자 또는 역할에 부여한 추가 권한은 취소되지 않습니다. 해당 사용자 또는 역할에 사용 권한을 다시 부여하면 사용 권한을 취소하기 전에 가졌던 추가 권한이 모두 유지됩니다.

```
REVOKE USAGE ON { DATABASE shared_database_name [, ...] | SCHEMA shared_schema}
 FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
```

범위가 지정된 권한 취소

범위가 지정된 권한을 사용하면 데이터베이스 또는 스키마 내 특정 유형의 모든 객체에 대한 권한을 사용자 또는 역할에 부여할 수 있습니다. 범위가 지정된 권한이 있는 사용자와 역할은 데이터베이스 또는 스키마 내의 모든 현재 및 미래 객체에 대한 지정된 권한을 갖습니다.

SVV_DATABASE_PRIVILEGES에서 데이터베이스 수준 범위 지정 권한의 범위를 볼 수 있습니다. SVV_SCHEMA_PRIVILEGES에서 스키마 수준 범위 지정 권한의 범위를 볼 수 있습니다.

범위가 지정된 권한에 대한 자세한 내용은 범위가 지정된 권한 섹션을 참조하세요.

다음은 사용자 또는 역할에서 범위가 지정된 권한을 취소할 때 사용하는 구문입니다.

```
REVOKE [ GRANT OPTION ]
{ CREATE | USAGE | ALTER | DROP } [,...] | ALL [ PRIVILEGES ] }
FOR SCHEMAS IN
DATABASE db_name
FROM { username | ROLE role_name } [, ...]

REVOKE [ GRANT OPTION ]
{ { SELECT | INSERT | UPDATE | DELETE | DROP | ALTER | TRUNCATE | REFERENCES }
 [, ...] } | ALL [PRIVILEGES] } }
FOR TABLES IN
{ SCHEMA schema_name [ DATABASE db_name ] | DATABASE db_name }
FROM { username | ROLE role_name } [, ...]

REVOKE [ GRANT OPTION ] { EXECUTE | ALL [ PRIVILEGES ] }
FOR FUNCTIONS IN
{ SCHEMA schema_name [DATABASE db_name ] | DATABASE db_name }
FROM { username | ROLE role_name } [, ...]

REVOKE [ GRANT OPTION ] { EXECUTE | ALL [ PRIVILEGES ] }
FOR PROCEDURES IN
{ SCHEMA schema_name [DATABASE db_name ] | DATABASE db_name }
FROM { username | ROLE role_name } [, ...]

REVOKE [ GRANT OPTION ] USAGE
FOR LANGUAGES IN
DATABASE db_name
FROM { username | ROLE role_name } [, ...]

REVOKE [GRANT_OPTION]
{ { CREATE | ALTER | DROP} [,...] | ALL [ PRIVILEGES ] }
FOR COPY JOBS
IN DATABASE db_name
FROM { username [ WITH GRANT OPTION ] | ROLE role_name } [, ...]
```

범위가 지정된 권한은 함수에 대한 권한과 프로시저에 대한 권한을 구별하지 않습니다. 예를 들어 다음 문은 Sales_schema 스키마의 함수와 프로시저 모두에 대한 bob의 EXECUTE 권한을 취소합니다.

```
REVOKE EXECUTE FOR FUNCTIONS IN SCHEMA Sales_schema FROM bob;
```

기계 학습 권한 취소

다음은 Amazon Redshift의 기계 학습 모델 권한을 위한 구문입니다.

```
REVOKE [ GRANT OPTION FOR ]
    CREATE MODEL FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
    [ RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { EXECUTE | ALL [ PRIVILEGES ] }
    ON MODEL model_name [, ...]

    FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
    [ RESTRICT ]
```

역할 권한 취소

다음은 Amazon Redshift에서 역할 권한을 취소하기 위한 구문입니다.

```
REVOKE [ ADMIN OPTION FOR ] { ROLE role_name } [, ...] FROM { user_name } [, ...]
```

```
REVOKE { ROLE role_name } [, ...] FROM { ROLE role_name } [, ...]
```

다음은 Amazon Redshift에서 역할에 대한 시스템 권한을 취소하기 위한 구문입니다.

```
REVOKE
  {
    { CREATE USER | DROP USER | ALTER USER |
    CREATE SCHEMA | DROP SCHEMA |
    ALTER DEFAULT PRIVILEGES |
    ACCESS CATALOG |
    CREATE TABLE | DROP TABLE | ALTER TABLE |
    CREATE OR REPLACE FUNCTION | CREATE OR REPLACE EXTERNAL FUNCTION |
    DROP FUNCTION |
    CREATE OR REPLACE PROCEDURE | DROP PROCEDURE |
    CREATE OR REPLACE VIEW | DROP VIEW |
    CREATE MODEL | DROP MODEL |
    CREATE DATASHARE | ALTER DATASHARE | DROP DATASHARE |
    CREATE LIBRARY | DROP LIBRARY |
    CREATE ROLE | DROP ROLE
    TRUNCATE TABLE
    VACUUM | ANALYZE | CANCEL }[, ...]
  }
  | { ALL [ PRIVILEGES ] }
FROM { ROLE role_name } [, ...]
```

행 수준 보안 정책 필터에 대한 설명 권한 취소

다음은 EXPLAIN 계획에서 쿼리의 행 수준 보안 정책 필터를 설명할 권한을 취소하는 구문입니다. REVOKE 문을 사용하여 이 권한을 취소할 수 있습니다.

```
REVOKE EXPLAIN RLS FROM ROLE rolename
```

다음은 쿼리에 대한 행 수준 보안 정책을 우회할 수 있는 권한을 부여하는 구문입니다.

```
REVOKE IGNORE RLS FROM ROLE rolename
```

다음은 지정된 행 수준 보안 정책에서 권한을 취소하는 구문입니다.

```
REVOKE SELECT ON [ TABLE ] table_name [, ...]
            FROM RLS POLICY policy_name [, ...]
```

파라미터

GRANT OPTION FOR

다른 사용자에게 지정된 권한을 허용하는 옵션만 취소하고 권한 자체를 취소하지는 않습니다. 그룹 또는 PUBLIC에서 GRANT OPTION을 취소할 수 없습니다.

SELECT

SELECT 문을 사용하여 테이블 또는 뷰에서 데이터를 선택하는 권한을 취소합니다.

INSERT

INSERT 문 또는 COPY 문을 사용하여 데이터를 테이블로 로드하는 권한을 취소합니다.

UPDATE

UPDATE 문을 사용하여 테이블 열을 업데이트하는 권한을 취소합니다.

DELETE

테이블에서 데이터 행을 삭제하는 권한을 취소합니다.

REFERENCES

외래 키 제약 조건을 생성하는 권한을 취소합니다. 참조되는 테이블과 참조하는 테이블 모두에 대해 이 권한을 취소해야 합니다.

TRUNCATE

테이블을 자를 수 있는 권한을 취소합니다. 이 권한이 없으면 테이블 소유자 또는 수퍼유저만 테이블을 자를 수 있습니다. TRUNCATE 명령에 대한 자세한 내용은 the section called “TRUNCATE” 섹션을 참조하세요.

ALL [PRIVILEGES]

지정된 사용자 또는 그룹에서 사용 가능한 모든 권한을 한 번에 취소합니다. PRIVILEGES 키워드는 옵션입니다.

> **Note**
>
> Amazon Redshift는 RULE 및 TRIGGER 권한을 지원하지 않습니다. 자세한 내용은 지원되지 않는 PostgreSQL 기능 섹션을 참조하세요.

ALTER

데이터베이스 객체에 따라, 사용자 또는 사용자 그룹에서 다음 권한을 취소합니다.

- 테이블의 경우 ALTER는 테이블 또는 뷰를 변경할 수 있는 권한을 철회합니다. 자세한 내용은 ALTER TABLE 단원을 참조하십시오.
- 데이터베이스의 경우 ALTER는 데이터베이스를 변경할 수 있는 권한을 철회합니다. 자세한 내용은 ALTER DATABASE 단원을 참조하십시오.
- 스키마의 경우 ALTER는 스키마를 변경할 수 있는 권한을 철회합니다. 자세한 내용은 ALTER SCHEMA 단원을 참조하십시오.
- 외부 테이블의 경우 ALTER는 Lake Formation에 활성화된 AWS Glue Data Catalog의 테이블을 변경할 수 있는 권한을 철회합니다. 이 권한은 Lake Formation을 사용하는 경우에만 적용됩니다.

DROP

데이터베이스 객체에 따라, 사용자 또는 역할에서 다음 권한을 취소합니다.

- 테이블의 경우 DROP은 테이블 또는 뷰를 삭제할 수 있는 권한을 취소합니다. 자세한 내용은 DROP TABLE 단원을 참조하십시오.
- 데이터베이스의 경우 DROP은 데이터베이스를 삭제할 수 있는 권한을 취소합니다. 자세한 내용은 DROP DATABASE 단원을 참조하십시오.
- 스키마의 경우 DROP은 스키마를 삭제할 수 있는 권한을 취소합니다. 자세한 내용은 DROP SCHEMA 단원을 참조하십시오.

ASSUMEROLE

지정된 역할을 가진 사용자, 역할 또는 그룹에서 COPY, UNLOAD, EXTERNAL FUNCTION 또는 CREATE MODEL 명령을 실행할 수 있는 권한을 취소합니다.

ON [TABLE] table_name

테이블 또는 뷰에 대해 지정된 권한을 취소합니다. TABLE 키워드는 옵션입니다.

ON ALL TABLES IN SCHEMA schema_name

참조되는 스키마에 있는 모든 테이블에 대해 지정된 권한을 취소합니다.

(column_name [,...]) ON TABLE table_name

Amazon Redshift 테이블 또는 뷰의 지정된 열에서 사용자, 그룹 또는 PUBLIC에서 지정된 권한을 취소합니다.

(column_list) ON EXTERNAL TABLE schema_name.table_name

참조된 스키마에서 Lake Formation 테이블의 지정된 열에 있는 IAM 역할에 지정된 권한을 취소합니다.

ON EXTERNAL TABLE schema_name.table_name

참조된 스키마에 지정된 Lake Formation 테이블의 IAM 역할에서 지정된 권한을 취소합니다.

ON EXTERNAL SCHEMA schema_name

참조된 스키마에 있는 IAM 역할에서 지정된 권한을 취소합니다.

FROM IAM_ROLE iam_role

권한을 잃는 IAM 사용자를 나타냅니다.

ROLE role_name

지정된 역할에서 권한을 취소합니다.

GROUP group_name

지정된 사용자 그룹에서 권한을 취소합니다.

PUBLIC

모든 사용자에게서 지정된 권한을 취소합니다. PUBLIC은 모든 사용자를 항상 포함하는 그룹을 나타냅니다. 개별 사용자의 권한은 PUBLIC에 부여된 권한, 사용자가 속한 그룹에 부여된 권한, 사용자에게 개별적으로 부여된 모든 권한의 합입니다.

Lake Formation 외부 테이블에서 PUBLIC을 취소하면 Lake Formation 모든 사람 그룹에서 권한이 취소됩니다.

CREATE

데이터베이스 객체에 따라, 사용자 또는 그룹에서 다음 권한을 취소합니다.

- 데이터베이스의 경우 REVOKE를 위한 CREATE 절을 사용하면 사용자가 데이터베이스 내에 스키마를 생성하지 못합니다.
- 스키마의 경우 REVOKE를 위한 CREATE 절을 사용하면 사용자가 스키마 내에 객체를 생성하지 못합니다. 객체의 이름을 바꾸려면 사용자가 CREATE 권한이 있고 이름을 바꿀 객체를 소유해야 합니다.

> **Note**
>
> 기본적으로 모든 사용자는 PUBLIC 스키마에서 CREATE 및 USAGE 권한을 갖습니다.

TEMPORARY | TEMP

지정된 데이터베이스에서 임시 테이블을 생성할 권한을 취소합니다.

> **Note**
>
> 기본적으로, 사용자는 PUBLIC 그룹에서 자동 멤버십으로 임시 테이블을 생성할 권한이 허용됩니다. 임의의 사용자가 임시 테이블을 생성할 권한을 제거하려면 PUBLIC 그룹에서 TEMP 권한을 취소한 다음, 특정 사용자 또는 사용자 그룹에 임시 테이블을 생성할 권한을 명시적으로 허용하세요.

ON DATABASE db_name

지정된 데이터베이스에 대한 권한을 취소합니다.

객체

특정 스키마 내의 객체에 대한 USAGE 권한을 취소하여 사용자가 객체에 액세스할 수 없게 만듭니다. 이런 객체에 대한 특정한 작업은 따로 취소해야 합니다(예: 함수에 대한 EXECUTE 권한).

> **Note**
>
> 기본적으로 모든 사용자는 PUBLIC 스키마에서 CREATE 및 USAGE 권한을 갖습니다.

ON SCHEMA schema_name

지정된 스키마에 대한 권한을 취소합니다. 스키마 권한을 사용하여 테이블의 생성을 제어할 수 있으며, 데이터베이스에 대한 CREATE 권한으로는 스키마의 생성만 제어할 수 있습니다.

RESTRICT

사용자가 직접 부여한 권한만 취소합니다. 이것이 기본 동작입니다.

EXECUTE ON PROCEDURE procedure_name

특정 저장 프로시저에 대해 EXECUTE 권한을 취소합니다. 저장 프로시저 이름은 오버로딩될 수 있기 때문에 해당 프로시저에 대한 인수 목록을 포함해야 합니다. 자세한 내용은 저장 프로시저 명령 단원을 참조하십시오.

EXECUTE ON ALL PROCEDURES IN SCHEMA procedure_name

참조되는 스키마에 있는 모든 프로시저에 대해 지정된 권한을 취소합니다.

USAGE ON LANGUAGE language_name

언어에 대한 USAGE 권한을 취소합니다. Python 사용자 정의 함수(UDF)에는 `plpythonu`를 사용합니다. SQL UDF에는 `sql`을 사용합니다. 저장 프로시저에는 `plpgsql`을 사용합니다.

UDF를 새로 만들려면 SQL 또는 `plpythonu`(Python)에 대한 언어에 사용 권한이 필요합니다. 기본적으로 USAGE ON LANGUAGE SQL은 PUBLIC에 허용됩니다. 그러나 특정 사용자 또는 그룹에 USAGE ON LANGUAGE PLPYTHONU 권한을 명시적으로 허용해야 합니다.

SQL에 대한 사용을 취소하려면 먼저 PUBLIC에서의 사용을 취소해야 합니다. 그런 다음 SQL UDF 생성이 허용된 사용자 또는 그룹에게만 SQL에 대한 사용 권한을 허용합니다. 다음은 PUBLIC에서의 SQL 사용을 취소한 후 사용자 그룹 `udf_devs`에 사용을 허용하는 예시입니다.

```
revoke usage on language sql from PUBLIC;
grant usage on language sql to group udf_devs;
```

자세한 내용은 UDF 보안 및 권한 단원을 참조하십시오.

저장 프로시저에 대한 사용을 취소하려면 먼저 PUBLIC에서의 사용을 취소해야 합니다. 그런 다음 저장 프로시저 생성이 허용된 사용자 또는 그룹에게만 `plpgsql`에 대한 사용 권한을 허용합니다. 자세한 내용은 저장 프로시저의 보안 및 권한 단원을 참조하십시오.

ON COPY JOB job_name

복사 작업에 대해 지정된 권한을 취소합니다.

FOR { ALL | COPY | UNLOAD | EXTERNAL FUNCTION | CREATE MODEL } [, ...]

권한이 취소되는 SQL 명령을 지정합니다. ALL을 지정하여 COPY, UNLOAD, EXTERNAL FUNCTION 및 CREATE MODEL 문에 대한 권한을 취소할 수 있습니다. 이 절은 ASSUMEROLE 권한을 취소하는 경우에만 적용됩니다.

ALTER

데이터 공유를 소유하지 않은 사용자 또는 사용자 그룹이 데이터 공유를 변경할 수 있도록 하는 ALTER 권한을 취소합니다. 이 권한은 데이터 공유에서 객체를 추가 또는 제거하거나 PUBLICACCESSIBLE 속성을 설정하는 데 필요합니다. 자세한 내용은 ALTER DATASHARE 단원을 참조하십시오.

SHARE

사용자 및 사용자 그룹이 데이터 공유에 소비자를 추가할 수 있는 권한을 취소합니다. 특정 소비자가 해당 클러스터에서 데이터 공유에 액세스하지 못하게 하려면 이 권한을 취소해야 합니다.

ON DATASHARE datashare_name

참조된 데이터 공유에 대해 지정된 권한을 부여합니다.

FROM username

권한을 잃는 사용자를 나타냅니다.

FROM GROUP group_name

권한을 잃는 사용자 그룹을 나타냅니다.

WITH GRANT OPTION

권한을 잃은 사용자가 다른 사용자의 동일한 권한을 취소할 수 있음을 나타냅니다. 그룹 또는 PUBLIC에 대해 WITH GRANT OPTION을 취소할 수 없습니다.

객체

소비자 계정 또는 동일한 계정 내의 네임스페이스에 대해 USAGE가 취소되면 지정된 소비자 계정 또는 계정 내의 네임스페이스는 읽기 전용 방식으로 datashare 및 datashare 객체에 액세스할 수 없습니다.

USAGE 권한을 취소하면 소비자에게서 데이터 공유에 대한 액세스 권한이 취소됩니다.

FROM NAMESPACE 'clusternamespace GUID'

소비자가 데이터 공유에 대한 권한을 상실한 동일한 계정의 네임스페이스를 나타냅니다. 네임스페이스는 128비트 영숫자 전역 고유 식별자(GUID)를 사용합니다.

TO ACCOUNT 'accountnumber' [VIA DATA CATALOG]

소비자가 데이터 공유에 대한 권한을 상실한 다른 계정의 계정 번호를 나타냅니다. 'VIA DATA CATALOG'를 지정하면 Lake Formation 계정에서 데이터 공유 사용이 취소됨을 나타냅니다. 계정 번호를 생략하면 클러스터를 소유한 계정에서 해지한다는 의미입니다.

ON DATABASE shared_database_name> [, ...]

지정된 데이터 공유에서 생성된 지정된 데이터베이스에 대해 지정된 사용 권한을 취소합니다.

ON SCHEMA shared_schema

지정된 데이터 공유에서 생성된 지정된 스키마에 대해 지정된 권한을 취소합니다.

FOR { SCHEMAS | TABLES | FUNCTIONS | PROCEDURES | LANGUAGES | COPY JOBS} IN

권한을 취소할 데이터베이스 객체를 지정합니다. IN 다음의 파라미터는 취소된 권한의 범위를 정의합니다.

CREATE MODEL

지정된 데이터베이스에서 기계 학습 모델을 생성하기 위한 CREATE MODEL 권한을 취소합니다.

ON MODEL model_name

특정 모델에 대해 EXECUTE 권한을 취소합니다.

ACCESS CATALOG

역할이 액세스할 수 있는 객체의 관련 메타데이터를 볼 수 있는 권한을 취소합니다.

[ADMIN OPTION FOR] { role } [, ...]

WITH ADMIN OPTION이 있는 지정된 사용자로부터 취소한 역할.

FROM { role } [, ...]

지정된 역할을 취소하는 역할.

사용 노트

REVOKE 사용 노트에 대한 자세한 내용은 the section called “사용 노트” 섹션을 참조하세요.

예시

REVOKE 사용 방법의 예는 the section called “예시” 섹션을 참조하세요.

사용 노트

객체에서 권한을 취소하려면 다음 조건 중 하나를 충족해야 합니다.

- 객체 소유자입니다.
- 수퍼유저입니다.
- 그 객체와 권한에 대해 허용된 권한이 있습니다.

  예를 들어, 다음 명령을 실행하면 사용자 HR은 직원 테이블에서 SELECT 명령을 수행할 뿐 아니라 다른 사용자에 대해 같은 권한을 허용하고 취소할 수 있습니다.

  ```
  grant select on table employees to HR with grant option;
  ```

  HR은 SELECT 이외의 작업 권한이나 직원 이외의 테이블에 대한 권한을 취소할 수 없습니다.

수퍼유저는 객체 권한을 설정하는 GRANT 및 REVOKE 명령과는 무관하게 모든 객체에 액세스할 수 있습니다.

PUBLIC은 모든 사용자를 항상 포함하는 그룹을 나타냅니다. 기본적으로 모든 PUBLIC 구성원은 PUBLIC 스키마에서 CREATE 및 USAGE 권한을 갖습니다. PUBLIC 스키마에 대한 사용자 권한을 제한하려면 먼저 PUBLIC 스키마에 대한 PUBLIC의 모든 권한을 취소한 다음 특정 사용자 혹은 그룹에게 권한을 허용합니다. 다음 예에서는 PUBLIC 스키마에서 테이블 생성 권한을 제어합니다.

```
revoke create on schema public from public;
```

Lake Formation 테이블에서 권한을 취소하려면 해당 테이블의 외부 스키마와 연결된 IAM 역할에 외부 테이블의 권한을 취소할 권한이 있어야 합니다. 다음 예제에서는 연결된 IAM 역할 `myGrantor`가 있는 외부 스키마를 생성합니다. IAM 역할 `myGrantor`는 다른 사람의 권한을 취소할 권한이 있습니다. REVOKE 명령은 외부 스키마와 연결된 IAM 역할 `myGrantor`의 권한을 사용하여 IAM 역할 `myGrantee`의 권한을 취소합니다.

```
create external schema mySchema
from data catalog
database 'spectrum_db'
iam_role 'arn:aws:iam::123456789012:role/myGrantor'
create external database if not exists;
```

```
revoke select
```

```
on external table mySchema.mytable
from iam_role 'arn:aws:iam::123456789012:role/myGrantee';
```

> **Note**
>
> IAM 역할에 Lake Formation에 사용되는 AWS Glue Data Catalog의 ALL 권한도 있는 경우 ALL 권한이 취소되지 않습니다. SELECT 권한만 취소됩니다. Lake Formation 콘솔에서 Lake Formation 권한을 볼 수 있습니다.

ASSUMEROLE 권한 취소에 대한 사용 노트

다음 사용 노트는 Amazon Redshift에서 ASSUMEROLE 권한을 취소하는 데 적용됩니다.

데이터베이스 슈퍼 사용자만 사용자 및 그룹에 대한 ASSUMEROLE 권한을 취소할 수 있습니다. 슈퍼 사용자는 항상 ASSUMEROLE 권한을 유지합니다.

사용자와 그룹에 ASSUMEROLE 권한을 사용하도록 설정하기 위해 슈퍼 사용자가 클러스터에서 다음 문을 한 번 실행합니다. 사용자와 그룹에 ASSUMEROLE 권한을 부여하기 전에 슈퍼 사용자가 클러스터에서 다음 문을 한 번 실행해야 합니다.

```
revoke assumerole on all from public for all;
```

기계 학습 권한 취소에 대한 사용 노트

ML 함수와 관련된 권한을 직접 부여하거나 취소할 수 없습니다. ML 함수는 ML 모델에 속하며 권한은 모델을 통해 제어됩니다. 대신 ML 모델과 관련된 권한을 취소할 수 있습니다. 다음 예제는 customer_churn 모델과 연결된 모든 사용자로부터 실행 권한을 취소하는 방법을 보여줍니다.

```
REVOKE EXECUTE ON MODEL customer_churn FROM PUBLIC;
```

ML 모델 customer_churn에 대한 사용자의 모든 권한을 취소할 수도 있습니다.

```
REVOKE ALL on MODEL customer_churn FROM ml_user;
```

스키마에 ML 함수가 있는 경우 해당 ML 함수가 이미 GRANT EXECUTE ON MODEL을 통해 EXECUTE 권한을 부여받았더라도 ML 함수와 관련된 EXECUTE 권한 부여 또는 취소는 실패합니다. CREATE MODEL 명령을 사용할 때는 별도의 스키마를 사용하여 ML 함수를 별도의 스키마에 따로 보관하는 것이 좋습니다. 다음 예제에서는 이 작업을 수행하는 방법을 보여 줍니다.

```
CREATE MODEL ml_schema.customer_churn
FROM customer_data
TARGET churn
FUNCTION ml_schema.customer_churn_prediction
IAM_ROLE default
SETTINGS (
 S3_BUCKET 'amzn-s3-demo-bucket'
);
```

예시

다음 예에서는 GUESTS 사용자 그룹에서 SALES 테이블에 대한 INSERT 권한을 취소합니다. 이 명령을 실행하면 GUESTS의 구성원이 INSERT 명령을 사용하여 SALES 테이블로 데이터를 로드할 수 없게 됩니다.

```
revoke insert on table sales from group guests;
```

다음 예에서는 사용자 fred에게서 QA_TICKIT 스키마에 있는 모든 테이블에 대한 SELECT 권한을 취소합니다.

```
revoke select on all tables in schema qa_tickit from fred;
```

다음 예에서는 사용자 bobr에 대한 뷰에서 선택할 권한을 취소합니다.

```
revoke select on table eventview from bobr;
```

다음 예에서는 모든 사용자에게서 TICKIT 데이터베이스에 임시 테이블을 만들 권한을 취소합니다.

```
revoke temporary on database tickit from public;
```

다음 예에서는 사용자 user1에게서 cust_profile 테이블의 cust_name 및 cust_phone 열에 대한 SELECT 권한을 취소합니다.

```
revoke select(cust_name, cust_phone) on cust_profile from user1;
```

다음 예에서는 sales_group 그룹에서 cust_profile 테이블의 cust_name 및 cust_phone 열에 대한 SELECT 권한과 cust_contact_preference 열에 대한 UPDATE 권한을 취소합니다.

```
revoke select(cust_name, cust_phone), update(cust_contact_preference) on cust_profile
 from group sales_group;
```

다음 예에서는 ALL 키워드를 사용하여 sales_admin 그룹에서 cust_profile 테이블의 세 열에 대한 SELECT 및 UPDATE 권한을 모두 취소하는 것을 보여 줍니다.

```
revoke ALL(cust_name, cust_phone,cust_contact_preference) on cust_profile from group
 sales_admin;
```

다음 예에서는 user2 사용자에게서 cust_profile_vw 뷰의 cust_name 열에 대한 SELECT 권한을 취소합니다.

```
revoke select(cust_name) on cust_profile_vw from user2;
```

데이터 공유에서 생성된 데이터베이스에서 USAGE 권한 취소의 예

다음 예시에서는 13b8833d-17c6-4f16-8fe4-1a018f5ed00d 네임스페이스에서 salesshare 데이터 공유에 대한 액세스 권한을 취소합니다.

```
REVOKE USAGE ON DATASHARE salesshare FROM NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

다음 예시에서는 Bob에게서 sales_db에 대한 USAGE 권한을 취소합니다.

```
REVOKE USAGE ON DATABASE sales_db FROM Bob;
```

다음 예시에서는 Analyst_role에게서 sales_schema에 대한 USAGE 권한을 취소합니다.

```
REVOKE USAGE ON SCHEMA sales_schema FROM ROLE Analyst_role;
```

범위가 지정된 권한 취소의 예

다음 예시에서는 Sales_db 데이터베이스의 모든 현재 및 미래 스키마에 대한 사용 권한을 Sales 역할에게서 취소합니다.

```
REVOKE USAGE FOR SCHEMAS IN DATABASE Sales_db FROM ROLE Sales;
```

다음 예시에서는 Sales_db 데이터베이스의 모든 현재 및 미래 테이블에 대한 SELECT 권한을 부여하는 권한을 alice라는 사용자에게서 취소합니다. alice는 Sales_db의 모든 테이블에 대한 액세스 권한을 유지합니다.

```
REVOKE GRANT OPTION SELECT FOR TABLES IN DATABASE Sales_db FROM alice;
```

다음 예시에서는 Sales_schema 스키마의 함수에 대한 EXECUTE 권한을 bob이라는 사용자에게서 취소합니다.

```
REVOKE EXECUTE FOR FUNCTIONS IN SCHEMA Sales_schema FROM bob;
```

다음 예시에서는 ShareDb 데이터베이스의 ShareSchema 스키마에 있는 모든 테이블에 대한 모든 권한을 Sales 역할에게서 취소합니다. 스키마를 지정할 때 두 부분으로 구성된 database.schema 형식을 사용하여 스키마의 데이터베이스를 지정할 수도 있습니다.

```
REVOKE ALL FOR TABLES IN SCHEMA ShareDb.ShareSchema FROM ROLE Sales;
```

다음 예시는 이전 예시와 동일합니다. 두 부분으로 구성된 형식을 사용하는 대신 DATABASE 키워드를 사용하여 스키마의 데이터베이스를 지정할 수 있습니다.

```
REVOKE ALL FOR TABLES IN SCHEMA ShareSchema DATABASE ShareDb FROM ROLE Sales;
```

ASSUMEROLE 권한 취소의 예

다음은 ASSUMEROLE 권한을 취소하는 예입니다.

슈퍼 사용자는 클러스터에서 다음 문을 한 번 실행하여 사용자와 그룹에 ASSUMEROLE 권한을 사용하도록 설정해야 합니다.

```
revoke assumerole on all from public for all;
```

다음 문은 모든 작업에 대한 모든 역할에 대한 사용자 reg_user1의 ASSUMEROLE 권한을 취소합니다.

```
revoke assumerole on all from reg_user1 for all;
```

ROLE 권한 취소의 예

다음은 sample_role1을 sample_role2에서 취소하는 예입니다.

```
CREATE ROLE sample_role2;
GRANT ROLE sample_role1 TO ROLE sample_role2;
REVOKE ROLE sample_role1 FROM ROLE sample_role2;
```

다음은 user1에서 시스템 권한을 취소하는 예입니다.

```
GRANT ROLE sys:DBA TO user1;
REVOKE ROLE sys:DBA FROM user1;
```

다음은 sample_role1 및 sample_role2를 user1에서 취소하는 예입니다.

```
CREATE ROLE sample_role1;
CREATE ROLE sample_role2;
GRANT ROLE sample_role1, ROLE sample_role2 TO user1;
REVOKE ROLE sample_role1, ROLE sample_role2 FROM user1;
```

다음은 user1에서 ADMIN OPTION을 사용하여 sample_role2를 취소하는 예입니다.

```
GRANT ROLE sample_role2 TO user1 WITH ADMIN OPTION;
REVOKE ADMIN OPTION FOR ROLE sample_role2 FROM user1;
REVOKE ROLE sample_role2 FROM user1;
```

다음은 sample_role1 및 sample_role2를 sample_role5에서 취소하는 예입니다.

```
CREATE ROLE sample_role5;
GRANT ROLE sample_role1, ROLE sample_role2 TO ROLE sample_role5;
REVOKE ROLE sample_role1, ROLE sample_role2 FROM ROLE sample_role5;
```

다음은 sample_role1에 대한 CREATE SCHEMA 및 DROP SCHEMA 시스템 권한을 취소하는 예입니다.

```
GRANT CREATE SCHEMA, DROP SCHEMA TO ROLE sample_role1;
REVOKE CREATE SCHEMA, DROP SCHEMA FROM ROLE sample_role1;
```

ROLLBACK

현재 트랜잭션을 중지하고 그 트랜잭션에서 이루어진 모든 업데이트를 삭제합니다.

이 명령은 ABORT 명령과 똑같은 기능을 수행합니다.

구문

```
ROLLBACK [ WORK | TRANSACTION ]
```

파라미터

Work

선택적 키워드입니다. 이 키워드는 저장 프로시저 내에서 지원되지 않습니다.

TRANSACTION

선택적 키워드입니다. WORK와 TRANSACTION은 동의어입니다. 둘 다 저장 프로시저 내에서 지원되지 않습니다.

저장 프로시저 내의 ROLLBACK 사용에 대한 자세한 내용은 트랜잭션 관리를 참조하십시오.

예제

다음 예에서는 테이블을 생성한 다음에 데이터가 테이블에 삽입되는 트랜잭션을 시작합니다. 그런 다음 ROLLBACK 명령으로 데이터 삽입을 롤백하여 테이블이 비어 있는 상태로 둡니다.

다음 명령을 실행하면 MOVIE_GROSS라는 예 테이블이 생성됩니다.

```
create table movie_gross( name varchar(30), gross bigint );
```

다음 명령 세트는 테이블에 2개의 데이터 행을 삽입하는 트랜잭션을 시작합니다.

```
begin;

insert into movie_gross values ( 'Raiders of the Lost Ark', 23400000);

insert into movie_gross values ( 'Star Wars', 10000000 );
```

다음으로, 아래 명령을 실행하면 데이터가 올바로 삽입되었음을 보여주기 위해 테이블에서 해당 데이터가 선택됩니다.

```
select * from movie_gross;
```

명령 출력에는 두 행 모두 올바로 삽입된 것으로 표시됩니다.

```
name           |  gross
-------------------------+----------
Raiders of the Lost Ark | 23400000
Star Wars               | 10000000
(2 rows)
```

이제는 다음 명령으로 트랜잭션이 시작된 지점으로 데이터 변경 내용을 롤백합니다.

```
rollback;
```

테이블에서 데이터를 선택하면 빈 테이블이 표시됩니다.

```
select * from movie_gross;

name | gross
------+-------
(0 rows)
```

SELECT

테이블, 뷰 및 사용자 정의 함수에서 행을 반환합니다.

> **Note**
>
> 단일 SQL 문의 최대 크기는 16MB입니다.

구문

```
[ WITH with_subquery [, ...] ]
SELECT
[ TOP number | [ ALL | DISTINCT ]
* | expression [ AS output_name ] [, ...] ]
[ FROM table_reference [, ...] ]
[ WHERE condition ]
[ [ START WITH expression ] CONNECT BY expression ]
[ GROUP BY expression [, ...] ]
```

```
[ HAVING condition ]
[ QUALIFY condition ]
[ { UNION | ALL | INTERSECT | EXCEPT | MINUS } query ]
[ ORDER BY expression [ ASC | DESC ] ]
[ LIMIT { number | ALL } ]
[ OFFSET start ]
```

주제

WITH 절

WITH 절은 쿼리에 있는 SELECT 목록에 선행하는 선택적 절입니다. WITH 절은 하나 이상의 common_table_expressions를 정의합니다. 각 공통 테이블 표현식(CTE) 은 뷰 정의와 유사한 임시 테이블을 정의합니다. FROM 절에서 이러한 임시 테이블을 참조할 수 있습니다. 임시 테이블은 자신이 속한 쿼리가 실행되는 동안에만 사용됩니다. WITH 절에 있는 각각의 CTE는 테이블 이름, 열 이름의 선택적 목록, 테이블로 평가되는 쿼리 표현식(SELECT 문)을 지정합니다. 임시 테이블 이름을 정의하는 동일한 쿼리 식의 FROM 절에서 임시 테이블 이름을 참조하는 경우 CTE는 재귀적입니다.

WITH 절 하위 쿼리는 단일 쿼리를 실행하는 내내 사용 가능한 테이블을 정의하는 효율적인 방법입니다. 모든 경우에 있어 SELECT 문의 본문에 하위 쿼리를 사용하여 같은 결과를 얻을 수 있지만, WITH 절 하위 쿼리는 더 간단하게 쓰고 읽을 수 있습니다. 가능한 경우 여러 번 참조되는 WITH 절 하위 쿼리는 공통 하위 표현식으로 최적화됩니다. 즉, WITH 하위 쿼리를 한 번 평가하고 그 결과를 재사용할 수 있습니다. (공통 하위 표현식은 WITH 절에 정의되는 하위 표현식으로 제한되지 않습니다.)

구문

```
[ WITH [RECURSIVE] common_table_expression [, common_table_expression , ...] ]
```

여기서 common_table_expression은 비재귀적이거나 재귀적일 수 있습니다. 다음은 비재귀 형식입니다.

```
CTE_table_name [ ( column_name [, ...] ) ] AS ( query )
```

다음은 common_table_expression의 재귀 형식입니다.

```
CTE_table_name (column_name [, ...] ) AS ( recursive_query )
```

파라미터

RECURSIVE

쿼리를 재귀 CTE로 식별하는 키워드입니다. WITH 절에 정의된 common_table_expression이 재귀적이면 이 키워드가 필요합니다. WITH 절에 여러 재귀 CTE가 포함된 경우에도 WITH 키워드 바로 다음에 RECURSIVE 키워드를 한 번만 지정할 수 있습니다. 일반적으로 재귀 CTE는 두 부분으로 구성된 UNION ALL 하위 쿼리입니다.

common_table_expression

FROM 절에서 참조할 수 있는 임시 테이블을 정의하고 해당 테이블이 속한 쿼리 실행 중에만 사용됩니다.

CTE_table_name

WITH 절 하위 쿼리의 결과를 정의하는 임시 테이블의 고유한 이름입니다. 단일 WITH 절 내에서 중복되는 이름을 사용할 수 없습니다. 각각의 하위 쿼리에는 FROM 절에서 참조될 수 있는 테이블 이름이 주어져야 합니다.

column_name

WITH 절 하위 쿼리에 대한 출력 열 이름의 목록으로, 쉼표로 구분됩니다. 지정되는 열 이름의 수는 하위 쿼리로 정의되는 열 개수보다 적거나 같아야 합니다. 비재귀 CTE의 경우 column_name 절은 옵션입니다. 재귀 CTE의 경우 column_name 목록은 필수입니다.

query

Amazon Redshift에서 지원하는 SELECT 쿼리입니다. SELECT 섹션을 참조하세요.

recursive_query

2개의 SELECT 하위 쿼리로 구성된 UNION ALL 쿼리:

- 첫 번째 SELECT 하위 쿼리에는 동일한 CTE_table_name에 대한 재귀 참조가 없습니다. 재귀의 초기 시드인 결과 집합을 반환합니다. 이 부분을 초기 멤버 또는 시드 멤버라고 합니다.
- 두 번째 SELECT 하위 쿼리는 FROM 절에서 동일한 CTE_table_name을 참조합니다. 이를 재귀 멤버라고합니다. recursive_query는 recursive_query를 종료하기 위한 WHERE 조건을 포함합니다.

사용 노트

다음 SQL 문에 WITH 절을 사용할 수 있습니다.

- SELECT
- SELECT INTO
- CREATE TABLE AS
- CREATE VIEW
- DECLARE
- EXPLAIN
- INSERT INTO...SELECT
- PREPARE
- UPDATE(WHERE 절 하위 쿼리 내. 하위 쿼리에서 재귀 CTE를 정의할 수 없습니다. 재귀 CTE는 UPDATE 절 앞에 와야 합니다.)
- DELETE

WITH 절을 포함한 쿼리의 FROM 절이 WITH 절로 정의되는 테이블 중 참조하지 않는 테이블이 있을 경우 WITH 절이 무시되고 쿼리가 정상적으로 실행됩니다.

WITH 절 하위 쿼리로 정의되는 테이블은 WITH 절이 시작하는 SELECT 쿼리의 범위에서만 참조될 수 있습니다. 예를 들어, SELECT 목록, WHERE 절 또는 HAVING 절에 있는 하위 쿼리의 FROM 절에서 그와 같은 테이블을 참조할 수 있습니다. 하위 쿼리에 WITH 절을 사용할 수 없고 기본 쿼리 또는 다른 하위 쿼리의 FROM 절에서 WITH 절의 테이블을 참조할 수 없습니다. 이 쿼리 패턴으로 인해 WITH 절 테이블에 대해 `relation table_name doesn't exist` 형식의 오류 메시지가 발생합니다.

WITH 절 하위 쿼리 내에서 다른 WITH 절을 지정할 수 없습니다.

WITH 절 하위 쿼리에 의해 정의되는 테이블에 대한 전방 참조를 할 수 없습니다. 예를 들어, 다음 쿼리는 테이블 W1의 정의에서 테이블 W2에 대한 전방 참조 때문에 오류를 반환합니다.

```
with w1 as (select * from w2), w2 as (select * from w1)
select * from sales;
ERROR:  relation "w2" does not exist
```

WITH 절 하위 쿼리는 SELECT INTO 문으로 구성되지 않을 수 있지만, SELECT INTO 문에 WITH 절을 사용할 수 있습니다.

재귀적인 공통 테이블 표현식

재귀 공통 테이블 표현식(CTE)은 자신을 참조하는 CTE입니다. 재귀 CTE는 직원과 관리자 간의 보고 관계를 보여주는 조직도와 같은 계층적 데이터를 쿼리하는 데 유용합니다. 예: 재귀 CTE 섹션을 참조하세요.

또 다른 일반적인 용도는 제품이 여러 구성 요소로 이루어지고 각 구성 요소 자체도 다른 구성 요소 또는 하위 어셈블리로 이루어진 다단계 BOM입니다.

재귀 쿼리의 두 번째 SELECT 하위 쿼리에 WHERE 절을 포함하여 재귀 깊이를 제한해야 합니다. 예시는 예: 재귀 CTE을 확인하세요. 그렇지 않으면 다음과 비슷한 오류가 발생할 수 있습니다.

- `Recursive CTE out of working buffers.`
- `Exceeded recursive CTE max rows limit, please add correct CTE termination predicates or change the max_recursion_rows parameter.`

> **Note**
>
> `max_recursion_rows`는 무한 재귀 루프를 방지하기 위해 재귀 CTE가 반환할 수 있는 최대 행 수를 설정하는 파라미터입니다. 이 값을 기본값보다 큰 값으로 변경하지 않는 것이 좋습니다. 이렇게 하면 쿼리의 무한 재귀 문제가 클러스터에서 과도한 공간을 차지하는 것을 방지할 수 있습니다.

재귀 CTE 결과에 대한 정렬 순서 및 제한을 지정할 수 있습니다. 재귀 CTE의 최종 결과에 group by 및 distinct 옵션을 포함할 수 있습니다.

하위 쿼리 내에 WITH RECURSIVE 절을 지정할 수 없습니다. recursive_query 멤버는 order by 또는 limit 절을 포함할 수 없습니다.

예시

다음 예에서는 WITH 절을 포함하는 쿼리로서 가능한 가장 간단한 사례를 보여줍니다. VENUECOPY라는 이름의 WITH 쿼리는 VENUE 테이블에서 모든 행을 선택합니다. 다음에는 기본 쿼리가 VENUECOPY에서 모든 행을 선택합니다. VENUECOPY 테이블은 이 쿼리의 지속 시간 동안에만 존재합니다.

```
with venuecopy as (select * from venue)
select * from venuecopy order by 1 limit 10;
```

```
 venueid |         venuename          |    venuecity    | venuestate | venueseats
---------+----------------------------+-----------------+------------+------------
1 | Toyota Park                | Bridgeview      | IL         |          0
2 | Columbus Crew Stadium      | Columbus        | OH         |          0
3 | RFK Stadium                | Washington      | DC         |          0
4 | CommunityAmerica Ballpark  | Kansas City     | KS         |          0
5 | Gillette Stadium           | Foxborough      | MA         |      68756
6 | New York Giants Stadium    | East Rutherford | NJ         |      80242
7 | BMO Field                  | Toronto         | ON         |          0
8 | The Home Depot Center      | Carson          | CA         |          0
9 | Dick's Sporting Goods Park | Commerce City   | CO         |          0
v     10 | Pizza Hut Park             | Frisco          | TX         |          0
(10 rows)
```

다음 예에서는 VENUE_SALES와 TOP_VENUES라는 두 테이블을 생성하는 WITH 절을 보여줍니다. 두 번째 WITH 절 테이블은 첫 번째 WITH 절 테이블에서 선택합니다. 다음에는, 기본 쿼리 블록의 WHERE 절이 TOP_VENUES 테이블을 포함하는 하위 쿼리를 포함합니다.

```
with venue_sales as
(select venuename, venuecity, sum(pricepaid) as venuename_sales
from sales, venue, event
where venue.venueid=event.venueid and event.eventid=sales.eventid
group by venuename, venuecity),

top_venues as
(select venuename
from venue_sales
where venuename_sales > 800000)

select venuename, venuecity, venuestate,
sum(qtysold) as venue_qty,
```

```
sum(pricepaid) as venue_sales
from sales, venue, event
where venue.venueid=event.venueid and event.eventid=sales.eventid
and venuename in(select venuename from top_venues)
group by venuename, venuecity, venuestate
order by venuename;
```

```
        venuename       |   venuecity   | venuestate | venue_qty | venue_sales
------------------------+---------------+------------+-----------+-------------
August Wilson Theatre   | New York City | NY         |      3187 |  1032156.00
Biltmore Theatre        | New York City | NY         |      2629 |   828981.00
Charles Playhouse       | Boston        | MA         |      2502 |   857031.00
Ethel Barrymore Theatre | New York City | NY         |      2828 |   891172.00
Eugene O'Neill Theatre  | New York City | NY         |      2488 |   828950.00
Greek Theatre           | Los Angeles   | CA         |      2445 |   838918.00
Helen Hayes Theatre     | New York City | NY         |      2948 |   978765.00
Hilton Theatre          | New York City | NY         |      2999 |   885686.00
Imperial Theatre        | New York City | NY         |      2702 |   877993.00
Lunt-Fontanne Theatre   | New York City | NY         |      3326 |  1115182.00
Majestic Theatre        | New York City | NY         |      2549 |   894275.00
Nederlander Theatre     | New York City | NY         |      2934 |   936312.00
Pasadena Playhouse      | Pasadena      | CA         |      2739 |   820435.00
Winter Garden Theatre   | New York City | NY         |      2838 |   939257.00
(14 rows)
```

다음 두 예에서는 WITH 절 하위 쿼리를 기반으로 테이블 참조의 범위에 대한 규칙을 보여줍니다. 첫 번째 쿼리가 실행되지만 두 번째 쿼리는 예상된 오류가 발생하며 실패합니다. 첫 번째 쿼리는 기본 쿼리의 SELECT 목록 내에 WITH 절 하위 쿼리가 있습니다. WITH 절(HOLIDAYS)에 의해 정의되는 테이블은 SELECT 목록에 있는 하위 쿼리의 FROM 절에 참조됩니다.

```
select caldate, sum(pricepaid) as daysales,
(with holidays as (select * from date where holiday ='t')
select sum(pricepaid)
from sales join holidays on sales.dateid=holidays.dateid
where caldate='2008-12-25') as dec25sales
from sales join date on sales.dateid=date.dateid
where caldate in('2008-12-25','2008-12-31')
group by caldate
order by caldate;

caldate   | daysales | dec25sales
----------+----------+------------
```

```
2008-12-25 | 70402.00 |   70402.00
2008-12-31 | 12678.00 |   70402.00
(2 rows)
```

기본 쿼리뿐 아니라 SELECT 목록 하위 쿼리에서 HOLIDAYS 테이블 참조를 시도하므로 두 번째 쿼리는 실패합니다. 기본 쿼리 참조는 범위를 벗어나는 주제입니다.

```
select caldate, sum(pricepaid) as daysales,
(with holidays as (select * from date where holiday ='t')
select sum(pricepaid)
from sales join holidays on sales.dateid=holidays.dateid
where caldate='2008-12-25') as dec25sales
from sales join holidays on sales.dateid=holidays.dateid
where caldate in('2008-12-25','2008-12-31')
group by caldate
order by caldate;

ERROR:  relation "holidays" does not exist
```

예: 재귀 CTE

다음은 John에게 직간접적으로 보고하는 직원을 반환하는 재귀 CTE의 예입니다. 재귀 쿼리에는 WHERE 절이 포함되어 있어 재귀 수준을 4단계 미만으로 제한합니다.

```
--create and populate the sample table
  create table employee (
  id int,
  name varchar (20),
  manager_id int
  );

  insert into employee(id, name, manager_id)  values
(100, 'Carlos', null),
(101, 'John', 100),
(102, 'Jorge', 101),
(103, 'Kwaku', 101),
(110, 'Liu', 101),
(106, 'Mateo', 102),
(110, 'Nikki', 103),
(104, 'Paulo', 103),
(105, 'Richard', 103),
(120, 'Saanvi', 104),
(200, 'Shirley', 104),
```

```
(201, 'Sofía', 102),
(205, 'Zhang', 104);

--run the recursive query
  with recursive john_org(id, name, manager_id, level) as
( select id, name, manager_id, 1 as level
  from employee
  where name = 'John'
  union all
  select e.id, e.name, e.manager_id, level + 1 as next_level
  from employee e, john_org j
  where e.manager_id = j.id and level < 4
  )
 select distinct id, name, manager_id from john_org order by manager_id;
```

다음은 쿼리 결과입니다.

```
   id        name      manager_id
  ------+-----------+--------------
   101    John           100
   102    Jorge          101
   103    Kwaku          101
   110    Liu            101
   201    Sofía          102
   106    Mateo          102
   110    Nikki          103
   104    Paulo          103
   105    Richard        103
   120    Saanvi         104
   200    Shirley        104
   205    Zhang          104
```

다음은 John의 부서의 조직도입니다.

SELECT 목록

주제

- 구문
- 파라미터
- 사용 노트

- 예시

SELECT 목록은 쿼리에서 반환하도록 하려는 열, 함수 및 표현식의 이름을 지정합니다. 목록은 쿼리의 출력을 나타냅니다.

SQL 함수에 대한 자세한 내용은 SQL 함수 참조 섹션을 참조하세요. 표현식에 대한 자세한 내용은 조건식 섹션을 참조하세요.

구문

```
SELECT
[ TOP number ]
[ ALL | DISTINCT ] * | expression [ AS column_alias ] [, ...]
```

파라미터

TOP number

TOP은 양의 정수를 인수로 취해 클라이언트로 반환되는 행의 수를 한정합니다. TOP 절을 이용한 동작은 LIMIT 절을 이용한 동작과 같습니다. 반환되는 행 수는 고정되지만 행 집합은 그렇지 않습니다. 일관된 행 집합을 반환하려면 ORDER BY 절과 함께 TOP 또는 LIMIT를 사용하십시오.

ALL

DISTINCT를 지정하지 않을 경우의 기본 동작을 정의하는 중복 키워드입니다. SELECT ALL *은 SELECT *와 동일한 의미입니다(즉, 모든 열에 대해 모든 행을 선택하고 중복 항목을 유지함).

DISTINCT

하나 이상의 열에서 일치하는 값을 바탕으로 결과 집합에서 중복된 행을 제거하는 옵션입니다.

> **Note**
>
> 애플리케이션이 잘못된 외래 키 또는 프라이머리 키를 허용하는 경우에는 쿼리가 잘못된 결과를 반환할 수 있습니다. 예를 들어 SELECT DISTINCT 쿼리는 프라이머리 키 열이 고유한 값만을 포함하지 않을 경우 중복 행을 반환할 수도 있습니다. 자세한 내용은 테이블 제약 조건 정의 단원을 참조하세요.

*(별표)

테이블의 전체 내용(모든 열과 모든 행)을 반환합니다.

expression

쿼리에서 참조하는 테이블에 존재하는 하나 이상의 열에서 형성되는 표현식입니다. 표현식은 SQL 함수를 포함할 수 있습니다. 예:

```
avg(datediff(day, listtime, saletime))
```

AS column_alias

최종 결과 집합에 사용될 열의 임시 이름입니다. AS 키워드는 옵션입니다. 예:

```
avg(datediff(day, listtime, saletime)) as avgwait
```

표현식에 대해 단순한 열 이름이 아닌 별칭을 지정하지 않을 경우 결과 집합은 그 열에 기본 이름을 적용합니다.

> **Note**
>
> 별칭은 대상 목록에 정의되는 즉시 인식됩니다. 동일한 대상 목록에서 별칭 뒤에 정의된 다른 표현식에 별칭을 사용할 수 있습니다. 다음 예는 이를 보여 줍니다.
>
> ```
> select clicks / impressions as probability, round(100 * probability, 1) as
>  percentage from raw_data;
> ```
>
> 측방 별칭 참조를 사용하면 동일한 대상 목록에서 더 복잡한 표현식을 작성할 때 별칭 처리된 표현식을 반복할 필요가 없다는 이점이 있습니다. Amazon Redshift에서는 이런 형식의 참조를 구문 분석할 때 이전에 정의된 별칭을 일렬로 정렬합니다. FROM 절에 이전에 별칭 처리된 표현식과 동일한 이름을 가진 열이 정의되어 있는 경우 FROM 절의 열에 우선 순위가 적용됩니다. 예를 들어 위의 쿼리에서 raw_data 테이블에 'probability' 열이 있는 경우 대상 목록의 두 번째 표현식에 있는 'probability'는 별칭 이름 'probability' 대신 이 열을 나타냅니다.

사용 노트

TOP은 SQL 확장으로, LIMIT 동작에 대한 대안을 제공합니다. 같은 쿼리에 TOP과 LIMIT를 사용할 수 없습니다.

예시

다음 예에서는 SALES 테이블의 행 10개를 반환합니다. 쿼리는 TOP 절을 사용하지만 ORDER BY 절이 지정되지 않았으므로, 여전히 예측할 수 없는 행 집합을 반환합니다.

```
select top 10 *
from sales;
```

다음 쿼리는 기능적으로는 동일하지만 TOP 절 대신 LIMIT 절을 사용합니다.

```
select *
from sales
limit 10;
```

다음 예에서는 TOP 절을 사용하여 SALES 테이블에서 최초 10개의 행을 반환하며, QTYSOLD 열을 기준으로 내림차순으로 정렬합니다.

```
select top 10 qtysold, sellerid
from sales
order by qtysold desc, sellerid;

qtysold | sellerid
--------+----------
8 |      518
8 |      520
8 |      574
8 |      718
8 |      868
8 |     2663
8 |     3396
8 |     3726
8 |     5250
8 |     6216
(10 rows)
```

다음 예에서는 QTYSOLD 열을 기준으로 정렬된 SALES 테이블에서 최초 2개의 QTYSOLD 및 SELLERID 값을 반환합니다.

```
select top 2 qtysold, sellerid
from sales
order by qtysold desc, sellerid;
```

```
qtysold | sellerid
--------+----------
8 |      518
8 |      520
(2 rows)
```

다음 예에서는 CATEGORY 테이블의 개별 범주 그룹 목록을 확인할 수 있습니다.

```
select distinct catgroup from category
order by 1;

catgroup
----------
Concerts
Shows
Sports
(3 rows)

--the same query, run without distinct
select catgroup from category
order by 1;

catgroup
----------
Concerts
Concerts
Concerts
Shows
Shows
Shows
Sports
Sports
Sports
Sports
Sports
(11 rows)
```

다음 예에서는 2008년 12월의 고유한 주 번호 집합을 반환합니다. DISTINCT 절을 사용하지 않으면 문은 행 31개를 반환하거나 해당 월의 날짜별로 행을 하나씩 반환합니다.

```
select distinct week, month, year
from date
```

```
where month='DEC' and year=2008
order by 1, 2, 3;

week | month | year
-----+-------+------
49 | DEC   | 2008
50 | DEC   | 2008
51 | DEC   | 2008
52 | DEC   | 2008
53 | DEC   | 2008
(5 rows)
```

FROM 절

쿼리의 FROM 절은 데이터가 선택되는 테이블 참조(테이블, 뷰, 하위 쿼리)를 나열합니다. 여러 개의 테이블 참조가 목록에 표시되는 경우 FROM 절 또는 WHERE 절에서 알맞은 구문을 사용하여 테이블을 조인해야 합니다. 조인 기준이 지정되지 않은 경우 시스템에서는 쿼리를 크로스 조인(데카르트 곱)으로 처리합니다.

주제

- 구문
- 파라미터
- 사용 노트
- PIVOT 및 UNPIVOT 예
- JOIN 예

구문

```
FROM table_reference [, ...]
```

여기서 table_reference는 다음 중 하나입니다.

```
with_subquery_table_name [ table_alias ]
table_name [ * ] [ table_alias ]
( subquery ) [ table_alias ]
table_reference [ NATURAL ] join_type table_reference
   [ ON join_condition | USING ( join_column [, ...] ) ]
table_reference PIVOT (
```

```
   aggregate(expr) [ [ AS ] aggregate_alias ]
   FOR column_name IN ( expression [ AS ] in_alias [, ...] )
) [ table_alias ]
table_reference UNPIVOT [ INCLUDE NULLS | EXCLUDE NULLS ] (
   value_column_name
   FOR name_column_name IN ( column_reference [ [ AS ]
   in_alias ] [, ...] )
) [ table_alias ]
UNPIVOT expression AS value_alias [ AT attribute_alias ]
```

선택적 table_alias를 사용하여 다음과 같이 테이블 및 복합 테이블 참조와 원하는 경우 해당 열에 임시 이름을 지정할 수 있습니다.

```
[ AS ] alias [ ( column_alias [, ...] ) ]
```

파라미터

with_subquery_table_name

WITH 절에서 하위 쿼리에 의해 정의되는 테이블입니다.

table_name

테이블 또는 뷰의 이름입니다.

별칭

테이블 또는 뷰의 임시 대체 이름입니다. 하위 쿼리에서 파생되는 테이블에 대해 별칭을 입력해야 합니다. 다른 테이블 참조에서 별칭은 옵션입니다. AS 키워드는 항상 옵션입니다. 테이블 별칭은 WHERE 절과 같이 쿼리의 다른 부분에 있는 테이블을 편리하게 식별하는 바로 가기의 역할을 합니다. 예:

```
select * from sales s, listing l
where s.listid=l.listid
```

column_alias

테이블 또는 뷰에 있는 열의 임시 대체 이름입니다.

subquery

테이블로 평가되는 쿼리 표현식입니다. 테이블은 쿼리의 지속 시간 동안만 존재하며 일반적으로 이름 또는 별칭이 주어집니다. 그러나 별칭이 필수는 아닙니다. 하위 쿼리에서 파생되는 테이블의 열

이름을 정의할 수도 있습니다. 하위 쿼리의 결과를 다른 테이블에 조인하고 쿼리의 다른 곳에서 열을 선택하거나 제한하려는 경우 열 별칭의 이름 지정이 중요합니다.

하위 쿼리는 ORDER BY 절을 포함할 수 있지만, LIMIT 또는 OFFSET 절도 지정하지 않으면 ORDER BY 절이 아무런 효과도 없을 수 있습니다.

NATURAL

두 테이블에서 조인 열로서 똑같이 명명된 열의 쌍을 전부 자동으로 사용하는 조인을 정의합니다. 명시적 조인 조건은 필요하지 않습니다. 예를 들어, CATEGORY 및 EVENT 테이블에 모두 CATID로 명명된 열이 있는 경우 이러한 테이블의 자연 조인은 CATID 열에 적용되는 조인입니다.

> **Note**
>
> NATURAL 조인이 지정되어 있지만 조인되는 테이블에 똑같은 이름의 열 쌍이 존재하지 않는 경우 쿼리는 기본적으로 크로스 조인이 됩니다.

join_type

다음과 같은 조인 유형 중 하나를 지정합니다.

- [INNER] JOIN
- LEFT [OUTER] JOIN
- RIGHT [OUTER] JOIN
- FULL [OUTER] JOIN
- CROSS JOIN

크로스 조인은 정규화되지 않은 조인으로, 두 테이블의 데카르트 곱을 반환합니다.

내부 및 외부 조인은 정규화된 조인입니다. 이런 조인은 FROM 절에서 ON 또는 USING 구문으로 암시적으로(자연 조인으로) 정규화되거나 WHERE 절 조건으로 암시적으로 정규화됩니다.

내부 조인은 조인 조건이나 조인 열의 목록을 기반으로 일치하는 행만 반환합니다. 외부 조인은 동등한 내부 조인이 반환하는 모든 행과 "왼쪽" 테이블, "오른쪽" 테이블 또는 두 테이블 모두에서 일치하지 않는 행을 반환합니다. 왼쪽 테이블은 처음에 목록으로 표시되는 테이블이고, 오른쪽 테이블은 두 번째로 목록으로 표시되는 테이블입니다. 일치하지 않는 행은 출력 열의 간격을 채우기 위해 NULL 값을 포함합니다.

ON join_condition

조인 열이 ON 키워드 뒤에 나오는 조건으로 규정되는 조인 사양의 유형입니다. 예:

```
sales join listing
on sales.listid=listing.listid and sales.eventid=listing.eventid
```

USING (join_column [, ...])

조인 열이 괄호 안에 묶여 표시되는 조인 사양의 유형입니다. 여러 개의 조인 열이 지정되어 있는 경우 이런 열은 쉼표로 구분됩니다. USING 키워드는 목록에 선행해야 합니다. 예:

```
sales join listing
using (listid,eventid)
```

PIVOT

읽기 쉬운 형식으로 테이블형 데이터를 표시하기 위해 행에서 열로 출력을 회전합니다. 출력은 여러 열에 가로로 표시됩니다. PIVOT은 집계 표현식을 사용하여 출력 형식을 지정하는 집계가 있는 GROUP BY 쿼리와 유사합니다. 그러나 GROUP BY와 달리 결과는 행 대신 열로 반환됩니다.

PIVOT 및 UNPIVOT을 사용하여 쿼리하는 방법을 보여주는 예는 PIVOT 및 UNPIVOT 예 섹션을 참조하세요.

UNPIVOT

UNPIVOT을 사용하여 열을 행으로 교체 - 이 연산자는 입력 테이블 또는 쿼리 결과의 결과 열을 행으로 변환하여 출력을 더 읽기 쉽게 만듭니다. UNPIVOT은 입력 열의 데이터를 이름 열과 값 열의 두 결과 열로 결합합니다. 이름 열에는 입력의 열 이름이 행 항목으로 포함됩니다. 값 열에는 집계 결과와 같은 입력 열의 값이 포함됩니다. 예를 들어 다양한 카테고리의 항목 수입니다.

UNPIVOT(SUPER)을 사용한 객체 피벗 해제 - 객체 피벗을 해제할 수 있습니다. 여기서 표현식은 다른 FROM 절 항목을 참조하는 SUPER 표현식입니다. 자세한 내용은 객체 피벗 해제 단원을 참조하십시오. JSON 형식 데이터와 같은 반정형 데이터를 쿼리하는 방법을 보여주는 예도 있습니다.

사용 노트

조인 열은 비교 가능한 데이터 형식이 있어야 합니다.

NATURAL 또는 USING 조인은 중간 결과 집합에 조인 열의 각 쌍 중 하나만 유지합니다.

ON 구문이 있는 조인은 중간 결과 집합에 두 조인 열을 모두 유지합니다.

또한 WITH 절 단원도 참조하세요.

PIVOT 및 UNPIVOT 예

PIVOT 및 UNPIVOT은 각각 쿼리 출력을 행에서 열로, 열에서 행으로 회전하는 FROM 절의 파라미터입니다. 테이블 쿼리 결과를 읽기 쉬운 형식으로 나타냅니다. 다음 예제에서는 테스트 데이터와 쿼리를 사용하여 사용 방법을 보여줍니다.

이러한 파라미터 및 기타 파라미터에 대한 자세한 내용은 FROM 절 단원을 참조하세요.

PIVOT 예

샘플 테이블과 데이터를 설정하고 이를 사용하여 후속 예제 쿼리를 실행합니다.

```
CREATE TABLE part (
    partname varchar,
    manufacturer varchar,
    quality int,
    price decimal(12, 2)
);

INSERT INTO part VALUES ('prop', 'local parts co', 2, 10.00);
INSERT INTO part VALUES ('prop', 'big parts co', NULL, 9.00);
INSERT INTO part VALUES ('prop', 'small parts co', 1, 12.00);

INSERT INTO part VALUES ('rudder', 'local parts co', 1, 2.50);
INSERT INTO part VALUES ('rudder', 'big parts co', 2, 3.75);
INSERT INTO part VALUES ('rudder', 'small parts co', NULL, 1.90);

INSERT INTO part VALUES ('wing', 'local parts co', NULL, 7.50);
INSERT INTO part VALUES ('wing', 'big parts co', 1, 15.20);
INSERT INTO part VALUES ('wing', 'small parts co', NULL, 11.80);
```

`price`에 대한 AVG 집계가 있는 `partname`에 대한 PIVOT.

```
SELECT *
FROM (SELECT partname, price FROM part) PIVOT (
    AVG(price) FOR partname IN ('prop', 'rudder', 'wing')
);
```

쿼리 결과는 다음과 같이 출력됩니다.

```
  prop   |  rudder  |  wing
---------+----------+---------
```

```
 10.33   | 2.71     | 11.50
```

이전 예에서 결과는 열로 변환됩니다. 다음 예는 평균 가격을 열이 아닌 행으로 반환하는 GROUP BY 쿼리를 보여줍니다.

```
SELECT partname, avg(price)
FROM (SELECT partname, price FROM part)
WHERE partname IN ('prop', 'rudder', 'wing')
GROUP BY partname;
```

쿼리 결과는 다음과 같이 출력됩니다.

```
 partname |  avg
----------+-------
 prop     | 10.33
 rudder   |  2.71
 wing     | 11.50
```

암시적 열로 manufacturer가 있는 PIVOT 예.

```
SELECT *
FROM (SELECT quality, manufacturer FROM part) PIVOT (
    count(*) FOR quality IN (1, 2, NULL)
);
```

쿼리 결과는 다음과 같이 출력됩니다.

```
 manufacturer      | 1  | 2  | null
-------------------+----+----+------
 local parts co    | 1  | 1  |  1
 big parts co      | 1  | 1  |  1
 small parts co    | 1  | 0  |  2
```

PIVOT 정의에서 참조되지 않는 입력 테이블 열은 암시적으로 결과 테이블에 추가됩니다. 이는 앞의 예에서 manufacturer 열의 경우입니다. 이 예는 또한 NULL이 IN 연산자에 유효한 값임을 보여줍니다.

위의 예에서 PIVOT은 GROUP BY가 포함된 다음 쿼리와 유사한 정보를 반환합니다. 차이점은 PIVOT이 열 2와 제조업체 small parts co에 대해 값 0을 반환한다는 것입니다. GROUP BY 쿼리에는 해당 행이 없습니다. 대부분의 경우 행에 지정된 열에 대한 입력 데이터가 없으면 PIVOT은 NULL을 삽입합니다. 그러나 count 집계는 NULL을 반환하지 않고 0이 기본값입니다.

```
SELECT manufacturer, quality, count(*)
FROM (SELECT quality, manufacturer FROM part)
WHERE quality IN (1, 2) OR quality IS NULL
GROUP BY manufacturer, quality
ORDER BY manufacturer;
```

쿼리 결과는 다음과 같이 출력됩니다.

```
 manufacturer        | quality | count
---------------------+---------+-------
 big parts co        |         |     1
 big parts co        |       2 |     1
 big parts co        |       1 |     1
 local parts co      |       2 |     1
 local parts co      |       1 |     1
 local parts co      |         |     1
 small parts co      |       1 |     1
 small parts co      |         |     2
```

PIVOT 연산자는 집계 표현식과 IN 연산자의 각 값에 대한 선택적 별칭을 허용합니다. 별칭을 사용하여 열 이름을 사용자 지정합니다. 집계 별칭이 없는 경우 IN 목록 별칭이 사용됩니다. 그렇지 않으면 이름을 구분하기 위해 집계 별칭이 밑줄과 함께 열 이름에 추가됩니다.

```
SELECT *
FROM (SELECT quality, manufacturer FROM part) PIVOT (
    count(*) AS count FOR quality IN (1 AS high, 2 AS low, NULL AS na)
);
```

쿼리 결과는 다음과 같이 출력됩니다.

```
 manufacturer      | high_count  | low_count | na_count
-------------------+-------------+-----------+----------
 local parts co    |           1 |         1 |        1
 big parts co      |           1 |         1 |        1
 small parts co    |           1 |         0 |        2
```

다음 샘플 테이블과 데이터를 설정하고 이를 사용하여 후속 예제 쿼리를 실행합니다. 데이터는 여러 호텔의 예약 날짜를 나타냅니다.

```
CREATE TABLE bookings (
    booking_id int,
```

```
    hotel_code char(8),
    booking_date date,
    price decimal(12, 2)
);

INSERT INTO bookings VALUES (1, 'FOREST_L', '02/01/2023', 75.12);
INSERT INTO bookings VALUES (2, 'FOREST_L', '02/02/2023', 75.00);
INSERT INTO bookings VALUES (3, 'FOREST_L', '02/04/2023', 85.54);

INSERT INTO bookings VALUES (4, 'FOREST_L', '02/08/2023', 75.00);
INSERT INTO bookings VALUES (5, 'FOREST_L', '02/11/2023', 75.00);
INSERT INTO bookings VALUES (6, 'FOREST_L', '02/14/2023', 90.00);

INSERT INTO bookings VALUES (7, 'FOREST_L', '02/21/2023', 60.00);
INSERT INTO bookings VALUES (8, 'FOREST_L', '02/22/2023', 85.00);
INSERT INTO bookings VALUES (9, 'FOREST_L', '02/27/2023', 90.00);

INSERT INTO bookings VALUES (10, 'DESERT_S', '02/01/2023', 98.00);
INSERT INTO bookings VALUES (11, 'DESERT_S', '02/02/2023', 75.00);
INSERT INTO bookings VALUES (12, 'DESERT_S', '02/04/2023', 85.00);

INSERT INTO bookings VALUES (13, 'DESERT_S', '02/05/2023', 75.00);
INSERT INTO bookings VALUES (14, 'DESERT_S', '02/06/2023', 34.00);
INSERT INTO bookings VALUES (15, 'DESERT_S', '02/09/2023', 85.00);

INSERT INTO bookings VALUES (16, 'DESERT_S', '02/12/2023', 23.00);
INSERT INTO bookings VALUES (17, 'DESERT_S', '02/13/2023', 76.00);
INSERT INTO bookings VALUES (18, 'DESERT_S', '02/14/2023', 85.00);

INSERT INTO bookings VALUES (19, 'OCEAN_WV', '02/01/2023', 98.00);
INSERT INTO bookings VALUES (20, 'OCEAN_WV', '02/02/2023', 75.00);
INSERT INTO bookings VALUES (21, 'OCEAN_WV', '02/04/2023', 85.00);

INSERT INTO bookings VALUES (22, 'OCEAN_WV', '02/06/2023', 75.00);
INSERT INTO bookings VALUES (23, 'OCEAN_WV', '02/09/2023', 34.00);
INSERT INTO bookings VALUES (24, 'OCEAN_WV', '02/12/2023', 85.00);

INSERT INTO bookings VALUES (25, 'OCEAN_WV', '02/13/2023', 23.00);
INSERT INTO bookings VALUES (26, 'OCEAN_WV', '02/14/2023', 76.00);
INSERT INTO bookings VALUES (27, 'OCEAN_WV', '02/16/2023', 85.00);

INSERT INTO bookings VALUES (28, 'CITY_BLD', '02/01/2023', 98.00);
INSERT INTO bookings VALUES (29, 'CITY_BLD', '02/02/2023', 75.00);
INSERT INTO bookings VALUES (30, 'CITY_BLD', '02/04/2023', 85.00);
```

```
INSERT INTO bookings VALUES (31, 'CITY_BLD', '02/12/2023', 75.00);
INSERT INTO bookings VALUES (32, 'CITY_BLD', '02/13/2023', 34.00);
INSERT INTO bookings VALUES (33, 'CITY_BLD', '02/17/2023', 85.00);

INSERT INTO bookings VALUES (34, 'CITY_BLD', '02/22/2023', 23.00);
INSERT INTO bookings VALUES (35, 'CITY_BLD', '02/23/2023', 76.00);
INSERT INTO bookings VALUES (36, 'CITY_BLD', '02/24/2023', 85.00);
```

이 샘플 쿼리에서는 예약 기록을 집계하여 각 주의 합계를 산출합니다. 각 주의 종료일은 열 이름이 됩니다.

```
SELECT * FROM
    (SELECT
       booking_id,
       (date_trunc('week', booking_date::date) + '5 days'::interval)::date as enddate,
       hotel_code AS "hotel code"
FROM bookings
) PIVOT (
    count(booking_id) FOR enddate IN ('2023-02-04','2023-02-11','2023-02-18')
);
```

쿼리 결과는 다음과 같이 출력됩니다.

```
 hotel code | 2023-02-04  | 2023-02-11 | 2023-02-18
------------+-------------+------------+----------
 FOREST_L   |           3 |          2 |        1
 DESERT_S   |           4 |          3 |        2
 OCEAN_WV   |           3 |          3 |        3
 CITY_BLD   |           3 |          1 |        2
```

Amazon Redshift는 여러 열에서 피벗하기 위한 CROSSTAB을 지원하지 않습니다. 그러나 다음과 같은 쿼리를 사용하여 PIVOT을 사용한 집계와 유사한 방식으로 행 데이터를 열로 변경할 수 있습니다. 이전 예와 동일한 예약 샘플 데이터를 사용합니다.

```
SELECT
  booking_date,
  MAX(CASE WHEN hotel_code = 'FOREST_L' THEN 'forest is booked' ELSE '' END) AS
 FOREST_L,
  MAX(CASE WHEN hotel_code = 'DESERT_S' THEN 'desert is booked' ELSE '' END) AS
 DESERT_S,
```

```
  MAX(CASE WHEN hotel_code = 'OCEAN_WV' THEN 'ocean is booked' ELSE '' END)  AS
 OCEAN_WV
FROM bookings
GROUP BY booking_date
ORDER BY booking_date asc;
```

샘플 쿼리를 실행하면 예약된 호텔을 나타내는 짧은 문구 옆에 예약 날짜가 표시됩니다.

```
 booking_date  | forest_l         | desert_s         | ocean_wv
---------------+------------------+------------------+--------------------
 2023-02-01    | forest is booked | desert is booked |  ocean is booked
 2023-02-02    | forest is booked | desert is booked |  ocean is booked
 2023-02-04    | forest is booked | desert is booked |  ocean is booked
 2023-02-05    |                  | desert is booked |
 2023-02-06    |                  | desert is booked |
```

다음은 PIVOT에 대한 사용 참고 사항입니다.

- PIVOT은 테이블, 하위 쿼리 및 공통 테이블 표현식(CTE)에 적용할 수 있습니다. JOIN 표현식, 재귀 CTE, PIVOT 또는 UNPIVOT 표현식에는 PIVOT을 적용할 수 없습니다. 또한 SUPER 비중첩 표현식과 Redshift Spectrum 중첩 테이블은 지원되지 않습니다.
- PIVOT에서는 COUNT, SUM, MIN, MAX 및 AVG 집계 함수를 지원합니다.
- PIVOT 집계 표현식은 지원되는 집계 함수의 호출이어야 합니다. 집계 위의 복잡한 표현식은 지원되지 않습니다. 집계 인수는 PIVOT 입력 테이블 이외의 테이블에 대한 참조를 포함할 수 없습니다. 상위 쿼리에 대한 상관 관계가 있는 참조도 지원되지 않습니다. 집계 인수에는 하위 쿼리가 포함될 수 있습니다. 이들은 내부적으로 또는 PIVOT 입력 테이블에서 상관될 수 있습니다.
- PIVOT IN 목록 값은 열 참조 또는 하위 쿼리일 수 없습니다. 각 값은 FOR 열 참조와 호환되는 유형이어야 합니다.
- IN 목록 값에 별칭이 없으면 PIVOT은 기본 열 이름을 생성합니다. 'abc' 또는 5와 같은 상수 IN 값의 경우 기본 열 이름은 상수 자체입니다. 복잡한 표현식의 경우 열 이름은 ?column?와 같은 표준 Amazon Redshift 기본 이름입니다.

UNPIVOT 예

샘플 데이터를 설정하고 이를 사용하여 후속 예제를 실행합니다.

```
CREATE TABLE count_by_color (quality varchar, red int, green int, blue int);

INSERT INTO count_by_color VALUES ('high', 15, 20, 7);
```

```
INSERT INTO count_by_color VALUES ('normal', 35, NULL, 40);
INSERT INTO count_by_color VALUES ('low', 10, 23, NULL);
```

빨간색, 녹색 및 파란색 입력 열의 UNPIVOT.

```
SELECT *
FROM (SELECT red, green, blue FROM count_by_color) UNPIVOT (
    cnt FOR color IN (red, green, blue)
);
```

쿼리 결과는 다음과 같이 출력됩니다.

```
 color | cnt
-------+-----
 red   |  15
 red   |  35
 red   |  10
 green |  20
 green |  23
 blue  |   7
 blue  |  40
```

기본적으로 입력 열의 NULL 값은 건너뛰고 결과 행을 생성하지 않습니다.

다음은 INCLUDE NULLS가 있는 UNPIVOT의 예입니다.

```
SELECT *
FROM (
    SELECT red, green, blue
    FROM count_by_color
) UNPIVOT INCLUDE NULLS (
    cnt FOR color IN (red, green, blue)
);
```

결과 출력값은 다음과 같습니다.

```
 color | cnt
-------+-----
 red   |  15
 red   |  35
 red   |  10
 green |  20
```

```
 green |
 green |  23
 blue  |   7
 blue  |  40
 blue  |
```

INCLUDING NULLS 파라미터가 설정되면 NULL 입력 값이 결과 행을 생성합니다.

암시적 열로 quality가 있는 The following query shows UNPIVOT.

```
SELECT *
FROM count_by_color UNPIVOT (
    cnt FOR color IN (red, green, blue)
);
```

쿼리 결과는 다음과 같이 출력됩니다.

```
 quality | color | cnt
---------+-------+-----
 high    | red   |  15
 normal  | red   |  35
 low     | red   |  10
 high    | green |  20
 low     | green |  23
 high    | blue  |   7
 normal  | blue  |  40
```

UNPIVOT 정의에서 참조되지 않는 입력 테이블의 열은 암시적으로 결과 테이블에 추가됩니다. 이 예에서는 quality 열의 경우입니다.

다음 예는 IN 목록의 값에 대한 별칭이 있는 UNPIVOT을 보여줍니다.

```
SELECT *
FROM count_by_color UNPIVOT (
    cnt FOR color IN (red AS r, green AS g, blue AS b)
);
```

이전 쿼리 결과는 다음과 같이 출력됩니다.

```
 quality | color | cnt
---------+-------+-----
```

```
high    | r     |  15
normal  | r     |  35
low     | r     |  10
high    | g     |  20
low     | g     |  23
high    | b     |   7
normal  | b     |  40
```

UNPIVOT 연산자는 각 IN 목록 값에서 선택적 별칭을 허용합니다. 각 별칭을 통해 각 value 열의 데이터를 사용자 지정할 수 있습니다.

다음은 UNPIVOT에 대한 사용 참고 사항입니다.

- UNPIVOT은 테이블, 하위 쿼리 및 공통 테이블 표현식(CTE)에 적용할 수 있습니다. JOIN 표현식, 재귀 CTE, UNPIVOT 또는 UNPIVOT 표현식에는 PIVOT을 적용할 수 없습니다. 또한 SUPER 비중첩 표현식과 Redshift Spectrum 중첩 테이블은 지원되지 않습니다.
- UNPIVOT IN 목록에는 입력 테이블 열 참조만 포함되어야 합니다. IN 목록 열에는 모두 호환되는 공통 유형이 있어야 합니다. UNPIVOT 값 열에는 이러한 공통 유형이 있습니다. UNPIVOT 이름 열은 VARCHAR유형입니다.
- IN 목록 값에 별칭이 없으면 UNPIVOT은 열 이름을 기본값으로 사용합니다.

JOIN 예

SQL JOIN 절은 공통 필드를 기반으로 두 개 이상의 테이블에서 데이터를 결합하는 데 사용됩니다. 지정된 조인 메서드에 따라 결과가 변경될 수도 있고 변경되지 않을 수도 있습니다. JOIN 절의 구문에 대한 자세한 내용은 파라미터 섹션을 참조하세요.

다음 예에서는 TICKIT 샘플 데이터의 데이터를 사용합니다. 데이터베이스 스키마에 대한 자세한 내용은 샘플 데이터베이스 섹션을 참조하세요. 샘플 데이터를 로드하는 방법을 알아보려면 Amazon Redshift 시작 안내서의 데이터 로드 단원을 참조하세요.

다음 쿼리는 LISTING 테이블과 SALES 테이블 간의 내부 조인(JOIN 키워드 사용 안 함)이며, 여기서 LISTING 테이블의 LISTID는 1에서 5 사이입니다. 이 쿼리는 LISTING 테이블(왼쪽 테이블) 및 SALES 테이블(오른쪽 테이블)에 있는 LISTID 열 값과 일치합니다. 결과는 LISTID 1, 4, 5가 조건과 일치한다는 것을 보여줍니다.

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from listing, sales
where listing.listid = sales.listid
and listing.listid between 1 and 5
```

```
group by 1
order by 1;

listid | price  |  comm
-------+--------+--------
     1 | 728.00 | 109.20
     4 |  76.00 |  11.40
     5 | 525.00 |  78.75
```

다음 쿼리는 왼쪽 외부 조인입니다. 왼쪽 및 오른쪽 외부 조인은 다른 테이블에서 일치 항목이 발견되지 않을 때 조인된 테이블 중 하나에서 값을 유지합니다. 왼쪽 및 오른쪽 테이블은 구문에 나열되는 첫 번째 및 두 번째 테이블입니다. NULL 값은 결과 집합의 "간격"을 채우는 데 사용됩니다. 이 쿼리는 LISTING 테이블(왼쪽 테이블) 및 SALES 테이블(오른쪽 테이블)에 있는 LISTID 열 값과 일치합니다. 결과는 LISTID 2 및 3이 어떤 판매로도 이어지지 않았음을 보여줍니다.

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from listing left outer join sales on sales.listid = listing.listid
where listing.listid between 1 and 5
group by 1
order by 1;

listid | price  |  comm
-------+--------+--------
     1 | 728.00 | 109.20
     2 | NULL   | NULL
     3 | NULL   | NULL
     4 |  76.00 |  11.40
     5 | 525.00 |  78.75
```

다음 쿼리는 오른쪽 외부 조인입니다. 이 쿼리는 LISTING 테이블(왼쪽 테이블) 및 SALES 테이블(오른쪽 테이블)에 있는 LISTID 열 값과 일치합니다. 결과는 LISTID 1, 4, 5가 조건과 일치한다는 것을 보여줍니다.

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from listing right outer join sales on sales.listid = listing.listid
where listing.listid between 1 and 5
group by 1
order by 1;

listid | price  |  comm
-------+--------+--------
     1 | 728.00 | 109.20
```

```
     4 |  76.00 |  11.40
     5 | 525.00 |  78.75
```

다음 쿼리는 전체 조인입니다. 전체 조인은 다른 테이블에서 일치 항목이 발견되지 않을 때 조인된 테이블의 값을 유지합니다. 왼쪽 및 오른쪽 테이블은 구문에 나열되는 첫 번째 및 두 번째 테이블입니다. NULL 값은 결과 집합의 "간격"을 채우는 데 사용됩니다. 이 쿼리는 LISTING 테이블(왼쪽 테이블) 및 SALES 테이블(오른쪽 테이블)에 있는 LISTID 열 값과 일치합니다. 결과는 LISTID 2 및 3이 어떤 판매로도 이어지지 않았음을 보여줍니다.

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from listing full join sales on sales.listid = listing.listid
where listing.listid between 1 and 5
group by 1
order by 1;

listid | price  |  comm
-------+--------+--------
     1 | 728.00 | 109.20
     2 | NULL   | NULL
     3 | NULL   | NULL
     4 |  76.00 |  11.40
     5 | 525.00 |  78.75
```

다음 쿼리는 전체 조인입니다. 이 쿼리는 LISTING 테이블(왼쪽 테이블) 및 SALES 테이블(오른쪽 테이블)에 있는 LISTID 열 값과 일치합니다. 판매로 이어지지 않는 행(LISTID 2 및 3)만 결과에 있습니다.

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from listing full join sales on sales.listid = listing.listid
where listing.listid between 1 and 5
and (listing.listid IS NULL or sales.listid IS NULL)
group by 1
order by 1;

listid | price  |  comm
-------+--------+--------
     2 | NULL   | NULL
     3 | NULL   | NULL
```

다음 예는 ON 절과의 내부 조인입니다. 이 경우 NULL 행은 반환되지 않습니다.

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
```

```
from sales join listing
on sales.listid=listing.listid and sales.eventid=listing.eventid
where listing.listid between 1 and 5
group by 1
order by 1;

listid | price  |  comm
-------+--------+--------
     1 | 728.00 | 109.20
     4 |  76.00 |  11.40
     5 | 525.00 |  78.75
```

다음 쿼리는 결과를 제한하는 술어가 있는, LISTING 테이블과 SALES 테이블의 교차 조인 또는 데카르트 조인입니다. 이 쿼리는 SALES 테이블과 LISTING 테이블의 LISTID(두 테이블 모두 LISTID 1, 2, 3, 4, 5) 열 값과 일치합니다. 결과는 20개의 행이 조건과 일치한다는 것을 보여줍니다.

```
select sales.listid as sales_listid, listing.listid as listing_listid
from sales cross join listing
where sales.listid between 1 and 5
and listing.listid between 1 and 5
order by 1,2;

sales_listid | listing_listid
-------------+---------------
1            | 1
1            | 2
1            | 3
1            | 4
1            | 5
4            | 1
4            | 2
4            | 3
4            | 4
4            | 5
5            | 1
5            | 1
5            | 2
5            | 2
5            | 3
5            | 3
5            | 4
5            | 4
5            | 5
```

```
5            | 5
```

다음 예는 두 테이블 간의 자연 조인입니다. 이 경우 listid, sellerid, eventid, dateid 열은 두 테이블 모두에서 동일한 이름과 데이터 형식을 가지므로 조인 열로 사용됩니다. 결과는 5개 행으로 제한됩니다.

```
select listid, sellerid, eventid, dateid, numtickets
from listing natural join sales
order by 1
limit 5;

listid | sellerid  | eventid | dateid | numtickets
-------+-----------+---------+--------+-----------
113    | 29704     | 4699    | 2075   | 22
115    | 39115     | 3513    | 2062   | 14
116    | 43314     | 8675    | 1910   | 28
118    | 6079      | 1611    | 1862   | 9
163    | 24880     | 8253    | 1888   | 14
```

다음 예는 USING 절을 사용한 두 테이블 간의 조인입니다. 이 경우 listid 및 eventid 열이 조인 열로 사용됩니다. 결과는 5개 행으로 제한됩니다.

```
select listid, listing.sellerid, eventid, listing.dateid, numtickets
from listing join sales
using (listid, eventid)
order by 1
limit 5;

listid | sellerid | eventid | dateid | numtickets
-------+----------+---------+--------+-----------
1      | 36861    | 7872    | 1850   | 10
4      | 8117     | 4337    | 1970   | 8
5      | 1616     | 8647    | 1963   | 4
5      | 1616     | 8647    | 1963   | 4
6      | 47402    | 8240    | 2053   | 18
```

다음 쿼리는 FROM 절에 있는 두 하위 쿼리의 내부 조인입니다. 다음 쿼리는 다양한 범주의 이벤트(콘서트 및 쇼)에 대해 판매된 티켓과 판매되지 않은 티켓의 수를 찾습니다. 이러한 FROM 절 하위 쿼리는 table 하위 쿼리로서, 여러 개의 열과 행을 반환할 수 있습니다.

```
select catgroup1, sold, unsold
from
(select catgroup, sum(qtysold) as sold
```

```
from category c, event e, sales s
where c.catid = e.catid and e.eventid = s.eventid
group by catgroup) as a(catgroup1, sold)
join
(select catgroup, sum(numtickets)-sum(qtysold) as unsold
from category c, event e, sales s, listing l
where c.catid = e.catid and e.eventid = s.eventid
and s.listid = l.listid
group by catgroup) as b(catgroup2, unsold)

on a.catgroup1 = b.catgroup2
order by 1;

catgroup1 |  sold  | unsold
----------+--------+--------
Concerts  | 195444 |1067199
Shows     | 149905 | 817736
```

WHERE 절

WHERE 절은 테이블을 조인하거나 테이블의 열에 조건자를 적용하는 조건을 포함합니다. WHERE 절 또는 FROM 절에서 알맞은 구문을 사용하여 테이블을 내부 조인할 수 있습니다. FROM 절에는 외부 조인 기준을 지정해야 합니다.

구문

```
[ WHERE condition ]
```

condition

테이블 열에서 조인 조건 또는 조건자 같이, 부울 결과를 포함한 임의의 검색 조건입니다. 다음 예는 유효한 조인 조건입니다.

```
sales.listid=listing.listid
sales.listid<>listing.listid
```

다음 예는 테이블의 열에 유효한 조건입니다.

```
catgroup like 'S%'
venueseats between 20000 and 50000
eventname in('Jersey Boys','Spamalot')
year=2008
```

```
length(catdesc)>25
date_part(month, caldate)=6
```

조건은 단순하거나 복잡할 수 있는데, 복잡한 조건의 경우 괄호를 사용하여 논리 단위를 분리할 수 있습니다. 다음 예에서는 조인 조건이 괄호로 묶여 있습니다.

```
where (category.catid=event.catid) and category.catid in(6,7,8)
```

사용 노트

select list 표현식을 참조하기 위해 WHERE 절의 별칭을 사용할 수 있습니다.

WHERE 절에서 집계 함수의 결과를 제한할 수 없습니다. 결과를 제한하려면 HAVING 절을 사용하십시오.

WHERE 절에서 제한되는 열은 FROM 절의 테이블 참조로부터 파생해야 합니다.

예

다음 쿼리는 SALES 및 EVENT 테이블에 대한 조인 조건, EVENTNAME 열의 조건자, STARTTIME 열의 두 조건자를 비롯한 다양한 WHERE 절 제한 사항의 조합을 사용합니다.

```
select eventname, starttime, pricepaid/qtysold as costperticket, qtysold
from sales, event
where sales.eventid = event.eventid
and eventname='Hannah Montana'
and date_part(quarter, starttime) in(1,2)
and date_part(year, starttime) = 2008
order by 3 desc, 4, 2, 1 limit 10;

eventname    |      starttime      |   costperticket   | qtysold
---------------+---------------------+-------------------+---------
Hannah Montana | 2008-06-07 14:00:00 |     1706.00000000 |       2
Hannah Montana | 2008-05-01 19:00:00 |     1658.00000000 |       2
Hannah Montana | 2008-06-07 14:00:00 |     1479.00000000 |       1
Hannah Montana | 2008-06-07 14:00:00 |     1479.00000000 |       3
Hannah Montana | 2008-06-07 14:00:00 |     1163.00000000 |       1
Hannah Montana | 2008-06-07 14:00:00 |     1163.00000000 |       2
Hannah Montana | 2008-06-07 14:00:00 |     1163.00000000 |       4
Hannah Montana | 2008-05-01 19:00:00 |      497.00000000 |       1
Hannah Montana | 2008-05-01 19:00:00 |      497.00000000 |       2
Hannah Montana | 2008-05-01 19:00:00 |      497.00000000 |       4
(10 rows)
```

WHERE 절의 Oracle 스타일 외부 조인

Oracle 호환성을 위해 Amazon Redshift는 WHERE 절 조인 조건에서 Oracle 외부 조인 연산자(+)를 지원합니다. 이 연산자는 외부 조인 조건을 정의하는 데만 사용하는 연산자이므로, 다른 컨텍스트에서는 사용하지 마십시오. 이 연산자를 달리 사용하면 대부분의 경우에는 자동으로 무시됩니다.

외부 조인은 동등한 내부 조인이 반환하는 모든 행과 한 테이블 또는 두 테이블 모두에서 일치하지 않는 행을 반환합니다. FROM 절에서 왼쪽, 오른쪽 및 전체 외부 조인을 지정할 수 있습니다. WHERE 절에서 왼쪽 및 오른쪽 외부 조인만 지정할 수 있습니다.

외부 조인 테이블 TABLE1 및 TABLE2를 지정하고 TABLE1(왼쪽 외부 조인)에서 일치하지 않는 행을 반환하려면 FROM 절에 `TABLE1 LEFT OUTER JOIN TABLE2`를 지정하거나 WHERE 절의 TABLE2에서 모든 조인 열에 (+) 연산자를 적용합니다. TABLE2에 일치하는 행이 없는 TABLE1의 모든 행에 대해, 쿼리의 결과는 TABLE2의 열을 포함한 모든 select list 표현식을 위한 null을 포함합니다.

TABLE1에 일치하는 행이 없는 TABLE2의 모든 행에 대해 동일한 동작을 생성하려면 FROM 절에 `TABLE1 RIGHT OUTER JOIN TABLE2`를 지정하거나 WHERE 절의 TABLE1에서 모든 조인 열에 (+) 연산자를 적용합니다.

기본 구문

```
[ WHERE {
[ table1.column1 = table2.column1(+) ]
[ table1.column1(+) = table2.column1 ]
}
```

첫 번째 조건은 다음과 동등합니다.

```
from table1 left outer join table2
on table1.column1=table2.column1
```

두 번째 조건은 다음과 동등합니다.

```
from table1 right outer join table2
on table1.column1=table2.column1
```

> **Note**
>
> 여기에 표시된 구문은 한 쌍의 조인 열에 대해 간단한 동등 조인 케이스를 포함합니다. 하지만 다른 유형의 비교 조건과 여러 쌍의 조인 열 역시 유효합니다.

예를 들어, 다음 WHERE 절은 두 쌍의 열에 대해 외부 조인을 정의합니다. 두 조건에서 모두 (+) 연산자를 같은 테이블에 연결해야 합니다.

```
where table1.col1 > table2.col1(+)
and table1.col2 = table2.col2(+)
```

사용 노트

가능하면 WHERE 절에 (+) 연산자 대신 표준 FROM 절 OUTER JOIN 구문을 사용하십시오. (+) 연산자를 포함하는 쿼리는 다음 규칙에 따릅니다.

- WHERE 절에는 (+) 연산자만 사용할 수 있고, 테이블 또는 뷰의 열에 대한 참조에만 사용할 수 있습니다.
- 표현식에는 (+) 연산자를 적용할 수 없습니다. 하지만 표현식은 (+) 연산자를 사용하는 열을 포함할 수 있습니다. 예를 들어 다음과 같은 조인 조건은 구문 오류를 반환합니다.

  ```
  event.eventid*10(+)=category.catid
  ```

  그러나 다음 조인 조건이 유효합니다.

  ```
  event.eventid(+)*10=category.catid
  ```

- FROM 절 조인 구문도 포함하는 쿼리 블록에 (+) 연산자를 사용할 수 없습니다.
- 여러 조인 조건에 걸쳐 두 테이블이 조인되는 경우 이런 조건 전부에 (+) 연산자를 사용하거나 아무런 조건에도 이 연산자를 사용하면 안 됩니다. 혼합 구문 스타일의 조인은 따로 경고 없이 내부 조인으로 실행됩니다.
- 외부 쿼리의 테이블을 내부 쿼리에서 생성되는 테이블과 조인하는 경우 (+) 연산자는 외부 조인을 생성하지 않습니다.
- (+) 연산자를 사용하여 테이블을 그 자체에 외부 조인하려면 FROM 절에서 테이블 별칭을 정의하고 이런 별칭을 조인 조건에서 참조해야 합니다.

  ```
  select count(*)
  from event a, event b
  where a.eventid(+)=b.catid;

  count
  -------
  8798
  ```

```
(1 row)
```

- (+) 연산자를 포함하는 조인 조건을 OR 조건 또는 IN 조건과 결합할 수 없습니다. 예:

```
select count(*) from sales, listing
where sales.listid(+)=listing.listid or sales.salesid=0;
ERROR:  Outer join operator (+) not allowed in operand of OR or IN.
```

- 2개보다 많은 테이블을 외부 조인하는 WHERE 절에서는 (+) 연산자를 주어진 테이블에 한 번만 적용할 수 있습니다. 다음 예에서는 2개의 연속 조인에서 (+) 연산자로 SALES 테이블을 참조할 수 없습니다.

```
select count(*) from sales, listing, event
where sales.listid(+)=listing.listid and sales.dateid(+)=date.dateid;
ERROR:  A table may be outer joined to at most one other table.
```

- WHERE 절 외부 조인 조건이 TABLE2의 열을 상수와 비교하는 경우 그 열에 (+) 연산자를 적용합니다. 이 연산자를 포함하지 않으면 제한된 열을 위한 null을 포함하는 TABLE1의 외부 조인된 행이 제거됩니다. 아래 예시 섹션을 참조하세요.

예시

다음 조인 쿼리는 LISTID 열에 대한 SALES 및 LISTING 테이블의 왼쪽 외부 조인을 지정합니다.

```
select count(*)
from sales, listing
where sales.listid = listing.listid(+);

count
--------
172456
(1 row)
```

위와 동등한 다음의 쿼리는 결과는 같지만 FROM 절 조인 구문을 사용합니다.

```
select count(*)
from sales left outer join listing on sales.listid = listing.listid;

count
--------
172456
```

```
(1 row)
```

SALES 테이블은 모든 목록이 판매로 이어지는 것은 아니므로 LISTING 테이블의 모든 목록에 대한 레코드를 포함하지는 않습니다. 다음 쿼리는 SALES와 LISTING을 외부 조인하고 SALES 테이블이 주어진 목록 ID에 대해 아무런 판매도 보고하지 않을 때도 LISTING에서 행을 반환합니다. SALES 테이블에서 파생되는 PRICE 및 COMM 열은 일치하지 않는 행에 대한 결과 집합에 null이 있습니다.

```
select listing.listid, sum(pricepaid) as price,
sum(commission) as comm
from listing, sales
where sales.listid(+) = listing.listid and listing.listid between 1 and 5
group by 1 order by 1;

listid | price  |  comm
--------+--------+--------
1 | 728.00 | 109.20
2 |        |
3 |        |
4 |  76.00 |  11.40
5 | 525.00 |  78.75
(5 rows)
```

WHERE 절 조인 연산자가 사용될 때 FROM 절에서 테이블의 순서는 중요하지 않습니다.

WHERE 절에서 더 복잡한 외부 조인 조건의 예시는 조건이 두 테이블 열 사이의 비교 and 상수와의 비교로 구성되는 경우입니다.

```
where category.catid=event.catid(+) and eventid(+)=796;
```

(+) 연산자는 두 곳에 사용되는데, 첫째로는 테이블 사이의 동등성 비교에 사용되고 둘째로는 EVENTID 열에 대한 비교 조건에 사용됩니다. 이 구문의 결과는 EVENTID에 대한 제한이 평가될 때 외부 조인된 행이 보존되는 것입니다. EVENTID 제한에서 (+) 연산자를 제거하는 경우 쿼리는 이 제한을 외부 조인 조건의 일부가 아니라 필터로 취급합니다. 그러면 EVENTID에 대해 null을 포함하는 외부 조인된 행이 결과 집합에서 제거됩니다.

다음은 이 동작을 설명하는 완전한 쿼리입니다.

```
select catname, catgroup, eventid
from category, event
where category.catid=event.catid(+) and eventid(+)=796;
```

```
catname | catgroup | eventid
-----------+----------+---------
Classical | Concerts |
Jazz | Concerts |
MLB | Sports   |
MLS | Sports   |
Musicals | Shows    | 796
NBA | Sports   |
NFL | Sports   |
NHL | Sports   |
Opera | Shows    |
Plays | Shows    |
Pop | Concerts |
(11 rows)
```

FROM 절 구문을 사용하는 동등한 쿼리는 다음과 같습니다.

```
select catname, catgroup, eventid
from category left join event
on category.catid=event.catid and eventid=796;
```

이 쿼리의 WHERE 절 버전에서 두 번째 (+) 연산자를 제거하는 경우 1개의 행만 반환합니다 (eventid=796인 행).

```
select catname, catgroup, eventid
from category, event
where category.catid=event.catid(+) and eventid=796;

catname | catgroup | eventid
-----------+----------+---------
Musicals | Shows    | 796
(1 row)
```

GROUP BY 절

GROUP BY 절은 쿼리에 대한 그룹화 열을 식별합니다. 쿼리가 SUM, AVG 및 COUNT와 같은 표준 함수로 집계를 계산할 때 그룹화 열을 선언해야 합니다. 자세한 내용은 집계 함수 단원을 참조하십시오.

구문

```
GROUP BY group_by_clause [, ...]
```

```
group_by_clause := {
    expr |
    GROUPING SETS ( () | group_by_clause [, ...] ) |
    ROLLUP ( expr [, ...] ) |
    CUBE ( expr [, ...] )
    }
```

Parameters

expr

열 또는 표현식의 목록은 쿼리의 선택 목록에 있는 비집계 표현식의 목록과 일치해야 합니다. 예를 들어, 다음과 같이 간단한 쿼리를 생각해 보세요.

```
select listid, eventid, sum(pricepaid) as revenue,
count(qtysold) as numtix
from sales
group by listid, eventid
order by 3, 4, 2, 1
limit 5;

listid | eventid | revenue | numtix
-------+---------+---------+--------
89397  |      47 |   20.00 |      1
106590 |      76 |   20.00 |      1
124683 |     393 |   20.00 |      1
103037 |     403 |   20.00 |      1
147685 |     429 |   20.00 |      1
(5 rows)
```

이 쿼리에서 선택 목록은 2개의 집계 표현식으로 구성됩니다. 첫 번째 표현식은 SUM 함수를 사용하고 두 번째 표현식은 COUNT 함수를 사용합니다. 나머지 두 개의 열 LISTID 및 EVENTID를 그룹화 열로 선언해야 합니다.

GROUP BY 절에서 표현식은 서수를 사용하여 선택 목록을 참조할 수도 있습니다. 예를 들어, 이전 예는 다음과 같이 줄일 수도 있습니다.

```
select listid, eventid, sum(pricepaid) as revenue,
count(qtysold) as numtix
from sales
group by 1,2
order by 3, 4, 2, 1
```

```
limit 5;

listid | eventid | revenue | numtix
-------+---------+---------+--------
89397  |      47 |   20.00 |      1
106590 |      76 |   20.00 |      1
124683 |     393 |   20.00 |      1
103037 |     403 |   20.00 |      1
147685 |     429 |   20.00 |      1
(5 rows)
```

GROUPING SETS/ROLLUP/CUBE

집계 확장 GROUPING SETS, ROLLUP 및 CUBE를 사용하여 단일 문에서 여러 GROUP BY 작업을 수행할 수 있습니다. 집계 확장 및 관련 기능에 대한 자세한 내용은 집계 확장을 참조하세요.

집계 확장

Amazon Redshift는 단일 문에서 여러 GROUP BY 작업을 수행하는 집계 확장을 지원합니다.

집계 확장의 예시에서는 전자 회사의 판매 데이터가 들어 있는`orders` 테이블을 사용합니다. 다음을 사용하여 `orders`를 생성할 수 있습니다.

```
CREATE TABLE ORDERS (
    ID INT,
    PRODUCT CHAR(20),
    CATEGORY CHAR(20),
    PRE_OWNED CHAR(1),
    COST DECIMAL
);

INSERT INTO ORDERS VALUES
    (0, 'laptop',       'computers',    'T', 1000),
    (1, 'smartphone',   'cellphones',   'T', 800),
    (2, 'smartphone',   'cellphones',   'T', 810),
    (3, 'laptop',       'computers',    'F', 1050),
    (4, 'mouse',        'computers',    'F', 50);
```

GROUPING SETS

단일 명령문에서 하나 이상의 그룹화 집합을 계산합니다. 그룹화 집합은 쿼리의 결과 집합을 그룹화할 수 있는 0개 이상의 열 집합인 단일 GROUP BY 절의 집합입니다. 집합을 그룹화하여 그룹화하는 것은

서로 다른 열로 그룹화된 하나의 결과 집합에서 UNION ALL 쿼리를 실행하는 것과 같습니다. 예를 들어, GROUP BY GROUPING SETS((a), (b))는 GROUP BY a UNION ALL GROUP BY b와 동일합니다.

다음 예에서는 제품 카테고리와 판매된 제품 종류에 따라 그룹화된 주문 테이블 제품의 비용을 반환합니다.

```
SELECT category, product, sum(cost) as total
FROM orders
GROUP BY GROUPING SETS(category, product);

       category       |       product        | total
----------------------+----------------------+-------
 computers            |                      |  2100
 cellphones           |                      |  1610
                      | laptop               |  2050
                      | smartphone           |  1610
                      | mouse                |    50

(5 rows)
```

ROLLUP

이전 열이 후속 열의 부모로 간주되는 계층 구조를 가정합니다. ROLLUP은 제공된 열을 기준으로 데이터를 그룹화하여 그룹화된 행 외에 그룹화 열의 모든 수준에서 총계를 나타내는 추가 소계 행을 반환합니다. 예를 들어 GROUP BY ROLLUP ((a), (b) 를 사용하여 b가 a의 하위 섹션이라고 가정하면서 먼저 a로 그룹화된 다음 b로 그룹화된 결과 집합을 반환할 수 있습니다. ROLLUP은 또한 열을 그룹화하지 않고 전체 결과 집합이 있는 행을 반환합니다.

GROUP BY ROLLUP((a), (b))는 GROUP BY GROUPING SETS((a,b), (a), ())와 같습니다.

다음 예는 먼저 범주별로 그룹화된 주문 테이블의 제품 비용을 반환한 다음 제품을 범주의 하위 부문으로 사용하여 반환합니다.

```
SELECT category, product, sum(cost) as total
FROM orders
GROUP BY ROLLUP(category, product) ORDER BY 1,2;

       category       |       product        | total
----------------------+----------------------+-------
 cellphones           | smartphone           |  1610
 cellphones           |                      |  1610
```

```
 computers            | laptop               |  2050
 computers            | mouse                |    50
 computers            |                      |  2100
                      |                      |  3710
(6 rows)
```

CUBE

제공된 열을 기준으로 데이터를 그룹화하여 그룹화된 행 외에 그룹화 열의 모든 수준에서 합계를 나타내는 추가 소계 행을 반환합니다. CUBE는 ROLLUP과 동일한 행을 반환하는 동시에 ROLLUP에서 다루지 않는 그룹화 열의 모든 조합에 대해 소계 행을 추가합니다. 예를 들어 GROUP BY CUBE ((a), (b)를 사용하여 b가 a의 하위 섹션이고 그 다음 b만으로 그룹화된 결과 집합을 반환할 수 있습니다. CUBE는 또한 열을 그룹화하지 않고 전체 결과 집합이 있는 행을 반환합니다.

GROUP BY CUBE((a), (b))는 GROUP BY GROUPING SETS((a, b), (a), (b), ())와 같습니다.

다음 예는 먼저 범주별로 그룹화된 주문 테이블의 제품 비용을 반환한 다음 제품을 범주의 하위 부문으로 사용하여 반환합니다. ROLLUP에 대한 앞의 예와 달리 문은 그룹화 열의 모든 조합에 대한 결과를 반환합니다.

```
SELECT category, product, sum(cost) as total
FROM orders
GROUP BY CUBE(category, product) ORDER BY 1,2;

       category       |       product        | total
----------------------+----------------------+-------
 cellphones           | smartphone           |  1610
 cellphones           |                      |  1610
 computers            | laptop               |  2050
 computers            | mouse                |    50
 computers            |                      |  2100
                      | laptop               |  2050
                      | mouse                |    50
                      | smartphone           |  1610
                      |                      |  3710
(9 rows)
```

GROUPING/GROUPING_ID 함수

ROLLUP 및 CUBE는 소계 행을 나타내기 위해 결과 집합에 NULL 값을 추가합니다. 예를 들어 GROUP BY ROLLUP((a), (b))는 b 그룹화 열에서 값이 NULL인 하나 이상의 행을 반환하여 그룹화 열

에 있는 필드의 소계임을 나타냅니다. 이러한 NULL 값은 반환 튜플의 형식을 충족하는 데만 사용됩니다.

NULL 값 자체를 저장하는 관계에서 ROLLUP 및 CUBE와 함께 GROUP BY 작업을 실행하면 동일한 그룹화 열이 있는 것으로 보이는 행이 있는 결과 집합이 생성될 수 있습니다. 이전 예제로 돌아가서 b 그룹화 열에 저장된 NULL 값이 포함된 경우 GROUP BY ROLLUP ((a), (b) 은 b 그룹화 열에서 소계가 아닌 값이 NULL인 행을 반환합니다.

ROLLUP 및 CUBE에 의해 생성된 NULL 값과 테이블 자체에 저장된 NULL 값을 구별하기 위해 GROUPING 함수 또는 별칭 GROUPING_ID를 사용할 수 있습니다. GROUPING은 단일 그룹화 집합을 인수로 사용하고 결과 집합의 각 행에 대해 해당 위치의 그룹화 열에 해당하는 0 또는 1비트 값을 반환한 다음 해당 값을 정수로 변환합니다. 해당 위치의 값이 집계 확장에 의해 생성된 NULL 값인 경우 GROUPING은 1을 반환합니다. 이 함수는 저장된 NULL 값을 비롯해 다른 모든 값에 대해 0을 반환합니다.

예를 들어 GROUPING(category, product)은 해당 행의 그룹화 열 값에 따라 지정된 행에 대해 다음 값을 반환할 수 있습니다. 이 예제의 목적을 위해 테이블의 모든 NULL 값은 집계 확장에 의해 생성된 NULL 값입니다.

범주 열	제품 열	GROUPING 함수 비트 값	십진수 값
Null이 아님	Null이 아님	00	0
Null이 아님	NULL	01	1
NULL	Null이 아님	10	2
NULL	NULL	11	3

GROUPING 함수는 쿼리의 SELECT 목록 부분에 다음 형식으로 나타납니다.

```
SELECT ... [GROUPING( expr )...] ...
  GROUP BY ... {CUBE | ROLLUP| GROUPING SETS} ( expr ) ...
```

다음 예제는 CUBE에 대한 이전 예제와 동일하지만 그룹화 집합에 대한 GROUPING 함수가 추가되었습니

```
SELECT category, product,
       GROUPING(category) as grouping0,
       GROUPING(product) as grouping1,
       GROUPING(category, product) as grouping2,
       sum(cost) as total
FROM orders
GROUP BY CUBE(category, product) ORDER BY 3,1,2;

       category       |       product        | grouping0 | grouping1 | grouping2 |
 total
----------------------+----------------------+-----------+-----------+-----------
+-------
 cellphones           | smartphone           |         0 |         0 |         0 |
 1610
 cellphones           |                      |         0 |         1 |         1 |
 1610
 computers            | laptop               |         0 |         0 |         0 |
 2050
 computers            | mouse                |         0 |         0 |         0 |
 50
 computers            |                      |         0 |         1 |         1 |
 2100
                      | laptop               |         1 |         0 |         2 |
 2050
                      | mouse                |         1 |         0 |         2 |
 50
                      | smartphone           |         1 |         0 |         2 |
 1610
                      |                      |         1 |         1 |         3 |
 3710
(9 rows)
```

부분 ROLLUP 및 CUBE

소계의 일부만으로 ROLLUP 및 CUBE 작업을 실행할 수 있습니다.

부분 ROLLUP 및 CUBE 작업의 구문은 다음과 같습니다.

```
GROUP BY expr1, { ROLLUP | CUBE }(expr2, [, ...])
```

여기에서 GROUP BY 절은 expr2 이후의 수준에서 소계 행만 생성합니다.

다음 예제는 주문 테이블에 대한 부분 ROLLUP 및 CUBE 작업을 보여줍니다. 먼저 제품이 중고인지 여부에 따라 그룹화한 다음 범주 및 제품 열에서 ROLLUP 및 CUBE를 실행합니다.

```
SELECT pre_owned, category, product,
       GROUPING(category, product, pre_owned) as group_id,
       sum(cost) as total
FROM orders
GROUP BY pre_owned, ROLLUP(category, product) ORDER BY 4,1,2,3;

 pre_owned |       category       |       product        | group_id | total
-----------+----------------------+----------------------+----------+-------
 F         | computers            | laptop               |        0 |  1050
 F         | computers            | mouse                |        0 |    50
 T         | cellphones           | smartphone           |        0 |  1610
 T         | computers            | laptop               |        0 |  1000
 F         | computers            |                      |        2 |  1100
 T         | cellphones           |                      |        2 |  1610
 T         | computers            |                      |        2 |  1000
 F         |                      |                      |        6 |  1100
 T         |                      |                      |        6 |  2610
(9 rows)

SELECT pre_owned, category, product,
       GROUPING(category, product, pre_owned) as group_id,
       sum(cost) as total
FROM orders
GROUP BY pre_owned, CUBE(category, product) ORDER BY 4,1,2,3;

 pre_owned |       category       |       product        | group_id | total
-----------+----------------------+----------------------+----------+-------
 F         | computers            | laptop               |        0 |  1050
 F         | computers            | mouse                |        0 |    50
 T         | cellphones           | smartphone           |        0 |  1610
 T         | computers            | laptop               |        0 |  1000
 F         | computers            |                      |        2 |  1100
 T         | cellphones           |                      |        2 |  1610
 T         | computers            |                      |        2 |  1000
 F         |                      | laptop               |        4 |  1050
 F         |                      | mouse                |        4 |    50
 T         |                      | laptop               |        4 |  1000
 T         |                      | smartphone           |        4 |  1610
 F         |                      |                      |        6 |  1100
 T         |                      |                      |        6 |  2610
```

```
(13 rows)
```

사전 소유 열은 ROLLUP 및 CUBE 작업에 포함되지 않으므로 다른 모든 행을 포함하는 총계 행이 없습니다.

연결된 그룹화

여러 GROUPING SETS/ROLLUP/CUBE 절을 연결하여 서로 다른 수준의 소계를 계산할 수 있습니다. 연결된 그룹화는 제공된 그룹화 집합의 데카르트 곱을 반환합니다.

GROUPING SETS/ROLLUP/CUBE 절을 연결하는 구문은 다음과 같습니다.

```
GROUP BY {ROLLUP|CUBE|GROUPING SETS}(expr1[, ...]),
         {ROLLUP|CUBE|GROUPING SETS}(expr1[, ...])[, ...]
```

다음 예제를 통해 서로 연결된 소규모 그룹화가 어떻게 큰 최종 결과 집합을 생성할 수 있는지 확인해 보세요.

```
SELECT pre_owned, category, product,
       GROUPING(category, product, pre_owned) as group_id,
       sum(cost) as total
FROM orders
GROUP BY CUBE(category, product), GROUPING SETS(pre_owned, ())
ORDER BY 4,1,2,3;

 pre_owned |       category       |       product        | group_id | total
-----------+----------------------+----------------------+----------+-------
 F         | computers            | laptop               |        0 |  1050
 F         | computers            | mouse                |        0 |    50
 T         | cellphones           | smartphone           |        0 |  1610
 T         | computers            | laptop               |        0 |  1000
           | cellphones           | smartphone           |        1 |  1610
           | computers            | laptop               |        1 |  2050
           | computers            | mouse                |        1 |    50
 F         | computers            |                      |        2 |  1100
 T         | cellphones           |                      |        2 |  1610
 T         | computers            |                      |        2 |  1000
           | cellphones           |                      |        3 |  1610
           | computers            |                      |        3 |  2100
 F         |                      | laptop               |        4 |  1050
 F         |                      | mouse                |        4 |    50
 T         |                      | laptop               |        4 |  1000
 T         |                      | smartphone           |        4 |  1610
```

```
           |                      | laptop               |        5 |  2050
           |                      | mouse                |        5 |    50
           |                      | smartphone           |        5 |  1610
 F         |                      |                      |        6 |  1100
 T         |                      |                      |        6 |  2610
           |                      |                      |        7 |  3710
(22 rows)
```

중첩된 그룹화

GROUPING SETS/ROLLUP/CUBE 작업을 GROUPING SETS expr로 사용하여 중첩된 그룹화를 형성할 수 있습니다. 중첩된 GROUPING SETS 내부의 하위 그룹화가 평면화됩니다.

중첩된 그룹화의 구문은 다음과 같습니다.

```
GROUP BY GROUPING SETS({ROLLUP|CUBE|GROUPING SETS}(expr[, ...])[, ...])
```

다음 예제를 살펴보세요.

```
SELECT category, product, pre_owned,
       GROUPING(category, product, pre_owned) as group_id,
       sum(cost) as total
FROM orders
GROUP BY GROUPING SETS(ROLLUP(category), CUBE(product, pre_owned))
ORDER BY 4,1,2,3;

       category       |       product        | pre_owned | group_id | total
----------------------+----------------------+-----------+----------+-------
 cellphones           |                      |           |        3 |  1610
 computers            |                      |           |        3 |  2100
                      | laptop               | F         |        4 |  1050
                      | laptop               | T         |        4 |  1000
                      | mouse                | F         |        4 |    50
                      | smartphone           | T         |        4 |  1610
                      | laptop               |           |        5 |  2050
                      | mouse                |           |        5 |    50
                      | smartphone           |           |        5 |  1610
                      |                      | F         |        6 |  1100
                      |                      | T         |        6 |  2610
                      |                      |           |        7 |  3710
                      |                      |           |        7 |  3710
(13 rows)
```

ROLLUP(category) 및 CUBE(product, pre_owned) 모두 그룹화 집합()을 포함하므로 총계를 나타내는 행이 중복됩니다.

사용 노트

- GROUP BY 절은 최대 64개의 그룹화 세트를 지원합니다. ROLLUP 및 CUBE 또는 GROUPING SETS, ROLLUP 및 CUBE의 일부 조합의 경우 이 제한은 포함된 그룹화 집합 수에 적용됩니다. 예를 들어 GROUP BY CUBE((a), (b))는 2가 아닌 4개의 그룹화 집합으로 계산됩니다.
- 집계 확장을 사용하는 경우 상수를 그룹화 열로 사용할 수 없습니다.
- 중복된 열이 포함된 그룹화 집합은 만들 수 없습니다.

HAVING 절

HAVING 절은 쿼리가 반환하는 중간 그룹화 결과 집합에 조건을 적용합니다.

구문

```
[ HAVING condition ]
```

예를 들어, SUM 함수의 결과를 제한할 수 있습니다.

```
having sum(pricepaid) >10000
```

모든 WHERE 절 조건이 적용되고 GROUP BY 작업이 완료된 후 HAVING 조건이 적용됩니다.

조건 자체는 WHERE 절 조건과 같은 형식을 취합니다.

사용 노트

- HAVING 절 조건에서 참조되는 열은 그룹화 열이거나 집계 함수의 결과를 참조하는 열이어야 합니다.
- HAVING 절에서 다음을 지정할 수는 없습니다.
  - 선택 목록 항목을 참조하는 서수. GROUP BY 및 ORDER BY 절만이 서수를 허용합니다.

예시

다음 쿼리는 이름을 기준으로 모든 이벤트에 대한 총 티켓 판매액을 계산한 다음, 총 판매액이 $800,000 미만인 이벤트를 제거합니다. HAVING 조건은 선택 목록에서 집계 함수의 결과에 적용됩니다. `sum(pricepaid)`.

```
select eventname, sum(pricepaid)
from sales join event on sales.eventid = event.eventid
group by 1
having sum(pricepaid) > 800000
order by 2 desc, 1;

eventname        |    sum
-----------------+-----------
Mamma Mia!       | 1135454.00
Spring Awakening |  972855.00
The Country Girl |  910563.00
Macbeth          |  862580.00
Jersey Boys      |  811877.00
Legally Blonde   |  804583.00
```

다음 쿼리는 비슷한 결과 집합을 계산합니다. 하지만 이 경우에는 HAVING 조건이 선택 목록에 지정되지 않은 집계에 적용됩니다(sum(qtysold)). 티켓이 2,000장보다 많이 팔리지 않은 이벤트가 최종 결과에서 제거됩니다.

```
select eventname, sum(pricepaid)
from sales join event on sales.eventid = event.eventid
group by 1
having sum(qtysold) >2000
order by 2 desc, 1;

eventname        |    sum
-----------------+-----------
Mamma Mia!       | 1135454.00
Spring Awakening |  972855.00
The Country Girl |  910563.00
Macbeth          |  862580.00
Jersey Boys      |  811877.00
Legally Blonde   |  804583.00
Chicago          |  790993.00
Spamalot         |  714307.00
```

다음 쿼리는 이름을 기준으로 모든 이벤트에 대한 총 티켓 판매액을 계산한 다음, 총 판매액이 $800,000 미만인 이벤트를 제거합니다. HAVING 조건은 선택 목록에서 sum(pricepaid)에 별칭 pp를 사용하여 집계 함수의 결과에 적용됩니다.

```
select eventname, sum(pricepaid) as pp
```

```
from sales join event on sales.eventid = event.eventid
group by 1
having pp > 800000
order by 2 desc, 1;

eventname        |    pp
-----------------+-----------
Mamma Mia!       | 1135454.00
Spring Awakening |  972855.00
The Country Girl |  910563.00
Macbeth          |  862580.00
Jersey Boys      |  811877.00
Legally Blonde   |  804583.00
```

QUALIFY 절

QUALIFY 절은 사용자가 지정한 검색 조건에 따라 이전에 계산된 윈도우 함수의 결과를 필터링합니다. 이 절을 사용하면 하위 쿼리를 사용하지 않고 윈도우 함수의 결과에 필터링 조건을 적용할 수 있습니다.

이는 조건을 적용하여 WHERE 절의 행을 추가로 필터링하는 HAVING 절과 유사합니다. QUALIFY와 HAVING의 차이점은 QUALIFY 절의 필터링된 결과가 데이터에 대해 윈도우 함수를 실행한 결과를 기반으로 할 수 있다는 것입니다. 한 쿼리에서 QUALIFY 절과 HAVING 절을 모두 사용할 수 있습니다.

구문

```
QUALIFY condition
```

> **Note**
>
> FROM 절 바로 뒤에 QUALIFY 절을 사용하는 경우 FROM 관계 이름에는 QUALIFY 절 앞에 별칭이 지정되어 있어야 합니다.

예시

이 섹션의 예에서는 아래 샘플 데이터를 사용합니다.

```
create table store_sales (ss_sold_date date, ss_sold_time time,
               ss_item text, ss_sales_price float);
```

```
insert into store_sales values ('2022-01-01', '09:00:00', 'Product 1', 100.0),
                               ('2022-01-01', '11:00:00', 'Product 2', 500.0),
                               ('2022-01-01', '15:00:00', 'Product 3', 20.0),
                               ('2022-01-01', '17:00:00', 'Product 4', 1000.0),
                               ('2022-01-01', '18:00:00', 'Product 5', 30.0),
                               ('2022-01-02', '10:00:00', 'Product 6', 5000.0),
                               ('2022-01-02', '16:00:00', 'Product 7', 5.0);
```

다음 예는 매일 12:00 이후에 판매된 가장 비싼 품목 두 개를 찾는 방법을 보여줍니다.

```
SELECT *
FROM store_sales ss
WHERE ss_sold_time > time '12:00:00'
QUALIFY row_number()
OVER (PARTITION BY ss_sold_date ORDER BY ss_sales_price DESC) <= 2


 ss_sold_date | ss_sold_time |  ss_item  | ss_sales_price
--------------+--------------+-----------+----------------
 2022-01-01   | 17:00:00     | Product 4 |           1000
 2022-01-01   | 18:00:00     | Product 5 |             30
 2022-01-02   | 16:00:00     | Product 7 |              5
```

그런 다음 매일 마지막으로 판매된 품목을 찾을 수 있습니다.

```
SELECT *
FROM store_sales ss
QUALIFY last_value(ss_item)
OVER (PARTITION BY ss_sold_date ORDER BY ss_sold_time ASC
      ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) = ss_item;

ss_sold_date | ss_sold_time |  ss_item  | ss_sales_price
--------------+--------------+-----------+----------------
 2022-01-01   | 18:00:00     | Product 5 |             30
 2022-01-02   | 16:00:00     | Product 7 |              5
```

다음 예에서는 이전 쿼리와 동일한 레코드, 즉 매일 마지막으로 판매된 품목을 반환하지만 QUALIFY 절을 사용하지 않습니다.

```
SELECT * FROM (
  SELECT *,
  last_value(ss_item)
```

```
  OVER (PARTITION BY ss_sold_date ORDER BY ss_sold_time ASC
        ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) ss_last_item
  FROM store_sales ss
)
WHERE ss_last_item = ss_item;

 ss_sold_date | ss_sold_time |  ss_item  | ss_sales_price | ss_last_item
--------------+--------------+-----------+----------------+--------------
 2022-01-02   | 16:00:00     | Product 7 |              5 | Product 7
 2022-01-01   | 18:00:00     | Product 5 |             30 | Product 5
```

UNION, INTERSECT 및 EXCEPT

주제

UNION, INTERSECT 및 EXCEPT 설정 연산자는 별개의 두 쿼리 표현식의 결과를 비교 및 병합하는 데 사용됩니다. 예를 들어, 웹사이트의 어떤 사용자가 구매자인 동시에 판매자인지 알고 싶지만 이런 사용자들의 사용자 이름이 별개의 열이나 테이블에 저장되어 있는 경우 이러한 두 가지 사용자 유형의 교집합을 찾을 수 있습니다. 어떤 웹사이트 사용자가 구매자이고 판매자는 아닌지 알고 싶으면 EXCEPT 연산자를 사용하여 두 사용자 목록 사이의 차이를 찾을 수 있습니다. 역할과는 상관없이 모든 사용자의 목록을 빌드하려면 UNION 연산자를 사용할 수 있습니다.

구문

```
query
{ UNION [ ALL ] | INTERSECT | EXCEPT | MINUS }
query
```

파라미터

query

UNION, INTERSECT 또는 EXCEPT 연산자 뒤에 쿼리 표현식의 선택 목록 형태로 제2의 쿼리 표현식에 상응하는 쿼리 표현식입니다. 이 두 표현식에는 호환 데이터 형식을 가진 같은 개수의 출력 열이 있어야 합니다. 그렇지 않으면 두 결과 집합을 비교 및 병합할 수 없습니다. 설정 연산은 서로 다른 범주의 데이터 형식 간의 암시적 변환을 허용하지 않습니다. 자세한 내용은 형식 호환성 및 변환 섹션을 참조하세요.

무제한 개수의 쿼리 표현식을 포함하는 쿼리를 빌드하고 임의의 조합으로 UNION, INTERSECT 및 EXCEPT 연산자와 연결할 수 있습니다. 예를 들어, 테이블 T1, T2 및 T3에 호환되는 열 집합이 포함되어 있다고 가정하면 다음 쿼리 구조가 유효합니다.

```
select * from t1
union
select * from t2
except
select * from t3
order by c1;
```

UNION

행이 한 표현식이나 두 표현식 모두에서 파생하는지에 상관없이, 두 쿼리 표현식에서 행을 반환하는 작업을 설정합니다.

INTERSECT

두 쿼리 표현식에서 파생하는 행을 반환하는 작업을 설정합니다. 두 표현식에서 모두 반환되지 않는 행은 삭제됩니다.

EXCEPT | MINUS

두 쿼리 표현식 중 하나에서 파생하는 행을 반환하는 작업을 설정합니다. 첫 번째 결과 테이블에는 있지만 두 번째 결과 테이블에는 없는 행에 대한 결과가 반환될 수 있다. MINUS 및 EXCEPT는 정확히 동의어입니다.

ALL

ALL 키워드는 UNION에 의해 생성되는 중복 행을 모두 유지합니다. ALL 키워드가 사용되지 않을 때의 기본 동작은 이러한 중복 항목을 삭제하는 것입니다. INTERSECT ALL, EXCEPT ALL 및 MINUS ALL은 지원되지 않습니다.

설정 연산자에 대한 평가 순서

UNION 및 EXCEPT 설정 연산자는 좌우선 결합 연산자입니다. 우선순위에 영향을 주기 위해 괄호가 지정되어 있지 않은 경우 이러한 설정 연산자의 조합은 왼쪽에서 오른쪽으로 계산됩니다. 예를 들어 다음 쿼리에서, T1 및 T2의 UNION이 먼저 계산된 다음 UNION 결과에 대해 EXCEPT 작업이 수행됩니다.

```
select * from t1
union
select * from t2
except
select * from t3
order by c1;
```

동일한 쿼리에 연산자 조합이 사용될 때 INTERSECT 연산자가 UNION 및 EXCEPT 연산자보다 우선합니다. 예를 들어 다음 쿼리는 T2 및 T3의 교집합을 계산한 다음 그 결과와 T1의 합집합을 구합니다.

```
select * from t1
union
select * from t2
intersect
select * from t3
order by c1;
```

괄호를 추가하면 다른 계산 순서를 적용할 수 있습니다. 다음 경우에는 T1 및 T2의 합집합 결과가 T3와 교집합을 이루고, 쿼리가 다른 결과를 낳을 가능성이 있습니다.

```
(select * from t1
union
select * from t2)
intersect
(select * from t3)
order by c1;
```

사용 노트

- 설정 작업 쿼리의 결과에 반환되는 열 이름은 첫 번째 쿼리 표현식의 테이블에서 가져온 열 이름(또는 별칭)입니다. 열의 값이 설정 연산자의 어느 한쪽에 있는 테이블에서 파생한다는 점에서 이런 열 이름은 오해를 불러일으킬 가능성이 있으므로, 결과 집합에 대해 의미 있는 별칭을 부여하고 싶을 수도 있습니다.

- 설정 연산자에 선행하는 쿼리 표현식에 ORDER BY 절을 포함하면 안 됩니다. ORDER BY 절은 설정 연산자를 포함하는 쿼리의 끝에 사용될 때만 의미 있게 정렬된 결과를 내놓습니다. 이 경우에는 ORDER BY 절이 모든 설정 작업의 최종 결과에 적용됩니다. 가장 바깥쪽 쿼리는 표준 LIMIT 및 OFFSET 절도 포함할 수 있습니다.
- 설정 연산자 쿼리가 10진수 결과를 반환할 때 그에 상응하는 결과 열은 같은 정밀도와 규모를 반환하도록 승격됩니다. 예를 들어, 다음 쿼리에서 T1.REVENUE가 DECIMAL(10,2) 열이고 T2.REVENUE가 DECIMAL(8,4) 열인 경우 10진수 결과는 DECIMAL(12,4)로 승격됩니다.

  ```
  select t1.revenue union select t2.revenue;
  ```

  규모는 두 열의 최대 규모인 4입니다. T1.REVENUE는 소수점 왼쪽에 8자리가 필요하므로(12 - 4 = 8) 정밀도는 12입니다. 이러한 유형 승격은 UNION 양쪽 모두의 값이 전부 결과에 부합하도록 합니다. 64비트 값의 경우, 최대 결과 정밀도는 19이고 최대 결과 규모는 18입니다. 128비트 값의 경우, 최대 결과 정밀도는 38이고 최대 결과 규모는 37입니다.

  결과 데이터 형식이 Amazon Redshift 전체 자릿수 및 소수 자릿수 제한을 초과하는 경우 쿼리는 오류를 반환합니다.
- 설정 작업의 경우, 각각 상응하는 열 쌍에 대해 두 데이터 값이 equal 또는 both NULL인 경우 두 행이 동일한 것으로 처리됩니다. 예를 들어, 테이블 T1과 T2에 모두 한 열과 한 행이 있고 그 행이 두 테이블에서 모두 NULL인 경우 두 테이블에 대해 INTERSECT 연산을 수행하면 바로 그 행이 반환됩니다.

UNION 쿼리 예

다음 UNION 쿼리에서 SALES 테이블의 행은 LISTING 테이블의 행과 병합됩니다. 각각의 테이블에서 호환되는 3개의 열이 선택되며, 이 경우에는 해당하는 열들의 이름과 데이터 형식이 동일합니다.

최종 결과 집합은 LISTING 테이블의 첫 번째 열을 기준으로 정렬되고 LISTID 값이 가장 높은 5개의 행으로 제한됩니다.

```
select listid, sellerid, eventid from listing
union select listid, sellerid, eventid from sales
order by listid, sellerid, eventid desc limit 5;

listid | sellerid | eventid
--------+----------+---------
1 |    36861 |    7872
2 |    16002 |    4806
```

```
3 |    21461 |    4256
4 |     8117 |    4337
5 |     1616 |    8647
(5 rows)
```

다음 예는 결과 집합에서 어떤 쿼리 표현식이 각각의 행을 생성했는지 볼 수 있도록 UNION 쿼리의 출력에 리터럴 값을 추가할 수 있는 방법을 보여줍니다. 이 쿼리는 첫 번째 쿼리 표현식의 행을 "B"(buyer)로 식별하고 두 번째 쿼리 표현식의 행을 "S"(seller)로 식별합니다.

이 쿼리는 $10,000 이상의 티켓 거래에 대해 구매자와 판매자를 식별합니다. UNION 연산자의 어느 한 쪽에서 두 쿼리 표현식의 유일한 차이점은 SALES 테이블에 대한 조인 열입니다.

```
select listid, lastname, firstname, username,
pricepaid as price, 'S' as buyorsell
from sales, users
where sales.sellerid=users.userid
and pricepaid >=10000
union
select listid, lastname, firstname, username, pricepaid,
'B' as buyorsell
from sales, users
where sales.buyerid=users.userid
and pricepaid >=10000
order by 1, 2, 3, 4, 5;

listid | lastname | firstname | username |   price   | buyorsell
--------+----------+-----------+----------+-----------+-----------
209658 | Lamb     | Colette   | VOR15LYI |  10000.00 | B
209658 | West     | Kato      | ELU81XAA |  10000.00 | S
212395 | Greer    | Harlan    | GXO71KOC |  12624.00 | S
212395 | Perry    | Cora      | YWR73YNZ |  12624.00 | B
215156 | Banks    | Patrick   | ZNQ69CLT |  10000.00 | S
215156 | Hayden   | Malachi   | BBG56AKU |  10000.00 | B
(6 rows)
```

중복된 행이 발견되는 경우 결과에 이런 행을 유지해야 하므로, 다음 예에서는 UNION ALL 연산자를 사용합니다. 이벤트 ID의 특정 시리즈에 대해, 쿼리는 각 이벤트와 관련된 각각의 판매에 대해 0개 이상의 행을 반환하고 그 이벤트의 각 목록에 대해 0개 또는 1개의 행을 반환합니다. 이벤트 ID는 LISTING 및 EVENT 테이블에서 각각의 행에 고유하지만, SALES 테이블에서 이벤트 및 목록 ID의 동일한 조합에 대해 여러 개의 판매 건이 있을 수 있습니다.

결과 집합의 세 번째 열은 행의 원본을 식별합니다. 행의 출처가 SALES 테이블인 경우 SALESROW 열에 "YES"로 표시됩니다. (SALESROW는 SALES.LISTID의 별칭입니다.) 행의 출처가 LISTING 테이블인 경우 SALESROW 열에 "No"로 표시됩니다.

이 경우, 결과 집합은 목록 500, 이벤트 7787에 대해 3개의 판매 행으로 구성됩니다. 즉, 이 목록 및 이벤트 조합에 대해 3가지 다른 트랜잭션이 발생했습니다. 다른 두 목록 501 및 502에서는 어떤 판매도 생성되지 않았으므로, 쿼리가 이들 목록 ID에 대해 생성하는 유일한 행의 출처는 LISTING 테이블입니다(SALESROW = 'No').

```
select eventid, listid, 'Yes' as salesrow
from sales
where listid in(500,501,502)
union all
select eventid, listid, 'No'
from listing
where listid in(500,501,502)
order by listid asc;

eventid | listid | salesrow
---------+--------+----------
7787 |    500 | No
7787 |    500 | Yes
7787 |    500 | Yes
7787 |    500 | Yes
6473 |    501 | No
5108 |    502 | No
(6 rows)
```

ALL 키워드 없이 같은 쿼리를 실행하는 경우 결과에는 판매 거래 중 하나만 유지됩니다.

```
select eventid, listid, 'Yes' as salesrow
from sales
where listid in(500,501,502)
union
select eventid, listid, 'No'
from listing
where listid in(500,501,502)
order by listid asc;

eventid | listid | salesrow
---------+--------+----------
7787 |    500 | No
```

```
7787 |    500 | Yes
6473 |    501 | No
5108 |    502 | No
(4 rows)
```

UNION ALL 쿼리 예

중복된 행이 발견되는 경우 결과에 이런 행을 유지해야 하므로, 다음 예에서는 UNION ALL 연산자를 사용합니다. 이벤트 ID의 특정 시리즈에 대해, 쿼리는 각 이벤트와 관련된 각각의 판매에 대해 0개 이상의 행을 반환하고 그 이벤트의 각 목록에 대해 0개 또는 1개의 행을 반환합니다. 이벤트 ID는 LISTING 및 EVENT 테이블에서 각각의 행에 고유하지만, SALES 테이블에서 이벤트 및 목록 ID의 동일한 조합에 대해 여러 개의 판매 건이 있을 수 있습니다.

결과 집합의 세 번째 열은 행의 원본을 식별합니다. 행의 출처가 SALES 테이블인 경우 SALESROW 열에 "YES"로 표시됩니다. (SALESROW는 SALES.LISTID의 별칭입니다.) 행의 출처가 LISTING 테이블인 경우 SALESROW 열에 "No"로 표시됩니다.

이 경우, 결과 집합은 목록 500, 이벤트 7787에 대해 3개의 판매 행으로 구성됩니다. 즉, 이 목록 및 이벤트 조합에 대해 3가지 다른 트랜잭션이 발생했습니다. 다른 두 목록 501 및 502에서는 어떤 판매도 생성되지 않았으므로, 쿼리가 이들 목록 ID에 대해 생성하는 유일한 행의 출처는 LISTING 테이블입니다(SALESROW = 'No').

```
select eventid, listid, 'Yes' as salesrow
from sales
where listid in(500,501,502)
union all
select eventid, listid, 'No'
from listing
where listid in(500,501,502)
order by listid asc;

eventid | listid | salesrow
---------+--------+----------
7787 |    500 | No
7787 |    500 | Yes
7787 |    500 | Yes
7787 |    500 | Yes
6473 |    501 | No
5108 |    502 | No
(6 rows)
```

ALL 키워드 없이 같은 쿼리를 실행하는 경우 결과에는 판매 거래 중 하나만 유지됩니다.

```
select eventid, listid, 'Yes' as salesrow
from sales
where listid in(500,501,502)
union
select eventid, listid, 'No'
from listing
where listid in(500,501,502)
order by listid asc;

eventid | listid | salesrow
---------+--------+----------
7787 |    500 | No
7787 |    500 | Yes
6473 |    501 | No
5108 |    502 | No
(4 rows)
```

INTERSECT 쿼리 예

다음 예를 첫 번째 UNION 예와 비교해 보십시오. 두 예에서는 사용되는 설정 연산자만 다를 뿐이지만, 그 결과는 매우 상이합니다. 다음과 같이 행들 중 하나만 같습니다.

```
235494 |    23875 |    8771
```

이 행이 양쪽 테이블에서 발견된 5개 행의 제한된 결과에 있는 유일한 행입니다.

```
select listid, sellerid, eventid from listing
intersect
select listid, sellerid, eventid from sales
order by listid desc, sellerid, eventid
limit 5;

listid | sellerid | eventid
--------+----------+---------
235494 |    23875 |    8771
235482 |     1067 |    2667
235479 |     1589 |    7303
235476 |    15550 |     793
235475 |    22306 |    7848
(5 rows)
```

다음 쿼리는 3월에 뉴욕과 로스앤젤레스의 두 도시에서 모두 현장에서 이루어진 (티켓이 판매된) 이벤트를 찾습니다. 두 쿼리 표현식의 차이점은 VENUECITY 열에 대한 제약 조건입니다.

```
select distinct eventname from event, sales, venue
where event.eventid=sales.eventid and event.venueid=venue.venueid
and date_part(month,starttime)=3 and venuecity='Los Angeles'
intersect
select distinct eventname from event, sales, venue
where event.eventid=sales.eventid and event.venueid=venue.venueid
and date_part(month,starttime)=3 and venuecity='New York City'
order by eventname asc;

eventname
----------------------------
A Streetcar Named Desire
Dirty Dancing
Electra
Running with Annalise
Hairspray
Mary Poppins
November
Oliver!
Return To Forever
Rhinoceros
South Pacific
The 39 Steps
The Bacchae
The Caucasian Chalk Circle
The Country Girl
Wicked
Woyzeck
(16 rows)
```

EXCEPT 쿼리 예

TICKIT 데이터베이스의 CATEGORY 테이블은 다음 11개의 행을 포함합니다.

```
 catid | catgroup |  catname  |                 catdesc
-------+----------+-----------+--------------------------------------------
   1   | Sports   | MLB       | Major League Baseball
   2   | Sports   | NHL       | National Hockey League
   3   | Sports   | NFL       | National Football League
   4   | Sports   | NBA       | National Basketball Association
```

```
   5   | Sports   | MLS       | Major League Soccer
   6   | Shows    | Musicals  | Musical theatre
   7   | Shows    | Plays     | All non-musical theatre
   8   | Shows    | Opera     | All opera and light opera
   9   | Concerts | Pop       | All rock and pop music concerts
  10   | Concerts | Jazz      | All jazz singers and bands
  11   | Concerts | Classical | All symphony, concerto, and choir concerts
(11 rows)
```

CATEGORY_STAGE 테이블(스테이징 테이블)에 추가적인 행이 한 개 있다고 가정합니다.

```
 catid | catgroup |  catname  |                 catdesc
-------+----------+-----------+--------------------------------------------
1 | Sports   | MLB       | Major League Baseball
2 | Sports   | NHL       | National Hockey League
3 | Sports   | NFL       | National Football League
4 | Sports   | NBA       | National Basketball Association
5 | Sports   | MLS       | Major League Soccer
6 | Shows    | Musicals  | Musical theatre
7 | Shows    | Plays     | All non-musical theatre
8 | Shows    | Opera     | All opera and light opera
9 | Concerts | Pop       | All rock and pop music concerts
10 | Concerts | Jazz      | All jazz singers and bands
11 | Concerts | Classical | All symphony, concerto, and choir concerts
12 | Concerts | Comedy    | All stand up comedy performances
(12 rows)
```

두 테이블 사이의 차이점을 반환합니다. 다시 말해, CATEGORY_STAGE 테이블에는 있지만 CATEGORY 테이블에는 없는 행을 반환합니다.

```
select * from category_stage
except
select * from category;

catid | catgroup | catname |             catdesc
-------+----------+---------+----------------------------------
12 | Concerts | Comedy  | All stand up comedy performances
(1 row)
```

다음과 같은 동등한 쿼리는 동의어 MINUS를 사용합니다.

```
select * from category_stage
```

```
minus
select * from category;

catid | catgroup | catname |             catdesc
-------+----------+---------+----------------------------------
12 | Concerts | Comedy  | All stand up comedy performances
(1 row)
```

SELECT 표현식의 순서를 반대로 하면 쿼리가 아무런 행도 반환하지 않습니다.

ORDER BY 절

주제

- 구문
- 파라미터
- 사용 노트
- ORDER BY 사용 예

ORDER BY 절은 쿼리의 결과 집합을 정렬합니다.

구문

```
[ ORDER BY expression [ ASC | DESC ] ]
[ NULLS FIRST | NULLS LAST ]
[ LIMIT { count | ALL } ]
[ OFFSET start ]
```

파라미터

expression

일반적으로 선택 목록에 하나 이상의 열을 지정하여 쿼리 결과 집합의 정렬 순서를 정의하는 표현식입니다. 결과는 이진 UTF-8 순서를 기준으로 반환됩니다. 다음을 지정할 수도 있습니다.

- 선택 목록에 없는 열
- 쿼리에서 참조하는 테이블에 존재하는 하나 이상의 열에서 형성되는 표현식
- 선택 목록 항목의 위치(또는 선택 목록이 없는 경우 테이블에서 열의 위치)를 나타내는 서수
- 선택 목록 항목을 정의하는 별칭

ORDER BY 절에 여러 개의 표현식이 포함되어 있을 때는 결과 집합이 첫 번째 표현식에 따라 정렬된 다음, 두 번째 표현식이 첫 번째 표현식의 일치하는 값을 가진 행에 적용되는 등의 방식이 적용됩니다.

ASC | DESC

표현식의 정렬 순서를 정의하는 옵션으로서 각각 다음과 같은 의미를 갖습니다.

- ASC: 오름차순(예: 숫자 값의 경우 낮은 값에서 높은 값 순, 문자열의 경우 'A'에서 'Z'의 순. 지정된 옵션이 없는 경우에는 데이터가 기본적으로 오름차순으로 정렬됩니다.
- DESC: 내림차순(숫자 값의 경우 높은 값에서 낮은 값 순, 문자열의 경우 'Z'에서 'A'의 순).

NULLS FIRST | NULLS LAST

NULL 값의 순서를 NULL 값 이외의 값 이전에 결정할지, 혹은 이후에 결정할지 지정하는 옵션입니다. 기본적으로 NULL 값은 ASC 순서에서는 마지막에 정렬 후 순위가 결정되며, DESC 순서에서는 처음에 정렬 후 순위가 결정됩니다.

LIMIT number | ALL

쿼리가 반환하는 정렬된 행의 수를 제어하는 옵션입니다. LIMIT 수는 양의 정수여야 합니다. 최댓값은 2147483647입니다.

LIMIT 0은 아무런 행도 반환하지 않습니다. 이 구문을 테스트 목적으로 사용할 수 있습니다. 즉, 쿼리가 실행되는지 확인하거나(어떤 행도 표시하지 않음) 테이블에서 열 목록을 반환합니다. LIMIT 0을 사용하여 열 목록을 반환하는 경우 ORDER BY 절은 중복입니다. 기본값은 LIMIT ALL입니다.

OFFSET start

행 반환을 위해 시작하기 전에 start 앞에 있는 행의 개수를 건너뛰도록 지정하는 옵션입니다. OFFSET 수는 양의 정수여야 합니다. 최댓값은 2147483647입니다. LIMIT 옵션과 함께 사용 시, OFFSET개의 행을 건너뛴 후 반환되는 LIMIT 행 수를 카운트하기 시작합니다. LIMIT 옵션이 사용되지 않는 경우 결과 집합의 행 개수는 건너뛰는 행 개수만큼 감소됩니다. OFFSET 절에 의해 건너뛰는 행을 계속 스캔해야 하므로, 큰 OFFSET 값을 사용하기에 부족할 수 있습니다.

사용 노트

ORDER BY 절을 사용할 때 다음과 같이 예상되는 동작에 유의하세요.

- NULL 값은 다른 모든 값보다 "높은 값"으로 간주됩니다. 기본 오름차순 정렬 순서에 따라 NULL 값은 끝에 정렬됩니다. 이 동작을 변경하려면 NULLS FIRST 옵션을 사용하세요.

- 쿼리에 ORDER BY 절이 포함되어 있지 않을 때, 시스템에서는 행 순서를 예측할 수 없는 결과 집합을 반환합니다. 같은 쿼리를 두 번 실행할 경우 결과 집합을 다른 순서로 반환할 수도 있습니다.
- ORDER BY 절 없이 LIMIT 및 OFFSET 옵션을 사용할 수 있지만, 일관성 있는 행 집합을 반환하려면 ORDER BY와 함께 이러한 옵션을 사용하세요.
- Amazon Redshift와 같은 병렬 시스템에서는 ORDER BY가 고유한 순서를 지정하지 않으면 행의 순서는 비확정적입니다. 다시 말해 ORDER BY 표현식에서 중복 값이 산출되면 해당하는 행의 반환 순서가 다른 시스템과는 다르거나 Amazon Redshift를 실행할 때마다 달라질 수 있습니다.
- Amazon Redshift는 ORDER BY 절에서 문자열 리터럴을 지원하지 않습니다.

ORDER BY 사용 예

두 번째 열인 CATGROUP 열을 기준으로 정렬된 CATEGORY 테이블에서 11개의 행을 전부 반환합니다. 같은 CATGROUP 값을 가진 결과에 대해서는 문자열의 길이를 기준으로 CATDESC 열 값의 순서를 지정합니다. 그런 다음 열 CATID 및 CATNAME을 기준으로 정렬합니다.

```
select * from category order by 2, length(catdesc), 1, 3;

catid | catgroup |  catname  |                  catdesc
------+----------+-----------+--------------------------------------
10    | Concerts | Jazz      | All jazz singers and bands
9     | Concerts | Pop       | All rock and pop music concerts
11    | Concerts | Classical | All symphony, concerto, and choir conce
6     | Shows    | Musicals  | Musical theatre
7     | Shows    | Plays     | All non-musical theatre
8     | Shows    | Opera     | All opera and light opera
5     | Sports   | MLS       | Major League Soccer
1     | Sports   | MLB       | Major League Baseball
2     | Sports   | NHL       | National Hockey League
3     | Sports   | NFL       | National Football League
4     | Sports   | NBA       | National Basketball Association
(11 rows)
```

가장 높은 QTYSOLD 값을 기준으로 정렬된 SALES 테이블에서 선택한 열을 반환합니다. 결과를 맨 위의 10개 행으로 제한합니다.

```
select salesid, qtysold, pricepaid, commission, saletime from sales
order by qtysold, pricepaid, commission, salesid, saletime desc
limit 10;
```

```
salesid | qtysold | pricepaid | commission |      saletime
--------+---------+-----------+------------+---------------------
15401   |       8 |    272.00 |      40.80 | 2008-03-18 06:54:56
61683   |       8 |    296.00 |      44.40 | 2008-11-26 04:00:23
90528   |       8 |    328.00 |      49.20 | 2008-06-11 02:38:09
74549   |       8 |    336.00 |      50.40 | 2008-01-19 12:01:21
130232  |       8 |    352.00 |      52.80 | 2008-05-02 05:52:31
55243   |       8 |    384.00 |      57.60 | 2008-07-12 02:19:53
16004   |       8 |    440.00 |      66.00 | 2008-11-04 07:22:31
489     |       8 |    496.00 |      74.40 | 2008-08-03 05:48:55
4197    |       8 |    512.00 |      76.80 | 2008-03-23 11:35:33
16929   |       8 |    568.00 |      85.20 | 2008-12-19 02:59:33
(10 rows)
```

LIMIT 0 구문을 사용하여 열 목록은 반환하고 행은 반환하지 않습니다.

```
select * from venue limit 0;
venueid | venuename | venuecity | venuestate | venueseats
---------+-----------+-----------+------------+------------
(0 rows)
```

CONNECT BY 절

계층 구조에서 행 간의 관계를 지정합니다. CONNECT BY를 사용하여 테이블을 자체에 조인하고 계층적 데이터를 처리하여 계층적 순서로 행을 선택할 수 있습니다. 예를 들어 조직도와 목록 데이터를 반복적으로 반복하는 데 사용할 수 있습니다.

계층적 쿼리는 다음 순서로 처리됩니다.

1. FROM 절에 조인이 있는 경우 조인이 먼저 처리됩니다.
2. CONNECT BY 절이 평가됩니다.
3. WHERE 절이 평가됩니다.

구문

```
[START WITH start_with_conditions]
CONNECT BY connect_by_conditions
```

> **Note**
>
> START 및 CONNECT는 예약어가 아니지만, 쿼리에서 START 및 CONNECT를 테이블 별칭으로 사용하는 경우에는 런타임 시 오류가 발생하지 않도록 구분된 식별자(큰따옴표) 또는 AS를 사용하세요.

```
SELECT COUNT(*)
FROM Employee "start"
CONNECT BY PRIOR id = manager_id
START WITH name = 'John'
```

```
SELECT COUNT(*)
FROM Employee AS start
CONNECT BY PRIOR id = manager_id
START WITH name = 'John'
```

파라미터

start_with_conditions

계층 구조의 루트 행을 지정하는 조건

connect_by_conditions

계층 구조의 상위 행과 하위 행 간의 관계를 지정하는 조건입니다. 하나 이상의 조건이 부모 행을 참조하는 데 사용되는 단항 연산자로 한정되어야 합니다.

```
PRIOR column = expression
-- or
expression > PRIOR column
```

연산자

CONNECT BY 쿼리에서 다음 연산자를 사용할 수 있습니다.

LEVEL

계층 구조에서 현재 행 수준을 반환하는 의사 열입니다. 루트 행에 대해 1, 루트 행의 자식에 대해 2 등을 반환합니다.

PRIOR

계층 구조에서 현재 행의 상위 행에 대한 표현식을 평가하는 단항 연산자입니다.

예시

다음 예는 John에게 직간접적으로 보고하는 직원 수를 반환하는 CONNECT BY 쿼리입니다(4단계 미만).

```
SELECT id, name, manager_id
FROM employee
WHERE LEVEL < 4
START WITH name = 'John'
CONNECT BY PRIOR id = manager_id;
```

다음은 쿼리 결과입니다.

```
id      name      manager_id
------+----------+--------------
  101     John        100
  102     Jorge       101
  103     Kwaku       101
  110     Liu         101
  201     Sofía       102
  106     Mateo       102
  110     Nikki       103
  104     Paulo       103
  105     Richard     103
  120     Saanvi      104
  200     Shirley     104
  205     Zhang       104
```

이 예에 대한 테이블 정의는 다음과 같습니다.

```
CREATE TABLE employee (
   id INT,
   name VARCHAR(20),
   manager_id INT
   );
```

다음은 테이블에 삽입된 행입니다.

```
INSERT INTO employee(id, name, manager_id)  VALUES
(100, 'Carlos', null),
(101, 'John', 100),
(102, 'Jorge', 101),
(103, 'Kwaku', 101),
(110, 'Liu', 101),
(106, 'Mateo', 102),
(110, 'Nikki', 103),
(104, 'Paulo', 103),
(105, 'Richard', 103),
(120, 'Saanvi', 104),
(200, 'Shirley', 104),
(201, 'Sofía', 102),
(205, 'Zhang', 104);
```

다음은 John의 부서의 조직도입니다.

하위 쿼리 예

다음 예에서는 하위 쿼리가 SELECT 쿼리에 적합한 다른 방법을 보여줍니다. 하위 쿼리의 다른 사용 예는 JOIN 예 섹션을 참조하세요.

SELECT 목록 하위 쿼리

다음 예에서는 SELECT 목록에 하위 쿼리를 포함합니다. 이 하위 쿼리는 스칼라이므로 한 개의 열과 한 개의 값만 반환하며, 이는 외부 쿼리에서 반환되는 각 행에 대한 결과에서 반복됩니다. 이 쿼리는 외부 쿼리에 의해 정의된 바와 같이 2008년의 다른 두 분기(2분기 및 3분기)에 대한 판매액 값과 하위 쿼리가 계산하는 Q1SALES 값을 비교합니다.

```
select qtr, sum(pricepaid) as qtrsales,
(select sum(pricepaid)
from sales join date on sales.dateid=date.dateid
where qtr='1' and year=2008) as q1sales
from sales join date on sales.dateid=date.dateid
where qtr in('2','3') and year=2008
group by qtr
order by qtr;

qtr  |  qtrsales   |   q1sales
-------+-------------+-------------
2     | 30560050.00 | 24742065.00
```

```
3     | 31170237.00 | 24742065.00
(2 rows)
```

WHERE 절 하위 쿼리

다음 예에서는 WHERE 절에 테이블 하위 쿼리를 포함합니다. 이 하위 쿼리는 여러 개의 행을 만들어냅니다. 이 경우에는 행에 한 개의 열만 포함되지만, 테이블 하위 쿼리는 다른 테이블과 마찬가지로 여러 개의 열과 행을 포함할 수 있습니다.

이 쿼리는 최대 판매 티켓 수를 기준으로 상위 10개의 판매사를 찾습니다. 톱 10 목록은 티켓 판매소가 있는 도시에 사는 사용자를 제거하는 하위 쿼리에 의해 한정됩니다. 이 쿼리는 다양한 방법으로 작성할 수 있습니다. 예를 들어, 하위 쿼리를 기본 쿼리 내의 조인으로 다시 작성할 수 있습니다.

```
select firstname, lastname, city, max(qtysold) as maxsold
from users join sales on users.userid=sales.sellerid
where users.city not in(select venuecity from venue)
group by firstname, lastname, city
order by maxsold desc, city desc
limit 10;

firstname | lastname  |      city      | maxsold
-----------+-----------+----------------+---------
Noah       | Guerrero | Worcester      |       8
Isadora    | Moss     | Winooski       |       8
Kieran     | Harrison | Westminster    |       8
Heidi      | Davis    | Warwick        |       8
Sara       | Anthony  | Waco           |       8
Bree       | Buck     | Valdez         |       8
Evangeline | Sampson  | Trenton        |       8
Kendall    | Keith    | Stillwater     |       8
Bertha     | Bishop   | Stevens Point  |       8
Patricia   | Anderson | South Portland |       8
(10 rows)
```

WITH 절 하위 쿼리

WITH 절 섹션을 참조하세요.

상관관계가 있는 하위 쿼리

다음 예에서는 WHERE 절에 상관관계가 있는 하위 쿼리가 포함됩니다. 이런 종류의 하위 쿼리는 자신의 열과 외부 쿼리에 의해 생성되는 열 사이에 하나 이상의 상관관계를 포함합니다. 이 경우 상관관계

는 where s.listid=l.listid입니다. 외부 쿼리가 생성하는 각각의 행에 자격을 주거나 자격을 취소하는 하위 쿼리가 실행됩니다.

```
select salesid, listid, sum(pricepaid) from sales s
where qtysold=
(select max(numtickets) from listing l
where s.listid=l.listid)
group by 1,2
order by 1,2
limit 5;

salesid | listid |   sum
--------+--------+----------
 27     |     28 | 111.00
 81     |    103 | 181.00
 142    |    149 | 240.00
 146    |    152 | 231.00
 194    |    210 | 144.00
(5 rows)
```

지원되지 않는 상관관계를 가진 하위 쿼리 패턴

쿼리 플래너는 하위 쿼리 상관관계 제거라는 쿼리 재작성 방법을 사용하여 MPP 환경에서 실행하기 위해 상관관계가 있는 하위 쿼리의 여러 패턴을 최적화합니다. 상관관계를 가진 몇 가지 유형의 하위 쿼리는 Amazon Redshift가 상관관계를 제거할 수 없고 지원하지 않는 패턴을 따릅니다. 다음 상관관계 참조를 포함하는 쿼리는 오류를 반환합니다.

- "건너뛰기 수준의 상관관계 참조"라고도 하는, 쿼리 블록을 건너뛰는 상관관계 참조. 예를 들어, 다음 쿼리에서 상관관계 참조를 포함하는 블록과 건너뛰는 블록은 NOT EXISTS 조건자에 의해 연결됩니다.

  ```
  select event.eventname from event
  where not exists
  (select * from listing
  where not exists
  (select * from sales where event.eventid=sales.eventid));
  ```

  이 경우에 건너뛰는 블록은 LISTING 테이블에 대한 하위 쿼리입니다. 상관관계 참조는 EVENT 테이블과 SALES 테이블의 상관관계를 지정합니다.
- 외부 조인에서 ON 절의 일부인 하위 쿼리에서의 상관관계 참조:

```
select * from category
left join event
on category.catid=event.catid and eventid =
(select max(eventid) from sales where sales.eventid=event.eventid);
```

ON 절은 외부 쿼리의 EVENT에 대한 하위 쿼리에 있는 SALES에서의 상관관계 참조를 포함합니다.

- Amazon Redshift 시스템 테이블에 대해 Null에 민감한 상관관계 참조. 예:

```
select attrelid
from stv_locks sl, pg_attribute
where sl.table_id=pg_attribute.attrelid and 1 not in
(select 1 from pg_opclass where sl.lock_owner = opcowner);
```

- 창 함수를 포함하는 하위 쿼리 내에서의 상관관계 참조.

```
select listid, qtysold
from sales s
where qtysold not in
(select sum(numtickets) over() from listing l where s.listid=l.listid);
```

- GROUP BY 열에서 상관관계를 가진 하위 쿼리의 결과에 대한 참조. 예:

```
select listing.listid,
(select count (sales.listid) from sales where sales.listid=listing.listid) as list
from listing
group by list, listing.listid;
```

- IN 조건자에 의해 외부 쿼리에 연결된 집계 함수와 GROUP BY 절이 있는 하위 쿼리에서의 상관관계 참조. (이 제한 사항은 MIN 및 MAX 집계 함수에는 적용되지 않습니다.) 예:

```
select * from listing where listid in
(select sum(qtysold)
from sales
where numtickets>4
group by salesid);
```

SELECT INTO

임의의 쿼리에 의해 정의된 행을 선택하여 새 테이블에 삽입합니다. 임시 테이블을 만들지 영구 테이블을 만들지 여부를 지정할 수 있습니다.

구문

```
[ WITH with_subquery [, ...] ]
SELECT
[ TOP number ] [ ALL | DISTINCT ]
* | expression [ AS output_name ] [, ...]
INTO [ TEMPORARY | TEMP ] [ TABLE ] new_table
[ FROM table_reference [, ...] ]
[ WHERE condition ]
[ GROUP BY expression [, ...] ]
[ HAVING condition [, ...] ]
[ { UNION | INTERSECT | { EXCEPT | MINUS } } [ ALL ] query ]
[ ORDER BY expression
[ ASC | DESC ]
[ LIMIT { number | ALL } ]
[ OFFSET start ]
```

이 명령의 파라미터에 대한 자세한 내용은 SELECT 섹션을 참조하세요.

예시

EVENT 테이블에서 모든 행을 선택하고 NEWEVENT 테이블을 만듭니다.

```
select * into newevent from event;
```

PROFITS라는 임시 테이블에 대한 집계 쿼리의 결과를 선택합니다.

```
select username, lastname, sum(pricepaid-commission) as profit
into temp table profits
from sales, users
where sales.sellerid=users.userid
group by 1, 2
order by 3 desc;
```

SET

서버 구성 파라미터의 값을 설정합니다. SET 명령을 사용하여 현재 세션 또는 트랜잭션에 한해 지속 시간 설정을 재정의합니다.

reset 명령을 사용하여 파라미터를 기본값으로 반환합니다.

여러 가지 방법으로 서버 구성 파라미터를 변경할 수 있습니다. 자세한 내용은 서버 구성 수정 단원을 참조하십시오.

구문

```
SET { [ SESSION | LOCAL ]
{ SEED | parameter_name } { TO | = }
{ value | 'value' | DEFAULT } |
SEED TO value }
```

다음 문은 세션 컨텍스트 변수의 값을 설정합니다.

```
SET { [ SESSION | LOCAL ]
variable_name { TO | = }
{ value | 'value'  }
```

파라미터

세션

현재 세션에 대해 설정이 유효함을 지정합니다. 기본값.

variable_name

세션에 설정된 컨텍스트 변수의 이름을 지정합니다.

명명 규칙은 점으로 구분된 두 부분으로 이루어진 이름입니다(예: identifier.identifier). 점 구분 기호는 하나만 사용할 수 있습니다. Amazon Redshift의 표준 식별자 규칙을 따르는 identifier를 사용합니다. 자세한 내용은 이름 및 식별자 섹션을 참조하세요. 구분된 식별자는 허용되지 않습니다.

LOCAL

현재 트랜잭션에 대해 설정이 유효함을 지정합니다.

SEED TO value

난수 생성을 위한 RANDOM 함수에서 사용할 내부 시드를 설정합니다.

SET SEED는 0과 1 사이의 숫자 값을 취하고 RANDOM 함수 함수과 함께 사용하기 위해 이 숫자에 (2^{31}-1)을 곱합니다. 여러 개의 RANDOM 호출을 하기 전에 SET SEED를 사용하면 RANDOM이 예측 가능한 순서대로 숫자를 생성합니다.

parameter_name

설정할 파라미터의 이름. 파라미터에 대한 자세한 내용은 서버 구성 수정 섹션을 참조하세요.

USD 상당

새 재료 파라미터 값입니다. 값을 특정 문자열로 설정하려면 작은따옴표를 사용하십시오. SET SEED를 사용하는 경우 이 파라미터는 SEED 값을 포함합니다.

DEFAULT

이 파라미터를 기본값으로 설정합니다.

예시

현재 세션을 위한 파라미터 변경

다음 예에서는 날짜 스타일을 설정합니다.

```
set datestyle to 'SQL,DMY';
```

워크로드 관리를 위한 쿼리 그룹 설정

쿼리 그룹이 클러스터의 WLM 구성의 일부로서 대기열 정의에 나열되는 경우 QUERY_GROUP 파라미터를 나열된 쿼리 그룹 이름으로 설정할 수 있습니다. 후속 쿼리는 관련 쿼리 대기열에 할당됩니다. QUERY_GROUP 설정은 세션 지속 시간 동안 또는 RESET QUERY_GROUP 명령이 발생할 때까지 계속 적용됩니다.

이 예에서는 쿼리 그룹 'priority'에 속한 쿼리 2개를 실행한 후 쿼리 그룹을 재설정합니다.

```
set query_group to 'priority';
select tbl, count(*)from stv_blocklist;
select query, elapsed, substring from svl_qlog order by query desc limit 5;
reset query_group;
```

자세한 내용은 워크로드 관리 단원을 참조하십시오.

세션의 기본 ID 네임스페이스 변경

데이터베이스 사용자가 default_identity_namespace를 설정할 수 있습니다. 이 샘플은 SET SESSION을 사용하여 현재 세션 기간 동안 설정을 재정의하는 방법과 새 ID 제공업체 값을 보여줍니다. 이는 Redshift 및 IAM Identity Center와 함께 ID 제공업체를 사용할 때 가장 일반적으로 사용됩니다. Redshift에서 ID 제공업체를 사용하는 방법에 대한 자세한 내용은 Redshift를 IAM Identity Center와 연결하여 사용자에게 Single Sign-On 경험을 제공합니다를 참조하세요.

```
SET SESSION default_identity_namespace = 'MYCO';

SHOW default_identity_namespace;
```

이 명령을 실행한 후 다음과 같이 GRANT 문 또는 CREATE 문을 실행할 수 있습니다.

```
GRANT SELECT ON TABLE mytable TO alice;

GRANT UPDATE ON TABLE mytable TO salesrole;

CREATE USER bob password 'md50c983d1a624280812631c5389e60d48c';
```

이 경우 기본 ID 네임스페이스를 설정하는 효과는 각 ID 앞에 네임스페이스를 붙이는 것과 같습니다. 이 예제에서는 alice가 MYCO:alice로 대체됩니다. IAM Identity Center를 사용한 Redshift 구성과 관련된 설정에 대한 자세한 내용은 ALTER SYSTEM 및 ALTER IDENTITY PROVIDER 섹션을 참조하세요.

쿼리의 그룹 레이블 설정

QUERY_GROUP 파라미터는 SET 명령 뒤에 나오는 동일한 세션에서 실행되는 하나 이상의 쿼리에 대한 레이블을 정의합니다. 그러면 쿼리가 실행될 때 이 레이블이 로그에 기록되어 STL_QUERY 및 STV_INFLIGHT 시스템 테이블과 SVL_QLOG 뷰에서 반환되는 결과를 제한하는 데 사용될 수 있습니다.

```
show query_group;
query_group
-------------
unset
(1 row)

set query_group to '6 p.m.';
```

```
show query_group;
query_group
-------------
6 p.m.
(1 row)

select * from sales where salesid=500;
salesid | listid | sellerid | buyerid | eventid | dateid | ...
---------+--------+----------+---------+---------+--------+-----
500 |    504 |     3858 |    2123 |    5871 |   2052 | ...
(1 row)

reset query_group;

select query, trim(label) querygroup, pid, trim(querytxt) sql
from stl_query
where label ='6 p.m.';
query | querygroup |  pid  |                  sql
-------+------------+-------+---------------------------------------
57 | 6 p.m.     | 30711 | select * from sales where salesid=500;
(1 row)
```

쿼리 그룹 레이블은 스크립트의 일부로서 실행되는 개별 쿼리 또는 쿼리 집합을 분리하기에 유용한 메커니즘입니다. ID를 기준으로 쿼리를 식별하고 추적할 필요가 없고, 레이블로 추적할 수 있습니다.

난수 생성을 위한 시드 값 설정

다음 예에서는 SET와 함께 SEED 옵션을 사용하여 RANDOM 함수가 예측 가능한 순서로 숫자를 생성하도록 합니다.

먼저 SEED 값을 설정하지 않고 RANDOM 정수 3개를 반환합니다.

```
select cast (random() * 100 as int);
int4
------
6
(1 row)

select cast (random() * 100 as int);
int4
------
68
```

```
(1 row)

select cast (random() * 100 as int);
int4
------
56
(1 row)
```

그런 다음 SEED 값을 .25로 설정한 후 RANDOM 숫자를 3개 더 반환합니다.

```
set seed to .25;

select cast (random() * 100 as int);
int4
------
21
(1 row)

select cast (random() * 100 as int);
int4
------
79
(1 row)

select cast (random() * 100 as int);
int4
------
12
(1 row)
```

마지막으로 SEED 값을 다시 .25로 설정한 후 RANDOM이 이전 세 번의 호출과 동일한 결과를 반환하는지 확인합니다.

```
set seed to .25;

select cast (random() * 100 as int);
int4
------
21
(1 row)

select cast (random() * 100 as int);
int4
```

```
------
79
(1 row)

select cast (random() * 100 as int);
int4
------
12
(1 row)
```

다음 예에서는 사용자 지정 컨텍스트 변수를 설정합니다.

```
SET app_context.user_id TO 123;
SET app_context.user_id TO 'sample_variable_value';
```

SET SESSION AUTHORIZATION

현재 세션의 사용자 이름을 설정합니다.

예를 들어, SET SESSION AUTHORIZATION 명령을 사용하여 어떤 세션이나 트랜잭션을 권한 없는 사용자 자격으로 임시로 실행하여 데이터베이스 액세스를 테스트할 수 있습니다. 이 명령을 실행하려면 데이터베이스 슈퍼 사용자여야 합니다.

구문

```
SET [ LOCAL ] SESSION AUTHORIZATION { user_name | DEFAULT }
```

파라미터

LOCAL

현재 트랜잭션에 대해 설정이 유효함을 지정합니다. 이 파라미터를 누락하면 현재 세션에 대해 설정이 유효함을 지정합니다.

user_name

설정할 사용자의 이름입니다. 사용자 이름을 식별자 또는 문자열 리터럴로 작성할 수 있습니다.

DEFAULT

세션 사용자 이름을 기본값으로 설정합니다.

예시

다음 예에서는 현재 세션의 사용자 이름을 dwuser:로 설정합니다.

```
SET SESSION AUTHORIZATION 'dwuser';
```

다음 예에서는 현재 트랜잭션의 사용자 이름을 dwuser:로 설정합니다.

```
SET LOCAL SESSION AUTHORIZATION 'dwuser';
```

이 예에서는 현재 세션의 사용자 이름을 기본 사용자 이름으로 설정합니다.

```
SET SESSION AUTHORIZATION DEFAULT;
```

SET SESSION CHARACTERISTICS

이 명령은 더 이상 사용되지 않습니다.

SET

서버 구성 파라미터의 현재 값을 표시합니다. SET 명령이 적용되는 경우 이 값은 현재 세션에 특정한 값일 수 있습니다. 구성 파라미터 목록은 구성 참조 섹션을 참조하세요.

구문

```
SHOW { parameter_name | ALL }
```

다음 문은 세션 컨텍스트 변수의 현재 값을 표시합니다. 변수가 없으면 Amazon Redshift에서 오류가 발생합니다.

```
SHOW variable_name
```

파라미터

parameter_name

지정된 파라미터의 현재 값을 표시합니다.

ALL

모든 파라미터의 현재 값을 표시합니다.

variable_name

지정된 변수의 현재 값을 표시합니다.

예시

다음 예에서는 query_group 파라미터의 값을 표시합니다.

```
show query_group;

query_group

unset
(1 row)
```

다음 예에서는 모든 파라미터의 목록과 그 값을 표시합니다.

```
show all;
name        |   setting
--------------------+--------------
datestyle           | ISO, MDY
extra_float_digits | 0
query_group         | unset
search_path         | $user,public
statement_timeout   | 0
```

다음 예에서는 지정된 변수의 현재 값을 표시합니다.

```
SHOW app_context.user_id;
```

SHOW COLUMNS

테이블의 열 목록과 일부 열 속성을 표시합니다.

각 출력 행은 쉼표로 구분된 데이터베이스 이름, 스키마 이름, 테이블 이름, 열 이름, 서수 위치, 열 기본값, null 가능, 데이터 유형, 문자 최대 길이, 숫자 정밀도 및 비고의 목록으로 구성됩니다. 이들 속성에 대한 자세한 내용은 SVV_ALL_COLUMNS 섹션을 참조하세요.

SHOW COLUMNS 명령으로 인해 10,000개 이상의 열이 반환되는 경우 오류가 반환됩니다.

구문

```
SHOW COLUMNS FROM TABLE database_name.schema_name.table_name [LIKE 'filter_pattern']
 [LIMIT row_limit ]
```

파라미터

database_name

나열할 테이블이 포함된 데이터베이스의 이름입니다.

AWS Glue Data Catalog에 테이블을 표시하려면 데이터베이스 이름으로 (awsdatacatalog)를 지정하고 시스템 구성 data_catalog_auto_mount가 true로 설정되어 있는지 확인합니다. 자세한 내용은 ALTER SYSTEM 단원을 참조하십시오.

schema_name

나열할 테이블이 포함된 스키마의 이름입니다.

AWS Glue Data Catalog 테이블을 표시하려면 AWS Glue 데이터베이스 이름을 스키마 이름으로 제공하세요.

table_name

나열할 열이 포함된 테이블의 이름입니다.

filter_pattern

테이블 이름과 일치하는 패턴이 있는 유효한 UTF-8 문자 표현식입니다. LIKE' 옵션은 다음과 같은 패턴 일치 메타문자를 지원하는 대/소문자 구분 일치를 수행합니다:

메타문자	설명
%	0개 이상의 문자 시퀀스를 일치시킵니다.
_	모든 문자를 일치시킵니다.

filter_pattern에 메타 문자가 포함되어 있지 않으면 패턴이 문자열 자체만 의미합니다. 이런 경우에는 LIKE가 등호 연산자와 동일한 역할을 합니다.

row_limit

반환할 최대 열 수입니다. row_limit는 0~10,000일 수 있습니다.

예시

다음 예제에서는 Amazon Redshift 데이터베이스에서 dev라는 이름의 열이 public 스키마 및 tb 테이블에 있는 경우를 보여 줍니다.

```
SHOW COLUMNS FROM TABLE dev.public.tb;

 database_name | schema_name | table_name | column_name | ordinal_position
 | column_default | is_nullable | data_type | character_maximum_length |
 numeric_precision | remarks
---------------+-------------+------------+-------------+-----------------
+----------------+-------------+-----------+-------------------------
+-------------------+---------
 dev           | public      | tb         | col         |               1 |
     | YES         | integer   |                           |                32 |
```

다음 예제에서는 awsdatacatalog라는 이름의 AWS Glue Data Catalog 데이터베이스에 있는 batman 스키마 및 nation 테이블에 있는 테이블을 보여줍니다. 출력은 2행으로 제한됩니다.

```
SHOW COLUMNS FROM TABLE awsdatacatalog.batman.nation LIMIT 2;

 database_name  | schema_name | table_name | column_name | ordinal_position
 | column_default | is_nullable | data_type | character_maximum_length |
 numeric_precision | remarks
----------------+-------------+------------+-------------+-----------------
+----------------+-------------+-----------+-------------------------
+-------------------+---------
 awsdatacatalog | batman      | nation     | n_nationkey |               1 |
      |             | integer   |                           |                 |
 awsdatacatalog | batman      | nation     | n_name      |               2 |
      |             | character |                           |                 |
```

SHOW EXTERNAL TABLE

테이블 속성과 열 속성을 포함하여 외부 테이블의 정의를 표시합니다. SHOW EXTERNAL TABLE 문의 출력을 사용하여 테이블을 다시 생성할 수 있습니다.

외부 테이블 생성에 대한 자세한 내용은 CREATE EXTERNAL TABLE 섹션을 참조하세요.

구문

```
SHOW EXTERNAL TABLE [external_database].external_schema.table_name [ PARTITION ]
```

파라미터

external_database

연결된 외부 데이터베이스의 이름입니다. 이 파라미터는 선택 사항입니다.

external_schema

연결된 외부 스키마의 이름입니다.

table_name

표시할 테이블의 이름입니다.

PARTITION

테이블 정의에 파티션을 추가하는 ALTER TABLE 문을 표시합니다.

예시

다음 예는 다음과 같이 정의된 외부 테이블을 기반으로 합니다.

```
CREATE EXTERNAL TABLE my_schema.alldatatypes_parquet_test_partitioned (
     csmallint smallint,
     cint int,
     cbigint bigint,
     cfloat float4,
     cdouble float8,
     cchar char(10),
     cvarchar varchar(255),
     cdecimal_small decimal(18,9),
     cdecimal_big decimal(30,15),
     ctimestamp TIMESTAMP,
     cboolean boolean,
     cstring varchar(16383)
)
PARTITIONED BY (cdate date, ctime TIMESTAMP)
STORED AS PARQUET
```

```
LOCATION 's3://amzn-s3-demo-bucket/alldatatypes_parquet_partitioned';
```

다음은 테이블 my_schema.alldatatypes_parquet_test_partitioned에 대한 SHOW EXTERNAL TABLE 명령과 출력의 예입니다.

```
SHOW EXTERNAL TABLE my_schema.alldatatypes_parquet_test_partitioned;
```

```
"CREATE EXTERNAL TABLE my_schema.alldatatypes_parquet_test_partitioned (
    csmallint smallint,
    cint int,
    cbigint bigint,
    cfloat float4,
    cdouble float8,
    cchar char(10),
    cvarchar varchar(255),
    cdecimal_small decimal(18,9),
    cdecimal_big decimal(30,15),
    ctimestamp timestamp,
    cboolean boolean,
    cstring varchar(16383)
)
PARTITIONED BY (cdate date, ctime timestamp)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://amzn-s3-demo-bucket/alldatatypes_parquet_partitioned';"
```

다음은 동일한 테이블이지만 데이터베이스도 파라미터에 지정되어 있는 SHOW EXTERNAL TABLE 명령 및 출력의 예입니다.

```
SHOW EXTERNAL TABLE my_database.my_schema.alldatatypes_parquet_test_partitioned;
```

```
"CREATE EXTERNAL TABLE my_database.my_schema.alldatatypes_parquet_test_partitioned (
    csmallint smallint,
    cint int,
    cbigint bigint,
    cfloat float4,
    cdouble float8,
    cchar char(10),
    cvarchar varchar(255),
```

```
    cdecimal_small decimal(18,9),
    cdecimal_big decimal(30,15),
    ctimestamp timestamp,
    cboolean boolean,
    cstring varchar(16383)
)
PARTITIONED BY (cdate date, ctime timestamp)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://amzn-s3-demo-bucket/alldatatypes_parquet_partitioned';"
```

다음은 PARTITION 파라미터 사용 시 SHOW EXTERNAL TABLE 명령과 출력의 예입니다. 테이블 정의에 파티션을 추가하는 ALTER TABLE 문이 출력에 포함됩니다.

```
SHOW EXTERNAL TABLE my_schema.alldatatypes_parquet_test_partitioned PARTITION;
```

```
"CREATE EXTERNAL TABLE my_schema.alldatatypes_parquet_test_partitioned (
    csmallint smallint,
    cint int,
    cbigint bigint,
    cfloat float4,
    cdouble float8,
    cchar char(10),
    cvarchar varchar(255),
    cdecimal_small decimal(18,9),
    cdecimal_big decimal(30,15),
    ctimestamp timestamp,
    cboolean boolean,
    cstring varchar(16383)
)
PARTITIONED BY (cdate date)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://amzn-s3-demo-bucket/alldatatypes_parquet_partitioned';
ALTER TABLE my_schema.alldatatypes_parquet_test_partitioned ADD IF NOT
 EXISTS PARTITION (cdate='2021-01-01') LOCATION 's3://amzn-s3-demo-bucket/
alldatatypes_parquet_partitioned2/cdate=2021-01-01';
ALTER TABLE my_schema.alldatatypes_parquet_test_partitioned ADD IF NOT
 EXISTS PARTITION (cdate='2021-01-02') LOCATION 's3://amzn-s3-demo-bucket/
alldatatypes_parquet_partitioned2/cdate=2021-01-02';"
```

SHOW DATABASES

Data Catalog 또는 Amazon Redshift 데이터 웨어하우스의 데이터베이스를 표시합니다. SHOW DATABASES는 데이터 웨어하우스 내의 데이터베이스, AWS Glue Data Catalog 데이터베이스(awsdatacatalog), 데이터 공유 데이터베이스 및 Lake Formation 데이터베이스와 같이 액세스 가능한 모든 데이터베이스를 나열합니다.

구문

Amazon Redshift 데이터 웨어하우스의 데이터베이스를 표시하는 방법:

```
SHOW DATABASES
[ LIKE '<expression>' ]
[ LIMIT row_limit ]
```

Data Catalog의 데이터베이스를 표시하는 방법:

```
SHOW DATABASES FROM DATA CATALOG
[ ACCOUNT  '<id1>', '<id2>', ... ]
[ LIKE '<expression>' ]
[ IAM_ROLE default | 'SESSION' | 'arn:aws:iam::<account-id>:role/<role-name>' ]
[ LIMIT row_limit ]
```

파라미터

계정 '<id1>', <id2>',...

데이터베이스를 나열할 AWS Glue Data Catalog 계정입니다. 이 파라미터를 생략하면 Amazon Redshift에서 클러스터를 소유한 계정의 데이터베이스를 표시한다는 의미입니다.

LIKE '<expression>'

지정한 표현식과 일치하는 기준으로 데이터베이스 목록을 필터링합니다. 이 파라미터는 와일드카드 문자 %(백분율) 및 _(밑줄)를 사용하는 패턴을 지원합니다.

IAM_ROLE default | 'SESSION' | 'arn:aws:iam::<account-id>:role/<role-name>'

SHOW DATABASES 명령을 실행할 때 클러스터와 연결된 IAM 역할을 지정하면 데이터베이스에서 쿼리를 실행할 때 Amazon Redshift가 해당 역할의 보안 인증 정보를 사용합니다.

`default` 키워드를 지정한다는 것은 기본값으로 설정되어 클러스터와 연결된 IAM 역할을 사용한다는 의미입니다.

페더레이션 자격 증명을 사용하여 Amazon Redshift 클러스터에 연결하고 the section called "데이터베이스 생성" 명령을 사용하여 생성된 외부 스키마에서 테이블에 액세스하는 경우에 'SESSION'을 사용합니다. 페더레이션 ID 사용의 예를 보려면 페더레이션형 ID 구성 방법이 설명된 페더레이션형 ID를 사용하여 로컬 리소스 및 Amazon Redshift Spectrum 외부 테이블에 대한 Amazon Redshift 액세스 관리 섹션을 참조하세요.

클러스터가 인증 및 권한 부여에 사용하는 IAM 역할의 Amazon 리소스 이름(ARN)을 사용합니다. 최소 IAM 역할은 액세스되는 Amazon S3 버킷에서 LIST 작업을 수행하고 버킷에 포함된 Amazon S3 객체에 대한 GET 작업을 수행할 수 있는 권한이 있어야 합니다. 데이터 공유를 위해 AWS Glue Data Catalog를 사용하여 데이터베이스를 생성하고 IAM_ROLE을 사용하는 방법에 대해 자세히 알아보려면 소비자로서 Lake Formation에서 관리하는 데이터 공유 사용을 참조하세요.

다음은 단일 ARN에 대한 IAM_ROLE 파라미터의 구문을 보여줍니다.

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-name>'
```

역할을 함께 묶어 클러스터가 다른 계정에 속한 다른 IAM 역할을 수임하도록 할 수 있습니다. 최대 10개의 역할을 함께 묶을 수 있습니다. 자세한 내용은 Amazon Redshift Spectrum에서 IAM 역할 연결 단원을 참조하십시오.

이 IAM 역할에 다음과 유사한 IAM 권한 정책을 연결합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessSecret",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecretVersionIds"
            ],
            "Resource": "arn:aws:secretsmanager:us-west-2:123456789012:secret:my-
rds-secret-VNenFy"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
```

```
            "Action": [
                "secretsmanager:GetRandomPassword",
                "secretsmanager:ListSecrets"
            ],
            "Resource": "*"
        }
    ]
}
```

연합 쿼리에 사용할 IAM 역할을 생성하는 단계는 연합 쿼리 사용을 위해 비밀 및 IAM 역할 생성 섹션을 참조하세요.

> **Note**
>
> 연결된 역할 목록에 공백을 포함하지 마십시오.

다음은 세 역할을 함께 묶기 위한 구문을 나타낸 것입니다.

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-1-name>,arn:aws:iam::<aws-account-id>:role/<role-2-name>,arn:aws:iam::<aws-account-id>:role/<role-3-name>'
```

LIMIT row_limit

반환되는 행의 수를 제한하는 절입니다. 여기서 row_limit은 반환할 최대 행 수입니다. row_limit는 0~10,000일 수 있습니다.

예시

다음 예는 계정 ID 123456789012의 모든 데이터 카탈로그 데이터베이스를 표시합니다.

```
SHOW DATABASES FROM DATA CATALOG ACCOUNT '123456789012'

  catalog_id  | database_name |                          database_arn
 |     type     |                                          target_database
                            | location | parameters
--------------+---------------+------------------------------------------------------------
+--------------
+-------------------------------------------------------------------------------------------
+----------+------------
```

```
 123456789012 |   database1   | arn:aws:glue:us-east-1:123456789012:database/database1
 | Data Catalog |
                             |          |
 123456789012 |   database2   | arn:aws:glue:us-east-1:123456789012:database/database2
 | Data Catalog | arn:aws:redshift:us-
east-1:123456789012:datashare:035c45ea-61ce-86f0-8b75-19ac6102c3b7/database2 |
 |
```

다음 예는 IAM 역할의 보안 인증 정보를 사용하여 계정 ID 123456789012의 모든 데이터 카탈로그 데이터베이스를 표시하는 방법을 보여줍니다.

```
SHOW DATABASES FROM DATA CATALOG ACCOUNT '123456789012' IAM_ROLE default;
```

```
SHOW DATABASES FROM DATA CATALOG ACCOUNT '123456789012' IAM_ROLE <iam-role-arn>;
```

다음 예시에서는 연결된 Amazon Redshift 데이터 웨어하우스의 모든 데이터베이스를 표시합니다.

```
SHOW DATABASES

database_name  | database_owner | database_type        | database_acl | parameters |
 database_isolation_level
---------------+----------------+----------------------+--------------+------------
+--------------------
awsdatacatalog | 1              | auto mounted catalog | NULL         | UNKNOWN    |
 UNKNOWN
dev            | 1              | local                | NULL         | NULL       |
 Snapshot Isolation
```

SHOW GRANTS

사용자, 역할 또는 객체에 대한 권한을 표시합니다. 객체는 데이터베이스, 스키마, 테이블 또는 함수일 수 있습니다.

구문

```
SHOW GRANTS ON
{DATABASE database_name | FUNCTION function_name | SCHEMA schema_name |
 TABLE table_name}
```

```
[FOR {username | ROLE role_name | PUBLIC}]
[LIMIT row_limit]
```

파라미터

database_name

권한 부여를 표시할 데이터베이스의 이름입니다.

function_name

권한 부여를 표시할 함수의 이름입니다.

schema_name

권한 부여를 표시할 스키마의 이름입니다.

table_name

권한 부여를 표시할 테이블의 이름입니다.

FOR username

사용자에 대한 권한 부여를 표시함을 나타냅니다.

ROLE role_name

역할에 대한 권한 부여를 표시함을 나타냅니다.

FOR PUBLIC

PUBLIC에 대한 권한 부여를 표시함을 나타냅니다.

row_limit

반환할 최대 열 수입니다. row_limit는 0~10,000일 수 있습니다.

예시

다음 예시에서는 이름이 dev인 데이터베이스의 모든 권한을 표시합니다.

```
SHOW GRANTS ON DATABASE dev;

 database_name | privilege_type | identity_id | identity_name | identity_type |
 admin_option | privilege_scope
---------------+----------------+-------------+---------------+---------------
+--------------+-----------------
```

```
 dev           | TRUNCATE       |         101 | alice         | user          | f
      | TABLES
 dev           | DROP           |         101 | alice         | user          | f
      | TABLES
 dev           | INSERT         |         101 | alice         | user          | f
      | TABLES
 dev           | ALTER          |         101 | alice         | user          | f
      | TABLES
 dev           | TEMP           |           0 | public        | public        | f
      | DATABASE
 dev           | DELETE         |         101 | alice         | user          | f
      | TABLES
 dev           | SELECT         |         101 | alice         | user          | f
      | TABLES
 dev           | UPDATE         |         101 | alice         | user          | f
      | TABLES
 dev           | REFERENCES     |         101 | alice         | user          | f
      | TABLES
(9 rows)
```

다음 명령은 이름이 demo인 스키마에 대한 모든 권한을 보여줍니다.

```
SHOW GRANTS ON SCHEMA demo;

 schema_name | object_name | object_type | privilege_type | identity_id | identity_name
 | identity_type | admin_option | privilege_scope
-------------+-------------+-------------+----------------+-------------
+---------------+---------------+--------------+-----------------
 demo        | demo        | SCHEMA      | ALTER          |         101 | alice
 | user          | f            | SCHEMA
 demo        | demo        | SCHEMA      | DROP           |         101 | alice
 | user          | f            | SCHEMA
 demo        | demo        | SCHEMA      | USAGE          |         101 | alice
 | user          | f            | SCHEMA
 demo        | demo        | SCHEMA      | CREATE         |         101 | alice
 | user          | f            | SCHEMA
(4 rows)
```

다음 명령은 이름이 alice인 사용자에 대한 모든 권한을 보여줍니다.

```
SHOW GRANTS FOR alice;
```

```
 database_name | schema_name | object_name | object_type | privilege_type | identity_id
 | identity_name | identity_type | privilege_scope
---------------+-------------+-------------+-------------+----------------
+-------------+--------------+---------------+-----------------
 dev           |             |             | DATABASE    | INSERT         |         101
 | alice         | user          | TABLES
 dev           |             |             | DATABASE    | SELECT         |         101
 | alice         | user          | TABLES
 dev           |             |             | DATABASE    | UPDATE         |         101
 | alice         | user          | TABLES
 dev           |             |             | DATABASE    | DELETE         |         101
 | alice         | user          | TABLES
 dev           |             |             | DATABASE    | REFERENCES     |         101
 | alice         | user          | TABLES
 dev           |             |             | DATABASE    | DROP           |         101
 | alice         | user          | TABLES
 dev           |             |             | DATABASE    | TRUNCATE       |         101
 | alice         | user          | TABLES
 dev           |             |             | DATABASE    | ALTER          |         101
 | alice         | user          | TABLES
 dev           | public      | t1          | TABLE       | INSERT         |         101
 | alice         | user          | TABLE
 dev           | public      | t1          | TABLE       | SELECT         |         101
 | alice         | user          | TABLE
 dev           | public      | t1          | TABLE       | UPDATE         |         101
 | alice         | user          | TABLE
 dev           | public      | t1          | TABLE       | DELETE         |         101
 | alice         | user          | TABLE
 dev           | public      | t1          | TABLE       | RULE           |         101
 | alice         | user          | TABLE
 dev           | public      | t1          | TABLE       | REFERENCES     |         101
 | alice         | user          | TABLE
 dev           | public      | t1          | TABLE       | TRIGGER        |         101
 | alice         | user          | TABLE
 dev           | public      | t1          | TABLE       | DROP           |         101
 | alice         | user          | TABLE
 dev           | public      | t1          | TABLE       | TRUNCATE       |         101
 | alice         | user          | TABLE
 dev           | public      | t1          | TABLE       | ALTER          |         101
 | alice         | user          | TABLE
 dev           | demo        |             | SCHEMA      | USAGE          |         101
 | alice         | user          | SCHEMA
 dev           | demo        |             | SCHEMA      | CREATE         |         101
 | alice         | user          | SCHEMA
```

```
 dev           | demo        |             | SCHEMA      | DROP           |         101
 | alice         | user          | SCHEMA
 dev           | demo        |             | SCHEMA      | ALTER          |         101
 | alice         | user          | SCHEMA
(22 rows)
```

SHOW MODEL

상태, 모델을 생성하는 데 사용된 파라미터 및 입력 인수 형식이 있는 예측 함수를 포함하여 기계 학습 모델에 대한 유용한 정보를 표시합니다. SHOW MODEL의 정보를 사용하여 모델을 재생성할 수 있습니다. 기본 테이블이 변경된 경우 동일한 SQL 문으로 CREATE MODEL을 실행하면 다른 모델이 생성됩니다. SHOW MODEL이 반환하는 정보는 모델 소유자와 EXECUTE 권한을 가진 사용자에 따라 다릅니다. SHOW MODEL은 모델이 Amazon Redshift에서 훈련된 경우 또는 BYOM 모델인 경우 다른 출력을 표시합니다.

구문

```
SHOW MODEL ( ALL | model_name )
```

파라미터

ALL

사용자가 사용할 수 있는 모든 모델과 해당 스키마를 반환합니다.

model_name

모델의 이름입니다. 스키마의 모델 이름은 고유해야 합니다.

사용 노트

SHOW MODEL 명령은 다음을 반환합니다.

- 모델 이름입니다.
- 모델이 생성된 스키마입니다.
- 모델의 소유자입니다.
- 모델 생성 시간입니다.
- READY, TRAINING 또는 FAILED와 같은 모델 상태입니다.

- 실패한 모델에 대한 이유 메시지입니다.
- 모델이 훈련을 마친 경우 검증 오류입니다.
- 비 BYOM 접근 방식에 대한 모델을 도출하는 데 필요한 예상 비용입니다. 모델 소유자만 이 정보를 볼 수 있습니다.
- 사용자 지정 파라미터와 해당 값의 목록입니다. 특히 다음 항목을 포함합니다.
  - 지정된 TARGET 열.
  - 모델 유형, AUTO 또는 XGBoost.
  - REGRESSION, BINARY_CLASSIFICATION, MULTICLASS_CLASSIFICATION 등의 문제 유형. 이 파라미터는 AUTO에만 해당됩니다.
  - 모델을 생성한 Amazon SageMaker AI 훈련 작업 또는 Amazon SageMaker AI Autopilot 작업의 이름입니다. 이 작업 이름을 사용하여 Amazon SageMaker AI에서 모델에 대한 자세한 정보를 찾을 수 있습니다.
  - MSE, F1, 정확도와 같은 목표. 이 파라미터는 AUTO에만 해당됩니다.
  - 생성된 함수의 이름.
  - 추론 유형, 로컬 또는 원격.
  - 예측 함수 입력 인수.
  - 기존 보유 모델 사용(BYOM)이 아닌 모델에 대한 예측 함수 입력 인수 형식.
  - 예측 함수의 반환 유형. 이 파라미터는 BYOM에만 해당됩니다.
  - 원격 추론이 포함된 BYOM 모델에 대한 Amazon SageMaker AI 엔드포인트의 이름입니다.
  - IAM 역할. 모델 소유자만 이를 볼 수 있습니다.
  - 사용되는 S3 버킷. 모델 소유자만 이를 볼 수 있습니다.
  - AWS KMS 키(제공된 경우). 모델 소유자만 이를 볼 수 있습니다.
  - 모델을 실행할 수 있는 최대 시간입니다.
- 모델 유형이 AUTO가 아닌 경우 Amazon Redshift는 제공된 하이퍼파라미터와 해당 값 목록도 표시합니다.

또한 pg_proc 등의 다른 카탈로그 테이블에서 SHOW MODEL이 제공하는 일부 정보를 볼 수 있습니다. Amazon Redshift는 pg_proc 카탈로그 테이블에 등록된 예측 함수에 대한 정보를 반환합니다. 이 정보에는 예측 함수에 대한 입력 인수 이름과 해당 유형이 포함됩니다. Amazon Redshift는 SHOW MODEL 명령에서 동일한 정보를 반환합니다.

```
SELECT * FROM pg_proc WHERE proname ILIKE '%<function_name>%';
```

예시

다음은 모델 출력을 보여주는 예입니다.

```
SHOW MODEL ALL;

Schema Name |  Model Name
------------+--------------
 public     | customer_churn
```

customer_churn의 소유자는 다음과 같은 출력을 볼 수 있습니다. EXECUTE 권한만 있는 사용자는 IAM 역할, Amazon S3 버킷 및 모드의 예상 비용을 볼 수 없습니다.

```
SHOW MODEL customer_churn;

       Key                  |           Value
----------------------------+-----------------------------------
 Model Name                 | customer_churn
 Schema Name                | public
 Owner                      | 'owner'
 Creation Time              | Sat, 15.01.2000 14:45:20
 Model State                | READY
 validation:F1              | 0.855
 Estimated Cost             | 5.7
                            |
 TRAINING DATA:             |
 Table                      | customer_data
 Target Column              | CHURN
                            |
 PARAMETERS:                |
 Model Type                 | auto
 Problem Type               | binary_classification
 Objective                  | f1
 Function Name              | predict_churn
 Function Parameters        | age zip average_daily_spend average_daily_cases
 Function Parameter Types   | int int float float
 IAM Role                   | 'iam_role'
 KMS Key                    | 'kms_key'
 Max Runtime                | 36000
```

SHOW DATASHARES

동일한 계정 또는 여러 계정에서 클러스터의 인바운드 및 아웃바운드 공유를 표시합니다. datashare 이름을 지정하지 않으면 Amazon Redshift는 클러스터의 모든 데이터베이스에 있는 모든 datashare를 표시합니다. ALTER 및 SHARE 권한이 있는 사용자는 권한을 갖고 있는 공유를 볼 수 있습니다.

구문

```
SHOW DATASHARES [ LIKE 'namepattern' ]
```

파라미터

LIKE

지정된 이름 패턴을 datashare에 대한 설명과 비교하는 선택적 절입니다. 이 절을 사용하면 Amazon Redshift는 지정된 이름 패턴과 이름이 일치하는 datashare만 표시합니다.

namepattern

요청된 datashare의 이름 또는 와일드카드 문자를 사용하여 일치시킬 이름의 일부입니다.

예시

다음 예에서는 클러스터의 인바운드 공유와 아웃바운드 공유를 표시합니다.

```
SHOW DATASHARES;
SHOW DATASHARES LIKE 'sales%';

share_name   | share_owner | source_database | consumer_database | share_type |
 createdate          | is_publicaccessible | share_acl | producer_account |
 producer_namespace
-------------+-------------+-----------------+-------------------+------------
+---------------------+---------------------+-----------+------------------
+--------------------------------------
'salesshare' | 100         | dev             |                   | outbound
 | 2020-12-09 01:22:54.| False               |           |   123456789012   |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d
```

SHOW PROCEDURE

서명을 포함하여 제공된 저장 프로시저의 정의를 보여 줍니다. SHOW PROCEDURE의 출력을 사용하여 저장 프로시저를 다시 생성할 수 있습니다.

구문

```
SHOW PROCEDURE sp_name [( [ [ argname ] [ argmode ] argtype [, ...] ] )]
```

파라미터

sp_name

표시할 프로시저의 이름입니다.

[argname] [argmode] argtype

저장 프로시저를 식별할 입력 인수 형식입니다. OUT 인수를 포함하여 전체 인수 데이터 형식을 포함시킬 수도 있습니다. 저장 프로시저의 이름이 고유한 경우(즉, 오버로드되지 않은 경우) 이 부분은 선택 사항입니다.

예시

다음 예제에서는 `test_spl2` 프로시저의 정의를 보여 줍니다.

```
show procedure test_sp2(int, varchar);
                                        Stored Procedure Definition
------------------------------------------------------------------------------------------------
CREATE OR REPLACE PROCEDURE public.test_sp2(f1 integer, INOUT f2 character varying, OUT
 character varying)
LANGUAGE plpgsql
AS $_$
DECLARE
out_var alias for $3;
loop_var int;
BEGIN
IF f1 is null OR f2 is null THEN
RAISE EXCEPTION 'input cannot be null';
END IF;
CREATE TEMP TABLE etl(a int, b varchar);
FOR loop_var IN 1..f1 LOOP
insert into etl values (loop_var, f2);
```

```
f2 := f2 || '+' || f2;
END LOOP;
SELECT INTO out_var count(*) from etl;
END;
$_$

(1 row)
```

SHOW SCHEMAS

데이터베이스의 스키마 목록과 일부 스키마 속성을 표시합니다.

각 출력 행은 데이터베이스 이름, 스키마 이름, 스키마 소유자, 스키마 유형, 스키마 ACL, 소스 데이터베이스 및 스키마 옵션으로 구성됩니다. 이들 속성에 대한 자세한 내용은 SVV_ALL_SCHEMAS 섹션을 참조하세요.

SHOW SCHEMAS 명령으로 인해 10,000개 이상의 스키마가 표시될 수 있는 경우 오류가 반환됩니다.

구문

```
SHOW SCHEMAS FROM DATABASE database_name [LIKE 'filter_pattern'] [LIMIT row_limit ]
```

파라미터

database_name

나열할 테이블이 포함된 데이터베이스의 이름입니다.

AWS Glue Data Catalog에 테이블을 표시하려면 데이터베이스 이름으로 (awsdatacatalog)를 지정하고 시스템 구성 data_catalog_auto_mount가 true로 설정되어 있는지 확인합니다. 자세한 내용은 ALTER SYSTEM 단원을 참조하십시오.

filter_pattern

스키마 이름과 일치하는 패턴이 있는 유효한 UTF-8 문자 표현식입니다. LIKE' 옵션은 다음과 같은 패턴 일치 메타문자를 지원하는 대/소문자 구분 일치를 수행합니다:

메타문자	설명
%	0개 이상의 문자 시퀀스를 일치시킵니다.
_	모든 문자를 일치시킵니다.

filter_pattern에 메타 문자가 포함되어 있지 않으면 패턴이 문자열 자체만 의미합니다. 이런 경우에는 LIKE가 등호 연산자와 동일한 역할을 합니다.

row_limit

반환할 최대 열 수입니다. row_limit는 0~10,000일 수 있습니다.

예시

다음 예제에서는 dev라는 Amazon Redshift 데이터베이스의 스키마를 보여줍니다.

```
SHOW SCHEMAS FROM DATABASE dev;

 database_name |     schema_name      | schema_owner | schema_type |         schema_acl
          | source_database | schema_option
---------------+----------------------+--------------+-------------
+----------------------------+-----------------+---------------
 dev           | pg_automv            |            1 | local       |
          |                 |
 dev           | pg_catalog           |            1 | local       | jpuser=UC/
jpuser~=U/jpuser  |                 |
 dev           | public               |            1 | local       | jpuser=UC/
jpuser~=UC/jpuser |                 |
 dev           | information_schema   |            1 | local       | jpuser=UC/
jpuser~=U/jpuser  |                 |
 dev           | schemad79cd6d93bf043 |            1 | local       |
          |                 |
```

다음 예제에서는 awsdatacatalog라는 AWS Glue Data Catalog 데이터베이스의 스키마를 보여줍니다. 최대 출력 행 수는 5입니다.

```
SHOW SCHEMAS FROM DATABASE awsdatacatalog LIMIT 5;

 database_name  |     schema_name      | schema_owner | schema_type | schema_acl |
 source_database | schema_option
----------------+----------------------+--------------+-------------+------------
+-----------------+---------------
 awsdatacatalog | 000_too_many_glue_db |              | EXTERNAL    |            |
            |
 awsdatacatalog | 123_default          |              | EXTERNAL    |            |
            |
 awsdatacatalog | adhoc                |              | EXTERNAL    |            |
            |
```

```
 awsdatacatalog | all_shapes_10mb      |             | EXTERNAL    |           |
           |
 awsdatacatalog | all_shapes_1g        |             | EXTERNAL    |           |
           |
```

SHOW TABLE

테이블 속성, 테이블 제약 조건, 열 속성 및 열 제약 조건을 포함하여 테이블의 정의를 표시합니다. SHOW TABLE 문의 출력을 사용하여 테이블을 다시 생성할 수 있습니다.

테이블 생성에 대한 자세한 내용은 CREATE TABLE 섹션을 참조하세요.

구문

```
SHOW TABLE [schema_name.]table_name
```

파라미터

schema_name

(옵션) 관련 스키마의 이름입니다.

table_name

표시할 테이블의 이름입니다.

예시

다음은 `sales` 테이블에 대한 SHOW TABLE 출력의 예입니다.

```
show table sales;
```

```
CREATE TABLE public.sales (
salesid integer NOT NULL ENCODE az64,
listid integer NOT NULL ENCODE az64 distkey,
sellerid integer NOT NULL ENCODE az64,
buyerid integer NOT NULL ENCODE az64,
eventid integer NOT NULL ENCODE az64,
dateid smallint NOT NULL,
qtysold smallint NOT NULL ENCODE az64,
pricepaid numeric(8,2) ENCODE az64,
```

```
commission numeric(8,2) ENCODE az64,
saletime timestamp without time zone ENCODE az64
)
DISTSTYLE KEY SORTKEY ( dateid );
```

다음은 스키마 public의 테이블 category에 대한 SHOW TABLE 출력의 예입니다.

```
show table public.category;
```

```
CREATE TABLE public.category (
catid smallint NOT NULL distkey,
catgroup character varying(10) ENCODE lzo,
catname character varying(10) ENCODE lzo,
catdesc character varying(50) ENCODE lzo
) DISTSTYLE KEY SORTKEY ( catid );
```

다음 예에서는 기본 키가 있는 테이블 foo를 생성합니다.

```
create table foo(a int PRIMARY KEY, b int);
```

SHOW TABLE 결과는 foo 테이블의 모든 속성과 함께 create 문을 표시합니다.

```
show table foo;
```

```
CREATE TABLE public.foo ( a integer NOT NULL ENCODE az64, b integer ENCODE az64,
 PRIMARY KEY (a) ) DISTSTYLE AUTO;
```

SHOW TABLES

스키마의 테이블 목록과 일부 테이블 속성을 표시합니다.

각 출력 행은 데이터베이스 이름, 스키마 이름, 테이블 이름, 테이블 유형, 테이블 ACL 및 설명으로 구성됩니다. 이들 속성에 대한 자세한 내용은 SVV_ALL_TABLES 섹션을 참조하세요.

SHOW TABLES 명령으로 인해 10,000개 이상의 테이블이 반환되는 경우 오류가 반환됩니다.

구문

```
SHOW TABLES FROM SCHEMA database_name.schema_name [LIKE 'filter_pattern']
 [LIMIT row_limit ]
```

파라미터

database_name

나열할 테이블이 포함된 데이터베이스의 이름입니다.

AWS Glue Data Catalog에 테이블을 표시하려면 데이터베이스 이름으로 (awsdatacatalog)를 지정하고 시스템 구성 data_catalog_auto_mount가 true로 설정되어 있는지 확인합니다. 자세한 내용은 ALTER SYSTEM 단원을 참조하십시오.

schema_name

나열할 테이블이 포함된 스키마의 이름입니다.

AWS Glue Data Catalog 테이블을 표시하려면 AWS Glue 데이터베이스 이름을 스키마 이름으로 제공하세요.

filter_pattern

테이블 이름과 일치하는 패턴이 있는 유효한 UTF-8 문자 표현식입니다. LIKE' 옵션은 다음과 같은 패턴 일치 메타문자를 지원하는 대/소문자 구분 일치를 수행합니다:

메타문자	설명
%	0개 이상의 문자 시퀀스를 일치시킵니다.
_	모든 문자를 일치시킵니다.

filter_pattern에 메타 문자가 포함되어 있지 않으면 패턴이 문자열 자체만 의미합니다. 이런 경우에는 LIKE가 등호 연산자와 동일한 역할을 합니다.

row_limit

반환할 최대 열 수입니다. row_limit는 0~10,000일 수 있습니다.

예시

다음 예제에서는 public 스키마에 있는 dev라는 이름의 Amazon Redshift 데이터베이스의 테이블을 보여줍니다.

```
SHOW TABLES FROM SCHEMA dev.public;
```

```
 database_name | schema_name | table_name | table_type | table_acl | remarks
---------------+-------------+------------+------------+-----------+---------
 dev           | public      | tb         | TABLE      |           |
 dev           | public      | tb2        | TABLE      |           |
 dev           | public      | tb3        | TABLE      |           |
```

다음 예제에서는 batman 스키마에 있는 awsdatacatalog라는 이름의 AWS Glue Data Catalog 데이터베이스에 있는 테이블을 보여줍니다.

```
SHOW TABLES FROM SCHEMA awsdatacatalog.batman;

 database_name  | schema_name |    table_name    | table_type | table_acl | remarks
----------------+-------------+------------------+------------+-----------+---------
 awsdatacatalog | batman      | nation           | EXTERNAL   |           |
 awsdatacatalog | batman      | part             | EXTERNAL   |           |
 awsdatacatalog | batman      | partsupp         | EXTERNAL   |           |
 awsdatacatalog | batman      | region           | EXTERNAL   |           |
 awsdatacatalog | batman      | supplier         | EXTERNAL   |           |
 awsdatacatalog | batman      | automount_nation | EXTERNAL   |           |
```

SHOW VIEW

구체화된 뷰와 후기 바인딩 뷰를 포함하여 뷰의 정의를 표시합니다. SHOW VIEW 문의 출력을 사용하여 뷰를 다시 생성할 수 있습니다.

구문

```
SHOW VIEW [schema_name.]view_name
```

파라미터

schema_name

(옵션) 관련 스키마의 이름입니다.

view_name

표시할 뷰의 이름입니다.

예시

다음은 뷰 LA_Venues_v에 대한 뷰 정의입니다.

```
create view LA_Venues_v as select * from venue where venuecity='Los Angeles';
```

다음은 앞에서 정의한 뷰에 대한 SHOW VIEW 명령 및 출력의 예입니다.

```
show view LA_Venues_v;
```

```
SELECT venue.venueid,
venue.venuename,
venue.venuecity,
venue.venuestate,
venue.venueseats
FROM venue WHERE ((venue.venuecity)::text = 'Los Angeles'::text);
```

다음은 스키마 public에서 뷰 public.Sports_v에 대한 뷰 정의입니다.

```
create view public.Sports_v as select * from category where catgroup='Sports';
```

다음은 앞에서 정의한 뷰에 대한 SHOW VIEW 명령 및 출력의 예입니다.

```
show view public.Sports_v;
```

```
SELECT category.catid,
category.catgroup,
category.catname,
category.catdesc
FROM category WHERE ((category.catgroup)::text = 'Sports'::text);
```

START TRANSACTION

BEGIN 함수의 동의어입니다.

BEGIN 섹션을 참조하세요.

TRUNCATE

테이블 스캔을 수행하지 않고 테이블에서 모든 행을 삭제합니다. 이 작업은 정규화되지 않은 DELETE 작업을 대신하여 더 빠르게 수행할 수 있는 수단입니다. TRUNCATE 명령을 실행하려면 테이블의 소유자 또는 수퍼유저이거나 TRUNCATE TABLE 권한이 부여되어야 합니다. 테이블을 잘라낼 수 있는 권한을 부여하려면 GRANT 명령을 사용합니다.

TRUNCATE는 DELETE보다 훨씬 더 효율적이며 VACUUM 및 ANALYZE가 필요하지 않습니다. 하지만 TRUNCATE는 이 명령이 실행되는 트랜잭션을 커밋합니다.

구문

```
TRUNCATE [ TABLE ] table_name
```

이 명령은 구체화된 뷰에서도 작동합니다.

```
TRUNCATE materialized_view_name
```

파라미터

TABLE

선택적 키워드입니다.

table_name

임시 또는 영구 테이블입니다. 테이블 소유자 또는 수퍼유저만이 테이블을 잘라낼 수 있습니다.

외래 키 제약 조건에서 참조되는 테이블을 포함한 어떤 테이블이든 잘라낼 수 있습니다.

잘라낸 후 테이블을 vacuum할 필요가 없습니다.

materialized_view_name

구체화된 뷰

구체화된 뷰로 스트리밍 모으기에 사용되는 구체화된 뷰를 잘라낼 수 있습니다.

사용 노트

TRUNCATE 명령은 명령이 실행되는 트랜잭션을 커밋하므로 TRUNCATE 작업을 롤백할 수 없고, TRUNCATE 명령은 스스로를 커밋할 때 다른 작업을 커밋할 수 있습니다.

예시

CATEGORY 테이블에서 모든 행을 삭제하려면 TRUNCATE 명령을 사용하십시오.

```
truncate category;
```

TRUNCATE 작업 롤백을 시도합니다.

```
begin;

truncate date;

rollback;

select count(*) from date;
count
-------
0
(1 row)
```

TRUNCATE 명령이 자동으로 커밋되었으므로 ROLLBACK 명령 후 DATE 테이블이 빈 상태로 남습니다.

다음 예제에서는 TRUNCATE 명령을 사용하여 구체화된 뷰에서 모든 행을 삭제합니다.

```
truncate my_materialized_view;
```

구체화된 뷰의 모든 레코드를 삭제하고 구체화된 뷰와 해당 스키마를 그대로 유지합니다. 쿼리에서 구체화된 뷰 이름은 샘플입니다.

UNLOAD

Amazon S3 서버 측 암호화(SSE-S3)를 사용하여 Amazon S3에서 하나 이상의 텍스트, JSON 또는 Apache Parquet 파일로 쿼리의 결과를 언로드합니다. AWS Key Management Service 키(SSE-KMS)를 사용하는 서버 측 암호화를 지정하거나 고객 관리형 키를 사용하는 클라이언트 측 암호화를 지정할 수도 있습니다.

기본적으로 언로드된 파일의 형식은 파이프로 구분된(|) 텍스트입니다.

MAXFILESIZE 파라미터를 설정하면 Amazon S3의 파일 크기를 비롯해 파일 수까지도 관리할 수 있습니다. S3 IP 범위가 허용 목록에 추가되었는지 확인합니다. 필요한 S3 IP 범위에 대한 자세한 내용은 네트워크 격리를 참조하세요.

이제 Amazon Redshift 쿼리 결과를 분석을 위한 효율적인 개방형 열 기반 스토리지 형식인 Apache Parquet의 Amazon S3 데이터 레이크에 언로드할 수 있습니다. Parquet 형식은 텍스트 형식에 비해 언로드 속도가 최대 2배 빠르고 Amazon S3에서 스토리지 사용량이 최대 6배 적습니다. 따라서 Amazon S3에서 Amazon S3 데이터 레이크로 데이터 변환 및 보강 작업의 결과물을 오픈 형식으로 저장할 수 있습니다. 그런 다음 Redshift Spectrum과 Amazon Athena, Amazon EMR, Amazon SageMaker AI 등의 기타 AWS 서비스를 사용하여 데이터를 분석할 수 있습니다.

UNLOAD 명령 사용에 대한 자세한 내용과 예제 시나리오는 Amazon Redshift에서 데이터 언로드 단원을 참조하세요.

필수 권한

UNLOAD 명령이 성공하려면 Amazon S3 위치에 쓸 수 있는 권한과 함께 데이터베이스의 데이터에 대해 최소 SELECT 권한이 필요합니다. UNLOAD 명령의 AWS 리소스에 액세스할 수 있는 권한에 대한 자세한 내용은 다른 AWS 리소스에 대한 액세스 권한 섹션을 참조하세요.

최소 권한 권한을 적용하려면 다음 권장 사항에 따라 명령을 실행하는 사용자에게 필요한 권한만 부여합니다.

- 사용자는 데이터에 대한 SELECT 권한이 있어야 합니다. 데이터베이스 권한을 제한하는 방법에 대한 자세한 내용은 GRANT 섹션을 참조하세요.
- 사용자는 AWS 계정의 Amazon S3 버킷에 쓸 IAM 역할을 수임할 권한이 필요합니다. 데이터베이스 사용자가 역할을 수임하도록 액세스를 제한하려면 Amazon Redshift 관리 안내서의 IAM 역할에 대한 액세스 제한을 참조하세요.
- 사용자는 Amazon S3 버킷에 대한 액세스 권한이 필요합니다. Amazon S3 버킷 정책을 사용하여 권한을 제한하려면 Amazon Simple Storage Service 사용 설명서의 Amazon S3의 버킷 정책을 참조하세요.

구문

```
UNLOAD ('select-statement')
TO 's3://object-path/name-prefix'
authorization
[ option, ...]

where authorization is
IAM_ROLE { default | 'arn:aws:iam::<AWS ##-id-1>:role/<role-name>[,arn:aws:iam::<AWS #
#-id-2>:role/<role-name>][,...]' }

where option is
| [ FORMAT [ AS ] ] CSV | PARQUET | JSON
| PARTITION BY ( column_name [, ... ] ) [ INCLUDE ]
| MANIFEST [ VERBOSE ]
| HEADER
| DELIMITER [ AS ] 'delimiter-char'
| FIXEDWIDTH [ AS ] 'fixedwidth-spec'
| ENCRYPTED [ AUTO ]
```

```
| BZIP2
| GZIP
| ZSTD
| ADDQUOTES
| NULL [ AS ] 'null-string'
| ESCAPE
| ALLOWOVERWRITE
| CLEANPATH
| PARALLEL [ { ON | TRUE } | { OFF | FALSE } ]
| MAXFILESIZE [AS] max-size [ MB | GB ]
| ROWGROUPSIZE [AS] size [ MB | GB ]
| REGION [AS] 'aws-region' }
| EXTENSION 'extension-name'
```

파라미터

('select-statement')

SELECT 쿼리입니다. 쿼리의 결과가 언로드됩니다. 대부분의 경우 쿼리에 ORDER BY 절을 지정하여 정렬된 순서대로 데이터를 언로드할 만한 가치가 있습니다. 이런 접근 방식을 취하면 데이터를 다시 로드할 때 데이터 정렬 소요 시간이 단축됩니다.

다음에 표시된 것처럼 쿼리를 작은따옴표로 묶어야 합니다.

```
('select * from venue order by venueid')
```

> **Note**
>
> 쿼리에 따옴표가 포함된 경우(예: 리터럴 값을 묶기 위해) 리터럴을 2개의 작은따옴표 집합 사이에 넣습니다. 쿼리도 작은따옴표로 묶어야 합니다.
>
> ```
> ('select * from venue where venuestate=''NV''')
> ```

TO 's3://object-path/name-prefix'

Amazon S3에서 Amazon Redshift가 출력 파일 객체(MANIFEST가 지정되어 있는 경우 매니페스트 파일 포함)를 작성할 위치의 전체 경로(버킷 이름 포함)입니다. 객체 이름에는 name-prefix가 접두사로 붙습니다. PARTITION BY를 사용하면 필요에 따라 슬래시(/)가 name-prefix 값의 끝에 자동으

로 추가됩니다. 추가적인 보안을 위해, UNLOAD는 HTTPS 연결을 사용하여 Amazon S3에 연결합니다. 기본적으로, UNLOAD는 조각당 하나 이상의 파일을 작성합니다. UNLOAD는 다음과 같이 조각 번호와 부품 번호를 지정된 이름 접두사에 추가합니다.

<object-path>/*<name-prefix>*<*slice-number*>_part_<*part-number*>.

MANIFEST가 지정되어 있는 경우 다음과 같이 매니페스트 파일이 작성됩니다.

<object_path>/*<name_prefix>*manifest.

PARALLEL을 OFF로 지정하면 데이터 파일은 다음과 같이 기록됩니다.

<object_path>/*<name_prefix>*<*part-number*>.

UNLOAD는 Amazon S3 서버 측 암호화(SSE)를 사용하여 암호화된 파일을 자동 생성합니다(MANIFEST가 사용되는 경우 매니페스트 파일 포함). COPY 명령은 로드 작업 중에 서버 측 암호화 파일을 자동으로 읽습니다. Amazon S3 콘솔 또는 API를 사용하여 버킷에서 서버 측 암호화 파일을 투명하게 다운로드할 수 있습니다. 자세한 내용은 서버 측 암호화를 사용하여 데이터 보호 섹션을 참조하세요.

Amazon S3 클라이언트 측 암호화를 사용하려면 ENCRYPTED 옵션을 지정합니다.

> **Important**
>
> Amazon S3 버킷이 Amazon Redshift 데이터베이스와 같은 AWS 리전에 있지 않을 때는 REGION이 필요합니다.

권한 부여

UNLOAD 명령으로 Amazon S3에 데이터를 쓰려면 권한 부여가 필요합니다. UNLOAD 명령은 COPY 명령이 권한 부여를 위해 사용하는 것과 동일한 파라미터를 사용합니다. 자세한 내용은 COPY 명령 구문 참조 설명서에서 권한 부여 파라미터 섹션을 참조하세요.

IAM_ROLE { default | 'arn:aws:iam::*<AWS ##-id-1>*:role/*<role-name>*'

기본 키워드를 사용하여 UNLOAD 명령이 실행될 때 Amazon Redshift에서 기본값으로 설정되고 클러스터와 연결된 IAM 역할을 사용하도록 합니다.

클러스터가 인증 및 권한 부여에 사용하는 IAM 역할의 Amazon 리소스 이름(ARN)을 사용합니다. IAM_ROLE을 지정하면 ACCESS_KEY_ID 및 SECRET_ACCESS_KEY, SESSION_TOKEN 또는

CREDENTIALS는 사용할 수 없습니다. IAM_ROLE을 연결할 수 있습니다. 자세한 내용은 Amazon Redshift 관리 가이드의 IAM 역할 연결을 참조하세요.

[FORMAT [AS]] CSV | PARQUET | JSON

기본 형식을 재정의하는 언로드 형식을 지정하는 키워드입니다.

CSV의 경우 쉼표(,) 문자를 기본 구분 기호로 사용하여 텍스트 파일을 CSV 형식으로 언로드합니다. 필드에 구분 기호, 큰따옴표, 줄 바꿈 또는 캐리지 리턴이 포함된 경우 언로드된 파일의 필드가 큰따옴표로 묶입니다. 데이터 필드 내의 큰따옴표는 추가 큰따옴표로 이스케이프됩니다. 행 0개가 언로드되면 Amazon Redshift는 빈 Amazon S3 객체를 기록할 수 있습니다.

PARQUET의 경우 Apache Parquet 버전 1.0 형식으로 파일에 언로드합니다. 기본적으로 각 행 그룹은 SNAPPY 압축을 사용하여 압축됩니다. Apache Parquet 형식에 대한 자세한 내용은 Parquet을 참조하십시오.

JSON을 사용하는 경우 쿼리 결과의 전체 레코드를 나타내는 JSON 객체가 각 행에 포함된 JSON 파일로 언로드합니다. Amazon Redshift는 쿼리 결과에 SUPER 열이 포함된 경우 중첩된 JSON 작성을 지원합니다. 유효한 JSON 객체를 만들려면 쿼리의 각 열 이름이 고유해야 합니다. JSON 파일에서 부울 값은 `t` 또는 `f`로 언로드되고 NULL 값은 `null`로 언로드됩니다. 행 0개가 언로드되면 Amazon Redshift는 Amazon S3 객체를 기록하지 않습니다.

FORMAT 및 AS 키워드는 선택 사항입니다. CSV는 FIXEDWIDTH 또는 ADDQUOTES와 함께 사용할 수 없습니다. PARQUET은 DELIMITER, FIXEDWIDTH, ADDQUOTES, ESCAPE, NULL AS, HEADER, GZIP, BZIP2 또는 ZSTD와 함께 사용할 수 없습니다. PARQUET with ENCRYPTED는 AWS Key Management Service 키(SSE-KMS)를 사용하는 서버측 암호화에서만 지원됩니다. JSON은 DELIMITER, HEADER, FIXEDWIDTH, ADDQUOTES, ESCAPE 또는 NULL AS와 함께 사용할 수 없습니다.

PARTITION BY (column_name [, ...]) [INCLUDE]

언로드 작업을 위한 파티션 키를 지정합니다. UNLOAD는 Apache Hive 규칙에 따라 파티션 키 값을 기반으로 출력 파일을 파티션 폴더로 자동 분할합니다. 예를 들어 파티션 연도가 2019년이고 월이 9월인 Parquet 파일의 접두사는 다음과 같습니다. `s3://amzn-s3-demo-bucket/my_prefix/year=2019/month=September/000.parquet`.

column_name의 값은 언로드 중인 쿼리 결과의 열이어야 합니다.

INCLUDE 옵션을 사용하여 PARTITION BY를 지정하면 파티션 열이 언로드된 파일에서 제거되지 않습니다.

Amazon Redshift는 PARTITION BY 절에서 문자열 리터럴을 지원하지 않습니다.

MANIFEST [VERBOSE]

UNLOAD 프로세스에 의해 생성되는 데이터 파일의 세부 정보를 명시적으로 나열하는 매니페스트 파일을 생성합니다. 매니페스트란 Amazon S3에 작성한 개별 파일의 URL을 나열한 JSON 형식의 텍스트 파일을 말합니다.

MANIFEST가 VERBOSE 옵션을 사용해 지정된 경우 매니페스트에는 다음 세부 정보가 포함됩니다.

- 열 이름 및 데이터 형식 그리고 CHAR, VARCHAR 또는 NUMERIC 데이터 형식의 경우 각 열에 대한 차원. CHAR 및 VARCHAR 데이터 형식의 경우 차원은 길이입니다. DECIMAL 또는 NUMERIC 데이터 형식의 경우 차원은 정밀도 및 배율입니다.
- 각 파일로 언로드되는 행 수. HEADER 옵션이 지정되면 행 수에는 헤더 행이 포함됩니다.
- 언로드되는 모든 파일의 총 파일 크기 및 모든 파일로 언로드되는 총 행 수. HEADER 옵션이 지정되면 행 수에는 헤더 행이 포함됩니다.
- 작성자. 작성자는 항상 "Amazon Redshift"입니다.

VERBOSE는 MANIFEST 다음에만 지정할 수 있습니다.

매니페스트 파일은 같은 Amazon S3 경로 접두사에 `<object_path_prefix>manifest` 형식의 언로드 파일로 작성됩니다. 예를 들어 UNLOAD가 Amazon S3 경로 접두사 '`s3://amzn-s3-demo-bucket/venue_`'를 지정하는 경우 매니페스트 파일 위치는 '`s3://amzn-s3-demo-bucket/venue_manifest`'가 됩니다.

HEADER

각 출력 파일의 맨 위에 열 이름을 포함하는 헤더 줄을 추가합니다. 또한 CSV, DELIMITER, ADDQUOTES, ESCAPE와 같은 텍스트 변환 옵션이 헤더 줄에 적용됩니다. HEADER는 FIXEDWIDTH와 함께 사용할 수 없습니다.

DELIMITER AS 'delimiter_character'

파이프 문자(|), 쉼표(,) 또는 탭(\t) 같이 출력 파일에서 필드를 구분할 때 사용할 단일 ASCII 문자를 지정합니다. 텍스트 파일의 기본 구분 기호는 파이프 문자입니다. CSV 파일의 기본 구분 기호는 쉼표 문자입니다. AS 키워드는 옵션입니다. DELIMITER는 FIXEDWIDTH와 함께 사용할 수 없습니다. 데이터에 구분 기호 문자가 포함되는 경우 ESCAPE 옵션을 지정하여 구분 기호를 이스케이프하거나 ADDQUOTES를 사용하여 데이터를 큰따옴표로 묶어야 합니다. 또는 데이터에 포함되지 않은 구분 기호를 지정합니다.

FIXEDWIDTH 'fixedwidth_spec'

각각의 열 너비가 구분 기호로 구분되지 않고 고정된 길이인 파일로 데이터를 언로드합니다. fixedwidth_spec은 열 개수와 열의 너비를 지정하는 문자열입니다. AS 키워드는 옵션입니다. FIXEDWIDTH는 데이터를 자르지 않으므로 UNLOAD 문에서 각 열에 대한 사양은 최소한 그 열에 대해 가장 긴 항목의 길이만큼은 되어야 합니다. fixedwidth_spec의 형식이 아래에 표시되어 있습니다.

```
'colID1:colWidth1,colID2:colWidth2, ...'
```

FIXEDWIDTH는 DELIMITER 또는 HEADER와 함께 사용할 수 없습니다.

ENCRYPTED [AUTO]

Amazon S3의 출력 파일은 Amazon S3 서버 측 암호화 또는 클라이언트 측 암호화를 사용하여 암호화될 것임을 지정하는 절입니다. MANIFEST가 지정되어 있는 경우 매니페스트 파일도 암호화됩니다. 자세한 내용은 암호화된 데이터 파일 언로드 단원을 참조하십시오. ENCRYPTED 파라미터를 지정하지 않는 경우 UNLOAD는 AWS 관리형 암호화 키를 사용하는 Amazon S3 서버 측 암호화(SSE-S3)를 사용하여 암호화된 파일을 자동 생성합니다.

ENCRYPTED의 경우 AWS KMS 키를 사용한 서버 측 암호화(SSE-KMS)를 사용하여 Amazon S3에 언로드할 수 있습니다. 이 경우 KMS_KEY_ID 파라미터를 사용하여 키 ID를 제공합니다. KMS_KEY_ID 파라미터와 함께 CREDENTIALS 파라미터를 사용할 수 없습니다. KMS_KEY_ID를 사용하여 데이터에 대해 UNLOAD 명령을 실행하면 키를 지정하지 않고 동일한 데이터에 대해 COPY 작업을 수행할 수 있습니다.

고객이 제공하는 대칭 키를 사용한 클라이언트 측 암호화를 사용하여 Amazon S3에 언로드하려면 두 가지 방법 중 하나로 키를 제공합니다. 키를 제공하려면 MASTER_SYMMETRIC_KEY 파라미터 또는 CREDENTIALS 자격 증명 문자열의 `master_symmetric_key` 부분을 사용합니다. 루트 대칭 키를 사용하여 데이터를 언로드하는 경우 암호화된 데이터에 대해 COPY 작업을 수행할 때 동일한 키를 제공해야 합니다.

UNLOAD는 고객 입력식 키를 사용하는 Amazon S3 서버 측 암호화(SSE-C)는 지원하지 않습니다.

ENCRYPTED AUTO를 사용하는 경우 UNLOAD 명령은 대상 Amazon S3 버킷 속성에서 기본 AWS KMS 암호화 키를 가져와서 AWS KMS 키로 Amazon S3에 기록된 파일을 암호화합니다. 버킷에 기본 AWS KMS 암호화 키가 없는 경우 UNLOAD는 AWS에서 관리하는 암호화 키(SSE-S3)를 사용한 Amazon Redshift 서버 측 암호화를 사용하여 암호화된 파일을 자동으로 생성합니다. 이 옵션은 KMS_KEY_ID, MASTER_SYMMETRIC_KEY, 또는 master_symmetric_key가 포함된 CREDENTIALS와 함께 사용할 수 없습니다.

KMS_KEY_ID 'key-id'

Amazon S3에서 데이터 파일 암호화에 사용할 AWS Key Management Service(AWS KMS) 키의 키 ID를 지정합니다. 자세한 내용은 AWS Key Management Service란 무엇입니까?를 참조하십시오. KMS_KEY_ID를 지정하는 경우 ENCRYPTED 파라미터도 지정해야 합니다. KMS_KEY_ID를 지정하는 경우 CREDENTIALS 파라미터를 사용하여 인증할 수 없습니다. 대신에 IAM_ROLE 또는 ACCESS_KEY_ID and SECRET_ACCESS_KEY를 사용하십시오.

MASTER_SYMMETRIC_KEY 'root_key'

Amazon S3에서 데이터 파일을 암호화할 때 사용되는 루트 대칭 키를 지정합니다. MASTER_SYMMETRIC_KEY를 지정하는 경우 ENCRYPTED 파라미터도 지정해야 합니다. MASTER_SYMMETRIC_KEY를 CREDENTIALS 파라미터와 함께 사용할 수는 없습니다. 자세한 내용은 Amazon S3에서 암호화된 데이터 파일 로드 단원을 참조하십시오.

bzip2

조각당 하나 이상의 bzip2 압축 파일로 데이터를 언로드합니다. 각각의 결과 파일에는 `.bz2` 확장명이 추가됩니다.

GZIP

조각당 하나 이상의 gzip 압축 파일로 데이터를 언로드합니다. 각각의 결과 파일에는 `.gz` 확장명이 추가됩니다.

ZSTD

조각당 하나 이상의 Zstandard 압축 파일로 데이터를 언로드합니다. 각각의 결과 파일에는 `.zst` 확장명이 추가됩니다.

ADDQUOTES

Amazon Redshift가 구분 기호 자체를 포함하는 데이터 값을 언로드할 수 있도록 언로드되는 각각의 데이터 필드 주변에 따옴표를 배치합니다. 예를 들어, 구분 기호가 쉼표인 경우 다음 데이터를 성공적으로 언로드하고 다시 로드할 수 있습니다.

```
"1","Hello, World"
```

따옴표를 추가하지 않을 경우 문자열 `Hello, World`는 두 개의 개별 필드로 구문 분석됩니다.

일부 출력 형식은 ADDQUOTES를 지원하지 않습니다.

ADDQUOTES를 사용하는 경우 데이터를 다시 로드하면 COPY에 REMOVEQUOTES를 지정해야 합니다.

NULL AS 'null-string'

언로드 파일에서 null 값을 나타내는 문자열을 지정합니다. 이 옵션이 사용되는 경우 모든 출력 파일에는 선택한 데이터에서 발견되는 null 값의 자리에 지정된 문자열이 있습니다. 이 옵션을 지정하지 않으면 null 값이 다음으로 언로드됩니다.

- 구분 기호로 분리된 출력에 대해 제로 길이의 문자열
- 고정 너비 출력을 위한 공백 문자열

Null 문자열이 고정 폭의 언로드에 대해 지정되어 있고 출력 열의 폭이 null 문자열의 폭보다 작은 경우 다음 동작이 이루어집니다.

- 비문자 열의 경우 빈 필드가 출력됨
- 문자 열의 경우 오류가 보고됨

사용자 정의 문자열이 null 값을 나타내는 다른 데이터 형식과 달리 Amazon Redshift는 JSON 형식을 사용하여 SUPER 데이터 열을 내보내고 JSON 형식에 따라 null로 나타냅니다. 결과적으로 SUPER 데이터 열은 UNLOAD 명령에 사용된 NULL [AS] 옵션을 무시합니다.

ESCAPE

구분 기호로 분리된 언로드 파일에 있는 CHAR 및 VARCHAR 열의 경우, 다음 문자가 나올 때마다 그 앞에 이스케이프 문자(\)가 배치됩니다.

- 라인 피드: \n
- 캐리지 리턴: \r
- 언로드된 데이터에 대해 지정되는 구분 기호 문자.
- 이스케이프 문자: \
- 따옴표: '' 또는 \'(UNLOAD 명령에 ESCAPE와 ADDQUOTES가 모두 지정된 경우).

> **Important**
>
> COPY 문에 ESCAPE 옵션을 사용해 데이터를 로드한 경우에는 UNLOAD 명령에서도 ESCAPE 옵션을 지정하여 역수 출력 파일을 생성해야 합니다. 마찬가지로 ESCAPE 옵션을 사용해 UNLOAD 작업을 하는 경우에도 동일한 데이터에 대해 COPY 작업을 하면서 ESCAPE 파라미터를 사용해야 합니다.

ALLOWOVERWRITE

기본적으로, UNLOAD는 덮어쓸 가능성이 있는 파일을 찾을 경우에 실패합니다. ALLOWOVERWRITE가 지정된 경우 UNLOAD는 매니페스트 파일을 포함한 기존 파일을 덮어씁니다.

CLEANPATH

CLEANPATH 옵션은 지정된 위치로 파일을 언로드하기 전에 TO 절에 지정된 Amazon S3 경로에 있는 기존 파일을 제거합니다.

PARTITION BY 절을 포함하면 UNLOAD 작업으로 생성된 새 파일을 수신하기 위해 파티션 폴더에서만 기존 파일이 제거됩니다.

Amazon S3 버킷에 대해 `s3:DeleteObject` 권한이 있어야 합니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 Amazon S3의 정책 및 권한을 참조하세요. CLEANPATH 옵션을 사용하여 제거한 파일은 영구적으로 삭제되며 복구할 수 없습니다. 대상 Amazon S3 버킷에 버전 관리가 활성화되어 있는 경우 CLEANPATH 옵션을 사용하여 UNLOAD 작업을 수행해도 파일의 이전 버전은 제거되지 않습니다.

ALLOWOVERWRITE 옵션을 지정하면 CLEANPATH 옵션을 지정할 수 없습니다.

PARALLEL

기본적으로 UNLOAD는 클러스터의 조각 수에 따라 다수의 파일에 병렬 방식으로 데이터를 작성합니다. 기본 옵션은 ON 또는 TRUE입니다. PARALLEL이 OFF 또는 FALSE인 경우 UNLOAD는 하나 이상의 데이터 파일에 직렬 방식으로 작성하여 절대적으로 ORDER BY 절(사용하는 경우)에 따라 정렬됩니다. 데이터 파일의 최대 크기는 6.2GB입니다. 예를 들어, 13.4GB의 데이터를 언로드하는 경우 UNLOAD는 다음 3개의 파일을 생성합니다.

```
s3://amzn-s3-demo-bucket/key000    6.2 GB
s3://amzn-s3-demo-bucket/key001    6.2 GB
s3://amzn-s3-demo-bucket/key002    1.0 GB
```

> **Note**
>
> UNLOAD 명령은 병렬 처리를 사용하도록 설계되었습니다. 따라서 대부분의 경우 특히, COPY 명령을 사용하여 테이블을 로드하는 데 파일이 사용되는 경우 PARALLEL을 활성화하는 것이 좋습니다.

MAXFILESIZE [AS] max-size [MB | GB]

UNLOAD가 Amazon S3에서 생성하는 파일의 최대 크기를 지정합니다. 5MB와 6.2GB 사이에서 소수 값을 지정하십시오. AS 키워드는 옵션입니다. 기본 단위는 MB입니다. MAXFILESIZE를 지정하지 않을 경우 최대 파일 크기의 기본값은 6.2GB입니다. 이 값을 지정하더라도 매니페스트 파일의 크기는 MAXFILESIZE의 영향을 받지 않습니다.

ROWGROUPSIZE [AS] size [MB | GB]

행 그룹의 크기를 지정합니다. 더 큰 크기를 선택하면 행 그룹 수를 줄여 네트워크 통신량을 줄일 수 있습니다. 32~128MB의 정수 값을 지정합니다. AS 키워드는 옵션입니다. 기본 단위는 MB입니다.

ROWGROUPSIZE가 지정되지 않은 경우 기본 크기는 32MB입니다. 이 파라미터를 사용하려면 스토리지 형식이 Parquet이어야 하고 노드 유형은 ra3.4xlarge, ra3.16xlarge 또는 dc2.8xlarge여야 합니다.

REGION [AS] 'aws-region'

대상 Amazon S3 버킷이 위치한 AWS 리전을 지정합니다. Amazon Redshift 데이터베이스와 동일한 AWS 리전에 있지 않은 Amazon S3 버킷에 대한 UNLOAD의 경우 REGION이 필요합니다.

aws_region의 값은 AWS 일반 참조의 Amazon Redshift 리전 및 엔드포인트 표에 나와 있는 AWS 리전과 일치해야 합니다.

기본적으로 UNLOAD는 대상 Amazon S3 버킷이 Amazon Redshift 데이터베이스와 동일한 AWS 리전에 있다고 가정합니다.

EXTENSION 'extension-name'

언로드된 파일의 이름에 추가할 파일 확장자를 지정합니다. Amazon Redshift는 검증을 실행하지 않으므로, 지정된 파일 확장자가 올바른지 직접 확인해야 합니다. 확장자를 제공하지 않고 압축 방법을 지정하면 Amazon Redshift는 파일 이름에 압축 방법의 확장자만 추가합니다. 확장자를 제공하지 않고 압축 방법을 지정하지 않으면 Amazon Redshift는 파일 이름에 아무 것도 추가하지 않습니다.

사용 노트

구분 기호로 분리된 모든 텍스트 UNLOAD 작업을 위해 ESCAPE 사용

구분 기호를 사용하여 UNLOAD하는 경우 데이터에는 ESCAPE 옵션 설명에 나와 있는 모든 문자 또는 구분 기호가 포함될 수 있습니다. 이 경우 UNLOAD 문과 함께 ESCAPE 옵션을 사용해야 합니다. UNLOAD와 함께 ESCAPE 옵션을 사용하지 않을 경우 언로드된 데이터를 사용하는 이후의 COPY 작업이 실패할 수 있습니다.

> **Important**
>
> ESCAPE는 UNLOAD 및 COPY 문 둘 다와 함께 사용하는 것이 좋습니다. 데이터가 구분 기호 또는 이스케이프될 필요가 있을 수 있는 다른 문자를 포함하고 있지 않은 경우는 예외입니다.

부동 소수점 정밀도의 상실

연속으로 언로드되었다가 다시 로드되는 부동 소수점 데이터의 정밀도가 상실되는 경우가 발생할 수 있습니다.

Limit 절

SELECT 쿼리는 외부 SELECT에 LIMIT 절을 사용할 수 없습니다. 예를 들어, 다음 UNLOAD 문은 실패합니다.

```
unload ('select * from venue limit 10')
to 's3://amzn-s3-demo-bucket/venue_pipe_' iam_role 'arn:aws:iam::0123456789012:role/
MyRedshiftRole';
```

대신, 다음 예와 같이 중첩된 LIMIT 절을 사용하십시오.

```
unload ('select * from venue where venueid in
(select venueid from venue order by venueid desc limit 10)')
to 's3://amzn-s3-demo-bucket/venue_pipe_' iam_role 'arn:aws:iam::0123456789012:role/
MyRedshiftRole';
```

또는 LIMIT 절을 사용하는 SELECT...INTO 또는 CREATE TABLE AS를 사용하여 테이블을 채운 다음 그 테이블에서 언로드할 수 있습니다.

GEOMETRY 데이터 형식의 열 언로드

GEOMETRY 열은 텍스트 또는 CSV 형식으로만 언로드할 수 있습니다. `FIXEDWIDTH` 옵션을 사용하여 GEOMETRY 데이터를 언로드할 수는 없습니다. 데이터는 EWKB(Extended Well-Known Binary) 형식의 16진수로 언로드됩니다. EWKB 데이터의 크기가 4MB를 초과하면 나중에 데이터를 테이블에 로드할 수 없으므로 경고가 발생합니다.

HLLSKETCH 데이터 형식 언로드

HLLSKETCH 열을 텍스트 또는 CSV 형식으로만 온로드할 수 있습니다. `FIXEDWIDTH` 옵션을 사용하여 HLLSKETCH 데이터를 언로드할 수는 없습니다. 데이터는 고밀도 HyperLogLog 스케치의 경우

Base64 형식으로, 희소 HyperLogLog 스케치의 경우 JSON 형식으로 언로드됩니다. 자세한 내용은 HyperLogLog 함수 단원을 참조하십시오.

다음 예에서는 HLLSKETCH 열이 포함된 테이블을 파일로 내보냅니다.

```
CREATE TABLE a_table(an_int INT, b_int INT);
INSERT INTO a_table VALUES (1,1), (2,1), (3,1), (4,1), (1,2), (2,2), (3,2), (4,2),
 (5,2), (6,2);

CREATE TABLE hll_table (sketch HLLSKETCH);
INSERT INTO hll_table select hll_create_sketch(an_int) from a_table group by b_int;

UNLOAD ('select * from hll_table') TO 's3://amzn-s3-demo-bucket/unload/'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole' NULL AS 'null' ALLOWOVERWRITE
 CSV;
```

VARBYTE 데이터 유형의 열 언로드

VARBYTE 열은 텍스트 또는 CSV 형식으로만 언로드할 수 있습니다. 데이터는 16진수 형식으로 언로드됩니다. FIXEDWIDTH 옵션을 사용하여 VARBYTE 데이터를 언로드할 수는 없습니다. CSV로 UNLOAD의 ADDQUOTES 옵션은 지원되지 않습니다. VARBYTE 열은 PARTITIONED BY 열이 될 수 없습니다.

FORMAT AS PARQUET 절

FORMAT AS PARQUET을 사용할 경우 다음을 고려하십시오.

- Parquet에 언로드할 때 파일 수준 압축을 사용하지 않습니다. 각 행 그룹이 SNAPPY로 압축됩니다.
- MAXFILESIZE를 지정하지 않을 경우 최대 파일 크기의 기본값은 6.2GB입니다. MAXFILESIZE를 사용하여 파일 크기를 5MB–6.2GB로 지정할 수 있습니다. 파일을 쓸 때 실제 파일 크기는 근사값이므로 지정한 숫자와 정확히 같지 않을 수 있습니다.

  스캔 성능을 최대화하기 위해에서 Amazon Redshift는 크기가 32MB로 동일한 행 그룹을 포함하는 Parquet 파일을 생성하려고 합니다. 지정한 MAXFLESZEE 값은 32MB의 가장 가까운 배수로 자동으로 내림됩니다. 예를 들어 MAXFILESIZE 200 MB를 지정한 경우 언로드되는 각 Parquet 파일은 약 192MB(32MB 행 그룹 x 6 = 192MB)입니다.
- 열에서 TIMESTAMPTZ 데이터 형식을 사용하는 경우 타임스탬프 값만 언로드됩니다. 시간대 정보는 언로드되지 않습니다.
- 밑줄(_) 또는 마침표(.) 문자로 시작하는 파일 이름 접두사를 지정하지 마십시오. Redshift Spectrum에서는 이러한 문자로 시작하는 파일을 숨김 파일로 간주하고 무시합니다.

PARTITION BY 절

PARTITION BY를 사용할 경우 다음을 고려하십시오.

- 파티션 열은 출력 파일에 포함되지 않습니다.
- UNLOAD 문에 사용되는 SELECT 쿼리에 파티션 열을 포함해야 합니다. UNLOAD 명령에서 원하는 수의 파티션 열을 지정할 수 있습니다. 하지만 파일의 일부가 될 파티션이 아닌 열이 하나 이상 있어야 한다는 제한이 있습니다.
- 파티션 키 값이 null이면 Amazon Redshift에서 해당 데이터를 `partition_column=__HIVE_DEFAULT_PARTITION__`이라는 기본 파티션으로 자동으로 언로드합니다.
- UNLOAD 명령은 외부 카탈로그를 호출하지 않습니다. 새 파티션을 기존 외부 테이블의 일부로 등록하려면 별도의 ALTER TABLE ... ADD PARTITION ... 명령을 사용합니다. 또는 CREATE EXTERNAL TABLE 명령을 실행하여 언로드된 데이터를 새 외부 테이블로 등록할 수 있습니다. AWS Glue 크롤러를 사용하여 Data Catalog를 채울 수도 있습니다. 자세한 내용은 AWS Glue Developer Guide의 Defining Crawlers 섹션을 참조하세요.
- MANIFEST 옵션을 사용하는 경우 Amazon Redshift는 루트 Amazon S3 폴더에 매니페스트 파일을 하나만 생성합니다.
- 파티션 키로 사용할 수 있는 열 데이터 형식은 SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, BOOLEAN, CHAR, VARCHAR, DATE 및 TIMESTAMP입니다.

ASSUMEROLE 권한을 사용하여 UNLOAD 작업에 대한 IAM 역할에 대한 액세스 권한 부여

UNLOAD 작업을 위해 IAM 역할에 대한 액세스 권한을 특정 사용자 및 그룹에 제공하려면 슈퍼 사용자는 IAM 역할에 대한 ASSUMEROLE 권한을 사용자 및 그룹에 부여할 수 있습니다. 자세한 내용은 GRANT을 참조하세요.

UNLOAD는 Amazon S3 액세스 포인트 별칭을 지원하지 않습니다.

UNLOAD 명령에는 Amazon S3 액세스 포인트 별칭을 사용할 수 없습니다.

예시

UNLOAD 명령을 사용하는 방법을 보여주는 예제는 UNLOAD 예 섹션을 참조하세요.

UNLOAD 예

이 예제에서는 UNLOAD 명령의 다양한 파라미터를 보여줍니다. 많은 예제에서 TICKIT 샘플 데이터가 사용됩니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

> **Note**
>
> 여기에서 설명하는 예는 가독성을 위해 줄 바꿈이 포함되었습니다. 실제 credentials-args 문자열에서는 줄 바꿈이나 공백을 입력하지 마십시오.

파이프(기본 구분 기호)로 구분된 파일로 VENUE 언로드

다음 예에서는 VENUE 테이블을 언로드하고 s3://amzn-s3-demo-bucket/unload/에 데이터를 씁니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

기본적으로, UNLOAD는 조각당 하나 이상의 파일을 작성합니다. 노드당 2개의 조각이 있는 2노드 클러스터를 가정할 때, 이전의 예에서는 다음 파일이 amzn-s3-demo-bucket에 생성됩니다.

```
unload/0000_part_00
unload/0001_part_00
unload/0002_part_00
unload/0003_part_00
```

출력 파일을 더 효과적으로 구분하려면 해당 위치에 접두사를 포함하면 됩니다. 다음 예에서는 VENUE 테이블을 언로드하고 s3://amzn-s3-demo-bucket/unload/venue_pipe_에 데이터를 씁니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/unload/venue_pipe_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

그 결과, unload 폴더에 다음 4개의 파일이 있으며, 이때도 4개의 조각이 있는 것으로 가정합니다.

```
venue_pipe_0000_part_00
venue_pipe_0001_part_00
venue_pipe_0002_part_00
venue_pipe_0003_part_00
```

분할된 Parquet 파일에 LINEITE 테이블 언로드

다음 예에서는 LINEITEM 테이블을 l_shipdate 열로 분할하여 Parquet 형식으로 언로드합니다.

```
unload ('select * from lineitem')
to 's3://amzn-s3-demo-bucket/lineitem/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
PARQUET
PARTITION BY (l_shipdate);
```

네 개의 조각을 가정할 때 결과 Parquet 파일은 다양한 폴더에 동적으로 분할됩니다.

```
s3://amzn-s3-demo-bucket/lineitem/l_shipdate=1992-01-02/0000_part_00.parquet
                                                0001_part_00.parquet
                                                0002_part_00.parquet
                                                0003_part_00.parquet
s3://amzn-s3-demo-bucket/lineitem/l_shipdate=1992-01-03/0000_part_00.parquet
                                                0001_part_00.parquet
                                                0002_part_00.parquet
                                                0003_part_00.parquet
s3://amzn-s3-demo-bucket/lineitem/l_shipdate=1992-01-04/0000_part_00.parquet
                                                0001_part_00.parquet
                                                0002_part_00.parquet
                                                0003_part_00.parquet
...
```

Note

경우에 따라 UNLOAD 명령에서 다음 SQL 문과 같이 INCLUDE 옵션을 사용했습니다.

```
unload ('select * from lineitem')
to 's3://amzn-s3-demo-bucket/lineitem/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
PARQUET
PARTITION BY (l_shipdate) INCLUDE;
```

이러한 경우 l_shipdate 열은 Parquet 파일의 데이터에도 있습니다. 그렇지 않은 경우 l_shipdate 열 데이터가 Parquet 파일에 없습니다.

JSON 파일로 VENUE 테이블 언로드

다음 예에서는 VENUE 테이블을 언로드하고 s3://amzn-s3-demo-bucket/unload/에 JSON 형식의 데이터를 씁니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
JSON;
```

다음은 VENUE 테이블의 샘플 행입니다.

```
venueid | venuename                  | venuecity       | venuestate | venueseats
--------+----------------------------+-----------------+------------+-----------
      1 | Pinewood Racetrack         | Akron           | OH         | 0
      2 | Columbus "Crew" Stadium    | Columbus        | OH         | 0
      4 | Community, Ballpark, Arena | Kansas City     | KS         | 0
```

JSON으로 언로드한 후 파일의 형식은 다음과 유사합니다.

```
{"venueid":1,"venuename":"Pinewood
 Racetrack","venuecity":"Akron","venuestate":"OH","venueseats":0}
{"venueid":2,"venuename":"Columbus \"Crew\" Stadium
 ","venuecity":"Columbus","venuestate":"OH","venueseats":0}
{"venueid":4,"venuename":"Community, Ballpark, Arena","venuecity":"Kansas
 City","venuestate":"KS","venueseats":0}
```

CSV 파일로 VENUE 언로드

다음 예에서는 VENUE 테이블을 언로드하고 s3://amzn-s3-demo-bucket/unload/에 CSV 형식의 데이터를 씁니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
CSV;
```

VENUE 테이블에 다음 행이 포함되어 있다고 가정합니다.

```
venueid | venuename                  | venuecity       | venuestate | venueseats
--------+----------------------------+-----------------+------------+-----------
      1 | Pinewood Racetrack         | Akron           | OH         | 0
      2 | Columbus "Crew" Stadium    | Columbus        | OH         | 0
      4 | Community, Ballpark, Arena | Kansas City     | KS         | 0
```

언로드 파일은 다음과 유사합니다.

```
1,Pinewood Racetrack,Akron,OH,0
2,"Columbus ""Crew"" Stadium",Columbus,OH,0
4,"Community, Ballpark, Arena",Kansas City,KS,0
```

구분 기호를 사용하여 CSV 파일로 VENUE 언로드

다음 예제에서는 VENUE 테이블을 언로드하고 파이프 문자(|)를 구분 기호로 사용하여 데이터를 CSV 형식으로 씁니다. 언로드된 파일이 s3://amzn-s3-demo-bucket/unload/에 기록됩니다. 이 예제의 VENUE 테이블에는 첫 번째 행의 값에 파이프 문자가 포함되어 있습니다(Pinewood Race|track). 이는 결과의 값이 큰따옴표로 묶여 있음을 보여 줍니다. 큰따옴표는 큰따옴표로 이스케이프되고 전체 필드는 큰따옴표로 묶여 있습니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
CSV DELIMITER AS '|';
```

VENUE 테이블에 다음 행이 포함되어 있다고 가정합니다.

```
venueid | venuename                  | venuecity       | venuestate | venueseats
--------+----------------------------+-----------------+------------+-------------
      1 | Pinewood Race|track        | Akron           | OH         | 0
      2 | Columbus "Crew" Stadium    | Columbus        | OH         | 0
      4 | Community, Ballpark, Arena | Kansas City     | KS         | 0
```

언로드 파일은 다음과 유사합니다.

```
1|"Pinewood Race|track"|Akron|OH|0
2|"Columbus ""Crew"" Stadium"|Columbus|OH|0
4|Community, Ballpark, Arena|Kansas City|KS|0
```

매니페스트 파일로 VENUE 언로드

매니페스트 파일을 생성하려면 MANIFEST 옵션을 포함하십시오. 다음 예에서는 VENUE 테이블을 언로드하고 데이터 파일과 함께 매니페스트 파일을 s3://amzn-s3-demo-bucket/venue_pipe_에 씁니다.

> **Important**
>
> MANIFEST 옵션으로 파일을 언로드하는 경우 파일을 로드할 때 COPY 명령과 함께 MANIFEST 옵션을 사용해야 합니다. 같은 접두사를 사용하여 파일을 로드하고 MANIFEST 옵션을 지정하지 않는 경우 COPY 작업에서는 매니페스트 파일이 데이터 파일이라 가정하므로 COPY가 실패하게 됩니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/venue_pipe_' iam_role 'arn:aws:iam::0123456789012:role/
MyRedshiftRole'
manifest;
```

그 결과는 다음과 같은 5개의 파일입니다.

```
s3://amzn-s3-demo-bucket/venue_pipe_0000_part_00
s3://amzn-s3-demo-bucket/venue_pipe_0001_part_00
s3://amzn-s3-demo-bucket/venue_pipe_0002_part_00
s3://amzn-s3-demo-bucket/venue_pipe_0003_part_00
s3://amzn-s3-demo-bucket/venue_pipe_manifest
```

다음은 매니페스트 파일 내용을 나타낸 것입니다.

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket/tickit/venue_0000_part_00"},
    {"url":"s3://amzn-s3-demo-bucket/tickit/venue_0001_part_00"},
    {"url":"s3://amzn-s3-demo-bucket/tickit/venue_0002_part_00"},
    {"url":"s3://amzn-s3-demo-bucket/tickit/venue_0003_part_00"}
  ]
}
```

MANIFEST VERBOSE를 사용해 VENUE 언로드

MANIFEST VERBOSE 옵션을 지정하면 매니페스트 파일에는 다음 섹션이 포함됩니다.

- entries 섹션에는 Amazon S3 경로, 파일 크기 및 각 파일에 대한 행 수가 나열됩니다.
- schema 섹션에는 열 이름, 데이터 유형, 및 각 열에 대한 차원이 나열됩니다.
- meta 섹션에는 총 파일 크기 및 모든 파일의 행 수가 표시됩니다.

다음 예에서는 MANIFEST VERBOSE 옵션을 사용하여 VENUE 테이블을 언로드합니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/unload_venue_folder/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest verbose;
```

다음은 매니페스트 파일 내용을 나타낸 것입니다.

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket/venue_pipe_0000_part_00", "meta":
 { "content_length": 32295, "record_count": 10 }},
    {"url":"s3://amzn-s3-demo-bucket/venue_pipe_0001_part_00", "meta":
 { "content_length": 32771, "record_count": 20 }},
    {"url":"s3://amzn-s3-demo-bucket/venue_pipe_0002_part_00", "meta":
 { "content_length": 32302, "record_count": 10 }},
    {"url":"s3://amzn-s3-demo-bucket/venue_pipe_0003_part_00", "meta":
 { "content_length": 31810, "record_count": 15 }}
  ],
  "schema": {
    "elements": [
      {"name": "venueid", "type": { "base": "integer" }},
      {"name": "venuename", "type": { "base": "character varying", 25 }},
      {"name": "venuecity", "type": { "base": "character varying", 25 }},
      {"name": "venuestate", "type": { "base": "character varying", 25 }},
      {"name": "venueseats", "type": { "base": "character varying", 25 }}
    ]
  },
  "meta": {
    "content_length": 129178,
    "record_count": 55
  },
  "author": {
    "name": "Amazon Redshift",
    "version": "1.0.0"
  }
```

```
}
```

헤더로 VENUE 언로드

다음 예에서는 헤더 행으로 VENUE를 언로드합니다.

```
unload ('select * from venue where venueseats > 75000')
to 's3://amzn-s3-demo-bucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
header
parallel off;
```

다음은 헤더 행을 포함하는 출력 파일 내용을 보여 줍니다.

```
venueid|venuename|venuecity|venuestate|venueseats
6|New York Giants Stadium|East Rutherford|NJ|80242
78|INVESCO Field|Denver|CO|76125
83|FedExField|Landover|MD|91704
79|Arrowhead Stadium|Kansas City|MO|79451
```

더 작은 크기의 파일로 VENUE 언로드

최대 파일 크기의 기본값은 6.2GB입니다. 언로드 데이터가 6.2GB보다 큰 경우 UNLOAD는 각각의 6.2GB 데이터 세그먼트에 대한 새 파일을 생성합니다. 더 작은 크기의 파일을 생성하려면 MAXFILESIZE 파라미터를 추가하십시오. 이전 예에서 데이터 크기가 20GB라고 가정했을 때 다음 UNLOAD 명령은 각각 1GB씩 20개의 파일을 생성합니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
maxfilesize 1 gb;
```

VENUE의 연속 언로드

연속으로 언로드하려면 PARALLEL OFF를 지정하십시오. 그러면 UNLOAD가 한 번에 한 개의 파일을 쓰는데, 파일당 최대 6.2GB의 데이터를 씁니다.

다음 예에서는 VENUE 테이블을 언로드하고 s3://amzn-s3-demo-bucket/unload/에 데이터를 연속으로 씁니다.

```
unload ('select * from venue')
```

```
to 's3://amzn-s3-demo-bucket/unload/venue_serial_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
parallel off;
```

그 결과 venue_serial_000이라는 파일이 한 개 생성됩니다.

언로드 데이터가 6.2GB보다 큰 경우 UNLOAD는 각각의 6.2GB 데이터 세그먼트에 대한 새 파일을 생성합니다. 다음 예에서는 LINEORDER 테이블을 언로드하고 s3://amzn-s3-demo-bucket/unload/에 데이터를 연속으로 씁니다.

```
unload ('select * from lineorder')
to 's3://amzn-s3-demo-bucket/unload/lineorder_serial_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
parallel off gzip;
```

그 결과는 다음과 같은 일련의 파일입니다.

```
lineorder_serial_0000.gz
lineorder_serial_0001.gz
lineorder_serial_0002.gz
lineorder_serial_0003.gz
```

출력 파일을 더 효과적으로 구분하려면 해당 위치에 접두사를 포함하면 됩니다. 다음 예에서는 VENUE 테이블을 언로드하고 s3://amzn-s3-demo-bucket/venue_pipe_에 데이터를 씁니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/unload/venue_pipe_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

그 결과, unload 폴더에 다음 4개의 파일이 있으며, 이때도 4개의 조각이 있는 것으로 가정합니다.

```
venue_pipe_0000_part_00
venue_pipe_0001_part_00
venue_pipe_0002_part_00
venue_pipe_0003_part_00
```

언로드 파일에서 VENUE 로드

언로드 파일 집합에서 테이블을 로드하려면 COPY 명령을 사용하여 단순히 절차를 거꾸로 진행하십시오. 다음 예에서는 새 테이블인 LOADVENUE를 생성하고 이전 예에서 생성된 데이터 파일에서 테이블을 로드합니다.

```
create table loadvenue (like venue);

copy loadvenue from 's3://amzn-s3-demo-bucket/venue_pipe_' iam_role
 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

MANIFEST 옵션을 사용하여 언로드 파일로 매니페스트 파일을 생성한 경우 동일한 매니페스트 파일을 사용하여 데이터를 로드할 수 있습니다. MANIFEST 옵션과 함께 COPY 명령을 사용하면 이렇게 할 수 있습니다. 다음 예에서는 매니페스트 파일을 사용하여 데이터를 로드합니다.

```
copy loadvenue
from 's3://amzn-s3-demo-bucket/venue_pipe_manifest' iam_role
 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest;
```

암호화된 파일로 VENUE 언로드

다음 예에서는 AWS KMS 키를 사용하여 암호화된 파일 집합으로 VENUE 테이블을 언로드합니다. ENCRYPTED 옵션으로 매니페스트 파일을 지정하는 경우 매니페스트 파일도 암호화됩니다. 자세한 내용은 암호화된 데이터 파일 언로드 단원을 참조하십시오.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/venue_encrypt_kms'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
kms_key_id '1234abcd-12ab-34cd-56ef-1234567890ab'
manifest
encrypted;
```

다음 예에서는 루트 대칭 키를 사용하여 암호화된 파일 집합으로 VENUE 테이블을 언로드합니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/venue_encrypt_cmk'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key 'EXAMPLEMASTERKEYtkbjk/OpCwtYSx/M4/t7DMCDIK722'
encrypted;
```

암호화된 파일에서 VENUE 로드

ENCRYPT 옵션과 함께 UNLOAD를 사용해 생성한 파일 집합에서 테이블을 로드하려면 COPY 명령을 사용해 이 프로세스를 거꾸로 수행하십시오. 이 명령과 함께 ENCRYPTED 옵션을 사용하고 UNLOAD

명령에 사용한 것과 동일한 루트 대칭 키를 지정합니다. 다음 예에서는 이전 예에서 생성된 암호화된 데이터 파일에서 LOADVENUE 테이블을 로드합니다.

```
create table loadvenue (like venue);

copy loadvenue
from 's3://amzn-s3-demo-bucket/venue_encrypt_manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key 'EXAMPLEMASTERKEYtkbjk/OpCwtYSx/M4/t7DMCDIK722'
manifest
encrypted;
```

VENUE 데이터를 탭으로 구분된 파일로 언로드

```
unload ('select venueid, venuename, venueseats from venue')
to 's3://amzn-s3-demo-bucket/venue_tab_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter as '\t';
```

출력 데이터 파일의 모습은 다음과 같습니다.

```
1 Toyota Park Bridgeview IL 0
2 Columbus Crew Stadium Columbus OH 0
3 RFK Stadium Washington DC 0
4 CommunityAmerica Ballpark Kansas City KS 0
5 Gillette Stadium Foxborough MA 68756
...
```

고정 폭 데이터 파일로 VENUE 언로드

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/venue_fw_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
fixedwidth as 'venueid:3,venuename:39,venuecity:16,venuestate:2,venueseats:6';
```

출력 데이터 파일은 다음과 같을 것입니다.

```
1  Toyota Park              Bridgeview  IL0
2  Columbus Crew Stadium    Columbus    OH0
3  RFK Stadium              Washington  DC0
4  CommunityAmerica BallparkKansas City KS0
```

```
5  Gillette Stadium         Foxborough  MA68756
...
```

탭으로 구분된 GZIP 압축 파일 집합으로 VENUE 언로드

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/venue_tab_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter as '\t'
gzip;
```

VENUE를 GZIP 압축 텍스트 파일로 언로드

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/venue_tab_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
extension 'txt.gz'
gzip;
```

구분 기호를 포함한 데이터 언로드

다음 예에서는 ADDQUOTES 옵션을 사용하여 실제 데이터 필드 중 일부에 쉼표가 있는, 쉼표로 구분된 데이터를 언로드합니다.

먼저, 따옴표가 포함되어 있는 테이블을 만듭니다.

```
create table location (id int, location char(64));

insert into location values (1,'Phoenix, AZ'),(2,'San Diego, CA'),(3,'Chicago, IL');
```

그런 다음, ADDQUOTES 옵션을 사용하여 데이터를 언로드합니다.

```
unload ('select id, location from location')
to 's3://amzn-s3-demo-bucket/location_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter ',' addquotes;
```

언로드된 데이터 파일의 모습은 다음과 같습니다.

```
1,"Phoenix, AZ"
```

```
2,"San Diego, CA"
3,"Chicago, IL"
...
```

조인 쿼리의 결과 언로드

다음 예에서는 창 함수를 포함하는 조인 쿼리의 결과를 언로드합니다.

```
unload ('select venuecity, venuestate, caldate, pricepaid,
sum(pricepaid) over(partition by venuecity, venuestate
order by caldate rows between 3 preceding and 3 following) as winsum
from sales join date on sales.dateid=date.dateid
join event on event.eventid=sales.eventid
join venue on event.venueid=venue.venueid
order by 1,2')
to 's3://amzn-s3-demo-bucket/tickit/winsum'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

출력 파일의 모습은 다음과 같습니다.

```
Atlanta|GA|2008-01-04|363.00|1362.00
Atlanta|GA|2008-01-05|233.00|2030.00
Atlanta|GA|2008-01-06|310.00|3135.00
Atlanta|GA|2008-01-08|166.00|8338.00
Atlanta|GA|2008-01-11|268.00|7630.00
...
```

NULL AS를 사용한 언로드

UNLOAD는 기본적으로 null 값을 빈 문자열로 출력합니다. 다음 예에서는 NULL AS를 사용하여 null에 대한 텍스트 문자열을 대체하는 방법을 보여 줍니다.

이들 예에서는 VENUE 테이블에 null 값을 추가할 것입니다.

```
update venue set venuestate = NULL
where venuecity = 'Cleveland';
```

열에 NULL이 포함되어 있음을 확인하기 위해 VENUESTATE가 null인 VENUE에서 선택합니다.

```
select * from venue where venuestate is null;
```

```
 venueid |        venuename          | venuecity | venuestate | venueseats
---------+---------------------------+-----------+------------+------------
      22 | Quicken Loans Arena       | Cleveland |            |          0
     101 | Progressive Field         | Cleveland |            |      43345
      72 | Cleveland Browns Stadium  | Cleveland |            |      73200
```

이제, NULL AS 옵션을 사용해 VENUE 테이블을 UNLOAD하여 null 값을 문자열 'fred'로 바꿉니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
null as 'fred';
```

언로드 파일에서 다음 샘플은 null 값이 fred로 바뀌었음을 보여줍니다. VENUESEATS의 일부 값 역시 null이고 fred로 바뀐 것으로 나타납니다. VENUESEATS의 데이터 형식이 정수형이지만 UNLOAD가 언로드 파일에서 값을 텍스트로 변환한 다음, COPY가 이를 다시 정수로 변환합니다. 고정 폭 파일로 언로드하는 경우 NULL AS 문자열이 필드 폭보다 크면 안 됩니다.

```
248|Charles Playhouse|Boston|MA|0
251|Paris Hotel|Las Vegas|NV|fred
258|Tropicana Hotel|Las Vegas|NV|fred
300|Kennedy Center Opera House|Washington|DC|0
306|Lyric Opera House|Baltimore|MD|0
308|Metropolitan Opera|New York City|NY|0
  5|Gillette Stadium|Foxborough|MA|5
 22|Quicken Loans Arena|Cleveland|fred|0
101|Progressive Field|Cleveland|fred|43345
...
```

언로드 파일에서 테이블을 로드하려면 동일한 NULL AS 옵션과 함께 COPY 명령을 사용하십시오.

> **Note**
>
> NULL을 NOT NULL로 정의된 열에 로드하려고 하면 COPY 명령이 실패합니다.

```
create table loadvenuenulls (like venue);

copy loadvenuenulls from 's3://amzn-s3-demo-bucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
```

```
null as 'fred';
```

열에 빈 문자열뿐 아니라 null이 포함되어 있는지 확인하려면 LOADVENUENULLS에서 선택하고 null이 있는지 필터링하십시오.

```
select * from loadvenuenulls where venuestate is null or venueseats is null;

 venueid |        venuename         | venuecity | venuestate | venueseats
---------+--------------------------+-----------+------------+------------
      72 | Cleveland Browns Stadium | Cleveland |            |      73200
     253 | Mirage Hotel             | Las Vegas | NV         |
     255 | Venetian Hotel           | Las Vegas | NV         |
      22 | Quicken Loans Arena      | Cleveland |            |          0
     101 | Progressive Field        | Cleveland |            |      43345
     251 | Paris Hotel              | Las Vegas | NV         |

...
```

기본 NULL AS 동작을 사용하여 null이 포함된 테이블을 UNLOAD한 다음 기본 NULL AS 동작을 사용하여 데이터를 테이블로 다시 COPY할 수 있습니다. 하지만 대상 테이블에서 숫자가 아닌 필드는 전부 null이 아니라 빈 문자열을 포함하게 됩니다. 기본적으로 UNLOAD는 null을 빈 문자열(공백 또는 제로 길이)로 변환합니다. COPY는 숫자 열에 대해 빈 문자열을 NULL로 변환하지만, 빈 문자열을 숫자가 아닌 열에 삽입합니다. 다음 예에서는 기본 NULL AS 동작을 사용하여 UNLOAD와 COPY를 차례대로 수행하는 방법을 보여줍니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole' allowoverwrite;

truncate loadvenuenulls;
copy loadvenuenulls from 's3://amzn-s3-demo-bucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

이 경우에는 null을 필터링할 때 VENUESEATS가 null을 포함했던 행만 선택됩니다. 테이블(VENUE)에서 VENUESTATE가 null을 포함한 경우 대상 테이블(LOADVENUENULLS)의 VENUESTATE는 빈 문자열을 포함합니다.

```
select * from loadvenuenulls where venuestate is null or venueseats is null;

 venueid |        venuename         | venuecity | venuestate | venueseats
```

```
---------+--------------------------+-----------+------------+------------
     253 | Mirage Hotel             | Las Vegas | NV         |
     255 | Venetian Hotel           | Las Vegas | NV         |
     251 | Paris Hotel              | Las Vegas | NV         |
...
```

빈 문자열을 NULL로서 숫자가 아닌 열에 로드하려면 EMPTYASNULL 또는 BLANKSASNULL 옵션을 포함하십시오. 둘 다 사용해도 괜찮습니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole' allowoverwrite;

truncate loadvenuenulls;
copy loadvenuenulls from 's3://amzn-s3-demo-bucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole' EMPTYASNULL;
```

열에 공백이나 빈 문자열뿐 아니라 NULL이 포함되어 있는지 확인하려면 LOADVENUENULLS에서 선택하고 null이 있는지 필터링합니다.

```
select * from loadvenuenulls where venuestate is null or venueseats is null;

 venueid |        venuename         | venuecity | venuestate | venueseats
---------+--------------------------+-----------+------------+------------
      72 | Cleveland Browns Stadium | Cleveland |            |      73200
     253 | Mirage Hotel             | Las Vegas | NV         |
     255 | Venetian Hotel           | Las Vegas | NV         |
      22 | Quicken Loans Arena      | Cleveland |            |          0
     101 | Progressive Field        | Cleveland |            |      43345
     251 | Paris Hotel              | Las Vegas | NV         |
     ...
```

ALLOWOVERWRITE 파라미터를 사용한 언로드

기본적으로 UNLOAD는 대상 버킷의 기존 파일을 덮어쓰지 않습니다. 예를 들어 대상 버킷에서 파일을 수정하지 않고 같은 UNLOAD 문을 두 번 실행하는 경우 두 번째 UNLOAD는 실패하게 됩니다. 매니페스트 파일을 포함하여 기존 파일을 덮어쓰려면 ALLOWOVERWRITE 옵션을 지정합니다.

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/venue_pipe_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
```

```
manifest allowoverwrite;
```

PARALLEL 및 MANIFEST 파라미터를 사용하여 EVENT 테이블 언로드

테이블을 병렬로 언로드하고 매니페스트 파일을 생성할 수 있습니다. Amazon S3 데이터 파일은 모두 동일한 수준에서 생성되며 이름에 `0000_part_00` 패턴이 접미사로 붙습니다. 매니페스트 파일은 데이터 파일과 동일한 폴더 수준에 있으며 `manifest` 텍스트가 접미사로 붙습니다. 다음 SQL은 EVENT 테이블을 언로드하고 기본 이름 `parallel`로 파일을 생성합니다.

```
unload ('select * from mytickit1.event')
to 's3://amzn-s3-demo-bucket/parallel'
iam_role 'arn:aws:iam::123456789012:role/MyRedshiftRole'
parallel on
manifest;
```

Amazon S3 파일 목록은 다음과 유사합니다.

```
 Name                        Last modified                         Size
 parallel0000_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 52.1 KB
 parallel0001_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 53.4 KB
 parallel0002_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 52.1 KB
 parallel0003_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 51.1 KB
 parallel0004_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 54.6 KB
 parallel0005_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 53.4 KB
 parallel0006_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 54.1 KB
 parallel0007_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 55.9 KB
 parallelmanifest       -   August 2, 2023, 14:54:39 (UTC-07:00) 886.0 B
```

`parallelmanifest` 파일 콘텐츠는 다음과 유사합니다.

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket/parallel0000_part_00", "meta": { "content_length":
 53316 }},
    {"url":"s3://amzn-s3-demo-bucket/parallel0001_part_00", "meta": { "content_length":
 54704 }},
    {"url":"s3://amzn-s3-demo-bucket/parallel0002_part_00", "meta": { "content_length":
 53326 }},
```

```
    {"url":"s3://amzn-s3-demo-bucket/parallel0003_part_00", "meta": { "content_length":
 52356 }},
    {"url":"s3://amzn-s3-demo-bucket/parallel0004_part_00", "meta": { "content_length":
 55933 }},
    {"url":"s3://amzn-s3-demo-bucket/parallel0005_part_00", "meta": { "content_length":
 54648 }},
    {"url":"s3://amzn-s3-demo-bucket/parallel0006_part_00", "meta": { "content_length":
 55436 }},
    {"url":"s3://amzn-s3-demo-bucket/parallel0007_part_00", "meta": { "content_length":
 57272 }}
  ]
}
```

PARALLEL OFF 및 MANIFEST 파라미터를 사용하여 EVENT 테이블 언로드

테이블을 연속적으로 언로드하고(PARALLEL OFF) 매니페스트 파일을 생성할 수 있습니다. Amazon S3 데이터 파일은 모두 동일한 수준에서 생성되며 이름에 0000 패턴이 접미사로 붙습니다. 매니페스트 파일은 데이터 파일과 동일한 폴더 수준에 있으며 manifest 텍스트가 접미사로 붙습니다.

```
unload ('select * from mytickit1.event')
to 's3://amzn-s3-demo-bucket/serial'
iam_role 'arn:aws:iam::123456789012:role/MyRedshiftRole'
parallel off
manifest;
```

Amazon S3 파일 목록은 다음과 유사합니다.

```
 Name                         Last modified                           Size
 serial0000             -     August 2, 2023, 15:54:39 (UTC-07:00) 426.7 KB
 serialmanifest         -     August 2, 2023, 15:54:39 (UTC-07:00) 120.0 B
```

serialmanifest 파일 콘텐츠는 다음과 유사합니다.

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket/serial000", "meta": { "content_length": 436991 }}
  ]
}
```

PARTITION BY 및 MANIFEST 파라미터를 사용하여 EVENT 테이블 언로드

파티션별로 테이블을 언로드하고 매니페스트 파일을 생성할 수 있습니다. Amazon S3에 하위 파티션 폴더가 있는 새 폴더가 생성되고 하위 폴더의 데이터 파일은 `0000_par_00`과 유사한 이름 패턴을 갖습니다. 매니페스트 파일은 하위 폴더와 동일한 폴더 레벨에 있으며 이름은 `manifest`입니다.

```
unload ('select * from mytickit1.event')
to 's3://amzn-s3-demo-bucket/partition'
iam_role 'arn:aws:iam::123456789012:role/MyRedshiftRole'
partition by (eventname)
manifest;
```

Amazon S3 파일 목록은 다음과 유사합니다.

```
Name                   Type     Last modified                         Size

partition             Folder
```

`partition` 폴더에는 파티션 이름을 가진 하위 폴더와 매니페스트 파일이 있습니다. `partition` 폴더의 폴더 목록 하단 부분은 다음과 유사합니다.

```
Name                   Type      Last modified                          Size

...
eventname=Zucchero/    Folder
eventname=Zumanity/    Folder
eventname=ZZ Top/      Folder
manifest               -     August 2, 2023, 15:54:39 (UTC-07:00) 467.6 KB
```

`eventname=Zucchero/` 폴더에는 다음과 유사한 데이터 파일이 있습니다.

```
Name               Last modified                         Size
0000_part_00 -   August 2, 2023, 15:59:19 (UTC-07:00) 70.0 B
0001_part_00 -   August 2, 2023, 15:59:16 (UTC-07:00) 106.0 B
0002_part_00 -   August 2, 2023, 15:59:15 (UTC-07:00) 70.0 B
0004_part_00 -   August 2, 2023, 15:59:17 (UTC-07:00) 141.0 B
```

```
 0006_part_00 -   August 2, 2023, 15:59:16 (UTC-07:00) 35.0 B
 0007_part_00 -   August 2, 2023, 15:59:19 (UTC-07:00) 108.0 B
```

manifest 파일 콘텐츠의 하단은 다음과 유사합니다.

```
{
  "entries": [
    ...
    {"url":"s3://amzn-s3-demo-bucket/partition/eventname=Zucchero/007_part_00", "meta":
 { "content_length": 108 }},
    {"url":"s3://amzn-s3-demo-bucket/partition/eventname=Zumanity/007_part_00", "meta":
 { "content_length": 72 }}
  ]
}
```

MAXFILESIZE, ROWGROUPSIZE 및 MANIFEST 파라미터를 사용하여 EVENT 테이블 언로드

테이블을 병렬로 언로드하고 매니페스트 파일을 생성할 수 있습니다. Amazon S3 데이터 파일은 모두 동일한 수준에서 생성되며 이름에 0000_part_00 패턴이 접미사로 붙습니다. 생성된 Parquet 데이터 파일은 256MB, 행 그룹 크기는 128MB로 제한됩니다. 매니페스트 파일은 데이터 파일과 동일한 폴더 수준에 있으며 manifest가 접미사로 붙습니다.

```
unload ('select * from mytickit1.event')
to 's3://amzn-s3-demo-bucket/eventsize'
iam_role 'arn:aws:iam::123456789012:role/MyRedshiftRole'
maxfilesize 256 MB
rowgroupsize 128 MB
parallel on
parquet
manifest;
```

Amazon S3 파일 목록은 다음과 유사합니다.

```
 Name                            Type      Last modified                     Size
 eventsize0000_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 24.5 KB
 eventsize0001_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 24.8 KB
 eventsize0002_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 24.4 KB
 eventsize0003_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 24.0 KB
 eventsize0004_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 25.3 KB
```

```
 eventsize0005_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 24.8 KB
 eventsize0006_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 25.0 KB
 eventsize0007_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 25.6 KB
 eventsizemanifest                 -       August 2, 2023, 17:35:21 (UTC-07:00) 958.0 B
```

`eventsizemanifest` 파일 콘텐츠는 다음과 유사합니다.

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket/eventsize0000_part_00.parquet", "meta":
 { "content_length": 25130 }},
    {"url":"s3://amzn-s3-demo-bucket/eventsize0001_part_00.parquet", "meta":
 { "content_length": 25428 }},
    {"url":"s3://amzn-s3-demo-bucket/eventsize0002_part_00.parquet", "meta":
 { "content_length": 25025 }},
    {"url":"s3://amzn-s3-demo-bucket/eventsize0003_part_00.parquet", "meta":
 { "content_length": 24554 }},
    {"url":"s3://amzn-s3-demo-bucket/eventsize0004_part_00.parquet", "meta":
 { "content_length": 25918 }},
    {"url":"s3://amzn-s3-demo-bucket/eventsize0005_part_00.parquet", "meta":
 { "content_length": 25362 }},
    {"url":"s3://amzn-s3-demo-bucket/eventsize0006_part_00.parquet", "meta":
 { "content_length": 25647 }},
    {"url":"s3://amzn-s3-demo-bucket/eventsize0007_part_00.parquet", "meta":
 { "content_length": 26256 }}
  ]
}
```

UPDATE

주제

- 구문
- 파라미터
- 사용 노트
- UPDATE 문 예

조건 충족 시 하나 이상의 테이블 열에서 값을 업데이트합니다.

> **Note**
>
> 단일 SQL 문의 최대 크기는 16MB입니다.

구문

```
[ WITH [RECURSIVE] common_table_expression [, common_table_expression , ...] ]
            UPDATE table_name [ [ AS ] alias ] SET column = { expression | DEFAULT }
 [,...]

[ FROM fromlist ]
[ WHERE condition ]
```

파라미터

WITH 절

하나 이상의 common-table-expressions를 지정하는 절(옵션)입니다. WITH 절 섹션을 참조하세요.

table_name

임시 또는 영구 테이블입니다. 테이블의 소유자 또는 테이블에 대한 UPDATE 권한을 가진 사용자만이 행을 업데이트할 수 있습니다. FROM 절을 사용하거나 표현식이나 조건에 있는 테이블에서 선택하는 경우 해당 테이블에 대한 SELECT 권한이 있어야 합니다. 여기서 테이블에 별칭을 지정할 수 없지만, FROM 절에서 별칭을 지정할 수 있습니다.

> **Note**
>
> Amazon Redshift Spectrum 외부 테이블은 읽기 전용입니다. 외부 테이블을 업데이트할 수 없습니다.

별칭

대상 테이블의 임시 대체 이름입니다. 별칭은 옵션입니다. AS 키워드는 항상 옵션입니다.

SET column =

수정하려는 하나 이상의 열입니다. 나열되지 않는 열은 현재 값을 유지합니다. 대상 열을 지정할 때 테이블 이름을 포함하지 마십시오. 예를 들어, `UPDATE tab SET tab.col = 1`은 유효하지 않습니다.

expression

지정된 열에 대한 새 값을 정의하는 표현식입니다.

DEFAULT

CREATE TABLE 문의 열에 할당된 기본값으로 열을 업데이트합니다.

FROM tablelist

다른 테이블에 있는 정보를 참조하여 테이블을 업데이트할 수 있습니다. FROM 절에 다른 테이블을 나열하거나 WHERE 조건의 일부로서 하위 쿼리를 사용합니다. FROM 절에 나열된 테이블은 별칭을 가질 수 있습니다. 목록에 UPDATE 문의 대상 테이블을 포함할 필요가 있는 경우에는 별칭을 사용하십시오.

WHERE condition

조건과 일치하는 행으로 업데이트를 제한하는 선택적인 절입니다. 이 조건이 `true`를 반환할 때, 지정된 SET 열이 업데이트됩니다. 조건은 열에 대한 간단한 조건자 또는 하위 쿼리의 결과를 바탕으로 하는 조건일 수 있습니다.

UPDATE를 위한 대상 테이블을 포함하여, 하위 쿼리의 어떤 테이블이든 이름을 지정할 수 있습니다.

사용 노트

테이블에서 많은 수의 행을 업데이트한 후

- 테이블을 완전히 비워 스토리지 공간을 회수하고 행을 다시 정렬합니다.
- 테이블을 분석하여 쿼리 플래너에 대한 통계를 업데이트합니다.

왼쪽, 오른쪽 및 전체 외부 조인은 UPDATE 문의 FROM 절에서 지원되지 않고 다음 오류를 반환합니다.

```
ERROR: Target table must be part of an equijoin predicate
```

외부 조인을 지정해야 할 경우 UPDATE 문의 WHERE 절에 하위 쿼리를 사용하십시오.

UPDATE 문이 대상 테이블에 대한 자체 조인을 요구할 경우 조인 조건뿐 아니라 업데이트 작업을 위한 행을 정규화하는 WHERE 절 기준을 지정해야 합니다. 일반적으로, 대상 테이블이 그 자신이나 다른 테

이블에 조인될 때의 모범 사례는 업데이트를 위해 행을 정규화하는 기준에서 조인 조건을 분명히 분리하는 하위 쿼리를 사용하는 것입니다.

구성 파라미터 error_on_nondeterministic_update가 true로 설정된 경우 행당 여러 개의 일치 항목이 있는 UPDATE 쿼리에서 오류가 발생합니다. 자세한 내용은 error_on_nondeterministic_update 단원을 참조하십시오.

GENERATED BY DEFAULT AS IDENTITY 열을 업데이트할 수 있습니다. GENERATED BY DEFAULT AS IDENTITY로 정의된 열을 직접 입력하는 값으로 업데이트할 수 있습니다. 자세한 내용은 GENERATED BY DEFAULT AS IDENTITY 단원을 참조하십시오.

UPDATE 문 예

다음 예에 사용된 테이블에 대한 자세한 내용은 샘플 데이터베이스 섹션을 참조하세요.

TICKIT 데이터베이스의 CATEGORY 테이블은 다음 행을 포함합니다.

```
+-------+----------+-----------+--------------------------------------------+
| catid | catgroup |  catname  |                  catdesc                   |
+-------+----------+-----------+--------------------------------------------+
| 5     | Sports   | MLS       | Major League Soccer                        |
| 11    | Concerts | Classical | All symphony, concerto, and choir concerts |
| 1     | Sports   | MLB       | Major League Baseball                      |
| 6     | Shows    | Musicals  | Musical theatre                            |
| 3     | Sports   | NFL       | National Football League                   |
| 8     | Shows    | Opera     | All opera and light opera                  |
| 2     | Sports   | NHL       | National Hockey League                     |
| 9     | Concerts | Pop       | All rock and pop music concerts            |
| 4     | Sports   | NBA       | National Basketball Association            |
| 7     | Shows    | Plays     | All non-musical theatre                    |
| 10    | Concerts | Jazz      | All jazz singers and bands                 |
+-------+----------+-----------+--------------------------------------------+
```

값의 범위를 기반으로 테이블 업데이트

CATID 열에 있는 값의 범위를 기반으로 CATGROUP 열을 업데이트합니다.

```
UPDATE category
SET catgroup='Theatre'
WHERE catid BETWEEN 6 AND 8;

SELECT * FROM category
```

```
WHERE catid BETWEEN 6 AND 8;

+-------+----------+----------+---------------------------+
| catid | catgroup | catname  |          catdesc          |
+-------+----------+----------+---------------------------+
| 6     | Theatre  | Musicals | Musical theatre           |
| 7     | Theatre  | Plays    | All non-musical theatre   |
| 8     | Theatre  | Opera    | All opera and light opera |
+-------+----------+----------+---------------------------+
```

현재 값을 기반으로 테이블 업데이트

현재 CATGROUP 값을 기반으로 CATNAME 및 CATDESC 열을 업데이트합니다.

```
UPDATE category
SET catdesc=default, catname='Shows'
WHERE catgroup='Theatre';

SELECT * FROM category
WHERE catname='Shows';

+-------+----------+---------+---------+
| catid | catgroup | catname | catdesc |
+-------+----------+---------+---------+
| 6     | Theatre  | Shows   | NULL    |
| 7     | Theatre  | Shows   | NULL    |
| 8     | Theatre  | Shows   | NULL    |
+-------+----------+---------+---------+)
```

이 경우에는 테이블 생성 시 아무런 기본값도 정의되지 않았으므로 CATDESC 열이 null로 설정되었습니다.

다음 명령을 실행하여 CATEGORY 테이블 데이터를 다시 원래 값으로 설정합니다.

```
TRUNCATE category;

COPY category
FROM 's3://redshift-downloads/tickit/category_pipe.txt'
DELIMITER '|'
IGNOREHEADER 1
REGION 'us-east-1'
IAM_ROLE default;
```

WHERE 절 하위 쿼리의 결과를 기반으로 테이블 업데이트

WHERE 절에 있는 하위 쿼리의 결과를 기반으로 CATEGORY 테이블을 업데이트합니다.

```
UPDATE category
SET catdesc='Broadway Musical'
WHERE category.catid IN
(SELECT category.catid FROM category
JOIN event ON category.catid = event.catid
JOIN venue ON venue.venueid = event.venueid
JOIN sales ON sales.eventid = event.eventid
WHERE venuecity='New York City' AND catname='Musicals');
```

업데이트되는 테이블을 확인합니다.

```
SELECT * FROM category ORDER BY catid;

+-------+----------+-----------+--------------------------------------------+
| catid | catgroup |  catname  |                  catdesc                   |
+-------+----------+-----------+--------------------------------------------+
| 2     | Sports   | NHL       | National Hockey League                     |
| 3     | Sports   | NFL       | National Football League                   |
| 4     | Sports   | NBA       | National Basketball Association            |
| 5     | Sports   | MLS       | Major League Soccer                        |
| 6     | Shows    | Musicals  | Broadway Musical                           |
| 7     | Shows    | Plays     | All non-musical theatre                    |
| 8     | Shows    | Opera     | All opera and light opera                  |
| 9     | Concerts | Pop       | All rock and pop music concerts            |
| 10    | Concerts | Jazz      | All jazz singers and bands                 |
| 11    | Concerts | Classical | All symphony, concerto, and choir concerts |
+-------+----------+-----------+--------------------------------------------+
```

WITH 절 하위 쿼리의 결과를 기반으로 테이블 업데이트

WITH 절을 사용하여 하위 쿼리의 결과를 기반으로 CATEGORY 테이블을 업데이트하려면 다음 예를 사용하세요.

```
WITH u1 as (SELECT catid FROM event ORDER BY catid DESC LIMIT 1)
UPDATE category SET catid='200' FROM u1 WHERE u1.catid=category.catid;

SELECT * FROM category ORDER BY catid DESC LIMIT 1;
```

```
+-------+----------+---------+---------------------------------+
| catid | catgroup | catname |             catdesc             |
+-------+----------+---------+---------------------------------+
| 200   | Concerts | Pop     | All rock and pop music concerts |
+-------+----------+---------+---------------------------------+
```

조인 조건의 결과를 기반으로 테이블 업데이트

EVENT 테이블에서 일치하는 CATID 행을 기반으로 CATEGORY 테이블에서 원래의 행 11개를 업데이트합니다.

```
UPDATE category SET catid=100
FROM event
WHERE event.catid=category.catid;

SELECT * FROM category ORDER BY catid;

+-------+----------+-----------+---------------------------------------------+
| catid | catgroup |  catname  |                  catdesc                    |
+-------+----------+-----------+---------------------------------------------+
| 2     | Sports   | NHL       | National Hockey League                      |
| 3     | Sports   | NFL       | National Football League                    |
| 4     | Sports   | NBA       | National Basketball Association             |
| 5     | Sports   | MLS       | Major League Soccer                         |
| 10    | Concerts | Jazz      | All jazz singers and bands                  |
| 11    | Concerts | Classical | All symphony, concerto, and choir concerts  |
| 100   | Concerts | Pop       | All rock and pop music concerts             |
| 100   | Shows    | Plays     | All non-musical theatre                     |
| 100   | Shows    | Opera     | All opera and light opera                   |
| 100   | Shows    | Musicals  | Broadway Musical                            |
+-------+----------+-----------+---------------------------------------------+
```

EVENT 테이블이 FROM 절에 나열되어 있고 대상 테이블에 대한 조인 조건이 WHERE 절에 정의되어 있습니다. 업데이트 자격이 있는 행은 4개뿐입니다. 이들 4개의 행은 CATID 값이 원래 6, 7, 8 및 9였던 행으로, 해당되는 4개의 범주만 EVENT 테이블에 표시됩니다.

```
SELECT DISTINCT catid FROM event;

+-------+
| catid |
+-------+
| 6     |
```

```
| 7     |
| 8     |
| 9     |
+-------+
```

이전 예를 확장하고 WHERE 절에 다른 조건을 추가하여 CATEGORY 테이블에 있는 원래의 행 11개를 업데이트합니다. CATGROUP 열에 대한 제한 때문에, (4개의 행이 조인 자격이 있지만) 업데이트 자격이 있는 행은 1개뿐입니다.

```
UPDATE category SET catid=100
FROM event
WHERE event.catid=category.catid
AND catgroup='Concerts';

SELECT * FROM category WHERE catid=100;

+-------+----------+---------+---------------------------------+
| catid | catgroup | catname |             catdesc             |
+-------+----------+---------+---------------------------------+
| 100   | Concerts | Pop     | All rock and pop music concerts |
+-------+----------+---------+---------------------------------+
```

이 예를 작성하는 또 다른 방법은 다음과 같습니다.

```
UPDATE category SET catid=100
FROM event JOIN category cat ON event.catid=cat.catid
WHERE cat.catgroup='Concerts';
```

이 접근 방식의 이점은 조인 기준이 행의 업데이트 자격을 부여하는 다른 기준과는 분명히 구분된다는 점입니다. FROM 절에서 CATEGORY 테이블에 CAT라는 별칭을 사용한다는 점에 유의하십시오.

FROM 절에서 외부 조인으로 업데이트

이전 예에서는 UPDATE 문의 FROM 절에 지정된 내부 조인을 보여주었습니다. 다음 예에서는 FROM 절이 대상 테이블에 대한 외부 조인을 지원하지 않으므로 오류를 반환합니다.

```
UPDATE category SET catid=100
FROM event LEFT JOIN category cat ON event.catid=cat.catid
WHERE cat.catgroup='Concerts';
ERROR:  Target table must be part of an equijoin predicate
```

외부 조인이 UPDATE 문에 필요한 경우 외부 조인 구문을 하위 쿼리로 이동할 수 있습니다.

```
UPDATE category SET catid=100
FROM
(SELECT event.catid FROM event LEFT JOIN category cat ON event.catid=cat.catid)
 eventcat
WHERE category.catid=eventcat.catid
AND catgroup='Concerts';
```

SET 절에 있는 다른 테이블의 열로 업데이트

TICKIT 샘플 데이터베이스의 listing 테이블을 sales 테이블에 있는 값으로 업데이트하려면 다음 예시를 사용합니다.

```
SELECT listid, numtickets FROM listing WHERE sellerid = 1 ORDER BY 1 ASC LIMIT 5;

+--------+------------+
| listid | numtickets |
+--------+------------+
| 100423 | 4          |
| 108334 | 24         |
| 117150 | 4          |
| 135915 | 20         |
| 205927 | 6          |
+--------+------------+

UPDATE listing
SET numtickets = sales.sellerid
FROM sales
WHERE sales.sellerid = 1 AND listing.sellerid = sales.sellerid;

SELECT listid, numtickets FROM listing WHERE sellerid = 1 ORDER BY 1 ASC LIMIT 5;

+--------+------------+
| listid | numtickets |
+--------+------------+
| 100423 | 1          |
| 108334 | 1          |
| 117150 | 1          |
| 135915 | 1          |
| 205927 | 1          |
+--------+------------+
```

VACUUM

현재 데이터베이스에서 지정된 테이블이나 모든 테이블의 공간을 회수하고 행을 다시 정렬합니다.

> **Note**
>
> 필요한 테이블 권한이 있는 사용자만 테이블을 유효하게 정리(vacuum)할 수 있습니다. 필요한 테이블 권한 없이 VACUUM을 실행할 경우 작업은 성공적으로 완료되지만 아무런 효과도 없습니다. VACUUM을 유요하게 실행하기 위한 유효한 테이블 권한 목록은 다음 필수 권한 섹션을 참조하세요.

Amazon Redshift는 백그라운드에서 자동으로 데이터를 정렬하고 VACUUM DELETE를 실행합니다. 이렇게 하면 VACUUM 명령을 실행할 필요성이 줄어듭니다. 자세한 내용은 테이블 Vacuum 단원을 참조하십시오.

기본적으로 VACUUM은 테이블 행의 95% 이상이 이미 정렬된 테이블에 대해서는 정렬 단계를 건너뜁니다. 정렬 단계를 건너뛰면 VACUUM 성능을 상당히 개선할 수 있습니다. 단일 테이블의 기본 정렬 또는 삭제 임계값을 변경하려면 VACUUM을 실행할 때 테이블 이름과 TO threshold PERCENT 파라미터를 포함시킵니다.

테이블이 vacuum되는 동안 사용자는 테이블에 액세스할 수 있습니다. 테이블이 vacuum되는 중에도 쿼리 및 쓰기 작업을 수행할 수 있지만 DML(Data Manipulation Language)과 vacuum이 동시에 실행될 경우 두 작업 모두 시간이 더 걸릴 수 있습니다. vacuum 도중에 UPDATE 및 DELETE 문을 실행하는 경우 시스템 성능이 저하될 수 있습니다. VACUUM DELETE는 업데이트 작업과 삭제 작업을 임시로 차단합니다.

Amazon Redshift는 DELETE ONLY vacuum을 백그라운드로 수행합니다. 사용자가 ALTER TABLE 같은 DDL(Data Definition Language) 작업을 실행할 경우 자동 vacuum 작업은 일시 중지됩니다.

> **Note**
>
> Amazon Redshift VACUUM 명령 구문과 동작은 PostgreSQL VACUUM 작업과는 크게 다릅니다. 예를 들어 Amazon Redshift의 기본 VACUUM 작업은 VACUUM FULL로서, 디스크 공간을 회수하고 모든 열을 다시 정렬합니다. 이와는 달리, PostgreSQL에서 기본 VACUUM 작업은 단순히 공간을 회수하여 재사용할 수 있게 만듭니다.

자세한 내용은 테이블 Vacuum 단원을 참조하십시오.

필수 권한

VACUUM에 필요한 권한은 다음과 같습니다.

- 수퍼유저
- VACUUM 권한이 있는 사용자
- 테이블 소유자
- 테이블이 공유되는 데이터베이스 소유자

구문

```
VACUUM [ FULL | SORT ONLY | DELETE ONLY | REINDEX | RECLUSTER ]
[ [ table_name ] [ TO threshold PERCENT ] [ BOOST ] ]
```

파라미터

FULL

지정된 테이블(또는 현재 데이터베이스의 모든 테이블)을 정렬하고 이전 UPDATE 및 DELETE 작업에서 삭제 표시가 된 행이 점유한 디스크 공간을 회수합니다. VACUUM FULL이 기본값입니다.

전체를 비우는(full vacuum) 작업에서는 인터리브된 테이블에 대해 인덱스 재지정을 수행하지 않습니다. 인터리브된 테이블의 인덱스 재지정을 수행한 후 전체를 비우려면 VACUUM REINDEX 옵션을 사용하십시오.

기본적으로 VACUUM FULL은 95% 이상이 이미 정렬된 테이블에 대해서는 정렬 단계를 건너뜁니다. VACUUM이 정렬 단계를 건너뛸 수 있는 경우에는 DELETE ONLY를 수행하고 삭제 단계에서 남아 있는 행의 95% 이상이 삭제 대기 상태가 되지 않도록 공간을 회수합니다.

정렬 임계값이 충족되지 않고(예: 행의 90%가 정렬되는 경우) VACUUM이 전체 정렬을 수행하는 경우에는 완전 삭제 작업도 수행하여 100%의 삭제된 행으로부터 공간을 회수합니다.

단일 테이블에 대해서만 기본 vaccum 임계값을 변경할 수 있습니다. 단일 테이블의 기본 vacuum 임계값을 변경하려면 테이블 이름과 TO threshold PERCENT 파라미터를 포함시킵니다.

정렬 전용

행을 삭제하여 생긴 빈 공간을 회수하지 않고 지정된 테이블(또는 현재 데이터베이스의 모든 테이블)을 정렬합니다. 이 옵션은 디스크 공간 회수는 중요하지 않지만 새 행을 다시 정렬하는 것이 중요할 때 유용합니다. SORT ONLY vacuum은 정렬되지 않은 리전이 많은 수의 삭제된 행을 포함하지

않고 정렬된 리전 전체에 미치지 못할 때 vacuum 작업의 경과 시간을 줄여줍니다. 디스크 공간 제약 조건은 없지만 정렬된 테이블 행의 유지와 관련된 쿼리 최적화에 의존하는 애플리케이션은 이런 종류의 vacuum의 도움을 받을 수 있습니다.

기본적으로 VACUUM SORT ONLY는 이미 95% 이상 정렬되어 있는 테이블은 전부 건너뜁니다. 단일 테이블의 기본 정렬 임계값을 변경하려면 VACUUM을 실행할 때 테이블 이름과 TO threshold PERCENT 파라미터를 포함시킵니다.

DELETE ONLY

Amazon Redshift는 백그라운드에서 DELETE ONLY vacuum을 자동으로 수행하기 때문에 DELETE ONLY vacuum을 실행해야 하는 경우는 거의 없습니다.

VACUUM DELETE는 이전 UPDATE 및 DELETE 작업에서 삭제 표시가 된 행이 점유한 디스크 공간을 회수하고 테이블을 압축하여 사용된 공간을 확보합니다. DELETE ONLY vacuum 작업에서는 테이블 데이터를 정렬하지 않습니다.

디스크 공간 회수는 중요하지만 새 행을 다시 정렬하는 것이 중요하지 않을 때 이 옵션을 사용하면 VACUUM 작업의 경과 시간이 단축됩니다. 쿼리 성능이 이미 최적일 때도 이 옵션이 유용할 수 있으며, 쿼리 성능 최적화를 위해 행을 다시 정렬할 필요는 없습니다.

기본적으로, VACUUM DELETE ONLY는 나머지 행의 95% 이상이 삭제 표시되지 않도록 공간을 회수합니다. 단일 테이블의 기본 삭제 임계값을 변경하려면 VACUUM을 실행할 때 테이블 이름과 TO threshold PERCENT 파라미터를 포함시킵니다.

`ALTER TABLE APPEND` 같은 일부 작업은 테이블을 분할시킬 수 있습니다. `DELETE ONLY` 절을 사용하면, vacuum 작업이 분할된 테이블에서 공간을 회수합니다. 조각 모음 작업에 동일한 기준 값인 95%가 적용됩니다.

REINDEX tablename

인터리브된 정렬 키 열의 값 분산을 분석한 다음 전체 VACUUM 작업을 수행합니다. REINDEX가 사용되는 경우 테이블 이름이 필요합니다.

VACUUM REINDEX는 인터리브된 정렬 키를 분석하기 위한 추가적인 패스를 만들기 때문에 VACUUM FULL보다 상당히 더 많은 시간이 소요됩니다. 인터리브된 정렬은 복합 정렬보다 많은 행을 다시 배열해야 하므로 정렬 및 병합 작업이 인터리브된 테이블에 대해 더 오래 걸릴 수 있습니다.

VACUUM REINDEX 작업이 완료되기 전에 종료되면 다음 번 VACUUM은 완전 vacuum 작업을 수행하기 전에 reindex 작업을 재개합니다.

VACUUM REINDEX는 TO threshold PERCENT와 함께 지원되지 않습니다.

table_name

Vacuum을 수행할 테이블의 이름입니다. 테이블 이름을 지정하지 않을 경우 vacuum 작업은 현재 데이터베이스의 모든 테이블에 적용됩니다. 사용자가 만든 영구 또는 임시 테이블을 지정할 수 있습니다. 이 명령은 뷰 및 시스템 테이블과 같은 다른 객체에는 의미가 없습니다.

TO threshold PERCENT 파라미터를 포함하는 경우 테이블 이름은 필수입니다.

RECLUSTER tablename

정렬되지 않은 테이블 부분을 정렬합니다. 자동 테이블 정렬로 이미 정렬된 테이블 부분은 그대로 유지됩니다. 이 명령은 새로 정렬된 데이터를 정렬된 영역과 병합하지 않습니다. 또한 삭제 표시된 모든 공간을 회수하지 않습니다. 이 명령이 완료되면 SVV_TABLE_INFO의 `unsorted` 필드에 표시된 대로 테이블이 완전히 정렬되어 표시되지 않을 수 있습니다.

수집이 잦은 큰 테이블과 가장 최근 데이터에만 액세스하는 쿼리에는 VACUUM RECLUSTER를 사용하는 것이 좋습니다.

VACUUM RECLUSTER는 TO threshold PERCENT와 함께 지원되지 않습니다. RECLUSTER가 사용되는 경우 테이블 이름이 필요합니다.

VACUUM RECLUSTER는 인터리브 정렬 키가 있는 테이블 및 ALL 배포 스타일의 테이블에서 지원되지 않습니다.

table_name

Vacuum을 수행할 테이블의 이름입니다. 사용자가 만든 영구 또는 임시 테이블을 지정할 수 있습니다. 이 명령은 뷰 및 시스템 테이블과 같은 다른 객체에는 의미가 없습니다.

TO threshold PERCENT

VACUUM이 그 이상이 되면 정렬 단계를 건너뛰는 임계값과 삭제 단계에서 공간을 회수하기 위한 목표 임계값을 지정하는 절입니다. 정렬 임계값은 vacuum을 실행하기 전에 지정된 테이블에 대한 정렬 순서가 이미 지정되어 있는 전체 행의 비율입니다. 삭제 임계값은 vacuum 실행 후에 삭제 표시되지 않은 전체 행의 최소 비율입니다.

VACUUM은 테이블에서 정렬된 행의 비율이 정렬 임계값보다 낮을 때만 행을 다시 정렬하므로, Amazon Redshift는 종종 VACUUM 횟수를 상당히 줄일 수 있습니다. 마찬가지로, VACUUM이 삭제 표시된 행의 100%에서 공간을 회수하도록 제한되어 있지 않을 때는 몇 개의 삭제된 행만 포함한 블록 다시 쓰기를 건너뛸 수 있는 경우가 많습니다.

예를 들어, 임계값에 대해 75를 지정한 경우 VACUUM은 테이블 행의 75% 이상이 이미 정렬되어 있는 경우에 정렬 단계를 건너뜁니다. 삭제 단계의 경우, VACUUMS는 테이블 행의 75% 이상이

vacuum 후에 삭제 표시되지 않도록 디스크 공간 회수 목표를 설정합니다. 임계값은 0 ~ 100 사이의 정수여야 합니다. 기본값은 95입니다. 100의 값을 지정하는 경우 VACUUM은 이미 완전히 정렬되어 있지 않는 한 항상 테이블을 정렬하고 삭제 표시된 모든 행에서 공간을 회수합니다. 0의 값을 지정하는 경우 VACUUM은 테이블을 절대 정렬하지 않고 공간을 절대 회수하지 않습니다.

TO threshold PERCENT 파라미터를 포함하는 경우 테이블 이름도 지정해야 합니다. 테이블 이름이 생략된 경우 VACUUM은 실패합니다.

TO 임계값 PERCENT 파라미터를 REINDEX와 함께 사용할 수 없습니다.

BOOST

사용 가능한 메모리 및 디스크 공간과 같은 추가 리소스로 VACUUM 명령을 실행합니다. BOOST 옵션을 사용하면 VACUUM은 하나의 창에서 작동하며 VACUUM 작업 기간 동안 동시 삭제 및 업데이트를 차단합니다. BOOST 옵션을 사용하여 실행하면 시스템 리소스 경합이 발생하여 쿼리 성능에 영향을 줄 수 있습니다. 유지 보수 작업 등 시스템 부하가 적을 때 VACUUM BOOST를 실행하십시오.

BOOST 옵션을 사용할 때는 다음을 고려하십시오.

- BOOST가 지정되면 table_name 값이 필요합니다.
- BOOST는 REINDEX와 함께 사용할 수 없습니다.
- DELETE ONLY를 사용할 경우 BOOST가 무시됩니다.

사용 노트

대부분의 Amazon Redshift 애플리케이션의 경우 완전 vacuum이 권장됩니다. 자세한 내용은 테이블 Vacuum 단원을 참조하십시오.

Vacuum 작업 실행 전, 다음 동작에 유의하십시오.

- 트랜잭션 블록(BEGIN ... END) 내에서 VACUUM을 실행할 수 없습니다. 버전 관리에 대한 자세한 내용은 직렬화 가능 격리 단원을 참조하십시오.
- 특정 시점에 클러스터당 한 개의 VACUUM 명령만 실행할 수 있습니다. 여러 vacuum 작업을 동시에 실행하려고 하면 Amazon Redshift가 오류를 반환합니다.
- 테이블이 vacuum될 때 테이블 크기가 어느 정도 증가할 수도 있습니다. 이는 회수할 삭제된 행이 없거나 테이블의 새 정렬 순서로 인해 데이터 압축률이 낮아질 때 예상되는 동작입니다.
- Vacuum 작업 중에는 쿼리 성능이 어느 정도 저하될 것으로 예상됩니다. Vacuum 작업이 완료되는 즉시 성능은 정상적인 수준으로 회복됩니다.

- Vacuum 작업 중에 동시 쓰기 작업이 진행되지만 vacuum 중에는 쓰기 작업을 수행하지 않는 것이 좋습니다. Vacuum 실행 전에 쓰기 작업을 완료하는 것이 더 효율적입니다. 또한 vacuum 작업이 시작된 후에 기록된 데이터는 해당 작업으로 vacuum을 수행할 수 없습니다. 이 경우 두 번째 vacuum 작업이 필요합니다.
- 로드 또는 삽입 작업이 이미 진행 중인 경우 vacuum 작업을 시작하지 못할 수도 있습니다. Vacuum 작업을 시작하려면 일시적으로 테이블에 배타적으로 액세스해야 합니다. 이런 배타적 액세스는 잠깐 필요할 뿐이므로, vacuum 작업이 상당한 시간 동안 동시 로드 및 삽입을 막는 것은 아닙니다.
- 특정 테이블에 대해 수행할 작업이 없을 때는 vacuum 작업을 건너뛰지만, 작업을 건너뛸 수 있음을 발견하는 것과 관련된 오버헤드가 있습니다. 테이블이 초기 상태이거나 vacuum 임계값을 충족하지 못한다는 점을 알고 있다면 이 테이블에 대한 vacuum 작업을 실행하지 마십시오.
- 작은 테이블에서의 DELETE ONLY vacuum 작업은 특히 테이블에 다수의 열이 있거나 클러스터가 노드당 많은 수의 조각을 사용할 때 데이터 저장에 사용되는 블록의 수를 줄이지 못할 수 있습니다. 이러한 vacuum 작업은 테이블로의 동시 삽입 수를 설명하기 위해 조각당 각 열에 한 개씩의 블록을 추가하고, 이 오버헤드가 회수된 디스크 공간에서 블록 수의 감소보다 많아질 가능성이 있습니다. 예를 들어 8개의 노드로 구성된 클러스터에 10개의 열이 있는 테이블이 vacuum 이전에 1,000개의 블록을 차지하고 있는 경우 삭제된 행 때문에 80개를 초과하는 블록의 디스크 공간이 회수되지 않을 경우 vacuum이 실제 블록 수를 줄여주지는 않습니다. (각 데이터 블록이 1MB씩 사용합니다.)

다음 중 어떤 조건이라도 충족이 될 경우 자동 vaccum 작업은 일시 중지됩니다.

- 사용자가 ALTER TABLE 같이 현재 자동 vaccum이 작동 중인 테이블에 대해 단독 잠금이 필요한 데이터 정의 언어(DDL) 작업을 실행합니다.
- 사용자가 클러스터의 어떤 테이블에서든 VACUUM을 트리거합니다(한 번에 오직 하나의 VACCUM만 실행 가능).
- 클러스터 로드가 많은 기간입니다.

예시

기본값인 95%의 VACUUM 임계값을 기준으로 모든 테이블에서 공간과 데이터베이스를 회수하고 행을 다시 정렬합니다.

```
vacuum;
```

기본값인 95%의 임계값을 기준으로 SALES 테이블에서 공간을 회수하고 행을 다시 정렬합니다.

```
vacuum sales;
```

항상 SALES 테이블에서 공간을 회수하고 행을 다시 정렬합니다.

```
vacuum sales to 100 percent;
```

이미 정렬된 행 비율이 75% 미만인 경우에만 SALES 테이블의 행을 다시 정렬합니다.

```
 vacuum sort only sales to 75 percent;
```

Vacuum 후에 남아 있는 행의 75% 이상이 삭제 표시되지 않도록 SALES 테이블에서 공간을 회수합니다.

```
vacuum delete only sales to 75 percent;
```

LISTING 테이블을 reindex한 다음 vacuum합니다.

```
vacuum reindex listing;
```

다음 명령은 오류를 반환합니다.

```
vacuum reindex listing to 75 percent;
```

LISTING 테이블을 다시 클러스터링한 다음 정리(vacuum)합니다.

```
vacuum recluster listing;
```

BOOST 옵션으로 LISTING 테이블을 다시 클러스터링한 다음 정리(vacuum)합니다.

```
vacuum recluster listing boost;
```

SQL 함수 참조

주제

- 리더 노드 전용 함수
- 집계 함수
- 배열 함수
- 비트 단위 집계 함수
- 조건식

- 데이터 형식 지정 함수
- 날짜 및 시간 함수
- 해시 함수
- HyperLogLog 함수
- JSON 함수
- 기계 학습 함수
- 수학 함수
- 객체 함수
- 공간 함수
- 문자열 함수
- SUPER 형식 정보 함수
- VARBYTE 함수 및 연산자
- 윈도 함수
- 시스템 관리 함수
- 시스템 정보 함수

Amazon Redshift는 SQL 표준을 확장한 다양한 함수를 비롯해 표준 집계 함수, 스칼라 함수, 윈도 함수 등을 지원합니다.

> **Note**
>
> Amazon Redshift는 PostgreSQL을 기반으로 합니다. Amazon Redshift와 PostgreSQL은 데이터웨어 하우스 애플리케이션을 설계하고 개발할 때 숙지해야 할 몇 가지 매우 중요한 차이점이 있습니다. Amazon Redshift SQL이 PostgreSQL과 어떻게 다른지 자세히 알아보려면 Amazon Redshift 및 PostgreSQL 섹션을 참조하세요.

리더 노드 전용 함수

일부 Amazon Redshift 쿼리는 컴퓨팅 노드에서 분산되고 실행됩니다. 다른 쿼리는 리더 노드에서만 실행됩니다.

리더 노드는 쿼리가 사용자 생성 테이블 또는 시스템 테이블(STL 또는 STV 접두사가 첨부된 테이블과 SVL 또는 SVV 접두사가 첨부된 시스템 뷰)을 참조할 때 SQL을 컴퓨팅 노드로 분산시킵니다.

CATALOG 테이블(PG_TABLE_DEF처럼 PG 접두사가 첨부된 테이블)만 참조하거나 어떤 테이블도 참조하지 않는 쿼리는 리더 노드에서만 실행됩니다.

일부 Amazon Redshift SQL 함수는 리더 노드에서만 지원되고, 컴퓨팅 노드에서는 지원되지 않습니다. 따라서 리더 노드 함수를 사용하는 쿼리는 컴퓨팅 노드가 아닌 리더 노드에서만 실행해야 합니다. 그렇지 않으면 오류를 반환합니다.

각 리더 노드 전용 함수에 대한 문서에는 이 함수가 사용자 정의 테이블이나 Amazon Redshift 시스템 테이블을 참조할 경우 오류를 반환한다는 설명이 포함되어 있습니다.

자세한 내용은 리더 노드에서 지원되는 SQL 함수 단원을 참조하십시오.

다음 SQL 함수는 리더 노드 전용 함수이며, 컴퓨팅 노드에서는 지원되지 않습니다.

시스템 정보 함수

- CURRENT_SCHEMA
- CURRENT_SCHEMAS
- HAS_DATABASE_PRIVILEGE
- HAS_SCHEMA_PRIVILEGE
- HAS_TABLE_PRIVILEGE

문자열 함수

- SUBSTR

수학 함수

- FACTORIAL()

다음 리더 노드 전용 함수는 여기에서 다루지 않으며 더 이상 지원되지 않습니다.

날짜 함수

- AGE
- CURRENT_TIME
- CURRENT_TIMESTAMP
- LOCALTIME

- ISFINITE
- NOW

문자열 함수

- GETBIT
- GET_BYTE
- SET_BIT
- SET_BYTE
- TO_ASCII

집계 함수

주제

집계 함수는 입력 값 집합에서 단일 결과 값을 계산합니다.

집계 함수를 사용하는 SELECT 문에는 옵션이지만 GROUP BY와 HAVING, 2가지 절이 포함될 수 있습니다. 이 2가지 절의 구문은 다음과 같습니다(예에서는 COUNT 함수 사용).

```
SELECT count (*) expression FROM table_reference
```

```
WHERE condition [GROUP BY expression ] [ HAVING condition]
```

GROUP BY 절은 집계 후 특정 열의 고유 값을 기준으로 결과를 그룹화합니다. HAVING 절은 반환되는 결과를 수량(*) > 1처럼 특정 집계 조건이 true인 행으로 제한합니다. 또한 WHERE와 동일한 방식으로 사용되어 열의 값에 따라 행을 제한합니다. 이러한 추가 절의 예는 COUNT 섹션을 참조하세요.

집계 함수는 중첩 집계 함수나 창 함수를 인수로 사용하지 않습니다.

ANY_VALUE 함수

ANY_VALUE 함수는 입력 표현식 값에서 비결정적으로 값을 반환합니다. 이 함수는 입력 식으로 반환되는 행이 없는 경우 NULL을 반환합니다. 입력 식에 NULL 값이 있는 경우 함수가 NULL을 반환할 수도 있습니다.

구문

```
ANY_VALUE( [ DISTINCT | ALL ] expression )
```

인수

DISTINCT | ALL

입력 표현식 값에서 값을 반환하려면 DISTINCT 또는 ALL을 지정합니다. DISTINCT 인수는 효과가 없으며 무시됩니다.

표현식

함수가 실행되는 대상 열 또는 표현식입니다. 표현식은 다음 데이터 유형 중 하나입니다.

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- REAL
- DOUBLE PRECISON
- BOOLEAN
- CHAR
- VARCHAR

- 날짜
- TIMESTAMP
- TIMESTAMPTZ
- TIME
- TIMETZ
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND
- VARBYTE
- SUPER
- HLLSKETCH
- GEOMETRY
- GEOGRAPHY

반환

expression과 동일한 데이터 형식을 반환합니다.

사용 노트

열에 대한 ANY_VALUE 함수를 지정하는 문이 두 번째 열 참조도 포함하는 경우 두 번째 열은 GROUP BY 절에 나타나거나 집계 함수에 포함되어야 합니다.

예시

이 예에서는 Amazon Redshift 시작 안내서의 4단계: Amazon S3에서 샘플 데이터 로드에서 생성된 이벤트 테이블을 사용합니다. 다음 예는 eventname이 Eagles인 모든 dateid의 인스턴스를 반환합니다.

```
select any_value(dateid) as dateid, eventname from event where eventname ='Eagles'
 group by eventname;
```

다음은 결과입니다.

```
dateid | eventname
-------+---------------
 1878  | Eagles
```

다음 예는 eventname이 Eagles 또는 Cold War Kids인 모든 dateid의 인스턴스를 반환합니다.

```
select any_value(dateid) as dateid, eventname from event where eventname in('Eagles',
 'Cold War Kids') group by eventname;
```

다음은 결과입니다.

```
dateid | eventname
-------+--------------
 1922  | Cold War Kids
 1878  | Eagles
```

APPROXIMATE PERCENTILE_DISC 함수

APPROXIMATE PERCENTILE_DISC는 이산 분포 모델을 가정하는 역분포 함수로서 백분위 값과 정렬 명세를 가지며, 지정된 집합에서 요소를 반환합니다. 이 함수는 근사치를 사용하기 때문에 실행 속도가 더욱 빠르며 상대 오차도 약 0.5%로 낮습니다.

APPROXIMATE PERCENTILE_DISC는 임의의 percentile 값에 대해 사분위 요약 알고리즘을 사용하여 ORDER BY 절에서 표현식의 이산 백분위에 대한 근사치를 구합니다. 또한 동일한 정렬 명세와 관련하여 가장 작지만 percentile보다는 크거나 같은 누적 분포 값을 반환합니다.

구문

```
APPROXIMATE  PERCENTILE_DISC ( percentile )
WITHIN GROUP (ORDER BY expr)
```

인수

Percentile

0과 1 사이의 숫자 상수입니다. 이 계산에서 Null 값은 무시됩니다.

WITHIN GROUP (ORDER BY expr)

숫자 또는 날짜/시간 값을 지정하여 백분위를 정렬 및 계산하는 절입니다.

반환

WITHIN GROUP 절의 ORDER BY 표현식과 동일한 데이터 형식

사용 노트

APPROXIMATE PERCENTILE_DISC 문에 GROUP BY 절이 포함된 경우에는 결과 집합이 제한적입니다. 이러한 제한은 노드 유형과 노드 수에 따라 달라집니다. 제한을 초과하면 함수가 중단되고 다음과 같은 오류를 반환합니다.

```
GROUP BY limit for approximate percentile_disc exceeded.
```

제한을 초과하여 더 많은 그룹을 평가해야 하는 경우에는 PERCENTILE_CONT 함수를 사용하는 것이 좋습니다.

예시

다음은 상위 10개 날짜일 때 판매 수량과 총 판매액, 그리고 50번째 백분위 값을 반환하는 예입니다.

```
select top 10 date.caldate,
count(totalprice), sum(totalprice),
approximate percentile_disc(0.5)
within group (order by totalprice)
from listing
join date on listing.dateid = date.dateid
group by date.caldate
order by 3 desc;

caldate    | count | sum        | percentile_disc
-----------+-------+------------+----------------
2008-01-07 |   658 | 2081400.00 |         2020.00
2008-01-02 |   614 | 2064840.00 |         2178.00
2008-07-22 |   593 | 1994256.00 |         2214.00
2008-01-26 |   595 | 1993188.00 |         2272.00
2008-02-24 |   655 | 1975345.00 |         2070.00
2008-02-04 |   616 | 1972491.00 |         1995.00
2008-02-14 |   628 | 1971759.00 |         2184.00
2008-09-01 |   600 | 1944976.00 |         2100.00
2008-07-29 |   597 | 1944488.00 |         2106.00
2008-07-23 |   592 | 1943265.00 |         1974.00
```

AVG 함수

AVG 함수는 입력 표현식 값의 평균(산술 평균)을 반환합니다. AVG 함수는 숫자 값을 사용하고 NULL 값을 무시합니다.

구문

```
AVG ( [ DISTINCT | ALL ] expression )
```

인수

표현식

함수가 실행되는 대상 열 또는 표현식입니다. 표현식은 다음 데이터 유형 중 하나입니다.

- SMALLINT
- INTEGER
- BIGINT
- NUMERIC
- DECIMAL
- REAL
- DOUBLE PRECISON
- SUPER

DISTINCT | ALL

인수가 DISTINCT일 때는 함수가 평균을 계산하기 전에 지정한 표현식에서 중복 값을 모두 제거합니다. 인수가 ALL일 때는 함수가 표현식의 모든 중복 값을 그대로 유지한 채 평균을 계산합니다. ALL이 기본값입니다.

데이터 타입

AVG 함수에서 지원되는 인수 형식은 SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL, REAL, DOUBLE PRECISION, SUPER입니다.

AVG 함수에서 지원되는 반환 형식은 다음과 같습니다.

- 모든 정수형 인수일 때 BIGINT
- 부동 소수점 인수일 때 DOUBLE PRECISION
- 다른 인수 형식의 표현과 동일한 데이터 형식을 반환합니다.

NUMERIC 또는 DECIMAL 인수가 포함된 AVG 함수 결과의 기본 정밀도는 38입니다. 함수 결과의 비율은 인수 비율과 동일합니다. 예를 들어 DEC(5,2) 열의 AVG는 DEC(38,2) 데이터 형식을 반환합니다.

예시

SALES 테이블에서 트랜잭션 1회당 판매된 평균 수량을 구합니다.

```
select avg(qtysold)from sales;

avg
-----
2
(1 row)
```

모든 목록에 나열된 평균 총 가격을 구합니다.

```
select avg(numtickets*priceperticket) as avg_total_price from listing;

avg_total_price
-----------------
3034.41
(1 row)
```

매월 내림차순으로 분류된 평균 지불 가격을 구합니다.

```
select avg(pricepaid) as avg_price, month
from sales, date
where sales.dateid = date.dateid
group by month
order by avg_price desc;

avg_price | month
----------+-------
659.34 | MAR
655.06 | APR
645.82 | JAN
643.10 | MAY
642.72 | JUN
642.37 | SEP
640.72 | OCT
640.57 | DEC
635.34 | JUL
635.24 | FEB
634.24 | NOV
632.78 | AUG
```

```
(12 rows)
```

COUNT 함수

COUNT 함수는 표현식에서 정의하는 행의 수를 계산합니다.

COUNT 함수는 다음과 같은 변형이 있습니다.

- COUNT (*)는 NULL 값의 유무에 상관없이 대상 테이블에서 모든 행의 수를 계산합니다.
- COUNT (expression)는 특정 열 또는 표현식에서 NULL을 제외한 값이 포함된 행의 수를 계산합니다.
- COUNT (DISTINCT expression)는 임의의 열 또는 표현식에서 NULL을 제외한 고유 값의 수를 계산합니다.
- APPROXIMATE COUNT DISTINCT는 임의의 열 또는 표현식에서 NULL을 제외한 고유 값의 수를 대략적으로 구합니다.

구문

```
COUNT( * | expression )
```

```
COUNT ( [ DISTINCT | ALL ] expression )
```

```
APPROXIMATE COUNT ( DISTINCT expression )
```

인수

표현식

함수가 실행되는 대상 열 또는 표현식입니다. COUNT 함수는 모든 인수 데이터 형식을 지원합니다.

DISTINCT | ALL

인수가 DISTINCT일 때는 행의 수를 계산하기 전에 지정한 표현식에서 중복 값을 모두 제거합니다. 인수가 ALL일 때는 함수가 표현식의 모든 중복 값을 그대로 유지한 채 행의 수를 계산합니다. ALL이 기본값입니다.

APPROXIMATE

APPROXIMATE와 함께 사용할 때는 COUNT DISTINCT 함수가 HyperLogLog 알고리즘을 사용하여 임의의 열 또는 표현식에서 NULL을 제외한 고유 값의 수를 대략적으로 구합니다. 쿼리에

APPROXIMATE 키워드를 사용하면 실행 속도가 빨라질 뿐만 아니라 상대 오차도 약 2%로 낮습니다. 쿼리마다, 혹은 GROUP BY 절이 있는 경우 그룹마다 수백만 개가 넘는 고유 값을 반환하는 쿼리일 때는 근사치가 타당한 것으로 간주됩니다. 하지만 고유 값이 수천 개로 적을 경우에는 근사치일 때 속도가 정확한 행의 수일 때 보다 느려질 수 있습니다. APPROXIMATE는 COUNT DISTINCT와만 사용할 수 있습니다.

반환 타입

COUNT 함수는 BIGINT를 반환합니다.

예시

Florida 주의 모든 사용자 수를 계산합니다.

```
select count(*) from users where state='FL';

count
-------
510
```

EVENT 테이블에서 모든 이벤트 이름의 수를 계산합니다.

```
select count(eventname) from event;

count
-------
8798
```

EVENT 테이블에서 모든 이벤트 이름의 수를 계산합니다.

```
select count(all eventname) from event;

count
-------
8798
```

EVENT 테이블에서 고유한 장소 ID의 수를 모두 계산합니다.

```
select count(distinct venueid) as venues from event;
```

```
venues
--------
204
```

개별 판매자가 4장 이상의 티켓을 한 묶음으로 판매한 횟수를 계산합니다. 결과는 판매자 ID로 구분합니다.

```
select count(*), sellerid from listing
where numtickets > 4
group by sellerid
order by 1 desc, 2;

count | sellerid
------+----------
12    |    6386
11    |    17304
11    |    20123
11    |    25428
...
```

다음은 COUNT와 APPROXIMATE COUNT의 반환 값 및 실행 시간을 서로 비교한 예입니다.

```
select  count(distinct pricepaid) from sales;

count
-------
  4528


Time: 48.048 ms


select approximate count(distinct pricepaid) from sales;

count
-------
  4553


Time: 21.728 ms
```

LISTAGG 함수

LISTAGG 집계 함수는 ORDER BY 표현식에 따라 쿼리 내 각 그룹의 행 순서를 지정한 다음, 값을 연결하여 문자열 하나를 만듭니다.

구문

```
LISTAGG( [DISTINCT] aggregate_expression [, 'delimiter' ] )
[ WITHIN GROUP (ORDER BY order_list) ]
```

인수

DISTINCT

연결하기 전에 지정된 표현식에서 중복 값을 없애는 절입니다. 후행 공백은 무시됩니다. 예를 들어, 문자열 `'a'` 및 `'a '`는 중복으로 처리됩니다. LISTAGG는 발생한 첫 번째 값을 사용합니다. 자세한 내용은 후행 공백의 중요성 단원을 참조하십시오.

aggregate_expression

집계할 값을 제공하는 모든 유효 표현식(열 이름 등)입니다. NULL 값과 빈 문자열은 무시됩니다.

delimiter

연결된 값을 구분하는 문자열 상수입니다. 기본값은 NULL입니다.

WITHIN GROUP (ORDER BY order_list)

집계된 값의 정렬 순서를 지정하는 절입니다.

반환

VARCHAR(최대). 결과 집합이 최대 VARCHAR 크기보다 클 경우에는 LISTAGG가 다음과 같은 오류를 반환합니다.

```
Invalid operation: Result size exceeds LISTAGG limit
```

사용 노트

- 문에 WITHIN GROUP 절을 사용하는 LISTAGG 함수가 다수 포함된 경우에는 WITHIN GROUP 절마다 동일한 ORDER BY 값을 사용해야 합니다.

예를 들어 다음과 같은 문은 오류를 반환합니다.

```
SELECT LISTAGG(sellerid)
WITHIN GROUP (ORDER BY dateid) AS sellers,
LISTAGG(dateid)
WITHIN GROUP (ORDER BY sellerid) AS dates
FROM sales;
```

다음 문은 성공적으로 실행됩니다.

```
SELECT LISTAGG(sellerid)
WITHIN GROUP (ORDER BY dateid) AS sellers,
LISTAGG(dateid)
WITHIN GROUP (ORDER BY dateid) AS dates
FROM sales;

SELECT LISTAGG(sellerid)
WITHIN GROUP (ORDER BY dateid) AS sellers,
LISTAGG(dateid) AS dates
FROM sales;
```

예시

다음은 판매자 ID에 따라 순서를 지정하여 판매자 ID를 집계하는 예입니다.

```
SELECT LISTAGG(sellerid, ', ')
WITHIN GROUP (ORDER BY sellerid)
FROM sales
WHERE eventid = 4337;

listagg

-------------------------------------------------------------------------------------------------
380, 380, 1178, 1178, 1178, 2731, 8117, 12905, 32043, 32043, 32043, 32432, 32432,
 38669, 38750, 41498, 45676, 46324, 47188, 47188, 48294
```

다음 예에서는 DISTINCT를 사용하여 고유한 판매자 ID 목록을 반환합니다.

```
SELECT LISTAGG(DISTINCT sellerid, ', ')
WITHIN GROUP (ORDER BY sellerid)
```

```
FROM sales
WHERE eventid = 4337;

listagg

-----------------------------------------------------------------------------------------------
380, 1178, 2731, 8117, 12905, 32043, 32432, 38669, 38750, 41498, 45676, 46324, 47188,
 48294
```

다음은 날짜 순으로 판매자 ID를 집계하는 예입니다.

```
SELECT LISTAGG(sellerid, ', ')
WITHIN GROUP (ORDER BY dateid)
FROM sales
WHERE eventid = 4337;

   listagg
-----------------------------------------------------------------------------------------------
 41498, 47188, 47188, 1178, 1178, 1178, 380, 45676, 46324, 48294, 32043, 32043, 32432,
 12905, 8117, 38750, 2731, 32432, 32043, 380, 38669
```

다음은 ID가 660인 구매자의 판매 날짜 목록을 파이프로 구분하여 반환하는 예입니다.

```
SELECT LISTAGG(
    (SELECT caldate FROM date WHERE date.dateid=sales.dateid), ' | '
)
WITHIN GROUP (ORDER BY sellerid DESC, salesid ASC)
FROM sales
WHERE buyerid = 660;

             listagg
-------------------------------------------------
2008-07-16 | 2008-07-09 | 2008-01-01 | 2008-10-26
```

다음은 각 구매자 ID 660, 661, 662의 판매 ID 목록을 쉼표로 구분하여 반환하는 예입니다.

```
SELECT buyerid,
LISTAGG(salesid,', ')
WITHIN GROUP (ORDER BY salesid) AS sales_id
FROM sales
WHERE buyerid BETWEEN 660 AND 662
GROUP BY buyerid
```

```
ORDER BY buyerid;

buyerid |                sales_id
--------+-----------------------------------------------------
660     | 32872, 33095, 33514, 34548
661     | 19951, 20517, 21695, 21931
662     | 3318, 3823, 4215, 51980, 53202, 55908, 57832, 171603
```

MAX 함수

MAX 함수는 행 집합에서 최댓값을 반환합니다. DISTINCT 또는 ALL은 사용할 수 있지만 결과에 아무런 영향도 끼치지 않습니다.

구문

```
MAX ( [ DISTINCT | ALL ] expression )
```

인수

표현식

함수가 실행되는 대상 열 또는 표현식입니다. 표현식은 다음 데이터 유형 중 하나입니다.

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- REAL
- DOUBLE PRECISON
- CHAR
- VARCHAR
- 날짜
- TIMESTAMP
- TIMESTAMPTZ
- TIME
- TIMETZ
- VARBYTE

- SUPER

DISTINCT | ALL

인수가 DISTINCT일 때는 함수가 최댓값을 계산하기 전에 지정한 표현식에서 중복 값을 모두 제거합니다. 인수가 ALL일 때는 함수가 표현식의 모든 중복 값을 그대로 유지한 채 최댓값을 계산합니다. ALL이 기본값입니다.

데이터 타입

expression과 동일한 데이터 형식을 반환합니다. MIN 함수일 때 부울 등가는 BOOL_AND 함수이고, MAX 함수일 때 부울 등가는 BOOL_OR 함수입니다.

예시

모든 판매에서 지불된 최고 가격을 구합니다.

```
select max(pricepaid) from sales;

max
----------
12624.00
(1 row)
```

모든 판매에서 티켓 1장당 지불된 최고 가격을 구합니다.

```
select max(pricepaid/qtysold) as max_ticket_price
from sales;

max_ticket_price
-----------------
2500.00000000
(1 row)
```

MEDIAN 함수

값 범위의 중앙값을 계산합니다. 범위의 NULL 값은 무시됩니다.

MEDIAN은 연속 분포 모델을 가정하는 역분포 함수입니다.

MEDIAN은 PERCENTILE_CONT의 특별 사례입니다.

구문

```
MEDIAN(median_expression)
```

인수

median_expression

함수가 실행되는 대상 열 또는 표현식입니다.

데이터 타입

반환 형식은 median_expression의 형식에 따라 결정됩니다. 다음 표는 각 median_expression 데이터 형식에 따른 반환 형식을 나타낸 것입니다.

입력 유형	반환 타입
INT2, INT4, INT8, NUMERIC, DECIMAL	DECIMAL
FLOAT, DOUBLE	DOUBLE
DATE	DATE
TIMESTAMP	TIMESTAMP
TIMESTAMPTZ	TIMESTAMPTZ

사용 노트

median_expression 인수가 최대 정밀도가 38자리로 정의된 DECIMAL 데이터 형식인 경우에는 MEDIAN이 부정확한 결과 또는 오류를 반환합니다. MEDIAN 함수의 반환 값이 38자리를 초과하면 정밀도가 손실될 수도 있기 때문에 알맞은 자리 수로 결과가 잘립니다. 보간 도중 중간 결과가 최대 정밀도를 초과하면 수치 오버플로우가 발생하고 함수는 오류를 반환합니다. 이러한 상황을 방지하려면 정밀도가 낮은 데이터 형식을 사용하거나, 혹은 median_expression 인수를 낮은 정밀도로 변환합니다.

하나의 문에서 정렬 기반 집계 함수(LISTAGG, PERCENTILE_CONT, MEDIAN)를 여러 차례 호출하는 경우에는 모두 동일한 ORDER BY 값을 사용해야 합니다. 단, MEDIAN은 표현식 값에서 묵시적인 ORDER BY를 적용합니다.

예를 들어 다음과 같은 문은 오류를 반환합니다.

```
SELECT TOP 10 salesid, SUM(pricepaid),
PERCENTILE_CONT(0.6) WITHIN GROUP(ORDER BY salesid),
MEDIAN(pricepaid)
FROM sales
GROUP BY salesid, pricepaid;

An error occurred when executing the SQL command:
SELECT TOP 10 salesid, SUM(pricepaid),
PERCENTILE_CONT(0.6) WITHIN GROUP(ORDER BY salesid),
MEDIAN(pricepaid)
FROM sales
GROUP BY salesid, pricepaid;

ERROR: within group ORDER BY clauses for aggregate functions must be the same
```

다음 문은 성공적으로 실행됩니다.

```
SELECT TOP 10 salesid, SUM(pricepaid),
PERCENTILE_CONT(0.6) WITHIN GROUP(ORDER BY salesid),
MEDIAN(salesid)
FROM sales
GROUP BY salesid, pricepaid;
```

예시

다음 예제에서는 TICKIT 샘플 데이터베이스를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

다음은 MEDIAN이 PERCENTILE_CONT(0.5)와 동일한 결과를 산출하는 예입니다.

```
SELECT TOP 10 DISTINCT sellerid, qtysold,
PERCENTILE_CONT(0.5) WITHIN GROUP(ORDER BY qtysold),
MEDIAN(qtysold)
FROM sales
GROUP BY sellerid, qtysold;

+----------+---------+-----------------+--------+
| sellerid | qtysold | percentile_cont | median |
+----------+---------+-----------------+--------+
|        2 |       2 |               2 |      2 |
```

```
|       26 |       1 |                1 |      1 |
|       33 |       1 |                1 |      1 |
|       38 |       1 |                1 |      1 |
|       43 |       1 |                1 |      1 |
|       48 |       2 |                2 |      2 |
|       48 |       3 |                3 |      3 |
|       77 |       4 |                4 |      4 |
|       85 |       4 |                4 |      4 |
|       95 |       2 |                2 |      2 |
+----------+---------+------------------+--------+
```

다음 예제에서는 각 sellerid의 판매 수량 중간값을 찾습니다.

```
SELECT sellerid,
MEDIAN(qtysold)
FROM sales
GROUP BY sellerid
ORDER BY sellerid
LIMIT 10;

+----------+--------+
| sellerid | median |
+----------+--------+
|        1 |    1.5 |
|        2 |      2 |
|        3 |      2 |
|        4 |      2 |
|        5 |      1 |
|        6 |      1 |
|        7 |    1.5 |
|        8 |      1 |
|        9 |      4 |
|       12 |      2 |
+----------+--------+
```

첫 번째 sellerid에 대한 이전 쿼리 결과를 확인하려면 다음 예제를 사용합니다.

```
SELECT qtysold
FROM sales
WHERE sellerid=1;

+---------+
| qtysold |
```

```
+---------+
|       2 |
|       1 |
+---------+
```

MIN 함수

MIN 함수는 행 집합에서 최솟값을 반환합니다. DISTINCT 또는 ALL은 사용할 수 있지만 결과에 아무런 영향도 끼치지 않습니다.

구문

```
MIN ( [ DISTINCT | ALL ] expression )
```

인수

표현식

함수가 실행되는 대상 열 또는 표현식입니다. 표현식은 다음 데이터 유형 중 하나입니다.

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- REAL
- DOUBLE PRECISON
- CHAR
- VARCHAR
- 날짜
- TIMESTAMP
- TIMESTAMPTZ
- TIME
- TIMETZ
- VARBYTE
- SUPER

DISTINCT | ALL

인수가 DISTINCT일 때는 함수가 최솟값을 계산하기 전에 지정한 표현식에서 중복 값을 모두 제거합니다. 인수가 ALL일 때는 함수가 표현식의 모든 중복 값을 그대로 유지한 채 최솟값을 계산합니다. ALL이 기본값입니다.

데이터 타입

expression과 동일한 데이터 형식을 반환합니다. MIN 함수일 때 부울 등가는 BOOL_AND 함수이고, MAX 함수일 때 부울 등가는 BOOL_OR 함수입니다.

예시

모든 판매에서 지불된 최저 가격을 구합니다.

```
select min(pricepaid) from sales;

min
-------
20.00
(1 row)
```

모든 판매에서 티켓 1장당 지불된 최저 가격을 구합니다.

```
select min(pricepaid/qtysold)as min_ticket_price
from sales;

min_ticket_price
------------------
20.00000000
(1 row)
```

PERCENTILE_CONT 함수

PERCENTILE_CONT는 연속 분포 모델을 가정하는 역분포 함수입니다. 백분위 값과 정렬 명세를 가지며, 정렬 명세와 관련하여 지정된 백분위 값에 해당하는 보간 값을 반환합니다.

PERCENTILE_CONT는 순서가 지정된 값 사이의 선형 보간을 계산합니다. 이 함수는 집계 그룹에서 백분위 값(P)과 NULL을 제외한 행들의 번호(N)를 사용하여 정렬 명세에 따라 행의 순서를 지정한 후 행 번호를 계산합니다. 행 번호(RN)를 계산하는 공식은 RN = (1+ (P*(N-1))입니다. 이 집계 함수

의 최종 결과는 행 번호가 CRN = CEILING(RN)과 FRN = FLOOR(RN)인 행의 값 사이 선형 보간을 통해 계산됩니다.

최종 결과는 다음과 같습니다.

(CRN = FRN = RN)일 때 결과는 (value of expression from row at RN)입니다.

그렇지 않다면 다음 결과가 표시됩니다.

(CRN - RN) * (value of expression for row at FRN) + (RN - FRN) * (value of expression for row at CRN).

구문

```
PERCENTILE_CONT(percentile)
WITHIN GROUP(ORDER BY expr)
```

인수

Percentile

0에서 1 사이의 숫자 상수입니다. 계산에서 NULL 값은 무시됩니다.

expr

숫자 또는 날짜/시간 값을 지정하여 백분위를 정렬 및 계산합니다.

반환

반환 형식은 WITHIN GROUP 절에서 ORDER BY 표현식의 데이터 형식에 따라 결정됩니다. 다음 표는 ORDER BY 표현식의 데이터 형식에 따른 반환 형식을 나타낸 것입니다.

입력 유형	반환 타입
INT2, INT4, INT8, NUMERIC, DECIMAL	DECIMAL
FLOAT, DOUBLE	DOUBLE
DATE	DATE
TIMESTAMP	TIMESTAMP

입력 유형	반환 타입
TIMESTAMPTZ	TIMESTAMPTZ

사용 노트

ORDER BY 표현식이 최대 정밀도가 38자리로 정의된 DECIMAL 데이터 형식인 경우에는 PERCENTILE_CONT가 부정확한 결과 또는 오류를 반환합니다. PERCENTILE_CONT 함수의 반환 값이 38자리를 초과하면 정밀도가 손실될 수도 있기 때문에 알맞은 자리 수로 결과가 잘립니다. 보간 도중 중간 결과가 최대 정밀도를 초과하면 수치 오버플로우가 발생하고 함수는 오류를 반환합니다. 이러한 상황을 방지하려면 정밀도가 낮은 데이터 형식을 사용하거나, 혹은 ORDER BY 표현식을 낮은 정밀도로 변환합니다.

하나의 문에서 정렬 기반 집계 함수(LISTAGG, PERCENTILE_CONT, MEDIAN)를 여러 차례 호출하는 경우에는 모두 동일한 ORDER BY 값을 사용해야 합니다. 단, MEDIAN은 표현식 값에서 묵시적인 ORDER BY를 적용합니다.

예를 들어 다음과 같은 문은 오류를 반환합니다.

```
SELECT TOP 10 salesid, SUM(pricepaid),
PERCENTILE_CONT(0.6) WITHIN GROUP(ORDER BY salesid),
MEDIAN(pricepaid)
FROM sales
GROUP BY salesid, pricepaid;

An error occurred when executing the SQL command:
SELECT TOP 10 salesid, SUM(pricepaid),
PERCENTILE_CONT(0.6) WITHIN GROUP(ORDER BY salesid),
MEDIAN(pricepaid)
FROM sales
GROUP BY salesid, pricepaid;

ERROR: within group ORDER BY clauses for aggregate functions must be the same
```

다음 문은 성공적으로 실행됩니다.

```
SELECT TOP 10 salesid, SUM(pricepaid),
PERCENTILE_CONT(0.6) WITHIN GROUP(ORDER BY salesid),
MEDIAN(salesid)
FROM sales
```

```
GROUP BY salesid, pricepaid;
```

예시

다음 예제에서는 TICKIT 샘플 데이터베이스를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

다음은 PERCENTILE_CONT(0.5)이 MEDIAN과 동일한 결과를 산출하는 예입니다.

```
SELECT TOP 10 DISTINCT sellerid, qtysold,
PERCENTILE_CONT(0.5) WITHIN GROUP(ORDER BY qtysold),
MEDIAN(qtysold)
FROM sales
GROUP BY sellerid, qtysold;

+----------+---------+-----------------+--------+
| sellerid | qtysold | percentile_cont | median |
+----------+---------+-----------------+--------+
|        2 |       2 |               2 |      2 |
|       26 |       1 |               1 |      1 |
|       33 |       1 |               1 |      1 |
|       38 |       1 |               1 |      1 |
|       43 |       1 |               1 |      1 |
|       48 |       2 |               2 |      2 |
|       48 |       3 |               3 |      3 |
|       77 |       4 |               4 |      4 |
|       85 |       4 |               4 |      4 |
|       95 |       2 |               2 |      2 |
+----------+---------+-----------------+--------+
```

다음 예제에서는 SALES 테이블의 각 sellerid별로 판매된 수량에 대한 PERCENTILE_CONT(0.5) 및 PERCENTILE_CONT(0.75)를 찾습니다.

```
SELECT sellerid,
PERCENTILE_CONT(0.5) WITHIN GROUP(ORDER BY qtysold) as pct_50,
PERCENTILE_CONT(0.75) WITHIN GROUP(ORDER BY qtysold) as pct_75
FROM sales
GROUP BY sellerid
ORDER BY sellerid
LIMIT 10;

+----------+--------+---------+
```

```
| sellerid | pct_50 | pct_75 |
+----------+--------+---------+
|        1 |    1.5 |    1.75 |
|        2 |      2 |    2.25 |
|        3 |      2 |       3 |
|        4 |      2 |       2 |
|        5 |      1 |     1.5 |
|        6 |      1 |       1 |
|        7 |    1.5 |    1.75 |
|        8 |      1 |       1 |
|        9 |      4 |       4 |
|       12 |      2 |    3.25 |
+----------+--------+---------+
```

첫 번째 sellerid에 대한 이전 쿼리 결과를 확인하려면 다음 예제를 사용합니다.

```
SELECT qtysold
FROM sales
WHERE sellerid=1;

+---------+
| qtysold |
+---------+
|       2 |
|       1 |
+---------+
```

STDDEV_SAMP 및 STDDEV_POP 함수

STDDEV_SAMP 및 STDDEV_POP 함수는 숫자 값(정수, 소수 또는 부동 소수점) 집합의 표본 표준 편차와 모 표준 편차를 반환합니다. STDDEV_SAMP 함수의 결과는 동일한 값 집합의 표본 분산 제곱근과 동일합니다.

STDDEV_SAMP와 STDDEV는 동일한 함수이기 때문에 동의어나 마찬가지입니다.

구문

```
STDDEV_SAMP | STDDEV ( [ DISTINCT | ALL ] expression)
STDDEV_POP ( [ DISTINCT | ALL ] expression)
```

표현식의 데이터 형식은 정수, 소수 또는 부동 소수점이 되어야 합니다. 표현식의 데이터 형식과 상관없이 이 함수의 반환 형식은 배정밀도 숫자입니다.

> **Note**
>
> 표준 편차는 부동 소수점 연산을 통해 계산하지만 약간 부정확할 수 있습니다.

사용 노트

단일 값으로 구성된 표현식에 대해 표본 표준 편차(STDDEV 또는 STDDEV_SAMP)를 계산할 경우 함수 결과는 0이 아닌 NULL이 됩니다.

예시

다음 쿼리는 VENUE 테이블에서 VENUESEATS 열의 값 평균과 그 뒤를 이어 동일한 값 집합의 표본 표준 편차 및 모 표준 편차를 반환합니다. VENUESEATS는 INTEGER 열입니다. 결과의 크기는 2자리로 줄어듭니다.

```
select avg(venueseats),
cast(stddev_samp(venueseats) as dec(14,2)) stddevsamp,
cast(stddev_pop(venueseats) as dec(14,2)) stddevpop
from venue;

avg  | stddevsamp | stddevpop
-------+------------+-----------
17503 |   27847.76 |  27773.20
(1 row)
```

다음 쿼리는 SALES 테이블에서 COMMISSION 열의 표본 표준 편차를 반환합니다. COMMISSION은 DECIMAL 열입니다. 결과의 크기는 10자리로 줄어듭니다.

```
select cast(stddev(commission) as dec(18,10))
from sales;

stddev
----------------
130.3912659086
(1 row)
```

다음 쿼리는 COMMISSION 열의 표본 표준 편차를 정수로 변환합니다.

```
select cast(stddev(commission) as integer)
from sales;
```

```
stddev
--------
130
(1 row)
```

다음 쿼리는 COMMISSION 열의 표본 표준 편차와 표본 분산 제곱근을 모두 반환합니다. 이 두 가지의 계산 결과는 동일합니다.

```
select
cast(stddev_samp(commission) as dec(18,10)) stddevsamp,
cast(sqrt(var_samp(commission)) as dec(18,10)) sqrtvarsamp
from sales;

stddevsamp   |  sqrtvarsamp
---------------+----------------
130.3912659086 | 130.3912659086
(1 row)
```

SUM 함수

SUM 함수는 입력 열 또는 표현식 값의 합을 반환합니다. SUM 함수는 숫자 값을 사용하고 NULL 값을 무시합니다.

구문

```
SUM ( [ DISTINCT | ALL ] expression )
```

인수

표현식

함수가 실행되는 대상 열 또는 표현식입니다. 표현식은 다음 데이터 유형 중 하나입니다.

- SMALLINT
- INTEGER
- BIGINT
- NUMERIC
- DECIMAL
- REAL

- DOUBLE PRECISON
- SUPER

DISTINCT | ALL

인수가 DISTINCT일 때는 함수가 합을 계산하기 전에 지정한 표현식에서 중복 값을 모두 제거합니다. 인수가 ALL일 때는 함수가 표현식의 모든 중복 값을 그대로 유지한 채 합을 계산합니다. ALL이 기본값입니다.

데이터 타입

SUM 함수에서 지원되는 인수 형식은 SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL, REAL, DOUBLE PRECISION, SUPER입니다.

SUM 함수에서 지원되는 반환 형식은 다음과 같습니다.

- BIGINT, SMALLINT 및 INTEGER 인수일 때 BIGINT
- NUMERIC 인수일 때 NUMERIC
- 부동 소수점 인수일 때 DOUBLE PRECISION
- 다른 인수 형식의 표현과 동일한 데이터 형식을 반환합니다.

NUMERIC 또는 DECIMAL 인수가 포함된 SUM 함수 결과의 기본 정밀도는 38입니다. 함수 결과의 비율은 인수 비율과 동일합니다. 예를 들어 DEC(5,2) 열의 SUM은 DEC(38,2) 데이터 형식을 반환합니다.

예시

SALES 테이블에서 지불된 모든 수수료의 합을 구합니다.

```
select sum(commission) from sales;

sum
-------------
16614814.65
(1 row)
```

Florida 주에 위치한 모든 장소의 좌석 수를 구합니다.

```
select sum(venueseats) from venue
where venuestate = 'FL';
```

```
sum
--------
250411
(1 row)
```

5월에 판매된 좌석 수를 구합니다.

```
select sum(qtysold) from sales, date
where sales.dateid = date.dateid and date.month = 'MAY';

sum
-------
32291
(1 row)
```

VAR_SAMP 및 VAR_POP 함수

VAR_SAMP 및 VAR_POP 함수는 숫자 값(정수, 소수 또는 부동 소수점) 집합의 표본 분산과 모 분산을 반환합니다. VAR_SAMP 함수의 결과는 동일한 값 집합의 표본 표준 편차를 제곱한 것과 동일합니다.

VAR_SAMP 및 VARIANCE는 동일한 함수이기 때문에 동의어나 마찬가지입니다.

구문

```
VAR_SAMP | VARIANCE ( [ DISTINCT | ALL ] expression)
VAR_POP ( [ DISTINCT | ALL ] expression)
```

표현식의 데이터 형식은 정수, 소수 또는 부동 소수점이 되어야 합니다. 표현식의 데이터 형식과 상관없이 이 함수의 반환 형식은 배정밀도 숫자입니다.

> **Note**
>
> 두 함수의 결과는 데이터 웨어하우스 클러스터에서 각각 클러스터 구성에 따라 다를 수 있습니다.

사용 노트

단일 값으로 구성된 표현식에 대해 표본 분산(VARIANCE 또는 VAR_SAMP)을 계산할 경우 함수 결과는 0이 아닌 NULL이 됩니다.

예시

다음 쿼리는 LISTING 테이블에서 NUMTICKETS 열의 표본 및 모 분산을 반올림하여 반환합니다.

```
select avg(numtickets),
round(var_samp(numtickets)) varsamp,
round(var_pop(numtickets)) varpop
from listing;

avg | varsamp | varpop
-----+---------+--------
10 |      54 |     54
(1 row)
```

다음 쿼리는 동일한 계산을 실행하지만 결과를 소수 값으로 변환합니다.

```
select avg(numtickets),
cast(var_samp(numtickets) as dec(10,4)) varsamp,
cast(var_pop(numtickets) as dec(10,4)) varpop
from listing;

avg | varsamp | varpop
-----+---------+---------
10 | 53.6291 | 53.6288
(1 row)
```

배열 함수

다음에서 Amazon Redshift가 배열 액세스 및 조작을 지원하는 SQL용 배열 함수에 대한 설명을 찾을 수 있습니다.

주제

- 배열 함수
- array_concat 함수
- array_flatten 함수
- get_array_length 함수
- split_to_array 함수
- 부분 배열 함수

배열 함수

SUPER 데이터 형식의 배열을 만듭니다.

구문

```
ARRAY( [ expr1 ] [ , expr2 [ , ... ] ] )
```

인수

expr1, expr2

Amazon Redshift는 날짜 및 시간 형식을 SUPER 데이터 형식으로 캐스팅하지 않으므로 날짜 및 시간 형식을 제외한 모든 Amazon Redshift 데이터 형식의 표현식입니다. 인수는 동일한 데이터 형식일 필요는 없습니다.

반환 타입

배열 함수는 SUPER 데이터 형식을 반환합니다.

예제

다음 예에서는 숫자 값의 배열과 다양한 데이터 형식의 배열을 보여줍니다.

```
--an array of numeric values
select array(1,50,null,100);
      array
------------------
 [1,50,null,100]
(1 row)

--an array of different data types
select array(1,'abc',true,3.14);
        array
-----------------------
 [1,"abc",true,3.14]
(1 row)
```

array_concat 함수

array_concat 함수는 두 배열을 연결하여 첫 번째 배열의 모든 요소와 두 번째 배열의 모든 요소를 차례로 포함하는 배열을 생성합니다. 두 인수는 유효한 배열이어야 합니다.

구문

```
array_concat( super_expr1,  super_expr2 )
```

인수

super_expr1

연결할 두 배열 중 첫 번째를 지정하는 값입니다.

super_expr2

연결할 두 배열 중 두 번째를 지정하는 값입니다.

반환 타입

array_concat 함수는 SUPER 데이터 값을 반환합니다.

예제

다음 예에서는 형식이 같은 두 배열의 연결과 형식이 다른 두 배열의 연결을 보여줍니다.

```
-- concatenating two arrays
SELECT ARRAY_CONCAT(ARRAY(10001,10002),ARRAY(10003,10004));
              array_concat
-----------------------------------
 [10001,10002,10003,10004]
(1 row)

-- concatenating two arrays of different types
SELECT ARRAY_CONCAT(ARRAY(10001,10002),ARRAY('ab','cd'));
          array_concat
------------------------------
 [10001,10002,"ab","cd"]
(1 row)
```

array_flatten 함수

여러 배열을 SUPER 형식의 단일 배열로 병합합니다.

구문

```
array_flatten( super_expr1,super_expr2,.. )
```

인수

super_expr1,super_expr2

배열 형식의 유효한 SUPER 표현식입니다.

반환 타입

array_flatten 함수는 SUPER 데이터 값을 반환합니다.

예제

다음 예에서는 array_flatten 함수를 보여줍니다.

```
SELECT ARRAY_FLATTEN(ARRAY(ARRAY(1,2,3,4),ARRAY(5,6,7,8),ARRAY(9,10)));
     array_flatten
------------------------
 [1,2,3,4,5,6,7,8,9,10]
(1 row)
```

get_array_length 함수

지정된 배열의 길이를 반환합니다. GET_ARRAY_LENGTH 함수는 객체 또는 배열 경로가 지정된 SUPER 배열의 길이를 반환합니다.

구문

```
get_array_length( super_expr )
```

인수

super_expr

배열 형식의 유효한 SUPER 표현식입니다.

반환 타입

get_array_length 함수는 BIGINT를 반환합니다.

예제

다음 예에서는 get_array_length 함수를 보여줍니다.

```
SELECT GET_ARRAY_LENGTH(ARRAY(1,2,3,4,5,6,7,8,9,10));
 get_array_length
----------------------
            10
(1 row)
```

split_to_array 함수

구분 기호를 옵션 파라미터로 사용합니다. 구분 기호가 없는 경우 기본값은 쉼표입니다.

구문

```
split_to_array( string,delimiter )
```

인수

string

분할할 입력 문자열입니다.

delimiter

입력 문자열이 분할될 옵션 값입니다. 기본값은 쉼표입니다.

반환 타입

split_to_array 함수는 SUPER 데이터 값을 반환합니다.

예제

다음 예에서는 split_to_array 함수를 보여줍니다.

```
SELECT SPLIT_TO_ARRAY('12|345|6789', '|');
     split_to_array
------------------------
 ["12","345","6789"]
(1 row)
```

부분 배열 함수

입력 배열의 하위 집합을 반환하도록 배열을 조작합니다.

구문

```
SUBARRAY( super_expr, start_position, length )
```

인수

super_expr

배열 형식의 유효한 SUPER 표현식입니다.

start_position

인덱스 위치 0에서 시작하여 추출을 시작할 배열 내의 위치입니다. 음수 위치는 배열 끝에서 역방향으로 계산됩니다.

length

추출할 요소 수(하위 문자열의 길이)입니다.

반환 타입

부분 배열 함수는 SUPER 데이터 값을 반환합니다.

예시

다음은 하위 배열 함수 예제입니다.

```
 SELECT SUBARRAY(ARRAY('a', 'b', 'c', 'd', 'e', 'f'), 2, 3);
   subarray
---------------
 ["c","d","e"]
(1 row)
```

비트 단위 집계 함수

비트 단위 집계 함수는 정수 값으로 변환하거나 반올림할 수 있는 정수 열과 열의 집계를 수행하기 위해 비트 연산을 계산합니다.

주제

- 비트 단위 집계 시 NULL 사용
- 비트 단위 집계를 위한 DISTINCT 지원

비트 단위 집계 시 NULL 사용

NULL 값을 허용하는 열에 비트 단위 함수를 적용하면 함수 결과를 계산하기 전에 모든 NULL 값이 제거됩니다. 집계가 가능한 행이 없을 경우에는 비트 단위 함수가 NULL을 반환합니다. 정규 집계 함수에도 동일한 방식이 적용됩니다. 다음은 한 예입니다.

```
select sum(venueseats), bit_and(venueseats) from venue
where venueseats is null;

sum  | bit_and
------+---------
null |    null
(1 row)
```

비트 단위 집계를 위한 DISTINCT 지원

다른 집계 함수와 마찬가지로 비트 단위 함수도 DISTINCT 키워드를 지원합니다.

하지만 이 함수에 DISTINCT를 사용하더라도 결과에는 아무런 영향도 미치지 않습니다. 값의 첫 번째 인스턴스는 비트 단위 AND 또는 OR 연산을 충족하기에 충분합니다. 평가 중인 표현식에 중복 값이 있는 경우에는 차이가 없습니다.

오히려 DISTINCT 처리 시 쿼리 실행에 따른 일부 오버헤드가 발생할 가능성이 많기 때문에 비트 단위 함수에서는 DISTINCT를 사용하지 않는 것이 좋습니다.

비트 단위 함수의 개요 예

다음에서 비트 단위 함수로 작업하는 방법을 보여주는 몇 가지 개요 예를 찾을 수 있습니다. 각 함수 설명과 함께 특정 코드 예제를 찾을 수도 있습니다.

비트 단위 함수의 예는 TICKIT 샘플 데이터베이스를 기반으로 합니다. TICKIT 샘플 데이터베이스의 USERS 테이블에는 스포츠, 영화, 오페라 등 다양한 이벤트 유형에 대한 각 사용자의 호불호를 나타내는 몇 가지 부울 열이 포함되어 있습니다. 예를 들면 다음과 같습니다.

```
select userid, username, lastname, city, state,
likesports, liketheatre
from users limit 10;

userid | username | lastname  |     city     | state | likesports | liketheatre
-------+----------+-----------+--------------+-------+------------+-------------
1 | JSG99FHE | Taylor    | Kent         | WA    | t          | t
9 | MSD36KVR | Watkins   | Port Orford  | MD    | t          | f
```

USERS 테이블의 새 버전이 다른 방식으로 빌드되었다고 가정합니다. 이 새 버전에서는 각 사용자가 좋아하거나 좋아하지 않는 8가지 유형의 이벤트를 이진 형식으로 정의하는 단일 정수 열입니다. 이 설계에서 각 비트 위치는 이벤트 유형을 나타냅니다. 8가지 유형을 모두 좋아하는 사용자는 다음 표의 첫 번째 행과 같이 8개 비트 모두 1로 설정합니다. 반대로 이벤트를 모두 좋아하지 않는 사용자는 두 번째 행과 같이 8개 비트가 모두 0으로 설정됩니다. 그리고 스포츠와 재즈만 좋아하는 사용자는 세 번째 행과 같이 표현됩니다.

USER	스포츠	영화	재즈	오페라	록	라스베이거스	브로드웨이	클래식
사용자 1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0

데이터베이스 테이블에서는 이러한 이진 값들 모두 다음과 같이 LIKES 열 하나에 정수로 저장됩니다.

User	이진 값	저장 값(정수)
사용자 1	11111111	255
255	00000000	0
0	10100000	160

BIT_AND 함수

BIT_AND 함수는 단일 정수 열 또는 표현식의 모든 값에 대해 비트 단위 AND 연산을 실행합니다. 이 함수는 표현식에서 각 정수 값에 해당하는 각 이진 값의 개별 비트를 집계합니다.

모든 값에서 1로 설정된 비트가 없는 경우에는 BIT_AND 함수가 결과로 0을 반환합니다. 모든 값에서 1로 설정된 비트가 1개 이상인 경우에는 함수가 정수 값을 반환합니다. 이 정수는 1로 설정된 비트의 이진 값에 해당하는 수입니다.

예를 들어 한 테이블의 열에 정수 값으로 3, 7, 10 및 22가 포함되어 있다고 가정할 때 이 정수의 이진 값은 다음과 같이 표현됩니다.

Integer	이진 값
3	11
7	111
10	1010
22	10110

이 데이터 세트에서 BIT_AND 연산을 실행하면 두 번째부터 마지막 자리까지만 모든 비트가 1로 설정되는 것을 알 수 있습니다. 결과는 정수 값 2를 나타내는 00000010의 이진 값입니다. 따라서 BIT_AND 함수는 2를 반환합니다.

구문

```
BIT_AND ( [DISTINCT | ALL] expression )
```

인수

표현식

함수가 실행되는 대상 열 또는 표현식입니다. 이 표현식의 데이터 형식은 INT, INT2 또는 INT8이 되어야 합니다. 함수도 동일한 INT, INT2 또는 INT8 데이터 형식을 반환합니다.

DISTINCT | ALL

인수가 DISTINCT일 때는 함수가 결과를 계산하기 전에 지정한 표현식의 중복 값을 모두 제거합니다. 인수가 ALL일 때는 함수가 모든 중복 값을 그대로 유지합니다. ALL이 기본값입니다. 자세한 내용은 비트 단위 집계를 위한 DISTINCT 지원 단원을 참조하십시오.

예시

유의적인 비즈니스 정보가 정수 열에 저장된다는 점을 고려했을 때 비트 단위 함수를 사용하여 해당 정보를 추출하거나 집계할 수 있습니다. 다음은 BIT_AND 함수를 USERLIKES라는 테이블의 LIKES 열에 적용한 후 CITY 열을 기준으로 그 결과를 그룹화한 쿼리입니다.

```
select city, bit_and(likes) from userlikes group by city
order by city;
city          | bit_and
--------------+---------
Los Angeles   |       0
Sacramento    |       0
San Francisco |       0
San Jose      |      64
Santa Barbara |     192
(5 rows)
```

이 결과를 다음과 같이 해석할 수 있습니다.

- Santa Barbara 시의 정수 값 192을 이진 값으로 변환하면 11000000이 됩니다. 다시 말해서 이 도시의 사용자들은 모두 스포츠와 영화는 좋아하지만 다른 유형의 이벤트는 좋아하지 않습니다.
- 정수 64는 01000000으로 변환됩니다. 따라서 San Jose의 사용자가 모두 좋아하는 유일한 이벤트 유형은 theatre입니다.
- 나머지 3개 도시는 정수 값이 0이므로 이 도시의 사용자들은 모두 좋아하는 이벤트 유형이 없는 것을 의미합니다.

BIT_OR 함수

BIT_OR 함수는 단일 정수 열 또는 표현식의 모든 값에 대해 비트 단위 OR 연산을 실행합니다. 이 함수는 표현식에서 각 정수 값에 해당하는 각 이진 값의 개별 비트를 집계합니다.

예를 들어 한 테이블의 열에 정수 값으로 3, 7, 10 및 22가 포함되어 있다고 가정합니다. 이 정수의 이진 값은 다음과 같이 표현됩니다.

Integer	이진 값
3	11
7	111
10	1010
22	10110

BIT_OR 함수를 정수 값 집합에 적용하면 연산은 각 위치에서 1이 있는 값을 찾습니다. 이때 1은 최소 1개 이상의 값에서 마지막 5자리에 존재하기 때문에 이진 값으로 00011111이 산출되고, 이에 따라 함수는 31(16 + 8 + 4 + 2 + 1)을 반환합니다.

구문

```
BIT_OR ( [DISTINCT | ALL] expression )
```

인수

표현식

함수가 실행되는 대상 열 또는 표현식입니다. 이 표현식의 데이터 형식은 INT, INT2 또는 INT8이 되어야 합니다. 함수도 동일한 INT, INT2 또는 INT8 데이터 형식을 반환합니다.

DISTINCT | ALL

인수가 DISTINCT일 때는 함수가 결과를 계산하기 전에 지정한 표현식의 중복 값을 모두 제거합니다. 인수가 ALL일 때는 함수가 모든 중복 값을 그대로 유지합니다. ALL이 기본값입니다. 자세한 내용은 비트 단위 집계를 위한 DISTINCT 지원 단원을 참조하십시오.

예제

다음은 BIT_OR 함수를 USERLIKES라는 테이블의 LIKES 열에 적용한 후 CITY 열을 기준으로 그 결과를 그룹화한 쿼리입니다.

```
select city, bit_or(likes) from userlikes group by city
order by city;
city          | bit_or
```

```
-------------+--------
Los Angeles   |    127
Sacramento    |    255
San Francisco |    255
San Jose      |    255
Santa Barbara |    255
(5 rows)
```

위의 4개 도시에서는 모든 이벤트 유형을 좋아하는 사용자가 1명 이상입니다(255=11111111). 하지만 Los Angeles에서는 스포츠를 제외한 모든 이벤트 유형을 좋아하는 사용자가 1명 이상입니다(127=01111111).

BOOL_AND 함수

BOOL_AND 함수는 단일 부울 또는 정수 열이나 표현식에서 실행됩니다. 이 함수는 BIT_AND 및 BIT_OR 함수와 비슷한 로직을 적용합니다. 이 함수의 반환 형식은 부울 값(`true` 또는 `false`)입니다.

집합의 모든 값이 true이면 BOOL_AND 함수가 `true`(`t`)를 반환합니다. 하나라도 값이 false이면 함수가 `false`(`f`)를 반환합니다.

구문

```
BOOL_AND ( [DISTINCT | ALL] expression )
```

인수

표현식

함수가 실행되는 대상 열 또는 표현식입니다. 이 표현식의 데이터 형식은 부울 또는 정수가 되어야 합니다. 함수의 반환 형식은 부울입니다.

DISTINCT | ALL

인수가 DISTINCT일 때는 함수가 결과를 계산하기 전에 지정한 표현식의 중복 값을 모두 제거합니다. 인수가 ALL일 때는 함수가 모든 중복 값을 그대로 유지합니다. ALL이 기본값입니다. 자세한 내용은 비트 단위 집계를 위한 DISTINCT 지원 단원을 참조하십시오.

예시

부울 함수는 부울 표현식이나 정수 표현식에 대해 사용할 수 있습니다. 예를 들어 다음 쿼리는 TICKIT 데이터베이스에서 부울 열이 일부 포함되어 있는 표준 USERS 테이블을 통해 결과를 반환합니다.

BOOL_AND 함수는 5개 행 모두에서 false를 반환합니다. 해당 주마다 모든 사용자가 스포츠를 좋아하는 것은 아닙니다.

```
select state, bool_and(likesports) from users
group by state order by state limit 5;

state | bool_and
------+---------
AB    | f
AK    | f
AL    | f
AZ    | f
BC    | f
(5 rows)
```

BOOL_OR 함수

BOOL_OR 함수는 단일 부울 또는 정수 열이나 표현식에서 실행됩니다. 이 함수는 BIT_AND 및 BIT_OR 함수와 비슷한 로직을 적용합니다. 이 함수의 반환 형식은 부울 값(true, false 또는 NULL)입니다.

집합에서 하나 이상의 값이 true이면 BOOL_OR 함수가 true(t)를 반환합니다. 집합의 모든 값이 false이면 함수가 false(f)를 반환합니다. 값을 알 수 없는 경우 NULL을 반환할 수 있습니다.

구문

```
BOOL_OR ( [DISTINCT | ALL] expression )
```

인수

표현식

함수가 실행되는 대상 열 또는 표현식입니다. 이 표현식의 데이터 형식은 부울 또는 정수가 되어야 합니다. 함수의 반환 형식은 부울입니다.

DISTINCT | ALL

인수가 DISTINCT일 때는 함수가 결과를 계산하기 전에 지정한 표현식의 중복 값을 모두 제거합니다. 인수가 ALL일 때는 함수가 모든 중복 값을 그대로 유지합니다. ALL이 기본값입니다. 비트 단위 집계를 위한 DISTINCT 지원 섹션을 참조하세요.

예시

부울 함수는 부울 표현식이나 정수 표현식에 사용할 수 있습니다. 예를 들어 다음 쿼리는 TICKIT 데이터베이스에서 부울 열이 일부 포함되어 있는 표준 USERS 테이블을 통해 결과를 반환합니다.

BOOL_OR 함수는 5개 행 모두에서 `true`를 반환합니다. 즉, 해당 주마다 1명 이상의 사용자가 스포츠를 좋아합니다.

```
select state, bool_or(likesports) from users
group by state order by state limit 5;

state | bool_or
------+--------
AB    | t
AK    | t
AL    | t
AZ    | t
BC    | t
(5 rows)
```

다음 예는 NULL을 반환합니다.

```
SELECT BOOL_OR(NULL = '123')
               bool_or
------
NULL
```

조건식

주제

- CASE 조건식
- DECODE 함수
- GREATEST 및 LEAST 함수
- NVL 및 COALESCE 함수
- NVL2 함수
- NULLIF 함수

Amazon Redshift는 SQL 표준의 확장인 조건 표현식을 몇 가지 지원합니다.

CASE 조건식

CASE 표현식은 다른 언어에서 발견되는 if/then/else 문과 비슷한 조건 표현식입니다. CASE는 다수의 조건이 있을 때 결과를 지정하는 데 사용됩니다. SELECT 명령과 같이 SQL 표현식이 유효한 경우 CASE를 사용합니다.

CASE 표현식은 단순(simple)과 검색(searched), 두 가지 유형이 있습니다.

- 단순 CASE 표현식에서는 표현식과 값을 비교합니다. 이때 일치하는 부분이 발견되면 THEN 절에서 지정된 작업이 적용됩니다. 일치하는 부분이 발견되지 않으면 ELSE 절에서 지정된 작업이 적용됩니다.
- 검색 CASE 표현식에서는 각 CASE가 부울 표현식에 따라 평가되고, CASE 문이 처음 일치하는 CASE를 반환합니다. WHEN 절 사이에서 일치하는 부분이 발견되지 않으면 ELSE 절의 작업이 반환됩니다.

구문

다음은 조건을 일치시키는 데 사용되는 단순 CASE 문입니다.

```
CASE expression
  WHEN value THEN result
  [WHEN...]
  [ELSE result]
END
```

다음은 각 조건을 평가하는 데 사용되는 검색 CASE 문입니다.

```
CASE
  WHEN condition THEN result
  [WHEN ...]
  [ELSE result]
END
```

인수

expression

열 이름 또는 유효한 표현식입니다.

USD 상당

숫자 상수나 문자열 같이 표현식과 함께 비교하는 값입니다.

result

표현식 또는 부울 조건을 평가할 때 반환되는 대상 값 또는 표현식입니다. 모든 결과 표현식의 데이터 형식은 단일 출력 형식으로 변환할 수 있어야 합니다.

condition

true 또는 false로 평가되는 부울 표현식 condition이 true이면 CASE 표현식의 값은 조건 다음에 오는 결과이며 나머지 CASE 표현식은 처리되지 않습니다. condition이 false이면 이후의 모든 WHEN 절이 평가됩니다. WHEN condition 결과가 true가 아닌 경우 CASE 표현식의 값은 ELSE 절의 결과입니다. ELSE 절을 생략한 상태에서 condition이 true가 아닌 경우 NULL 값이 결과로 반환됩니다.

예시

다음 예시에서는 샘플 TICKIT 데이터의 VENUE 테이블과 SALES 테이블을 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

VENUE 테이블에 대한 쿼리에서 단순 CASE 표현식을 사용하여 `New York City`를 `Big Apple`로 변경합니다. 그 밖의 도시 이름은 모두 `other`로 변경합니다.

```
select venuecity,
  case venuecity
    when 'New York City'
    then 'Big Apple' else 'other'
  end
from venue
order by venueid desc;

venuecity        |   case
-----------------+-----------
Los Angeles      | other
New York City    | Big Apple
San Francisco    | other
Baltimore        | other
...
```

검색 CASE 표현식을 사용하여 개별 티켓 판매에 대한 PRICEPAID 값을 기준으로 그룹 번호를 할당합니다.

```
select pricepaid,
  case when pricepaid <10000 then 'group 1'
    when pricepaid >10000 then 'group 2'
    else 'group 3'
  end
from sales
order by 1 desc;

pricepaid |  case
----------+---------
12624     | group 2
10000     | group 3
10000     | group 3
9996      | group 1
9988      | group 1
...
```

DECODE 함수

DECODE 표현식은 등식 조건의 결과에 따라 특정 값을 다른 특정 값이나 기본값으로 변경합니다. 이 연산은 단순 CASE 표현식이나 IF-THEN-ELSE 문의 연산과 동일합니다.

구문

```
DECODE ( expression, search, result [, search, result ]... [ ,default ] )
```

이러한 유형의 표현식은 테이블에 저장된 축약어나 코드를 보고서에 필요한 유의적인 비즈니스 값으로 변경하는 데 유용합니다.

파라미터

expression

테이블 열과 같이 비교하고자 하는 값의 소스입니다.

search

숫자 값이나 문자열 같이 소스 표현식과 비교할 대상 값입니다. 검색 표현식은 단일 고정 값으로 평가되어야 합니다. `age between 20 and 29` 같이 범위 값으로 평가되는 표현식은 지정할 수 없습니다. 이런 경우에는 변경할 값마다 검색/결과 페어를 별도로 지정해야 합니다.

모든 검색 표현식 인스턴스의 데이터 형식은 동일하거나 호환 가능해야 합니다. expression과 search 파라미터 역시 호환 가능해야 합니다.

result

표현식이 검색 값과 일치할 때 쿼리가 반환하는 변경 값입니다. DECODE 표현식에는 검색/결과 페어가 하나 이상 포함되어야 합니다.

모든 결과 표현식 인스턴스의 데이터 형식은 동일하거나 호환 가능해야 합니다. result와 default 파라미터 역시 호환 가능해야 합니다.

기본값

검색 조건이 맞지 않는 경우 사용하는 기본값(옵션)입니다. 기본값을 지정하지 않으면 DECODE 표현식은 NULL을 반환합니다.

사용 노트

expression 값과 search 값이 모두 NULL이면 DECODE 결과는 해당하는 result 값이 됩니다. 이러한 함수 사용에 대한 설명은 예 섹션을 참조하세요.

이러한 방식으로 사용할 때는 DECODE가 NVL2 함수과 비슷하지만 몇 가지 차이점도 있습니다. 차이점에 대한 자세한 내용은 NVL2 사용 시 주의 사항을 참조하십시오.

예시

다음은 datetable의 caldate 열에 `2008-06-01` 값이 존재할 때 이 값을 `June 1st, 2008`로 변경하는 예입니다. 그 밖에 모든 caldate 값은 NULL로 바뀝니다.

```
select decode(caldate, '2008-06-01', 'June 1st, 2008')
from datetable where month='JUN' order by caldate;

case
----------------
June 1st, 2008

...
(30 rows)
```

다음은 DECODE 예를 사용하여 CATEGORY 테이블에서 축약된 CATNAME 열 5개를 전체 이름으로 변경하고 해당 열의 나머지 값은 Unknown으로 변경하는 예입니다.

```
select catid, decode(catname,
'NHL', 'National Hockey League',
'MLB', 'Major League Baseball',
'MLS', 'Major League Soccer',
```

```
'NFL', 'National Football League',
'NBA', 'National Basketball Association',
'Unknown')
from category
order by catid;

catid  | case
-------+----------------------------------
1      | Major League Baseball
2      | National Hockey League
3      | National Football League
4      | National Basketball Association
5      | Major League Soccer
6      | Unknown
7      | Unknown
8      | Unknown
9      | Unknown
10     | Unknown
11     | Unknown
(11 rows)
```

다음은 DECODE 표현식을 사용하여 Colorado와 Nevada에서 VENUESEATS 열에 NULL 값을 가진 장소를 찾아 NULL 값을 0으로 변경하는 예입니다. VENUESEATS 열에 NULL 값이 없으면 결과로 1을 반환합니다.

```
select venuename, venuestate, decode(venueseats,null,0,1)
from venue
where venuestate in('NV','CO')
order by 2,3,1;

venuename               | venuestate     | case
------------------------------+----------------+-----------
Coors Field                   | CO        |   1
Dick's Sporting Goods Park    | CO        |   1
Ellie Caulkins Opera House    | CO        |   1
INVESCO Field        | CO        |   1
Pepsi Center        | CO        |   1
Ballys Hotel        | NV        |   0
Bellagio Hotel                | NV        |   0
Caesars Palace                | NV        |   0
Harrahs Hotel                 | NV        |   0
Hilton Hotel                  | NV        |   0
...
```

```
(20 rows)
```

GREATEST 및 LEAST 함수

다수의 표현식 목록에서 가장 크거나 가장 작은 값을 반환합니다.

구문

```
GREATEST (value [, ...])
LEAST (value [, ...])
```

파라미터

expression_list

열 이름과 같이 쉼표로 구분된 표현식 목록입니다. 표현식은 모두 공통 데이터 형식으로 변환 가능해야 합니다. 목록에서 NULL 값은 무시됩니다. 표현식이 모두 NULL로 평가되면 결과로 NULL이 반환됩니다.

반환

제공된 표현식 목록에서 가장 큰 값(GREATEST) 또는 가장 작은 값(LEAST)을 반환합니다.

예제

다음은 `firstname` 또는 `lastname`에서 알파벳 순으로 가장 높은 값을 반환하는 예입니다.

```
select firstname, lastname, greatest(firstname,lastname) from users
where userid < 10
order by 3;

 firstname | lastname  | greatest
-----------+-----------+-----------
 Lars      | Ratliff   | Ratliff
 Reagan    | Hodge     | Reagan
 Colton    | Roy       | Roy
 Barry     | Roy       | Roy
 Tamekah   | Juarez    | Tamekah
 Rafael    | Taylor    | Taylor
 Victor    | Hernandez | Victor
 Vladimir  | Humphrey  | Vladimir
 Mufutau   | Watkins   | Watkins
```

```
(9 rows)
```

NVL 및 COALESCE 함수

일련의 표현식에서 NULL이 아닌 첫 번째 표현식의 값을 반환합니다. NULL이 아닌 값이 발견되면 목록의 나머지 표현식은 평가되지 않습니다.

NVL은 COALESCE와 동일합니다. 이 둘은 동의어입니다. 이 항목에서는 구문을 설명하고 두 가지에 대한 예제를 모두 제공합니다.

구문

```
NVL( expression, expression, ... )
```

COALESCE의 구문은 동일합니다.

```
COALESCE( expression, expression, ... )
```

모든 표현식이 NULL이면 결과는 NULL입니다.

이러한 함수는 기본 값이 없거나 NULL일 때 보조 값을 반환하려는 경우에 유용합니다. 예를 들어 쿼리는 사용 가능한 세 가지 전화번호(휴대폰, 집 또는 회사) 중 첫 번째 전화번호를 반환할 수 있습니다. 함수의 표현식 순서에 따라 평가 순서가 결정됩니다.

인수

expression

NULL 상태로 평가되는 표현식(열 이름 등)입니다.

반환 타입

Amazon Redshift는 입력 식을 기반으로 반환된 값의 데이터 유형을 결정합니다. 입력 표현식의 데이터 유형에 공통 유형이 없는 경우 오류가 반환됩니다.

예시

목록에 정수 표현식이 포함된 경우 함수는 정수를 반환합니다.

```
SELECT COALESCE(NULL, 12, NULL);

coalesce
```

```
--------------
12
```

이 예는 NVL을 사용한다는 점을 제외하면 이전 예와 동일하며 동일한 결과를 반환합니다.

```
SELECT NVL(NULL, 12, NULL);

coalesce
--------------
12
```

다음은 문자열 유형을 반환하는 예입니다.

```
SELECT COALESCE(NULL, 'Amazon Redshift', NULL);

coalesce
--------------
Amazon Redshift
```

다음 예에서는 표현식 목록에서 데이터 유형이 다양하기 때문에 오류가 발생합니다. 이 경우 목록에 문자열 유형과 숫자 유형이 모두 있습니다.

```
SELECT COALESCE(NULL, 'Amazon Redshift', 12);
ERROR: invalid input syntax for integer: "Amazon Redshift"
```

이 예에서는 START_DATE 및 END_DATE 열과 함께 테이블을 생성한 후 NULL 값이 포함된 행을 삽입합니다. 그런 다음 NVL 표현식을 두 열에 적용합니다.

```
create table datetable (start_date date, end_date date);
insert into datetable values ('2008-06-01','2008-12-31');
insert into datetable values (null,'2008-12-31');
insert into datetable values ('2008-12-31',null);
```

```
select nvl(start_date, end_date)
from datetable
order by 1;

coalesce
------------
2008-06-01
2008-12-31
```

```
2008-12-31
```

NVL 표현식의 기본 열 이름이 COALESCE입니다. 다음 쿼리에서도 동일한 결과가 반환됩니다.

```
select coalesce(start_date, end_date)
from datetable
order by 1;
```

다음 예제 쿼리에서는 샘플 호텔 예약 정보가 포함된 테이블을 만들고 여러 행을 삽입합니다. 일부 레코드에는 NULL 값이 포함되어 있습니다.

```
create table booking_info (booking_id int, booking_code character(8), check_in date,
 check_out date, funds_collected numeric(12,2));
```

다음 샘플 데이터를 삽입합니다. 일부 레코드에는 check_out 날짜 또는 funds_collected 금액이 없습니다.

```
insert into booking_info values (1, 'OCEAN_WV', '2023-02-01','2023-02-03',100.00);
insert into booking_info values (2, 'OCEAN_WV', '2023-04-22','2023-04-26',120.00);
insert into booking_info values (3, 'DSRT_SUN', '2023-03-13','2023-03-16',125.00);
insert into booking_info values (4, 'DSRT_SUN', '2023-06-01','2023-06-03',140.00);
insert into booking_info values (5, 'DSRT_SUN', '2023-07-10',null,null);
insert into booking_info values (6, 'OCEAN_WV', '2023-08-15',null,null);
```

다음 쿼리는 날짜 목록을 반환합니다. check_out 날짜가 없는 경우 check_in 날짜가 표시됩니다.

```
select coalesce(check_out, check_in)
from booking_info
order by booking_id;
```

결과는 다음과 같습니다. 마지막 두 레코드에는 check_in 날짜가 표시된다는 점에 유의하세요.

```
coalesce
------------
2023-02-03
2023-04-26
2023-03-16
2023-06-03
2023-07-10
2023-08-15
```

쿼리가 일부 함수 또는 열에서 NULL 값을 반환할 가능성이 있다면 NVL 표현식을 사용하여 NULL 값을 다른 값으로 변경할 수 있습니다. 예를 들어 SUM 같은 집계 함수는 평가할 행이 없으면 0이 아닌 NULL 값을 반환합니다. 이때는 NVL 표현식을 사용하여 NULL 값을 700.0으로 변경할 수 있습니다. funds_collected를 합산한 결과는 485가 아닌 1885입니다. NULL 값을 갖는 두 행이 700으로 대체되기 때문입니다.

```
select sum(nvl(funds_collected, 700.0)) as sumresult from booking_info;

sumresult
------
 1885
```

NVL2 함수

지정하는 표현식의 평가 결과가 NULL 또는 NOT NULL인지 여부에 따라 두 값 중 하나를 반환합니다.

구문

```
NVL2 ( expression, not_null_return_value, null_return_value )
```

인수

expression

NULL 상태로 평가되는 표현식(열 이름 등)입니다.

not_null_return_value

expression이 NOT NULL로 평가되면 반환되는 값입니다. not_null_return_value 값은 expression과 동일한 데이터 형식이거나, 혹은 묵시적으로 이 데이터 형식으로 변환 가능해야 합니다.

null_return_value

expression이 NULL로 평가되면 반환되는 값입니다. null_return_value 값은 expression과 동일한 데이터 형식이거나, 혹은 묵시적으로 이 데이터 형식으로 변환 가능해야 합니다.

반환 타입

NVL2 반환 형식은 다음과 같이 결정됩니다.

- not_null_return_value 또는 null_return_value이 NULL이면 not null 표현식의 데이터 형식이 반환됩니다.

not_null_return_value와 null_return_value 모두 NULL이 아닌 경우에는 다음과 같습니다.

- not_null_return_value와 null_return_value의 데이터 형식이 동일하면 해당 데이터 형식이 반환됩니다.
- not_null_return_value와 null_return_value의 숫자 데이터 형식이 다르면 가장 작으면서 호환도 가능한 숫자 데이터 형식이 반환됩니다.
- not_null_return_value와 null_return_value의 날짜/시간 데이터 형식이 다르면 타임스탬프 데이터 형식이 반환됩니다.
- not_null_return_value와 null_return_value의 문자 데이터 형식이 다르면 not_null_return_value의 데이터 형식이 반환됩니다.
- not_null_return_value와 null_return_value의 데이터 형식이 숫자와 비숫자로 섞여 있으면 not_null_return_value의 데이터 형식이 반환됩니다.

> **Important**
>
> not_null_return_value의 데이터 형식이 반환되는 마지막 두 경우에서는 null_return_value가 묵시적으로 해당 데이터 형식으로 변환됩니다. 이때 데이터 형식이 서로 호환되지 않으면 함수가 중단됩니다.

사용 노트

expression 및 search 파라미터가 모두 null인 경우 NVL2와 유사한 방식으로 DECODE 함수을 사용할 수 있습니다. 차이점은, DECODE일 때는 result 파라미터의 값과 데이터 형식이 모두 반환되는 데 있습니다. 이와는 반대로 NVL2일 때는 not_null_return_value 또는 null_return_value 파라미터의 값 중에서 함수에서 선택하는 값과 함께 not_null_return_value의 데이터 형식이 반환됩니다.

예를 들어 column1이 NULL이라고 가정하면 다음 두 쿼리에서 동일한 값이 반환됩니다. 하지만 DECODE의 반환 값 데이터 형식은 INTEGER인 반면 NVL2의 반환 값 데이터 형식은 VARCHAR입니다.

```
select decode(column1, null, 1234, '2345');
select nvl2(column1, '2345', 1234);
```

예제

다음은 일부 샘플 데이터를 수정한 후 두 필드를 평가하여 적합한 사용자 연락처를 제공하는 예입니다.

```
update users set email = null where firstname = 'Aphrodite' and lastname = 'Acevedo';

select (firstname + ' ' + lastname) as name,
nvl2(email, email, phone) AS contact_info
from users
where state = 'WA'
and lastname  like 'A%'
order by lastname, firstname;

name        contact_info
--------------------+--------------------------------------------
Aphrodite Acevedo (906) 632-4407
Caldwell Acevedo  Nunc.sollicitudin@Duisac.ca
Quinn Adams    vel@adipiscingligulaAenean.com
Kamal Aguilar  quis@vulputaterisusa.com
Samson Alexander  hendrerit.neque@indolorFusce.ca
Hall Alford    ac.mattis@vitaediamProin.edu
Lane Allen     et.netus@risusDonec.org
Xander Allison    ac.facilisis.facilisis@Infaucibus.com
Amaya Alvarado    dui.nec.tempus@eudui.edu
Vera Alvarez   at.arcu.Vestibulum@pellentesque.edu
Yetta Anthony  enim.sit@risus.org
Violet Arnold  ad.litora@at.com
August Ashley  consectetuer.euismod@Phasellus.com
Karyn Austin   ipsum.primis.in@Maurisblanditenim.org
Lucas Ayers    at@elitpretiumet.com
```

NULLIF 함수

구문

NULLIF 표현식은 두 인수를 비교하여 동일한 경우에는 NULL을 반환합니다. 동일하지 않으면 첫 번째 인수가 반환됩니다. 이 표현식은 NVL 또는 COALESCE 표현식의 정반대입니다.

```
NULLIF ( expression1, expression2 )
```

인수

expression1, expression2

비교 대상인 열 또는 표현식입니다. 반환 형식은 첫 번째 표현식의 형식과 동일합니다. NULLIF 결과의 기본 열 이름은 첫 번째 표현식의 열 이름과 일치합니다.

예시

다음 예에서는 인수가 같지 않기 때문에 쿼리가 문자열 first를 반환합니다.

```
SELECT NULLIF('first', 'second');

case
-------
first
```

다음 예에서는 리터럴 인수가 같기 때문에 쿼리가 NULL을 반환합니다.

```
SELECT NULLIF('first', 'first');

case
-------
NULL
```

다음 예에서는 정수 인수가 같지 않기 때문에 쿼리가 1을 반환합니다.

```
SELECT NULLIF(1, 2);

case
-------
1
```

다음 예에서는 정수 인수가 같기 때문에 쿼리가 NULL을 반환합니다.

```
SELECT NULLIF(1, 1);

case
-------
NULL
```

다음은 LISTID와 SALESID 값이 일치할 때 쿼리가 NULL을 반환하는 예입니다.

```
select nullif(listid,salesid), salesid
from sales where salesid<10 order by 1, 2 desc;

listid  | salesid
```

```
--------+---------
     4  |       2
     5  |       4
     5  |       3
     6  |       5
     10 |       9
     10 |       8
     10 |       7
     10 |       6
        |       1
(9 rows)
```

NULLIF를 사용하여 빈 문자열은 항상 NULL로 반환되도록 할 수 있습니다. 다음은 NULLIF 표현식이 NULL 값을, 혹은 문자가 1개 이상 포함된 문자열을 반환하는 예입니다.

```
insert into category
values(0,'','Special','Special');

select nullif(catgroup,'') from category
where catdesc='Special';

catgroup
----------
null
(1 row)
```

NULLIF는 후행 공백을 무시합니다. 빈 문자열이 아니더라도 공백이 포함되어 있으면 NULLIF가 NULL을 반환합니다.

```
create table nulliftest(c1 char(2), c2 char(2));

insert into nulliftest values ('a','a ');

insert into nulliftest values ('b','b');


select nullif(c1,c2) from nulliftest;
c1
------
null
null
(2 rows)
```

데이터 형식 지정 함수

주제

데이터 형식 지정 함수는 데이터 형식의 값을 다른 데이터 형식의 값으로 쉽게 변환할 수 있습니다. 이러한 함수에서는 모두 첫 번째 인수가 형식을 지정할 대상이 되는 값이고, 두 번째 인수에는 새로운 형식을 위한 템플릿이 포함되어 있습니다. Amazon Redshift는 여러 데이터 형식 서식 지정 기능을 지원합니다. Amazon Redshift는 여러 데이터 형식 지정 함수를 지원합니다.

CAST 함수

CAST 함수는 한 데이터 유형을 다른 호환 가능한 데이터 유형으로 변환합니다. 예를 들어 문자열을 날짜로 변환하거나 숫자 형식을 문자열로 변환할 수 있습니다. CAST는 런타임 변환을 수행합니다. 즉, 변환을 수행해도 원본 테이블의 값 데이터 형식은 변경되지 않습니다. 쿼리의 컨텍스트에서만 변경됩니다.

CAST 함수는 한 데이터 유형을 다른 데이터 유형으로 변환한다는 점에서 the section called “CONVERT”와 매우 유사하지만 호출되는 방식이 다릅니다.

CAST 또는 CONVERT 함수를 사용하여 다른 데이터 형식으로 명시적인 변환이 필요한 데이터 형식이 몇 가지 있습니다. 그 밖에 CAST 또는 CONVERT를 사용하지 않고 다른 명령의 일부로서 묵시적으로 변환할 수 있는 데이터 형식들도 있습니다. 형식 호환성 및 변환 섹션을 참조하세요.

구문

한 가지 데이터 형식에서 다른 데이터 형식으로 표현식을 변환할 때는 다음과 같이 두 가지 동등한 구문 형식을 사용할 수 있습니다.

```
CAST ( expression AS type )
expression :: type
```

인수

expression

열 이름이나 리터럴 같이 하나 이상의 값으로 평가되는 표현식입니다. null 값을 변환하면 마찬가지로 null이 반환됩니다. 또한 표현식에는 공백이나 빈 문자열이 포함되어서도 안 됩니다.

type

지원되는 데이터 타입 중 한 가지입니다.

반환 타입

CAST는 type 인수에서 지정하는 데이터 형식을 반환합니다.

> **Note**
>
> 다음과 같이 문제가 발생할 수 있는 변환을 실행하면 Amazon Redshift가 오류를 반환합니다. 첫째, 정밀도를 떨어뜨리는 DECIMAL 변환입니다.
>
> ```
> select 123.456::decimal(2,1);
> ```
>
> 둘째, 오버플로우를 야기하는 INTEGER 변환입니다.
>
> ```
> select 12345678::smallint;
> ```

예시

이 안내서에 나오는 예는 대부분 샘플 TICKIT 데이터베이스를 사용합니다. 샘플 데이터 설정에 관한 자세한 내용은 데이터 로드 단원을 참조하세요.

다음에 동등한 2개의 쿼리입니다. 두 쿼리 모두 소수 값을 정수로 변환합니다.

```
select cast(pricepaid as integer)
from sales where salesid=100;

pricepaid
```

```
-----------
162
(1 row)
```

```
select pricepaid::integer
from sales where salesid=100;

pricepaid
-----------
162
(1 row)
```

다음과 같은 결과가 나옵니다. 실행하는 데 샘플 데이터가 필요하지 않습니다.

```
select cast(162.00 as integer) as pricepaid;

pricepaid
-----------
162
(1 row)
```

이 예에서는 각 결과에서 시간이 제거되고 타임스탬프 열의 값이 날짜로 변환됩니다.

```
select cast(saletime as date), salesid
from sales order by salesid limit 10;

 saletime  | salesid
-----------+---------
2008-02-18 |       1
2008-06-06 |       2
2008-06-06 |       3
2008-06-09 |       4
2008-08-31 |       5
2008-07-16 |       6
2008-06-26 |       7
2008-07-10 |       8
2008-07-22 |       9
2008-08-06 |      10
(10 rows)
```

이전 예에서 설명한 대로 CAST를 사용하지 않은 경우 결과에 2008-02-18 02:36:48처럼 시간이 포함됩니다.

다음 쿼리는 가변 문자 데이터를 날짜로 캐스팅합니다. 실행하는 데 샘플 데이터가 필요하지 않습니다.

```
select cast('2008-02-18 02:36:48' as date) as mysaletime;

mysaletime
--------------------
2008-02-18
(1 row)
```

다음은 날짜 열의 값이 타임스탬프로 변환된 예입니다.

```
select cast(caldate as timestamp), dateid
from date order by dateid limit 10;

      caldate       | dateid
--------------------+--------
2008-01-01 00:00:00 |   1827
2008-01-02 00:00:00 |   1828
2008-01-03 00:00:00 |   1829
2008-01-04 00:00:00 |   1830
2008-01-05 00:00:00 |   1831
2008-01-06 00:00:00 |   1832
2008-01-07 00:00:00 |   1833
2008-01-08 00:00:00 |   1834
2008-01-09 00:00:00 |   1835
2008-01-10 00:00:00 |   1836
(10 rows)
```

이전 예와 같은 경우에는 TO_CHAR를 사용하여 출력 형식을 추가로 제어할 수 있습니다.

다음은 정수가 문자열로 변환된 예입니다.

```
select cast(2008 as char(4));

bpchar
--------
2008
```

다음은 DECIMAL(6,3) 값이 DECIMAL(4,1) 값으로 변환된 예입니다.

```
select cast(109.652 as decimal(4,1));
```

```
numeric
---------
109.7
```

이 예에서는 좀 더 복잡한 표현식을 보여줍니다. 다음은 SALES 테이블의 PRICEPAID 열(DECIMAL(8,2) 열)을 DECIMAL(38,2) 열로 변환하고, 값을 10000000000000000000과 곱합니다.

```
select salesid, pricepaid::decimal(38,2)*10000000000000000000
as value from sales where salesid<10 order by salesid;


 salesid |           value
---------+----------------------------
       1 | 72800000000000000000000.00
       2 |  76000000000000000000000.00
       3 | 35000000000000000000000.00
       4 | 17500000000000000000000.00
       5 | 15400000000000000000000.00
       6 | 39400000000000000000000.00
       7 | 78800000000000000000000.00
       8 | 19700000000000000000000.00
       9 | 59100000000000000000000.00
(9 rows)
```

> **Note**
>
> GEOMETRY 데이터 형식에 대해 CAST 또는 CONVERT 작업을 수행하여 다른 데이터 형식으로 변경할 수 없습니다. 그러나 GEOMETRY 인수를 허용하는 함수에 대한 입력으로 EWKB(Extended Well-Known Binary) 형식으로 문자열 리터럴의 16진수 표현을 제공할 수 있습니다. 예를 들어, ST_AsText 함수는 GEOMETRY 데이터 형식을 사용합니다.
>
> ```
> SELECT ST_AsText('010100000000000000000001C40000000000000002040');
> ```
>
> ```
> st_astext
> ------------
>  POINT(7 8)
> ```
>
> GEOMETRY 데이터 형식을 명시적으로 지정할 수도 있습니다.
>
> ```
> SELECT ST_AsText('010100000000000000000001440000000000000001840'::geometry);
> ```

```
st_astext
------------
 POINT(5 6)
```

CONVERT 함수

CAST 함수와 마찬가지로 CONVERT 함수는 한 데이터 유형을 호환되는 다른 데이터 유형으로 변환합니다. 예를 들어 문자열을 날짜로 변환하거나 숫자 형식을 문자열로 변환할 수 있습니다. CONVERT는 런타임 변환을 수행합니다. 즉, 변환을 수행해도 원본 테이블의 값 데이터 유형은 변경되지 않습니다. 쿼리의 컨텍스트에서만 변경됩니다.

CONVERT 함수를 사용하여 다른 데이터 유형으로 명시적인 변환이 필요한 데이터 유형이 몇 가지 있습니다. 그 밖에 CAST 또는 CONVERT를 사용하지 않고 다른 명령의 일부로서 묵시적으로 변환할 수 있는 데이터 형식들도 있습니다. 형식 호환성 및 변환 섹션을 참조하세요.

구문

```
CONVERT ( type, expression )
```

인수

type

지원되는 데이터 타입 중 한 가지입니다.

expression

열 이름이나 리터럴 같이 하나 이상의 값으로 평가되는 표현식입니다. null 값을 변환하면 마찬가지로 null이 반환됩니다. 또한 표현식에는 공백이나 빈 문자열이 포함되어서도 안 됩니다.

반환 타입

CONVERT는 type 인수에서 지정하는 데이터 형식을 반환합니다.

> **Note**
>
> 다음과 같이 문제가 발생할 수 있는 변환을 실행하면 Amazon Redshift가 오류를 반환합니다. 첫째, 정밀도를 떨어뜨리는 DECIMAL 변환입니다.

```
SELECT CONVERT(decimal(2,1), 123.456);
```

둘째, 오버플로우를 야기하는 INTEGER 변환입니다.

```
SELECT CONVERT(smallint, 12345678);
```

예시

이 안내서에 나오는 예는 대부분 샘플 TICKIT 데이터베이스를 사용합니다. 샘플 데이터 설정에 관한 자세한 내용은 데이터 로드 단원을 참조하세요.

다음 쿼리는 CONVERT 함수를 사용하여 십진수 열을 정수로 변환합니다.

```
SELECT CONVERT(integer, pricepaid)
FROM sales WHERE salesid=100;
```

이 예제에서는 정수를 문자열로 변환합니다.

```
SELECT CONVERT(char(4), 2008);
```

이 예에서는 현재 날짜 및 시간이 가변 문자 데이터 유형으로 변환됩니다.

```
SELECT CONVERT(VARCHAR(30), GETDATE());

getdate
---------
2023-02-02 04:31:16
```

이 예제에서는 saletime 열을 시간만 있는 값으로 변환하고 각 행에서 날짜를 제거합니다.

```
SELECT CONVERT(time, saletime), salesid
FROM sales order by salesid limit 10;
```

한 시간대에서 다른 시간대로 타임스탬프를 변환하는 방법에 대한 자세한 내용은 CONVERT_TIMEZONE 함수 단원을 참조하세요. 추가 날짜 및 시간 함수는 날짜 및 시간 함수 단원을 참조하세요.

다음 예제에서는 가변 문자 데이터를 datetime 객체로 변환합니다.

```
SELECT CONVERT(datetime, '2008-02-18 02:36:48') as mysaletime;
```

> **Note**
>
> GEOMETRY 데이터 형식에 대해 CAST 또는 CONVERT 작업을 수행하여 다른 데이터 형식으로 변경할 수 없습니다. 그러나 GEOMETRY 인수를 허용하는 함수에 대한 입력으로 EWKB(Extended Well-Known Binary) 형식으로 문자열 리터럴의 16진수 표현을 제공할 수 있습니다. 예를 들어, ST_AsText 함수는 GEOMETRY 데이터 형식을 사용합니다.
>
> ```
> SELECT ST_AsText('010100000000000000000001C40000000000002040');
> ```
>
> ```
> st_astext
> ------------
>  POINT(7 8)
> ```
>
> GEOMETRY 데이터 형식을 명시적으로 지정할 수도 있습니다.
>
> ```
> SELECT ST_AsText('010100000000000000000014400000000000001840'::geometry);
> ```
>
> ```
> st_astext
> ------------
>  POINT(5 6)
> ```

TO_CHAR

TO_CHAR는 타임스탬프 또는 숫자 표현식을 문자열 데이터 형식으로 변환합니다.

구문

```
TO_CHAR (timestamp_expression | numeric_expression , 'format')
```

인수

timestamp_expression

TIMESTAMP 또는 TIMESTAMPTZ 형식 값이나, 혹은 묵시적으로 타임스탬프로 강제 변환할 수 있는 값으로 평가되는 표현식입니다.

numeric_expression

숫자 데이터 형식 값이나, 혹은 묵시적으로 숫자 형식으로 강제 변환할 수 있는 값으로 평가되는 표현식입니다. 자세한 내용은 숫자형 단원을 참조하십시오. TO_CHAR는 숫자 문자열의 왼쪽에 공백을 삽입합니다.

> **Note**
>
> TO_CHAR는 128비트 DECIMAL 값을 지원하지 않습니다.

format

새로운 값의 형식입니다. 유효한 형식은 날짜/시간 형식 문자열 및 숫자 형식 문자열 섹션을 참조하세요.

반환 타입

VARCHAR

예시

다음 예는 타임스탬프를 9자에 덧붙인 월 이름, 요일 이름, 날짜를 포함한 형식의 날짜 및 시간 값으로 변환합니다.

```
select to_char(timestamp '2009-12-31 23:15:59', 'MONTH-DY-DD-YYYY HH12:MIPM');

to_char
-------------------------
DECEMBER -THU-31-2009 11:15PM
```

다음 예는 타임스탬프를 연중 일 번호 값으로 변환합니다.

```
select to_char(timestamp '2009-12-31 23:15:59', 'DDD');

to_char
-------------------------
365
```

다음 예는 타임스탬프를 ISO 요일 번호로 변환합니다.

```
select to_char(timestamp '2022-05-16 23:15:59', 'ID');

to_char
-------------------------
1
```

다음 예는 날짜에서 월 이름을 추출합니다.

```
select to_char(date '2009-12-31', 'MONTH');

to_char
-------------------------
DECEMBER
```

다음은 EVENT 테이블의 STARTTIME 값을 각각 시간, 분 및 초로 구성된 문자열로 변환하는 예입니다.

```
select to_char(starttime, 'HH12:MI:SS')
from event where eventid between 1 and 5
order by eventid;

to_char
----------
02:30:00
08:00:00
02:30:00
02:30:00
07:00:00
```

다음은 전체 타임스탬프 값을 다른 형식으로 변환하는 예입니다.

```
select starttime, to_char(starttime, 'MON-DD-YYYY HH12:MIPM')
from event where eventid=1;

      starttime      |       to_char
---------------------+---------------------
 2008-01-25 14:30:00 | JAN-25-2008 02:30PM
```

다음은 타임스탬프 리터럴을 문자열로 변환하는 예입니다.

```
select to_char(timestamp '2009-12-31 23:15:59','HH24:MI:SS');
```

```
to_char
----------
23:15:59
```

다음은 10진수를 문자열로 변환하는 예입니다.

```
select to_char(125.8, '999.99');

to_char
---------
125.80
```

다음은 10진수를 문자열로 변환하는 예입니다.

```
select to_char(125.8, '999D99');

to_char
---------
125.80
```

다음은 앞에 0이 붙는 문자열로 숫자를 변환하는 예입니다.

```
select to_char(125.8, '0999D99');

to_char
---------
0125.80
```

다음은 숫자를 끝에 음의 부호가 있는 문자열로 변환하는 예입니다.

```
select to_char(-125.8, '999D99S');

to_char
---------
125.80-
```

다음은 지정된 위치에 양수 또는 음수 부호가 있는 문자열로 숫자를 반환하는 예입니다.

```
select to_char(125.8, '999D99SG');
```

```
to_char
---------
125.80+
```

다음은 지정된 위치에 양수 부호가 있는 문자열로 숫자를 반환하는 예입니다.

```
select to_char(125.8, 'PL999D99');

to_char
---------
+ 125.80
```

다음은 숫자를 통화 기호가 있는 문자열로 변환하는 예입니다.

```
select to_char(-125.88, '$S999D99');

to_char
---------
$-125.88
```

다음은 지정된 위치에 통화 기호가 있는 문자열로 숫자를 변환하는 예입니다.

```
select to_char(-125.88, 'S999D99L');

to_char
---------
-125.88$
```

다음은 천 단위(쉼표) 구분 기호를 사용하여 문자열로 숫자를 변환하는 예입니다.

```
select to_char(1125.8, '9,999.99');

to_char
---------
1,125.80
```

다음은 음수에 꺾쇠 괄호를 사용하여 숫자를 문자열로 변환하는 예입니다.

```
select to_char(-125.88, '$999D99PR');
```

```
to_char
---------
$<125.88>
```

다음은 숫자를 로마 숫자 문자열로 변환하는 예입니다.

```
select to_char(125, 'RN');

to_char
---------
   CXXV
```

다음 예에서는 날짜를 세기 코드로 변환합니다.

```
select to_char(date '2020-12-31', 'CC');

to_char
---------
21
```

다음 예에서는 요일을 표시합니다.

```
SELECT to_char(current_timestamp, 'FMDay, FMDD HH12:MI:SS');

to_char
-----------------------
Wednesday, 31 09:34:26
```

다음 예에서는 숫자의 서수 접미사를 표시합니다.

```
SELECT to_char(482, '999th');

to_char
-----------------------
 482nd
```

다음은 SALES 테이블의 지불 가격에서 수수료를 빼는 예입니다. 그런 다음 차액을 반올림하여 to_char 열과 같이 로마 숫자로 변환합니다.

```
select salesid, pricepaid, commission, (pricepaid - commission)
```

```
as difference, to_char(pricepaid - commission, 'rn') from sales
group by sales.pricepaid, sales.commission, salesid
order by salesid limit 10;

 salesid | pricepaid | commission | difference |     to_char
---------+-----------+------------+------------+-----------------
       1 |    728.00 |     109.20 |     618.80 |           dcxix
       2 |     76.00 |      11.40 |      64.60 |             lxv
       3 |    350.00 |      52.50 |     297.50 |        ccxcviii
       4 |    175.00 |      26.25 |     148.75 |           cxlix
       5 |    154.00 |      23.10 |     130.90 |           cxxxi
       6 |    394.00 |      59.10 |     334.90 |         cccxxxv
       7 |    788.00 |     118.20 |     669.80 |           dclxx
       8 |    197.00 |      29.55 |     167.45 |          clxvii
       9 |    591.00 |      88.65 |     502.35 |             dii
      10 |     65.00 |       9.75 |      55.25 |              lv
```

다음은 to_char 열과 같이 통화 기호를 차액 값에 추가하는 예입니다.

```
select salesid, pricepaid, commission, (pricepaid - commission)
as difference, to_char(pricepaid - commission, 'l999999D99') from sales
group by sales.pricepaid, sales.commission, salesid
order by salesid limit 10;

salesid | pricepaid | commission | difference |  to_char
--------+-----------+------------+------------+------------
      1 |    728.00 |     109.20 |     618.80 | $   618.80
      2 |     76.00 |      11.40 |      64.60 | $    64.60
      3 |    350.00 |      52.50 |     297.50 | $   297.50
      4 |    175.00 |      26.25 |     148.75 | $   148.75
      5 |    154.00 |      23.10 |     130.90 | $   130.90
      6 |    394.00 |      59.10 |     334.90 | $   334.90
      7 |    788.00 |     118.20 |     669.80 | $   669.80
      8 |    197.00 |      29.55 |     167.45 | $   167.45
      9 |    591.00 |      88.65 |     502.35 | $   502.35
     10 |     65.00 |       9.75 |      55.25 | $    55.25
```

다음은 각 판매가 이루어진 세기를 나열하는 예입니다.

```
select salesid, saletime, to_char(saletime, 'cc') from sales
order by salesid limit 10;

 salesid |      saletime       | to_char
```

```
---------+---------------------+---------
       1 | 2008-02-18 02:36:48 | 21
       2 | 2008-06-06 05:00:16 | 21
       3 | 2008-06-06 08:26:17 | 21
       4 | 2008-06-09 08:38:52 | 21
       5 | 2008-08-31 09:17:02 | 21
       6 | 2008-07-16 11:59:24 | 21
       7 | 2008-06-26 12:56:06 | 21
       8 | 2008-07-10 02:12:36 | 21
       9 | 2008-07-22 02:23:17 | 21
      10 | 2008-08-06 02:51:55 | 21
```

다음은 EVENT 테이블의 STARTTIME 값을 각각 시간, 분, 초 및 시간대로 구성된 문자열로 변환하는 예입니다.

```
select to_char(starttime, 'HH12:MI:SS TZ')
from event where eventid between 1 and 5
order by eventid;

to_char
----------
02:30:00 UTC
08:00:00 UTC
02:30:00 UTC
02:30:00 UTC
07:00:00 UTC
```

다음은 초, 밀리초, 마이크로초에 따라 형식을 지정하는 예입니다.

```
select sysdate,
to_char(sysdate, 'HH24:MI:SS') as seconds,
to_char(sysdate, 'HH24:MI:SS.MS') as milliseconds,
to_char(sysdate, 'HH24:MI:SS:US') as microseconds;

timestamp           | seconds  | milliseconds | microseconds
--------------------+----------+--------------+----------------
2015-04-10 18:45:09 | 18:45:09 | 18:45:09.325 | 18:45:09:325143
```

TO_DATE 함수

TO_DATE는 문자열로 표현된 날짜를 DATE 데이터 형식으로 변환합니다.

> **Note**
>
> TO_DATE는 Q(분기 번호)가 있는 형식 문자열을 지원하지 않습니다.

구문

```
TO_DATE(string, format)
```

```
TO_DATE(string, format, is_strict)
```

인수

string

변활할 문자열입니다.

format

입력 문자열의 형식을 날짜 부분과 관련하여 정의하는 문자열 리터럴입니다. 유효한 일, 월 및 연도 형식 목록은 날짜/시간 형식 문자열 섹션을 참조하세요.

is_strict

입력 날짜 값이 범위를 벗어날 경우 오류가 반환되는지 여부를 지정하는 옵션 부울 값입니다. is_strict가 TRUE로 설정되면 범위를 벗어난 값이 있는 경우 오류가 반환됩니다. is_strict가 기본값인 FALSE로 설정되면 오버플로 값이 허용됩니다.

반환 타입

TO_DATE는 format 값에 따라 DATE를 반환합니다.

format으로의 변환이 실패하면 오류가 반환됩니다.

예시

다음 SQL 문은 날짜 `02 Oct 2001`을 날짜 데이터 형식으로 변환합니다.

```
select to_date('02 Oct 2001', 'DD Mon YYYY');
```

```
to_date
------------
2001-10-02
(1 row)
```

다음 SQL 문은 문자열 20010631을 날짜로 변환합니다.

```
select to_date('20010631', 'YYYYMMDD', FALSE);
```

결과는 2001년 7월 1일입니다. 6월은 30일만 있기 때문입니다.

```
to_date
------------
2001-07-01
```

다음 SQL 문은 문자열 20010631을 날짜로 변환합니다.

```
to_date('20010631', 'YYYYMMDD', TRUE);
```

결과는 오류입니다. 6월은 30일만 있기 때문입니다.

```
ERROR:  date/time field date value out of range: 2001-6-31
```

TO_NUMBER

TO_NUMBER는 문자열을 숫자(소수) 값으로 변환합니다.

> **Note**
>
> 패딩 공백과 0을 억제하기 위해 형식 문자열에 FM을 사용하는 것이 좋습니다. 유효한 형식 목록은 숫자 형식 문자열 섹션을 참조하세요.

구문

```
to_number(string, format)
```

인수

string

실행할 문자열입니다. 형식은 리터럴 값이 되어야 합니다.

format

두 번째 인수는 숫자 값 생성을 위한 문자열의 구문 분석 방식을 나타내는 형식 문자열입니다. 예를 들어 format이 'FM99D999'이면 변환 대상인 문자열이 5자리로 구성되어 있으며 세 번째 자리에 소수점이 있는 것을 의미합니다. 따라서 to_number('12.345','FM99D999')는 숫자 값으로 12.345를 반환합니다. 유효한 형식 목록은 숫자 형식 문자열 섹션을 참조하세요.

반환 타입

TO_NUMBER는 DECIMAL 숫자를 반환합니다.

format으로의 변환이 실패하면 오류가 반환됩니다.

예시

다음은 문자열 12,454.8-을 숫자로 변환하는 예입니다.

```
select to_number('12,454.8-', 'FM99G999D9S');

to_number
-----------
-12454.8
```

다음은 문자열 $ 12,454.88을 숫자로 변환하는 예입니다.

```
select to_number('$ 12,454.88', 'FML99G999D99');

to_number
-----------
12454.88
```

다음은 문자열 $ 2,012,454.88을 숫자로 변환하는 예입니다.

```
select to_number('$ 2,012,454.88', 'FML9,999,999.99');

to_number
-----------
```

```
2012454.88
```

TEXT_TO_INT_ALT

TEXT_TO_INT_ALT는 Teradata 스타일 형식을 사용하여 문자열을 정수로 변환합니다. 결과의 소수 자릿수가 잘립니다.

구문

```
TEXT_TO_INT_ALT (expression [ , 'format'])
```

인수

expression

열 이름이나 리터럴 문자열과 같이 하나 이상의 CHAR 또는 VARCHAR 값을 생성하는 표현식입니다. null 값을 변환하면 마찬가지로 null이 반환됩니다. 이 함수는 빈 문자열을 0으로 변환합니다.

format

입력 표현식의 형식을 정의하는 문자열 리터럴입니다. 지정할 수 있는 형식 지정 문자에 대한 자세한 내용은 숫자 데이터에 대한 Teradata 스타일 형식 지정 문자 섹션을 참조하세요.

반환 타입

TEXT_TO_INT_ALT는 INTEGER 값을 반환합니다.

캐스트 결과의 소수 부분이 잘립니다.

Amazon Redshift는 지정한 format 구문으로의 변환이 성공하지 못한 경우 오류를 반환합니다.

예시

다음 예에서는 입력 expression 문자열 '123-'를 정수 -123으로 변환합니다.

```
select text_to_int_alt('123-');
```

```
text_to_int_alt
----------
      -123
```

다음 예에서는 입력 expression 문자열 '2147483647+'를 정수 2147483647로 변환합니다.

```
select text_to_int_alt('2147483647+');
```

```
text_to_int_alt
----------
2147483647
```

다음 예에서는 입력 expression 문자열 '-123E-2'를 정수 -1로 변환합니다.

```
select text_to_int_alt('-123E-2');
```

```
text_to_int_alt
----------
        -1
```

다음 예에서는 입력 expression 문자열 '2147483647+'를 정수 2147483647로 변환합니다.

```
select text_to_int_alt('2147483647+');
```

```
text_to_int_alt
----------
2147483647
```

다음 예에서는 format 구문이 '999S'인 입력 expression 문자열 '123{'를 정수 1230으로 변환합니다. S 문자는 Signed Zoned Decimal을 나타냅니다. 자세한 내용은 숫자 데이터에 대한 Teradata 스타일 형식 지정 문자 단원을 참조하십시오.

```
select text_to_int_alt('123{', '999S');
```

```
text_to_int_alt
----------
      1230
```

다음 예에서는 format 구문이 'C9(I)'인 입력 expression 문자열 'USD123'을 정수 123으로 변환합니다. 숫자 데이터에 대한 Teradata 스타일 형식 지정 문자 섹션을 참조하세요.

```
select text_to_int_alt('USD123', 'C9(I)');
```

```
text_to_int_alt
```

```
----------
       123
```

다음 예에서는 테이블 열을 입력 expression으로 지정합니다.

```
select text_to_int_alt(a), text_to_int_alt(b) from t_text2int order by 1;
```

```
 text_to_int_alt | text_to_int_alt
-----------------+-----------------
            -123 |            -123
            -123 |            -123
             123 |             123
             123 |             123
```

다음은 이 예에 대한 테이블 정의 및 삽입 문입니다.

```
create table t_text2int (a varchar(200), b char(200));
```

```
insert into t_text2int VALUES('123', '123'),('123.123', '123.123'), ('-123', '-123'),
 ('123-', '123-');
```

TEXT_TO_NUMERIC_ALT

TEXT_TO_NUMERIC_ALT는 Teradata 스타일 캐스트 작업을 수행하여 문자열을 숫자 데이터 형식으로 변환합니다.

구문

```
TEXT_TO_NUMERIC_ALT (expression [, 'format'] [, precision, scale])
```

인수

expression

열 이름이나 리터럴 같이 하나 이상의 CHAR 또는 VARCHAR 값으로 평가되는 표현식입니다. null 값을 변환하면 마찬가지로 null이 반환됩니다. 빈 문자열은 0으로 변환됩니다.

format

입력 표현식의 형식을 정의하는 문자열 리터럴입니다. 자세한 내용은 숫자 데이터에 대한 Teradata 스타일 형식 지정 문자 단원을 참조하십시오.

precision

숫자 결과의 자릿수입니다. 기본값은 38입니다.

사용

숫자 결과에서 소수점 오른쪽의 자릿수입니다. 기본값은 0입니다.

반환 타입

TEXT_TO_NUMERIC_ALT는 DECIMAL 숫자를 반환합니다.

Amazon Redshift는 지정한 format 구문으로의 변환이 성공하지 못한 경우 오류를 반환합니다.

Amazon Redshift는 사용자가 precision 옵션에서 해당 형식에 대해 지정한 가장 높은 정밀도를 가진 숫자 형식으로 입력 expression 문자열을 캐스팅합니다. 숫자 값의 길이가 precision에 대해 지정한 값을 초과하는 경우 Amazon Redshift는 다음 규칙에 따라 숫자 값을 반올림합니다.

- 캐스트 결과의 길이가 format 구문에 지정한 길이를 초과하는 경우 Amazon Redshift는 오류를 반환합니다.
- 결과가 숫자 값으로 캐스트되면 결과는 가장 가까운 값으로 반올림됩니다. 분수 부분이 상단 및 하단 캐스트 결과의 정확히 중간인 경우 결과는 가장 가까운 짝수 값으로 반올림됩니다.

예시

다음 예에서는 입력 expression 문자열 '1.5'를 정수 값 '2'로 변환합니다. 문이 scale을 지정하지 않기 때문에 scale의 기본값은 0이고 캐스트 결과에는 분수 결과가 포함되지 않습니다. .5는 1과 2의 중간이므로 캐스트 결과는 짝수 값인 2로 반올림됩니다.

```
select text_to_numeric_alt('1.5');
```

```
 text_to_numeric_alt
---------------------
                   2
```

다음 예에서는 입력 expression 문자열 '2.51'을 정수 값 3으로 변환합니다. 문이 scale 값을 지정하지 않기 때문에 scale의 기본값은 0이고 캐스트 결과에는 분수 결과가 포함되지 않습니다. .51은 2보다 3에 더 가깝기 때문에 캐스트 결과는 3 값으로 반올림됩니다.

```
select text_to_numeric_alt('2.51');
```

```
 text_to_numeric_alt
---------------------
                   3
```

다음 예에서는 precision이 10이고 scale이 2인 입력 expression 문자열 123.52501을 숫자 값 123.53으로 변환합니다.

```
select text_to_numeric_alt('123.52501', 10, 2);
```

```
 text_to_numeric_alt
---------------------
              123.53
```

다음 예에서는 format 구문이 '999S'인 입력 expression 문자열 '123{'를 숫자 1230으로 변환합니다. S 문자는 Signed Zoned Decimal을 나타냅니다. 자세한 내용은 숫자 데이터에 대한 Teradata 스타일 형식 지정 문자 단원을 참조하십시오.

```
select text_to_numeric_alt('123{', '999S');
```

```
text_to_int_alt
----------
      1230
```

다음 예에서는 format 구문이 'C9(I)'인 입력 expression 문자열 'USD123'을 숫자 124로 변환합니다. 숫자 데이터에 대한 Teradata 스타일 형식 지정 문자 섹션을 참조하세요.

```
select text_to_numeric_alt('USD123.9', 'C9(I)');
```

```
text_to_numeric_alt
----------
       124
```

다음 예에서는 테이블 열을 입력 expression으로 지정합니다.

```
select text_to_numeric_alt(a), text_to_numeric_alt(b) from t_text2numeric order by 1;
```

```
          text_to_numeric_alt           |          text_to_numeric_alt
```

```
----------------------------------------+----------------------------------------
 -99999999999999999999999999999999999999 | -99999999999999999999999999999999999999
                                  -12300 |                                  -12300
                                     123 |                                     123
                                     123 |                                     123
  99999999999999999999999999999999999999 |  99999999999999999999999999999999999999
```

다음은 이 예에 대한 테이블 정의 및 삽입 문입니다.

```
create table  t_text2numeric (a varchar(200), b char(200));
```

```
insert into  t_text2numeric values
('123', '123'),
('+123.456', '+123.456'),
('-' || repeat('9', 38), '-' || repeat('9', 38)),
(repeat('9', 38) || '+', repeat('9', 38) || '+'),
('-123E2', '-123E2');
```

날짜/시간 형식 문자열

날짜/시간 형식 문자열을 위한 참조 정보를 찾아볼 수 있습니다.

다음 형식 문자열은 TO_CHAR 등의 함수에 적용됩니다. 이 문자열은 날짜/시간 구분자('-', '/', ':' 등)와 그 뒤를 이어 "날짜 부분" 및 "시간 부분"으로 구성됩니다.

날짜 부분 또는 시간 부분	의미
BC 또는 B.C., AD 또는 A.D., b.c. 또는 bc, ad 또는 a.d.	대문자와 소문자의 기원 표시자
CC	2자리 세기 숫자
YYYY, YYY, YY, Y	4자리, 3자리, 2자리, 1자리 연도 숫자
Y,YYY	쉼표가 포함된 4자리 연도 숫자
IYYY, IYY, IY, I	4자리, 3자리, 2자리, 1자리 국제표준화기구(ISO) 연도 숫자
Q	분기 숫자(1~4)

날짜 부분 또는 시간 부분	의미
MONTH, Month, month	월 이름(대문자, 대/소문자 혼용, 소문자, 9자까지 공백 채움)
MON, Mon, mon	월 이름 축약어(대문자, 대/소문자 혼용, 소문자, 3자까지 공백 채움)
MM	월 숫자(01-12)
RM, rm	로마 숫자로 표기한 월 숫자(I–XII, I는 1월임, 대문자 또는 소문자)
W	월중 주차(1~5, 1주차는 매월 첫 번째 날짜에 시작됨)
WW	연중 주차(1~53, 1주차는 매년 첫 번째 날짜에 시작됨)
IW	ISO 연중 주차(새해 첫 번째 목요일이 1주차의 시작임)
DAY, Day, day	요일 이름(대문자, 대/소문자 혼용, 소문자, 9자까지 공백 채움)
DY, Dy, dy	요일 이름 축약어(대문자, 대/소문자 혼용, 소문자, 3자까지 공백 채움)
DDD	연중 일(001~366)
IDDD	ISO 8601 주수 매기기 연도의 일(001-371, 해당 연도의 1일은 첫 ISO 주의 월요일)
DD	숫자 형태의 월중 일(01~31)

날짜 부분 또는 시간 부분	의미
D	주중 일(1~7, 일요일이 1임) **Note** 날짜 부분에서 D는 DATE_PART 및 EXTRACT 날짜 함수에서 사용되는 주중 일(DOW)과 다릅니다. DOW는 정수 1~6을 기반으로 합니다. 여기서 일요일은 0입니다. 자세한 내용은 날짜 또는 타임스탬프 함수의 날짜 부분 단원을 참조하십시오.
ID	ISO 8601 요일, 월요일(1)~일요일(7)
J	율리우스 일(BC 4712, 1월 1일 이후 일 수)
HH24	시간(24시간 방식, 00~23)
HH 또는 HH12	시간(12시간 방식, 01~12)
MI	분(00~59)
SS	초(00~59)
MS	MS
US	US
AM 또는 PM, A.M. 또는 P.M., a.m. 또는 p.m., am 또는 pm	대문자와 소문자의 오전/오후 표시자(12시간 방식일 때)
TZ, tz	대문자와 소문자의 시간대 축약어로서 TIMESTAMPTZ에 한해 유효함
OF	UTC 기준 오프셋으로서 TIMESTAMPTZ에 한해 유효함

> **Note**
>
> 날짜/시간 구분자(예: '-', '/' 또는 ':')를 작은따옴표로 묶어야 하지만, 이전 테이블에 나열된 "날짜 부분"과 "시간 부분"은 큰따옴표로 묶어야 합니다.

예시

날짜를 문자열 형식으로 지정하는 예는 TO_CHAR 단원을 참조하세요.

숫자 형식 문자열

다음에서 숫자 형식 문자열에 대한 참조를 찾을 수 있습니다.

다음 형식 문자열은 TO_NUMBER 및 TO_CHAR 등의 함수에 적용됩니다.

- 문자열을 숫자 형식으로 지정하는 예는 TO_NUMBER 섹션을 참조하세요.
- 숫자를 문자열 형식으로 지정하는 예는 TO_CHAR 섹션을 참조하세요.

형식	설명
9	지정한 자리 수를 포함한 숫자 값
0	선행 제로를 포함한 숫자 값
.(마침표), D	소수점
, (comma)	천 단위 구분자
CC	세기 코드. 예를 들어 21세기는 2001-01-01부터 시작됩니다(TO_CHAR에서만 지원됨).
FM	채우기 모드. 공백 및 제로 채움은 제한됩니다.
PR	꺾쇠 괄호 안의 음수 값
S	숫자에 첨부되는 부호
L	지정한 위치의 통화 기호

형식	설명
G	그룹 구분자
MI	0보다 작은 숫자일 때 지정한 위치의 마이너스 부호
PL	0보다 큰 숫자일 때 지정한 위치의 플러스 부호
SG	지정한 위치의 플러스 또는 마이너스 부호
RN	1부터 3999 사이의 로마 숫자(TO_CHAR에 한해 지원됨)
TH 또는 th	서수 접미사. 분수 또는 0보다 작은 값은 변환하지 않습니다.

숫자 데이터에 대한 Teradata 스타일 형식 지정 문자

다음으로 TEXT_TO_INT_ALT 및 TEXT_TO_NUMERIC_ALT 함수가 입력 expression 문자열의 문자를 해석하는 방법을 찾아볼 수 있습니다. format 구문에서 지정할 수 있는 문자 목록도 찾아볼 수 있습니다. 또한 format 옵션에 대해 Teradata 스타일 형식 지정과 Amazon Redshift 간의 차이점에 대한 설명을 찾아볼 수 있습니다.

형식	설명
G	입력 expression 문자열에서 그룹 구분 기호로 지원되지 않습니다. format 구문에 이 문자를 지정할 수 없습니다.
D	기수 기호입니다. format 구문에 이 문자를 지정할 수 있습니다. 이 문자는 .(마침표)에 해당합니다. 기수 기호는 다음 문자가 포함된 format 구문에 나타날 수 없습니다. • . (마침표) • S(대문자 's')

형식	설명
	• V(대문자 'v')
/ , : %	삽입 문자 /(슬래시), 쉼표(,), :(콜론) 및 %(퍼센트 기호)입니다. format 구문에 이러한 문자를 포함할 수 없습니다. Amazon Redshift는 입력 expression 문자열에서 이러한 문자를 무시합니다.
.	소수점을 나타내는 기수 문자로서의 마침표입니다. 이 문자는 다음 문자가 포함된 format 구문에 나타날 수 없습니다. • D(대문자 'd') • S(대문자 's') • V(대문자 'v')
B	format 구문에 공백 문자(B)를 포함할 수 없습니다. 입력 expression 문자열에서 선행 및 후행 공백은 무시되고 숫자 사이의 공백은 허용되지 않습니다.
+ -	format 구문에 더하기 기호(+) 또는 빼기 기호(-)를 포함할 수 없습니다. 그러나 더하기 기호(+) 및 빼기 기호(-)가 입력 expression 문자열에 나타나는 경우 숫자 값의 일부로 암시적으로 구문 분석됩니다.

형식	설명
V	소수점 위치 표시기입니다. 이 문자는 다음 문자가 포함된 format 구문에 나타날 수 없습니다. • D(대문자 'd') • . (마침표)
Z	0이 숨겨진 소수점 이하 자릿수입니다. Amazon Redshift는 선행 0을 자릅니다. Z 문자는 9 문자를 따를 수 없습니다. 분수 부분에 9 문자가 포함된 경우 Z 문자는 기수 문자 왼쪽에 있어야 합니다.
9	소수점 이하 자릿수입니다.
CHAR(n)	이 형식의 경우 다음을 지정할 수 있습니다. • CHAR는 Z 또는 9 문자로 구성됩니다. Amazon Redshift는 CHAR 값에서 +(더하기) 또는 -(빼기)를 지원하지 않습니다. • n은 정수 상수, I 또는 F입니다. I의 경우 숫자 또는 정수 데이터의 정수 부분을 표시하는 데 필요한 문자 수입니다. F의 경우 숫자 데이터의 소수 부분을 표시하는 데 필요한 문자 수입니다.
-	하이픈(-) 문자입니다. format 구문에 이 문자를 포함할 수 없습니다. Amazon Redshift는 입력 expression 문자열에서 이 문자를 무시합니다.

형식	설명
S	Signed Zoned Decimal입니다. S 문자는 format 구문의 마지막 소수점 이하 자릿수 뒤에 와야 합니다. Signed Zone Decimal, Teradata 스타일의 숫자 데이터 형식 지정을 위한 데이터 형식 지정 문자입니다.에는 입력 expression 문자열의 마지막 문자와 해당 숫자 변환이 나열됩니다. S 문자는 다음 문자가 포함된 format 구문에 나타날 수 없습니다. • +(더하기 기호) • . (마침표) • D(대문자 'd') • Z(대문자 'z') • F(대문자 'f') • E(대문자 'e')
E	지수 표기법입니다. 입력 expression 문자열은 지수 문자를 포함할 수 있습니다. format 구문에서 E를 지수 문자로 지정할 수 없습니다.
FN9	Amazon Redshift에서는 지원되지 않습니다.
FNE	Amazon Redshift에서는 지원되지 않습니다.
$, USD, 미국 달러	달러 기호($), ISO 통화 기호(USD) 및 통화 이름 미국 달러입니다. ISO 통화 기호 USD와 통화 이름 미국 달러는 대/소문자를 구분합니다. Amazon Redshift는 USD 통화만 지원합니다. 입력 expression 문자열은 USD 통화 기호와 숫자 값 사이에 공백을 포함할 수 있습니다(예: '$ 123E2' 또는 '123E2 $').

형식	설명
L	통화 기호입니다. 이 통화 기호 문자는 format 구문에 한 번만 나타날 수 있습니다. 반복되는 통화 기호 문자를 지정할 수 없습니다.
C	ISO 통화 기호입니다. 이 통화 기호 문자는 format 구문에 한 번만 나타날 수 있습니다. 반복되는 통화 기호 문자를 지정할 수 없습니다.
N	전체 통화 이름입니다. 이 통화 기호 문자는 format 구문에 한 번만 나타날 수 있습니다. 반복되는 통화 기호 문자를 지정할 수 없습니다.
O	이중 통화 기호입니다. format 구문에 이 문자를 지정할 수 없습니다.
U	이중 ISO 통화 기호입니다. format 구문에 이 문자를 지정할 수 없습니다.
A	전체 이중 통화 이름입니다. format 구문에 이 문자를 지정할 수 없습니다.

Signed Zone Decimal, Teradata 스타일의 숫자 데이터 형식 지정을 위한 데이터 형식 지정 문자입니다.

Signed Zoned Decimal 값에 대해 TEXT_TO_INT_ALT 및 TEXT_TO_NUMERIC_ALT 함수의 format 구문에 다음 문자를 사용할 수 있습니다.

입력 문자열의 마지막 문자	숫자 변환
{ 또는 0	n … 0
A 또는 1	n … 1
B 또는 2	n … 2
C 또는 3	n … 3

입력 문자열의 마지막 문자	숫자 변환
D 또는 4	n … 4
E 또는 5	n … 5
F 또는 6	n … 6
G 또는 7	n … 7
H 또는 8	n … 8
I 또는 9	n … 9
}	-n … 0
J	-n … 1
K	-n … 2
L	-n … 3
M	-n … 4
N	-n … 5
O	-n … 6
P	-n … 7
Q	-n … 8
R	-n … 9

날짜 및 시간 함수

이번 섹션에서는 Amazon Redshift에서 지원되는 날짜 및 시간 스칼라 함수에 대한 정보를 찾아볼 수 있습니다.

주제

- 날짜 및 시간 함수 요약

- 트랜잭션의 날짜 및 시간 함수
- 지원되지 않는 리더 노드 전용 함수
- +(연결) 연산자
- ADD_MONTHS 함수
- AT TIME ZONE 함수
- CONVERT_TIMEZONE 함수
- CURRENT_DATE 함수
- DATE_CMP 함수
- DATE_CMP_TIMESTAMP 함수
- DATE_CMP_TIMESTAMPTZ 함수
- DATEADD 함수
- DATEDIFF 함수
- DATE_PART 함수
- DATE_PART_YEAR 함수
- DATE_TRUNC 함수
- EXTRACT 함수
- GETDATE 함수
- INTERVAL_CMP 함수
- LAST_DAY 함수
- MONTHS_BETWEEN 함수
- NEXT_DAY 함수
- SYSDATE 함수
- TIMEOFDAY 함수
- TIMESTAMP_CMP 함수
- TIMESTAMP_CMP_DATE 함수
- TIMESTAMP_CMP_TIMESTAMPTZ 함수
- TIMESTAMPTZ_CMP 함수
- TIMESTAMPTZ_CMP_DATE 함수
- TIMESTAMPTZ_CMP_TIMESTAMP 함수

- TIMEZONE 함수
- TO_TIMESTAMP 함수
- TRUNC 함수
- 날짜 또는 타임스탬프 함수의 날짜 부분

날짜 및 시간 함수 요약

함수	구문	반환 값
+(연결) 연산자 날짜를 + 기호의 양쪽에 있는 시간에 연결하고 TIMESTAMP 또는 TIMESTAMPTZ를 반환합니다.	date + time	TIMESTAMP 또는 TIMESTAMPTZ
ADD_MONTHS 지정한 월 수를 날짜 또는 타임스탬프에 더합니다.	ADD_MONTHS ({date|timestamp}, integer)	TIMESTAMP
AT TIME ZONE TIMESTAMP 또는 TIMESTAMPTZ 표현식에 사용할 시간대를 지정합니다.	AT TIME ZONE 'timezone'	TIMESTAMP 또는 TIMESTAMPTZ
CONVERT_TIMEZONE 시간대끼리 타임스탬프를 변환합니다.	CONVERT_TIMEZONE (['timezone',] 'timezone', timestamp)	TIMESTAMP
CURRENT_DATE 현재 세션 시간대(기본 UTC)의 날짜를 현재 트랜잭션 시작에 맞춰 반환합니다.	CURRENT_DATE	DATE
DATE_CMP 두 날짜를 비교하여 날짜가 동일하면 0, date1이 더 크면 1, date2가 더 크면 -1을 반환합니다.	DATE_CMP (date1, date2)	INTEGER

함수	구문	반환 값
DATE_CMP_TIMESTAMP 날짜를 타임스탬프와 비교하여 값이 동일하면 0, date 값이 더 크면 1, timestamp 값이 더 크면 -1을 반환합니다.	DATE_CMP_TIMESTAMP (date, timestamp)	INTEGER
DATE_CMP_TIMESTAMPTZ 날짜 및 타임스탬프를 시간대와 비교하여 값이 동일하면 0, date 값이 더 크면 1, timestamptz 값이 더 크면 -1을 반환합니다.	DATE_CMP_TIMESTAMPTZ (date, timestamptz)	INTEGER
DATE_PART_YEAR 날짜에서 연도를 추출합니다.	DATE_PART_YEAR (date)	INTEGER
DATEADD 날짜 또는 시간을 지정하는 간격으로 늘립니다.	DATEADD (datepart, interval, {date|time|timetz|timestamp})	TIMESTAMP , TIME 또는 TIMETZ
DATEDIFF 일 또는 월처럼 임의의 날짜 부분에 대한 두 날짜 또는 시간의 차이점을 반환합니다.	DATEDIFF (datepart, {date|time|timetz|timestamp}, {date|time|timetz|timestamp})	BIGINT
DATE_PART 날짜 또는 시간에서 날짜 부분 값을 추출합니다.	DATE_PART (datepart, {date|timestamp})	DOUBLE
DATE_TRUNC 날짜 부분을 기준으로 타임스탬프를 자릅니다.	DATE_TRUNC ('datepart', timestamp)	TIMESTAMP
EXTRACT timestamp, timestamptz, time 또는 timetz에서 날짜 또는 시간 부분을 추출합니다.	EXTRACT (datepart FROM source)	INTEGER or DOUBLE

함수	구문	반환 값
GETDATE 현재 세션 시간대(기본 UTC)의 현재 날짜 및 시간을 반환합니다. 괄호가 필요합니다.	GETDATE()	TIMESTAMP
INTERVAL_CMP 두 간격을 서로 비교하여 간격이 같으면 0, interval1이 더 크면 1, interval2가 더 크면 -1을 반환합니다.	INTERVAL_CMP (interval1, interval2)	INTEGER
LAST_DAY date가 포함된 월의 마지막 날짜를 반환합니다.	LAST_DAY(date)	DATE
MONTHS_BETWEEN 두 날짜 사이의 월 수를 반환합니다.	MONTHS_BETWEEN (date, date)	FLOAT8
NEXT_DAY date 이후 day가 처음 도래하는 날짜를 반환합니다.	NEXT_DAY (date, day)	DATE
SYSDATE 날짜 및 시간을 현재 트랜잭션 시작에 맞춰 UTC로 반환합니다.	SYSDATE	TIMESTAMP
TIMEOFDAY 현재 세션 시간대(기본 UTC)의 현재 평일, 날짜 및 시간을 문자열 값으로 반환합니다.	TIMEOFDAY()	VARCHAR
TIMESTAMP_CMP 두 타임스탬프를 비교하여 타임스탬프가 같으면 0, timestamp1이 더 크면 1, timestamp2가 더 크면 -1을 반환합니다.	TIMESTAMP_CMP (timestamp1, timestamp2)	INTEGER

함수	구문	반환 값
TIMESTAMP_CMP_DATE 타임스탬프를 날짜와 비교하여 값이 동일하면 0, timestamp가 더 크면 1, date가 더 크면 -1을 반환합니다.	TIMESTAMP_CMP_DATE (timestamp, date)	INTEGER
TIMESTAMP_CMP_TIMESTAMPTZ 타임스탬프를 타임스탬프 및 시간대와 비교하여 값이 같으면 0, timestamp가 더 크면 1, timestamptz가 더 크면 -1을 반환합니다.	TIMESTAMP_CMP_TIMESTAMPTZ (timestamp, timestamptz)	INTEGER
TIMESTAMPTZ_CMP 두 타임스탬프를 시간대 값과 비교하여 값이 같으면 0, timestamptz1이 더 크면 1, timestamptz2가 더 크면 -1을 반환합니다.	TIMESTAMPTZ_CMP (timestamptz1, timestamptz2)	INTEGER
TIMESTAMPTZ_CMP_DATE 타임스탬프 값을 시간대 및 날짜와 비교하여 값이 같으면 0, timestamptz가 더 크면 1, date가 더 크면 -1을 반환합니다.	TIMESTAMPTZ_CMP_DATE (timestamptz, date)	INTEGER
TIMESTAMPTZ_CMP_TIMESTAMP 타임스탬프를 시간대 및 타임스탬프와 비교하여 값이 같으면 0, timestamptz가 더 크면 1, timestamp가 더 크면 -1을 반환합니다.	TIMESTAMPTZ_CMP_TIMESTAMP (timestamptz, timestamp)	INTEGER
TIMEZONE 지정한 시간대와 타임스탬프 값에 대한 타임스탬프를 반환합니다.	TIMEZONE ('timezone' { timestamp | timestamptz)	TIMESTAMP 또는 TIMESTAMPTZ

함수	구문	반환 값
TO_TIMESTAMP 지정한 타임스탬프와 시간대 형식에 대하여 시간대를 포함한 타임스탬프를 반환합니다.	TO_TIMESTAMP ('timestamp', 'format')	TIMESTAMP TZ
TRUNC 타임스탬프를 자르고 날짜를 반환합니다.	TRUNC(timestamp)	DATE

> **Note**
>
> 경과 시간을 계산할 때 윤초는 고려하지 않습니다.

트랜잭션의 날짜 및 시간 함수

다음 함수를 트랜잭션 블록(BEGIN … END) 내에서 실행할 경우에는 함수가 현재 문이 아닌 현재 트랜잭션의 시작 날짜 또는 시간을 반환합니다.

- SYSDATE
- TIMESTAMP
- CURRENT_DATE

다음 함수는 트랜잭션 블록 내에서도 항상 현재 문의 시작 날짜 또는 시간을 반환합니다.

- GETDATE
- TIMEOFDAY

지원되지 않는 리더 노드 전용 함수

다음 날짜 함수는 리더 노드에서만 실행되기 때문에 더 이상 지원되지 않습니다. 자세한 내용은 리더 노드 전용 함수 단원을 참조하십시오.

- age. 대신 DATEDIFF 함수을 사용하세요.
- Current Time. 대신에 GETDATE 함수 또는 SYSDATE를 사용하십시오.

- CURRENT_TIMESTAMP. 대신에 GETDATE 함수 또는 SYSDATE를 사용하십시오.
- LOCALTIME. 대신에 GETDATE 함수 또는 SYSDATE를 사용하십시오.
- LOCALTIMESTAMP. 대신에 GETDATE 함수 또는 SYSDATE를 사용하십시오.
- ISFINITE
- NOW. 대신에 GETDATE 함수 또는 SYSDATE를 사용하십시오.

+(연결) 연산자

DATE를 + 기호의 양쪽에 있는 TIME 또는 TIMETZ에 연결하고 TIMESTAMP 또는 TIMESTAMPTZ를 반환합니다.

구문

```
date + {time | timetz}
```

인수의 순서는 반대로 할 수 있습니다. 예를 들어 time + date입니다.

인수

date

DATE 데이터 형식의 열 또는 암시적으로 DATE 형식으로 평가되는 표현식입니다.

time

TIME 데이터 형식의 열 또는 암시적으로 TIME 형식으로 평가되는 표현식입니다.

timetz

TIMETZ 데이터 형식의 열 또는 암시적으로 TIMETZ 형식으로 평가되는 표현식입니다.

반환 타입

입력이 date + time인 경우 TIMESTAMP입니다.

입력이 date + timetz인 경우 TIMESTAMPTZ입니다.

예시

예제 설정

예제에서 사용된 TIME_TEST 및 TIMETZ_TEST 테이블을 설정하려면 다음 명령을 사용합니다.

```
create table time_test(time_val time);

insert into time_test values
('20:00:00'),
('00:00:00.5550'),
('00:58:00');

create table timetz_test(timetz_val timetz);

insert into timetz_test values
('04:00:00+00'),
('00:00:00.5550+00'),
('05:58:00+00');
```

시간 열이 있는 예

다음 예제 테이블 TIME_TEST에는 3개의 값이 삽입된 TIME_VAL(TIME 형식) 열이 있습니다.

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

다음 예에서는 날짜 리터럴과 TIME_VAL 열을 연결합니다.

```
select date '2000-01-02' + time_val as ts from time_test;

ts
---------------------
2000-01-02 20:00:00
2000-01-02 00:00:00.5550
2000-01-02 00:58:00
```

다음 예에서는 날짜 리터럴과 시간 리터럴을 연결합니다.

```
select date '2000-01-01' + time '20:00:00' as ts;

         ts
```

```
---------------------
 2000-01-01 20:00:00
```

다음 예에서는 시간 리터럴과 날짜 리터럴을 연결합니다.

```
select time '20:00:00' + date '2000-01-01' as ts;

         ts
---------------------
 2000-01-01 20:00:00
```

TIMETZ 열이 있는 예

다음 예제 테이블 TIMETZ_TEST에는 3개의 값이 삽입된 TIMETZ_VAL(TIMETZ 형식) 열이 있습니다.

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

다음 예에서는 날짜 리터럴과 TIMETZ_VAL 열을 연결합니다.

```
select date '2000-01-01' + timetz_val as ts from timetz_test;
ts
---------------------
2000-01-01 04:00:00+00
2000-01-01 00:00:00.5550+00
2000-01-01 05:58:00+00
```

다음 예에서는 TIMETZ_VAL 열과 날짜 리터럴을 연결합니다.

```
select timetz_val + date '2000-01-01' as ts from timetz_test;
ts
---------------------
2000-01-01 04:00:00+00
2000-01-01 00:00:00.5550+00
2000-01-01 05:58:00+00
```

다음 예에서는 DATE 리터럴과 TIMETZ 리터럴을 연결합니다. 이 예제에서는 기본적으로 UTC 표준 시간대인 TIMESTAMPTZ를 반환합니다. UTC는 PST보다 8시간 빠르므로 결과는 입력 시간보다 8시간 앞당겨집니다.

```
select date '2000-01-01' + timetz '20:00:00 PST' as ts;

          ts
------------------------
 2000-01-02 04:00:00+00
```

ADD_MONTHS 함수

ADD_MONTHS는 지정한 월 수를 날짜 또는 타임스탬프 값이나 표현식에 더합니다. DATEADD 함수가 이와 유사한 기능을 제공합니다.

구문

```
ADD_MONTHS( {date | timestamp}, integer)
```

인수

date | timestamp

DATE 또는 TIMESTAMP 데이터 형식의 DATE 또는 TIMESTAMP 형식으로 암시적으로 계산되는 표현식입니다. 날짜가 월의 마지막 날이거나, 혹은 결과에 따른 월이 더 짧은 경우에는 함수가 월의 마지막 날을 결과로 반환합니다. 그 밖에 다른 날짜일 때는 날짜 표현식과 동일한 날의 숫자가 결과로 반환됩니다.

integer

INTEGER 데이터 형식의 값입니다. 날짜에서 월을 뺄 때는 음의 정수를 사용합니다.

반환 타입

TIMESTAMP

예시

다음은 TRUNC 함수 내에서 ADD_MONTHS 함수를 사용하는 쿼리입니다. TRUNC 함수는 ADD_MONTHS 결과에서 시간 부분을 제거합니다. ADD_MONTHS 함수는 CALDATE 열의 각 값에 12개월을 합산합니다. CALDATE 열의 값은 날짜입니다.

```
select distinct trunc(add_months(caldate, 12)) as calplus12,
trunc(caldate) as cal
from date
order by 1 asc;

 calplus12  |    cal
------------+------------
 2009-01-01 | 2008-01-01
 2009-01-02 | 2008-01-02
 2009-01-03 | 2008-01-03
...
(365 rows)
```

다음 예제에서는 ADD_MONTHS 함수를 사용하여 타임스탬프에 1개월을 추가합니다.

```
select add_months('2008-01-01 05:07:30', 1);

add_months
---------------------
2008-02-01 05:07:30
```

다음은 일 수가 서로 다른 월의 날짜에 대해 ADD_MONTHS 함수를 실행할 경우 동작에 대해서 설명하는 예입니다. 이 예제에서는 함수가 3월 31일에 1개월 추가 및 4월 30일에 1개월 추가를 처리하는 방법을 보여 줍니다. 4월은 30일이므로 3월 31일에 1개월을 더하면 4월 30일이 됩니다. 5월은 31일이므로 4월 30일에 1개월을 더하면 5월 31일이 됩니다.

```
select add_months('2008-03-31',1);

add_months
---------------------
2008-04-30 00:00:00

select add_months('2008-04-30',1);

add_months
---------------------
2008-05-31 00:00:00
```

AT TIME ZONE 함수

AT TIME ZONE은 TIMESTAMP 또는 TIMESTAMPTZ 표현식에 사용할 시간대를 지정합니다.

구문

```
AT TIME ZONE 'timezone'
```

인수

시간대

반환 값의 TIMEZONE입니다. 시간대는 시간대 이름(**'Africa/Kampala'**, **'Singapore'** 등) 또는 시간대 약어(**'UTC'**, **'PDT'** 등)로 지정할 수 있습니다.

지원되는 시간대 이름 목록을 보려면 다음 명령을 실행합니다.

```
select pg_timezone_names();
```

지원되는 시간대 이름 약어 목록을 보려면 다음 명령을 실행합니다.

```
select pg_timezone_abbrevs();
```

자세한 정보와 지침은 시간대 사용 노트 섹션을 참조하세요.

반환 타입

TIMESTAMP 표현식과 함께 사용할 경우에는 TIMESTAMPTZ이고, TIMESTAMPTZ 표현식과 함께 사용할 경우에는 TIMESTAMP입니다.

예시

다음은 시간대를 제외한 타임스탬프 값을 변환하여 MST 시간(POSIX의 UTC+7)로 해석하는 예입니다. 이 예시는 UTC 시간대에 대해 TIMESTAMPTZ 데이터 유형의 값을 반환합니다. 기본 시간대를 UTC 이외의 시간대로 구성하면 다른 결과가 표시될 수 있습니다.

```
SELECT TIMESTAMP '2001-02-16 20:38:40' AT TIME ZONE 'MST';

timezone
------------------------
2001-02-17 03:38:40+00
```

다음은 지정한 시간대가 EST(POSIX의 UTC+5)일 때 시간대 값을 포함한 입력 타임스탬프를 MST(POSIX의 UTC+7)로 변환하는 예입니다. 이 예에서는 TIMESTAMP 데이터 형식의 값을 반환합니다.

```
SELECT TIMESTAMPTZ '2001-02-16 20:38:40-05' AT TIME ZONE 'MST';

timezone
------------------------
2001-02-16 18:38:40
```

CONVERT_TIMEZONE 함수

CONVERT_TIMEZONE은 시간대끼리 타임스탬프를 변환합니다. 이 함수는 일광 절약 시간에 맞춰 자동으로 조정됩니다.

구문

```
CONVERT_TIMEZONE( ['source_timezone',] 'target_timezone', 'timestamp')
```

인수

source_timezone

(옵션) 현재 타임스탬프의 시간대입니다. 기본값은 UTC입니다. 자세한 내용은 시간대 사용 노트 단원을 참조하십시오.

target_timezone

새로운 타임스탬프의 시간대입니다. 자세한 내용은 시간대 사용 노트 단원을 참조하십시오.

timestamp

타임스탬프 열 또는 묵시적으로 타임스탬프로 변환되는 표현식입니다.

반환 타입

TIMESTAMP

시간대 사용 노트

source_timezone 또는 target_timezone은 시간대 이름('Africa/Kampala', 'Singapore' 등)이나 시간대 약어('UTC', 'PDT' 등)로 지정할 수 있습니다. 시간대 이름을 이름으로 변환하거나 약어를 약어로 변환할 필요가 없습니다. 예를 들어 소스 시간대 이름인 'Singapore'에서 타임스탬프를 선택하고 이를 시간대 약어 'PDT'의 타임스탬프로 변환할 수 있습니다.

> **Note**
>
> 시간대 이름 또는 시간대 약어 사용의 결과는 일광 절약 시간 등 지역별 계절 시간으로 인해 다를 수 있습니다.

시간대 이름 사용

현재 시간대 이름 및 시간대 이름의 전체 목록을 보려면 다음 명령을 실행하세요.

```
select pg_timezone_names();
```

각 행에는 시간대 이름, 약어, UTC 오프셋 및 시간대가 일광 절약 시간제(t 또는 f)를 준수하는지 여부를 표시하는 쉼표로 구분된 문자열이 포함됩니다. 예를 들어 다음 코드 조각은 두 개의 결과 행을 보여줍니다. 첫 번째 행은 Europe/Paris 시간대(약어 CET)이며 UTC에서 01:00:00 시간이 오프셋되고 f는 일광 절약 시간제를 준수하지 않음을 나타냅니다. 두 번째 행은 Israel 시간대(약어 IST)이며 UTC에서 02:00:00 시간이 오프셋되고 f는 일광 절약 시간제를 준수하지 않음을 나타냅니다.

```
pg_timezone_names
-----------------
(Europe/Paris,CET,01:00:00,f)
(Israel,IST,02:00:00,f)
```

SQL 문을 실행하여 전체 목록을 가져오고 시간대 이름을 찾으세요. 대략 600개의 행이 반환됩니다. 반환된 일부 시간대 이름이 대문자 머리글자이거나 약어(GB, PRC, ROK 등)이기는 하지만 CONVERT_TIMEZONE 함수는 이를 시간대 약어가 아닌 시간대 이름으로 처리합니다.

시간대를 시간대 이름으로 지정하는 경우에는 CONVERT_TIMEZONE 함수가 일광 절약 시간(DST)에 맞춰, 혹은 서머 타임, 스탠다드 타임, 윈터 타임 같이 'timestamp'에서 지정한 날짜 및 시간에 해당하는 시간대의 지역별 계절 프로토콜에 맞춰 자동 조정합니다. 예를 들어 'Europe/London'은 겨울에는 UTC를 나타내고 여름에는 1시간이 추가됩니다.

시간대 약어 사용

현재 시간대 약어 및 시간대 약어의 전체 목록을 보려면 다음 명령을 실행하세요.

```
select pg_timezone_abbrevs();
```

결과에는 시간대 약어, UTC 오프셋 및 시간대가 일광 절약 시간제(t 또는 f)를 준수하는지 여부를 표시하는 쉼표로 구분된 문자열이 포함됩니다. 예를 들어 다음 코드 조각은 두 개의 결과 행을 보여줍

니다. 첫 번째 행에는 UTC에서 -07:00:00 오프셋이 있는 일광 절약 태평양 표준시(Pacific Daylight Time, 약어 PDT)가 포함되며 t는 일광 절약 시간제를 준수함을 나타냅니다. 두 번째 행에는 UTC에서 -08:00:00 오프셋이 있는 태평양 표준시(Pacific Standard Time, 약어 PST)가 포함되며 f는 일광 절약 시간제를 준수하지 않을 나타냅니다.

```
pg_timezone_abbrevs
--------------------
(PDT,-07:00:00,t)
(PST,-08:00:00,f)
```

SQL 문을 실행하여 전체 목록을 얻고 해당 오프셋 및 일광 절약 표시기를 기반으로 약어를 찾으세요. 대략 200개의 행이 반환됩니다.

시간대 약어는 UTC의 고정 오프셋을 나타냅니다. 시간대를 시간대 약어로 지정하는 경우에는 CONVERT_TIMEZONE 함수가 UTC의 고정 오프셋을 사용하기 때문에 지역별 계정 프로토콜에 따라 조정하지 않습니다.

POSIX 스타일 형식 사용

POSIX-스타일 시간대 명세는 STDoffset 또는 STDoffsetDST 형식을 따릅니다. 여기서 STD는 시간대의 약어이고, offset은 UTC의 숫자 오프셋(서부 시간)이고, DST는 일광 절약 시간대의 약어(옵션)입니다. 일광 절약 시간은 지정한 오프셋보다 1시간 앞서는 것으로 가정합니다.

POSIX-스타일 시간대 형식은 Greenwich에서 서쪽으로 양의 오프셋을 사용하는 반면 ISO-8601 규약은 Greenwich에서 동쪽으로 양의 오프셋을 사용합니다.

다음은 POSIX-스타일 시간대의 예입니다.

- PST8
- PST8PDT
- EST5
- EST5EDT

> **Note**
>
> Amazon Redshift는 POSIX 스타일 시간대 사양의 유효성을 검사하지 않으므로 시간대를 잘못된 값으로 설정할 수 있습니다. 예를 들어 다음 명령을 시간대를 잘못된 값으로 설정하지만 오류를 반환하지 않습니다.

```
set timezone to ‘xxx36’;
```

예시

이 안내서에 나오는 예는 대부분 TICKIT 샘플 데이터 세트를 사용합니다. 자세한 내용은 샘플 데이터베이스를 참조하세요.

다음은 타임스탬프 값을 기본 UTC 시간대에서 PST로 변환하는 예입니다.

```
select convert_timezone('PST', '2008-08-21 07:23:54');

 convert_timezone
------------------------
2008-08-20 23:23:54
```

다음은 LISTTIME 열의 타임스탬프 값을 기본 UTC 시간대에서 PST로 변환하는 예입니다. 타임스탬프가 일광 절약 시간(PST)에 해당하더라도 대상 시간대가 약어로 지정되어 있기 때문에 스탠다드 타임으로 변환됩니다.

```
select listtime, convert_timezone('PST', listtime) from listing
where listid = 16;

     listtime       |   convert_timezone
--------------------+---------------------
2008-08-24 09:36:12     2008-08-24 01:36:12
```

다음은 LISTTIME 열의 타임스탬프 값을 기본 UTC 시간대에서 US/Pacific 시간대로 변환하는 예입니다. 대상 시간대가 시간대 이름을 사용하고 있고, 타임스탬프가 일광 절약 시간에 해당하기 때문에 함수가 일광 절약 시간을 반환합니다.

```
select listtime, convert_timezone('US/Pacific', listtime) from listing
where listid = 16;

     listtime       |   convert_timezone
--------------------+---------------------
2008-08-24 09:36:12 | 2008-08-24 02:36:12
```

다음은 타임스탬프 문자열을 EST에서 PST로 변환하는 예입니다.

```
select convert_timezone('EST', 'PST', '20080305 12:25:29');

 convert_timezone
-------------------
2008-03-05 09:25:29
```

다음은 대상 시간대가 시간대 이름(America/New_York)을 사용하고 있고, 타임스탬프가 스탠다드 타임에 해당하기 때문에 타임스탬프를 미국 동부 스탠다드 타임으로 변환하는 예입니다.

```
select convert_timezone('America/New_York', '2013-02-01 08:00:00');

 convert_timezone
---------------------
2013-02-01 03:00:00
(1 row)
```

다음은 대상 시간대가 시간대 이름(America/New_York)을 사용하고 있고, 타임스탬프가 일광 절약 시간에 해당하기 때문에 타임스탬프를 미국 동부 일광 절약 시간으로 변환하는 예입니다.

```
select convert_timezone('America/New_York', '2013-06-01 08:00:00');

 convert_timezone
---------------------
2013-06-01 04:00:00
(1 row)
```

다음은 오프셋의 사용을 설명하는 예입니다.

```
SELECT CONVERT_TIMEZONE('GMT','NEWZONE +2','2014-05-17 12:00:00') as newzone_plus_2,
CONVERT_TIMEZONE('GMT','NEWZONE-2:15','2014-05-17 12:00:00') as newzone_minus_2_15,
CONVERT_TIMEZONE('GMT','America/Los_Angeles+2','2014-05-17 12:00:00') as la_plus_2,
CONVERT_TIMEZONE('GMT','GMT+2','2014-05-17 12:00:00') as gmt_plus_2;

   newzone_plus_2    | newzone_minus_2_15  |      la_plus_2      |     gmt_plus_2
---------------------+---------------------+---------------------+---------------------
2014-05-17 10:00:00 | 2014-05-17 14:15:00 | 2014-05-17 10:00:00 | 2014-05-17 10:00:00
(1 row)
```

CURRENT_DATE 함수

CURRENT_DATE는 현재 세션 시간대(기본 UTC)의 날짜를 기본 형식(YYYY-MM-DD)으로 반환합니다.

> **Note**
>
> CURRENT_DATE는 현재 문이 아닌 현재 트랜잭션의 시작 날짜를 반환합니다. 10/01/08 23:59에 여러 문이 포함된 트랜잭션을 시작하고, CURRENT_DATE가 포함된 문이 10/02/08 00:00에 실행되는 시나리오를 생각해 보세요. CURRENT_DATE는 `10/02/08`이 아닌 `10/01/08`을 반환합니다.

구문

```
CURRENT_DATE
```

반환 타입

날짜

예시

다음은 현재 날짜(함수가 실행되는 AWS 리전 내)를 반환하는 예입니다.

```
select current_date;

   date
------------
2008-10-01
```

다음 예제에서는 테이블을 만들고 `todays_date` 열의 기본값이 CURRENT_DATE인 행을 삽입한 다음 테이블의 모든 행을 선택합니다.

```
CREATE TABLE insert_dates(
    label varchar(128) NOT NULL,
    todays_date DATE DEFAULT CURRENT_DATE);

INSERT INTO insert_dates(label)
VALUES('Date row inserted');
```

```
SELECT * FROM insert_dates;


 label            | todays_date
------------------+-------------
Date row inserted | 2023-05-10
```

DATE_CMP 함수

DATE_CMP는 두 날짜를 서로 비교합니다. 이 함수는 날짜가 동일하면 0, date1이 더 크면 1, date2가 더 크면 -1을 반환합니다.

구문

```
DATE_CMP(date1, date2)
```

인수

date1

DATE 데이터 형식의 열 또는 DATE 형식으로 계산되는 표현식입니다.

date2

DATE 데이터 형식의 열 또는 DATE 형식으로 계산되는 표현식입니다.

반환 타입

INTEGER

예시

다음은 CALDATE 열의 DATE 값과 2008년 1월 4일을 서로 비교하여 CALDATE 열의 값이 2008년 1월 4일 이전인지(-1), 동일한지(0) 또는 이후인지(1) 결과를 반환하는 예입니다.

```
select caldate, '2008-01-04',
date_cmp(caldate,'2008-01-04')
from date
order by dateid
limit 10;

 caldate   |  ?column?  | date_cmp
```

```
-----------+------------+----------
2008-01-01 | 2008-01-04 |       -1
2008-01-02 | 2008-01-04 |       -1
2008-01-03 | 2008-01-04 |       -1
2008-01-04 | 2008-01-04 |        0
2008-01-05 | 2008-01-04 |        1
2008-01-06 | 2008-01-04 |        1
2008-01-07 | 2008-01-04 |        1
2008-01-08 | 2008-01-04 |        1
2008-01-09 | 2008-01-04 |        1
2008-01-10 | 2008-01-04 |        1
(10 rows)
```

DATE_CMP_TIMESTAMP 함수

DATE_CMP_TIMESTAMP는 날짜와 타임스탬프를 비교하여 값이 같으면 0을, date가 시간순으로 더 크면 1을, timestamp가 더 크면 -1을 반환합니다.

구문

```
DATE_CMP_TIMESTAMP(date, timestamp)
```

인수

date

DATE 데이터 형식의 열 또는 DATE 형식으로 계산되는 표현식입니다.

timestamp

TIMESTAMP 데이터 형식의 열 또는 TIMESTAMP 형식으로 계산되는 표현식입니다.

반환 타입

INTEGER

예시

다음은 날짜 2008-06-18을 LISTTIME 열과 비교하는 예입니다. LISTTIME 열의 값은 타임스탬프입니다. 이 날짜 이전의 목록은 1을, 그리고 이 날짜 이후의 목록은 -1을 반환합니다.

```
select listid, '2008-06-18', listtime,
```

```
date_cmp_timestamp('2008-06-18', listtime)
from listing
order by 1, 2, 3, 4
limit 10;

 listid |  ?column?  |      listtime       | date_cmp_timestamp
--------+------------+---------------------+--------------------
      1 | 2008-06-18 | 2008-01-24 06:43:29 |                  1
      2 | 2008-06-18 | 2008-03-05 12:25:29 |                  1
      3 | 2008-06-18 | 2008-11-01 07:35:33 |                 -1
      4 | 2008-06-18 | 2008-05-24 01:18:37 |                  1
      5 | 2008-06-18 | 2008-05-17 02:29:11 |                  1
      6 | 2008-06-18 | 2008-08-15 02:08:13 |                 -1
      7 | 2008-06-18 | 2008-11-15 09:38:15 |                 -1
      8 | 2008-06-18 | 2008-11-09 05:07:30 |                 -1
      9 | 2008-06-18 | 2008-09-09 08:03:36 |                 -1
     10 | 2008-06-18 | 2008-06-17 09:44:54 |                  1
(10 rows)
```

DATE_CMP_TIMESTAMPTZ 함수

DATE_CMP_TIMESTAMPTZ는 날짜를 타임스탬프와 표준 시간대를 비교하여 값이 같으면 0을, date가 시간순으로 더 크면 1을, timestamptz가 더 크면 -1을 반환합니다.

구문

```
DATE_CMP_TIMESTAMPTZ(date, timestamptz)
```

인수

date

DATE 데이터 형식의 열 또는 암시적으로 DATE 형식으로 평가되는 표현식입니다.

timestamptz

TIMESTAMPTZ 데이터 형식의 열 또는 암시적으로 TIMESTAMPTZ 형식으로 평가되는 표현식입니다.

반환 타입

INTEGER

예시

다음은 날짜 2008-06-18을 LISTTIME 열과 비교하는 예입니다. 이 날짜 이전의 목록은 1을, 그리고 이 날짜 이후의 목록은 -1을 반환합니다.

```
select listid, '2008-06-18', CAST(listtime AS timestamptz),
date_cmp_timestamptz('2008-06-18', CAST(listtime AS timestamptz))
from listing
order by 1, 2, 3, 4
limit 10;

 listid |  ?column?  |      timestamptz       | date_cmp_timestamptz
--------+------------+------------------------+----------------------
      1 | 2008-06-18 | 2008-01-24 06:43:29+00 |                    1
      2 | 2008-06-18 | 2008-03-05 12:25:29+00 |                    1
      3 | 2008-06-18 | 2008-11-01 07:35:33+00 |                   -1
      4 | 2008-06-18 | 2008-05-24 01:18:37+00 |                    1
      5 | 2008-06-18 | 2008-05-17 02:29:11+00 |                    1
      6 | 2008-06-18 | 2008-08-15 02:08:13+00 |                   -1
      7 | 2008-06-18 | 2008-11-15 09:38:15+00 |                   -1
      8 | 2008-06-18 | 2008-11-09 05:07:30+00 |                   -1
      9 | 2008-06-18 | 2008-09-09 08:03:36+00 |                   -1
     10 | 2008-06-18 | 2008-06-17 09:44:54+00 |                    1
(10 rows)
```

DATEADD 함수

DATE, TIME, TIMETZ 또는 TIMESTAMP 값을 지정된 간격만큼 늘립니다.

구문

```
DATEADD( datepart, interval, {date|time|timetz|timestamp} )
```

인수

datepart

함수가 작동하는 날짜 부분(예: 년, 월, 일 또는 시)입니다. 자세한 내용은 날짜 또는 타임스탬프 함수의 날짜 부분 단원을 참조하십시오.

interval

대상 표현식에 합산할 간격(일 수 등)을 지정한 정수입니다. 음의 정수는 간격을 감산합니다.

date|time|timetz|timestamp

DATE, TIME, TIMETZ 또는 TIMESTAMP 열 또는 암시적으로 DATE, TIME, TIMETZ 또는 TIMESTAMP로 변환하는 표현식입니다. DATE, TIME, TIMETZ 또는 TIMESTAMP 표현식에 지정한 날짜 부분이 포함되어야 합니다.

반환 타입

입력 데이터 형식에 따라 TIMESTAMP, TIME 또는 TIMETZ입니다.

DATE 열이 있는 예

다음은 DATE 테이블에 존재하는 11월 데이터에 각각 30일을 합산하는 예입니다.

```
select dateadd(day,30,caldate) as novplus30
from date
where month='NOV'
order by dateid;

novplus30
---------------------
2008-12-01 00:00:00
2008-12-02 00:00:00
2008-12-03 00:00:00
...
(30 rows)
```

다음 예에서는 리터럴 날짜 값에 18개월을 추가합니다.

```
select dateadd(month,18,'2008-02-28');

date_add
---------------------
2009-08-28 00:00:00
(1 row)
```

DATEADD 함수에 사용되는 기본 열 이름은 DATE_ADD입니다. 날짜 값을 나타내는 기본 타임스탬프는 `00:00:00`입니다.

다음 예에서는 타임스탬프를 지정하지 않는 날짜 값에 30분을 추가합니다.

```
select dateadd(m,30,'2008-02-28');
```

```
date_add
---------------------
2008-02-28 00:30:00
(1 row)
```

날짜 부분은 전체 이름으로 또는 약어로 지정할 수 있습니다. 이 경우 m은 월이 아닌 분을 나타냅니다.

TIME 열이 있는 예

다음 예제 테이블 TIME_TEST에는 3개의 값이 삽입된 TIME_VAL(TIME 형식) 열이 있습니다.

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

다음 예에서는 TIME_TEST 테이블의 각 TIME_VAL에 5분을 추가합니다.

```
select dateadd(minute,5,time_val) as minplus5 from time_test;

minplus5
---------------
20:05:00
00:05:00.5550
01:03:00
```

다음 예에서는 리터럴 시간 값에 8시간을 추가합니다.

```
select dateadd(hour, 8, time '13:24:55');

date_add
---------------
21:24:55
```

다음 예에서는 시간이 24:00:00을 초과하거나 00:00:00 미만인 경우를 보여줍니다.

```
select dateadd(hour, 12, time '13:24:55');
```

```
date_add
---------------
01:24:55
```

TIMETZ 열이 있는 예

이 예의 출력 값은 기본 표준 시간대인 UTC를 기준으로 합니다.

다음 예제 테이블 TIMETZ_TEST에는 3개의 값이 삽입된 TIMETZ_VAL(TIMETZ 형식) 열이 있습니다.

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

다음 예에서는 TIMETZ_TEST 테이블의 각 TIMETZ_VAL에 5분을 추가합니다.

```
select dateadd(minute,5,timetz_val) as minplus5_tz from timetz_test;

minplus5_tz
---------------
04:05:00+00
00:05:00.5550+00
06:03:00+00
```

다음 예에서는 리터럴 timetz 값에 2시간을 추가합니다.

```
select dateadd(hour, 2, timetz '13:24:55 PST');

date_add
---------------
23:24:55+00
```

TIMESTAMP 열이 있는 예시

이 예의 출력 값은 기본 표준 시간대인 UTC를 기준으로 합니다.

다음 예시 테이블 TIMESTAMP_TEST에는 3개의 값이 삽입된 TIMESTAMP_VAL(TIMESTAMP 형식) 열이 있습니다.

```
SELECT timestamp_val FROM timestamp_test;

timestamp_val
------------------
1988-05-15 10:23:31
2021-03-18 17:20:41
2023-06-02 18:11:12
```

다음 예시에서는 2000년 이전의 TIMESTAMP_TEST의 TIMESTAMP_VAL 값에만 20년을 더합니다.

```
SELECT dateadd(year,20,timestamp_val)
FROM timestamp_test
WHERE timestamp_val < to_timestamp('2000-01-01 00:00:00', 'YYYY-MM-DD HH:MI:SS');

date_add
---------------
2008-05-15 10:23:31
```

다음 예시에서는 초 표시기 없이 작성된 리터럴 타임스탬프 값에 5초를 추가합니다.

```
SELECT dateadd(second, 5, timestamp '2001-06-06');

date_add
---------------
2001-06-06 00:00:05
```

사용 노트

DATEADD(month, ...) 함수와 ADD_MONTHS 함수는 매월 마지막 날짜를 서로 다르게 처리합니다.

- ADD_MONTHS: 합산하려는 날짜가 월의 마지막 날짜인 경우에는 월의 길이에 상관없이 항상 결과에 따른 월의 마지막 날짜가 반환됩니다. 예를 들어 다음과 같이 4월 30일에 1개월을 합산하면 5월 31일이 됩니다.

  ```
  select add_months('2008-04-30',1);

  add_months
  ---------------------
  2008-05-31 00:00:00
  (1 row)
  ```

- DATEADD: 합산하려는 날짜의 일 수가 결과에 따른 월보다 적은 경우에는 결과에 따른 월의 마지막 날짜가 아니고 해당하는 날짜가 반환됩니다. 예를 들어 다음과 같이 4월 30일에 1개월을 합산하면 5월 30일이 됩니다.

```
select dateadd(month,1,'2008-04-30');

date_add
---------------------
2008-05-30 00:00:00
(1 row)
```

DATEADD 함수는 dateadd(month, 12,…)를 사용할 때와 dateadd(year, 1, …)을 사용할 때 윤년 날짜인 2월 29일을 각각 다르게 처리합니다.

```
select dateadd(month,12,'2016-02-29');

date_add
---------------------
2017-02-28 00:00:00

select dateadd(year, 1, '2016-02-29');

date_add
---------------------
2017-03-01 00:00:00
```

DATEDIFF 함수

DATEDIFF는 두 날짜 또는 시간 표현식에서 날짜 부분의 차이점을 반환합니다.

구문

```
DATEDIFF( datepart, {date|time|timetz|timestamp}, {date|time|timetz|timestamp} )
```

인수

datepart

함수가 작동하는 날짜 또는 시간 값의 특정 부분(년, 월 또는 일, 시, 분, 초, 밀리초 또는 마이크로초)입니다. 자세한 내용은 날짜 또는 타임스탬프 함수의 날짜 부분 단원을 참조하십시오.

특히 DATEDIFF는 두 표현식이 서로 교차하는 날짜 부분 경계의 수를 결정합니다. 예를 들어 두 날짜 12-31-2008과 01-01-2009 사이의 연도 차이를 계산한다고 가정합니다. 이 경우 함수는 두 날짜가 단 하루 차이임에도 불구하고 1년을 반환합니다. 하지만 두 타임스탬프인 01-01-2009 8:30:00과 01-01-2009 10:00:00 사이의 시간차를 구하는 경우에는 함수 결과로 2시간이 반환됩니다. 하지만 두 타임스탬프인 8:30:00과 10:00:00 사이의 시간차를 구하는 경우에는 함수 결과로 2시간이 반환됩니다.

date|time|timetz|timestamp

DATE, TIME, TIMETZ 또는 TIMESTAMP 열 또는 암시적으로 DATE, TIME, TIMETZ 또는 TIMESTAMP로 변환하는 표현식입니다. 표현식에는 지정하는 날짜 또는 시간 부분이 모두 포함되어야 합니다. 두 번째 날짜 또는 시간이 첫 번째 날짜 또는 시간보다 이후인 경우에는 결과 값이 양수입니다. 하지만 두 번째 날짜 또는 시간이 첫 번째 날짜 또는 시간보다 이전인 경우에는 결과 값이 음수입니다.

반환 타입

BIGINT

DATE 열이 있는 예

다음은 두 리터럴 날짜 값 사이의 차이 값(주 수)을 구하는 예입니다.

```
select datediff(week,'2009-01-01','2009-12-31') as numweeks;

numweeks
----------
52
(1 row)
```

다음은 두 리터럴 날짜 값 사이의 차이 값(시간)을 구하는 예입니다. 날짜의 시간 값을 제공하지 않는 경우 기본값은 00:00:00입니다.

```
select datediff(hour, '2023-01-01', '2023-01-03 05:04:03');

date_diff
----------
53
(1 row)
```

다음은 두 리터럴 TIMESTAMETZ 값 사이의 차이(일수)를 구하는 예시입니다.

```
Select datediff(days, 'Jun 1,2008  09:59:59 EST', 'Jul 4,2008  09:59:59 EST')

date_diff
----------
33
```

다음은 테이블의 동일한 행에 있는 두 날짜 사이의 차이 값(일)을 구하는 예입니다.

```
select * from date_table;

start_date |   end_date
-----------+-----------
2009-01-01 | 2009-03-23
2023-01-04 | 2024-05-04
(2 rows)

select datediff(day, start_date, end_date) as duration from date_table;

duration
---------
      81
     486
(2 rows)
```

다음은 이전 날짜와 오늘 날짜의 리터럴 값 사이에서 차이 값(분기 수)을 구하는 예입니다. 이번 예는 오늘 날짜가 2008년 6월 5일이라는 가정을 전제로 합니다. 날짜 부분은 전체 이름으로 또는 약어로 지정할 수 있습니다. DATEDIFF 함수에 사용되는 기본 열 이름은 DATE_DIFF입니다.

```
select datediff(qtr, '1998-07-01', current_date);

date_diff
-----------
40
(1 row)
```

다음은 SALES 테이블과 LISTING 테이블을 조인하여 두 테이블의 나열 이후 목록 1000부터 1005까지 티켓이 판매된 일수를 계산하는 예입니다. 두 목록에서 가장 길게 기다린 판매 일수는 15일이었고, 가장 짧은 기다린 판매 일수는 1일 미만이었습니다(0일).

```
select priceperticket,
datediff(day, listtime, saletime) as wait
```

```
from sales, listing where sales.listid = listing.listid
and sales.listid between 1000 and 1005
order by wait desc, priceperticket desc;

priceperticket | wait
---------------+------
 96.00         |   15
 123.00        |   11
 131.00        |    9
 123.00        |    6
 129.00        |    4
 96.00         |    4
 96.00         |    0
(7 rows)
```

다음은 판매자들이 모든 티켓이 판매될 때까지 기다린 평균 시간을 계산하는 예입니다.

```
select avg(datediff(hours, listtime, saletime)) as avgwait
from sales, listing
where sales.listid = listing.listid;

avgwait
---------
465
(1 row)
```

TIME 열이 있는 예

다음 예제 테이블 TIME_TEST에는 3개의 값이 삽입된 TIME_VAL(TIME 형식) 열이 있습니다.

```
select time_val from time_test;

time_val
--------------------
20:00:00
00:00:00.5550
00:58:00
```

다음 예에서는 TIME_VAL 열과 시간 리터럴 간의 시간 차이를 찾습니다.

```
select datediff(hour, time_val, time '15:24:45') from time_test;
```

```
 date_diff
-----------
        -5
        15
        15
```

다음 예에서는 두 리터럴 시간 값 간의 분 수 차이를 찾습니다.

```
select datediff(minute, time '20:00:00', time '21:00:00') as nummins;

nummins
----------
60
```

TIMETZ 열이 있는 예

다음 예제 테이블 TIMETZ_TEST에는 3개의 값이 삽입된 TIMETZ_VAL(TIMETZ 형식) 열이 있습니다.

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

다음 예에서는 TIMETZ 리터럴과 timetz_val 간의 시간 차이를 찾습니다.

```
select datediff(hours, timetz '20:00:00 PST', timetz_val) as numhours from timetz_test;

numhours
----------
0
-4
1
```

다음 예에서는 두 리터럴 TIMETZ 값 간의 시간 차이를 찾습니다.

```
select datediff(hours, timetz '20:00:00 PST', timetz '00:58:00 EST') as numhours;

numhours
----------
```

```
1
```

DATE_PART 함수

DATE_PART는 표현식에서 날짜 부분 값을 추출합니다. DATE_PART는 PGDATE_PART 함수의 동의어입니다.

구문

```
DATE_PART(datepart, {date|timestamp})
```

인수

datepart

날짜 값에서 함수가 실행되는 특정 부분(예: 년, 월 또는 일)의 식별자 리터럴 또는 문자열입니다. 자세한 내용은 날짜 또는 타임스탬프 함수의 날짜 부분 단원을 참조하십시오.

{date|timestamp}

날짜 열, 타임스탬프 열 또는 묵시적으로 날짜 또는 타임스탬프로 변환되는 표현식입니다. date 또는 timestamp의 열 또는 표현식에는 datepart에 지정된 날짜 부분이 포함되어야 합니다.

반환 타입

DOUBLE

예시

DATE_PART 함수에 사용되는 기본 열 이름은 pgdate_part입니다.

이러한 예 중 일부에 사용되는 데이터에 대한 자세한 내용은 샘플 데이터베이스 섹션을 참조하세요.

다음 예는 타임스탬프 리터럴에서 분을 찾습니다.

```
SELECT DATE_PART(minute, timestamp '20230104 04:05:06.789');

pgdate_part
-----------
          5
```

다음 예는 타임스탬프 리터럴에서 주 번호를 찾습니다. 주 번호 계산은 ISO 8601 표준을 따릅니다. 자세한 내용은 Wikipedia의 ISO 8601을 참조하세요.

```
SELECT DATE_PART(week, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
         18
```

다음 예는 타임스탬프 리터럴에서 날짜를 찾습니다.

```
SELECT DATE_PART(day, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
         2
```

다음 예는 타임스탬프 리터럴에서 요일을 찾습니다. 요일 번호 계산에는 일요일부터 시작하여 0~6의 정수가 사용됩니다.

```
SELECT DATE_PART(dayofweek, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
         1
```

다음 예는 타임스탬프 리터럴에서 세기를 찾습니다. 세기 계산은 ISO 8601 표준을 따릅니다. 자세한 내용은 Wikipedia의 ISO 8601을 참조하세요.

```
SELECT DATE_PART(century, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
         21
```

다음 예는 타임스탬프 리터럴에서 천 년 단위를 찾습니다. 천 년 단위 계산은 ISO 8601 표준을 따릅니다. 자세한 내용은 Wikipedia의 ISO 8601을 참조하세요.

```
SELECT DATE_PART(millennium, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
         3
```

다음 예는 타임스탬프 리터럴에서 마이크로초를 찾습니다. 마이크로초 계산은 ISO 8601 표준을 따릅니다. 자세한 내용은 Wikipedia의 ISO 8601을 참조하세요.

```
SELECT DATE_PART(microsecond, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
    789000
```

다음 예는 날짜 리터럴에서 월을 찾습니다.

```
SELECT DATE_PART(month, date '20220502');

pgdate_part
-----------
          5
```

다음 예에서는 테이블의 열에 DATE_PART 함수를 적용합니다.

```
SELECT date_part(w, listtime) AS weeks, listtime
FROM listing
WHERE listid=10


weeks |      listtime
------+---------------------
 25   | 2008-06-17 09:44:54
(1 row)
```

날짜 부분은 전체 이름 또는 약어로 지정할 수 있으며, 여기서 w는 주를 의미합니다.

요일을 나타내는 날짜 부분은 일요일부터 시작하여 0~6의 정수를 반환합니다. DATE_PART를 dow(DAYOFWEEK)와 함께 사용하면 일요일의 이벤트를 볼 수 있습니다.

```
SELECT date_part(dow, starttime) AS dow, starttime
FROM event
WHERE date_part(dow, starttime)=6
ORDER BY 2,1;

 dow |      starttime
-----+---------------------
   6 | 2008-01-05 14:00:00
```

```
   6 | 2008-01-05 14:00:00
   6 | 2008-01-05 14:00:00
   6 | 2008-01-05 14:00:00
...
(1147 rows)
```

DATE_PART_YEAR 함수

DATE_PART_YEAR 함수는 날짜에서 연도를 추출합니다.

구문

```
DATE_PART_YEAR(date)
```

인수

date

DATE 데이터 형식의 열 또는 암시적으로 DATE 형식으로 평가되는 표현식입니다.

반환 타입

INTEGER

예시

다음 예는 날짜 리터럴에서 연도를 찾습니다.

```
SELECT DATE_PART_YEAR(date '20220502 04:05:06.789');

date_part_year
---------------
2022
```

다음은 CALDATE 열에서 연도를 추출하는 예입니다. CALDATE 열의 값은 날짜입니다. 이 예에 사용되는 데이터에 대한 자세한 내용은 샘플 데이터베이스 섹션을 참조하세요.

```
select caldate, date_part_year(caldate)
from date
order by
dateid limit 10;
```

```
 caldate   | date_part_year
-----------+----------------
2008-01-01 |           2008
2008-01-02 |           2008
2008-01-03 |           2008
2008-01-04 |           2008
2008-01-05 |           2008
2008-01-06 |           2008
2008-01-07 |           2008
2008-01-08 |           2008
2008-01-09 |           2008
2008-01-10 |           2008
(10 rows)
```

DATE_TRUNC 함수

DATE_TRUNC 함수는 시간, 일 또는 월 등 지정하는 날짜 부분을 기준으로 타임스탬프 표현식 또는 리터럴을 자릅니다.

구문

```
DATE_TRUNC('datepart', timestamp)
```

인수

datepart

타임스탬프 값을 자를 때 기준이 되는 날짜 부분입니다. 입력 timestamp(타임스탬프)는 입력 datepart의 정밀도로 잘립니다. 예를 들어, `month`로 설정하면 해당 달의 첫 번째 날로 잘립니다. 유효한 형식은 다음과 같습니다.

- microsecond, microseconds
- millisecond, milliseconds
- second, seconds
- minute, minutes
- hour, hours
- day, days
- week, weeks
- month, months
- quarter, quarters

- year, years
- decade, decades
- century, centuries
- millennium, millennia

일부 형식의 약어에 대한 자세한 내용은 날짜 또는 타임스탬프 함수의 날짜 부분 섹션을 참조하세요.

timestamp

타임스탬프 열 또는 묵시적으로 타임스탬프로 변환되는 표현식입니다.

반환 타입

TIMESTAMP

예시

입력 타임스탬프를 초로 자릅니다.

```
SELECT DATE_TRUNC('second', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-30 04:05:06
```

입력 타임스탬프를 분으로 자릅니다.

```
SELECT DATE_TRUNC('minute', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-30 04:05:00
```

입력 타임스탬프를 시간으로 자릅니다.

```
SELECT DATE_TRUNC('hour', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-30 04:00:00
```

입력 타임스탬프를 일로 자릅니다.

```
SELECT DATE_TRUNC('day', TIMESTAMP '20200430 04:05:06.789');
date_trunc
```

```
2020-04-30 00:00:00
```

입력 타임스탬프를 해당 월의 첫 번째 날로 자릅니다.

```
SELECT DATE_TRUNC('month', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-01 00:00:00
```

입력 타임스탬프를 해당 분기의 첫 번째 날로 자릅니다.

```
SELECT DATE_TRUNC('quarter', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-01 00:00:00
```

입력 타임스탬프를 해당 연도의 첫 번째 날로 자릅니다.

```
SELECT DATE_TRUNC('year', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-01-01 00:00:00
```

입력 타임스탬프를 해당 세기의 첫 번째 날로 자릅니다.

```
SELECT DATE_TRUNC('millennium', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2001-01-01 00:00:00
```

입력 타임스탬프를 해당 주의 월요일로 자릅니다.

```
select date_trunc('week', TIMESTAMP '20220430 04:05:06.789');
date_trunc
2022-04-25 00:00:00
```

다음은 DATE_TRUNC 함수에서 '주' 날짜 부분을 사용하여 각 주의 월요일에 해당하는 날짜를 반환하는 예입니다.

```
select date_trunc('week', saletime), sum(pricepaid) from sales where
saletime like '2008-09%' group by date_trunc('week', saletime) order by 1;

date_trunc  |    sum
------------+-------------
```

```
2008-09-01  | 2474899
2008-09-08  | 2412354
2008-09-15  | 2364707
2008-09-22  | 2359351
2008-09-29  |  705249
```

EXTRACT 함수

EXTRACT 함수는 TIMESTAMP, TIMESTAMPTZ, TIME, TIMETZ, INTERVAL YEAR TO MONTH 또는 INTERVAL DAY TO SECOND 값에서 날짜 또는 시간 부분을 반환합니다. 예에는 타임스탬프의 일, 월, 년, 시, 분, 초, 밀리초 또는 마이크로초가 포함됩니다.

구문

```
EXTRACT(datepart FROM source)
```

인수

datepart

일, 월, 년, 시, 분, 초, 밀리초 또는 마이크로초 등 추출할 날짜 또는 시간의 하위 필드입니다. 에 대해 가능한 값은 날짜 또는 타임스탬프 함수의 날짜 부분 섹션을 참조하세요.

source

TIMESTAMP, TIMESTAMPTZ, TIME, TIMETZ, INTERVAL YEAR TO MONTH 또는 INTERVAL DAY TO SECOND의 데이터 유형으로 평가되는 열 또는 표현식입니다.

반환 타입

source 값이 TIMESTAMP, TIME, TIMETZ, INTERVAL YEAR TO MONTH 또는 INTERVAL DAY TO SECOND의 데이터 유형으로 평가되는 경우 INTEGER입니다.

source 값이 TIMESTAMPTZ의 데이터 유형으로 평가되는 경우 DOUBLE PRECISION입니다.

TIMESTAMP를 사용한 예제

다음은 판매에서 지불 가격이 $10,000 이상이었던 주차 번호(week numbers)를 확인하는 예입니다. 이 예제에서는 TICKIT 데이터를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

```
select salesid, extract(week from saletime) as weeknum
from sales
```

```
where pricepaid > 9999
order by 2;

salesid | weeknum
--------+---------
 159073 |       6
 160318 |       8
 161723 |      26
```

다음 예에서는 리터럴 타임스탬프 값에서 분 값을 반환합니다.

```
select extract(minute from timestamp '2009-09-09 12:08:43');

date_part
-----------
8
```

다음 예제에서는 리터럴 timestamp 값에서 밀리초 값을 반환합니다.

```
select extract(ms from timestamp '2009-09-09 12:08:43.101');

date_part
-----------
101
```

TIMESTAMPTZ를 사용한 예제

다음 예제에서는 리터럴 timestamptz 값에서 년 값을 반환합니다.

```
select extract(year from timestamptz '1.12.1997 07:37:16.00 PST');

date_part
-----------
1997
```

TIME을 사용한 예제

다음 예제 테이블 TIME_TEST에는 3개의 값이 삽입된 TIME_VAL(TIME 형식) 열이 있습니다.

```
select time_val from time_test;

time_val
```

```
---------------------
20:00:00
00:00:00.5550
00:58:00
```

다음 예에서는 각 time_val에서 분을 추출합니다.

```
select extract(minute from time_val) as minutes from time_test;

minutes
-----------
         0
         0
         58
```

다음 예에서는 각 time_val에서 시간을 추출합니다.

```
select extract(hour from time_val) as hours from time_test;

hours
-----------
         20
         0
         0
```

다음 예에서는 리터럴 값에서 밀리초를 추출합니다.

```
select extract(ms from time '18:25:33.123456');

 date_part
-----------
     123
```

TIMETZ를 사용한 예제

다음 예제 테이블 TIMETZ_TEST에는 3개의 값이 삽입된 TIMETZ_VAL(TIMETZ 형식) 열이 있습니다.

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
```

```
00:00:00.5550+00
05:58:00+00
```

다음 예에서는 각 timetz_val에서 시간을 추출합니다.

```
select extract(hour from timetz_val) as hours from time_test;

hours
-----------
         4
         0
         5
```

다음 예에서는 리터럴 값에서 밀리초를 추출합니다. 추출이 처리되기 전에 리터럴은 UTC로 변환되지 않습니다.

```
select extract(ms from timetz '18:25:33.123456 EST');

 date_part
-----------
     123
```

다음 예제에서는 리터럴 timetz 값에서 UTC로부터의 시간대 오프셋 시간을 반환합니다.

```
select extract(timezone_hour from timetz '1.12.1997 07:37:16.00 PDT');

date_part
-----------
-7
```

INTERVAL YEAR TO MONTH 및 INTERVAL DAY TO SECOND 예제

다음 예제에서는 36시간, 즉 1일 12시간을 정의하는 INTERVAL DAY TO SECOND에서 일 부분인 1을 추출합니다.

```
select EXTRACT('days' from INTERVAL '36 hours' DAY TO SECOND)

 date_part
------------------
 1
```

다음 예제에서는 15개월, 즉 1년 3개월을 정의하는 YEAR TO MONTH에서 월 부분인 3을 추출합니다.

```
select EXTRACT('month' from INTERVAL '15 months' YEAR TO MONTH)

 date_part
------------------
 3
```

다음 예제에서는 30개월, 즉 2년 6개월에서 월 부분인 6을 추출합니다.

```
select EXTRACT('month' from INTERVAL '30' MONTH)

 date_part
------------------
 6
```

다음 예제에서는 50시간, 즉 2일 2시간에서 시간 부분인 2를 추출합니다.

```
select EXTRACT('hours' from INTERVAL '50' HOUR)

 date_part
------------------
 2
```

다음 예제에서는 1시간 11분 11.123초에서 분 부분인 11을 추출합니다.

```
select EXTRACT('minute' from INTERVAL '70 minutes 70.123 seconds' MINUTE TO SECOND)

 date_part
------------------
 11
```

다음 예제에서는 1일 1시간 1분 1.11초에서 초 부분인 1.11을 추출합니다.

```
select EXTRACT('seconds' from INTERVAL '1 day 1:1:1.11' DAY TO SECOND)

 date_part
------------------
 1.11
```

다음 예제에서는 INTERVAL의 총 시간 수를 추출합니다. 각 부분이 추출되어 합계에 더해집니다.

```
select EXTRACT('days' from INTERVAL '50' HOUR) * 24 + EXTRACT('hours' from INTERVAL
 '50' HOUR)

 ?column?
------------------
 50
```

다음 예제에서는 INTERVAL의 총 초 수를 추출합니다. 각 부분이 추출되어 합계에 더해집니다.

```
select EXTRACT('days' from INTERVAL '1 day 1:1:1.11' DAY TO SECOND) * 86400 +
       EXTRACT('hours' from INTERVAL '1 day 1:1:1.11' DAY TO SECOND) * 3600 +
       EXTRACT('minutes' from INTERVAL '1 day 1:1:1.11' DAY TO SECOND) * 60 +
       EXTRACT('seconds' from INTERVAL '1 day 1:1:1.11' DAY TO SECOND)

 ?column?
------------------
 90061.11
```

GETDATE 함수

GETDATE는 현재 세션 시간대(기본 UTC)의 현재 날짜 및 시간을 반환합니다. 트랜잭션 블록 내에 있는 경우에도 현재 문의 시작 날짜 또는 시간을 반환합니다.

구문

```
GETDATE()
```

괄호가 필요합니다.

반환 타입

TIMESTAMP

예시

다음은 GETDATE 함수를 사용하여 현재 날짜의 전체 타임스탬프를 반환하는 예입니다.

```
select getdate();

timestamp
---------------------
2008-12-04 16:10:43
```

다음은 TRUNC 함수 내에서 GETDATE 함수를 사용하여 시간을 제외하고 현재 날짜를 반환하는 예입니다.

```
select trunc(getdate());

trunc
------------
2008-12-04
```

INTERVAL_CMP 함수

INTERVAL_CMP는 두 간격을 서로 비교하여 첫 번째 간격이 더 큰 경우에는 1을, 두 번째 간격이 더 큰 경우에는 -1을, 그리고 두 간격이 동일한 경우에는 0을 반환합니다. 자세한 내용은 한정자 구문이 없는 간격 리터럴의 예 단원을 참조하십시오.

구문

```
INTERVAL_CMP(interval1, interval2)
```

인수

INTERVAL

간격 리터럴 값입니다.

INTERVAL

간격 리터럴 값입니다.

반환 타입

INTEGER

예시

다음은 3 days의 값을 1 year의 값과 서로 비교하는 예입니다.

```
select interval_cmp('3 days','1 year');

interval_cmp
--------------
-1
```

이 예에서는 값 7 days와 1 week를 비교합니다.

```
select interval_cmp('7 days','1 week');

interval_cmp
--------------
0
```

다음은 1 year의 값을 3 days의 값과 서로 비교하는 예입니다.

```
select interval_cmp('1 year','3 days');

interval_cmp
--------------
1
```

LAST_DAY 함수

LAST_DAY는 date가 포함된 월의 마지막 날짜를 반환합니다. 반환 형식은 date 인수의 데이터 형식과 상관없이 항상 DATE입니다.

특정 날짜 부분을 검색하는 방법에 대한 자세한 내용은 DATE_TRUNC 함수 섹션을 참조하세요.

구문

```
LAST_DAY( { date | timestamp } )
```

인수

date | timestamp

DATE 또는 TIMESTAMP 데이터 형식의 DATE 또는 TIMESTAMP 형식으로 암시적으로 계산되는 표현식입니다.

반환 타입

날짜

예시

다음은 당월의 마지막 날짜를 반환하는 예입니다.

```
select last_day(sysdate);
```

```
  last_day
------------
 2014-01-31
```

다음은 당월 마지막 7일 중 매일 판매된 티켓 수를 반환하는 예입니다. SALETIME 열의 값은 타임스탬프입니다.

```
select datediff(day, saletime, last_day(saletime)) as "Days Remaining", sum(qtysold)
from sales
where datediff(day, saletime, last_day(saletime)) < 7
group by 1
order by 1;

days remaining |  sum
---------------+-------
             0 | 10140
             1 | 11187
             2 | 11515
             3 | 11217
             4 | 11446
             5 | 11708
             6 | 10988
(7 rows)
```

MONTHS_BETWEEN 함수

MONTHS_BETWEEN은 두 날짜 사이의 월 수를 계산합니다.

첫 번째 날짜가 두 번째 날짜 이후인 경우에는 결과 값이 양수이고, 그렇지 않으면 결과 값이 음수입니다.

두 인수 중 하나라도 NULL이면 결과 값도 NULL입니다.

구문

```
MONTHS_BETWEEN( date1, date2 )
```

인수

date1

DATE 데이터 형식의 열 또는 암시적으로 DATE 형식으로 평가되는 표현식입니다.

date2

DATE 데이터 형식의 열 또는 암시적으로 DATE 형식으로 평가되는 표현식입니다.

반환 타입

FLOAT8

결과 값에서 정수 부는 날짜의 년 값과 월 값의 차이에 따라 달라집니다. 결과 값에서 소수 부는 날짜 및 타임스탬프 값을 기준으로 계산되며, 한 달이 31일이라는 가정을 전제로 합니다.

date1과 date2 모두 월에서 동일한 날짜가 포함되어 있는 경우(예: 1/15/14와 2/15/14), 혹은 월에서 마지막 날짜가 포함되어 있는 경우(예: 8/31/14와 9/30/14)에는 타임스탬프 구간(있는 경우)의 일치 여부와 상관없이 날짜의 년 및 월 값에 따라 결과 값으로 정수가 반환됩니다.

예시

다음은 1/18/1969와 3/18/1969 사이의 월 수를 반환하는 예입니다.

```
select months_between('1969-01-18', '1969-03-18')
as months;

months
----------
-2
```

다음은 1/18/1969와 1/18/1969 사이의 월 수를 반환하는 예입니다.

```
select months_between('1969-01-18', '1969-01-18')
as months;

months
----------
0
```

다음은 첫 번째 이벤트 공연 시점부터 마지막 공연 시점까지 월 수를 반환하는 예입니다.

```
select eventname,
min(starttime) as first_show,
max(starttime) as last_show,
months_between(max(starttime),min(starttime)) as month_diff
from event
```

```
group by eventname
order by eventname
limit 5;

eventname         first_show              last_show               month_diff
-------------------------------------------------------------------------
.38 Special       2008-01-21 19:30:00.0  2008-12-25 15:00:00.0  11.12
3 Doors Down      2008-01-03 15:00:00.0  2008-12-01 19:30:00.0  10.94
70s Soul Jam      2008-01-16 19:30:00.0  2008-12-07 14:00:00.0  10.7
A Bronx Tale      2008-01-21 19:00:00.0  2008-12-15 15:00:00.0  10.8
A Catered Affair  2008-01-08 19:30:00.0  2008-12-19 19:00:00.0  11.35
```

NEXT_DAY 함수

NEXT_DAY는 지정한 날짜 이후 지정한 요일이 처음 도래하는 날짜를 반환합니다.

day 값이 주어진 날짜와 동일한 날짜인 경우에는 해당 요일의 다음 발생 날짜가 반환됩니다.

구문

```
NEXT_DAY( { date | timestamp }, day )
```

인수

date | timestamp

DATE 또는 TIMESTAMP 데이터 형식의 DATE 또는 TIMESTAMP 형식으로 암시적으로 계산되는 표현식입니다.

day

요일 이름을 포함한 문자열입니다. 대소문자는 구분하지 않습니다.

유효한 값은 다음과 같습니다.

일	값
일요일	Su, Sun, Sunday
월요일	M, Mo, Mon, Monday
화요일	Tu, Tue, Tues, Tuesday

일	값
수요일	W, We, Wed, Wednesday
목요일	Th, Thu, Thurs, Thursday
금요일	F, Fr, Fri, Friday
토요일	Sa, Sat, Saturday

반환 타입

날짜

예시

다음은 8/20/2014 이후 첫 번째 화요일의 날짜를 반환하는 예입니다.

```
select next_day('2014-08-20','Tuesday');

next_day
-----------
2014-08-26
```

다음 예에서는 2008년 1월 1일 이후 첫 번째 화요일의 날짜를 5:54:44로 반환합니다.

```
select listtime, next_day(listtime, 'Tue') from listing limit 1;

listtime            | next_day
--------------------+-----------
2008-01-01 05:54:44 | 2008-01-08
```

다음은 3분기 목표 마케팅 날짜를 가져오는 예입니다.

```
select username, (firstname ||' '|| lastname) as name,
eventname, caldate, next_day (caldate, 'Monday') as marketing_target
from sales, date, users, event
where sales.buyerid = users.userid
and sales.eventid = event.eventid
and event.dateid = date.dateid
and date.qtr = 3
```

```
order by marketing_target, eventname, name;

username  |     name          |     eventname        |    caldate   |
 marketing_target
----------+-------------------+----------------------+--------------
+------------------
MBO26QSG  |   Callum Atkinson | .38 Special          |  2008-07-06  | 2008-07-07
WCR50YIU  |   Erasmus Alvarez | A Doll's House       |  2008-07-03  | 2008-07-07
CKT70OIE  |   Hadassah Adkins | Ana Gabriel          |  2008-07-06  | 2008-07-07
VVG07OUO  |   Nathan Abbott   | Armando Manzanero    |  2008-07-04  | 2008-07-07
GEW77SII  |   Scarlet Avila   | August: Osage County |  2008-07-06  | 2008-07-07
ECR71CVS  |   Caryn Adkins    | Ben Folds            |  2008-07-03  | 2008-07-07
KUW82CYU  |   Kaden Aguilar   | Bette Midler         |  2008-07-01  | 2008-07-07
WZE78DJZ  |   Kay Avila       | Bette Midler         |  2008-07-01  | 2008-07-07
HXY04NVE  |   Dante Austin    | Britney Spears       |  2008-07-02  | 2008-07-07
URY81YWF  |   Wilma Anthony   | Britney Spears       |  2008-07-02  | 2008-07-07
```

SYSDATE 함수

SYSDATE는 현재 세션 시간대(기본 UTC)의 현재 날짜 및 시간을 반환합니다.

> **Note**
>
> SYSDATE는 현재 문이 아닌 현재 트랜잭션의 시작 날짜 및 시간을 반환합니다.

구문

```
SYSDATE
```

이 함수는 인수가 필요 없습니다.

반환 타입

TIMESTAMP

예시

다음은 SYSDATE 함수를 사용하여 현재 날짜의 전체 타임스탬프를 반환하는 예입니다.

```
select sysdate;
```

```
timestamp
----------------------------
2008-12-04 16:10:43.976353
```

다음은 TRUNC 함수 내에서 SYSDATE 함수를 사용하여 시간을 제외하고 현재 날짜를 반환하는 예입니다.

```
select trunc(sysdate);

trunc
------------
2008-12-04
```

다음은 쿼리 실행 날짜와 어떤 날짜든 120일 이전 사이에 해당하는 날짜의 판매 정보를 반환하는 쿼리입니다.

```
select salesid, pricepaid, trunc(saletime) as saletime, trunc(sysdate) as now
from sales
where saletime between trunc(sysdate)-120 and trunc(sysdate)
order by saletime asc;

 salesid | pricepaid |  saletime  |    now
---------+-----------+------------+------------
91535    |    670.00 | 2008-08-07 | 2008-12-05
91635    |    365.00 | 2008-08-07 | 2008-12-05
91901    |   1002.00 | 2008-08-07 | 2008-12-05
...
```

TIMEOFDAY 함수

TIMEOFDAY는 평일, 날짜 및 시간을 문자열 값으로 반환하는 데 사용되는 특수 별칭입니다. 트랜잭션 블록 내에 있는 경우에도 현재 문의 시간대 문자열을 반환합니다.

구문

```
TIMEOFDAY()
```

반환 타입

VARCHAR

예시

다음은 TIMEOFDAY 함수를 사용하여 현재 날짜와 시간을 반환하는 예입니다.

```
select timeofday();

timeofday
------------
Thu Sep 19 22:53:50.333525 2013 UTC
```

TIMESTAMP_CMP 함수

타임스탬프 2개의 값을 서로 비교한 후 정수를 반환합니다. 타임스탬프가 동일하면 함수가 0을 반환합니다. 첫 번째 타임스탬프가 더 크면 함수가 1을 반환합니다. 반대로 두 번째 타임스탬프가 더 크면 함수가 -1을 반환합니다.

구문

```
TIMESTAMP_CMP(timestamp1, timestamp2)
```

인수

timestamp1

TIMESTAMP 데이터 형식의 열 또는 암시적으로 TIMESTAMP 형식으로 평가되는 표현식입니다.

timestamp2

TIMESTAMP 데이터 형식의 열 또는 암시적으로 TIMESTAMP 형식으로 평가되는 표현식입니다.

반환 타입

INTEGER

예시

다음 예시는 타임스탬프를 비교하고 비교 결과를 보여줍니다.

```
SELECT TIMESTAMP_CMP('2008-01-24 06:43:29', '2008-01-24 06:43:29'),
 TIMESTAMP_CMP('2008-01-24 06:43:29', '2008-02-18 02:36:48'), TIMESTAMP_CMP('2008-02-18
 02:36:48', '2008-01-24 06:43:29');
```

```
timestamp_cmp  | timestamp_cmp | timestamp_cmp
---------------+---------------+---------------
             0 |            -1 |             1
```

다음은 LISTTIME 열과 SALETIME 열을 목록별로 비교하는 예입니다. 예를 보면 판매 타임스탬프가 목록 타임스탬프 이후이기 때문에 TIMESTAMP_CMP 값이 -1임을 알 수 있습니다.

```
select listing.listid, listing.listtime,
sales.saletime, timestamp_cmp(listing.listtime, sales.saletime)
from listing, sales
where listing.listid=sales.listid
order by 1, 2, 3, 4
limit 10;

 listid |      listtime       |      saletime       | timestamp_cmp
--------+---------------------+---------------------+---------------
      1 | 2008-01-24 06:43:29 | 2008-02-18 02:36:48 |            -1
      4 | 2008-05-24 01:18:37 | 2008-06-06 05:00:16 |            -1
      5 | 2008-05-17 02:29:11 | 2008-06-06 08:26:17 |            -1
      5 | 2008-05-17 02:29:11 | 2008-06-09 08:38:52 |            -1
      6 | 2008-08-15 02:08:13 | 2008-08-31 09:17:02 |            -1
     10 | 2008-06-17 09:44:54 | 2008-06-26 12:56:06 |            -1
     10 | 2008-06-17 09:44:54 | 2008-07-10 02:12:36 |            -1
     10 | 2008-06-17 09:44:54 | 2008-07-16 11:59:24 |            -1
     10 | 2008-06-17 09:44:54 | 2008-07-22 02:23:17 |            -1
     12 | 2008-07-25 01:45:49 | 2008-08-04 03:06:36 |            -1
(10 rows)
```

다음은 타임스탬프가 동일할 때 TIMESTAMP_CMP가 0을 반환하는 예입니다.

```
select listid, timestamp_cmp(listtime, listtime)
from listing
order by 1 , 2
limit 10;

 listid | timestamp_cmp
--------+---------------
      1 |             0
      2 |             0
      3 |             0
      4 |             0
      5 |             0
```

```
      6 |             0
      7 |             0
      8 |             0
      9 |             0
     10 |             0
(10 rows)
```

TIMESTAMP_CMP_DATE 함수

TIMESTAMP_CMP_DATE는 타임스탬프 값과 날짜 값을 서로 비교합니다. 타임스탬프 값과 날짜 값이 동일하면 함수가 0을 반환합니다. 타임스탬프가 시간순으로 더 큰 경우 함수가 1을 반환합니다. 날짜가 더 크면 함수가 -1을 반환합니다.

구문

```
TIMESTAMP_CMP_DATE(timestamp, date)
```

인수

timestamp

TIMESTAMP 데이터 형식의 열 또는 암시적으로 TIMESTAMP 형식으로 평가되는 표현식입니다.

date

DATE 데이터 형식의 열 또는 암시적으로 DATE 형식으로 평가되는 표현식입니다.

반환 타입

INTEGER

예시

다음은 LISTTIME을 날짜 2008-06-18과 비교하는 예입니다. 이 날짜 이후의 목록은 1을, 그리고 이 날짜 이전의 목록은 -1을 반환합니다. LISTTIME 값은 타임스탬프입니다.

```
select listid, listtime,
timestamp_cmp_date(listtime, '2008-06-18')
from listing
order by 1, 2, 3
limit 10;
```

```
 listid |      listtime       | timestamp_cmp_date
--------+---------------------+--------------------
      1 | 2008-01-24 06:43:29 |                 -1
      2 | 2008-03-05 12:25:29 |                 -1
      3 | 2008-11-01 07:35:33 |                  1
      4 | 2008-05-24 01:18:37 |                 -1
      5 | 2008-05-17 02:29:11 |                 -1
      6 | 2008-08-15 02:08:13 |                  1
      7 | 2008-11-15 09:38:15 |                  1
      8 | 2008-11-09 05:07:30 |                  1
      9 | 2008-09-09 08:03:36 |                  1
     10 | 2008-06-17 09:44:54 |                 -1
(10 rows)
```

TIMESTAMP_CMP_TIMESTAMPTZ 함수

TIMESTAMP_CMP_TIMESTAMPTZ는 타임스탬프 표현식 값과 시간대를 포함한 타임스탬프 표현식을 서로 비교합니다. 타임스탬프와 시간대를 포함한 타임스탬프가 동일하면 함수가 0을 반환합니다. 타임스탬프가 시간순으로 더 큰 경우 함수가 1을 반환합니다. 시간대를 포함한 타임스탬프가 더 크면 함수가 -1을 반환합니다.

구문

```
TIMESTAMP_CMP_TIMESTAMPTZ(timestamp, timestamptz)
```

인수

timestamp

TIMESTAMP 데이터 형식의 열 또는 암시적으로 TIMESTAMP 형식으로 평가되는 표현식입니다.

timestamptz

TIMESTAMPTZ 데이터 형식의 열 또는 암시적으로 TIMESTAMPTZ 형식으로 평가되는 표현식입니다.

반환 타입

INTEGER

예시

다음 예에서는 타임스탬프와 표준 시간대가 있는 타임스탬프를 비교하고 그 결과를 보여줍니다.

```
SELECT TIMESTAMP_CMP_TIMESTAMPTZ('2008-01-24 06:43:29', '2008-01-24 06:43:29+00'),
 TIMESTAMP_CMP_TIMESTAMPTZ('2008-01-24 06:43:29', '2008-02-18 02:36:48+00'),
 TIMESTAMP_CMP_TIMESTAMPTZ('2008-02-18 02:36:48', '2008-01-24 06:43:29+00');

timestamp_cmp_timestamptz  | timestamp_cmp_timestamptz | timestamp_cmp_timestamptz
---------------------------+---------------------------+--------------------------
             0             |            -1             |             1
```

TIMESTAMPTZ_CMP 함수

TIMESTAMPTZ_CMP는 시간대를 포함한 타임스탬프 값 2개를 서로 비교한 후 정수를 반환합니다. 타임스탬프가 동일하면 함수가 0을 반환합니다. 첫 번째 타임스탬프가 시간순으로 더 큰 경우 함수가 1을 반환합니다. 반대로 두 번째 타임스탬프가 더 크면 함수가 -1을 반환합니다.

구문

```
TIMESTAMPTZ_CMP(timestamptz1, timestamptz2)
```

인수

timestamptz1

TIMESTAMPTZ 데이터 형식의 열 또는 암시적으로 TIMESTAMPTZ 형식으로 평가되는 표현식입니다.

timestamptz2

TIMESTAMPTZ 데이터 형식의 열 또는 암시적으로 TIMESTAMPTZ 형식으로 평가되는 표현식입니다.

반환 타입

INTEGER

예시

다음 예제에서는 타임스탬프와 표준 시간대를 비교하고 그 결과를 보여 줍니다.

```
SELECT TIMESTAMPTZ_CMP('2008-01-24 06:43:29+00', '2008-01-24 06:43:29+00'),
 TIMESTAMPTZ_CMP('2008-01-24 06:43:29+00', '2008-02-18 02:36:48+00'),
 TIMESTAMPTZ_CMP('2008-02-18 02:36:48+00', '2008-01-24 06:43:29+00');

timestamptz_cmp  | timestamptz_cmp | timestamptz_cmp
-----------------+-----------------+----------------
        0        |       -1        |       1
```

TIMESTAMPTZ_CMP_DATE 함수

TIMESTAMPTZ_CMP_DATE는 타임스탬프 값과 날짜 값을 서로 비교합니다. 타임스탬프 값과 날짜 값이 동일하면 함수가 0을 반환합니다. 타임스탬프가 시간순으로 더 큰 경우 함수가 1을 반환합니다. 날짜가 더 크면 함수가 -1을 반환합니다.

구문

```
TIMESTAMPTZ_CMP_DATE(timestamptz, date)
```

인수

timestamptz

TIMESTAMPTZ 데이터 형식의 열 또는 암시적으로 TIMESTAMPTZ 형식으로 평가되는 표현식입니다.

date

DATE 데이터 형식의 열 또는 암시적으로 DATE 형식으로 평가되는 표현식입니다.

반환 타입

INTEGER

예시

다음 예는 타임스탬프와 표준 시간대가 포함된 타임스탬프인 LISTTIME을 날짜 2008-06-18과 비교하는 예입니다. 이 날짜 이후의 목록은 1을, 그리고 이 날짜 이전의 목록은 -1을 반환합니다.

```
select listid, CAST(listtime as timestamptz) as tstz,
timestamp_cmp_date(tstz, '2008-06-18')
from listing
order by 1, 2, 3
```

```
limit 10;


 listid |          tstz          | timestamptz_cmp_date
--------+------------------------+----------------------
      1 | 2008-01-24 06:43:29+00 |                   -1
      2 | 2008-03-05 12:25:29+00 |                   -1
      3 | 2008-11-01 07:35:33+00 |                    1
      4 | 2008-05-24 01:18:37+00 |                   -1
      5 | 2008-05-17 02:29:11+00 |                   -1
      6 | 2008-08-15 02:08:13+00 |                    1
      7 | 2008-11-15 09:38:15+00 |                    1
      8 | 2008-11-09 05:07:30+00 |                    1
      9 | 2008-09-09 08:03:36+00 |                    1
     10 | 2008-06-17 09:44:54+00 |                   -1
(10 rows)
```

TIMESTAMPTZ_CMP_TIMESTAMP 함수

TIMESTAMPTZ_CMP_TIMESTAMP는 시간대를 포함한 타임스탬프 표현식 값과 타임스탬프 표현식을 서로 비교합니다. 시간대를 포함한 타임스탬프 값과 타임스탬프 값이 동일하면 함수가 0을 반환합니다. 시간대를 포함한 타임스탬프가 시간순으로 더 크면 함수가 1을 반환합니다. 반대로 타임스탬프가 더 크면 함수가 -1을 반환합니다.

구문

```
TIMESTAMPTZ_CMP_TIMESTAMP(timestamptz, timestamp)
```

인수

timestamptz

TIMESTAMPTZ 데이터 형식의 열 또는 암시적으로 TIMESTAMPTZ 형식으로 평가되는 표현식입니다.

timestamp

TIMESTAMP 데이터 형식의 열 또는 암시적으로 TIMESTAMP 형식으로 평가되는 표현식입니다.

반환 타입

INTEGER

예시

다음 예제에서는 타임스탬프와 표준 시간대를 타임스탬프와 비교하고 그 결과를 보여 줍니다.

```
SELECT TIMESTAMPTZ_CMP_TIMESTAMP('2008-01-24 06:43:29+00', '2008-01-24 06:43:29'),
 TIMESTAMPTZ_CMP_TIMESTAMP('2008-01-24 06:43:29+00', '2008-02-18 02:36:48'),
 TIMESTAMPTZ_CMP_TIMESTAMP('2008-02-18 02:36:48+00', '2008-01-24 06:43:29');

timestamptz_cmp_timestamp  | timestamptz_cmp_timestamp | timestamptz_cmp_timestamp
---------------------------+---------------------------+--------------------------
             0             |            -1             |             1
```

TIMEZONE 함수

TIMEZONE은 지정한 시간대와 타임스탬프 값에 대한 타임스탬프를 반환합니다.

시간대 설정 방법에 대한 자세한 내용과 예는 timezone 섹션을 참조하세요.

시간대를 변환하는 방법에 대한 자세한 내용과 예는 CONVERT_TIMEZONE 섹션을 참조하세요.

구문

```
TIMEZONE('timezone', { timestamp | timestamptz })
```

인수

시간대

반환 값의 시간대입니다. 시간대는 시간대 이름(**'Africa/Kampala'**, **'Singapore'** 등) 또는 시간대 약어(**'UTC'**, **'PDT'** 등)로 지정할 수 있습니다. 지원되는 시간대 이름 목록을 보려면 다음 명령을 실행합니다.

```
select pg_timezone_names();
```

지원되는 시간대 이름 약어 목록을 보려면 다음 명령을 실행합니다.

```
select pg_timezone_abbrevs();
```

자세한 정보와 지침은 시간대 사용 노트 섹션을 참조하세요.

timestamp | timestamptz

타임스탬프 또는 TIMESTAMP 형식이나 TIMESTAMPTZ 형식 또는 표준 시간대가 있는 타임스탬프로 암시적으로 강제로 지정될 수 있는 값을 생성하는 표현식입니다.

반환 타입

TIMESTAMP 표현식과 함께 사용할 경우에는 TIMESTAMPTZ이고,

TIMESTAMPTZ 표현식과 함께 사용할 경우에는 TIMESTAMP입니다.

예시

다음은 PST 시간대의 타임스탬프 `2008-06-17 09:44:54`를 사용하여 UTC 시간대에 대한 타임스탬프를 반환합니다.

```
SELECT TIMEZONE('PST', '2008-06-17 09:44:54');

timezone
-----------------------
2008-06-17 17:44:54+00
```

다음은 UTC 시간대 `2008-06-17 09:44:54+00`의 타임스탬프를 사용하여 PST 시간대에 대한 타임스탬프를 반환합니다.

```
SELECT TIMEZONE('PST', timestamptz('2008-06-17 09:44:54+00'));

timezone
-----------------------
2008-06-17 01:44:54
```

TO_TIMESTAMP 함수

TO_TIMESTAMP는 TIMESTAMP 문자열을 TIMESTAMPTZ로 변환합니다. Amazon Redshift의 추가 날짜 및 시간 함수 목록은 날짜 및 시간 함수 섹션을 참조하세요.

구문

```
to_timestamp(timestamp, format)
```

```
to_timestamp (timestamp, format, is_strict)
```

인수

timestamp

타임스탬프 값을 format에서 지정하는 형식으로 표현한 문자열입니다. 이 인수를 비워 두면 타임스탬프 값의 기본값은 `0001-01-01 00:00:00`입니다.

format

timestamp 값의 형식을 정의하는 문자열 리터럴입니다. 시간대(**TZ**, **tz** 또는 **OF**)가 포함된 형식은 입력 값으로 지원되지 않습니다. 유효한 타임스탬프 형식은 날짜/시간 형식 문자열 섹션을 참조하세요.

is_strict

입력 타임스탬프 값이 범위를 벗어날 경우 오류가 반환되는지 여부를 지정하는 옵션 부울 값입니다. is_strict가 TRUE로 설정되면 범위를 벗어난 값이 있는 경우 오류가 반환됩니다. is_strict가 기본값인 FALSE로 설정되면 오버플로 값이 허용됩니다.

반환 타입

TIMESTAMPTZ

예시

다음 예는 TO_TIMESTAMP 함수를 사용하여 TIMESTAMP 문자열을 TIMESTAMPTZ로 변환하는 것을 보여줍니다.

```
select sysdate, to_timestamp(sysdate, 'YYYY-MM-DD HH24:MI:SS') as second;

timestamp                  | second
---------------------------   ----------------------
2021-04-05 19:27:53.281812 | 2021-04-05 19:27:53+00
```

날짜의 TO_TIMESTAMP 부분을 전달할 수 있습니다. 나머지 날짜 부분은 기본값으로 설정됩니다. 출력에 시간이 포함됩니다.

```
SELECT TO_TIMESTAMP('2017','YYYY');

to_timestamp
---------------------------
2017-01-01 00:00:00+00
```

다음 SQL 문은 '2011-12-18 24:38:15' 문자열을 TIMESTAMPTZ로 변환합니다. 결과는 시간 수가 24시간을 초과하므로 다음 날에 해당하는 TIMESTAMPTZ입니다.

```
SELECT TO_TIMESTAMP('2011-12-18 24:38:15', 'YYYY-MM-DD HH24:MI:SS');

to_timestamp
----------------------
2011-12-19 00:38:15+00
```

다음 SQL 문은 '2011-12-18 24:38:15' 문자열을 TIMESTAMPTZ로 변환합니다. 타임스탬프의 시간 값이 24시간을 초과하므로 결과는 오류입니다.

```
SELECT TO_TIMESTAMP('2011-12-18 24:38:15', 'YYYY-MM-DD HH24:MI:SS', TRUE);

ERROR:  date/time field time value out of range: 24:38:15.0
```

TRUNC 함수

TIMESTAMP를 잘라내고 DATE를 반환합니다.

이 함수를 사용하면 숫자를 잘라낼 수도 있습니다. 자세한 내용은 TRUNC 함수 단원을 참조하십시오.

구문

```
TRUNC(timestamp)
```

인수

timestamp

TIMESTAMP 데이터 형식의 열 또는 암시적으로 TIMESTAMP 형식으로 평가되는 표현식입니다.

00:00:00을 시간으로 하여 타임스탬프 값을 반환하려면 함수 결과를 TIMESTAMP로 캐스팅합니다.

반환 타입

날짜

예시

다음은 SYSDATE 함수(타임스탬프 반환)의 결과에서 날짜 구간을 반환하는 예입니다.

```
SELECT SYSDATE;

+----------------------------+
|         timestamp          |
+----------------------------+
| 2011-07-21 10:32:38.248109 |
+----------------------------+

SELECT TRUNC(SYSDATE);

+------------+
|   trunc    |
+------------+
| 2011-07-21 |
+------------+
```

다음 예에서는 TRUNC 함수를 TIMESTAMP 열에 적용합니다. 반환 형식은 날짜입니다.

```
SELECT TRUNC(starttime) FROM event
ORDER BY eventid LIMIT 1;

+------------+
|   trunc    |
+------------+
| 2008-01-25 |
+------------+
```

다음 예는 TRUNC 함수 결과를 TIMESTAMP로 캐스팅하여 00:00:00을 시간으로 하는 타임스탬프 값을 반환하는 예입니다.

```
SELECT CAST((TRUNC(SYSDATE)) AS TIMESTAMP);

+---------------------+
|        trunc        |
+---------------------+
| 2011-07-21 00:00:00 |
+---------------------+
```

날짜 또는 타임스탬프 함수의 날짜 부분

다음 표는 아래 함수의 인수로 허용되는 날짜 부분과 시간 부분의 이름 및 약어를 구분한 것입니다.

- DATEADD
- DATEDIFF
- DATE_PART
- EXTRACT

날짜 부분 또는 시간 부분	약어
millennium, millennia	mil, mils
century, centuries	c, cent, cents
decade, decades	dec, decs
Epoch	epoch(EXTRACT에서 지원됨)
year, years	y, yr, yrs
quarter, quarters	qtr, qtrs
month, months	mon, mons
week, weeks	w
요일	dayofweek, dow, dw, weekday(DATE_PART 및 EXTRACT 함수에서 지원됨) 일요일부터 시작하여 0~6의 정수를 반환합니다. **Note** DOW 날짜 부분은 날짜/시간 형식 문자열에 사용되는 요일(D) 날짜 부분과 동작이 다릅니다. D는 정수 1~7을 기반으로합니다. 여기서 일요일은 1입니다. 자세한 내용은 날짜/시간 형식 문자열 단원을 참조하십시오.
day_of_year	dayofyear, doy, dy, yearday(EXTRACT에서 지원됨)
day, days	d

날짜 부분 또는 시간 부분	약어
hour, hours	h, hr, hrs
minute, minutes	m, min, mins
second, seconds	s, sec, secs
millisecond, milliseconds	ms, msec, msecs, msecond, mseconds, millisec, millisecs, millisecon
microsecond, microseconds	microsec, microsecs, microsecond, usecond, useconds, us, usec, usecs
timezone, timezone_hour, timezone_minute	시간대를 포함한 타임스탬프(TIMESTAMPTZ)인 경우에만 EXTRACT에서 지원됩니다.

초, 밀리초 및 마이크로초에서 결과의 차이

다른 날짜 함수에서 초, 밀리초 또는 마이크로초를 날짜 부분으로 지정하더라도 쿼리 결과에서는 최소한의 차이만 발생합니다.

- EXTRACT 함수는 지정한 날짜 부분에 한해 정수를 반환하고 더 높거나 낮은 단위의 날짜 부분은 무시합니다. 예를 들어 지정한 날짜 부분이 초라면 밀리초와 마이크로초는 결과에 포함되지 않습니다. 지정한 날짜 부분이 밀리초라면 초와 마이크로초가 포함되지 않습니다. 지정한 날짜 부분이 마이크로초라면 초와 밀리초가 포함되지 않습니다.
- DATE_PART 함수는 지정하는 날짜 부분에 상관없이 타임스탬프에서 전체 초 부분을 반환하며, 이때 반환되는 값은 필요에 따라 소수 값이 될 수도 있고, 정수가 될 수도 있습니다.

예를 들어 다음 쿼리의 결과를 서로 비교하십시오.

```
create table seconds(micro timestamp);

insert into seconds values('2009-09-21 11:10:03.189717');

select extract(sec from micro) from seconds;

date_part
-----------
```

```
3

select date_part(sec, micro) from seconds;

pgdate_part
-------------
3.189717
```

CENTURY, EPOCH, DECADE 및 MIL 참고 사항

CENTURY 또는 CENTURIES

Amazon Redshift는 CENTURY를 ###1년부터 시작하여 ###0년으로 끝나는 것으로 해석합니다.

```
select extract (century from timestamp '2000-12-16 12:21:13');
date_part
-----------
20

select extract (century from timestamp '2001-12-16 12:21:13');
date_part
-----------
21
```

EPOCH

Amazon Redshift의 EPOCH 구현은 클러스터가 상주하는 시간대와 별도로 1970-01-01 00:00:00.000000을 따릅니다. 따라서 클러스터가 상주하는 시간대에 따라 시간차를 기준으로 결과를 오프셋 처리해야 할 수도 있습니다.

다음은 아래 과정을 설명한 예입니다.

1. EVENT 테이블에 EVENT_EXAMPLE이라는 이름의 테이블을 생성합니다. 이 CREATE AS 명령은 DATE_PART 함수를 사용하여 각 이벤트마다 epoch 값을 저장할 날짜 열(기본 이름 PGDATE_PART)을 생성합니다.
2. PG_TABLE_DEF에서 EVENT_EXAMPLE의 열과 데이터 형식을 선택합니다.
3. EVENT_EXAMPLE 테이블에서 EVENTNAME, STARTTIME 및 PGDATE_PART를 선택하여 다른 날짜 및 시간 형식을 확인합니다.
4. EVENT EXAMPLE에서 EVENTNAME과 STARTTIME을 선택합니다. PGDATE_PART의 epoch 값을 1초 간격으로 시간대를 제외한 타임스탬프로 변환한 후 CONVERTED_TIMESTAMP라는 이름의 열에 결과를 반환합니다.

```
create table event_example
as select eventname, starttime, date_part(epoch, starttime) from event;

select "column", type from pg_table_def where tablename='event_example';

     column    |            type
---------------+-----------------------------
 eventname     | character varying(200)
 starttime     | timestamp without time zone
 pgdate_part   | double precision
(3 rows)
```

```
select eventname, starttime, pgdate_part from event_example;

   eventname          |      starttime      | pgdate_part
----------------------+---------------------+-------------
 Mamma Mia!           | 2008-01-01 20:00:00 |  1199217600
 Spring Awakening     | 2008-01-01 15:00:00 |  1199199600
 Nas                  | 2008-01-01 14:30:00 |  1199197800
 Hannah Montana       | 2008-01-01 19:30:00 |  1199215800
 K.D. Lang            | 2008-01-01 15:00:00 |  1199199600
 Spamalot             | 2008-01-02 20:00:00 |  1199304000
 Macbeth              | 2008-01-02 15:00:00 |  1199286000
 The Cherry Orchard   | 2008-01-02 14:30:00 |  1199284200
 Macbeth              | 2008-01-02 19:30:00 |  1199302200
 Demi Lovato          | 2008-01-02 19:30:00 |  1199302200


select eventname,
starttime,
timestamp with time zone 'epoch' + pgdate_part * interval '1 second' AS
 converted_timestamp
from event_example;

       eventname      |      starttime      | converted_timestamp
----------------------+---------------------+---------------------
 Mamma Mia!           | 2008-01-01 20:00:00 | 2008-01-01 20:00:00
 Spring Awakening     | 2008-01-01 15:00:00 | 2008-01-01 15:00:00
 Nas                  | 2008-01-01 14:30:00 | 2008-01-01 14:30:00
 Hannah Montana       | 2008-01-01 19:30:00 | 2008-01-01 19:30:00
 K.D. Lang            | 2008-01-01 15:00:00 | 2008-01-01 15:00:00
 Spamalot             | 2008-01-02 20:00:00 | 2008-01-02 20:00:00
 Macbeth              | 2008-01-02 15:00:00 | 2008-01-02 15:00:00
```

```
The Cherry Orchard   | 2008-01-02 14:30:00 | 2008-01-02 14:30:00
Macbeth              | 2008-01-02 19:30:00 | 2008-01-02 19:30:00
Demi Lovato          | 2008-01-02 19:30:00 | 2008-01-02 19:30:00
...
```

DECADE 또는 DECADES

Amazon Redshift는 DECADE 또는 DECADES DATEPART를 일반 달력에 따라 해석합니다. 예를 들어 일반 역법은 0001년부터 시작하기 때문에 첫 10년(decade 1)은 0001-01-01부터 0009-12-31까지이며, 두 번째 10년(decade 2)은 0010-01-01부터 0019-12-31까지입니다. 이러한 식으로 decade 201은 2000-01-01부터 2009-12-31까지입니다.

```
select extract(decade from timestamp '1999-02-16 20:38:40');
date_part
-----------
200

select extract(decade from timestamp '2000-02-16 20:38:40');
date_part
-----------
201

select extract(decade from timestamp '2010-02-16 20:38:40');
date_part
-----------
202
```

MIL 또는 MILS

Amazon Redshift는 MIL을 #001년 첫 번째 날짜부터 시작하여 #000년 마지막 날짜로 끝나는 것으로 해석합니다.

```
select extract (mil from timestamp '2000-12-16 12:21:13');
date_part
-----------
2

select extract (mil from timestamp '2001-12-16 12:21:13');
date_part
-----------
3
```

해시 함수

주제

해시 함수는 숫자 입력 값을 다른 값으로 변환하는 수학 함수입니다.

CHECKSUM 함수

해시 인덱스를 빌드하기 위한 체크섬 값을 계산합니다.

구문

```
CHECKSUM(expression)
```

인수

expression

입력 표현식의 데이터 형식은 VARCHAR, INTEGER 또는 DECIMAL이 되어야 합니다.

반환 타입

CHECKSUM 함수는 정수를 반환합니다.

예제

다음은 COMMISSION 열의 체크섬 값을 계산하는 예입니다.

```
select checksum(commission)
```

```
from sales
order by salesid
limit 10;

checksum
----------
10920
1140
5250
2625
2310
5910
11820
2955
8865
975
(10 rows)
```

farmFingerprint64 함수

Fingerprint64 함수를 사용하여 입력 인수의 팜해시 값을 계산합니다.

구문

```
farmFingerprint64(expression)
```

인수

expression

입력 표현식은 VARCHAR 또는 VARBYTE 데이터 형식이어야 합니다.

반환 타입

farmFingerprint64 함수는 BIGINT를 반환합니다.

예제

다음 예에서는 입력인 Amazon Redshift의 farmFingerprint64 값을 VARCHAR 데이터 형식으로 반환합니다.

```
SELECT farmFingerprint64('Amazon Redshift');
```

```
  farmfingerprint64
---------------------
 8085098817162212970
```

다음 예에서는 입력인 Amazon Redshift의 farmFingerprint64 값을 VARBYTE 데이터 형식으로 반환합니다.

```
SELECT farmFingerprint64('Amazon Redshift'::varbyte);
```

```
  farmfingerprint64
---------------------
 8085098817162212970
```

FUNC_SHA1 함수

SHA1 함수의 동의어입니다.

SHA1 함수 섹션을 참조하세요.

FNV_HASH 함수

모든 기본 데이터 형식에 대해 64비트 FNV-1a 비암호화 해시 함수를 계산합니다.

구문

```
FNV_HASH(value [, seed])
```

인수

USD 상당

입력 값은 해시 처리할 수 있습니다. Amazon Redshift는 값의 이진 표현을 사용하여 입력 값을 해시 처리합니다. 예를 들어 INTEGER 값은 4바이트를 사용하여 해시 처리되고 BIGINT 값은 8바이트를 사용하여 해시 처리됩니다. 또한 해싱 CHAR 및 VARCHAR 입력은 후행 공백을 무시하지 않습니다.

시드

해시 함수의 BIGINT 시드는 선택 사항입니다. 지정하지 않으면 Amazon Redshift는 기본 FNV 시드를 사용합니다. 이렇게 하면 변환 또는 연결 없이 여러 열의 해시를 결합할 수 있습니다.

반환 타입

BIGINT

예제

다음은 숫자의 FNV 해시, 'Amazon Redshift' 문자열 및 이 둘의 연결을 반환하는 예입니다.

```
select fnv_hash(1);
        fnv_hash
----------------------
 -5968735742475085980
(1 row)
```

```
select fnv_hash('Amazon Redshift');
      fnv_hash
---------------------
 7783490368944507294
(1 row)
```

```
select fnv_hash('Amazon Redshift', fnv_hash(1));
       fnv_hash
----------------------
 -2202602717770968555
(1 row)
```

사용 노트

- 여러 열이 있는 테이블의 해시를 계산하려면 첫 번째 열의 FNV 해시를 계산하여 두 번째 열의 해시에 시드로 전달하면 됩니다. 그런 다음 두 번째 열의 FNV 해시를 세 번째 열의 해시에 시드로 전달합니다.

  다음은 여러 열이 있는 테이블을 해시 처리할 시드를 생성하는 예입니다.

  ```
  select fnv_hash(column_3, fnv_hash(column_2, fnv_hash(column_1))) from sample_table;
  ```

- 동일한 속성을 사용하여 문자열 연결 해시를 계산할 수 있습니다.

  ```
  select fnv_hash('abcd');
         fnv_hash
  ```

```
---------------------
 -281581062704388899
(1 row)
```

```
select fnv_hash('cd', fnv_hash('ab'));
      fnv_hash
---------------------
 -281581062704388899
(1 row)
```

- 해시 함수는 입력 유형을 사용하여 해시 처리할 바이트 수를 결정합니다. 필요한 경우 캐스팅을 사용하여 특정 유형을 적용합니다.

  다음은 다른 결과를 생성하기 위해 다른 유형의 입력을 사용하는 예입니다.

```
select fnv_hash(1::smallint);
      fnv_hash
--------------------
 589727492704079044
(1 row)
```

```
select fnv_hash(1);
       fnv_hash
----------------------
 -5968735742475085980
(1 row)
```

```
select fnv_hash(1::bigint);
       fnv_hash
----------------------
 -8517097267634966620
(1 row)
```

MD5 함수

MD5 암호화 해시 함수를 사용하여 가변 길이 문자열을 128비트 체크섬의 16진수 값을 텍스트로 표현한 32자 문자열로 변환합니다.

구문

```
MD5(string)
```

인수

string

가변 길이 문자열입니다.

반환 타입

MD5 함수는 128비트 체크섬의 16진수 값을 텍스트로 표현한 32자 문자열을 반환합니다.

예시

다음은 문자열 'Amazon Redshift'를 128비트 값으로 표현한 예입니다.

```
select md5('Amazon Redshift');
md5
----------------------------------
f7415e33f972c03abd4f3fed36748f7a
(1 row)
```

SHA 함수

SHA1 함수의 동의어입니다.

SHA1 함수 섹션을 참조하세요.

SHA1 함수

SHA1 함수는 SHA1 암호화 해시 함수를 사용하여 가변 길이 문자열을 160비트 체크섬의 16진수 값을 텍스트로 표현한 40자 문자열로 변환합니다.

구문

SHA1은 SHA 함수 및 FUNC_SHA1 함수의 동의어입니다.

```
SHA1(string)
```

인수

string

가변 길이 문자열입니다.

반환 타입

SHA1 함수는 160비트 체크섬의 16진수 값을 텍스트로 표현한 40자 문자열을 반환합니다.

예제

다음은 단어 'Amazon Redshift'를 160비트 값으로 반환하는 예입니다.

```
select sha1('Amazon Redshift');
```

SHA2 함수

SHA2 함수는 SHA2 암호화 해시 함수를 사용하여 가변 길이 문자열을 문자열로 변환합니다. 문자열은 지정된 비트 수가 있는 체크섬의 16진수 값을 텍스트로 표현한 것입니다.

구문

```
SHA2(string, bits)
```

인수

string

가변 길이 문자열입니다.

integer

해시 함수의 비트 수입니다. 유효한 값은 0(256과 동일), 224, 256, 384 및 512입니다.

반환 타입

SHA2 함수는 체크섬의 16진수 값을 텍스트로 표현한 문자열을 반환하거나 비트 수가 유효하지 않은 경우 빈 문자열을 반환합니다.

예제

다음은 단어 'Amazon Redshift'를 256비트 값으로 반환하는 예입니다.

```
select sha2('Amazon Redshift', 256);
```

MURMUR3_32_HASH

MURMUR3_32_HASH 함수는 숫자 및 문자열 유형을 비롯한 모든 일반 데이터 유형에 대해 32비트 Murmur3A 비암호화 해시를 계산합니다.

구문

```
MURMUR3_32_HASH(value [, seed])
```

인수

USD 상당

입력 값은 해시 처리합니다. Amazon Redshift 는 입력 값의 바이너리 표현을 해시한 예입니다. 이 동작은 FNV_HASH 함수과 비슷하지만 값이 Apache Iceberg 32비트 Murmur3 해시 사양에 지정된 바이너리 표현으로 변환됩니다.

시드

해시 함수의 INT 시드입니다. 이 인수는 선택 사항입니다. 지정하지 않으면 Amazon Redshift는 기본 0 시드를 사용합니다. 이렇게 하면 변환 또는 연결 없이 여러 열의 해시를 결합할 수 있습니다.

반환 타입

함수는 INT를 반환합니다.

예제

다음은 숫자의 Murmur3 해시, 'Amazon Redshift' 문자열 및 이 둘의 연결을 반환하는 예입니다.

```
select MURMUR3_32_HASH(1);

    MURMUR3_32_HASH
----------------------
 1392991556
```

```
(1 row)
```

```
select MURMUR3_32_HASH('Amazon Redshift');

    MURMUR3_32_HASH
----------------------
 -1563580564
(1 row)
```

```
select MURMUR3_32_HASH('Amazon Redshift', MURMUR3_32_HASH(1));

    MURMUR3_32_HASH
----------------------
 -1346554171
(1 row)
```

사용 노트

여러 열이 있는 테이블의 해시를 계산하려면 첫 번째 열의 Murmur3 해시를 계산하여 두 번째 열의 해시에 시드로 전달하면 됩니다. 그런 다음 두 번째 열의 Murmur3 해시를 세 번째 열의 해시에 시드로 전달합니다.

다음은 여러 열이 있는 테이블을 해시 처리할 시드를 생성하는 예입니다.

```
select MURMUR3_32_HASH(column_3, MURMUR3_32_HASH(column_2, MURMUR3_32_HASH(column_1)))
 from sample_table;
```

동일한 속성을 사용하여 문자열 연결 해시를 계산할 수 있습니다.

```
select MURMUR3_32_HASH('abcd');

   MURMUR3_32_HASH
----------------------
 1139631978
(1 row)
```

```
select MURMUR3_32_HASH('cd', MURMUR3_32_HASH('ab'));

   MURMUR3_32_HASH
----------------------
 1711522338
```

```
(1 row)
```

해시 함수는 입력 유형을 사용하여 해시 처리할 바이트 수를 결정합니다. 필요한 경우 캐스팅을 사용하여 특정 유형을 적용합니다.

다음은 다른 결과를 생성하기 위해 다른 입력 유형을 사용하는 예입니다.

```
select MURMUR3_32_HASH(1, MURMUR3_32_HASH(1));

   MURMUR3_32_HASH
--------------------
 -1193428387
(1 row)
```

```
select MURMUR3_32_HASH(1);

   MURMUR3_32_HASH
----------------------
 1392991556
(1 row)
```

```
select MURMUR3_32_HASH(1, MURMUR3_32_HASH(2));

   MURMUR3_32_HASH
----------------------
 1179621905
(1 row)
```

HyperLogLog 함수

다음으로 Amazon Redshift에서 지원하는 SQL용 HyperLogLog 함수에 대한 설명을 찾아볼 수 있습니다.

주제

- HLL 함수
- HLL_CREATE_SKETCH 함수
- HLL_CARDINALITY 함수
- HLL_COMBINE 함수
- HLL_COMBINE_SKETCHES 함수

HLL 함수

HLL 함수는 입력 표현식 값의 HyperLogLog 카디널리티를 반환합니다. HLL 함수는 HLLSKETCH 데이터 형식을 제외한 모든 데이터 형식에서 작동합니다. HLL 함수는 NULL 값을 무시합니다. 테이블에 행이 없거나 모든 행이 NULL이면 결과 카디널리티는 0입니다.

구문

```
HLL (aggregate_expression)
```

인수

aggregate_expression

열 이름과 같이 집계에 값을 제공하는 모든 유효 표현식입니다. 이 함수는 HLLSKETCH, GEOMETRY, GEOGRAPHY 및 VARBYTE를 제외한 모든 데이터 유형을 입력으로 지원합니다.

반환 타입

HLL 함수는 BIGINT 또는 INT8 값을 반환합니다.

예시

다음 예에서는 테이블 a_table에 있는 열 an_int의 카디널리티를 반환합니다.

```
CREATE TABLE a_table(an_int INT);
INSERT INTO a_table VALUES (1), (2), (3), (4);

SELECT hll(an_int) AS cardinality FROM a_table;
cardinality
-------------
4
```

HLL_CREATE_SKETCH 함수

HLL_CREATE_SKETCH 함수는 입력 표현식 값을 캡슐화하는 HLLSKETCH 데이터 형식을 반환합니다. HLL_CREATE_SKETCH 함수는 모든 데이터 형식에서 작동하며 NULL 값을 무시합니다. 테이블에 행이 없거나 모든 행이 NULL이면 결과 스케치에 인덱스-값 페어(예: `{"version":1,"logm":15,"sparse":{"indices":[],"values":[]}}`)가 없습니다.

구문

```
HLL_CREATE_SKETCH (aggregate_expression)
```

인수

aggregate_expression

열 이름과 같이 집계에 값을 제공하는 모든 유효 표현식입니다. NULL 값은 무시됩니다. 이 함수는 HLLSKETCH, GEOMETRY, GEOGRAPHY 및 VARBYTE를 제외한 모든 데이터 유형을 입력으로 지원합니다.

반환 타입

HLL_CREATE_SKETCH 함수는 HLLSKETCH 값을 반환합니다.

예시

다음 예에서는 테이블 a_table의 열 an_int에 대한 HLLSKETCH 형식을 반환합니다. JSON 객체는 스케치를 가져오거나 내보내거나 인쇄할 때 희소 HyperLogLog 스케치를 나타내는 데 사용됩니다. 문자열 표현(Base64 형식)은 고밀도 HyperLogLog 스케치를 나타내는 데 사용됩니다.

```
CREATE TABLE a_table(an_int INT);
INSERT INTO a_table VALUES (1), (2), (3), (4);

SELECT hll_create_sketch(an_int) AS sketch FROM a_table;
sketch
--------------------------------------------------------------------------------------------------
{"version":1,"logm":15,"sparse":{"indices":
[20812342,20850007,22362299,47158030],"values":[1,2,1,1]}}
(1 row)
```

HLL_CARDINALITY 함수

HLL_CARDINALITY 함수는 입력 HLLSKETCH 데이터 형식의 카디널리티를 반환합니다.

구문

```
HLL_CARDINALITY (hllsketch_expression)
```

인수

hllsketch_expression

열 이름과 같이 HLLSKETCH 형식으로 평가되는 모든 유효한 표현식입니다. 입력 값은 HLLSKETCH 데이터 형식입니다.

반환 타입

HLL_CARDINALITY 함수는 BIGINT 또는 INT8 값을 반환합니다.

예시

다음 예에서는 테이블 `hll_table`에 있는 열 `sketch`의 카디널리티를 반환합니다.

```
CREATE TABLE a_table(an_int INT, b_int INT);
INSERT INTO a_table VALUES (1,1), (2,1), (3,1), (4,1), (1,2), (2,2), (3,2), (4,2),
 (5,2), (6,2);

CREATE TABLE hll_table (sketch HLLSKETCH);
INSERT INTO hll_table select hll_create_sketch(an_int) from a_table group by b_int;

SELECT hll_cardinality(sketch) AS cardinality FROM hll_table;
cardinality
-------------
6
4
(2 rows)
```

HLL_COMBINE 함수

HLL_COMBINE 집계 함수는 모든 입력 HLLSKETCH 값을 결합하는 HLLSKETCH 데이터 형식을 반환합니다.

둘 이상의 HyperLogLog 스케치 조합은 각 입력 스케치가 나타내는 고유 값의 조합에 대한 정보를 캡슐화하는 새로운 HLLSKETCH입니다. 스케치를 결합한 후 Amazon Redshift는 둘 이상의 데이터 집합 조합의 카디널리티를 추출합니다. 여러 스케치를 결합하는 방법에 대한 자세한 내용은 예: 여러 스케치를 결합하여 HyperLogLog 스케치 반환 섹션을 참조하세요.

구문

```
HLL_COMBINE (hllsketch_expression)
```

인수

hllsketch_expression

열 이름과 같이 HLLSKETCH 형식으로 평가되는 모든 유효한 표현식입니다. 입력 값은 HLLSKETCH 데이터 형식입니다.

반환 타입

HLL_COMBINE 함수는 HLLSKETCH 형식을 반환합니다.

예시

다음 예에서는 테이블 `hll_table`에 결합된 HLLSKETCH 값을 반환합니다.

```
CREATE TABLE a_table(an_int INT, b_int INT);
INSERT INTO a_table VALUES (1,1), (2,1), (3,1), (4,1), (1,2), (2,2), (3,2), (4,2),
 (5,2), (6,2);

CREATE TABLE hll_table (sketch HLLSKETCH);
INSERT INTO hll_table select hll_create_sketch(an_int) from a_table group by b_int;

SELECT hll_combine(sketch) AS sketches FROM hll_table;
sketches
--------------------------------------------------------------------------------------------------------
{"version":1,"logm":15,"sparse":{"indices":
[20812342,20850007,22362299,40314817,42650774,47158030],"values":[1,2,1,3,2,1]}}
(1 row)
```

HLL_COMBINE_SKETCHES 함수

HLL_COMBINE_SKETCHES는 두 개의 HLLSKETCH 값을 입력으로 받고 단일 HLLSKETCHE로 결합하는 스칼라 함수입니다.

둘 이상의 HyperLogLog 스케치 조합은 각 입력 스케치가 나타내는 고유 값의 조합에 대한 정보를 캡슐화하는 새로운 HLLSKETCH입니다.

구문

```
HLL_COMBINE_SKETCHES (hllsketch_expression1, hllsketch_expression2)
```

인수

hllsketch_expression1 및 hllsketch_expression2

열 이름과 같이 HLLSKETCH 형식으로 평가되는 모든 유효한 표현식입니다.

반환 타입

HLL_COMBINE_SKETCHES 함수는 HLLSKETCH 형식을 반환합니다.

예시

다음 예에서는 테이블 `hll_table`에 결합된 HLLSKETCH 값을 반환합니다.

```
WITH tbl1(x, y)
     AS (SELECT Hll_create_sketch(1),
                Hll_create_sketch(2)
         UNION ALL
         SELECT Hll_create_sketch(3),
                Hll_create_sketch(4)
         UNION ALL
         SELECT Hll_create_sketch(5),
                Hll_create_sketch(6)
         UNION ALL
         SELECT Hll_create_sketch(7),
                Hll_create_sketch(8)),
     tbl2(x, y)
     AS (SELECT Hll_create_sketch(9),
                Hll_create_sketch(10)
         UNION ALL
         SELECT Hll_create_sketch(11),
                Hll_create_sketch(12)
         UNION ALL
         SELECT Hll_create_sketch(13),
                Hll_create_sketch(14)
         UNION ALL
         SELECT Hll_create_sketch(15),
                Hll_create_sketch(16)
         UNION ALL
         SELECT Hll_create_sketch(NULL),
                Hll_create_sketch(NULL)),
     tbl3(x, y)
     AS (SELECT *
```

```
          FROM   tbl1
          UNION ALL
          SELECT *
          FROM   tbl2)
SELECT Hll_combine_sketches(x, y)
FROM   tbl3;
```

JSON 함수

주제

- JSON_PARSE 함수
- CAN_JSON_PARSE 함수
- JSON_SERIALIZE 함수
- JSON_SERIALIZE_TO_VARBYTE 함수
- 텍스트 기반 JSON 함수

> **Note**
>
> JSON 작업 시 다음 함수를 사용하는 것이 좋습니다.
>
> - JSON_PARSE 함수
> - CAN_JSON_PARSE 함수
> - JSON_SERIALIZE 함수
> - JSON_SERIALIZE_TO_VARBYTE 함수
>
> JSON_PARSE를 사용하면 모으기 시 JSON 텍스트를 SUPER 유형 값으로 한 번만 변환하면 되며, 그 이후에는 SUPER 값을 사용하여 작업을 수행할 수 있습니다. Amazon Redshift는 텍스트 기반 JSON 함수의 출력인 VARCHAR보다 SUPER 값을 더 효율적으로 구문 분석합니다. SUPER 데이터 유형에 대한 자세한 내용은 Amazon Redshift의 반정형 데이터 섹션을 참조하세요.

비교적 작은 용량의 키-값 페어 집합을 저장해야 할 때는 JSON 형식으로 저장하면 공간을 절약할 수 있습니다. JSON 문자열은 단일 열에 저장할 수 있기 때문에 테이블 형식의 데이터 저장보다는 JSON을 사용하는 것이 더욱 효율적입니다. 예를 들어 가능한 모든 속성을 완전하게 표현하려면 다수의 열이 필요하지만 임의의 행 또는 열에서 대부분 열 값이 NULL인 희소(sparse) 테이블이 하나 있다고 가정하

겠습니다. 이때 JSON을 사용하면 행 데이터를 키:값 페어로 단일 JSON 문자열에 저장하여 희박하게 채워진 테이블 열을 제거할 수 있습니다.

또한 JSON 스키마가 변경될 때 테이블에 열을 추가할 필요 없이 JSON 문자열을 쉽게 수정하여 추가 키:값 페어를 저장할 수 있습니다.

JSON은 적게 사용하는 것이 바람직합니다. 특히 대용량의 데이터 집합을 저장할 때는 JSON을 사용하지 않는 것이 좋습니다. 이때는 이질적인 데이터가 단일 열에 저장되면서 JSON가 Amazon Redshift 열 저장 아키텍처를 사용하지 못하기 때문입니다. Amazon Redshift는 CHAR 및 VARCHAR 열에 대한 JSON 함수를 지원하지만 JSON 직렬화 형식의 데이터를 처리하는 데 SUPER를 사용하는 것이 좋습니다. SUPER는 계층 데이터를 효율적으로 쿼리할 수 있는 사후 구문 분석 스키마 없는 표현을 사용합니다. SUPER 데이터 형식에 대한 자세한 내용은 Amazon Redshift의 반정형 데이터 섹션을 참조하세요.

JSON은 UTF-8로 인코딩된 텍스트 문자열을 사용합니다. 따라서 JSON 문자열은 CHAR 또는 VARCHAR 데이터 형식으로 저장될 수 있습니다.

JSON 문자열은 다음 규칙에 따라 올바른 형식의 JSON이 되어야 합니다.

- 루트 레벨 JSON은 JSON 객체 또는 JSON 배열일 수 있습니다. JSON 객체는 쉼표로 구분된 키:값 페어의 집합으로서 순서 지정 없이 중괄호로 묶입니다.

  예제: `{"one":1, "two":2}`
- JSON 배열은 쉼표로 구분된 값의 집합으로서 순서 지정과 함께 대괄호로 묶입니다.

  예제는 다음과 같습니다: `["first", {"one":1}, "second", 3, null]`
- JSON 배열은 0부터 시작되는 인덱스를 사용하기 때문에 배열의 첫 요소가 0 위치에 자리합니다. JSON 키:값 페어에서 키는 큰따옴표로 묶이는 문자열입니다.
- JSON 값은 다음 중 하나일 수 있습니다.
  - JSON 객체
  - array
  - 문자열
    - 큰따옴표를 사용하여 표시
  - number
    - 정수, 소수, 부동 소수점 포함
  - boolean
  - null

- 빈 객체와 빈 배열도 유효한 JSON 값입니다.
- JSON 필드는 대/소문자를 구분합니다.
- JSON 구조 요소 사이의 공백({ }, [ ] 등)은 무시됩니다.

Amazon Redshift JSON 함수와 Amazon Redshift COPY 명령은 동일한 메서드를 사용하여 JSON 형식 데이터를 처리합니다. JSON 작업에 대한 자세한 내용은 JSON 형식의 COPY 지원 섹션을 참조하세요.

JSON_PARSE 함수

JSON_PARSE 함수는 JSON 형식의 데이터를 구문 분석하고 SUPER 표현으로 변환합니다.

INSERT 또는 UPDATE 명령을 사용하여 SUPER 데이터 형식으로 수집하려면 JSON_PARSE 함수를 사용합니다. JSON_PARSE()를 사용하여 JSON 문자열을 SUPER 값으로 구문 분석하는 경우 특정 제한 사항이 적용됩니다. 자세한 내용은 SUPER에 대한 구문 분석 옵션 섹션을 참조하세요.

구문

```
JSON_PARSE( {json_string | binary_value} )
```

인수

json_string

직렬화된 JSON을 VARBYTE 또는 VARCHAR 형식으로 반환하는 표현식입니다.

binary_value

VARBYTE 유형의 이진 값입니다.

반환 타입

SUPER

예시

JSON 배열 [10001,10002,"abc"]를 SUPER 데이터 형식으로 변환하려면 다음 예제를 사용합니다.

```
SELECT JSON_PARSE('[10001,10002,"abc"]');
```

```
+---------------------+
|     json_parse      |
+---------------------+
| [10001,10002,"abc"] |
+---------------------+
```

함수가 JSON 배열을 SUPER 데이터 형식으로 변환했는지 확인하려면 다음 예제를 사용합니다. 자세한 내용은 JSON_TYPEOF 함수 단원을 참조하세요.

```
SELECT JSON_TYPEOF(JSON_PARSE('[10001,10002,"abc"]'));

+-------------+
| json_typeof |
+-------------+
| array       |
+-------------+
```

CAN_JSON_PARSE 함수

CAN_JSON_PARSE 함수는 JSON 형식의 데이터를 구문 분석하고 JSON_PARSE 함수를 사용하여 결과를 SUPER 값으로 변환할 수 있는 경우 `true`를 반환합니다.

구문

```
CAN_JSON_PARSE( {json_string | binary_value} )
```

인수

json_string

직렬화된 JSON을 VARCHAR 형식으로 반환하는 표현식입니다.

binary_value

VARBYTE 유형의 이진 값입니다.

반환 타입

BOOLEAN

사용 노트

- CAN_JSON_PARSE는 빈 문자열에 대해 false를 반환합니다. 입력 인수가 null이면 NULL을 반환합니다.

예시

다음 예제에서는 CASE 조건을 사용하여 올바른 형식의 JSON 배열에 대해 실행되는 CAN_JSON_PARSE를 보여줍니다. true가 반환되므로 Amazon Redshift는 예제 값에 대해 JSON_PARSE 함수를 실행합니다.

```
SELECT CASE
            WHEN CAN_JSON_PARSE('[10001,10002,"abc"]')
            THEN JSON_PARSE('[10001,10002,"abc"]')
        END;

 case
---------------------
'[10001,10002,"abc"]'
```

다음 예제에서는 CASE 조건을 사용하여 JSON 형식이 아닌 값에 대해 실행되는 CAN_JSON_PARSE를 보여줍니다. false가 반환되므로 Amazon Redshift는 CASE 조건의 ELSE 절에 있는 세그먼트를 대신 반환합니다.

```
SELECT CASE
            WHEN CAN_JSON_PARSE('This is a string.')
            THEN JSON_PARSE('This is a string.')
            ELSE 'This is not JSON.'
        END;

 case
---------------------
"This is not JSON."
```

JSON_SERIALIZE 함수

JSON_SERIALIZE 함수는 RFC 8259에 따라 SUPER 표현식을 텍스트 JSON 표현으로 직렬화합니다. RFC에 대한 자세한 내용은 The JavaScript Object Notation (JSON) Data Interchange Format을 참조하세요.

SUPER 크기 제한은 블록 제한과 거의 동일하고 VARCHAR 제한은 SUPER 크기 제한보다 작습니다. 따라서 JSON_SERIALIZE 함수는 JSON 형식이 시스템의 varchar 제한을 초과하면 오류를 반환합니다. SUPER 표현식의 크기를 확인하려면 JSON_SIZE 함수를 참조하세요.

구문

```
JSON_SERIALIZE(super_expression)
```

인수

super_expression

SUPER 표현식 또는 열입니다.

반환 타입

VARCHAR

예시

SUPER 값을 문자열로 직렬화하려면 다음 예제를 사용합니다.

```
SELECT JSON_SERIALIZE(JSON_PARSE('[10001,10002,"abc"]'));

+----------------------+
|   json_serialize     |
+----------------------+
| [10001,10002,"abc"] |
+----------------------+
```

JSON_SERIALIZE_TO_VARBYTE 함수

JSON_SERIALIZE_TO_VARBYTE 함수는 SUPER 값을 JSON_SERIALIZE()와 유사한 JSON 문자열로 변환하지만 대신 VARBYTE 값에 저장합니다.

구문

```
JSON_SERIALIZE_TO_VARBYTE(super_expression)
```

인수

super_expression

SUPER 표현식 또는 열입니다.

반환 타입

VARBYTE

예시

SUPER 값을 직렬화하고 결과를 VARBYTE 형식으로 반환하려면 다음 예제를 사용합니다.

```
SELECT JSON_SERIALIZE_TO_VARBYTE(JSON_PARSE('[10001,10002,"abc"]'));

+----------------------------------------+
|       json_serialize_to_varbyte        |
+----------------------------------------+
| 5b31303030312c31303030322c22616263225d |
+----------------------------------------+
```

SUPER 값을 직렬화하고 결과를 VARCHAR 형식으로 캐스팅하려면 다음 예제를 사용합니다. 자세한 내용은 CAST 함수 단원을 참조하십시오.

```
SELECT CAST((JSON_SERIALIZE_TO_VARBYTE(JSON_PARSE('[10001,10002,"abc"]'))) AS VARCHAR);

+---------------------------+
| json_serialize_to_varbyte |
+---------------------------+
| [10001,10002,"abc"]       |
+---------------------------+
```

텍스트 기반 JSON 함수

다음 함수는 JSON 값을 VARCHAR로 구문 분석합니다. JSON을 구문 분석하려면 JSON 값을 SUPER로 구문 분석하는 다음 함수를 대신 사용하는 것이 좋습니다. Amazon Redshift는 VARCHAR보다 SUPER 값을 더 효율적으로 구문 분석합니다.

- JSON_PARSE 함수

- CAN_JSON_PARSE 함수
- JSON_SERIALIZE 함수
- JSON_SERIALIZE_TO_VARBYTE 함수

주제

- IS_VALID_JSON 함수
- IS_VALID_JSON_ARRAY 함수
- JSON_ARRAY_LENGTH 함수
- JSON_EXTRACT_ARRAY_ELEMENT_TEXT 함수
- JSON_EXTRACT_PATH_TEXT 함수

IS_VALID_JSON 함수

> **Note**
>
> CAN_JSON_PARSE 및 관련 함수는 JSON 값을 SUPER로 구문 분석하는데, Amazon Redshift는 VARCHAR보다 더 효율적으로 구문 분석합니다.
> IS_VALID_JSON을 사용하는 대신 CAN_JSON_PARSE 함수를 사용하여 JSON 문자열을 검증하는 것이 좋습니다.

IS_VALID_JSON 함수는 JSON 문자열을 검증합니다. 이 함수는 문자열이 제대로 구성된 JSON인 경우 부울 값 `true`를, 문자열이 잘못 구성된 경우에는 부울 값 `false`를 반환합니다. JSON 배열을 확인하는 방법은 IS_VALID_JSON_ARRAY 함수 섹션을 참조하세요.

자세한 내용은 JSON 함수 단원을 참조하십시오.

구문

```
IS_VALID_JSON('json_string')
```

인수

json_string

JSON 문자열로 평가되는 문자열 또는 식입니다.

반환 타입

BOOLEAN

예시

테이블을 만들고 테스트용 JSON 문자열을 삽입하려면 다음 예제를 사용합니다.

```
CREATE TABLE test_json(id int IDENTITY(0,1), json_strings VARCHAR);

-- Insert valid JSON strings --
INSERT INTO test_json(json_strings) VALUES
('{"a":2}'),
('{"a":{"b":{"c":1}}}'),
('{"a": [1,2,"b"]}');

-- Insert invalid JSON strings --
INSERT INTO test_json(json_strings) VALUES
('{{}}'),
('{1:"a"}'),
('[1,2,3]');
```

앞선 예제에 있는 문자열의 유효성을 검사하려면 다음 예제를 사용합니다.

```
SELECT id, json_strings, IS_VALID_JSON(json_strings)
FROM test_json
ORDER BY id;

+----+--------------------+---------------+
| id |    json_strings    | is_valid_json |
+----+--------------------+---------------+
|  0 | {"a":2}            | true          |
|  4 | {"a":{"b":{"c":1}}} | true          |
|  8 | {"a": [1,2,"b"]}   | true          |
| 12 | {{}}               | false         |
| 16 | {1:"a"}            | false         |
| 20 | [1,2,3]            | false         |
+----+--------------------+---------------+
```

IS_VALID_JSON_ARRAY 함수

> **Note**
>
> JSON_PARSE 및 관련 함수는 JSON 값을 SUPER로 구문 분석하는데, Amazon Redshift는 VARCHAR보다 더 효율적으로 구문 분석합니다.
> IS_VALID_JSON_ARRAY를 사용하는 대신 JSON_PARSE 함수를 사용해 JSON 문자열을 구문 분석하여 SUPER 값을 얻는 것이 좋습니다. 그런 다음 IS_ARRAY 함수 함수를 사용하여 배열이 올바른 형식인지 확인합니다.

IS_VALID_JSON_ARRAY 함수는 JSON 배열을 검증합니다. 이 함수는 배열이 제대로 구성된 JSON인 경우 부울 값 `true`를, 배열이 잘못 구성된 경우에는 부울 값 `false`를 반환합니다. JSON 문자열을 확인하는 방법은 IS_VALID_JSON 함수 섹션을 참조하세요.

자세한 내용은 JSON 함수 단원을 참조하십시오.

구문

```
IS_VALID_JSON_ARRAY('json_array')
```

인수

json_array

JSON 배열로 평가되는 문자열 또는 식입니다.

반환 타입

BOOLEAN

예시

테이블을 만들고 테스트용 JSON 문자열을 삽입하려면 다음 예제를 사용합니다.

```
CREATE TABLE test_json_arrays(id int IDENTITY(0,1), json_arrays VARCHAR);

-- Insert valid JSON array strings --
INSERT INTO test_json_arrays(json_arrays)
VALUES('[]'),
```

```
('["a","b"]'),
('["a",["b",1,["c",2,3,null]]]');

-- Insert invalid JSON array strings --
INSERT INTO test_json_arrays(json_arrays)
VALUES('{"a":1}'),
('a'),
('[1,2,]');
```

앞선 예제에 있는 문자열의 유효성을 검사하려면 다음 예제를 사용합니다.

```
SELECT json_arrays, IS_VALID_JSON_ARRAY(json_arrays)
FROM test_json_arrays ORDER BY id;

+------------------------------+---------------------+
|         json_arrays          | is_valid_json_array |
+------------------------------+---------------------+
| []                           | true                |
| ["a","b"]                    | true                |
| ["a",["b",1,["c",2,3,null]]] | true                |
| {"a":1}                      | false               |
| a                            | false               |
| [1,2,]                       | false               |
+------------------------------+---------------------+
```

JSON_ARRAY_LENGTH 함수

> **Note**
>
> JSON_PARSE 및 관련 함수는 JSON 값을 SUPER로 구문 분석하는데, Amazon Redshift는 VARCHAR보다 더 효율적으로 구문 분석합니다.
> JSON_ARRAY_LENGTH를 사용하는 대신 JSON_PARSE 함수를 사용해 JSON 문자열을 구문 분석하여 SUPER 값을 얻는 것이 좋습니다. 그런 다음 get_array_length 함수를 사용하여 배열의 길이를 구합니다.

JSON_ARRAY_LENGTH 함수는 JSON 문자열의 바깥쪽 배열에 속한 요소 수를 반환합니다. null_if_invalid 인수가 `true`로 설정되어 있는데 JSON 문자열이 잘못된 경우, 이 함수는 오류 대신 NULL을 반환합니다.

자세한 내용은 JSON 함수 단원을 참조하십시오.

구문

```
JSON_ARRAY_LENGTH('json_array' [, null_if_invalid ] )
```

인수

json_array

올바른 형식의 JSON 배열입니다.

null_if_invalid

(선택) 입력 JSON 문자열이 잘못된 경우 오류 대신 NULL을 반환할지 여부를 지정하는 BOOLEAN 값입니다. JSON이 잘못되었을 때 NULL을 반환하게 하려면 true(t)를 지정합니다. JSON이 잘못되었을 때 오류를 반환하게 하려면 false(f)를 지정합니다. 기본값은 false입니다.

반환 타입

INTEGER

예시

배열의 요소 수를 반환하려면 다음 예제를 사용합니다.

```
SELECT JSON_ARRAY_LENGTH('[11,12,13,{"f1":21,"f2":[25,26]},14]');

+-------------------+
| json_array_length |
+-------------------+
|                 5 |
+-------------------+
```

JSON이 유효하지 않아 오류를 반환하려면 다음 예제를 사용합니다.

```
SELECT JSON_ARRAY_LENGTH('[11,12,13,{"f1":21,"f2":[25,26]},14');

ERROR: invalid json array object [11,12,13,{"f1":21,"f2":[25,26]},14
```

잘못된 JSON에 대해 오류를 반환하는 대신 NULL을 반환하도록 null_if_invalid를 true로 설정하려면 다음 예제를 사용합니다.

```
SELECT JSON_ARRAY_LENGTH('[11,12,13,{"f1":21,"f2":[25,26]},14',true);

+-------------------+
| json_array_length |
+-------------------+
| NULL              |
+-------------------+
```

JSON_EXTRACT_ARRAY_ELEMENT_TEXT 함수

> **Note**
>
> JSON_PARSE 및 관련 함수는 JSON 값을 SUPER로 구문 분석하는데, Amazon Redshift는 VARCHAR보다 더 효율적으로 구문 분석합니다.
> JSON_EXTRACT_ARRAY_ELEMENT_TEXT를 사용하는 대신 JSON_PARSE 함수를 사용해 JSON 문자열을 구문 분석하여 SUPER 값을 얻는 것이 좋습니다. 그런 다음 `value[element position]` 구문으로 배열 인덱스를 사용하여 원하는 요소를 쿼리합니다. SUPER 값의 배열 요소를 쿼리하는 방법에 대한 자세한 내용은 비정형 데이터 쿼리 섹션을 참조하세요.

JSON_EXTRACT_ARRAY_ELEMENT_TEXT 함수는 0부터 시작되는 인덱스를 사용하여 가장 바깥쪽 JSON 문자열 배열의 JSON 배열 요소를 반환합니다. 배열의 첫 번째 요소는 0 위치에 자리합니다. 인덱스가 음의 값이거나 경계를 벗어나면 JSON_EXTRACT_ARRAY_ELEMENT_TEXT 함수가 빈 문자열을 반환합니다. null_if_invalid 인수가 `true`로 설정되어 있는데 JSON 문자열이 잘못된 경우, 이 함수는 오류 대신 NULL을 반환합니다.

자세한 내용은 JSON 함수 단원을 참조하십시오.

구문

```
JSON_EXTRACT_ARRAY_ELEMENT_TEXT('json string', pos [, null_if_invalid ] )
```

인수

json_string

올바른 형식의 JSON 문자열입니다.

pos

반환할 배열 요소의 인덱스를 0부터 시작되는 배열 인덱스를 사용하여 나타내는 INTEGER입니다.

null_if_invalid

(선택) 입력 JSON 문자열이 잘못된 경우 오류 대신 NULL을 반환할지 여부를 지정하는 BOOLEAN 값입니다. JSON이 잘못되었을 때 NULL을 반환하게 하려면 true(t)를 지정합니다. JSON이 잘못되었을 때 오류를 반환하게 하려면 false(f)를 지정합니다. 기본값은 false입니다.

반환 타입

VARCHAR

pos에서 참조한 JSON 배열 요소를 나타내는 VARCHAR 문자열입니다.

예시

0부터 시작하는 배열 인덱스의 세 번째 요소인 위치 2에 있는 배열 요소를 반환하려면 다음 예제를 사용합니다.

```
SELECT JSON_EXTRACT_ARRAY_ELEMENT_TEXT('[111,112,113]', 2);

+---------------------------------+
| json_extract_array_element_text |
+---------------------------------+
|                             113 |
+---------------------------------+
```

JSON이 유효하지 않아 오류를 반환하려면 다음 예제를 사용합니다.

```
SELECT JSON_EXTRACT_ARRAY_ELEMENT_TEXT('["a",["b",1,["c",2,3,null,]]]',1);

ERROR: invalid json array object ["a",["b",1,["c",2,3,null,]]]
```

다음 예에서는 null_if_invalid를 true로 설정해 문이 잘못된 JSON에 대해 오류가 아니라 NULL을 반환하도록 합니다.

```
SELECT JSON_EXTRACT_ARRAY_ELEMENT_TEXT('["a",["b",1,["c",2,3,null,]]]',1,true);

+---------------------------------+
| json_extract_array_element_text |
+---------------------------------+
```

```
| NULL                              |
+-----------------------------------+
```

JSON_EXTRACT_PATH_TEXT 함수

> **Note**
>
> JSON_PARSE 및 관련 함수는 JSON 값을 SUPER로 구문 분석하는데, Amazon Redshift는 VARCHAR보다 더 효율적으로 구문 분석합니다.
> JSON_EXTRACT_PATH_TEXT를 사용하는 대신 JSON_PARSE 함수를 사용해 JSON 문자열을 구문 분석하여 SUPER 값을 얻는 것이 좋습니다. 그런 다음 `value.attribute` 구문으로 원하는 요소를 쿼리합니다. SUPER 값의 배열 요소를 쿼리하는 방법에 대한 자세한 내용은 비정형 데이터 쿼리 섹션을 참조하세요.

JSON_EXTRACT_PATH_TEXT 함수는 JSON 문자열의 연속된 경로 요소에서 참조하는 키-값 페어 값을 반환합니다. JSON 경로는 최대 5개 레벨까지 중첩될 수 있습니다. 경로 요소는 대/소문자를 구분합니다. JSON 문자열에 경로 요소가 존재하지 않으면 JSON_EXTRACT_PATH_TEXT가 NULL을 반환합니다.

null_if_invalid 인수가 `true`로 설정되어 있는데 JSON 문자열이 잘못된 경우, 이 함수는 오류 대신 NULL을 반환합니다.

JSON_EXTRACT_PATH_TEXT의 최대 데이터 크기는 64KB입니다. 따라서 JSON 레코드가 64KB보다 큰 경우 JSON_EXTRACT_PATH_TEXT로 처리 시 오류가 발생합니다.

추가 JSON 함수에 대한 자세한 내용은 JSON 함수 섹션을 참조하세요. JSON 작업에 대한 자세한 내용은 JSON 형식의 COPY 지원 섹션을 참조하세요.

구문

```
JSON_EXTRACT_PATH_TEXT('json_string', 'path_elem' [,'path_elem'[, …] ]
 [, null_if_invalid ] )
```

인수

json_string

올바른 형식의 JSON 문자열입니다.

path_elem

JSON 문자열의 경로 요소입니다. 경로 요소 1개는 필수이며, 추가로 5개 레벨까지 경로 요소를 지정할 수 있습니다.

null_if_invalid

(선택) 입력 JSON 문자열이 잘못된 경우 오류 대신 NULL을 반환할지 여부를 지정하는 BOOLEAN 값입니다. JSON이 잘못되었을 때 NULL을 반환하게 하려면 `true`(`t`)를 지정합니다. JSON이 잘못되었을 때 오류를 반환하게 하려면 `false`(`f`)를 지정합니다. 기본값은 `false`입니다.

JSON 문자열에서는 Amazon Redshift가 `\n`을 줄 바꿈 문자로, 그리고 `\t`를 탭 문자로 인식합니다. 백슬래시를 로드하려면 백슬래시(`\\`)로 이스케이프하십시오. 자세한 내용은 JSON의 이스케이프 문자 단원을 참조하십시오.

반환 타입

VARCHAR

경로 요소에서 참조한 JSON 값을 나타내는 VARCHAR 문자열입니다.

예시

`'f4', 'f6'` 경로에 대한 값을 반환하려면 다음 예제를 사용합니다.

```
SELECT JSON_EXTRACT_PATH_TEXT('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}}','f4', 'f6');

+------------------------+
| json_extract_path_text |
+------------------------+
| star                   |
+------------------------+
```

JSON이 유효하지 않아 오류를 반환하려면 다음 예제를 사용합니다.

```
SELECT JSON_EXTRACT_PATH_TEXT('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}','f4', 'f6');

ERROR: invalid json object {"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}
```

이 문이 오류를 반환하는 대신 유효하지 않은 JSON에 대해 NULL을 반환하도록 null_if_invalid를 true로 설정하려면 다음 예제를 사용합니다.

```
SELECT JSON_EXTRACT_PATH_TEXT('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}','f4',
 'f6',true);

+------------------------+
| json_extract_path_text |
+------------------------+
| NULL                   |
+------------------------+
```

검색된 값이 세 번째 수준인 'farm', 'barn', 'color' 경로에 대한 값을 반환하려면 다음 예제를 사용합니다. 이 샘플은 가독성을 높여주는 JSON lint 도구로 형식이 지정되었습니다.

```
SELECT JSON_EXTRACT_PATH_TEXT('{
    "farm": {
        "barn": {
            "color": "red",
            "feed stocked": true
        }
    }
}', 'farm', 'barn', 'color');
+------------------------+
| json_extract_path_text |
+------------------------+
| red                    |
+------------------------+
```

'color' 요소가 누락되어 NULL을 반환하려면 다음 예제를 사용합니다. 이 샘플은 JSON lint 도구로 형식이 지정되었습니다.

```
SELECT JSON_EXTRACT_PATH_TEXT('{
    "farm": {
        "barn": {}
    }
}', 'farm', 'barn', 'color');

+------------------------+
| json_extract_path_text |
+------------------------+
| NULL                   |
```

```
+------------------------+
```

JSON이 유효한 경우 누락된 요소를 추출하려고 하면 NULL이 반환됩니다.

'house', 'appliances', 'washing machine', 'brand' 경로에 대한 값을 반환하려면 다음 예제를 사용합니다.

```
SELECT JSON_EXTRACT_PATH_TEXT('{
  "house": {
    "address": {
      "street": "123 Any St.",
      "city": "Any Town",
      "state": "FL",
      "zip": "32830"
    },
    "bathroom": {
      "color": "green",
      "shower": true
    },
    "appliances": {
      "washing machine": {
        "brand": "Any Brand",
        "color": "beige"
      },
      "dryer": {
        "brand": "Any Brand",
        "color": "white"
      }
    }
  }
}', 'house', 'appliances', 'washing machine', 'brand');

+------------------------+
| json_extract_path_text |
+------------------------+
| Any Brand              |
+------------------------+
```

다음 예시에서는 샘플 테이블을 만들고 SUPER 값으로 채운 다음, 두 행의 경로 'f2' 값을 반환합니다.

```
CREATE TABLE json_example(id INT, json_text SUPER);
```

```
INSERT INTO json_example VALUES
(1, JSON_PARSE('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}}')),
(2, JSON_PARSE('{
    "farm": {
        "barn": {
            "color": "red",
            "feed stocked": true
        }
    }
}'));

SELECT * FROM json_example;
id          | json_text
------------+--------------------------------------------
1           | {"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}}
2           | {"farm":{"barn":{"color":"red","feed stocked":true}}}


SELECT id, JSON_EXTRACT_PATH_TEXT(JSON_SERIALIZE(json_text), 'f2') FROM json_example;

id          | json_text
------------+--------------------------------------------
1           | {"f3":1}
2           |
```

기계 학습 함수

Amazon Redshift 기계 학습을 사용하면 SQL 문으로 기계 학습 모델을 훈련하고 예측을 위해 SQL 쿼리에서 해당 기계 학습 모델을 호출할 수 있습니다. Amazon Redshift 모델 설명에는 훈련 데이터의 각 속성이 예측 결과에 어떻게 기여하는지 이해하는 데 도움이 되는 특성 중요도 값이 포함되어 있습니다.

다음으로 Amazon Redshift에서 지원하는 SQL용 기계 학습 함수에 대한 설명을 찾아볼 수 있습니다.

주제

- EXPLAIN_MODEL 함수

EXPLAIN_MODEL 함수

EXPLAIN_MODEL 함수는 JSON 형식의 모델 설명 보고서가 포함된 SUPER 데이터 유형을 반환합니다. 설명 보고서에는 모든 모델 특성의 Shapley 값에 대한 정보가 들어 있습니다.

EXPLAIN_MODEL 함수는 현재 AUTO ON 또는 AUTO OFF XGBoost 모델만 지원합니다.

설명 보고서를 사용할 수 없는 경우 함수는 모델의 진행 상황을 보여주는 상태를 반환합니다. `Waiting for training job to complete`, `Waiting for processing job to complete` 및 `Processing job failed`가 포함됩니다.

CREATE MODEL 문을 실행하면 설명 상태가 `Waiting for training job to complete`이 됩니다. 모델이 훈련되고 설명 요청이 전송되면 설명 상태는 `Waiting for processing job to complete`이 됩니다. 모델 설명이 성공적으로 완료되면 전체 설명 보고서를 사용할 수 있습니다. 그렇지 않으면 상태는 `Processing job failed`가 됩니다.

CREATE MODEL 문을 실행할 때 선택적 `MAX_RUNTIME` 파라미터를 사용하여 교육에 소요되는 최대 시간을 지정할 수 있습니다. 모델 생성 시간이 해당 시간에 도달하면 Amazon Redshift는 모델 생성을 중지합니다. Autopilot 모델을 생성하는 동안 해당 시간 제한에 도달하면 Amazon Redshift가 지금까지 가장 적합한 모델을 반환합니다. 모델 설명 가능성은 모델 학습이 완료되면 사용할 수 있으므로 `MAX_RUNTIME`을 짧은 시간으로 설정하면 설명 가능성 보고서를 사용할 수 없을 수 있습니다. 학습 시간은 모델 복잡성, 데이터 크기 및 기타 요인에 따라 달라집니다.

구문

```
EXPLAIN_MODEL ('schema_name.model_name')
```

인수

schema_name

스키마의 이름입니다. schema_name을 지정하지 않으면 현재 스키마가 선택됩니다.

model_name

모델의 이름입니다. 스키마의 모델 이름은 고유해야 합니다.

반환 타입

EXPLAIN_MODEL 함수는 다음과 같이 SUPER 데이터 유형을 반환합니다.

```
{"version":"1.0","explanations":{"kernel_shap":{"label0":{"global_shap_values":
{"x0":0.05,"x1":0.10,"x2":0.30,"x3":0.15},"expected_value":0.50}}}}
```

예시

다음 예에서는 설명 상태 `waiting for training job to complete`을 반환합니다.

```
select explain_model('customer_churn_auto_model');
                 explain_model
-------------------------------------------------------
{"explanations":"waiting for training job to complete"}
(1 row)
```

모델 설명이 성공적으로 완료되면 다음과 같이 전체 설명 보고서를 사용할 수 있습니다.

```
select explain_model('customer_churn_auto_model');
                                        explain_model
-------------------------------------------------------------------------------------------------
{"version":"1.0","explanations":{"kernel_shap":{"label0":{"global_shap_values":
{"x0":0.05386043365892927,"x1":0.10801289723274592,"x2":0.23227865827017378,"x3":0.067668513394
(1 row)
```

EXPLAIN_MODEL 함수는 SUPER 데이터 유형을 반환하므로 설명 보고서를 쿼리할 수 있습니다. 이렇게 하면 global_shap_values, expected_value 또는 특성별 Shapley 값을 추출할 수 있습니다.

다음 예에서는 모델에 대해 global_shap_values를 추출합니다.

```
select json_table.report.explanations.kernel_shap.label0.global_shap_values from
 (select explain_model('customer_churn_auto_model') as report) as json_table;
                                                    global_shap_values
-------------------------------------------------------------------------------------------------
{"state":0.10983770427197151,"account_length":0.1772441398408543,"area_code":0.0862682396863959
(1 row)
```

다음 예에서는 특성 x0에 대해 global_shap_values를 추출합니다.

```
select json_table.report.explanations.kernel_shap.label0.global_shap_values.x0 from
 (select explain_model('customer_churn_auto_model') as report) as json_table;
          x0
------------------------
  0.05386043365892927
(1 row)
```

특정 스키마에서 모델이 생성되고 생성된 모델에 대한 액세스 권한이 있는 경우 다음과 같이 모델 설명을 쿼리할 수 있습니다.

```
-- Check the current schema
```

```
SHOW search_path;
   search_path
------------------
  $user, public
(1 row)
-- If you have the privilege to access the model explanation
-- in `test_schema`
SELECT explain_model('test_schema.test_model_name');
                       explain_model
-------------------------------------------------------------
{"explanations":"waiting for training job to complete"}
(1 row)
```

수학 함수

주제

- 수학 연산자 기호
- ABS 함수
- ACOS 함수
- ASIN 함수
- ATAN 함수
- ATAN2 함수
- CBRT 함수
- CEILING(또는 CEIL) 함수
- COS 함수
- COT 함수
- DEGREES 함수
- DEXP 함수
- DLOG1 함수
- DLOG10 함수
- EXP 함수
- FLOOR 함수
- LN 함수
- LOG 함수

- MOD 함수
- PI 함수
- POWER 함수
- RADIANS 함수
- RANDOM 함수
- ROUND 함수
- SIN 함수
- SIGN 함수
- SQRT 함수
- TAN 함수
- TRUNC 함수

이번 섹션에서는 Amazon Redshift에서 지원되는 수학 연산자 및 함수에 대해 설명합니다.

수학 연산자 기호

다음 표는 지원되는 수학 연산자를 나열한 것입니다.

지원되는 연산자

연산자	설명	예제	결과
+	더하기	2 + 3	5
-	빼기	2 - 3	-1
*	곱하기	2 * 3	6
/	나누기	/	2
%	모듈로	5 % 4	1
^	거듭제곱	2.0 ^ 3.0	8
|/	제곱근	| / 25.0	5
||/	세제곱근	|| / 27.0	3

연산자	설명	예제	결과
@	절대값	@ -5.0	5
<<	비트 왼쪽 이동	1 << 4	16
>>	비트 오른쪽 이동	8 >> 2	2
&	비트 논리곱	8 & 2	0

예시

다음 예제에서는 TICKIT 샘플 데이터베이스를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

주어진 거래에 대해 지불한 수수료와 2.00 USD의 수수료를 계산하려면 다음 예제를 사용합니다.

```
SELECT
    commission,
    (commission + 2.00) AS comm
FROM
    sales
WHERE
    salesid = 10000;

+------------+-------+
| commission | comm  |
+------------+-------+
|      28.05 | 30.05 |
+------------+-------+
```

주어진 거래에 대한 판매 가격의 20%를 계산하려면 다음 예제를 사용합니다.

```
SELECT pricepaid, (pricepaid * .20) as twentypct
FROM sales
WHERE salesid=10000;

+-----------+-----------+
| pricepaid | twentypct |
```

```
+-----------+-----------+
|       187 |      37.4 |
+-----------+-----------+
```

지속적인 성장 패턴을 기반으로 티켓 판매를 예측하려면 다음 예제를 사용합니다. 이번 예에서는 하위 쿼리가 2008년 판매된 티켓 수량을 반환합니다. 그런 다음 그 결과를 10년 연속 성장률 5%와 거듭제곱합니다.

```
SELECT (SELECT SUM(qtysold) FROM sales, date
WHERE sales.dateid=date.dateid AND year=2008)^((5::float/100)*10) AS qty10years;

+------------------+
|    qty10years    |
+------------------+
| 587.664019657491 |
+------------------+
```

날짜 ID가 2000보다 크거나 같은 판매에 대한 총 지불 가격 및 수수료를 찾으려면 다음 예제를 사용합니다. 그런 다음 가격 총액에서 수수료 총액을 뺍니다.

```
SELECT SUM(pricepaid) AS sum_price, dateid,
SUM(commission) AS sum_comm, (SUM(pricepaid) - SUM(commission)) AS value
FROM sales
WHERE dateid >= 2000
GROUP BY dateid
ORDER BY dateid
LIMIT 10;

+-----------+--------+----------+-----------+
| sum_price | dateid | sum_comm |   value   |
+-----------+--------+----------+-----------+
|    305885 |   2000 | 45882.75 | 260002.25 |
|    316037 |   2001 | 47405.55 | 268631.45 |
|    358571 |   2002 | 53785.65 | 304785.35 |
|    366033 |   2003 | 54904.95 | 311128.05 |
|    307592 |   2004 |  46138.8 |  261453.2 |
|    333484 |   2005 |  50022.6 |  283461.4 |
|    317670 |   2006 |  47650.5 |  270019.5 |
|    351031 |   2007 | 52654.65 | 298376.35 |
|    313359 |   2008 | 47003.85 | 266355.15 |
|    323675 |   2009 | 48551.25 | 275123.75 |
+-----------+--------+----------+-----------+
```

ABS 함수

ABS는 절대 숫자 값을 계산합니다. 여기에서 숫자란 리터럴이거나, 혹은 숫자로 평가되는 표현식이 될 수 있습니다.

구문

```
ABS(number)
```

인수

number

숫자, 또는 숫자로 평가되는 표현식입니다. SMALLINT, INTEGER, BIGINT, DECIMAL, FLOAT4, FLOAT8, 또는 SUPER 형식이 될 수 있습니다.

반환 타입

ABS는 인수와 동일한 데이터 형식을 반환합니다.

예시

-38의 절대값을 계산하려면 다음 예제를 사용합니다.

```
SELECT ABS(-38);

+-----+
| abs |
+-----+
|  38 |
+-----+
```

(14-76)의 절대값을 계산하려면 다음 예제를 사용합니다.

```
SELECT ABS(14-76);

+-----+
| abs |
+-----+
|  62 |
```

```
+-----+
```

ACOS 함수

ACOS는 숫자의 아크 코사인을 반환하는 삼각 함수입니다. 반환 값은 라디안 단위이며 0과 PI 사이입니다.

구문

```
ACOS(number)
```

인수

number

입력 파라미터는 DOUBLE PRECISION 수입니다.

반환 타입

DOUBLE PRECISION

예시

-1의 아크 코사인을 반환하려면 다음 예제를 사용합니다.

```
SELECT ACOS(-1);

+-------------------+
|       acos        |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

.5의 아크 코사인을 상응하는 도수로 변환하려면 다음 예제를 사용합니다.

```
SELECT (ACOS(.5) * 180/(SELECT PI())) AS degrees;

+-------------------+
|      degrees      |
+-------------------+
| 60.00000000000001 |
```

```
+-------------------+
```

ASIN 함수

ASIN은 숫자의 아크 사인을 반환하는 삼각 함수입니다. 반환 값은 라디안 단위이며 PI/2과 -PI/2 사이입니다.

구문

```
ASIN(number)
```

인수

number

입력 파라미터는 DOUBLE PRECISION 수입니다.

반환 타입

DOUBLE PRECISION

예시

1의 아크 사인을 반환하려면 다음 예제를 사용합니다.

```
SELECT ASIN(1) AS halfpi;

+--------------------+
|       halfpi       |
+--------------------+
| 1.5707963267948966 |
+--------------------+
```

.5의 아크 사인을 상응하는 도수로 변환하려면 다음 예제를 사용합니다.

```
SELECT (ASIN(.5) * 180/(SELECT PI())) AS degrees;

+--------------------+
|      degrees       |
+--------------------+
| 30.000000000000004 |
```

```
+--------------------+
```

ATAN 함수

ATAN은 숫자의 아크 탄젠트를 반환하는 삼각 함수입니다. 반환 값은 라디안 단위이며 -PI과 PI 사이입니다.

구문

```
ATAN(number)
```

인수

number

입력 파라미터는 DOUBLE PRECISION 수입니다.

반환 타입

DOUBLE PRECISION

예시

1의 아크 탄젠트를 반환하여 4와 곱하려면 다음 예제를 사용합니다.

```
SELECT ATAN(1) * 4 AS pi;

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

1의 아크 탄젠트를 상응하는 도수로 변환하려면 다음 예제를 사용합니다.

```
SELECT (ATAN(1) * 180/(SELECT PI())) AS degrees;

+---------+
| degrees |
+---------+
|      45 |
```

```
+---------+
```

ATAN2 함수

ATAN2는 다른 숫자로 나눈 숫자의 아크 탄젠트를 반환하는 삼각 함수입니다. 반환 값은 라디안 단위이며 PI/2과 -PI/2 사이입니다.

구문

```
ATAN2(number1, number2)
```

인수

number1

DOUBLE PRECISION 수입니다.

number2

DOUBLE PRECISION 수입니다.

반환 타입

DOUBLE PRECISION

예시

2/2의 아크 탄젠트를 반환하여 4와 곱하려면 다음 예제를 사용합니다.

```
SELECT ATAN2(2,2) * 4 AS PI;

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

1/0의 아크 탄젠트(0으로 평가됨)를 상응하는 도수로 변환하려면 다음 예제를 사용합니다.

```
SELECT (ATAN2(1,0) * 180/(SELECT PI())) AS degrees;
```

```
+---------+
| degrees |
+---------+
|      90 |
+---------+
```

CBRT 함수

CBRT 함수는 주어진 숫자의 세제곱근을 계산하는 수학 함수입니다.

구문

```
CBRT(number)
```

인수

CBRT는 DOUBLE PRECISION 숫자를 인수로 사용합니다.

반환 타입

DOUBLE PRECISION

예시

다음 예제에서는 TICKIT 샘플 데이터베이스를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

주어진 거래에 대해 지불한 수수료의 세제곱근을 계산하려면 다음 예제를 사용합니다.

```
SELECT CBRT(commission) FROM sales WHERE salesid=10000;

+--------------------+
|        cbrt        |
+--------------------+
| 3.0383953904884344 |
+--------------------+
```

CEILING(또는 CEIL) 함수

CEILING 또는 CEIL 함수는 숫자를 다음 정수(whole number)로 올림하는 데 사용됩니다. 반면 FLOOR 함수는 숫자를 다음 정수로 내림합니다.

구문

```
{CEIL | CEILING}(number)
```

인수

number

숫자 또는 숫자로 평가되는 표현식입니다. SMALLINT, INTEGER, BIGINT, DECIMAL, FLOAT4, FLOAT8, 또는 SUPER 형식이 될 수 있습니다.

반환 타입

CEILING과 CEIL은 인수와 동일한 데이터 형식을 반환합니다.

입력이 SUPER 형식이면 출력은 입력과 동일한 동적 형식을 유지하는 반면 정적 형식은 SUPER 형식을 유지합니다. SUPER의 동적 형식이 숫자가 아니면 Amazon Redshift는 null을 반환합니다.

예시

다음 예제에서는 TICKIT 샘플 데이터베이스를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

주어진 판매 거래에 대해 지급되는 수수료의 한도를 계산하려면 다음 예제를 사용합니다.

```
SELECT CEILING(commission) FROM sales
WHERE salesid=10000;

+---------+
| ceiling |
+---------+
|      29 |
+---------+
```

COS 함수

COS는 숫자의 코사인을 반환하는 삼각 함수입니다. 반환 값은 라디안 단위이며 -1과 1 사이(경계값 포함)입니다.

구문

```
COS(double_precision)
```

인수

number

입력 파라미터는 DOUBLE PRECISION 수입니다.

반환 타입

COS 함수는 DOUBLE PRECISION 숫자를 반환합니다.

예시

0의 코사인을 반환하려면 다음 예제를 사용합니다.

```
SELECT COS(0);

+-----+
| cos |
+-----+
|   1 |
+-----+
```

pi의 코사인을 반환하려면 다음 예제를 사용합니다.

```
SELECT COS(PI());

+-----+
| cos |
+-----+
|  -1 |
+-----+
```

COT 함수

COT는 숫자의 코탄젠트를 반환하는 삼각 함수입니다. 입력 파라미터는 0이 아닌 값이어야 합니다.

구문

```
COT(number)
```

인수

number

입력 파라미터는 DOUBLE PRECISION 수입니다.

반환 타입

DOUBLE PRECISION

예시

1의 코탄젠트를 반환하려면 다음 예제를 사용합니다.

```
SELECT COT(1);

+--------------------+
|        cot         |
+--------------------+
| 0.6420926159343306 |
+--------------------+
```

DEGREES 함수

라디안 단위의 각도를 도 단위의 등가로 변환합니다.

구문

```
DEGREES(number)
```

인수

number

입력 파라미터는 DOUBLE PRECISION 수입니다.

반환 타입

DOUBLE PRECISION

예시

0.5 라디안에 상응하는 도수를 반환하려면 다음 예제를 사용합니다.

```
SELECT DEGREES(.5);

+-------------------+
|      degrees      |
+-------------------+
| 28.64788975654116 |
+-------------------+
```

PI 라디안을 도 단위로 변환하려면 다음 예제를 사용합니

```
SELECT DEGREES(pi());

+---------+
| degrees |
+---------+
|     180 |
+---------+
```

DEXP 함수

DEXP 함수는 거듭제곱 값을 배정밀도 숫자의 유효숫자 표기법으로 반환합니다. DEXP 함수와 EXP 함수의 유일한 차이점은 DEXP 파라미터가 DOUBLE PRECISION이어야 한다는 데 있습니다.

구문

```
DEXP(number)
```

인수

number

입력 파라미터는 DOUBLE PRECISION 수입니다.

반환 타입

DOUBLE PRECISION

예제

다음 예제에서는 TICKIT 샘플 데이터베이스를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

DEXP 함수를 사용하여 연속 성장 패턴에 따른 티켓 판매를 예측합니다. 이번 예에서는 하위 쿼리가 2008년 판매된 티켓 수량을 반환합니다. 그런 다음 그 결과를 DEXP 함수 결과와 곱합니다. 이때 10년 연속 성장률은 7%로 지정합니다.

```
SELECT (SELECT SUM(qtysold)
FROM sales, date
WHERE sales.dateid=date.dateid
AND year=2008) * DEXP((7::FLOAT/100)*10) qty2010;

+-------------------+
|      qty2010      |
+-------------------+
| 695447.4837722216 |
+-------------------+
```

DLOG1 함수

DLOG1 함수는 입력 파라미터의 자연 로그를 반환합니다. LN 함수의 동의어입니다.

DLOG10 함수

DLOG10 함수는 입력 파라미터의 밑이 10인 로그를 반환합니다.

LOG 함수의 동의어입니다.

구문

```
DLOG10(number)
```

인수

number

입력 파라미터는 DOUBLE PRECISION 수입니다.

반환 타입

DOUBLE PRECISION

예제

숫자 100의 밑수 10 로그를 반환하려면 다음 예제를 사용합니다.

```
SELECT DLOG10(100);

+--------+
| dlog10 |
+--------+
|      2 |
+--------+
```

EXP 함수

EXP 함수는 숫자 표현식의 지수 함수, 또는 자연 알고리즘 기반인 거듭제곱된 e를 실행합니다. EXP 함수는 LN 함수의 역입니다.

구문

```
EXP(expression)
```

인수

expression

입력 표현식은 INTEGER, DECIMAL 또는 DOUBLE PRECISION 데이터 형식이어야 합니다.

반환 타입

DOUBLE PRECISION

예제

다음 예제에서는 TICKIT 샘플 데이터베이스를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

EXP 함수를 사용하여 연속 성장 패턴에 따른 티켓 판매를 예측합니다. 이번 예에서는 하위 쿼리가 2008년 판매된 티켓 수량을 반환합니다. 그런 다음 그 결과를 EXP 함수 결과와 곱합니다. 이때 10년 연속 성장률은 7%로 지정합니다.

```
SELECT (SELECT SUM(qtysold)
FROM sales, date
WHERE sales.dateid=date.dateid
AND year=2008) * EXP((7::FLOAT/100)*10) qty2018;

+-------------------+
|      qty2018      |
+-------------------+
| 695447.4837722216 |
+-------------------+
```

FLOOR 함수

FLOOR 함수는 숫자를 다음 정수(whole number)로 내림합니다.

구문

```
FLOOR(number)
```

인수

number

숫자 또는 숫자로 평가되는 표현식입니다. SMALLINT, INTEGER, BIGINT, DECIMAL, FLOAT4, FLOAT8, 또는 SUPER 형식이 될 수 있습니다.

반환 타입

FLOOR는 인수와 동일한 데이터 형식을 반환합니다.

입력이 SUPER 형식이면 출력은 입력과 동일한 동적 형식을 유지하는 반면 정적 형식은 SUPER 형식을 유지합니다. SUPER의 동적 형식이 숫자가 아니면 Amazon Redshift는 NULL을 반환합니다.

예시

다음 예제에서는 TICKIT 샘플 데이터베이스를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

FLOOR 함수 사용 전후에 주어진 판매 거래에 대해 지불한 수수료의 값을 표시하려면 다음 예제를 사용합니다.

```
SELECT commission
FROM sales
WHERE salesid=10000;

+------------+
| commission |
+------------+
|      28.05 |
+------------+

SELECT FLOOR(commission)
FROM sales
WHERE salesid=10000;

+-------+
| floor |
+-------+
|    28 |
+-------+
```

LN 함수

인수의 자연 로그를 반환합니다.

DLOG1 함수의 동의어입니다.

구문

```
LN(expression)
```

인수

expression

함수가 실행되는 대상 열 또는 표현식입니다.

> **Note**
>
> 이 함수는 표현식이 Amazon Redshift 사용자 생성 테이블이나 Amazon Redshift STL 또는 STV 시스템 테이블을 참조하는 경우 일부 데이터 형식에 대해 오류를 반환합니다.

다음과 같은 데이터 형식의 표현식은 사용자 생성 또는 시스템 테이블을 참조할 경우 오류를 나타냅니다. 이러한 데이터 형식의 표현식들은 리더 노드에서만 실행되기 때문입니다.

- BOOLEAN
- CHAR
- DATE
- DECIMAL 또는 NUMERIC
- TIMESTAMP
- VARCHAR

다음과 같은 데이터 형식의 표현식은 사용자 생성 테이블이나 STL 또는 STV 시스템 테이블에서 성공적으로 실행됩니다.

- BIGINT
- DOUBLE PRECISION
- INTEGER
- REAL
- SMALLINT

반환 타입

LN 함수는 입력 표현식과 동일한 형식을 반환합니다.

예시

숫자 2.718281828의 자연 로그, 즉 밑이 e인 로그를 반환하려면 다음 예제를 사용합니다.

```
SELECT LN(2.718281828);

+--------------------+
|         ln         |
+--------------------+
| 0.9999999998311267 |
```

```
+--------------------+
```

반환되는 값은 거의 1에 일치합니다.

다음 예제에서는 TICKIT 샘플 데이터베이스를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

USERS 테이블의 userid 열에 있는 값의 자연 로그를 반환하려면 다음 예제를 사용합니다.

```
SELECT username, LN(userid) FROM users ORDER BY userid LIMIT 10;

+----------+--------------------+
| username |         ln         |
+----------+--------------------+
| JSG99FHE |                  0 |
| PGL08LJI | 0.6931471805599453 |
| IFT66TXU | 1.0986122886681098 |
| XDZ38RDD | 1.3862943611198906 |
| AEB55QTM | 1.6094379124341003 |
| NDQ15VBM |  1.791759469228055 |
| OWY35QYB | 1.9459101490553132 |
| AZG78YIP | 2.0794415416798357 |
| MSD36KVR | 2.1972245773362196 |
| WKW41AIW |  2.302585092994046 |
+----------+--------------------+
```

LOG 함수

숫자의 로그를 반환합니다.

이 함수를 사용하여 밑이 10인 로그를 계산하는 경우 DLOG10 함수을 사용할 수도 있습니다.

구문

```
LOG([base, ]argument)
```

파라미터

base

(선택) 로그 함수의 밑입니다. 이 숫자는 양수여야 하며 1과 같을 수 없습니다. 이 파라미터를 생략하면 Amazon Redshift는 인수의 밑이 10인 로그를 계산합니다.

인수

로그 함수의 인수입니다. 이 숫자는 양수여야 합니다. 인수 값이 1이면 함수는 0을 반환합니다.

반환 타입

LOG 함수는 DOUBLE PRECISION 숫자를 반환합니다.

예시

밑이 2인 로그 100을 구하려면 다음 예제를 사용합니다.

```
SELECT LOG(2, 100);
+-------------------+
|        log        |
+-------------------+
| 6.643856189774725 |
+-------------------+
```

밑이 10인 로그 100을 구하려면 다음 예제를 사용합니다. 밑 파라미터를 생략하면 Amazon Redshift는 밑이 10이라고 가정합니다.

```
SELECT LOG(100);

+-----+
| log |
+-----+
|   2 |
+-----+
```

MOD 함수

모듈로 연산이라고도 하는 두 숫자의 나머지를 반환합니다. 결과를 계산하려면 첫 번째 파라미터를 두 번째 파라미터로 나눕니다.

구문

```
MOD(number1, number2)
```

인수

number1

첫 번째 입력 파라미터는 INTEGER, SMALLINT, BIGINT 또는 DECIMAL 숫자입니다. 두 파라미터 중 하나가 DECIMAL 형식이면 다른 파라미터도 DECIMAL 형식이어야 합니다 둘 중 한 파라미터가 INTEGER 형식이라면 나머지 파라미터는 INTEGER, SMALLINT 또는 BIGINT 형식이 될 수 있습니다. 두 파라미터 모두 SMALLINT 또는 BIGINT가 될 수 있지만 한 파라미터가 BIGINT라면 나머지 파라미터는 SMALLINT가 될 수 없습니다.

number2

두 번째 파라미터는 INTEGER, SMALLINT, BIGINT, 또는DECIMAL 숫자입니다. number2에도 number1과 동일한 데이터 형식 규칙이 적용됩니다.

반환 타입

MOD 함수의 반환 형식은 두 입력 파라미터의 형식이 동일하다는 가정 하에 입력 파라미터와 동일한 숫자 형식입니다. 하지만 둘 중 한 파라미터가 INTEGER이라면 반환 형식도 INTEGER가 됩니다. 유효한 반환 형식은 DECIMAL, INT, SMALLINT 및 BIGINT입니다.

사용 노트

%를 모듈로 연산자로 사용할 수 있습니다.

예시

숫자를 다른 숫자로 나눌 때 나머지를 반환하려면 다음 예제를 사용합니다.

```
SELECT MOD(10, 4);

+-----+
| mod |
+-----+
|   2 |
+-----+
```

MOD 함수를 사용할 때 DECIMAL 결과를 반환하려면 다음 예제를 사용합니다.

```
SELECT MOD(10.5, 4);

+-----+
```

```
| mod |
+-----+
| 2.5 |
+-----+
```

MOD 함수를 실행하기 전에 숫자를 캐스팅하려면 다음 예제를 사용합니다. 자세한 내용은 CAST 함수 단원을 참조하십시오.

```
SELECT MOD(CAST(16.4 AS INTEGER), 5);

+-----+
| mod |
+-----+
|   1 |
+-----+
```

첫 번째 파라미터를 2로 나누어 짝수인지 확인하려면 다음 예제를 사용합니다.

```
SELECT mod(5,2) = 0 AS is_even;

+---------+
| is_even |
+---------+
| false   |
+---------+
```

%를 모듈 연산자로 사용하려면 다음 예제를 사용합니다.

```
SELECT 11 % 4 as remainder;

 +-----------+
| remainder |
+-----------+
|         3 |
+-----------+
```

다음 예제에서는 TICKIT 샘플 데이터베이스를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

CATEGORY 테이블에서 홀수 카테고리에 대한 정보를 반환하려면 다음 예제를 사용합니다.

```
SELECT catid, catname
```

```
FROM category
WHERE MOD(catid,2)=1
ORDER BY 1,2;

+-------+-----------+
| catid |  catname  |
+-------+-----------+
|     1 | MLB       |
|     3 | NFL       |
|     5 | MLS       |
|     7 | Plays     |
|     9 | Pop       |
|    11 | Classical |
+-------+-----------+
```

PI 함수

pi 함수는 PI 값을 소수점 14자리까지 반환합니다.

구문

```
PI()
```

반환 타입

DOUBLE PRECISION

예시

pi 값을 반환하려면 다음 예제를 사용합니다.

```
SELECT PI();

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

POWER 함수

POWER 함수는 숫자 표현식을 두 번째 숫자 표현식의 거듭제곱으로 제곱하는 지수 함수입니다. 예를 들어 2의 세제곱은 POWER(2,3)으로 계산되어 8이라는 결과를 반환합니다.

구문

```
{POW | POWER}(expression1, expression2)
```

인수

expression1

제곱할 숫자 표현식입니다. 데이터 형식은 INTEGER, DECIMAL 또는 FLOAT여야 합니다.

expression2

expression1을 제곱할 거듭제곱입니다. 데이터 형식은 INTEGER, DECIMAL 또는 FLOAT여야 합니다.

반환 타입

DOUBLE PRECISION

예시

다음 예제에서는 TICKIT 샘플 데이터베이스를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

다음은 POWER 함수를 사용하여 2008년 티켓 판매 수량(하위 쿼리 결과)을 근거로 향후 10년간 티켓 판매 추이를 예측하는 예입니다. 이번 예에서는 연간 성장률을 7%로 설정하였습니다.

```
SELECT (SELECT SUM(qtysold) FROM sales, date
WHERE sales.dateid=date.dateid
AND year=2008) * POW((1+7::FLOAT/100),10) qty2010;

+-------------------+
|      qty2010      |
+-------------------+
| 679353.7540885945 |
+-------------------+
```

다음은 연간 성장률은 7%이지만 간격을 월로 설정했을 때(10년 = 120개월) 위의 예에 대한 분산도를 나타낸 예입니다.

```
SELECT (SELECT SUM(qtysold) FROM sales, date
WHERE sales.dateid=date.dateid
```

```
AND year=2008) * POW((1+7::FLOAT/100/12),120) qty2010;

+-----------------+
|     qty2010     |
+-----------------+
| 694034.54678046 |
+-----------------+
```

RADIANS 함수

RADIANS 함수는 도 단위의 각도를 라디안 단위의 등가로 변환합니다.

구문

```
RADIANS(number)
```

인수

number

입력 파라미터는 DOUBLE PRECISION 수입니다.

반환 타입

DOUBLE PRECISION

예시

라디안 환산 180도를 반환하려면 다음 예제를 사용합니다.

```
SELECT RADIANS(180);

+-------------------+
|      radians      |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

RANDOM 함수

RANDOM 함수는 0.0(포함)과 1.0(제외) 사이에서 무작위로 값을 생성합니다.

구문

```
RANDOM()
```

반환 타입

DOUBLE PRECISION

사용 노트

RANDOM이 예측 가능한 순서로 숫자를 생성할 수 있도록 SET 명령으로 시드 값을 설정한 후 RANDOM을 호출하십시오.

예시

0에서 99 사이의 무작위 값을 계산하려면 다음 예제를 사용합니다. 무작위 숫자가 0에서 1이면 이 쿼리는 0에서 100 사이의 무작위 숫자를 생성합니다.

```
SELECT CAST(RANDOM() * 100 AS INT);

+------+
| int4 |
+------+
|   59 |
+------+
```

다음은 RANDOM이 예측 가능한 순서로 숫자를 생성할 수 있도록 SET 명령을 사용하여 SEED 값을 설정하는 예입니다.

SEED 값을 설정하지 않고 3개의 RANDOM 정수를 반환하려면 다음 예제를 사용합니다.

```
SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|    6 |
+------+

SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|   68 |
```

```
+------+

SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|   56 |
+------+
```

SEED 값을 .25로 설정하고 RANDOM 숫자를 3개 더 반환하려면 다음 예제를 사용합니다.

```
SET SEED TO .25;
SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|   21 |
+------+

SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|   79 |
+------+

SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|   12 |
+------+
```

SEED 값을 .25로 재설정하고 RANDOM이 이전 세 번의 호출과 동일한 결과를 반환하는지 확인하려면 다음 예제를 사용합니다.

```
SET SEED TO .25;
SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|   21 |
+------+
```

```
SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|   79 |
+------+

SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|   12 |
+------+
```

다음 예제에서는 TICKIT 샘플 데이터베이스를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

SALES 테이블에서 10개 항목의 균일한 무작위 샘플을 검색하려면 다음 예제를 사용합니다.

```
SELECT *
FROM sales
ORDER BY RANDOM()
LIMIT 10;

+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
| salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
 commission |      saletime       |
+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
|   45422 |  51114 |     5983 |   24482 |    4369 |   2118 |       1 |       195 |
 29.25 | 2008-10-19 05:20:07 |
|   42481 |  47638 |     4573 |    6198 |    6479 |   1987 |       4 |      1140 |
   171 | 2008-06-10 09:39:19 |
|   31494 |  34759 |    18895 |    4719 |    7753 |   2090 |       4 |      1024 |
 153.6 | 2008-09-21 03:44:26 |
|  119388 | 136685 |    21815 |   41905 |    2071 |   1884 |       1 |       359 |
 53.85 | 2008-02-27 10:43:10 |
|  166990 | 225037 |    18529 |    7628 |     746 |   2113 |       1 |      2009 |
 301.35 | 2008-10-14 10:07:44 |
|   11146 |  12096 |    42685 |    6619 |    1876 |   2123 |       1 |        29 |
  4.35 | 2008-10-24 06:23:54 |
```

```
|  148537 | 172056 |    15102 |   11787 |    6122 |   1923 |       2 |       480 |
    72 | 2008-04-07 03:58:23 |
|   68945 |  78387 |     7359 |   18323 |    6636 |   1910 |       1 |       457 |
 68.55 | 2008-03-25 08:31:03 |
|   52796 |  59576 |     9909 |   15102 |    7958 |   1951 |       1 |       479 |
 71.85 | 2008-05-05 02:25:08 |
|   90684 | 103522 |    38052 |   21549 |    7384 |   2117 |       1 |       313 |
 46.95 | 2008-10-18 05:43:11 |
+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
```

10개 품목의 무작위 샘플을 검색하되 가격에 비례하여 품목을 선택하려면 다음 예제를 사용합니다. 예를 들어 다른 항목보다 가격이 두 배 높은 항목은 쿼리 결과에 나타날 가능성이 두 배 더 높습니다.

```
SELECT *
FROM sales
ORDER BY -LOG(RANDOM()) / pricepaid
LIMIT 10;

+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
| salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
 commission |      saletime       |
+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
|  158340 | 208208 |    17082 |   42018 |    1211 |   2160 |       4 |      6852 |
 1027.8 | 2008-11-30 12:21:43 |
|   53250 |  60069 |    12644 |    7066 |    7942 |   1838 |       4 |      1528 |
 229.2 | 2008-01-12 11:24:56 |
|   22929 |  24938 |    47314 |    6503 |     179 |   2000 |       3 |       741 |
 111.15 | 2008-06-23 08:04:50 |
|  164980 | 221181 |     1949 |   19670 |    1471 |   1906 |       1 |      1330 |
 199.5 | 2008-03-21 07:59:51 |
|  159641 | 211179 |    44897 |   16652 |    7458 |   2128 |       1 |      1019 |
 152.85 | 2008-10-29 02:02:15 |
|   73143 |  83439 |     5716 |    5727 |    7314 |   1903 |       1 |       248 |
  37.2 | 2008-03-18 11:07:42 |
|   84778 |  96749 |    46608 |   32980 |    3883 |   1999 |       2 |       958 |
 143.7 | 2008-06-22 12:13:31 |
|  171096 | 232929 |    43683 |    8536 |    8353 |   1870 |       1 |       929 |
 139.35 | 2008-02-13 01:36:36 |
|   74212 |  84697 |    39809 |   15569 |    5525 |   2105 |       2 |       896 |
 134.4 | 2008-10-06 11:47:50 |
```

```
|  158011 | 207556 |    25399 |   16881 |     232 |   2088 |       2 |      2526 |
 378.9 | 2008-09-19 06:00:26 |
+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
```

ROUND 함수

ROUND 함수는 숫자를 가장 가까운 정수 또는 소수로 반올림합니다.

ROUND 함수는 옵션으로 두 번째 인수를 추가할 수 있습니다. 이 두 번째 인수는 어느 방향이든 반올림할 소수 자릿수를 나타내는 INTEGER입니다. 두 번째 인수를 제공하지 않으면 함수는 가장 가까운 정수로 반올림됩니다. 두 번째 인수 integer거 지정되면 함수는 전체 자릿수의 소수 자릿수가 integer인 가장 가까운 숫자로 반올림됩니다.

구문

```
ROUND(number [ , integer ] )
```

인수

number

숫자 또는 숫자로 평가되는 표현식입니다. DECIMAL, FLOAT8 또는 SUPER 형식이 될 수 있습니다. Amazon Redshift는 다른 숫자 데이터 형식을 암시적으로 변환할 수 있습니다.

integer

(선택) 어느 방향으로든 반올림을 위한 소수 자릿수를 나타내는 INTEGER입니다. SUPER 데이터 형식은 이 인수에 대해 지원되지 않습니다.

반환 타입

ROUND는 입력 숫자와 동일한 숫자 데이터 형식을 반환합니다.

입력이 SUPER 형식이면 출력은 입력과 동일한 동적 형식을 유지하는 반면 정적 형식은 SUPER 형식을 유지합니다. SUPER의 동적 형식이 숫자가 아니면 Amazon Redshift는 NULL을 반환합니다.

예시

다음 예제에서는 TICKIT 샘플 데이터베이스를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

주어진 거래에서 지불되는 수수료를 가장 가까운 정수로 반올림하려면 다음 예제를 사용합니다.

```
SELECT commission, ROUND(commission)
FROM sales WHERE salesid=10000;

+------------+-------+
| commission | round |
+------------+-------+
|      28.05 |    28 |
+------------+-------+
```

주어진 거래에서 지불되는 수수료를 첫 번째 소수점 자리로 반올림하려면 다음 예제를 사용합니다.

```
SELECT commission, ROUND(commission, 1)
FROM sales WHERE salesid=10000;

+------------+-------+
| commission | round |
+------------+-------+
|      28.05 |  28.1 |
+------------+-------+
```

이전 예제와 반대 방향으로 정밀도를 확장하려면 다음 예제를 사용합니다.

```
SELECT commission, ROUND(commission, -1)
FROM sales WHERE salesid=10000;

+------------+-------+
| commission | round |
+------------+-------+
|      28.05 |    30 |
+------------+-------+
```

SIN 함수

SIN은 숫자의 사인을 반환하는 삼각 함수입니다. 반환 값은 -1과 1 사이입니다.

구문

```
SIN(number)
```

인수

number

라디안 단위의 DOUBLE PRECISION 숫자입니다.

반환 타입

DOUBLE PRECISION

예시

-PI의 사인을 반환하려면 다음 예제를 사용합니다.

```
SELECT SIN(-PI());

+-------------------------+
|           sin           |
+-------------------------+
| -0.00000000000000012246 |
+-------------------------+
```

SIGN 함수

SIGN 함수는 숫자의 부호(양의 부호 또는 음의 부호)를 반환합니다. SIGN 함수의 결과는 인수가 양수이면 1, 음수이면 -1, 0이면 0이 됩니다.

구문

```
SIGN(number)
```

인수

number

숫자, 또는 숫자로 평가되는 표현식입니다. DECIMAL, FLOAT8, 또는 SUPER 형식이 될 수 있습니다. 다른 데이터 형식은 암시적 변환 규칙에 따라 Amazon Redshift에 의해 변환될 수 있습니다.

반환 타입

SIGN은 입력 인수와 동일한 숫자 데이터 형식을 반환합니다. 입력이 DECIMAL인 경우 출력은 DECIMAL(1,0)입니다.

입력이 SUPER 형식이면 출력은 입력과 동일한 동적 형식을 유지하는 반면 정적 형식은 SUPER 형식을 유지합니다. SUPER의 동적 형식이 숫자가 아니면 Amazon Redshift는 NULL을 반환합니다.

예시

다음 예제에서는 입력이 DOUBLE PRECISION이므로 테이블 t2의 열 d의 형식이 DOUBLE PRECISION이고 입력이 NUMERIC이므로 테이블 t2의 열 n이 출력으로 NUMERIC(1,0)임을 보여 줍니다.

```
CREATE TABLE t1(d DOUBLE PRECISION, n NUMERIC(12, 2));
INSERT INTO t1 VALUES (4.25, 4.25), (-4.25, -4.25);
CREATE TABLE t2 AS SELECT SIGN(d) AS d, SIGN(n) AS n FROM t1;
SELECT table_name, column_name, data_type FROM SVV_REDSHIFT_COLUMNS WHERE
 table_name='t1' OR table_name='t2';

+------------+-------------+----------------------+
| table_name | column_name |       data_type      |
+------------+-------------+----------------------+
| t1         | d           | double precision     |
| t1         | n           | numeric(12,2)        |
| t2         | d           | double precision     |
| t2         | n           | numeric(1,0)         |
| t1         | col1        | character varying(20) |
+------------+-------------+----------------------+
```

다음 예제에서는 TICKIT 샘플 데이터베이스를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

판매 테이블에서 주어진 거래에 대해 지급된 수수료의 부호를 확인하려면 다음 예제를 사용합니다.

```
SELECT commission, SIGN(commission)
FROM sales WHERE salesid=10000;

+------------+------+
| commission | sign |
+------------+------+
|      28.05 |    1 |
+------------+------+
```

SQRT 함수

SQRT 함수는 NUMERIC 값의 제곱근을 반환합니다. 한 숫자에 동일한 숫자를 곱하면 지정된 값을 얻을 경우 해당 숫자를 제곱근이라고 합니다.

구문

```
SQRT(expression)
```

인수

expression

표현식에는 INTEGER, DECIMAL 또는 FLOAT 데이터 형식 또는 이러한 데이터 형식으로 암시적으로 변환되는 데이터 형식이 있어야 합니다. 표현식에는 함수가 포함될 수 있습니다. 표현식에는 함수가 포함될 수 있습니다.

반환 타입

DOUBLE PRECISION

예시

16의 제곱근을 반환하려면 다음 예제를 사용합니다.

```
SELECT SQRT(16);

+------+
| sqrt |
+------+
|    4 |
+------+
```

암시적 형식 변환을 사용하여 문자열 16의 제곱근을 반환하려면 다음 예제를 사용합니다.

```
SELECT SQRT('16');

+------+
| sqrt |
+------+
|    4 |
```

```
+------+
```

ROUND 함수를 사용한 후 16.4의 제곱근을 반환하려면 다음 예제를 사용합니다.

```
SELECT SQRT(ROUND(16.4));

+------+
| sqrt |
+------+
|    4 |
+------+
```

원의 넓이가 주어졌을 때 반지름의 길이를 반환하려면 다음 예제를 사용합니다. 예를 들어 면적을 제곱 인치로 지정하면, 반지름을 인치 단위로 계산합니다. 샘플에서 면적은 20입니다.

```
SELECT SQRT(20/PI()) AS radius;

+--------------------+
|      radius        |
+--------------------+
| 2.5231325220201604 |
+--------------------+
```

다음 예제에서는 TICKIT 샘플 데이터베이스를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

SALES 테이블에서 COMMISSION 값의 제곱근을 반환하려면 다음 예제를 사용합니다. COMMISSION 열은 DECIMAL 열입니다. 이 예에서는 복잡한 조건부 논리가 있는 쿼리에서 함수를 사용하는 방법을 보여줍니다.

```
SELECT SQRT(commission)
FROM sales WHERE salesid < 10 ORDER BY salesid;

+--------------------+
|        sqrt        |
+--------------------+
| 10.449880382090505 |
| 3.3763886032268267 |
|  7.245688373094719 |
|  5.123475382979799 |
|  4.806245936279167 |
```

```
|  7.687652437513028 |
| 10.871982339941507 |
| 5.4359911699707535 |
|   9.41541289588513 |
+--------------------+
```

동일한 COMMISSION 값 집합에 대해 반올림된 제곱근을 반환하려면 다음 예제를 사용합니다.

```
SELECT ROUND(SQRT(commission))
FROM sales WHERE salesid < 10 ORDER BY salesid;

+-------+
| round |
+-------+
|    10 |
|     3 |
|     7 |
|     5 |
|     5 |
|     8 |
|    11 |
|     5 |
|     9 |
+-------+
```

TAN 함수

TAN은 숫자의 탄젠트를 반환하는 삼각 함수입니다. 입력 인수는 숫자(라디안 단위)입니다.

구문

```
TAN(number)
```

인수

number

DOUBLE PRECISION 수입니다.

반환 타입

DOUBLE PRECISION

예시

0의 탄젠트를 반환하려면 다음 예제를 사용합니다.

```
SELECT TAN(0);

+-----+
| tan |
+-----+
|   0 |
+-----+
```

TRUNC 함수

TRUNC 함수는 숫자를 이전 정수 또는 소수로 자릅니다.

TRUNC 함수는 옵션으로 두 번째 인수를 추가할 수 있습니다. 이 두 번째 인수는 어느 방향이든 반올림할 소수 자릿수를 나타내는 INTEGER입니다. 두 번째 인수를 제공하지 않으면 함수는 가장 가까운 정수로 반올림됩니다. 두 번째 인수 integer거 지정되면 함수는 전체 자릿수의 소수 자릿수가 integer인 가장 가까운 숫자로 반올림됩니다.

이 함수는 TIMESTAMP를 절사하고 DATE를 반환할 수도 있습니다. 자세한 내용은 TRUNC 함수 단원을 참조하십시오.

구문

```
TRUNC(number [ , integer ])
```

인수

number

숫자 또는 숫자로 평가되는 표현식입니다. DECIMAL, FLOAT8 또는 SUPER 형식이 될 수 있습니다. 다른 데이터 형식은 암시적 변환 규칙에 따라 Amazon Redshift에 의해 변환될 수 있습니다.

integer

(선택) 어느 방향이든 정밀도의 소수점 자리 수를 나타내는 INTEGER입니다. integer를 지정하지 않으면 숫자가 정수로 잘립니다. integer를 지정하면 숫자가 지정한 소수점 자리에서 절사됩니다. SUPER 데이터 형식에는 지원되지 않습니다.

반환 타입

TRUNC는 첫 번째 입력 숫자와 동일한 숫자 데이터 형식을 반환합니다.

입력이 SUPER 형식이면 출력은 입력과 동일한 동적 형식을 유지하는 반면 정적 형식은 SUPER 형식을 유지합니다. SUPER의 동적 형식이 숫자가 아니면 Amazon Redshift는 NULL을 반환합니다.

예시

다음 예제 중 일부는 TICKIT 샘플 데이터베이스를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

주어진 판매 거래에 대해 지급된 수수료를 절사하려면 다음 예제를 사용합니다.

```
SELECT commission, TRUNC(commission)
FROM sales WHERE salesid=784;

+------------+-------+
| commission | trunc |
+------------+-------+
|     111.15 |   111 |
+------------+-------+
```

동일한 커미션 값을 소수점 첫째 자리에서 절사하려면 다음 예제를 사용합니다.

```
SELECT commission, TRUNC(commission,1)
FROM sales WHERE salesid=784;

+------------+-------+
| commission | trunc |
+------------+-------+
|     111.15 | 111.1 |
+------------+-------+
```

두 번째 인수의 음수 값으로 수수료를 절사하려면 다음 예제를 사용합니다. 111.15는 110으로 반올림됩니다.

```
SELECT commission, TRUNC(commission,-1)
FROM sales WHERE salesid=784;

+------------+-------+
| commission | trunc |
+------------+-------+
```

```
|     111.15 |   110 |
+------------+-------+
```

객체 함수

다음은 Amazon Redshift가 SUPER 유형 객체를 생성하기 위해 지원하는 SQL 객체 함수입니다.

주제

- LOWER_ATTRIBUTE_NAMES 함수
- OBJECT 함수
- OBJECT_TRANSFORM 함수
- UPPER_ATTRIBUTE_NAMES 함수

LOWER_ATTRIBUTE_NAMES 함수

LOWER 함수와 같은 대소문자 변환 루틴을 사용하여 SUPER 값의 모든 해당 속성 이름을 소문자로 변환합니다. LOWER_ATTRIBUTE_NAMES는 UTF-8 멀티바이트 문자를 지원하여 문자당 최대 4바이트까지 가능합니다.

SUPER 속성 이름을 대문자로 변환하려면 UPPER_ATTRIBUTE_NAMES 함수를 사용합니다.

구문

```
LOWER_ATTRIBUTE_NAMES(super_expression)
```

인수

super_expression

SUPER 표현식입니다.

반환 타입

SUPER

사용 노트

Amazon Redshift에서 열 식별자는 일반적으로 대소문자를 구분하지 않고 소문자로 변환됩니다. JSON과 같이 대소문자를 구분하는 데이터 형식의 데이터를 수집하는 경우 데이터에 대소문자가 혼합된 속성 이름이 포함될 수 있습니다.

다음 예제를 살펴보세요.

```
CREATE TABLE t1 (s) AS SELECT JSON_PARSE('{"AttributeName": "Value"}');


SELECT s.AttributeName FROM t1;

attributename
-------------
NULL


SELECT s."AttributeName" FROM t1;

attributename
-------------
NULL
```

Amazon Redshift는 두 쿼리 모두에 대해 NULL을 반환합니다. `AttributeName`을 쿼리하려면 LOWER_ATTRIBUTE_NAMES를 사용하여 데이터의 속성 이름을 소문자로 변환합니다. 다음 예제를 살펴보세요.

```
CREATE TABLE t2 (s) AS SELECT LOWER_ATTRIBUTE_NAMES(s) FROM t1;


SELECT s.attributename FROM t2;

attributename
-------------
"Value"


SELECT s.AttributeName FROM t2;

attributename
-------------
"Value"


SELECT s."attributename" FROM t2;

attributename
-------------
```

```
"Value"


SELECT s."AttributeName" FROM t2;

attributename
-------------
"Value"
```

대소문자가 혼합된 객체 속성 이름을 사용하기 위한 관련 옵션은 Amazon Redshift가 SUPER 속성 이름의 대소문자를 인식할 수 있도록 하는 enable_case_sensitive_super_attribute 구성 옵션입니다. LOWER_ATTRIBUTE_NAMES 대신 이 옵션을 사용할 수 있습니다. enable_case_sensitive_super_attribute에 대한 자세한 내용은 enable_case_sensitive_super_attribute 섹션을 참조하세요.

예시

SUPER 속성 이름을 소문자로 변환

다음 예제에서는 LOWER_ATTRIBUTE_NAMES를 사용하여 테이블에 있는 모든 SUPER 값의 속성 이름을 변환합니다.

```
-- Create a table and insert several SUPER values.
CREATE TABLE t (i INT, s SUPER);

INSERT INTO t VALUES
  (1, NULL),
  (2, 'A'::SUPER),
  (3, JSON_PARSE('{"AttributeName": "B"}')),
  (4, JSON_PARSE(
     '[{"Subobject": {"C": "C"},
       "Subarray": [{"D": "D"}, "E"]
      }]'));

-- Convert all attribute names to lowercase.
UPDATE t SET s = LOWER_ATTRIBUTE_NAMES(s);

SELECT i, s FROM t ORDER BY i;

 i |                         s
---+---------------------------------------------------
 1 | NULL
 2 | "A"
```

```
 3 | {"attributename":"B"}
 4 | [{"subobject":{"c":"C"},"subarray":[{"d":"D"}, "E"]}]
```

LOWER_ATTRIBUTE_NAMES가 어떻게 작동하는지 살펴보세요.

- NULL 값 및 스칼라 SUPER 값(예: "A")은 변경되지 않습니다.
- SUPER 객체에서는 모든 속성 이름이 소문자로 변경되지만 "B"와 같은 속성 값은 변경되지 않습니다.
- LOWER_ATTRIBUTE_NAMES는 SUPER 배열 또는 다른 객체 내에 중첩된 모든 SUPER 객체에 재귀적으로 적용됩니다.

속성 이름이 중복된 SUPER 객체에 LOWER_ATTRIBUTE_NAMES 사용

SUPER 객체에 이름의 대소문자만 다른 속성이 포함되어 있는 경우 LOWER_ATTRIBUTE_NAMES에서 오류가 발생합니다. 다음 예제를 살펴보세요.

```
SELECT LOWER_ATTRIBUTE_NAMES(JSON_PARSE('{"A": "A", "a": "a"}'));

error:   Invalid input
code:    8001
context: SUPER value has duplicate attributes after case conversion.
```

OBJECT 함수

SUPER 데이터 형식의 객체를 만듭니다.

구문

```
OBJECT ( [ key1, value1 ], [ key2, value2 ...] )
```

인수

key1, key2

VARCHAR 형식 문자열로 평가되는 표현식입니다.

value1, value2

Amazon Redshift는 datetime 형식을 SUPER 데이터 형식으로 캐스팅하지 않으므로 datetime 형식을 제외한 모든 Amazon Redshift 데이터 형식의 표현식입니다. datetime 형식에 대한 자세한 내용은 날짜/시간 형식 섹션을 참조하세요.

객체의 value 표현식들이 동일한 데이터 형식일 필요는 없습니다.

반환 타입

SUPER

예제

```
-- Creates an empty object.
select object();

object
--------
{}
(1 row)

-- Creates objects with different keys and values.
select object('a', 1, 'b', true, 'c', 3.14);

object
---------------------------
{"a":1,"b":true,"c":3.14}
(1 row)

select object('a', object('aa', 1), 'b', array(2,3), 'c', json_parse('{}'));

object
---------------------------------
{"a":{"aa":1},"b":[2,3],"c":{}}
(1 row)

-- Creates objects using columns from a table.
create table bar (k varchar, v super);
insert into bar values ('k1', json_parse('[1]')), ('k2', json_parse('{}'));
select object(k, v) from bar;

object
------------
{"k1":[1]}
{"k2":{}}
(2 rows)

-- Errors out because DATE type values can't be converted to SUPER type.
```

```
select object('k', '2008-12-31'::date);

ERROR:  OBJECT could not convert type date to super
```

OBJECT_TRANSFORM 함수

SUPER 객체를 변환합니다.

구문

```
OBJECT_TRANSFORM(
  input
  [KEEP path1, ...]
  [SET
    path1, value1,
    ...,  ...
  ]
)
```

인수

입력

SUPER 유형 객체로 해석되는 식입니다.

KEEP

이 절에 지정된 모든 경로 값이 유지되고 출력 객체로 전달됩니다.

이 절은 선택 사항입니다.

path1, path2, ...

마침표로 구분되고 큰따옴표로 묶인 경로 구성 요소 형식의 상수 문자열 리터럴입니다. 예를 들어, `'"a"."b"."c"'`는 유효한 경로 값입니다. 이는 KEEP 절과 SET 절의 경로 파라미터에 적용됩니다.

SET

경로와 값 쌍을 사용하여 기존 경로를 수정하거나 새 경로를 추가하고 출력 객체에 해당 경로의 값을 설정합니다.

이 절은 선택 사항입니다.

value1, value2, ...

SUPER 유형 값으로 해석되는 식입니다. 참고로 숫자, 텍스트, 부울 유형은 SUPER로 해석할 수 있습니다.

반환 타입

SUPER

사용 노트

OBJECT_TRANSFORM은 KEEP에 지정된 입력의 경로 값과 SET에 지정된 경로 및 값 쌍을 포함하는 SUPER 유형 객체를 반환합니다.

KEEP과 SET가 모두 비어 있는 경우 OBJECT_TRANSFORM은 입력을 반환합니다.

입력이 SUPER 유형 객체가 아닌 경우 OBJECT_TRANSFORM은 KEEP 또는 SET 값과 관계없이 입력을 반환합니다.

예제

다음 예시에서는 SUPER 객체를 다른 SUPER 객체로 변환합니다.

```
CREATE TABLE employees (
    col_person SUPER
);

INSERT INTO employees
VALUES
    (
        json_parse('
            {
                "name": {
                    "first": "John",
                    "last": "Doe"
                },
                "age": 25,
                "ssn": "111-22-3333",
                "company": "Company Inc.",
                "country": "U.S."
            }
        ')
    ),
```

```
    (
        json_parse('
            {
                "name": {
                    "first": "Jane",
                    "last": "Appleseed"
                },
                "age": 34,
                "ssn": "444-55-7777",
                "company": "Organization Org.",
                "country": "Ukraine"
            }
        ')
    )
;

SELECT
    OBJECT_TRANSFORM(
        col_person
        KEEP
            '"name"."first"',
            '"age"',
            '"company"',
            '"country"'
        SET
            '"name"."first"', UPPER(col_person.name.first::TEXT),
            '"age"', col_person.age + 5,
            '"company"', 'Amazon'
    ) AS col_person_transformed
FROM employees;

--This result is formatted for ease of reading.
                  col_person_transformed
-------------------------------------------------------------
{
    "name": {
        "first": "JOHN"
    },
    "age": 30,
    "company": "Amazon",
    "country": "U.S."
}
{
    "name": {
```

```
        "first": "JANE"
    },
    "age": 39,
    "company": "Amazon",
    "country": "Ukraine"
}
```

UPPER_ATTRIBUTE_NAMES 함수

UPPER 함수와 같은 대소문자 변환 루틴을 사용하여 SUPER 값의 모든 해당 속성 이름을 대문자로 변환합니다. UPPER_ATTRIBUTE_NAMES는 UTF-8 멀티바이트 문자를 지원하여 문자당 최대 4바이트까지 가능합니다.

SUPER 속성 이름을 소문자로 변환하려면 LOWER_ATTRIBUTE_NAMES 함수를 사용합니다.

구문

```
UPPER_ATTRIBUTE_NAMES(super_expression)
```

인수

super_expression

SUPER 표현식입니다.

반환 타입

SUPER

예시

SUPER 속성 이름을 대문자로 변환

다음 예제에서는 UPPER_ATTRIBUTE_NAMES를 사용하여 테이블에 있는 모든 SUPER 값의 속성 이름을 변환합니다.

```
-- Create a table and insert several SUPER values.
CREATE TABLE t (i INT, s SUPER);

INSERT INTO t VALUES
```

```
  (1, NULL),
  (2, 'a'::SUPER),
  (3, JSON_PARSE('{"AttributeName": "b"}')),
  (4, JSON_PARSE(
     '[{"Subobject": {"c": "c"},
       "Subarray": [{"d": "d"}, "e"]
      }]'));

-- Convert all attribute names to uppercase.
UPDATE t SET s = UPPER_ATTRIBUTE_NAMES(s);

SELECT i, s FROM t ORDER BY i;

 i |                         s
---+---------------------------------------------------
 1 | NULL
 2 | "a"
 3 | {"ATTRIBUTENAME":"B"}
 4 | [{"SUBOBJECT":{"C":"c"},"SUBARRAY":[{"D":"d"}, "e"]}]
```

UPPER_ATTRIBUTE_NAMES가 어떻게 작동하는지 살펴보세요.

- NULL 값 및 스칼라 SUPER 값(예: "a")은 변경되지 않습니다.
- SUPER 객체에서는 모든 속성 이름이 대문자로 변경되지만 "b"와 같은 속성 값은 변경되지 않습니다.
- UPPER_ATTRIBUTE_NAMES는 SUPER 배열 또는 다른 객체 내에 중첩된 모든 SUPER 객체에 재귀적으로 적용됩니다.

속성 이름이 중복된 SUPER 객체에 UPPER_ATTRIBUTE_NAMES 사용

SUPER 객체에 이름의 대소문자만 다른 속성이 포함되어 있는 경우 UPPER_ATTRIBUTE_NAMES에서 오류가 발생합니다. 다음 예제를 살펴보세요.

```
SELECT UPPER_ATTRIBUTE_NAMES(JSON_PARSE('{"A": "A", "a": "a"}'));

error:   Invalid input
code:    8001
context: SUPER value has duplicate attributes after case conversion.
```

공간 함수

지오메트리 객체 간의 관계는 DE-9IM(Dimensionally Extended 9-Intersection Model)을 기반으로 합니다. 이 모델은 같음, 포함, 덮임과 같은 조건자를 정의합니다. 공간 관계 정의에 대한 자세한 내용은 Wikipedia의 DE-9IM을 참조하십시오.

Amazon Redshift에서 공간 데이터를 사용하는 방법에 대한 자세한 내용은 Amazon Redshift에서 공간 데이터 쿼리 단원을 참조하세요.

Amazon Redshift는 `GEOMETRY` 및 `GEOGRAPHY` 데이터 유형과 함께 작동하는 공간 함수를 제공합니다. 다음은 `GEOGRAPHY` 데이터 유형을 지원하는 함수 목록입니다.

- ST_Area
- ST_AsEWKT
- ST_AsGeoJSON
- ST_AsHexEWKB
- ST_AsHexWKB
- ST_AsText
- ST_Distance
- ST_GeogFromText
- ST_GeogFromWKB
- ST_Length
- ST_NPoints
- ST_Perimeter

다음은 Amazon Redshift에서 지원하는 전체 공간 함수 집합 목록입니다.

주제

- AddBBox
- DropBBox
- GeometryType
- H3_FromLongLat
- H3_FromPoint

- H3_Polyfill
- ST_AddPoint
- ST_Angle
- ST_Area
- ST_AsBinary
- ST_AsEWKB
- ST_AsEWKT
- ST_AsGeoJSON
- ST_AsHexWKB
- ST_AsHexEWKB
- ST_AsText
- ST_Azimuth
- ST_Boundary
- ST_Buffer
- ST_Centroid
- ST_Collect
- ST_Contains
- ST_ContainsProperly
- ST_ConvexHull
- ST_CoveredBy
- ST_Covers
- ST_Crosses
- ST_Dimension
- ST_Disjoint
- ST_Distance
- ST_DistanceSphere
- ST_DWithin
- ST_EndPoint

- ST_Envelope
- ST_Equals
- ST_ExteriorRing
- ST_Force2D
- ST_Force3D
- ST_Force3DM
- ST_Force3DZ
- ST_Force4D
- ST_GeoHash
- ST_GeogFromText
- ST_GeogFromWKB
- ST_GeometryN
- ST_GeometryType
- ST_GeomFromEWKB
- ST_GeomFromEWKT
- ST_GeomFromGeoHash
- ST_GeomFromGeoJSON
- ST_GeomFromGeoSquare
- ST_GeomFromText
- ST_GeomFromWKB
- ST_GeoSquare
- ST_InteriorRingN
- ST_Intersects
- ST_Intersection
- ST_IsPolygonCCW
- ST_IsPolygonCW
- ST_IsClosed
- ST_IsCollection
- ST_IsEmpty

- ST_IsRing
- ST_IsSimple
- ST_IsValid
- ST_Length
- ST_LengthSphere
- ST_Length2D
- ST_LineFromMultiPoint
- ST_LineInterpolatePoint
- ST_M
- ST_MakeEnvelope
- ST_MakeLine
- ST_MakePoint
- ST_MakePolygon
- ST_MemSize
- ST_MMax
- ST_MMin
- ST_Multi
- ST_NDims
- ST_NPoints
- ST_NRings
- ST_NumGeometries
- ST_NumInteriorRings
- ST_NumPoints
- ST_Perimeter
- ST_Perimeter2D
- ST_Point
- ST_PointN
- ST_Points
- ST_Polygon

- ST_RemovePoint
- ST_Reverse
- ST_SetPoint
- ST_SetSRID
- ST_Simplify
- ST_SRID
- ST_StartPoint
- ST_Touches
- ST_Transform
- ST_Union
- ST_Within
- ST_X
- ST_XMax
- ST_XMin
- ST_Y
- ST_YMax
- ST_YMin
- ST_Z
- ST_ZMax
- ST_ZMin
- SupportsBBox

AddBBox

AddBBox는 미리 계산된 경계 상자로 인코딩을 지원하는 입력 지오메트리의 복사본을 반환합니다. 경계 상자 지원에 대한 자세한 내용은 경계 상자 섹션을 참조하세요.

구문

```
AddBBox(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

GEOMETRY

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 경계 상자로 인코딩되는 것을 지원하는 입력 다각형 지오메트리의 복사본을 반환합니다.

```
SELECT ST_AsText(AddBBox(ST_GeomFromText('POLYGON((0 0,1 0,0 1,0 0))')));
```

```
 st_astext
----------
 POLYGON((0 0,1 0,0 1,0 0))
```

DropBBox

DropBBox는 미리 계산된 경계 상자로 인코딩을 지원하지 않는 입력 지오메트리의 복사본을 반환합니다. 경계 상자 지원에 대한 자세한 내용은 경계 상자 섹션을 참조하세요.

구문

```
DropBBox(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

GEOMETRY

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 경계 상자로 인코딩되는 것을 지원하지 않는 입력 다각형 지오메트리의 복사본을 반환합니다.

```
SELECT ST_AsText(DropBBox(ST_GeomFromText('POLYGON((0 0,1 0,0 1,0 0))')));
```

```
 st_astext
----------
 POLYGON((0 0,1 0,0 1,0 0))
```

GeometryType

GeometryType은 입력 지오메트리의 하위 유형을 문자열로 반환합니다.

구문

```
GeometryType(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

geom의 하위 유형을 나타내는 VARCHAR입니다.

geom이 null이면 null이 반환됩니다.

값은 다음과 같이 반환됩니다.

2D, 3DZ, 4D 지오메트리에 대해 반환된 문자열 값	3DM 지오메트리에 대해 반환된 문자열 값	지오메트리 하위 유형
POINT	POINTM	geom이 POINT 하위 유형인 경우 반환됩니다.

2D, 3DZ, 4D 지오메트리에 대해 반환된 문자열 값	3DM 지오메트리에 대해 반환된 문자열 값	지오메트리 하위 유형
LINESTRING	LINESTRINGM	geom이 LINESTRING 하위 유형인 경우 반환됩니다.
POLYGON	POLYGONM	geom이 POLYGON 하위 유형인 경우 반환됩니다.
MULTIPOINT	MULTIPOINTM	geom이 MULTIPOINT 하위 유형인 경우 반환됩니다.
MULTILINESTRING	MULTILINESTRINGM	geom이 MULTILINESTRING 하위 유형인 경우 반환됩니다.
MULTIPOLYGON	MULTIPOLYGONM	geom이 MULTIPOLYGON 하위 유형인 경우 반환됩니다.
GEOMETRYCOLLECTION	GEOMETRYCOLLECTIONM	geom이 GEOMETRYCOLLECTION 하위 유형인 경우 반환됩니다.

예시

다음 SQL은 다각형의 WKT(Well-Known Text) 표현을 변환하고 GEOMETRY 하위 유형을 문자열로 반환합니다.

```
SELECT GeometryType(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'));
```

```
geometrytype
-------------
 POLYGON
```

H3_FromLongLat

H3_FromLonglat은 입력된 경도, 위도 및 해상도에 따라 해당하는 H3 셀 ID를 반환합니다. H3 인덱싱에 대한 자세한 내용은 H3 섹션을 참조하세요.

구문

```
H3_FromLongLat(longitude, lattitude, resolution)
```

인수

longitude

DOUBLE PRECISION 데이터 형식의 값 또는 DOUBLE PRECISION 형식으로 계산되는 표현식입니다.

latitude

DOUBLE PRECISION 데이터 형식의 값 또는 DOUBLE PRECISION 형식으로 계산되는 표현식입니다.

resolution

INTEGER 데이터 유형의 값 또는 INTEGER 유형으로 계산되는 식입니다. 값은 H3 그리드 시스템의 해상도를 나타냅니다. 값은 0에서 15까지의 정수여야 합니다. 0이 가장 거칠고 15가 가장 정교합니다.

반환 타입

BIGINT - H3 셀 ID를 나타냅니다.

해상도가 범위를 벗어나면 오류가 반환됩니다.

예시

다음 SQL은 경도 0, 위도 0 및 해상도 10에 대한 H3 셀 ID를 반환합니다.

```
SELECT H3_FromLongLat(0, 0, 10);
```

```
 h3_fromlonglat
-------------------
 623560421467684863
```

H3_FromPoint

H3_FromPoint는 입력된 지오메트리 포인트 및 해상도에 따라 해당하는 H3 셀 ID를 반환합니다. H3 인덱싱에 대한 자세한 내용은 H3 섹션을 참조하세요.

구문

```
H3_FromPoint(geom, resolution)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. geom은 POINT여야 합니다.

resolution

INTEGER 데이터 유형의 값 또는 INTEGER 유형으로 계산되는 식입니다. 값은 H3 그리드 시스템의 해상도를 나타냅니다. 값은 0에서 15까지의 정수여야 합니다. 0이 가장 거칠고 15가 가장 정교합니다.

반환 타입

BIGINT - H3 셀 ID를 나타냅니다.

geom이 POINT이 아니면 오류가 반환됩니다.

해상도가 범위를 벗어나면 오류가 반환됩니다.

geom이 비어 있으면 NULL이 반환됩니다.

예시

다음 SQL은 포인트 0,0 및 해상도 10에 대한 H3 셀 ID를 반환합니다.

```
SELECT H3_FromPoint(ST_GeomFromText('POINT(0 0)'), 10);
```

```
 h3_frompoint
--------------------
```

```
623560421467684863
```

H3_Polyfill

H3_Polyfill은 주어진 해상도의 입력 다각형에 포함된 육각형 및 오각형에 해당하는 H3 셀 ID를 반환합니다. H3 인덱싱에 대한 자세한 내용은 H3 섹션을 참조하세요.

구문

```
H3_Polyfill(geom, resolution)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. geom은 POLYGON여야 합니다.

resolution

INTEGER 데이터 유형의 값 또는 INTEGER 유형으로 계산되는 식입니다. 값은 H3 그리드 시스템의 해상도를 나타냅니다. 값은 0에서 15까지의 정수여야 합니다. 0이 가장 거칠고 15가 가장 정교합니다.

반환 타입

SUPER - H3 셀 ID 목록을 나타냅니다.

geom이 POLYGON이 아니면 오류가 반환됩니다.

해상도가 범위를 벗어나면 오류가 반환됩니다.

geom이 비어 있으면 NULL이 반환됩니다.

예시

다음 SQL은 다각형과 해상도 4에 대해 H3 셀 ID로 구성된 SUPER 데이터 유형 배열을 반환합니다.

```
SELECT H3_Polyfill(ST_GeomFromText('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))'), 4);
```

```
 h3_polyfill
-------------------------------------------------------------------------------------------
 [596538848238895103,596538805289222143,596538856828829695,596538813879156735,5965379205259591
```

ST_AddPoint

ST_AddPoint는 점이 추가된 입력 지오메트리와 동일한 라인스트링 지오메트리를 반환합니다. 인덱스가 제공되면 인덱스 위치에 점이 추가됩니다. 인덱스가 -1이거나 제공되지 않으면 점이 라인스트링에 추가됩니다.

인덱스는 0부터 시작합니다. 결과의 SRID(공간 참조 시스템 식별자)는 입력 지오메트리의 값과 동일합니다.

반환된 지오메트리의 차원은 geom1 값의 차원과 같습니다. geom1과 geom2의 차원이 다른 경우 geom2는 geom1의 차원에 나타납니다.

구문

```
ST_AddPoint(geom1, geom2)
```

```
ST_AddPoint(geom1, geom2, index)
```

인수

geom1

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 LINESTRING이어야 합니다.

geom2

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 POINT이어야 합니다. 점은 빈 점일 수 있습니다.

인덱스를 구축하고 배포할 것입니다

0부터 시작하는 인덱스의 위치를 나타내는 INTEGER 데이터 형식의 값입니다.

반환 타입

GEOMETRY

geom1, geom2 또는 index가 null이면 null이 반환됩니다.

geom2가 빈 점이면 geom1의 복사본이 반환됩니다.

geom1이 LINESTRING이 아니면 오류가 반환됩니다.

geom2가 POINT가 아니면 오류가 반환됩니다.

index가 범위를 벗어나면 오류가 반환됩니다. 인덱스 위치에 대해 유효한 값은 -1 또는 0과 ST_NumPoints(geom1) 사이의 값입니다.

예시

다음 SQL은 닫힌 라인스트링으로 만들도록 라인스트링에 점을 추가합니다.

```
WITH tmp(g) AS (SELECT ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0 5)',4326))
SELECT ST_AsEWKT(ST_AddPoint(g, ST_StartPoint(g))) FROM tmp;
```

```
 st_asewkt
-------------------------------------------------
 SRID=4326;LINESTRING(0 0,10 0,10 10,5 5,0 5,0 0)
```

다음 SQL은 라인스트링의 특정 위치에 점을 추가합니다.

```
WITH tmp(g) AS (SELECT ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0 5)',4326))
SELECT ST_AsEWKT(ST_AddPoint(g, ST_SetSRID(ST_Point(5, 10), 4326), 3)) FROM tmp;
```

```
 st_asewkt
-------------------------------------------------
 SRID=4326;LINESTRING(0 0,10 0,10 10,5 10,5 5,0 5)
```

ST_Angle

ST_Angle은 다음과 같이 시계 방향으로 측정된 점 사이의 각도를 라디안 단위로 반환합니다.

- 3개의 점이 입력되면 P1에서 P3까지 P2를 중심으로 시계 방향으로 회전하여 각도를 얻은 것처럼 반환된 각도 P1-P2-P3이 측정됩니다.

- 4개의 점이 입력되면 방향선 P1-P2 및 P3-P4에 의해 형성된 반환된 시계 방향 각도가 반환됩니다. 입력이 퇴화되면(즉, P1이 P2와 같거나 P3이 P4와 같음) null이 반환됩니다.

반환 값은 라디안 단위이며 범위는 [0, 2π)입니다.

ST_Angle은 입력 지오메트리의 2D 프로젝션에서 작동합니다.

구문

```
ST_Angle(geom1, geom2, geom3)
```

```
ST_Angle(geom1, geom2, geom3, geom4)
```

인수

geom1

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 POINT이어야 합니다.

geom2

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 POINT이어야 합니다.

geom3

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 POINT이어야 합니다.

geom4

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 POINT이어야 합니다.

반환 타입

DOUBLE PRECISION.

geom1이 geom2와 같거나 geom2가 geom3과 같으면 null이 반환됩니다.

geom1, geom2, geom3 또는 geom4가 null이면 null이 반환됩니다.

geom1, geom2, geom3, geom4중 하나라도 빈 점이면 오류가 반환됩니다.

geom1, geom2, geom3, geom4의 SRID(공간 참조 시스템 식별자) 값이 같지 않으면 오류가 반환됩니다.

예시

다음 SQL은 세 입력 점의 각도로 변환된 각도를 반환합니다.

```
SELECT ST_Angle(ST_Point(1,1), ST_Point(0,0), ST_Point(1,0)) / Pi() * 180.0 AS angle;
```

```
 angle
---------------
    45
```

다음 SQL은 네 입력 점의 각도로 변환된 각도를 반환합니다.

```
SELECT ST_Angle(ST_Point(1,1), ST_Point(0,0), ST_Point(1,0), ST_Point(2,0)) / Pi() *
 180.0 AS angle;
```

```
 angle
---------------
   225
```

ST_Area

입력 기하학의 경우 ST_Area는 2D 프로젝션의 데카르트 영역을 반환합니다. 영역 단위는 입력 지오메트리의 좌표가 표현되는 단위와 동일합니다. 점, 라인스트링, 다중 점 및 다중 라인스트링의 경우 이 함수는 0을 반환합니다. 지오메트리 컬렉션의 경우 컬렉션에 있는 지오메트리 영역의 합계를 반환합니다.

입력 지오그래피의 경우 ST_Area는 SRID에 의해 결정된 회전 타원체에서 계산된 입력 영역 지오그래피의 2D 투영의 측지 영역을 반환합니다. 길이 단위는 평방 미터입니다. 이 함수는 점, 다중 점 및 선형 지오그래피에 대해 0을 반환합니다. 입력이 지오메트리 컬렉션인 경우 함수는 컬렉션에 있는 영역 지오그래피 영역의 합계를 반환합니다.

구문

```
ST_Area(geo)
```

인수

geo

GEOMETRY 또는 GEOGRAPHY 데이터 유형의 값이나 GEOMETRY 또는 GEOGRAPHY 유형으로 계산되는 표현식입니다.

반환 타입

DOUBLE PRECISION

geo가 null이면 null이 반환됩니다.

예시

다음 SQL은 다중 다각형의 데카르트 영역을 반환합니다.

```
SELECT ST_Area(ST_GeomFromText('MULTIPOLYGON(((0 0,10 0,0 10,0 0)),((10 0,20 0,20 10,10
 0)))'));
```

```
 st_area
---------
     100
```

다음 SQL은 지오그래피에서 다각형의 영역을 반환합니다.

```
SELECT ST_Area(ST_GeogFromText('polygon((34 35, 28 30, 25 34, 34 35))'));
```

```
     st_area
------------------
 201824655743.383
```

다음 SQL은 선형 지오그래피에 대해 0을 반환합니다.

```
SELECT ST_Area(ST_GeogFromText('multipoint(0 0, 1 1, -21.32 121.2)'));
```

```
 st_area
---------
       0
```

ST_AsBinary

ST_AsBinary는 입력 지오메트리의 16진수 WKB(Well-known Binary) 표현을 반환합니다. 3DZ, 3DM 및 4D 지오메트리의 경우 ST_AsBinary는 지오메트리 유형에 대해 OGC(Open Geospatial Consortium) 표준 값을 사용합니다.

구문

```
ST_AsBinary(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

VARBYTE

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 다각형의 16진수 WKB 표현을 반환합니다.

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

```
st_asbinary
--------------------------------
```

```
01030000000100000005000000000000000000000000000000000000000000000000000000000000000000000000000000F03F00000
```

ST_AsEWKB

ST_AsEWKT는 입력 지오메트리의 EWKB(well-known binary) 표현을 반환합니다. 3DZ, 3DM 및 4D 지오메트리의 경우 ST_AsEWKB는 지오메트리 유형에 대해Open Geospatial Consortium(OGC) 표준 값을 사용합니다.

구문

```
ST_AsEWKB(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

VARBYTE

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 다각형의 16진수 EWKB 표현을 반환합니다.

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

```
st_asewkb
--------------------------------
0103000020E6100000010000000500000000000000000000000000000000000000000000000000000000000000000000000000000000F
```

ST_AsEWKT

ST_AsEWKT는 입력 지오메트리 또는 지오그래피의 EWKT(Extended Well-Known Text) 표현을 반환합니다. 3DZ, 3DM 및 4D 지오메트리의 경우 ST_AsEWKT는 지오메트리 유형의 WKT 값에 Z, M 또는 ZM을 추가합니다.

구문

```
ST_AsEWKT(geo)
```

```
ST_AsEWKT(geo, precision)
```

인수

geo

GEOMETRY 또는 GEOGRAPHY 데이터 유형의 값이나 GEOMETRY 또는 GEOGRAPHY 유형으로 계산되는 표현식입니다.

precision

INTEGER 데이터 형식의 값입니다. 지오메트리의 경우 geo의 좌표는 지정된 정밀도 1~20을 사용하여 표시됩니다. precision이 지정되지 않은 경우 기본값은 15입니다. 지오그래피의 경우 geo의 좌표는 지정된 정밀도를 사용하여 표시됩니다. precision이 지정되지 않은 경우 기본값은 15입니다.

반환 타입

VARCHAR

geo가 null이면 null이 반환됩니다.

precision이 null이면 null이 반환됩니다.

결과가 64KB VARCHAR보다 크면 오류가 반환됩니다.

예시

다음 SQL은 라인스트링의 EWKT 표현을 반환합니다.

```
SELECT ST_AsEWKT(ST_GeomFromText('LINESTRING(3.141592653589793
 -6.283185307179586,2.718281828459045 -1.414213562373095)', 4326));
```

```
st_asewkt
--------------------------------
```

```
 SRID=4326;LINESTRING(3.14159265358979 -6.28318530717959,2.71828182845905
 -1.41421356237309)
```

다음 SQL은 라인스트링의 EWKT 표현을 반환합니다. 지오메트리의 좌표는 6자리의 정밀도로 표시됩니다.

```
SELECT ST_AsEWKT(ST_GeomFromText('LINESTRING(3.141592653589793
 -6.283185307179586,2.718281828459045 -1.414213562373095)', 4326), 6);
```

```
st_asewkt
---------------------------------
 SRID=4326;LINESTRING(3.14159 -6.28319,2.71828 -1.41421)
```

다음 SQL은 지오그래피의 EWKT 표현을 반환합니다.

```
SELECT ST_AsEWKT(ST_GeogFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)'));
```

```
                  st_asewkt
-----------------------------------------------
 SRID=4326;LINESTRING(110 40,2 3,-10 80,-7 9)
```

ST_AsGeoJSON

ST_AsGeoJSON은 입력 지오메트리 또는 지오그래피의 GeoJSON 표현을 반환합니다. GeoJSON에 대한 자세한 내용은 Wikipedia의 GeoJSON을 참조하십시오.

3DZ 및 4D 지오메트리의 경우 출력 지오메트리는 입력 3DZ 또는 4D 지오메트리의 3DZ 프로젝션입니다. 즉, x, y 및 z 좌표가 출력에 있습니다. 3DM 지오메트리의 경우 출력 지오메트리는 입력 3DM 지오메트리의 2D 프로젝션입니다. 즉, x 및 y 좌표만 출력에 있습니다.

입력 지오그래피의 경우 ST_AsGeoJSON은 입력 지오그래피의 GeoJSON 표현을 반환합니다. 지오그래피의 좌표는 지정된 정밀도를 사용하여 표시됩니다.

구문

```
ST_AsGeoJSON(geo)
```

```
ST_AsGeoJSON(geo, precision)
```

인수

geo

GEOMETRY 또는 GEOGRAPHY 데이터 유형의 값이나 GEOMETRY 또는 GEOGRAPHY 유형으로 계산되는 표현식입니다.

precision

INTEGER 데이터 형식의 값입니다. 지오메트리의 경우 geo의 좌표는 지정된 정밀도 1~20을 사용하여 표시됩니다. precision이 지정되지 않은 경우 기본값은 15입니다. 지오그래피의 경우 geo의 좌표는 지정된 정밀도를 사용하여 표시됩니다. precision이 지정되지 않은 경우 기본값은 15입니다.

반환 타입

VARCHAR

geo가 null이면 null이 반환됩니다.

precision이 null이면 null이 반환됩니다.

결과가 64KB VARCHAR보다 크면 오류가 반환됩니다.

예시

다음 SQL은 라인스트링의 GeoJSON 표현을 반환합니다.

```
SELECT ST_AsGeoJSON(ST_GeomFromText('LINESTRING(3.141592653589793
 -6.283185307179586,2.718281828459045 -1.414213562373095)'));
```

```
st_asgeojson
--------------------------------------------------------------------------------------------
 {"type":"LineString","coordinates":[[3.14159265358979,-6.28318530717959],
[2.71828182845905,-1.41421356237309]]}
```

다음 SQL은 라인스트링의 GeoJSON 표현을 반환합니다. 지오메트리의 좌표는 6자리의 정밀도로 표시됩니다.

```
SELECT ST_AsGeoJSON(ST_GeomFromText('LINESTRING(3.141592653589793
 -6.283185307179586,2.718281828459045 -1.414213562373095)'), 6);
```

```
st_asgeojson
--------------------------------------------------------------------------------
  {"type":"LineString","coordinates":[[3.14159,-6.28319],[2.71828,-1.41421]]}
```

다음 SQL은 지오그래피의 GeoJSON 표현을 반환합니다.

```
SELECT ST_AsGeoJSON(ST_GeogFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)'));
```

```
                              st_asgeojson
---------------------------------------------------------------------------
 {"type":"LineString","coordinates":[[110,40],[2,3],[-10,80],[-7,9]]}
```

ST_AsHexWKB

ST_AsHexWKB는 ASCII 16진수 문자(0~9, A~F)를 사용하여 입력 지오메트리 또는 지오그래피의 16진수 WKB(Well-known Binary) 표현을 반환합니다. 3DZ, 3DM 및 4D 지오메트리 또는 지오그래피의 경우 ST_AsHexWKB는 지오메트리 또는 지오그래피 유형에 대해 OGC(Open Geospatial Consortium) 표준 값을 사용합니다.

구문

```
ST_AsHexWKB(geo)
```

인수

geo

GEOMETRY 또는 GEOGRAPHY 데이터 유형의 값이나 GEOMETRY 또는 GEOGRAPHY 유형으로 계산되는 표현식입니다.

반환 타입

VARCHAR

geo가 null이면 null이 반환됩니다.

결과가 64KB VARCHAR보다 크면 오류가 반환됩니다.

예시

다음 SQL은 지오메트리에서 다각형의 16진수 WKB 표현을 반환합니다.

```
SELECT ST_AsHexWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

```
st_ashexwkb
--------------------------------
0103000000010000000500000000000000000000000000000000000000000000000000F03F00000
```

다음 SQL은 지오그래피에서 다각형의 16진수 WKB 표현을 반환합니다.

```
SELECT ST_AsHexWKB(ST_GeogFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))'));
```

```
st_ashexwkb
--------------------------------
0103000000010000000500000000000000000000000000000000000000000000000000F03F00000
```

ST_AsHexEWKB

ST_AsHexEWKB는 ASCII 16진수 문자(0~9, A~F)를 사용하여 입력 지오메트리 또는 지오그래피의 16진수 EWKB(Extended Well-Known Binary) 표현을 반환합니다. 3DZ, 3DM 및 4D 지오메트리 또는 지오그래피의 경우 ST_AsHexEWKB는 지오메트리또는 지오그래피 유형에 대해 PostGIS 확장 WKB 값을 사용합니다.

구문

```
ST_AsHexEWKB(geo)
```

인수

geo

GEOMETRY 또는 GEOGRAPHY 데이터 유형의 값이나 GEOMETRY 또는 GEOGRAPHY 유형으로 계산되는 표현식입니다.

반환 타입

VARCHAR

geo가 null이면 null이 반환됩니다.

결과가 64KB VARCHAR보다 크면 오류가 반환됩니다.

예시

다음 SQL은 지오메트리에서 다각형의 16진수 EWKB 표현을 반환합니다.

```
SELECT ST_AsHexEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

```
st_ashexewkb
---------------------------------
0103000020E6100000010000000500000000000000000000000000000000000000000000000000F03F000000000000F03F000000000000F03F000000000000F03F0000000000000000000000000000000000000000000000F
```

다음 SQL은 지오그래피에서 다각형의 16진수 EWKB 표현을 반환합니다.

```
SELECT ST_AsHexEWKB(ST_GeogFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))'));
```

```
st_ashexewkb
---------------------------------
0103000020E6100000010000000500000000000000000000000000000000000000000000000000F03F000000000000F03F000000000000F03F000000000000F03F0000000000000000000000000000000000000000000000F
```

ST_AsText

ST_AsText는 입력 지오메트리 또는 지오그래피의 WKT(Well-Known Text) 표현을 반환합니다. 3DZ, 3DM 및 4D 지오메트리 또는 지오그래피의 경우 ST_AsEWKT는 지오메트리 또는 지오그래피 유형의 WKT 값에 Z, M 또는 ZM을 추가합니다.

구문

```
ST_AsText(geo)
```

```
ST_AsText(geo, precision)
```

인수

geo

GEOMETRY 또는 GEOGRAPHY 데이터 유형의 값이나 GEOMETRY 또는 GEOGRAPHY 유형으로 계산되는 표현식입니다.

precision

INTEGER 데이터 형식의 값입니다. 지오메트리의 경우 geo의 좌표는 지정된 정밀도 1~20을 사용하여 표시됩니다. precision이 지정되지 않은 경우 기본값은 15입니다. 지오그래피의 경우 geo의 좌표는 지정된 정밀도를 사용하여 표시됩니다. precision이 지정되지 않은 경우 기본값은 15입니다.

반환 타입

VARCHAR

geo가 null이면 null이 반환됩니다.

precision이 null이면 null이 반환됩니다.

결과가 64KB VARCHAR보다 크면 오류가 반환됩니다.

예시

다음 SQL은 라인스트링의 WKT 표현을 반환합니다.

```
SELECT ST_AsText(ST_GeomFromText('LINESTRING(3.141592653589793
 -6.283185307179586,2.718281828459045 -1.414213562373095)', 4326));
```

```
st_astext
--------------------------------
LINESTRING(3.14159265358979 -6.28318530717959,2.71828182845905 -1.41421356237309)
```

다음 SQL은 라인스트링의 WKT 표현을 반환합니다. 지오메트리의 좌표는 6자리의 정밀도로 표시됩니다.

```
SELECT ST_AsText(ST_GeomFromText('LINESTRING(3.141592653589793
 -6.283185307179586,2.718281828459045 -1.414213562373095)', 4326), 6);
```

```
st_astext
---------------------------------------------
 LINESTRING(3.14159 -6.28319,2.71828 -1.41421)
```

다음 SQL은 지오그래피의 WKT 표현을 반환합니다.

```
SELECT ST_AsText(ST_GeogFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)'));
```

```
             st_astext
-----------------------------------
 LINESTRING(110 40,2 3,-10 80,-7 9)
```

ST_Azimuth

ST_Azimuth는 두 입력 점의 2D 프로젝션을 사용하여 북쪽 기반 직교 방위각을 반환합니다.

구문

```
ST_Azimuth(point1, point2)
```

인수

point1

GEOMETRY 데이터 형식의 POINT 값입니다. point1의 SRID(공간 참조 시스템 식별자)는 point2의 SRID와 일치해야 합니다.

point2

GEOMETRY 데이터 형식의 POINT 값입니다. point2의 SRID는 point1의 SRID와 일치해야 합니다.

반환 타입

DOUBLE PRECISION 데이터 형식의 라디안 각도입니다. 값의 범위는 0(포함)부터 2pi(제외)까지입니다.

point1 또는 point2가 빈 점이 아니면 오류가 반환됩니다.

point1 또는 point2가 null이면 null이 반환됩니다.

point1과 point2이 같으면 null이 반환됩니다.

point1 또는 point2가 점이 아니면 오류가 반환됩니다.

point1과 point2에 공간 참조 시스템 식별자(SRID) 값이 없으면 오류가 반환됩니다.

예시

다음 SQL은 입력 지점의 방위각을 반환합니다.

```
SELECT ST_Azimuth(ST_Point(1,2), ST_Point(5,6));
```

```
st_azimuth
-------------------
 0.7853981633974483
```

ST_Boundary

ST_Boundine은 다음과 같이 입력 지오메트리의 경계를 반환합니다.

- 입력 지오메트리가 비어 있으면(즉, 점이 없음) 있는 그대로 반환됩니다.
- 입력 지오메트리가 점이거나 비어 있지 않은 다중 점이면 빈 지오메트리 컬렉션이 반환됩니다.
- 입력이 라인스트링 또는 다중 라인스트링이면 경계의 모든 점을 포함하는 다중 점이 반환됩니다. 다중 점은 비어 있을 수 있습니다.
- 입력이 내부 링이 없는 다각형이면 경계를 나타내는 닫힌 라인스트링이 반환됩니다.
- 입력이 내부 링이 있는 다각형이거나 다중 다각형인 경우 다중 라인스트링이 반환됩니다. 다중 라인스트링 문자열에는 영역 지오메트리에 있는 모든 링의 모든 경계가 닫힌 라인스트링으로 포함됩니다.

점 동등성을 판별하기 위해 ST_Boundary는 입력 지오메트리의 2D 프로젝션에서 작동합니다. 입력 지오메트리가 비어 있으면 해당 복사본이 입력과 동일한 차원으로 반환됩니다. 비어 있지 않은 3DM 및 4D 지오메트리의 경우 m 좌표가 삭제됩니다. 특수한 경우의 3DZ 및 4D 다중 라인스트링에서 다중 라인스트링 경계 점의 z 좌표는 동일한 2D 프로젝션을 사용하는 라인스트링 경계 점의 고유한 z 값의 평균으로 계산됩니다.

구문

```
ST_Boundary(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

GEOMETRY

geom이 null이면 null이 반환됩니다.

geom이 GEOMETRYCOLLECTION이 아니면 오류가 반환됩니다.

예시

다음 SQL은 입력 다각형의 경계를 다중 라인스트링으로 반환합니다.

```
SELECT ST_AsEWKT(ST_Boundary(ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(1 1,1
 2,2 1,1 1))')));
```

```
st_asewkt
--------------------
 MULTILINESTRING((0 0,10 0,10 10,0 10,0 0),(1 1,1 2,2 1,1 1))
```

ST_Buffer

ST_Buffer는 xy 데카르트 평면에 투영된 입력 형상으로부터의 거리가 입력 거리보다 작거나 같은 모든 점을 나타내는 2D 형상을 반환합니다.

구문

```
ST_Buffer(geom, distance)
```

```
ST_Buffer(geom, distance, number_of_segments_per_quarter_circle)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

distance

버퍼의 거리(또는 반지름)를 나타내는 데이터 형식 DOUBLE PRECISION 값입니다.

number_of_segments_per_quarter_circle

INTEGER 데이터 형식의 값입니다. 이 값은 입력 형상의 각 꼭짓점 주위의 1/4 원을 근사화하는 점의 수를 결정합니다. 음수 값은 기본적으로 0입니다. 기본값은 8입니다.

반환 타입

GEOMETRY

ST_Buffer 함수는 xy 데카르트 평면에서 2차원(2D) 형상을 반환합니다.

geom이 GEOMETRYCOLLECTION이 아니면 오류가 반환됩니다.

예시

다음 SQL은 입력 라인스트링의 버퍼를 반환합니다.

```
SELECT ST_AsEwkt(ST_Buffer(ST_GeomFromText('LINESTRING(1 2,5 2,5 8)'), 2));
```

```
               st_asewkt
POLYGON((-1 2,-0.96157056080646 2.39018064403226,-0.847759065022573
 2.76536686473018,-0.662939224605089 3.11114046603921,-0.414213562373093
 3.4142135623731,-0.111140466039201 3.66293922460509,0.234633135269824
 3.84775906502257,0.609819355967748 3.96157056080646,1 4,3 4,3 8,3.03842943919354
 8.39018064403226,3.15224093497743 8.76536686473018,3.33706077539491
 9.11114046603921,3.58578643762691 9.4142135623731,3.8888595339608
 9.66293922460509,4.23463313526982 9.84775906502257,4.60981935596775
 9.96157056080646,5 10,5.39018064403226 9.96157056080646,5.76536686473018
 9.84775906502257,6.11114046603921 9.66293922460509,6.4142135623731
 9.41421356237309,6.66293922460509 9.1111404660392,6.84775906502258
 8.76536686473017,6.96157056080646 8.39018064403225,7 8,7 2,6.96157056080646
 1.60981935596774,6.84775906502257 1.23463313526982,6.66293922460509
 0.888859533960796,6.41421356237309 0.585786437626905,6.1111404660392
```

```
 0.33706077539491,5.76536686473018 0.152240934977427,5.39018064403226
 0.0384294391935391,5 0,1 0,0.609819355967744 0.0384294391935391,0.234633135269821
 0.152240934977427,-0.111140466039204 0.337060775394909,-0.414213562373095
 0.585786437626905,-0.662939224605091 0.888859533960796,-0.847759065022574
 1.23463313526982,-0.961570560806461 1.60981935596774,-1 2))
```

다음 SQL은 원을 근사화하는 입력 점 형상의 버퍼를 반환합니다. 이 명령은 1/4 원당 세그먼트 수를 지정하지 않으므로 함수에서는 기본값인 8개 세그먼트를 사용하여 1/4 원을 근사화합니다.

```
SELECT ST_AsEwkt(ST_Buffer(ST_GeomFromText('POINT(3 4)'), 2));
```

```
               st_asewkt
POLYGON((1 4,1.03842943919354 4.39018064403226,1.15224093497743
 4.76536686473018,1.33706077539491 5.11114046603921,1.58578643762691
 5.4142135623731,1.8888595339608 5.66293922460509,2.23463313526982
 5.84775906502257,2.60981935596775 5.96157056080646,3 6,3.39018064403226
 5.96157056080646,3.76536686473019 5.84775906502257,4.11114046603921
 5.66293922460509,4.4142135623731 5.41421356237309,4.66293922460509
 5.1111404660392,4.84775906502258 4.76536686473017,4.96157056080646 4.39018064403225,5
 4,4.96157056080646 3.60981935596774,4.84775906502257 3.23463313526982,4.66293922460509
 2.8888595339608,4.41421356237309 2.58578643762691,4.1111404660392
 2.33706077539491,3.76536686473018 2.15224093497743,3.39018064403226 2.03842943919354,3
 2,2.60981935596774 2.03842943919354,2.23463313526982 2.15224093497743,1.8888595339608
 2.33706077539491,1.58578643762691 2.58578643762691,1.33706077539491
 2.8888595339608,1.15224093497743 3.23463313526982,1.03842943919354 3.60981935596774,1
 4))
```

다음 SQL은 원을 근사화하는 입력 점 형상의 버퍼를 반환합니다. 이 명령은 1/4 원당 세그먼트 수로 3을 지정하므로 함수에서는 기본값인 3개 세그먼트를 사용하여 1/4 원을 근사화합니다.

```
SELECT ST_AsEwkt(ST_Buffer(ST_GeomFromText('POINT(3 4)'), 2, 3));
```

```
               st_asewkt
POLYGON((1 4,1.26794919243112 5,2 5.73205080756888,3 6,4
 5.73205080756888,4.73205080756888 5,5 4,4.73205080756888 3,4 2.26794919243112,3 2,2
 2.26794919243112,1.26794919243112 3,1 4))
```

ST_Centroid

ST_Centroid는 다음과 같이 지오메트리의 중심을 나타내는 점을 반환합니다.

- POINT 지오메트리의 경우 좌표가 지오메트리의 점 좌표의 평균인 점을 반환합니다.
- LINESTRING 지오메트리의 경우 좌표가 지오메트리 세그먼트의 중간점의 가중 평균인 점을 반환합니다. 여기서 가중치는 지오메트리 세그먼트의 길이입니다.
- POLYGON 지오메트리의 경우 좌표가 영역 지오메트리의 삼각 분할 중심의 가중 평균인 점을 반환합니다. 여기서 가중치는 삼각 분할에서 삼각형의 면적입니다.
- 지오메트리 컬렉션의 경우 지오메트리 컬렉션에서 최대 토폴로지 차원의 지오메트리 중심의 가중 평균을 반환합니다.

구문

```
ST_Centroid(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

GEOMETRY

geom이 null이면 null이 반환됩니다.

geom이 비어 있으면 null이 반환됩니다.

예시

다음 SQL은 입력 라인스트링의 중심점을 반환합니다.

```
SELECT ST_AsEWKT(ST_Centroid(ST_GeomFromText('LINESTRING(110 40, 2 3, -10 80, -7 9, -22
 -33)', 4326)))
```

```
                      st_asewkt
-----------------------------------------------------
 SRID=4326;POINT(15.6965103455214 27.0206782881905)
```

ST_Collect

ST_Collect에는 두 가지 변형이 있습니다. 하나는 2개의 지오메트리를 허용하고 다른 하나는 집계 표현식을 허용합니다.

ST_Collect의 첫 번째 변형은 입력 지오메트리에서 지오메트리를 생성합니다. 입력 지오메트리의 순서는 유지됩니다. 이 변형은 다음과 같이 작동합니다.

- 두 입력 지오메트리가 모두 점이면 2개의 점이 있는 MULTIPOINT가 반환됩니다.
- 두 입력 지오메트리가 모두 라인스트링이면 2개의 라인스트링이 있는 MULTILINESTRING이 반환됩니다.
- 두 입력 지오메트리가 모두 다각형이면 2개의 다각형이 있는 MULTIPOLYGON이 반환됩니다.
- 그렇지 않으면 2개의 지오메트리가 있는 GEOMETRYCOLLECTION이 반환됩니다.

ST_Collect의 두 번째 변형은 지오메트리 열의 지오메트리에서 지오메트리를 생성합니다. 지오메트리의 결정된 반환 순서는 없습니다. 반환된 지오메트리의 순서를 지정하려면 WITHIN GROUP (ORDER BY ...) 절을 지정합니다. 이 변형은 다음과 같이 작동합니다.

- 입력 집계 표현식의 NULL이 아닌 모든 행이 점이면 집계 표현식의 모든 점을 포함하는 다중 점이 반환됩니다.
- 집계 표현식의 NULL이 아닌 모든 행이 라인스트링이면 집계 표현식의 모든 라인스트링을 포함하는 다중 라인스트링이 반환됩니다.
- 집계 표현식의 NULL이 아닌 모든 행이 다각형이면 집계 표현식의 모든 다각형을 포함하는 다중 다각형이 반환됩니다.
- 그렇지 않으면 집계 표현식의 모든 지오메트리를 포함하는 GEOMETRYCOLLECTION이 반환됩니다.

ST_Collect는 입력 지오메트리와 동일한 차원의 지오메트리를 반환합니다. 모든 입력 지오메트리의 차원이 같아야 합니다.

구문

```
ST_Collect(geom1, geom2)
```

```
ST_Collect(aggregate_expression)  [WITHIN GROUP (ORDER BY sort_expression1 [ASC | DESC]
 [, sort_expression2 [ASC | DESC] ...])]
```

인수

geom1

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

geom2

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

aggregate_expression

GEOMETRY 데이터 형식의 열 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

[WITHIN GROUP (ORDER BY sort_expression1 [ASC | DESC] [, sort_expression2 [ASC | DESC] ...])]

(옵션) 집계된 값의 정렬 순서를 지정하는 절입니다. ORDER BY 절에는 정렬 표현식 목록이 포함되어 있습니다. 정렬 표현식은 열 이름과 같은 쿼리 선택 목록의 유효한 정렬 표현식과 유사한 표현식입니다. 오름차순(ASC) 또는 내림차순(DESC)을 지정할 수 있습니다. 기본값은 ASC입니다.

반환 타입

하위 유형 MULTIPOINT, MULTILINESTRING, MULTIPOLYGON 또는 GEOMETRYCOLLECTION의 GEOMETRY입니다.

반환된 지오메트리의 SRID(공간 참조 시스템 식별자) 값은 입력 지오메트리의 SRID 값입니다.

geom1 또는 geom2가 null이면 null이 반환됩니다.

aggregate_expression의 모든 행이 null이면 null이 반환됩니다.

geom1이 null이면 geom2의 복사본이 반환됩니다. 마찬가지로 geom2가 null이면 geom1의 복사본이 반환됩니다.

geom1과 geom2의 SRID 값이 다른 경우 오류가 반환됩니다.

aggregate_expression에 있는 두 지오메트리의 SRID 값이 다르면 오류가 반환됩니다.

반환된 지오메트리가 최대 크기 GEOMETRY보다 크면 오류가 반환됩니다.

geom1과 geom2의 차원이 다르면 오류가 반환됩니다.

aggregate_expression에 있는 두 지오메트리의 차원이 다르면 오류가 반환됩니다.

예시

다음 SQL은 2개의 지오메트리를 포함하는 지오메트리 컬렉션을 반환합니다.

```
SELECT ST_AsText(ST_Collect(ST_GeomFromText('LINESTRING(0 0,1 1)'),
 ST_GeomFromText('POLYGON((10 10,20 10,10 20,10 10))')));
```

```
st_astext
-----------
 GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),POLYGON((10 10,20 10,10 20,10 10)))
```

다음 SQL은 테이블의 모든 지오메트리를 지오메트리 컬렉션으로 수집합니다.

```
WITH tbl(g) AS (SELECT ST_GeomFromText('POINT(1 2)', 4326) UNION ALL
SELECT ST_GeomFromText('LINESTRING(0 0,10 0)', 4326) UNION ALL
SELECT ST_GeomFromText('MULTIPOINT(13 4,8 5,4 4)', 4326) UNION ALL
SELECT NULL::geometry UNION ALL
SELECT ST_GeomFromText('POLYGON((0 0,10 0,0 10,0 0))', 4326))
SELECT ST_AsEWKT(ST_Collect(g)) FROM tbl;
```

```
st_astext
-----------
 SRID=4326;GEOMETRYCOLLECTION(POINT(1 2),LINESTRING(0 0,10 0),MULTIPOINT((13 4),(8 5),
(4 4)),POLYGON((0 0,10 0,0 10,0 0)))
```

다음 SQL은 ID 열별로 그룹화되고 이 ID로 정렬된 테이블의 모든 지오메트리를 수집합니다. 이 예에서 결과 지오메트리는 다음과 같이 ID별로 그룹화됩니다.

- id 1 – 다중 점의 점.
- id 2 – 다중 라인스트링의 라인스트링.
- id 3 – 지오메트리 컬렉션의 혼합 하위 유형.
- id 4 – 다중 다각형의 다각형.
- id 5 – null이고 결과는 null임.

```
WITH tbl(id, g) AS (SELECT 1, ST_GeomFromText('POINT(1 2)', 4326) UNION ALL
SELECT 1, ST_GeomFromText('POINT(4 5)', 4326) UNION ALL
```

```
SELECT 2, ST_GeomFromText('LINESTRING(0 0,10 0)', 4326) UNION ALL
SELECT 2, ST_GeomFromText('LINESTRING(10 0,20 -5)', 4326) UNION ALL
SELECT 3, ST_GeomFromText('MULTIPOINT(13 4,8 5,4 4)', 4326) UNION ALL
SELECT 3, ST_GeomFromText('MULTILINESTRING((-1 -1,-2 -2),(-3 -3,-5 -5))', 4326) UNION
 ALL
SELECT 4, ST_GeomFromText('POLYGON((0 0,10 0,0 10,0 0))', 4326) UNION ALL
SELECT 4, ST_GeomFromText('POLYGON((20 20,20 30,30 20,20 20))', 4326) UNION ALL
SELECT 1, NULL::geometry UNION ALL SELECT 2, NULL::geometry UNION ALL
SELECT 5, NULL::geometry UNION ALL SELECT 5, NULL::geometry)
SELECT id, ST_AsEWKT(ST_Collect(g)) FROM tbl GROUP BY id ORDER BY id;
```

```
 id |                                                st_asewkt

----
+-----------------------------------------------------------------------------------------------
  1 | SRID=4326;MULTIPOINT((1 2),(4 5))
  2 | SRID=4326;MULTILINESTRING((0 0,10 0),(10 0,20 -5))
  3 | SRID=4326;GEOMETRYCOLLECTION(MULTIPOINT((13 4),(8 5),(4 4)),MULTILINESTRING((-1
 -1,-2 -2),(-3 -3,-5 -5)))
  4 | SRID=4326;MULTIPOLYGON(((0 0,10 0,0 10,0 0)),((20 20,20 30,30 20,20 20)))
  5 |
```

다음 SQL은 테이블의 모든 지오메트리를 지오메트리 컬렉션에 수집합니다. 결과는 id의 내림차순으로 정렬된 다음 최소 및 최대 x 좌표에 따라 사전순으로 정렬됩니다.

```
WITH tbl(id, g) AS (
SELECT 1, ST_GeomFromText('POINT(4 5)', 4326) UNION ALL
SELECT 1, ST_GeomFromText('POINT(1 2)', 4326) UNION ALL
SELECT 2, ST_GeomFromText('LINESTRING(10 0,20 -5)', 4326) UNION ALL
SELECT 2, ST_GeomFromText('LINESTRING(0 0,10 0)', 4326) UNION ALL
SELECT 3, ST_GeomFromText('MULTIPOINT(13 4,8 5,4 4)', 4326) UNION ALL
SELECT 3, ST_GeomFromText('MULTILINESTRING((-1 -1,-2 -2),(-3 -3,-5 -5))', 4326) UNION
 ALL
SELECT 4, ST_GeomFromText('POLYGON((20 20,20 30,30 20,20 20))', 4326) UNION ALL
SELECT 4, ST_GeomFromText('POLYGON((0 0,10 0,0 10,0 0))', 4326) UNION ALL
SELECT 1, NULL::geometry UNION ALL SELECT 2, NULL::geometry UNION ALL
SELECT 5, NULL::geometry UNION ALL SELECT 5, NULL::geometry)
SELECT ST_AsEWKT(ST_Collect(g) WITHIN GROUP (ORDER BY id DESC, ST_XMin(g), ST_XMax(g)))
 FROM tbl;
```

```
                          st_asewkt

-------------------------------------------------------------------------------------------
 SRID=4326;GEOMETRYCOLLECTION(POLYGON((0 0,10 0,0 10,0 0)),POLYGON((20 20,20 30,30
 20,20 20)),MULTILINESTRING((-1 -1,-2 -2),(-3 -3,-5 -5)),MULTIPOINT((13 4),(8 5),(4
 4)),LINESTRING(0 0,10 0),LINESTRING(10 0,20 -5),POINT(1 2),POINT(4 5)
```

ST_Contains

ST_Contains는 첫 번째 입력 지오메트리의 2D 프로젝션에 두 번째 입력 지오메트리의 2D 프로젝션이 포함된 경우 true를 반환합니다. 지오메트리 B의 모든 점이 지오메트리 A의 점이고 그 내부에 비어 있지 않은 교차점이 있는 경우, A는 B를 포함합니다.

ST_Contains(A, B)는 ST_Within(B, A)와 동등합니다.

구문

```
ST_Contains(geom1, geom2)
```

인수

geom1

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

geom2

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 이 값을 geom1과 비교하여 해당 값이 geom1에 포함되어 있는지 판별합니다.

반환 타입

BOOLEAN

geom1 또는 geom2가 null이면 null이 반환됩니다.

geom1과 geom2의 SRID(공간 참조 시스템 식별자) 값이 같지 않으면 오류가 반환됩니다.

geom1 또는 geom2가 지오메트리 컬렉션인 경우 오류가 반환됩니다.

예시

다음 SQL은 첫 번째 다각형에 두 번째 다각형이 포함되어 있는지 확인합니다.

```
SELECT ST_Contains(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
```

```
st_contains
-----------
 false
```

ST_ContainsProperly

ST_ContainsProperly는 두 입력 지오메트리가 비어 있지 않고 두 번째 지오메트리의 2D 프로젝션의 모든 점이 첫 번째 지오메트리의 2D 프로젝션의 내부 점인 경우 true를 반환합니다.

구문

```
ST_ContainsProperly(geom1, geom2)
```

인수

geom1

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 GEOMETRYCOLLECTION일 수 없습니다.

geom2

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 GEOMETRYCOLLECTION일 수 없습니다. 이 값을 geom1과 비교하여 모든 점이 geom1의 내부 점인지 확인합니다.

반환 타입

BOOLEAN

geom1 또는 geom2가 null이면 null이 반환됩니다.

geom1과 geom2의 SRID(공간 참조 시스템 식별자) 값이 같지 않으면 오류가 반환됩니다.

geom1 또는 geom2가 지오메트리 컬렉션인 경우 오류가 반환됩니다.

예시

다음 SQL은 입력 라인스트링이 입력 다각형의 경계와 내부를 교차하는 ST_Contains 및 ST_ContainsProperly의 값을 반환합니다(외부는 아님). 다각형에 라인스트링이 포함되어 있지만 라인스트링이 제대로 포함되어 있지 않습니다.

```
WITH tmp(g1, g2)
AS (SELECT ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0))'),
 ST_GeomFromText('LINESTRING(5 5,10 5,10 6,5 5)')) SELECT ST_Contains(g1, g2),
 ST_ContainsProperly(g1, g2)
FROM tmp;
```

```
 st_contains | st_containsproperly
-------------+---------------------
 t           | f
```

ST_ConvexHull

ST_ConvexHull은 입력 지오메트리에 포함된 비어 있지 않은 점의 볼록 껍질을 나타내는 지오메트리를 반환합니다.

빈 입력의 경우 결과 지오메트리는 입력 지오메트리와 동일합니다. 비어 있지 않은 모든 입력에 대해 이 함수는 입력 지오메트리의 2D 프로젝션에서 작동합니다. 그러나 출력 지오메트리의 차원은 입력 지오메트리의 차원에 따라 달라집니다. 보다 구체적으로 입력 지오메트리가 비어 있지 않은 3DM 또는 3D 지오메트리인 경우 m 좌표가 삭제됩니다. 즉, 반환된 지오메트리의 차원은 각각 2D 또는 3DZ입니다. 입력이 비어 있지 않은 2D 또는 3DZ 지오메트리인 경우 결과 지오메트리는 동일한 차원을 갖습니다.

구문

```
ST_ConvexHull(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

GEOMETRY

반환된 지오메트리의 공간 참조 시스템 식별자(SRID) 값은 입력 지오메트리의 SRID 값입니다.

geom이 null이면 null이 반환됩니다.

값은 다음과 같이 반환됩니다.

볼록 껍질의 점 수	지오메트리 하위 유형
0	geom의 복사본이 반환됩니다.
1	POINT 하위 유형이 반환됩니다.
2	LINESTRING 하위 유형이 반환됩니다. 반환된 라인스트링의 두 점은 사전순으로 정렬됩니다.
3 이상	내부 링이 없는 POLYGON 하위 유형이 반환됩니다. 다각형은 시계 방향이고 외부 링의 첫 번째 점은 사전순으로 링의 가장 작은 점입니다.

예시

다음 SQL은 라인스트링의 EWKT(Extended Well-Known Text) 표현을 반환합니다. 이 경우 반환된 블록 껍질이 다각형입니다.

```
SELECT ST_AsEWKT(ST_ConvexHull(ST_GeomFromText('LINESTRING(0 0,1 0,0 1,1 1,0.5 0.5)')))
 as output;
```

```
output
-------------
POLYGON((0 0,0 1,1 1,1 0,0 0))
```

다음 SQL은 라인스트링의 EWKT 표현을 반환합니다. 이 경우 반환된 블록 껍질이 라인스트링입니다.

```
SELECT ST_AsEWKT(ST_ConvexHull(ST_GeomFromText('LINESTRING(0 0,1 1,0.2 0.2,0.6 0.6,0.5
 0.5)'))) as output;
```

```
output
-------------
LINESTRING(0 0,1 1)
```

다음 SQL은 다중 점의 EWKT 표현을 반환합니다. 이 경우 반환된 블록 껍질이 점입니다.

```
SELECT ST_AsEWKT(ST_ConvexHull(ST_GeomFromText('MULTIPOINT(0 0,0 0,0 0)'))) as output;
```

```
output
-------------
POINT(0 0)
```

ST_CoveredBy

ST_CoveredBy는 첫 번째 입력 지오메트리의 2D 프로젝션이 두 번째 입력 지오메트리의 2D 프로젝션으로 덮인 경우 true를 반환합니다. 지오메트리 A의 모든 점이 지오메트리 B의 점이고 둘 다 비어 있지 않은 경우 A는 B에 의해 덮여 있습니다.

ST_CoveredBy(A, B)는 ST_Covers(B, A)와 동등합니다.

구문

```
ST_CoveredBy(geom1, geom2)
```

인수

geom1

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 이 값을 geom2와 비교하여 geom2에 의해 덮여 있는지 판별합니다.

geom2

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

BOOLEAN

geom1 또는 geom2가 null이면 null이 반환됩니다.

geom1과 geom2의 SRID(공간 참조 시스템 식별자) 값이 같지 않으면 오류가 반환됩니다.

geom1 또는 geom2가 지오메트리 컬렉션인 경우 오류가 반환됩니다.

예시

다음 SQL은 첫 번째 다각형이 두 번째 다각형에 의해 덮여 있는지 확인합니다.

```
SELECT ST_CoveredBy(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
```

```
st_coveredby
-----------
 true
```

ST_Covers

ST_Covers는 첫 번째 입력 지오메트리의 2D 프로젝션이 두 번째 입력 지오메트리의 2D 프로젝션을 덮고 있는 경우 true를 반환합니다. 지오메트리 B의 모든 점이 지오메트리 A의 점이고 둘 다 비어 있지 않은 경우 A가 B를 덮고 있습니다.

ST_Covers(A, B)는 ST_CoveredBy(B, A)와 동등합니다.

구문

```
ST_Covers(geom1, geom2)
```

인수

geom1

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

geom2

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 이 값을 geom1과 비교하여 geom1을 덮고 있는지 판별합니다.

반환 타입

BOOLEAN

geom1 또는 geom2가 null이면 null이 반환됩니다.

geom1과 geom2의 SRID(공간 참조 시스템 식별자) 값이 같지 않으면 오류가 반환됩니다.

geom1 또는 geom2가 지오메트리 컬렉션인 경우 오류가 반환됩니다.

예시

다음 SQL은 첫 번째 다각형이 두 번째 다각형을 덮고 있는지 확인합니다.

```
SELECT ST_Covers(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
```

```
st_covers
-----------
 false
```

ST_Crosses

ST_Crosses는 두 입력 지오메트리의 2D 프로젝션이 서로 교차하면 true를 반환합니다.

구문

```
ST_Crosses(geom1, geom2)
```

인수

geom1

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

geom2

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

BOOLEAN

geom1 또는 geom2가 null이면 오류가 반환됩니다.

geom1 또는 geom2가 지오메트리 컬렉션인 경우 오류가 반환됩니다.

geom1과 geom2의 SRID(공간 참조 시스템 식별자) 값이 같지 않으면 오류가 반환됩니다.

예시

다음 SQL은 첫 번째 다각형이 두 번째 다중 점과 교차하는지 확인합니다. 이 예에서 다중 점은 다각형의 내부와 외부를 모두 가로지르므로 ST_Crosses가 true를 반환합니다.

```
SELECT ST_Crosses (ST_GeomFromText('polygon((0 0,10 0,10 10,0 10,0 0))'),
 ST_GeomFromText('multipoint(5 5,0 0,-1 -1)'));
```

```
st_crosses
-------------
 true
```

다음 SQL은 첫 번째 다각형이 두 번째 다중 점과 교차하는지 확인합니다. 이 예에서 다중 점은 다각형의 외부를 가로지르므로 ST_Crosses가 false를 반환합니다.

```
SELECT ST_Crosses (ST_GeomFromText('polygon((0 0,10 0,10 10,0 10,0 0))'),
 ST_GeomFromText('multipoint(0 0,-1 -1)'));
```

```
st_crosses
-------------
 false
```

ST_Dimension

ST_Dimension은 입력 지오메트리의 고유 차원을 반환합니다. 고유 차원은 지오메트리에 정의된 하위 유형의 차원 값입니다.

3DM, 3DZ 및 4D 지오메트리 입력의 경우 ST_Dimension은 2D 지오메트리 입력과 동일한 결과를 반환합니다.

구문

```
ST_Dimension(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

geom의 고유한 차원을 나타내는 INTEGER입니다.

geom이 null이면 null이 반환됩니다.

값은 다음과 같이 반환됩니다.

반환 값	지오메트리 하위 유형
0	geom이 POINT 또는 MULTIPOINT 하위 유형인 경우 반환됩니다.
1	geom이 LINESTRING 또는 MULTILINESTRING 하위 유형인 경우 반환됩니다.
2	geom이 POLYGON 또는 MULTIPOLYGON 하위 유형인 경우 반환됩니다.
0	geom이 빈 GEOMETRYCOLLECTION 하위 유형인 경우 반환됩니다.
컬렉션 구성 요소의 가장 큰 차원	geom이 GEOMETRYCOLLECTION 하위 유형인 경우 반환됩니다.

예시

다음 SQL은 4개의 점 LINESTRING의 WKT(Well-Known Text) 표현을 GEOMETRY 객체로 변환하고 라인스트링의 차원을 반환합니다.

```
SELECT ST_Dimension(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27
 29.31,77.29 29.07)'));
```

```
st_dimension
-------------
1
```

ST_Disjoint

ST_Disjoint는 두 입력 지오메트리의 2D 프로젝션에 공통되는 점이 없는 경우 true를 반환합니다.

구문

```
ST_Disjoint(geom1, geom2)
```

인수

geom1

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

geom2

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

BOOLEAN

geom1 또는 geom2가 null이면 null이 반환됩니다.

geom1과 geom2의 SRID(공간 참조 시스템 식별자) 값이 같지 않으면 오류가 반환됩니다.

geom1 또는 geom2가 지오메트리 컬렉션인 경우 오류가 반환됩니다.

예시

다음 SQL은 첫 번째 다각형이 두 번째 다각형과 분리되어 있는지 확인합니다.

```
SELECT ST_Disjoint(ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(2 2,2 5,5 5,5
 2,2 2))'), ST_Point(4, 4));
```

```
st_disjoint
-----------
 true
```

ST_Distance

입력 지오메트리의 경우 ST_Distance는 두 입력 지오메트리 값의 2D 프로젝션 간 최소 유클리드 거리를 반환합니다.

3DM, 3DZ, 4D 기하학의 경우 ST_Distance는 두 입력 지오메트리 값의 2D 프로젝션 간 유클리드 거리를 반환합니다.

입력 지오그래피의 경우 ST_Distance는 두 2D 점의 측지 거리를 반환합니다. 거리 단위는 미터입니다. 점과 빈 점이 아닌 지오그래피의 경우 오류가 반환됩니다.

구문

```
ST_Distance(geo1, geo2)
```

인수

geo1

GEOMETRY 또는 GEOGRAPHY 데이터 유형의 값이나 GEOMETRY 또는 GEOGRAPHY 유형으로 계산되는 표현식입니다. geo1의 데이터 유형은 geo2와 동일해야 합니다.

geo2

GEOMETRY 또는 GEOGRAPHY 데이터 유형의 값이나 GEOMETRY 또는 GEOGRAPHY 유형으로 계산되는 표현식입니다. geo2의 데이터 유형은 geo1과 동일해야 합니다.

반환 타입

입력 지오메트리 또는 지오그래피와 동일한 단위의 DOUBLE PRECISION입니다.

geo1 또는 geo2가 null이거나 비어 있으면 null이 반환됩니다.

geo1과 geo2의 SRID(공간 참조 시스템 식별자) 값이 같지 않으면 오류가 반환됩니다.

geo1 또는 geo2가 지오메트리 컬렉션인 경우 오류가 반환됩니다.

예시

다음 SQL은 두 다각형 사이의 거리를 반환합니다.

```
SELECT ST_Distance(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 -3,-2 -1,0 -3,-1 -3))'));
```

```
  st_distance
-----------
1.4142135623731
```

다음 SQL은 GEOGRAPHY 데이터 유형을 사용하여 베를린의 브란덴부르크 문과 독일 의회 건물 사이의 거리(미터)를 반환합니다.

```
SELECT ST_Distance(ST_GeogFromText('POINT(13.37761826722198 52.516411678282445)'),
 ST_GeogFromText('POINT(13.377950831464005 52.51705102546893)'));
```

```
   st_distance
------------------
 74.64129172609631
```

ST_DistanceSphere

ST_DistanceSphere는 구에 있는 두 점 지오메트리 사이의 거리를 반환합니다.

구문

```
ST_DistanceSphere(geom1, geom2)
```

```
ST_DistanceSphere(geom1, geom2, radius)
```

인수

geom1

구에 놓인 GEOMETRY 데이터 형식의 점 값(도)입니다. 점의 첫 번째 좌표는 경도 값입니다. 점의 두 번째 좌표는 위도 값입니다. 3DZ, 3DM 또는 4D 지오메트리의 경우 처음 두 좌표만 사용됩니다.

geom2

구에 놓인 GEOMETRY 데이터 형식의 점 값(도)입니다. 점의 첫 번째 좌표는 경도 값입니다. 점의 두 번째 좌표는 위도 값입니다. 3DZ, 3DM 또는 4D 지오메트리의 경우 처음 두 좌표만 사용됩니다.

radius

데이터 형식이 DOUBLE PRECISION인 구의 반경입니다. radius가 제공되지 않으면 구의 기본값은 Earth로 설정되며 타원체의 WGS(World Geodetic System) 84 표현에서 반경이 계산됩니다.

반환 타입

반경과 같은 단위의 DOUBLE PRECISION입니다. 반경이 제공되지 않은 경우 거리는 미터 단위입니다.

geom1 또는 geom2가 null이거나 비어 있으면 null이 반환됩니다.

radius가 제공되지 않으면 결과는 지표면을 따라 측정된 미터 단위가 반환됩니다.

radius가 음수이면 오류가 반환됩니다.

geom1과 geom2의 SRID(공간 참조 시스템 식별자) 값이 같지 않으면 오류가 반환됩니다.

geom1 또는 geom2가 점이 아니면 오류가 반환됩니다.

예시

다음 예시 SQL은 지구상의 두 지점 사이의 거리를 킬로미터 단위로 계산합니다.

```
SELECT ROUND(ST_DistanceSphere(ST_Point(-122, 47), ST_Point(-122.1, 47.1))/ 1000, 0);
```

```
  round
-----------
 13
```

다음 예제 SQL은 독일의 세 공항 위치 Berlin Tegel(TXL), Munich International(MUC) 및 Frankfurt International(FRA) 사이의 거리를 킬로미터 단위로 계산합니다.

```
WITH airports_raw(code,lon,lat) AS (
(SELECT 'MUC', 11.786111, 48.353889) UNION
(SELECT 'FRA', 8.570556, 50.033333) UNION
(SELECT 'TXL', 13.287778, 52.559722)),
airports1(code,location) AS (SELECT code, ST_Point(lon, lat) FROM airports_raw),
airports2(code,location) AS (SELECT * from airports1)
SELECT (airports1.code || ' <-> ' || airports2.code) AS airports,
round(ST_DistanceSphere(airports1.location, airports2.location) / 1000, 0) AS
 distance_in_km
FROM airports1, airports2 WHERE airports1.code < airports2.code ORDER BY 1;
```

```
  airports   | distance_in_km
-------------+----------------
 FRA <-> MUC |            299
```

```
 FRA <-> TXL |            432
 MUC <-> TXL |            480
```

ST_DWithin

ST_DWithin은 두 입력 지오메트리 값의 2D 프로젝션 간 유클리드 거리가 임계값보다 크지 않은 경우 true를 반환합니다.

구문

```
ST_DWithin(geom1, geom2, threshold)
```

인수

geom1

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

geom2

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

threshold

DOUBLE PRECISION 데이터 형식의 값입니다. 이 값은 입력 인수 단위입니다.

반환 타입

BOOLEAN

geom1 또는 geom2가 null이면 null이 반환됩니다.

threshold가 음수이면 오류가 반환됩니다.

geom1과 geom2의 SRID(공간 참조 시스템 식별자) 값이 같지 않으면 오류가 반환됩니다.

geom1 또는 geom2가 지오메트리 컬렉션인 경우 오류가 반환됩니다.

예시

다음 SQL은 두 다각형 사이의 거리가 5단위 이내인지 확인합니다.

```
SELECT ST_DWithin(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'),5);
```

```
st_dwithin
-----------
 true
```

ST_EndPoint

ST_EndPoint는 입력 라인스트링의 마지막 점을 반환합니다. 결과의 SRID(공간 참조 시스템 식별자) 값은 입력 지오메트리의 값과 동일합니다. 반환된 지오메트리의 차원은 입력 지오메트리의 차원과 같습니다.

구문

```
ST_EndPoint(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 LINESTRING이어야 합니다.

반환 타입

GEOMETRY

geom이 null이면 null이 반환됩니다.

geom이 비어 있으면 null이 반환됩니다.

geom이 LINESTRING이 아니면 null이 반환됩니다.

예시

다음 SQL은 GEOMETRY 객체에 대한 4개의 점 LINESTRING의 EWKT(Extended Well-Known Text) 표현을 반환하고 라인스트링의 종료 지점을 반환합니다.

```
SELECT ST_AsEWKT(ST_EndPoint(ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0
 5)',4326)));
```

```
st_asewkt
-------------
 SRID=4326;POINT(0 5)
```

ST_Envelope

ST_Envelope은 다음과 같이 입력 지오메트리의 최소 경계 상자를 반환합니다.

- 입력 지오메트리가 비어 있으면 반환된 지오메트리는 입력 지오메트리의 복사본입니다.
- 입력 지오메트리의 최소 경계 상자가 점으로 변형되는 경우 반환된 지오메트리는 점입니다.
- 입력 지오메트리의 최소 경계 상자가 1차원이면 두 점 라인스트링이 반환됩니다.
- 이전 항목 중 어느 것도 true가 아니면 이 함수는 정점이 최소 경계 상자의 모서리인 시계 방향 다각형을 반환합니다.

반환된 지오메트리의 SRID(공간 참조 시스템 식별자)는 입력 지오메트리의 값과 동일합니다.

비어 있지 않은 모든 입력에 대해 이 함수는 입력 지오메트리의 2D 프로젝션에서 작동합니다.

구문

```
ST_Envelope(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

GEOMETRY

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 4개의 점 LINESTRING의 WKT(Well-Known Text) 표현을 GEOMETRY 객체로 변환하고 모서리가 최소 경계 상자인 정점을 가진 다각형을 반환합니다.

```
SELECT ST_AsText(ST_Envelope(ST_GeomFromText('GEOMETRYCOLLECTION(POLYGON((0 0,10 0,0
 10,0 0)),LINESTRING(20 10,20 0,10 0))')));
```

```
    st_astext
-----------------------------------
  POLYGON((0 0,0 10,20 10,20 0,0 0))
```

ST_Equals

ST_Equals는 입력 지오메트리의 2D 프로젝션이 기하학적으로 동일한 경우 true를 반환합니다. 지오메트리는 점 세트가 동일하고 그 내부에 비어 있지 않은 교차점이 있는 경우 기하학적으로 동일한 것으로 간주됩니다.

구문

```
ST_Equals(geom1, geom2)
```

인수

geom1

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

geom2

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 이 값을 geom1과 비교하여 geom1과 같은지 판별합니다.

반환 타입

BOOLEAN

geom1 또는 geom2가 null이면 오류가 반환됩니다.

geom1과 geom2의 SRID(공간 참조 시스템 식별자) 값이 같지 않으면 오류가 반환됩니다.

geom1 또는 geom2가 지오메트리 컬렉션인 경우 오류가 반환됩니다.

예시

다음 SQL은 두 다각형이 기하학적으로 같은지 확인합니다.

```
SELECT ST_Equals(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
```

```
st_equals
-----------
 false
```

다음 SQL은 두 라인스트링이 기하학적으로 같은지 확인합니다.

```
SELECT ST_Equals(ST_GeomFromText('LINESTRING(1 0,10 0)'), ST_GeomFromText('LINESTRING(1
 0,5 0,10 0)'));
```

```
st_equals
-----------
 true
```

ST_ExteriorRing

ST_ExteriorRing은 입력 다각형의 외부 링을 나타내는 닫힌 라인스트링을 반환합니다. 반환된 지오메트리의 차원은 입력 지오메트리의 차원과 같습니다.

구문

```
ST_ExteriorRing(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

LINESTRING 하위 유형의 GEOMETRY입니다.

반환된 지오메트리의 공간 참조 시스템 식별자(SRID) 값은 입력 지오메트리의 SRID 값입니다.

geom이 null이면 null이 반환됩니다.

geom이 다각형이 아니면 null이 반환됩니다.

geom이 비어 있으면 빈 다각형이 반환됩니다.

예시

다음 SQL은 다각형의 외부 링을 닫힌 라인스트링으로 반환합니다.

```
SELECT ST_AsEWKT(ST_ExteriorRing(ST_GeomFromText('POLYGON((7 9,8 7,11 6,15 8,16 6,17
 7,17 10,18 12,17 14,15 15,11 15,10 13,9 12,7 9),(9 9,10 10,11 11,11 10,10 8,9 9),(12
 14,15 14,13 11,12 14))')));
```

```
st_asewkt
-----------
 LINESTRING(7 9,8 7,11 6,15 8,16 6,17 7,17 10,18 12,17 14,15 15,11 15,10 13,9 12,7 9)
```

ST_Force2D

ST_Force2D는 입력 지오메트리의 2D 지오메트리를 반환합니다. 2D 지오메트리의 경우 입력 복사본이 반환됩니다. 3DZ, 3DM 및 4D 지오메트리의 경우 ST_Force2D는 지오메트리를 xy-데카르트 평면에 표시합니다. 입력 지오메트리의 빈 점은 출력 지오메트리의 빈 점으로 남아 있습니다.

구문

```
ST_Force2D(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

GEOMETRY.

반환된 지오메트리의 공간 참조 시스템 식별자(SRID) 값은 입력 지오메트리의 SRID 값입니다.

geom이 null이면 null이 반환됩니다.

geom이 비어 있으면 빈 지오메트리가 반환됩니다.

예시

다음 SQL은 3DZ 지오메트리에서 2D 지오메트리를 반환합니다.

```
SELECT ST_AsEWKT(ST_Force2D(ST_GeomFromText('MULTIPOINT Z(0 1 2, EMPTY, 2 3 4, 5 6
 7)')));
```

```
st_asewkt
-----------
  MULTIPOINT((0 1),EMPTY,(2 3),(5 6))
```

ST_Force3D

ST_Force3D는 ST_Force3DZ의 별칭입니다. 자세한 내용은 ST_Force3DZ 단원을 참조하십시오.

ST_Force3DM

ST_Force3DM은 입력 지오메트리의 3DM 지오메트리를 반환합니다. 2D 지오메트리의 경우 출력 지오메트리의 비어 있지 않은 점의 m 좌표는 모두 0으로 설정됩니다. 3DM 지오메트리의 경우 입력 지오메트리 복사본이 반환됩니다. 3DZ 지오메트리의 경우 지오메트리는 xy-데카르트 평면에 나타나고 출력 지오메트리에서 비어 있지 않은 점의 m 좌표는 모두 0으로 설정됩니다. 4D 지오메트리의 경우 지오메트리가 xym-데카르트 공간에 나타납니다. 입력 지오메트리의 빈 점은 출력 지오메트리의 빈 점으로 남아 있습니다.

구문

```
ST_Force3DM(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

GEOMETRY.

반환된 지오메트리의 공간 참조 시스템 식별자(SRID) 값은 입력 지오메트리의 SRID 값입니다.

geom이 null이면 null이 반환됩니다.

geom이 비어 있으면 빈 지오메트리가 반환됩니다.

예시

다음 SQL은 3DZ 지오메트리에서 3DM 지오메트리를 반환합니다.

```
SELECT ST_AsEWKT(ST_Force3DM(ST_GeomFromText('MULTIPOINT Z(0 1 2, EMPTY, 2 3 4, 5 6
 7)')));
```

```
st_asewkt
-----------
  MULTIPOINT M ((0 1 0),EMPTY,(2 3 0),(5 6 0))
```

ST_Force3DZ

ST_Force3DZ는 입력 지오메트리에서 3DZ 지오메트리를 반환합니다. 2D 지오메트리의 경우 출력 지오메트리의 비어 있지 않은 점의 z 좌표는 모두 0으로 설정됩니다. 3DM 지오메트리의 경우 지오메트리는 xy-데카르트 평면에 나타나고 출력 지오메트리에서 비어 있지 않은 점의 z 좌표는 모두 0으로 설정됩니다. 3DZ 지오메트리의 경우 입력 지오메트리 복사본이 반환됩니다. 4D 지오메트리의 경우 지오메트리가 xyz-데카르트 공간에 나타납니다. 입력 지오메트리의 빈 점은 출력 지오메트리의 빈 점으로 남아 있습니다.

구문

```
ST_Force3DZ(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

GEOMETRY.

반환된 지오메트리의 공간 참조 시스템 식별자(SRID) 값은 입력 지오메트리의 SRID 값입니다.

geom이 null이면 null이 반환됩니다.

geom이 비어 있으면 빈 지오메트리가 반환됩니다.

예시

다음 SQL은 3DM 지오메트리에서 3DZ 지오메트리를 반환합니다.

```
SELECT ST_AsEWKT(ST_Force3DZ(ST_GeomFromText('MULTIPOINT M(0 1 2, EMPTY, 2 3 4, 5 6
 7)')));
```

```
st_asewkt
-----------
  MULTIPOINT Z ((0 1 0),EMPTY,(2 3 0),(5 6 0))
```

ST_Force4D

ST_Force4D는 입력 지오메트리의 4D 지오메트리를 반환합니다. 2D 지오메트리의 경우 출력 지오메트리의 비어 있지 않은 점의 z 및 m 좌표는 모두 0으로 설정됩니다. 3DM 지오메트리의 경우 출력 지오메트리의 비어 있지 않은 점의 z 좌표는 모두 0으로 설정됩니다. 3DZ 지오메트리의 경우 출력 지오메트리의 비어 있지 않은 점의 m 좌표는 모두 0으로 설정됩니다. 4D 지오메트리의 경우 입력 지오메트리 복사본이 반환됩니다. 입력 지오메트리의 빈 점은 출력 지오메트리의 빈 점으로 남아 있습니다.

구문

```
ST_Force4D(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

GEOMETRY.

반환된 지오메트리의 공간 참조 시스템 식별자(SRID) 값은 입력 지오메트리의 SRID 값입니다.

geom이 null이면 null이 반환됩니다.

geom이 비어 있으면 빈 지오메트리가 반환됩니다.

예시

다음 SQL은 3DM 지오메트리에서 4D 지오메트리를 반환합니다.

```
SELECT ST_AsEWKT(ST_Force4D(ST_GeomFromText('MULTIPOINT M(0 1 2, EMPTY, 2 3 4, 5 6
 7)')));
```

```
st_asewkt
-----------
  MULTIPOINT ZM ((0 1 0 2),EMPTY,(2 3 0 4),(5 6 0 7))
```

ST_GeoHash

ST_GeoHash는 지정된 정밀도로 입력 점의 geohash 표현을 반환합니다. 기본 정밀도 값은 20입니다. Geohash의 정의에 대한 자세한 내용은 Wikipedia의 Geohash를 참조하세요.

구문

```
ST_GeoHash(geom)
```

```
ST_GeoHash(geom, precision)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

precision

INTEGER 데이터 형식의 값입니다. 기본값은 20입니다.

반환 타입

GEOMETRY

이 함수는 입력 점의 geohash 표현을 반환합니다.

입력 점이 비어 있으면 null을 반환합니다.

입력 형상이 점이 아니면 오류를 반환합니다.

예시

다음 SQL은 입력 점의 geohash 표현을 반환합니다.

```
SELECT ST_GeoHash(ST_GeomFromText('POINT(45 -45)'), 25) AS geohash;
```

```
          geohash
---------------------------
 m0000000000000000000gzz
```

입력 점이 비어 있으므로 다음 SQL은 null을 반환합니다.

```
SELECT ST_GeoHash(ST_GeomFromText('POINT EMPTY'), 10) IS NULL AS result;
```

```
 result
---------
 true
```

ST_GeogFromText

ST_GeogFromText는 입력 지오그래피의 WKT(Well-Known Text) 또는 EWKT(Extended Well-Known Text) 표현으로 지오그래피 객체를 구성합니다.

구문

```
ST_GeogFromText(wkt_string)
```

인수

wkt_string

지오그래피의 WKT 또는 EWKT 표현인 VARCHAR 데이터 유형의 값입니다.

반환 타입

GEOGRAPHY

SRID 값이 입력에 제공된 값으로 설정된 경우. SRID가 제공되지 않으면 4326으로 설정됩니다.

wkt_string이 null이면 null이 반환됩니다.

wkt_string이 유효하지 않으면 오류가 반환됩니다.

예시

다음 SQL은 SRID 값을 사용하여 지오그래피 객체로 다각형을 구성합니다.

```
SELECT ST_AsEWKT(ST_GeogFromText('SRID=4324;POLYGON((0 0,0 1,1 1,10 10,1 0,0 0))'));
```

```
  st_asewkt
-----------------------------------------------
 SRID=4324;POLYGON((0 0,0 1,1 1,10 10,1 0,0 0))
```

다음 SQL은 지오그래피 객체로 다각형을 구성합니다. SRID 값이 4326으로 설정됩니다.

```
SELECT ST_AsEWKT(ST_GeogFromText('POLYGON((0 0,0 1,1 1,10 10,1 0,0 0))'));
```

```
 st_asewkt
-----------------------------------------------
 SRID=4326;POLYGON((0 0,0 1,1 1,10 10,1 0,0 0))
```

ST_GeogFromWKB

ST_GeogFromWKB는 입력 지오그래피의 16진수 WKB(Well-Known Binary) 표현으로 지오그래피 객체를 구성합니다.

구문

```
ST_GeogFromWKB(wkb_string)
```

인수

wkb_string

지오그래피의 16진수 WKB 표현인 VARCHAR 데이터 유형의 값입니다.

반환 타입

GEOGRAPHY

SRID 값이 제공되면 제공된 값으로 설정됩니다. SRID가 제공되지 않으면 4326으로 설정됩니다.

wkb_string이 null이면 null이 반환됩니다.

wkb_string이 유효하지 않으면 오류가 반환됩니다.

예시

다음 SQL은 16진수 WKB 값으로 지오그래피를 구성합니다.

```
SELECT
 ST_AsEWKT(ST_GeogFromWKB('0103000000010000000500000000000000000000000000000000000000000000000000F03F0000000000F03F000000000000F03F0000000000F03F0000000000000000000000000000000000000000'));
```

```
 st_asewkt
-------------------------------------------
 SRID=4326;POLYGON((0 0,0 1,1 1,1 0,0 0))
```

ST_GeometryN

ST_GeometryN은 다음과 같이 입력 지오메트리의 입력 인덱스가 가리키는 지오메트리를 반환합니다.

- 입력이 점, 라인스트링 또는 다각형인 경우 인덱스가 1과 같으면 지오메트리가 그대로 반환되고 인덱스가 1이 아니면 null이 반환됩니다.
- 입력이 다중 점, 다중 라인스트링, 다중 다각형 또는 지오메트리 컬렉션이면 입력 인덱스가 가리키는 대로 점, 라인스트링, 다각형 또는 지오메트리 컬렉션이 반환됩니다.

인덱스는 1부터 시작합니다. 결과의 SRID(공간 참조 시스템 식별자)는 입력 지오메트리의 값과 동일합니다. 반환된 지오메트리의 차원은 입력 지오메트리의 차원과 같습니다.

구문

```
ST_GeometryN(geom, index)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

인덱스를 구축하고 배포할 것입니다

1부터 시작하는 인덱스의 위치를 나타내는 INTEGER 데이터 형식의 값입니다.

반환 타입

GEOMETRY

geom 또는 index가 null이면 null이 반환됩니다.

index가 범위를 벗어나면 오류가 반환됩니다.

예시

다음 SQL은 지오메트리 컬렉션의 지오메트리를 반환합니다.

```
WITH tmp1(idx) AS (SELECT 1 UNION SELECT 2),
tmp2(g) AS (SELECT ST_GeomFromText('GEOMETRYCOLLECTION(POLYGON((0 0,10 0,0 10,0
 0)),LINESTRING(20 10,20 0,10 0))'))
SELECT idx, ST_AsEWKT(ST_GeometryN(g, idx)) FROM tmp1, tmp2 ORDER BY idx;
```

```
 idx |          st_asewkt
-----+-----------------------------
   1 | POLYGON((0 0,10 0,0 10,0 0))
   2 | LINESTRING(20 10,20 0,10 0)
```

ST_GeometryType

ST_GeometryType은 입력 지오메트리의 하위 유형을 문자열로 반환합니다.

3DM, 3DZ 및 4D 지오메트리 입력의 경우 ST_GeometryType은 2D 지오메트리 입력과 동일한 결과를 반환합니다.

구문

```
ST_GeometryType(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

geom의 하위 유형을 나타내는 VARCHAR입니다.

geom이 null이면 null이 반환됩니다.

값은 다음과 같이 반환됩니다.

반환된 문자열 값	지오메트리 하위 유형
ST_Point	geom이 POINT 하위 유형인 경우 반환됩니다.
ST_LineString	geom이 LINESTRING 하위 유형인 경우 반환됩니다.
ST_Polygon	geom이 POLYGON 하위 유형인 경우 반환됩니다.
ST_MultiPoint	geom이 MULTIPOINT 하위 유형인 경우 반환됩니다.
ST_MultiLineString	geom이 MULTILINESTRING 하위 유형인 경우 반환됩니다.
ST_MultiPolygon	geom이 MULTIPOLYGON 하위 유형인 경우 반환됩니다.

반환된 문자열 값	지오메트리 하위 유형
ST_GeometryCollection	geom이 GEOMETRYCOLLECTION 하위 유형인 경우 반환됩니다.

예시

다음 SQL은 입력 라인스트링 지오메트리의 하위 유형을 반환합니다.

```
SELECT ST_GeometryType(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27
 29.31,77.29 29.07)'));
```

```
st_geometrytype
-------------
 ST_LineString
```

ST_GeomFromEWKB

ST_GeomFromEWKB는 입력 지오메트리의 EWKB(Extended Well-Known Binary) 표현으로부터 지오메트리 객체를 구성합니다.

ST_GeomFromEWKB는 WKB 및 EWKB 16진수 형식으로 작성된 3DZ, 3DM 및 4D 지오메트리를허용합니다.

구문

```
ST_GeomFromEWKB(ewkb_string)
```

인수

ewkb_string

지오메트리의 16진수 EWKB 표현인 VARCHAR 데이터 형식의 값입니다.

반환 타입

GEOMETRY

ewkb_string이 null이면 null이 반환됩니다.

ewkb_string이 유효하지 않으면 오류가 반환됩니다.

예시

다음 SQL은 EWKB 값에서 다각형을 구성하고 다각형의 EWKT 표현을 반환합니다.

```
SELECT
 ST_AsEWKT(ST_GeomFromEWKB('0103000020E6100000010000000500000000000000000000000000000000000000000
```

```
 st_asewkt
--------------------------------
 SRID=4326;POLYGON((0 0,0 1,1 1,1 0,0 0))
```

ST_GeomFromEWKT

ST_GeomFromEWKT는 입력 지오메트리의 EWKT(Extended Well-Known Text) 표현으로부터 지오메트리 객체를 구성합니다.

ST_GeomFromEWKT는 3DZ, 3DM 및 4D를 허용합니다. 여기서 지오메트리 유형에는 각각 Z, M 또는 ZM 접두사가 붙습니다.

구문

```
ST_GeomFromEWKT(ewkt_string)
```

인수

ewkt_string

데이터 형식 VARCHAR의 값 또는 지오메트리의 EWKT 표현인 VARCHAR 형식으로 평가되는 표현식입니다.

WKT 키워드 EMPTY를 사용하여 빈 점, 빈 점이 있는 다중 점 또는 빈 점이 있는 지오메트리 컬렉션을 지정할 수 있습니다. 다음 예에서는 빈 점을 생성합니다.

```
ST_GeomFromEWKT('SRID=4326;POINT EMPTY');
```

반환 타입

GEOMETRY

ewkt_string이 null이면 null이 반환됩니다.

ewkt_string이 유효하지 않으면 오류가 반환됩니다.

예시

다음 SQL은 EWKT 값에서 다중 라인스트링을 구성하고 지오메트리를 반환합니다. 지오메트리의 ST_AsEWKT 결과도 반환합니다.

```
SELECT ST_GeomFromEWKT('SRID=4326;MULTILINESTRING((1 0,1 0),(2 0,3 0),(4 0,5 0,6 0))')
 as geom, ST_AsEWKT(geom);
```

```
                                                    geom

                                          |                          st_asewkt

-----------------------------------------------------------------------------------------------------
+----------------------------------------------------------
 0105000020E6100000030000000102000000020000000000000000F03F0000000000000000000000000000F03F00
 | SRID=4326;MULTILINESTRING((1 0,1 0),(2 0,3 0),(4 0,5 0,6 0))
```

ST_GeomFromGeoHash

ST_GeomFromGeoHash는 입력 지오메트리의 geohash 표현으로부터 지오메트리 객체를 구성합니다. ST_GeomFromGeoHash는 공간 참조 식별자(SRID)가 영(0)인 2차원(2D) 기하학을 반환합니다. geohash 형식에 대한 자세한 내용은 Wikipedia의 Geohash를 참조하세요.

구문

```
ST_GeomFromGeoHash(geohash_string)
```

```
ST_GeomFromGeoHash(geohash_string, precision)
```

인수

geohash_string

데이터 형식 VARCHAR의 값 또는 지오메트리의 geohash 표현인 VARCHAR 형식으로 평가되는 표현식입니다.

precision

geohash의 정밀도를 나타내는 데이터 형식 INTEGER의 값입니다. 값은 정밀도로 사용할 geohash의 문자 수입니다. 값이 지정되지 않은 경우 0보다 작거나 geohash_string 길이보다 큽니다. 그런 다음 geohash_string 길이가 사용됩니다.

반환 타입

GEOMETRY

geohash_string이 null이면 null이 반환됩니다.

geohash_string이 유효하지 않으면 오류가 반환됩니다.

예시

다음 SQL은 정밀도가 높은 다각형을 반환합니다.

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxncgyy4d0dbxqz0'));
```

```
 st_asewkt
-----------------------
 POLYGON((-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646,-115.172816
 36.114646,-115.172816 36.114646))
```

다음 SQL은 정밀도가 높은 점을 반환합니다.

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxncgyy4d0dbxqz00'));
```

```
 st_asewkt
-----------------------
 POINT(-115.172816 36.114646)
```

다음 SQL은 정밀도가 낮은 다각형을 반환합니다.

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qq'));
```

```
 st_asewkt
-----------------------
 POLYGON((-115.3125 35.15625,-115.3125 36.5625,-113.90625 36.5625,-113.90625
 35.15625,-115.3125 35.15625))
```

다음 SQL은 정밀도가 낮은 점을 반환합니다.

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 3));
```

```
 st_asewkt
-----------------------
 POLYGON((-115.3125 35.15625,-115.3125 36.5625,-113.90625 36.5625,-113.90625
 35.15625,-115.3125 35.15625))
```

ST_GeomFromGeoJSON

ST_GeomFromGeoJSON은 입력 지오메트리의 GeoJSON 표현으로부터 지오메트리 객체를 구성합니다. GeoJSON 형식에 대한 자세한 내용은 Wikipedia의 GeoJSON을 참조하세요.

좌표가 세 개 이상인 점이 하나 이상 있는 경우 결과 지오메트리는 3DZ이며, 여기서 Z 구성 요소는 좌표가 두 개뿐인 점의 경우 0입니다. 입력 GeoJSON의 모든 점이 두 개의 좌표를 포함하거나 비어 있는 경우 ST_geomFromGeoJSON은 2D 지오메트리를 반환합니다. 반환된 지오메트리의 공간 참조 식별자(SRID) 값은 언제나 4326입니다.

구문

```
ST_GeomFromGeoJSON(geojson_string)
```

인수

geojson_string

데이터 형식 VARCHAR의 값 또는 지오메트리의 GeoJSON 표현인 VARCHAR 형식으로 평가되는 표현식입니다.

반환 타입

GEOMETRY

geojson_string이 null이면 null이 반환됩니다.

geojson_string이 유효하지 않으면 오류가 반환됩니다.

예시

다음 SQL은 입력 GeoJSON에 표현된 2D 지오메트리를 반환합니다.

```
SELECT ST_AsEWKT(ST_GeomFromGeoJSON('{"type":"Point","coordinates":[1,2]}'));
```

```
 st_asewkt
-----------------------
 SRID=4326;POINT(1 2)
```

다음 SQL은 입력 GeoJSON에 표현된 3DZ 지오메트리를 반환합니다.

```
SELECT ST_AsEWKT(ST_GeomFromGeoJSON('{"type":"LineString","coordinates":[[1,2,3],
[4,5,6],[7,8,9]]}'));
```

```
 st_asewkt
-----------------------------------------
 SRID=4326;LINESTRING Z (1 2 3,4 5 6,7 8 9)
```

다음 SQL은 한 점에만 세 개의 좌표가 있고 다른 모든 점은 입력 GeoJSON에 두 개의 좌표가 있는 경우 3DZ 지오메트리를 반환합니다.

```
SELECT ST_AsEWKT(ST_GeomFromGeoJSON('{"type":"Polygon","coordinates":[[[0, 0],[0, 1,
 8],[1, 0],[0, 0]]]}'));
```

```
 st_asewkt
-----------------------------------------------
 SRID=4326;POLYGON Z ((0 0 0,0 1 8,1 0 0,0 0 0))
```

ST_GeomFromGeoSquare

ST_GeomFromGeoSquare는 입력된 지오스퀘어 값으로 표시되는 영역을 포함하는 지오메트리를 반환합니다. 반환되는 지오메트리는 항상 2차원입니다. 지리 제곱 값을 계산하려면 ST_GeoSquare 단원을 참조하세요.

구문

```
ST_GeomFromGeoSquare(geosquare)
```

```
ST_GeomFromGeoSquare(geosquare, max_depth)
```

인수

geosquare

원하는 제곱에 도달하기 위해 초기 도메인에서 수행한 세분화 시퀀스를 설명하는 지오스퀘어 값인 데이터 형식 BIGINT 또는 BIGINT 형식으로 평가되는 표현식의 값입니다. 이 값은 ST_GeoSquare에 의해 계산됩니다.

max_depth

초기 도메인에서 만들어진 최대 도메인 세분화 수를 나타내는 데이터 형식 INTEGER의 값입니다. 값은 1보다 크거나 같아야 합니다.

반환 타입

GEOMETRY

geosquare가 유효하지 않으면 함수는 오류를 반환합니다.

입력 max_depth가 범위 내에 있지 않으면 함수는 오류를 반환합니다.

예시

다음 SQL은 지오스퀘어 값에서 지오메트리를 반환합니다.

```
SELECT ST_AsText(ST_GeomFromGeoSquare(797852));
```

```
 st_astext
```

```
----------------------------------------------------------------------------------------------------
 POLYGON((13.359375 52.3828125,13.359375 52.734375,13.7109375 52.734375,13.7109375
 52.3828125,13.359375 52.3828125))
```

다음 SQL은 지오스퀘어 값과 최대 깊이 3의 지오메트리를 반환합니다.

```
SELECT ST_AsText(ST_GeomFromGeoSquare(797852, 3));
```

```
 st_astext
-------------------------------------
 POLYGON((0 45,0 90,45 90,45 45,0 45))
```

다음 SQL은 먼저 x 좌표를 경도로, y 좌표를 위도(-122.3, 47.6)로 지정하여 시애틀에 대한 지오스퀘어 값을 계산합니다. 그런 다음 지오스퀘어에 대한 폴리곤을 반환합니다. 출력은 2차원 지오메트리이지만 경도와 위도 측면에서 공간 데이터를 계산하는 데 사용할 수 있습니다.

```
SELECT ST_AsText(ST_GeomFromGeoSquare(ST_GeoSquare(ST_Point(-122.3, 47.6))));
```

```
 st_astext
----------------------------------------------------------------------------------------------------
POLYGON((-122.335167014971 47.6080129947513,-122.335167014971
 47.6080130785704,-122.335166931152 47.6080130785704,-122.335166931152
 47.6080129947513,-122.335167014971 47.6080129947513))
```

ST_GeomFromText

ST_GeomFromText는 입력 지오메트리의 WKT(Well-Known Text) 표현으로부터 지오메트리 객체를 구성합니다.

ST_GeomFromText는 3DZ, 3DM 및 4D를 허용합니다. 여기서 지오메트리 유형에는 각각 Z, M 또는 ZM 접두사가 붙습니다.

구문

```
ST_GeomFromText(wkt_string)
```

```
ST_GeomFromText(wkt_string, srid)
```

인수

wkt_string

지오메트리의 WKT 표현인 VARCHAR 데이터 형식의 값입니다.

WKT 키워드 EMPTY를 사용하여 빈 점, 빈 점이 있는 다중 점 또는 빈 점이 있는 지오메트리 컬렉션을 지정할 수 있습니다. 다음 예에서는 하나의 빈 점과 하나의 비어 있지 않은 점이 있는 다중 점을 생성합니다.

```
ST_GeomFromEWKT('MULTIPOINT(1 0,EMPTY)');
```

srid

SRID(공간 참조 식별자)인 INTEGER 데이터 형식의 값입니다. SRID 값이 제공되면 반환된 지오메트리에 이 SRID 값이 있습니다. 그렇지 않으면 반환된 지오메트리의 SRID 값이 0으로 설정됩니다.

반환 타입

GEOMETRY

wkt_string 또는 srid가 null이면 null이 반환됩니다.

srid가 음수이면 null이 반환됩니다.

wkt_string이 유효하지 않으면 오류가 반환됩니다.

srid가 유효하지 않으면 오류가 반환됩니다.

예시

다음 SQL은 WKT 표현 및 SRID 값으로부터 지오메트리 객체를 구성합니다.

```
SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326);
```

```
st_geomfromtext
----------------------------------
0103000020E6100000010000000500000000000000000000000000000000000000000000000000F0
```

ST_GeomFromWKB

ST_GeomFromWKB는 입력 지오메트리의 16진수 WKB(Well-Known Binary) 표현으로부터 지오메트리 객체를 구성합니다.

ST_GeomFromWKB는 WKB 16진수 형식으로 작성된 3DZ, 3DM 및 4D 지오메트리를허용합니다.

구문

```
ST_GeomFromWKB(wkb_string)
```

```
ST_GeomFromWKB(wkb_string, srid)
```

인수

wkb_string

지오메트리의 16진수 WKB 표현인 VARCHAR 데이터 형식의 값입니다.

srid

SRID(공간 참조 식별자)인 INTEGER 데이터 형식의 값입니다. SRID 값이 제공되면 반환된 지오메트리에 이 SRID 값이 있습니다. 그렇지 않으면 반환된 지오메트리의 SRID 값이 0으로 설정됩니다.

반환 타입

GEOMETRY

wkb_string 또는 srid가 null이면 null이 반환됩니다.

srid가 음수이면 null이 반환됩니다.

wkb_string이 유효하지 않으면 오류가 반환됩니다.

srid가 유효하지 않으면 오류가 반환됩니다.

예시

다음 SQL은 WKB 값에서 다각형을 구성하고 다각형의 WKT 표현을 반환합니다.

```
SELECT
 ST_AsText(ST_GeomFromWKB('010300000001000000050000000000000000000000000000000000000000000000000
```

```
 st_astext
-------------------------------
 POLYGON((0 0,0 1,1 1,1 0,0 0))
```

ST_GeoSquare

ST_GeoSquare는 도메인([-180, 180], [-90, 90])을 지정된 깊이까지 지오스퀘어로 불리는 동일한 정사각형 리전으로 재귀적으로 세분화합니다. 세분화는 제공된 점의 위치를 기반으로 합니다. 점을 포함하는 지오스퀘어 중 하나는 최대 깊이에 도달할 때까지 각 단계에서 세분화됩니다. 이 지오스퀘어의 선택은 안정적입니다. 즉 함수 결과는 입력 인자에만 의존합니다. 이 함수는 점이 위치한 최종 지오스퀘어를 식별하는 고유 값을 반환합니다.

ST_GeoSquare는 x 좌표가 경도를 나타내고 y 좌표가 위도를 나타내는 POINT를 받습니다. 경도와 위도는 각각 [-180, 180] 및 [-90, 90]으로 제한됩니다. ST_GeoSquare의 출력은 ST_GeomFromGeoSquare 함수에 대한 입력으로 사용할 수 있습니다.

지구의 적도 원주 호를 중심으로 360°를 두 개의 반구(동반구 및 서반구)로 나누고, 각각 0° 자오선에서 180°의 세로선(자오선)을 갖습니다. 규칙에 따라 동경은 데카르트 평면의 X축에 투영할 때 "+"(양수) 좌표이고, 서경은 데카르트 평면의 X축에 투영할 때 "-"(음수) 좌표입니다. 지구의 적도 둘레 0°를 기준으로 북쪽과 남쪽에 각각 90°의 위도선이 있으며, 이 위도선은 지구의 적도 둘레 0°와 평행합니다. 규칙에 따라 북위도선은 데카르트 평면에 투영할 때 "+"(양수) y축과 교차하고, 남위도선은 데카르트 평면에 투영할 때 "-"(음수) y축과 교차합니다. 경도선과 위도선이 교차하여 형성된 구형 격자는 직교 평면에 투영된 격자로 변환되며, 직교 평면의 표준 양수 및 음수 x 좌표와 양수 및 음수 y 좌표로 변환됩니다.

ST_GeoSquare의 목적은 동일한 코드 값으로 가까운 점을 태그하거나 표시하는 것입니다. 동일한 지오스퀘어에 있는 포인트는 동일한 코드 값을 받습니다. 지오스퀘어는 지리적 좌표(위도 및 경도)를 정수로 인코딩하는 데 사용됩니다. 더 큰 리전은 다양한 해상도로 맵에 영역을 묘사하기 위해 그리드로 나뉩니다. 지오스퀘어는 공간 인덱싱, 공간 비닝, 근접 검색, 위치 검색, 고유한 장소 식별자 생성에 사용할 수 있습니다. ST_GeoHash 함수는 리전을 그리드로 나누는 유사한 프로세스를 따르지만 인코딩이 다릅니다.

구문

```
ST_GeoSquare(geom)
```

```
ST_GeoSquare(geom, max_depth)
```

인수

geom

데이터 형식 GEOMETRY의 POINT 값 또는 POINT 하위 유형으로 평가되는 표현식입니다. 점의 x 좌표(경도)는 -180 - 180 범위 내에 있어야 합니다. 점의 y 좌표(위도)는 -90 - 90 범위 내에 있어야 합니다.

max_depth

INTEGER 데이터 형식의 값입니다. 점을 포함하는 도메인이 재귀적으로 세분되는 최대 횟수입니다. 값은 1 ~ 32 사이의 정수여야 합니다. 기본값은 32입니다. 세분화의 실제 최종 개수가 지정된 max_depth보다 작거나 같아야 합니다.

반환 타입

BIGINT

이 함수는 입력 지점이 위치한 최종 지오스퀘어를 식별하는 고유 값을 반환합니다.

입력 geom이 점이 아닌 경우 함수는 오류를 반환합니다.

입력 점이 비어 있는 경우 반환 값은 ST_GeomFromGeoSquare 함수에 대한 유효한 입력이 아닙니다. 빈 점으로 ST_GeoSquare를 호출하지 않도록 하려면 ST_IsEmpty 함수를 사용하세요.

입력 점이 범위 내에 있지 않으면 함수는 오류를 반환합니다.

입력 max_depth가 범위를 벗어나면 함수는 오류를 반환합니다.

예시

다음 SQL은 입력 점에서 지오스퀘어를 반환합니다.

```
SELECT ST_GeoSquare(ST_Point(13.5, 52.5));
```

```
  st_geosquare
-----------------------
 -4410772491521635895
```

다음 SQL은 입력 점에서 최대 깊이가 10인 지오스퀘어를 반환합니다.

```
SELECT ST_GeoSquare(ST_Point(13.5, 52.5), 10);
```

```
 st_geosquare
--------------
 797852
```

ST_InteriorRingN

ST_InteriorRingN은 인덱스 위치에서 입력 다각형의 내부 링에 해당하는 닫힌 라인스트링을 반환합니다. 반환된 지오메트리의 차원은 입력 지오메트리의 차원과 같습니다.

구문

```
ST_InteriorRingN(geom, index)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

인덱스를 구축하고 배포할 것입니다

1부터 시작하는 인덱스 링의 위치를 나타내는 INTEGER 데이터 형식의 값입니다.

반환 타입

LINESTRING 하위 유형의 GEOMETRY입니다.

반환된 지오메트리의 공간 참조 시스템 식별자(SRID) 값은 입력 지오메트리의 SRID 값입니다.

geom 또는 index가 null이면 null이 반환됩니다.

index가 범위를 벗어난 경우 null이 반환됩니다.

geom이 다각형이 아니면 null이 반환됩니다.

geom이 빈 다각형이면 null이 반환됩니다.

예시

다음 SQL은 다각형의 두 번째 링을 닫힌 라인스트링으로 반환합니다.

```
SELECT ST_AsEWKT(ST_InteriorRingN(ST_GeomFromText('POLYGON((7 9,8 7,11 6,15 8,16 6,17
 7,17 10,18 12,17 14,15 15,11 15,10 13,9 12,7 9),(9 9,10 10,11 11,11 10,10 8,9 9),(12
 14,15 14,13 11,12 14))'),2));
```

```
st_asewkt
-----------
 LINESTRING(12 14,15 14,13 11,12 14)
```

ST_Intersects

ST_Intersects는 두 입력 지오메트리의 2D 프로젝션에 공통되는 점이 하나 이상인 경우 true를 반환합니다.

구문

```
ST_Intersects(geom1, geom2)
```

인수

geom1

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

geom2

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

BOOLEAN

geom1 또는 geom2가 null이면 null이 반환됩니다.

geom1과 geom2의 SRID(공간 참조 시스템 식별자) 값이 같지 않으면 오류가 반환됩니다.

geom1 또는 geom2가 지오메트리 컬렉션인 경우 오류가 반환됩니다.

예시

다음 SQL은 첫 번째 다각형이 두 번째 다각형과 교차하는지 확인합니다.

```
SELECT ST_Intersects(ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(2 2,2 5,5 5,5
 2,2 2))'), ST_GeomFromText('MULTIPOINT((4 4),(6 6))'));
```

```
st_intersects
-------------
 true
```

ST_Intersection

ST_Intersection은 두 지오메트리의 점 집합 교차를 나타내는 지오메트리를 반환합니다. 즉, 두 입력 지오메트리 간에 공유되는 부분을 반환합니다.

구문

```
ST_Intersection(geom1, geom2)
```

인수

geom1

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

geom2

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

GEOMETRY

geom1과 geom2가 공간을 공유하지 않으면(분리되어 있음) 빈 지오메트리가 반환됩니다.

geom1 또는 geom2가 비어 있으면 빈 지오메트리가 반환됩니다.

geom1과 geom2의 SRID(공간 참조 시스템 식별자) 값이 같지 않으면 오류가 반환됩니다.

geom1 또는 geom2가 지오메트리 컬렉션인 경우 오류가 반환됩니다.

geom1 또는 geom2가 2차원(2D) 지오메트리가 아니면 오류가 반환됩니다.

예시

다음 SQL은 두 입력 지오메트리의 교차를 나타내는 비어 있지 않은 지오메트리를 반환합니다.

```
SELECT ST_AsEWKT(ST_Intersection(ST_GeomFromText('polygon((0 0,100 100,0 200,0 0))'),
 ST_GeomFromText('polygon((0 0,10 0,0 10,0 0))')));
```

```
        st_asewkt
-------------------------
 POLYGON((0 0,0 10,5 5,0 0))
```

다음 SQL은 분리되어 있는(교차하지 않는) 입력 지오메트리를 전달할 때 빈 지오메트리를 반환합니다.

```
SELECT ST_AsEWKT(ST_Intersection(ST_GeomFromText('linestring(0 100,0 0)'),
 ST_GeomFromText('polygon((1 0,10 0,1 10,1 0))')));
```

```
    st_asewkt
------------------
 LINESTRING EMPTY
```

ST_IsPolygonCCW

ST_IsPolygonCCW는 입력 Polygon 또는 Multipolygon의 2D 프로젝션이 시계 반대 방향인 경우 true를 반환합니다. 입력 지오메트리가 점, 라인스트링, 다중 점 또는 다중 라인스트링이면 true가 반환됩니다. 지오메트리 컬렉션의 경우 컬렉션의 모든 지오메트리가 시계 반대 방향이면 ST_IsPolygonCCW가 true를 반환합니다.

구문

```
ST_IsPolygonCCW(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

BOOLEAN

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 다각형이 시계 반대 방향인지 확인합니다.

```
SELECT ST_IsPolygonCCW(ST_GeomFromText('POLYGON((7 9,8 7,11 6,15 8,16 6,17 7,17 10,18
 12,17 14,15 15,11 15,10 13,9 12,7 9),(9 9,10 10,11 11,11 10,10 8,9 9),(12 14,15 14,13
 11,12 14))'));
```

```
 st_ispolygonccw
----------
 true
```

ST_IsPolygonCW

ST_IsPolygonCW는 입력 Polygon 또는 Multipolygon의 2D 프로젝션이 시계 방향인 경우 true를 반환합니다. 입력 지오메트리가 점, 라인스트링, 다중 점 또는 다중 라인스트링이면 true가 반환됩니다. 지오메트리 컬렉션의 경우 컬렉션의 모든 지오메트리가 시계 방향이면 ST_IsPolygonCW가 true를 반환합니다.

구문

```
ST_IsPolygonCW(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

BOOLEAN

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 다각형이 시계 방향인지 확인합니다.

```
SELECT ST_IsPolygonCW(ST_GeomFromText('POLYGON((7 9,8 7,11 6,15 8,16 6,17 7,17 10,18
 12,17 14,15 15,11 15,10 13,9 12,7 9),(9 9,10 10,11 11,11 10,10 8,9 9),(12 14,15 14,13
 11,12 14))'));
```

```
 st_ispolygonccw
----------
 true
```

ST_IsClosed

ST_IsClosed는 입력 지오메트리의 2D 프로젝션이 닫혀 있으면 true를 반환합니다. 다음 규칙은 닫힌 지오메트리를 정의합니다.

- 입력 지오메트리가 점 또는 다중 점입니다.
- 입력 지오메트리가 라인스트링이며 라인스트링의 시작 점과 끝 점이 일치합니다.
- 입력 지오메트리가 비어 있지 않은 다중 라인스트링이며 해당 라인스트링이 모두 닫혀 있습니다.
- 입력 지오메트리가 비어 있지 않은 다각형이고 모든 다각형의 링이 비어 있지 않으며 모든 링의 시작 점과 끝 점이 일치합니다.
- 입력 지오메트리는 비어 있지 않은 다중 다각형이며 모든 다각형이 닫혀 있습니다.
- 입력 지오메트리가 비어 있지 않은 지오메트리 컬렉션이며 모든 구성 요소가 닫혀 있습니다.

구문

```
ST_IsClosed(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

BOOLEAN

geom이 빈 점이면 false가 반환됩니다.

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 다각형이 닫혀 있는지 확인합니다.

```
SELECT ST_IsClosed(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'));
```

```
st_isclosed
-----------
 true
```

ST_IsCollection

ST_IsCollection은 입력 지오메트리가 GEOMETRYCOLLECTION, MULTIPOINT, MULTILINESTRING 또는 MULTIPOLYGON 하위 유형 중 하나인 경우 true를 반환합니다.

구문

```
ST_IsCollection(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

BOOLEAN

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 다각형이 컬렉션인지 확인합니다.

```
SELECT ST_IsCollection(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'));
```

```
st_iscollection
-----------
 false
```

ST_IsEmpty

ST_IsEmpty는 입력 지오메트리가 비어 있는 경우 true를 반환합니다. 하나 이상의 비어 있지 않은 점이 포함된 지오메트리는 비어 있지 않습니다.

ST_IsEmpty는 입력 지오메트리에 비어 있지 않은 점이 하나 이상 있으면 true를 반환합니다.

구문

```
ST_IsEmpty(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

BOOLEAN

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 지정된 다각형이 비어 있는지 확인합니다.

```
SELECT ST_IsEmpty(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'));
```

```
st_isempty
-----------
 false
```

ST_IsRing

입력 라인스트링이 링이면 ST_IsRing은 true를 반환합니다. 라인스트링은 닫혀 있고 단순하면 링입니다.

구문

```
ST_IsRing(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 지오메트리는 LINESTRING이어야 합니다.

반환 타입

BOOLEAN

geom이 LINESTRING이 아니면 오류가 반환됩니다.

예시

다음 SQL은 지정된 라인스트링이 링인지 확인합니다.

```
SELECT ST_IsRing(ST_GeomFromText('linestring(0 0, 1 1, 1 2, 0 0)'));
```

```
st_isring
-----------
 true
```

ST_IsSimple

ST_IsSimple은 입력 지오메트리의 2D 프로젝션이 단순하면 true를 반환합니다. 단순 지오메트리의 정의에 대한 자세한 내용은 기하학적 단순성 섹션을 참조하세요.

구문

```
ST_IsSimple(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

BOOLEAN

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 지정된 라인스트링이 단순한지 확인합니다. 이 예에서는 자체 교차점이 있기 때문에 단순하지 않습니다.

```
SELECT ST_IsSimple(ST_GeomFromText('LINESTRING(0 0,10 0,5 5,5 -5)'));
```

```
 st_issimple
-----------
 false
```

ST_IsValid

ST_IsValid는 입력 지오메트리의 2D 프로젝션이 유효하면 true를 반환합니다. 유효한 지오메트리의 정의에 대한 자세한 내용은 기하학적 유효성 섹션을 참조하세요.

구문

```
ST_IsValid(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

BOOLEAN

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 지정된 다각형이 유효한지 확인합니다. 이 예에서는 다각형 내부가 단순히 연결되어 있지 않기 때문에 다각형이 유효하지 않습니다.

```
SELECT ST_IsValid(ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(5 0,10 5,5 10,0
 5,5 0))'));
```

```
 st_isvalid
------------
 false
```

ST_Length

선형 지오메트리의 경우 ST_Length는 2D 프로젝션의 데카르트 길이를 반환합니다. 길이 단위는 입력 지오메트리의 좌표가 표현되는 단위와 동일합니다. 이 함수는 점, 다중 점 및 영역 지오메트리에 대해 0을 반환합니다. 입력이 지오메트리 컬렉션인 경우 이 함수는 컬렉션에 있는 지오메트리 길이의 합계를 반환합니다.

지오그래피의 경우 ST_Length는 SRID에 의해 결정된 회전 타원체에서 계산된 입력 선형 지오그래피의 2D 투영의 측지 길이를 반환합니다. 길이 단위는 미터입니다. 이 함수는 점, 다중 점 및 영역 지오그래피에 대해 0을 반환합니다. 입력이 지오메트리 컬렉션인 경우 이 함수는 컬렉션에 있는 지오그래피 길이의 합계를 반환합니다.

구문

```
ST_Length(geo)
```

인수

geo

GEOMETRY 또는 GEOGRAPHY 데이터 유형의 값이나 GEOMETRY 또는 GEOGRAPHY 유형으로 계산되는 표현식입니다.

반환 타입

DOUBLE PRECISION

geo가 null이면 null이 반환됩니다.

SRID 값을 찾을 수 없으면 오류가 반환됩니다.

예시

다음 SQL은 다중 라인스트링의 데카르트 길이를 반환합니다.

```
SELECT ST_Length(ST_GeomFromText('MULTILINESTRING((0 0,10 0,0 10),(10 0,20 0,20
 10))'));
```

```
st_length
--------------------------------
  44.142135623731
```

다음 SQL은 지오그래피에서 라인스트링의 길이를 반환합니다.

```
SELECT ST_Length(ST_GeogFromText('SRID=4326;LINESTRING(5 0,6 0,4 0)'));
```

```
 st_length
------------------
 333958.472379804
```

다음 SQL은 지오그래피에서 점의 길이를 반환합니다.

```
SELECT ST_Length(ST_GeogFromText('SRID=4326;POINT(4 5)'));
```

```
 st_length
-----------
 0
```

ST_LengthSphere

ST_LengthSphere는 선형 지오메트리의 길이(미터 단위)를 반환합니다. 점, 다중 점 및 영역 지오메트리의 경우 ST_LengthSphere는 0을 반환합니다. 지오메트리 컬렉션의 경우 ST_LengthSphere는 컬렉션에 있는 선형 지오메트리의 총 길이(미터 단위)를 반환합니다.

ST_LengthSphere는 입력 지오메트리의 각 점 좌표를 경도 및 위도(도)로 해석합니다. 3DZ, 3DM 또는 4D 지오메트리의 경우 처음 두 좌표만 사용됩니다.

구문

```
ST_LengthSphere(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

DOUBLE PRECISION 길이(미터 단위)입니다. 길이 계산은 반지름이 Earth의 WGS(World Geodetic System) 84 타원체 모델의 평균 반지름인 Earth의 구형 모델을 기반으로 합니다.

geom이 null이면 null이 반환됩니다.

예시

다음 예제 SQL은 라인스트링의 길이(미터 단위)를 계산합니다.

```
SELECT ST_LengthSphere(ST_GeomFromText('LINESTRING(10 10,45 45)'));
```

```
 st_lengthsphere
------------------
 5127736.08292556
```

ST_Length2D

ST_Length2D는 ST_Length의 별칭입니다. 자세한 내용은 ST_Length 단원을 참조하십시오.

ST_LineFromMultiPoint

ST_LineFromMultiPoint는 입력 다중 점 지오메트리에서 라인스트링을 반환합니다. 점의 순서는 유지됩니다. 반환된 지오메트리의 SRID(공간 참조 시스템 식별자)는 입력 지오메트리의 값과 동일합니다. 반환된 지오메트리의 차원은 입력 지오메트리의 차원과 같습니다.

구문

```
ST_LineFromMultiPoint(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 MULTIPOINT이어야 합니다.

반환 타입

GEOMETRY

geom이 null이면 null이 반환됩니다.

geom이 비어 있으면 빈 LINESTRING이 반환됩니다.

geom에 빈 점이 포함된 경우 이러한 빈 점은 무시됩니다.

geom이 MULTIPOINT가 아니면 오류가 반환됩니다.

예시

다음 SQL은 다중 점에서 라인스트링을 생성합니다.

```
SELECT ST_AsEWKT(ST_LineFromMultiPoint(ST_GeomFromText('MULTIPOINT(0 0,10 0,10 10,5 5,0
 5)',4326)));
```

```
 st_asewkt
--------------------------------------------
 SRID=4326;LINESTRING(0 0,10 0,10 10,5 5,0 5)
```

ST_LineInterpolatePoint

ST_LineInterpolatePoint는 선의 시작 부분에서 분수 거리에 있는 선을 따라 점을 반환합니다.

점 동등성을 판별하기 위해 ST_LineInterpolatePoint는 입력 지오메트리의 2D 프로젝션에서 작동합니다. 입력 지오메트리가 비어 있으면 해당 복사본이 입력과 동일한 차원으로 반환됩니다. 3DZ, 3DM 및 4D 지오메트리의 경우 z 또는 m 좌표는 점이 있는 세그먼트의 z 또는 m 좌표의 평균입니다.

구문

```
ST_LineInterpolatePoint(geom, fraction)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 LINESTRING입니다.

fraction

선에 대한 라인스트링을 따라 점의 위치를 나타내는 DOUBLE PRECISION 데이터 형식의 값입니다. 값은 0~1(포함) 범위의 분수입니다.

반환 타입

POINT 하위 유형의 GEOMETRY입니다.

geom 또는 fraction이 null이면 null이 반환됩니다.

geom이 비어 있으면 빈 점이 반환됩니다.

반환된 지오메트리의 공간 참조 시스템 식별자(SRID) 값은 입력 지오메트리의 SRID 값입니다.

fraction이 범위를 벗어나면 오류가 반환됩니다.

geom이 라인스트링이 아니면 오류가 반환됩니다.

예시

다음 SQL은 라인스트링을 따라 중간 점을 반환합니다.

```
SELECT ST_AsEWKT(ST_LineInterpolatePoint(ST_GeomFromText('LINESTRING(0 0, 5 5, 7 7, 10
 10)'), 0.50));
```

```
st_asewkt
-----------
 POINT(5 5)
```

다음 SQL은 라인스트링을 따라 90% 점을 반환합니다.

```
SELECT ST_AsEWKT(ST_LineInterpolatePoint(ST_GeomFromText('LINESTRING(0 0, 5 5, 7 7, 10
 10)'), 0.90));
```

```
st_asewkt
-----------
 POINT(9 9)
```

ST_M

ST_M은 입력 점의 m 좌표를 반환합니다.

구문

```
ST_M(point)
```

인수

point

GEOMETRY 데이터 형식의 POINT 값입니다.

반환 타입

m 좌표의 DOUBLE PRECISION 값입니다.

point가 null이면 null이 반환됩니다.

point가 2D 또는 3DZ 점이면 null이 반환됩니다.

point가 빈 점이면 null이 반환됩니다.

point가 POINT가 아니면 오류가 반환됩니다.

예시

다음 SQL은 3DM 지오메트리에서 점의 m 좌표를 반환합니다.

```
SELECT ST_M(ST_GeomFromEWKT('POINT M (1 2 3)'));
```

```
st_m
-----------
 3
```

다음 SQL은 4D 지오메트리에서 점의 m 좌표를 반환합니다.

```
SELECT ST_M(ST_GeomFromEWKT('POINT ZM (1 2 3 4)'));
```

```
st_m
-----------
 4
```

ST_MakeEnvelope

ST_MakeEnvelope은 다음과 같이 지오메트리를 반환합니다.

- 입력 좌표에서 점을 지정하면 반환된 지오메트리는 점입니다.
- 입력 좌표에서 선을 지정하면 반환된 지오메트리는 라인스트링입니다.
- 그렇지 않으면 반환된 지오메트리는 입력 좌표가 상자의 왼쪽 아래 모서리와 오른쪽 위 모서리를 지정하는 다각형입니다.

제공되는 경우 반환된 지오메트리의 공간 참조 시스템 식별자(SRID) 값은 입력 지오메트리의 SRID 값으로 설정됩니다.

구문

```
ST_MakeEnvelope(xmin, ymin, xmax, ymax)
```

```
ST_MakeEnvelope(xmin, ymin, xmax, ymax, srid)
```

인수

xmin

DOUBLE PRECISION 데이터 형식의 값입니다. 이 값은 상자의 왼쪽 아래 모서리의 첫 번째 좌표입니다.

ymin

DOUBLE PRECISION 데이터 형식의 값입니다. 이 값은 상자의 왼쪽 아래 모서리의 두 번째 좌표입니다.

xmax

DOUBLE PRECISION 데이터 형식의 값입니다. 이 값은 상자의 오른쪽 위 모서리의 첫 번째 좌표입니다.

ymax

DOUBLE PRECISION 데이터 형식의 값입니다. 이 값은 상자의 오른쪽 위 모서리의 두 번째 좌표입니다.

srid

공간 참조 시스템 식별자(SRID)를 나타내는 INTEGER 데이터 형식의 값입니다. SRID 값이 제공되지 않으면 0으로 설정됩니다.

반환 타입

하위 유형 POINT, LINESTRING 또는 POLYGON의 GEOMETRY입니다.

반환된 지오메트리의 SRID는 srid 또는 0(srid가 설정되지 않은 경우)으로 설정됩니다.

xmin, ymin, xmax, ymax 또는 srid가 null이면 null이 반환됩니다.

srid가 음수이면 오류가 반환됩니다.

예시

다음 SQL은 4개의 입력 좌표 값으로 정의된 봉투를 나타내는 다각형을 반환합니다.

```
SELECT ST_AsEWKT(ST_MakeEnvelope(2,4,5,7));
```

```
 st_astext
```

```
---------------
 POLYGON((2 4,2 7,5 7,5 4,2 4))
```

다음 SQL은 4개의 입력 좌표 값과 SRID 값으로 정의된 봉투를 나타내는 다각형을 반환합니다.

```
SELECT ST_AsEWKT(ST_MakeEnvelope(2,4,5,7,4326));
```

```
 st_astext
-----------------------------------
 SRID=4326;POLYGON((2 4,2 7,5 7,5 4,2 4))
```

ST_MakeLine

ST_MakeLine은 입력 지오메트리에서 라인스트링을 생성합니다.

반환된 지오메트리의 차원은 입력 지오메트리의 차원과 같습니다. 두 입력 지오메트리의 차원이 같아야 합니다.

구문

```
ST_MakeLine(geom1, geom2)
```

인수

geom1

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 POINT, LINESTRING, 또는 MULTIPOINT이어야 합니다.

geom2

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 POINT, LINESTRING, 또는 MULTIPOINT이어야 합니다.

반환 타입

LINESTRING 하위 유형의 GEOMETRY입니다.

geom1 또는 geom2가 null이면 null이 반환됩니다.

geom1 및 geom2가 빈 점이거나 빈 점을 포함하면 이러한 빈 점은 무시됩니다.

geom1 및 geom2가 비어 있으면 빈 LINESTRING이 반환됩니다.

반환된 지오메트리의 SRID(공간 참조 시스템 식별자) 값은 입력 지오메트리의 SRID 값입니다.

geom1과 geom2의 SRID 값이 다른 경우 오류가 반환됩니다.

geom1 또는 geom2가 POINT, LINESTRING 또는 MULTIPOINT가 아니면 오류가 반환됩니다.

geom1과 geom2의 차원이 다른 경우 오류가 반환됩니다.

예시

다음 SQL은 두 개의 입력 라인스트링에서 라인스트링을 구성합니다.

```
SELECT ST_MakeLine(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27
 29.31,77.29 29.07)'), ST_GeomFromText('LINESTRING(88.29 39.07,88.42 39.26,88.27
 39.31,88.29 39.07)'));
```

```
st_makeline
-----------
 01020000000800000000C3F5285C8F52534052B81E85EB113D407B14AE47E15A5340C3F5285C8F423D40E17A14AE4751
```

ST_MakePoint

ST_MakePoint는 좌표 값이 입력 값인 점 지오메트리를 반환합니다.

구문

```
ST_MakePoint(x, y)
```

```
ST_MakePoint(x, y, z)
```

```
ST_MakePoint(x, y, z, m)
```

인수

x

첫 번째 좌표를 나타내는 DOUBLE PRECISION 데이터 형식의 값입니다.

y

두 번째 좌표를 나타내는 DOUBLE PRECISION 데이터 형식의 값입니다.

z

세 번째 좌표를 나타내는 DOUBLE PRECISION 데이터 형식의 값입니다.

m

네 번째 좌표를 나타내는 DOUBLE PRECISION 데이터 형식의 값입니다.

반환 타입

POINT 하위 유형의 GEOMETRY입니다.

반환된 지오메트리의 SRID(공간 참조 시스템 식별자) 값은 0으로 설정됩니다.

x, y, z 또는 m이 null이면 null이 반환됩니다.

예시

다음 SQL은 제공된 좌표 값을 갖는 POINT 하위 유형의 GEOMETRY 유형을 반환합니다.

```
SELECT ST_AsText(ST_MakePoint(1,3));
```

```
st_astext
-----------
 POINT(1 3)
```

다음 SQL은 제공된 좌표 값을 갖는 POINT 하위 유형의 GEOMETRY 유형을 반환합니다.

```
SELECT ST_AsEWKT(ST_MakePoint(1, 2, 3));
```

```
st_asewkt
----------------
 POINT Z (1 2 3)
```

다음 SQL은 제공된 좌표 값을 갖는 POINT 하위 유형의 GEOMETRY 유형을 반환합니다.

```
SELECT ST_AsEWKT(ST_MakePoint(1, 2, 3, 4));
```

```
st_asewkt
-------------------
 POINT ZM (1 2 3 4)
```

ST_MakePolygon

ST_MakePolygon에는 다각형을 반환하는 두 가지 변형이 있습니다. 하나는 단일 지오메트리를 사용하고 다른 하나는 2개의 지오메트리를 사용합니다.

- 첫 번째 변형의 입력은 출력 다각형의 외부 링을 정의하는 라인스트링입니다.
- 두 번째 변형의 입력은 라인스트링과 다중 라인스트링입니다. 둘 다 비어 있거나 닫혀 있습니다.

  출력 다각형의 외부 링 경계는 입력 라인스트링이고 다각형의 내부 링 경계는 입력 다중 라인스트링의 라인스트링입니다. 입력 라인스트링이 비어 있으면 빈 다각형이 반환됩니다. 다중 라인스트링의 빈 라인스트링은 무시됩니다. 결과 지오메트리의 공간 참조 시스템 식별자(SRID)는 두 입력 지오메트리의 공통 SRID입니다.

반환된 지오메트리의 차원은 입력 지오메트리의 차원과 같습니다. 외부 링과 내부 링의 차원이 같아야 합니다.

구문

```
ST_MakePolygon(geom1)
```

```
ST_MakePolygon(geom1, geom2)
```

인수

geom1

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 LINESTRING이어야 합니다. linestring 값은 닫혀 있거나 비어 있어야 합니다.

geom2

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 MULTILINESTRING이어야 합니다.

반환 타입

POLYGON 하위 유형의 GEOMETRY입니다.

반환된 지오메트리의 공간 참조 시스템 식별자(SRID)는 입력의 SRID와 같습니다.

geom1 또는 geom2가 null이면 null이 반환됩니다.

geom1이 라인스트링이 아니면 오류가 반환됩니다.

geom2가 다중 라인스트링이 아니면 오류가 반환됩니다.

geom1이 닫히지 않으면 오류가 반환됩니다.

geom1이 단일 점이거나 닫혀 있지 않으면 오류가 반환됩니다.

geom2에 단일 점이 있거나 닫혀 있지 않은 라인스트링이 하나 이상 포함되어 있으면 오류가 반환됩니다.

geom1과 geom2의 SRID 값이 다른 경우 오류가 반환됩니다.

geom1과 geom2의 차원이 다른 경우 오류가 반환됩니다.

예시

다음 SQL은 입력 라인스트링에서 다각형을 반환합니다.

```
SELECT ST_AsText(ST_MakePolygon(ST_GeomFromText('LINESTRING(77.29 29.07,77.42
 29.26,77.27 29.31,77.29 29.07)')));
```

```
 st_astext
---------------
POLYGON((77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07))
```

다음 SQL은 닫힌 라인스트링과 닫힌 다중 라인스트링에서 다각형을 생성합니다. 라인스트링은 다각형의 외부 링에 사용됩니다. 다중 라인스트링의 라인스트링은 다각형의 내부 링에 사용됩니다.

```
SELECT ST_AsEWKT(ST_MakePolygon(ST_GeomFromText('LINESTRING(0 0,10 0,10 10,0 10,0 0)'),
 ST_GeomFromText('MULTILINESTRING((1 1,1 2,2 1,1 1),(3 3,3 4,4 3,3 3))')));
```

```
 st_astext
```

```
---------------------------------
POLYGON((0 0,10 0,10 10,0 10,0 0),(1 1,1 2,2 1,1 1),(3 3,3 4,4 3,3 3))
```

ST_MemSize

ST_MemSize는 입력 지오메트리에서 사용하는 메모리 공간의 양(바이트)을 반환합니다. 이 크기는 지오메트리의 Amazon Redshift 내부 표현에 따라 달라지므로 내부 표현이 변경되면 변경될 수 있습니다. 이 크기를 Amazon Redshift에서 지오메트리 객체의 상대적 크기를 나타내는 표시로 사용할 수 있습니다.

구문

```
ST_MemSize(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

geom의 고유한 차원을 나타내는 INTEGER입니다.

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 지오메트리 컬렉션의 메모리 크기를 반환합니다.

```
SELECT ST_MemSize(ST_GeomFromText('GEOMETRYCOLLECTION(POLYGON((0 0,10 0,0 10,0
 0)),LINESTRING(20 10,20 0,10 0))')::varchar + ' bytes';
```

```
 ?column?
-----------
 172 bytes
```

ST_MMax

ST_MMax는 입력 지오메트리의 최대 m 좌표를 반환합니다.

구문

```
ST_MMax(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

최대 m 좌표의 DOUBLE PRECISION 값입니다.

geom이 비어 있으면 null이 반환됩니다.

geom이 null이면 null이 반환됩니다.

geom이 2D 또는 3DZ 지오메트리이면 null이 반환됩니다.

예시

다음 SQL은 3DM 지오메트리에서 라인스트링의 가장 큰 m 좌표를 반환합니다.

```
SELECT ST_MMax(ST_GeomFromEWKT('LINESTRING M (0 1 2, 3 4 5, 6 7 8)'));
```

```
st_mmax
-----------
  8
```

다음 SQL은 4D 지오메트리에서 라인스트링의 가장 큰 m 좌표를 반환합니다.

```
SELECT ST_MMax(ST_GeomFromEWKT('LINESTRING ZM (0 1 2 3, 4 5 6 7, 8 9 10 11)'));
```

```
st_mmax
-----------
  11
```

ST_MMin

ST_MMin은 입력 지오메트리의 최소 m 좌표를 반환합니다.

구문

```
ST_MMin(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

최소 m 좌표의 DOUBLE PRECISION 값입니다.

geom이 비어 있으면 null이 반환됩니다.

geom이 null이면 null이 반환됩니다.

geom이 2D 또는 3DZ 지오메트리이면 null이 반환됩니다.

예시

다음 SQL은 3DM 지오메트리에서 라인스트링의 가장 작은 m 좌표를 반환합니다.

```
SELECT ST_MMin(ST_GeomFromEWKT('LINESTRING M (0 1 2, 3 4 5, 6 7 8)'));
```

```
st_mmin
-----------
  2
```

다음 SQL은 4D 지오메트리에서 라인스트링의 가장 작은 m 좌표를 반환합니다.

```
SELECT ST_MMin(ST_GeomFromEWKT('LINESTRING ZM (0 1 2 3, 4 5 6 7, 8 9 10 11)'));
```

```
st_mmin
-----------
  3
```

ST_Multi

ST_Multi는 지오메트리를 해당 다중 유형으로 변환합니다. 입력 지오메트리가 이미 다중 유형 또는 지오메트리 컬렉션인 경우 해당 지오메트리의 복사본이 반환됩니다. 입력 지오메트리가 점, 라인스트링 또는 다각형이면 입력 지오메트리를 포함하는 다중 점, 다중 라인스트링 또는 다중 다각형이 반환됩니다.

구문

```
ST_Multi(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

하위 유형이 MULTIPOINT, MULTILINESTRING, MULTIPOLYGON 또는 GEOMETRYCOLLECTION인 GEOMETRY입니다.

반환된 지오메트리의 SRID(공간 참조 시스템 식별자)는 입력 지오메트리의 값과 동일합니다.

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 입력 다중 점에서 다중 점을 반환합니다.

```
SELECT ST_AsEWKT(ST_Multi(ST_GeomFromText('MULTIPOINT((1 2),(3 4))', 4326)));
```

```
    st_asewkt
-----------------------------------
  SRID=4326;MULTIPOINT((1 2),(3 4))
```

다음 SQL은 입력 점에서 다중 점을 반환합니다.

```
SELECT ST_AsEWKT(ST_Multi(ST_GeomFromText('POINT(1 2)', 4326)));
```

```
    st_asewkt
-----------------------------------
  SRID=4326;MULTIPOINT((1 2))
```

다음 SQL은 입력 지오메트리 컬렉션의 지오메트리 컬렉션을 반환합니다.

```
SELECT ST_AsEWKT(ST_Multi(ST_GeomFromText('GEOMETRYCOLLECTION(POINT(1 2),MULTIPOINT((1
 2),(3 4)))', 4326)));
```

```
    st_asewkt
-----------------------------------
  SRID=4326;GEOMETRYCOLLECTION(POINT(1 2),MULTIPOINT((1 2),(3 4)))
```

ST_NDims

ST_NDims는 지오메트리의 좌표 차원을 반환합니다. ST_NDims는 지오메트리의 토폴로지 차원을 고려하지 않습니다. 대신 지오메트리의 차원에 따라 상수 값을 반환합니다.

구문

```
ST_NDims(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

geom의 고유한 차원을 나타내는 INTEGER입니다.

geom이 null이면 null이 반환됩니다.

값은 다음과 같이 반환됩니다.

반환 값	입력 지오메트리의 차원
2	2D
3	3DZ 또는 3DM
4	4D

예시

다음 SQL은 2D 라인스트링의 차원 수를 반환합니다.

```
SELECT ST_NDims(ST_GeomFromText('LINESTRING(0 0,1 1,2 2,0 0)'));
```

```
st_ndims
-------------
 2
```

다음 SQL은 3DZ 라인스트링의 차원 수를 반환합니다.

```
SELECT ST_NDims(ST_GeomFromText('LINESTRING Z(0 0 3,1 1 3,2 2 3,0 0 3)'));
```

```
st_ndims
-------------
 3
```

다음 SQL은 3DM 라인스트링의 차원 수를 반환합니다.

```
SELECT ST_NDims(ST_GeomFromText('LINESTRING M(0 0 4,1 1 4,2 2 4,0 0 4)'));
```

```
st_ndims
-------------
 3
```

다음 SQL은 4D 라인스트링의 차원 수를 반환합니다.

```
SELECT ST_NDims(ST_GeomFromText('LINESTRING ZM(0 0 3 4,1 1 3 4,2 2 3 4,0 0 3 4)'));
```

```
st_ndims
-------------
 4
```

ST_NPoints

ST_NPoints는 입력 지오메트리 또는 지오그래피의 비어 있지 않은 점 수를 반환합니다.

구문

```
ST_NPoints(geo)
```

인수

geo

GEOMETRY 또는 GEOGRAPHY 데이터 유형의 값이나 GEOMETRY 또는 GEOGRAPHY 유형으로 계산되는 표현식입니다.

반환 타입

INTEGER

geo가 빈 점이면 0이 반환됩니다.

geo가 null이면 null이 반환됩니다.

예시

다음 SQL은 라인스트링의 점 수를 반환합니다.

```
SELECT ST_NPoints(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)'));
```

```
st_npoints
-------------
 4
```

다음 SQL은 지오그래피에서 라인스트링의 점 수를 반환합니다.

```
SELECT ST_NPoints(ST_GeogFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)'));
```

```
st_npoints
-------------
 4
```

ST_NRings

ST_NRings는 입력 지오메트리의 링 수를 반환합니다.

구문

```
ST_NRings(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

INTEGER

geom이 null이면 null이 반환됩니다.

값은 다음과 같이 반환됩니다.

반환 값	지오메트리 하위 유형
0	geom이 POINT, LINESTRING , MULTIPOINT 또는MULTILINESTRING 하위 유형인 경우 반환됩니다.

반환 값	지오메트리 하위 유형
링 수	geom이 POLYGON 또는 MULTIPOLYGON 하위 유형인 경우 반환됩니다.
모든 구성 요소의 링 수	geom이 GEOMETRYCOLLECTION 하위 유형인 경우 반환됩니다.

예시

다음 SQL은 다중 다각형의 링 수를 반환합니다.

```
SELECT ST_NRings(ST_GeomFromText('MULTIPOLYGON(((0 0,10 0,0 10,0 0)),((0 0,-10 0,0
 -10,0 0)))'));
```

```
 st_nrings
-------------
 2
```

ST_NumGeometries

ST_NumGeometries는 입력 지오메트리의 지오메트리 수를 반환합니다.

구문

```
ST_NumGeometries(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

geom의 지오메트리 수를 나타내는 INTEGER입니다.

geom이 null이면 null이 반환됩니다.

geom이 비어 있는 단일 지오메트리이면 0이 반환됩니다.

geom이 비어 있지 않은 단일 지오메트리이면 1이 반환됩니다.

geom이 GEOMETRYCOLLECTION 또는 MULTI 하위 유형인 경우 지오메트리 수가 반환됩니다.

예시

다음 SQL은 입력 다중 라인스트링의 지오메트리 수를 반환합니다.

```
SELECT ST_NumGeometries(ST_GeomFromText('MULTILINESTRING((0 0,1 0,0 5),(3 4,13 26))'));
```

```
st_numgeometries
-------------
 2
```

ST_NumInteriorRings

ST_NumInteriorRings는 입력 다각형 지오메트리의 링 수를 반환합니다.

구문

```
ST_NumInteriorRings(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

INTEGER

geom이 null이면 null이 반환됩니다.

geom이 다각형이 아니면 null이 반환됩니다.

예시

다음 SQL은 입력 다각형의 내부 링 수를 반환합니다.

```
SELECT ST_NumInteriorRings(ST_GeomFromText('POLYGON((0 0,100 0,100 100,0 100,0 0),(1
 1,1 5,5 1,1 1),(7 7,7 8,8 7,7 7))'));
```

```
 st_numinteriorrings
-------------
 2
```

ST_NumPoints

ST_NumPoints는 입력 지오메트리의 점 수를 반환합니다.

구문

```
ST_NumPoints(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

INTEGER

geom이 null이면 null이 반환됩니다.

geom이 하위 유형 LINESTRING이 아니면 null이 반환됩니다.

예시

다음 SQL은 입력 라인스트링의 점 수를 반환합니다.

```
SELECT ST_NumPoints(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27
 29.31,77.29 29.07)'));
```

```
st_numpoints
-------------
```

```
4
```

입력 geom이 하위 유형 LINESTRING이 아니기 때문에 다음 SQL이 null을 반환합니다.

```
SELECT ST_NumPoints(ST_GeomFromText('MULTIPOINT(1 2,3 4)'));
```

```
st_numpoints
-------------
```

ST_Perimeter

입력 영역 지오메트리의 경우 ST_Perimeter는 2D 프로젝션의 데카르트 둘레(경계의 길이)를 반환합니다. 둘레 단위는 입력 지오메트리의 좌표가 표현되는 단위와 동일합니다. 이 함수는 점, 다중 점 및 선형 지오메트리에 대해 0을 반환합니다. 입력이 지오메트리 컬렉션인 경우 이 함수는 컬렉션에 있는 지오메트리 둘레의 합계를 반환합니다.

입력 지오그래피의 경우 ST_Perimeter는 SRID에 의해 결정된 회전 타원체에서 계산된 입력 영역 지오그래피의 2D 투영의 측지 둘레(경계 길이)를 반환합니다. 둘레 단위는 미터입니다. 이 함수는 점, 다중 점 및 선형 지오그래피에 대해 0을 반환합니다. 입력이 지오메트리 컬렉션인 경우 이 함수는 컬렉션에 있는 지오그래피 둘레의 합계를 반환합니다.

구문

```
ST_Perimeter(geo)
```

인수

geo

GEOMETRY 또는 GEOGRAPHY 데이터 유형의 값이나 GEOMETRY 또는 GEOGRAPHY 유형으로 계산되는 표현식입니다.

반환 타입

DOUBLE PRECISION

geo가 null이면 null이 반환됩니다.

SRID 값을 찾을 수 없으면 오류가 반환됩니다.

예시

다음 SQL은 다중 다각형의 데카르트 둘레를 반환합니다.

```
SELECT ST_Perimeter(ST_GeomFromText('MULTIPOLYGON(((0 0,10 0,0 10,0 0)),((10 0,20 0,20
 10,10 0)))'));
```

```
 st_perimeter
-------------------------------
    68.2842712474619
```

다음 SQL은 다중 다각형의 데카르트 둘레를 반환합니다.

```
SELECT ST_Perimeter(ST_GeomFromText('MULTIPOLYGON(((0 0,10 0,0 10,0 0)),((10 0,20 0,20
 10,10 0)))'));
```

```
 st_perimeter
-------------------------------
    68.2842712474619
```

다음 SQL은 지오그래피에서 다각형의 둘레를 반환합니다.

```
SELECT ST_Perimeter(ST_GeogFromText('SRID=4326;POLYGON((0 0,1 0,0 1,0 0))'));
```

```
 st_perimeter
------------------
 378790.428393693
```

다음 SQL은 지오그래피에서 라인스트링의 둘레를 반환합니다.

```
SELECT ST_Perimeter(ST_GeogFromText('SRID=4326;LINESTRING(5 0,10 0)'));
```

```
 st_perimeter
```

```
--------------
 0
```

ST_Perimeter2D

ST_Perimeter2D는 ST_Perimeter의 별칭입니다. 자세한 내용은 ST_Perimeter 단원을 참조하십시오.

ST_Point

ST_Point는 입력 좌표 값에서 점 지오메트리를 반환합니다.

구문

```
ST_Point(x, y)
```

인수

x

첫 번째 좌표를 나타내는 DOUBLE PRECISION 데이터 형식의 값입니다.

y

두 번째 좌표를 나타내는 DOUBLE PRECISION 데이터 형식의 값입니다.

반환 타입

POINT 하위 유형의 GEOMETRY입니다.

반환된 지오메트리의 SRID(공간 참조 시스템 식별자) 값은 0으로 설정됩니다.

x 또는 y가 null이면 null이 반환됩니다.

예시

다음 SQL은 입력 좌표에서 점 지오메트리를 구성합니다.

```
SELECT ST_AsText(ST_Point(5.0, 7.0));
```

```
st_astext
-------------
```

```
POINT(5 7)
```

ST_PointN

ST_PointN은 인덱스 값으로 지정된 라인스트링의 점을 반환합니다. 음수 인덱스 값은 라인스트링 끝에서 역방향으로 계산되므로 -1이 마지막 점입니다.

반환된 지오메트리의 차원은 입력 지오메트리의 차원과 같습니다.

구문

```
ST_PointN(geom, index)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 LINESTRING이어야 합니다.

인덱스를 구축하고 배포할 것입니다

라인스트링에 있는 점의 인덱스를 나타내는 INTEGER 데이터 형식의 값입니다.

반환 타입

POINT 하위 유형의 GEOMETRY입니다.

반환된 지오메트리의 SRID(공간 참조 시스템 식별자) 값은 0으로 설정됩니다.

geom 또는 index가 null이면 null이 반환됩니다.

index가 범위를 벗어난 경우 null이 반환됩니다.

geom이 비어 있으면 null이 반환됩니다.

geom이 LINESTRING이 아니면 null이 반환됩니다.

예시

다음 SQL은 GEOMETRY 객체에 대한 6개의 점 LINESTRING의 EWKT(Extended Well-Known Text) 표현을 반환하고 라인스트링의 인덱스 5에 있는 점을 반환합니다.

```
SELECT ST_AsEWKT(ST_PointN(ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0 5,0
 0)',4326), 5));
```

```
st_asewkt
-------------
 SRID=4326;POINT(0 5)
```

ST_Points

ST_Points는 입력 지오메트리의 비어 있지 않은 모든 점을 포함하는 다중 점 지오메트리를 반환합니다. ST_Points는 링 지오메트리의 시작 점과 끝 점을 포함하여 입력에 중복된 점을 제거하지 않습니다.

구문

```
ST_Points(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

MULTIPOINT 하위 유형의 GEOMETRY입니다.

반환된 지오메트리의 공간 참조 시스템 식별자(SRID) 값은 geom과 같습니다.

geom이 null이면 null이 반환됩니다.

geom이 비어 있으면 빈 다중 점이 반환됩니다.

예시

다음 SQL 예는 입력 지오메트리에서 다중 점 지오메트리를 구성합니다. 결과는 입력 지오메트리의 비어 있지 않은 점을 포함하는 다중 점 지오메트리입니다.

```
SELECT ST_AsEWKT(ST_Points(ST_SetSRID(ST_GeomFromText('LINESTRING(1 0,2 0,3 0)'),
 4326)));
```

```
st_asewkt
-------------
SRID=4326;MULTIPOINT((1 0),(2 0),(3 0))
```

```
SELECT ST_AsEWKT(ST_Points(ST_SetSRID(ST_GeomFromText('MULTIPOLYGON(((0 0,1 0,0 1,0
 0)))'), 4326)));
```

```
st_asewkt
-------------
SRID=4326;MULTIPOINT((0 0),(1 0),(0 1),(0 0))
```

ST_Polygon

ST_Polygon은 외부 링이 SRID(공간 참조 시스템 식별자)에 대해 입력된 값을 갖는 입력 라인스트링인 다각형 지오메트리를 반환합니다.

반환된 지오메트리의 차원은 입력 지오메트리의 차원과 같습니다.

구문

```
ST_Polygon(linestring, srid)
```

인수

linestring

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 라인스트링을 나타내는 LINESTRING이어야 합니다. linestring 값은 닫혀 있어야 합니다.

srid

SRID를 나타내는 INTEGER 데이터 형식의 값입니다.

반환 타입

POLYGON 하위 유형의 GEOMETRY입니다.

반환된 지오메트리의 SRID 값은 srid로 설정됩니다.

linestring 또는 srid가 null이면 null이 반환됩니다.

linestring이 라인스트링이 아니면 오류가 반환됩니다.

linestring이 닫혀 있지 않으면 오류가 반환됩니다.

srid가 음수이면 오류가 반환됩니다.

예시

다음 SQL은 SRID 값으로 다각형을 구성합니다.

```
SELECT ST_AsEWKT(ST_Polygon(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27
 29.31,77.29 29.07)'),4356));
```

```
st_asewkt
-------------
 SRID=4356;POLYGON((77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07))
```

ST_RemovePoint

ST_RemovePoint는 인덱스 위치에서 입력 지오메트리의 점이 제거된 라인스트링 지오메트리를 반환합니다.

인덱스는 0부터 시작합니다. 결과의 SRID(공간 참조 시스템 식별자)는 입력 지오메트리와 동일합니다. 반환된 지오메트리의 차원은 입력 지오메트리의 차원과 같습니다.

구문

```
ST_RemovePoint(geom, index)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 LINESTRING이어야 합니다.

인덱스를 구축하고 배포할 것입니다

0부터 시작하는 인덱스의 위치를 나타내는 INTEGER 데이터 형식의 값입니다.

반환 타입

GEOMETRY

geom 또는 index가 null이면 null이 반환됩니다.

geom이 하위 유형 LINESTRING이 아니면 오류가 반환됩니다.

index가 범위를 벗어나면 오류가 반환됩니다. 인덱스 위치에 대한 유효한 값은 0과 ST_NumPoints(geom) - 1 사이의 값입니다.

예시

다음 SQL은 라인스트링의 마지막 점을 제거합니다.

```
WITH tmp(g) AS (SELECT ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0 5)',4326))
SELECT ST_AsEWKT(ST_RemovePoint(g, ST_NumPoints(g) - 1)) FROM tmp;
```

```
   st_asewkt
----------------------------------------
 SRID=4326;LINESTRING(0 0,10 0,10 10,5 5)
```

ST_Reverse

ST_Reverse는 선형 및 영역 지오메트리의 꼭짓점 순서를 반대로 합니다. 점 또는 다중 점 지오메트리의 경우 원래 지오메트리의 복사본이 반환됩니다. 지오메트리 컬렉션의 경우 ST_Reverse는 컬렉션의 각 지오메트리 꼭짓점 순서를 반대로 합니다.

반환된 지오메트리의 차원은 입력 지오메트리의 차원과 같습니다.

구문

```
ST_Reverse(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

GEOMETRY

반환된 지오메트리의 SRID(공간 참조 시스템 식별자)는 입력 지오메트리의 값과 동일합니다.

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 라인스트링에서 점의 순서를 반대로 합니다.

```
SELECT ST_AsEWKT(ST_Reverse(ST_GeomFromText('LINESTRING(1 0,2 0,3 0,4 0)', 4326)));
```

```
    st_asewkt
-----------------------------------
  SRID=4326;LINESTRING(4 0,3 0,2 0,1 0)
```

ST_SetPoint

ST_SetPoint는 인덱스에 의해 지정된 입력 라인스트링의 위치와 관련하여 업데이트된 좌표가 있는 라인스트링을 반환합니다. 새 좌표는 입력 점의 좌표입니다.

반환된 지오메트리의 차원은 geom1 값의 차원과 같습니다. geom1과 geom2의 차원이 다른 경우 geom2는 geom1의 차원에 나타납니다.

구문

```
ST_SetPoint(geom1, index, geom2)
```

인수

geom1

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 LINESTRING이어야 합니다.

인덱스를 구축하고 배포할 것입니다

인덱스의 위치를 나타내는 INTEGER 데이터 형식의 값입니다. 0은 라인스트링의 왼쪽에서 첫 번째 점을 가리키고, 1은 두 번째 점을 가리킵니다. 인덱스는 음수 값일 수 있습니다. -1은 라인스트링의 오른쪽에서 첫 번째 점을 가리키고, -2는 라인스트링의 오른쪽에서 두 번째 점을 가리킵니다.

geom2

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 POINT이어야 합니다.

반환 타입

GEOMETRY

geom2가 빈 점이면 geom1이 반환됩니다.

geom1, geom2 또는 index가 null이면 null이 반환됩니다.

geom1이 라인스트링이 아니면 오류가 반환됩니다.

index가 유효한 인덱스 범위 내에 있지 않으면 오류가 반환됩니다.

geom2가 점이 아니면 오류가 반환됩니다.

geom1과 geom2의 SRID(공간 참조 시스템 식별자) 값이 같지 않으면 오류가 반환됩니다.

예시

다음 SQL은 입력 라인스트링의 두 번째 점을 지정된 점으로 설정한 새로운 라인스트링을 반환합니다.

```
SELECT ST_AsText(ST_SetPoint(ST_GeomFromText('LINESTRING(1 2, 3 2, 5 2, 1 2)'), 2,
 ST_GeomFromText('POINT(7 9)')));
```

```
st_astext
-------------
 LINESTRING(1 2,3 2,7 9,1 2)
```

다음 SQL 예는 라인스트링의 오른쪽에서 세 번째 점(인덱스가 음수임)를 지정된 점으로 설정한 새로운 라인스트링을 반환합니다.

```
SELECT ST_AsText(ST_SetPoint(ST_GeomFromText('LINESTRING(1 2, 3 2, 5 2, 1 2)'), -3,
 ST_GeomFromText('POINT(7 9)')));
```

```
st_astext
-------------
 LINESTRING(1 2,7 9,5 2,1 2)
```

ST_SetSRID

ST_SetSRID는 SRID(공간 참조 시스템 식별자)에 대한 입력 값으로 업데이트된 것을 제외하고 입력 지오메트리와 동일한 지오메트리를 반환합니다.

구문

```
ST_SetSRID(geom, srid)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

srid

SRID를 나타내는 INTEGER 데이터 형식의 값입니다.

반환 타입

GEOMETRY

반환된 지오메트리의 SRID 값은 srid로 설정됩니다.

geom 또는 srid가 null이면 null이 반환됩니다.

srid가 음수이면 오류가 반환됩니다.

예시

다음 SQL은 라인스트링의 SRID 값을 설정합니다.

```
SELECT ST_AsEWKT(ST_SetSRID(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27
 29.31,77.29 29.07)'),50));
```

```
st_asewkt
-------------
 SRID=50;LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)
```

ST_Simplify

ST_Simplify는 주어진 허용치로 Ramer-Douglas-Peucker 알고리즘을 사용하여 입력 지오메트리의 단순화된 복사본을 반환합니다. 입력 지오메트리의 토폴로지가 유지되지 않을 수 있습니다. 알고리즘에 대한 자세한 내용은 Wikipedia의 Ramer–Douglas–Peucker algorithm을 참조하세요.

ST_Simplify가 지오메트리를 단순화하기 위해 거리를 계산할 때 ST_Simplify는 입력 지오메트리의 2D 프로젝션에서 작동합니다.

구문

```
ST_Simplify(geom, tolerance)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

tolerance

Ramer-Douglas-Peucker 알고리즘의 허용치 수준을 나타내는 DOUBLE PRECISION 데이터 형식의 값입니다. tolerance가 음수이면 0이 사용됩니다.

반환 타입

GEOMETRY.

반환된 지오메트리의 공간 참조 시스템 식별자(SRID) 값은 입력 지오메트리의 SRID 값입니다.

반환된 지오메트리의 차원은 입력 지오메트리의 차원과 같습니다.

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 Ramer-Douglas-Peucker 알고리즘과 함께 유클리드 거리 허용치 1을 사용하여 입력 라인 스트링을 단순화합니다. 거리의 단위는 지오메트리 좌표의 단위와 같습니다.

```
SELECT ST_AsEWKT(ST_Simplify(ST_GeomFromText('LINESTRING(0 0,1 2,1 1,2 2,2 1)'), 1));
```

```
 st_asewkt
-----------
LINESTRING(0 0,1 2,2 1)
```

ST_SRID

ST_SRID는 입력 지오메트리의 SRID(공간 참조 시스템 식별자)를 반환합니다. SRID에 대한 자세한 내용은 Amazon Redshift에서 공간 데이터 쿼리 단원을 참조하세요.

구문

```
ST_SRID(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

geom의 SRID 값을 나타내는 INTEGER입니다.

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 SRID 4326으로 설정된 라인 문자열의 SRID 값을 반환합니다.

```
SELECT ST_SRID(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)',4326));
```

```
st_srid
-------------
```

```
 4326
```

다음 SQL은 생성 시 설정되지 않은 라인 문자열의 SRID 값을 반환합니다. 그 결과 SRID 값은 0이 됩니다.

```
SELECT ST_SRID(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)'));
```

```
st_srid
-------------
 0
```

ST_StartPoint

ST_StartPoint는 입력 라인스트링의 첫 번째 점을 반환합니다. 결과의 SRID(공간 참조 시스템 식별자) 값은 입력 지오메트리의 값과 동일합니다. 반환된 지오메트리의 차원은 입력 지오메트리의 차원과 같습니다.

구문

```
ST_StartPoint(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 하위 유형은 LINESTRING이어야 합니다.

반환 타입

GEOMETRY

geom이 null이면 null이 반환됩니다.

geom이 비어 있으면 null이 반환됩니다.

geom이 LINESTRING이 아니면 null이 반환됩니다.

예시

다음 SQL은 GEOMETRY 객체에 대한 4개의 점 LINESTRING의 EWKT(Extended Well-Known Text) 표현을 반환하고 라인스트링의 시작 지점을 반환합니다.

```
SELECT ST_AsEWKT(ST_StartPoint(ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0
 5)',4326)));
```

```
st_asewkt
-------------
 SRID=4326;POINT(0 0)
```

ST_Touches

ST_Touches는 두 입력 지오메트리의 2D 프로젝션이 접촉하면 true를 반환합니다. 두 지오메트리가 비어 있지 않고 교차하며 공통된 내부 점이 없는 경우 두 지오메트리가 접촉합니다.

구문

```
ST_Touches(geom1, geom2)
```

인수

geom1

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

geom2

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

BOOLEAN

geom1 또는 geom2가 null이면 null이 반환됩니다.

geom1과 geom2의 SRID(공간 참조 시스템 식별자) 값이 같지 않으면 오류가 반환됩니다.

geom1 또는 geom2가 지오메트리 컬렉션인 경우 오류가 반환됩니다.

예시

다음 SQL은 다각형이 라인스트링과 접촉하는지 확인합니다.

```
SELECT ST_Touches(ST_GeomFromText('POLYGON((0 0,10 0,0 10,0 0))'),
 ST_GeomFromText('LINESTRING(20 10,20 0,10 0)'));
```

```
 st_touches
-------------
 t
```

ST_Transform

ST_Transform은 입력 공간 참조 시스템 식별자(SRID)에 의해 정의된 공간 참조 시스템에서 변환된 좌표가 있는 새 지오메트리를 반환합니다.

구문

```
ST_Transform(geom, srid)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

srid

SRID를 나타내는 INTEGER 데이터 유형의 값입니다.

반환 타입

GEOMETRY.

반환된 지오메트리의 SRID 값은 srid로 설정됩니다.

geom 또는 srid가 null이면 null이 반환됩니다.

입력 geom과 연결된 SRID 값이 없으면 오류가 반환됩니다.

srid가 없으면 오류가 반환됩니다.

예시

다음 SQL은 빈 지오메트리 컬렉션의 SRID를 변환합니다.

```
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromText('GEOMETRYCOLLECTION EMPTY', 3857),
 4326));
```

```
             st_asewkt
-----------------------------------
 SRID=4326;GEOMETRYCOLLECTION EMPTY
```

다음 SQL은 라인스트링의 SRID를 변환합니다.

```
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromText('LINESTRING(110 40, 2 3, -10 80, -7 9,
 -22 -33)', 4326), 26918));
```

```
     st_asewkt
-----------------------------------------------------------------------------------------------
 SRID=26918;LINESTRING(73106.6977300955 15556182.9688576,14347201.5059964
 1545178.32934967,1515090.41262989 9522193.25115316,10491250.83295
 2575457.28410878,5672303.72135968 -5233682.61176205)
```

다음 SQL은 다각형의 SRID를 변환합니다.

```
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromText('POLYGON Z ((-10 10 -7, -65 10 -6, -10 64
 -5, -10 10 -7), (-11 11 5, -11 12 6, -12 11 7, -11 11 5))', 6989), 6317));
```

```
                                        st_asewkt
-----------------------------------------------------------------------------------------------
 SRID=6317;POLYGON Z ((6186430.2771091 -1090834.57212608
 1100247.33216237,2654831.67853801 -5693304.90741276 1100247.50581055,2760987.41750022
 -486836.575101877 5709710.44137268,6186430.2771091 -1090834.57212608
```

```
1100247.33216237),(6146675.25029258 -1194792.63532103 1209007.1115113,6125027.87562215
-1190584.81194058 1317403.77865723,6124888.99555252 -1301885.3455052
1209007.49312929,6146675.25029258 -1194792.63532103 1209007.1115113))
```

ST_Union

ST_Union은 두 지오메트리의 유니언을 나타내는 지오메트리를 반환합니다. 즉, 입력 지오메트리를 병합하여 겹치지 않는 결과 형상을 생성합니다.

구문

```
ST_Union(geom1, geom2)
```

인수

geom1

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

geom2

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

GEOMETRY

반환된 지오메트리의 SRID(공간 참조 시스템 식별자) 값은 입력 지오메트리의 SRID 값입니다.

geom1 또는 geom2가 null이면 null이 반환됩니다.

geom1 또는 geom2가 비어 있으면 빈 지오메트리가 반환됩니다.

geom1과 geom2의 SRID(공간 참조 시스템 식별자) 값이 같지 않으면 오류가 반환됩니다.

geom1 또는 geom2가 지오메트리 컬렉션, 라인스트링 또는 다중 라인스트링인 경우 오류가 반환됩니다.

geom1 또는 geom2가 2차원(2D) 지오메트리가 아니면 오류가 반환됩니다.

예시

다음 SQL은 두 입력 지오메트리의 유니언을 나타내는 비어 있지 않은 지오메트리를 반환합니다.

```
SELECT ST_AsEWKT(ST_Union(ST_GeomFromText('POLYGON((0 0,100 100,0 200,0 0))'),
 ST_GeomFromText('POLYGON((0 0,10 0,0 10,0 0))')));
```

```
        st_asewkt
-------------------------
 POLYGON((0 0,0 200,100 100,5 5,10 0,0 0))
```

ST_Within

ST_Within은 첫 번째 입력 지오메트리의 2D 프로젝션이 두 번째 입력 지오메트리의 2D 프로젝션 내에 있는 경우 true를 반환합니다.

예를 들어 지오메트리 A의 모든 점이 지오메트리 B의 점이고 그 내부에 비어 있지 않은 교차점이 있는 경우, A는 B 내부에 있습니다.

ST_Within(A, B)는 ST_Contains(B, A)와 동등합니다.

구문

```
ST_Within(geom1, geom2)
```

인수

geom1

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다. 이 값을 geom2와 비교하여 geom2 내부에 있는지 판별합니다.

geom2

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

BOOLEAN

geom1 또는 geom2가 null이면 null이 반환됩니다.

geom1과 geom2의 SRID(공간 참조 시스템 식별자) 값이 동일하지 않으면 오류가 반환됩니다.

geom1 또는 geom2가 지오메트리 컬렉션인 경우 오류가 반환됩니다.

예시

다음 SQL은 첫 번째 다각형이 두 번째 다각형 내부에 있는지 확인합니다.

```
SELECT ST_Within(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
```

```
st_within
-----------
 true
```

ST_X

ST_X는 입력 점의 첫 번째 좌표를 반환합니다.

구문

```
ST_X(point)
```

인수

point

GEOMETRY 데이터 형식의 POINT 값입니다.

반환 타입

첫 번째 좌표의 DOUBLE PRECISION 값입니다.

point가 null이면 null이 반환됩니다.

point가 빈 점이면 null이 반환됩니다.

point가 POINT가 아니면 오류가 반환됩니다.

예시

다음 SQL은 점의 첫 번째 좌표를 반환합니다.

```
SELECT ST_X(ST_Point(1,2));
```

```
st_x
-----------
 1.0
```

ST_XMax

ST_XMax는 입력 지오메트리의 최대 첫 번째 좌표를 반환합니다.

구문

```
ST_XMax(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

최대 첫 번째 좌표의 DOUBLE PRECISION 값입니다.

geom이 비어 있으면 null이 반환됩니다.

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 라인스트링의 가장 큰 첫 번째 좌표를 반환합니다.

```
SELECT ST_XMax(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)'));
```

```
st_xmax
-----------
 77.42
```

ST_XMin

ST_XMin은 입력 지오메트리의 최소 첫 번째 좌표를 반환합니다.

구문

```
ST_XMin(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

최소 첫 번째 좌표의 DOUBLE PRECISION 값입니다.

geom이 비어 있으면 null이 반환됩니다.

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 라인스트링의 가장 작은 첫 번째 좌표를 반환합니다.

```
SELECT ST_XMin(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)'));
```

```
st_xmin
-----------
 77.27
```

ST_Y

ST_Y는 입력 점의 두 번째 좌표를 반환합니다.

구문

```
ST_Y(point)
```

인수

point

GEOMETRY 데이터 형식의 POINT 값입니다.

반환 타입

두 번째 좌표의 DOUBLE PRECISION 값입니다.

point가 null이면 null이 반환됩니다.

point가 빈 점이면 null이 반환됩니다.

point가 POINT가 아니면 오류가 반환됩니다.

예시

다음 SQL은 점의 두 번째 좌표를 반환합니다.

```
SELECT ST_Y(ST_Point(1,2));
```

```
st_y
-----------
 2.0
```

ST_YMax

ST_YMax는 입력 지오메트리의 최대 두 번째 좌표를 반환합니다.

구문

```
ST_YMax(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

최대 두 번째 좌표의 DOUBLE PRECISION 값입니다.

geom이 비어 있으면 null이 반환됩니다.

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 라인스트링의 가장 큰 두 번째 좌표를 반환합니다.

```
SELECT ST_YMax(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)'));
```

```
st_ymax
-----------
 29.31
```

ST_YMin

ST_YMin은 입력 지오메트리의 최소 두 번째 좌표를 반환합니다.

구문

```
ST_YMin(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

최소 두 번째 좌표의 DOUBLE PRECISION 값입니다.

geom이 비어 있으면 null이 반환됩니다.

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 라인스트링의 가장 작은 두 번째 좌표를 반환합니다.

```
SELECT ST_YMin(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)'));
```

```
st_ymin
-----------
 29.07
```

ST_Z

ST_Z은 입력 점의 z 좌표를 반환합니다.

구문

```
ST_Z(point)
```

인수

point

GEOMETRY 데이터 형식의 POINT 값입니다.

반환 타입

m 좌표의 DOUBLE PRECISION 값입니다.

point가 null이면 null이 반환됩니다.

point가 2D 또는 3DM 점이면 null이 반환됩니다.

point가 빈 점이면 null이 반환됩니다.

point가 POINT가 아니면 오류가 반환됩니다.

예시

다음 SQL은 3DZ 지오메트리에서 점의 z 좌표를 반환합니다.

```
SELECT ST_Z(ST_GeomFromEWKT('POINT Z (1 2 3)'));
```

```
st_z
-----------
 3
```

다음 SQL은 4D 지오메트리에서 점의 z 좌표를 반환합니다.

```
SELECT ST_Z(ST_GeomFromEWKT('POINT ZM (1 2 3 4)'));
```

```
st_z
-----------
 3
```

ST_ZMax

ST_ZMax는 입력 지오메트리의 최대 z 좌표를 반환합니다.

구문

```
ST_ZMax(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

최대 z 좌표의 DOUBLE PRECISION 값입니다.

geom이 비어 있으면 null이 반환됩니다.

geom이 null이면 null이 반환됩니다.

geom이 2D 또는 3DM 지오메트리이면 null이 반환됩니다.

예시

다음 SQL은 3DZ 지오메트리에서 라인스트링의 가장 큰 z 좌표를 반환합니다.

```
SELECT ST_ZMax(ST_GeomFromEWKT('LINESTRING Z (0 1 2, 3 4 5, 6 7 8)'));
```

```
st_zmax
-----------
  8
```

다음 SQL은 4D 지오메트리에서 라인스트링의 가장 큰 z 좌표를 반환합니다.

```
SELECT ST_ZMax(ST_GeomFromEWKT('LINESTRING ZM (0 1 2 3, 4 5 6 7, 8 9 10 11)'));
```

```
st_zmax
-----------
  10
```

ST_ZMin

ST_ZMin은 입력 지오메트리의 최소 z 좌표를 반환합니다.

구문

```
ST_ZMin(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

최소 z 좌표의 DOUBLE PRECISION 값입니다.

geom이 비어 있으면 null이 반환됩니다.

geom이 null이면 null이 반환됩니다.

geom이 2D 또는 3DM 지오메트리이면 null이 반환됩니다.

예시

다음 SQL은 3DZ 지오메트리에서 라인스트링의 가장 작은 z 좌표를 반환합니다.

```
SELECT ST_ZMin(ST_GeomFromEWKT('LINESTRING Z (0 1 2, 3 4 5, 6 7 8)'));
```

```
st_zmin
-----------
  2
```

다음 SQL은 4D 지오메트리에서 라인스트링의 가장 작은 z 좌표를 반환합니다.

```
SELECT ST_ZMin(ST_GeomFromEWKT('LINESTRING ZM (0 1 2 3, 4 5 6 7, 8 9 10 11)'));
```

```
st_zmin
-----------
  2
```

SupportsBBox

SupportsBBox는 입력 지오메트리가 미리 계산된 경계 상자로 인코딩을 지원하면 true를 반환합니다. 경계 상자 지원에 대한 자세한 내용은 경계 상자 섹션을 참조하세요.

구문

```
SupportsBBox(geom)
```

인수

geom

GEOMETRY 데이터 형식의 값 또는 GEOMETRY 형식으로 계산되는 표현식입니다.

반환 타입

BOOLEAN

geom이 null이면 null이 반환됩니다.

예시

다음 SQL은 입력 점 지오메트리가 경계 상자로 인코딩을 지원하기 때문에 true를 반환합니다.

```
SELECT SupportsBBox(AddBBox(ST_GeomFromText('POLYGON((0 0,1 0,0 1,0 0))')));
```

```
supportsbbox
--------------
t
```

다음 SQL은 입력 점 지오메트리가 경계 상자로 인코딩을 지원하지 않기 때문에 false를 반환합니다.

```
SELECT SupportsBBox(DropBBox(ST_GeomFromText('POLYGON((0 0,1 0,0 1,0 0))')));
```

```
supportsbbox
--------------
f
```

문자열 함수

주제

- ||(연결) 연산자
- ASCII 함수
- BPCHARCMP 함수
- BTRIM 함수
- BTTEXT_PATTERN_CMP 함수
- CHAR_LENGTH 함수
- CHARACTER_LENGTH 함수
- CHARINDEX 함수
- CHR 함수
- COLLATE 함수

- CONCAT 함수
- CRC32 함수
- DIFFERENCE 함수
- INITCAP 함수
- LEFT 및 RIGHT 함수
- LEN 함수
- LENGTH 함수
- LOWER 함수
- LPAD 및 RPAD 함수
- ltrim 함수
- OCTETINDEX 함수
- OCTET_LENGTH 함수
- POSITION 함수
- QUOTE_IDENT 함수
- QUOTE_LITERAL 함수
- REGEXP_COUNT 함수
- REGEXP_INSTR 함수
- REGEXP_REPLACE 함수
- REGEXP_SUBSTR 함수
- REPEAT 함수
- REPLACE 함수
- REPLICATE 함수
- REVERSE 함수
- RTRIM 함수
- SOUNDEX 함수
- SPLIT_PART 함수
- STRPOS 함수
- STRTOL 함수
- SUBSTRING 함수

- TEXTLEN 함수
- TRANSLATE 함수
- TRIM 함수
- UPPER 함수

문자열 함수는 문자열을, 혹은 문자열로 평가되는 표현식을 처리 및 조작합니다. 이 함수에서 string 인수가 리터럴 값일 때는 작은따옴표로 묶어야 합니다. 지원되는 데이터 형식은 CHAR와 VARCHAR입니다.

다음 단원에서는 함수 이름과 구문, 그리고 지원되는 함수에 대한 설명에 대해서 살펴보겠습니다. 문자열에 대한 오프셋은 모두 1부터 시작됩니다.

지원되지 않는 리더 노드 전용 함수

다음 문자열 함수는 리더 노드에서만 실행되기 때문에 여기에서 다루지 않습니다. 자세한 내용은 리더 노드 전용 함수 섹션을 참조하세요.

- GET_BYTE
- SET_BIT
- SET_BYTE
- TO_ASCII

||(연결) 연산자

|| 기호의 양쪽으로 두 표현식을 연결하여 연결된 표현식을 반환합니다.

CONCAT 함수와 유사합니다.

> **Note**
>
> 표현식 중 하나 또는 둘 모두가 null인 경우 연결 결과는 NULL입니다.

구문

```
expression1 || expression2
```

인수

expression1

CHAR 문자열, VARCHAR 문자열, 이진 표현식 또는 이러한 형식 중 하나로 평가되는 표현식입니다.

expression2

CHAR 문자열, VARCHAR 문자열, 이진 표현식 또는 이러한 형식 중 하나로 평가되는 표현식입니다.

반환 타입

문자열의 반환 형식은 입력 인수의 형식과 동일합니다. 두 개의 VARCHAR 형식의 문자열을 연결하면 VARCHAR 형식의 문자열이 반환됩니다.

예시

다음 예제에서는 TICKIT 샘플 데이터베이스의 USERS 및 VENUE 테이블을 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

샘플 데이터베이스의 USERS 테이블에서 FIRSTNAME 및 LASTNAME 필드를 연결하려면 다음 예제를 사용합니다.

```
SELECT (firstname || ' ' || lastname) as fullname
FROM users
ORDER BY 1
LIMIT 10;

+-----------------+
|    fullname     |
+-----------------+
| Aaron Banks     |
| Aaron Booth     |
| Aaron Browning  |
| Aaron Burnett   |
| Aaron Casey     |
| Aaron Cash      |
| Aaron Castro    |
| Aaron Dickerson |
| Aaron Dixon     |
```

```
| Aaron Dotson    |
+-----------------+
```

NULL 값이 포함되었을 수도 있는 열을 연결하려면 NVL 및 COALESCE 함수 표현식을 사용해야 합니다. 다음 예제에서는 NVL을 사용하여 NULL이 발생할 때마다 0을 반환합니다.

```
SELECT (venuename || ' seats ' || NVL(venueseats, 0)) as seating
FROM venue
WHERE venuestate = 'NV' or venuestate = 'NC'
ORDER BY 1
LIMIT 10;

+-------------------------------------+
|               seating               |
+-------------------------------------+
| Ballys Hotel seats 0                |
| Bank of America Stadium seats 73298 |
| Bellagio Hotel seats 0              |
| Caesars Palace seats 0              |
| Harrahs Hotel seats 0               |
| Hilton Hotel seats 0                |
| Luxor Hotel seats 0                 |
| Mandalay Bay Hotel seats 0          |
| Mirage Hotel seats 0                |
| New York New York seats 0           |
+-------------------------------------+
```

ASCII 함수

ASCII 함수는 지정한 문자열에서 첫 번째 문자의 ASCII 코드나 유니코드 코드 포인트를 반환합니다. 이 함수는 문자열이 비어 있으면 0을 반환합니다. 문자열이 null이면 NULL을 반환합니다.

구문

```
ASCII('string')
```

인수

string

CHAR 문자열 또는 VARCHAR 문자열입니다.

반환 타입

INTEGER

예시

NULL을 반환하려면 다음 예제를 사용합니다. NULLIF 함수는 두 인수가 동일한 경우 NULL을 반환하므로 ASCII 함수의 입력 인수는 NULL입니다. 자세한 내용은 NULLIF 함수 단원을 참조하십시오.

```
SELECT ASCII(NULLIF('',''));

+-------+
| ascii |
+-------+
|  NULL |
+-------+
```

ASCII 코드 0을 반환하려면 다음 예제를 사용합니다.

```
SELECT ASCII('');

+-------+
| ascii |
+-------+
|     0 |
+-------+
```

amazon이라는 단어의 첫 글자에 대한 ASCII 코드 97을 반환하려면 다음 예제를 사용합니다.

```
SELECT ASCII('amazon');

+-------+
| ascii |
+-------+
|    97 |
+-------+
```

amazon이라는 단어의 첫 글자에 대한 ASCII 코드 65를 반환하려면 다음 예제를 사용합니다.

```
SELECT ASCII('Amazon');
```

```
+-------+
| ascii |
+-------+
|    65 |
+-------+
```

BPCHARCMP 함수

두 문자열의 값을 비교한 후 정수를 반환합니다. 문자열이 서로 동일한 경우에는 함수가 0을 반환합니다. 첫 번째 문자열이 알파벳 순으로 더 큰 경우에는 함수가 1을 반환합니다. 두 번째 문자열이 더 크면 함수가 -1을 반환합니다.

멀티바이트 문자일 때는 바이트 인코딩을 기준으로 비교합니다.

BTTEXT_PATTERN_CMP 함수의 동의어입니다.

구문

```
BPCHARCMP(string1, string2)
```

인수

string1

CHAR 문자열 또는 VARCHAR 문자열입니다.

string2

CHAR 문자열 또는 VARCHAR 문자열입니다.

반환 타입

INTEGER

예시

다음 예제에서는 TICKIT 샘플 데이터베이스의 USERS 테이블을 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

USERS 테이블의 처음 10개 항목에 대해 사용자의 이름이 알파벳 순서로 사용자의 성보다 높은지 확인하려면 다음 예를 사용합니다. FIRSTNAME의 문자열이 LASTNAME의 문자열보다 알파벳순으로 뒤

에 있는 항목의 경우 이 함수는 1을 반환합니다. 반대로 LASTNAME이 FIRSTNAME보다 뒤에 있으면 함수는 -1을 반환합니다.

```
SELECT userid, firstname, lastname, BPCHARCMP(firstname, lastname)
FROM users
ORDER BY 1, 2, 3, 4
LIMIT 10;

+--------+-----------+-----------+-----------+
| userid | firstname | lastname  | bpcharcmp |
+--------+-----------+-----------+-----------+
|      1 | Rafael    | Taylor    |        -1 |
|      2 | Vladimir  | Humphrey  |         1 |
|      3 | Lars      | Ratliff   |        -1 |
|      4 | Barry     | Roy       |        -1 |
|      5 | Reagan    | Hodge     |         1 |
|      6 | Victor    | Hernandez |         1 |
|      7 | Tamekah   | Juarez    |         1 |
|      8 | Colton    | Roy       |        -1 |
|      9 | Mufutau   | Watkins   |        -1 |
|     10 | Naida     | Calderon  |         1 |
+--------+-----------+-----------+-----------+
```

함수가 0을 반환하는 USERS 테이블의 모든 항목을 반환하려면 다음 예제를 사용합니다. 이 함수는 FIRSTNAME이 LASTNAME과 같으면 0을 반환합니다.

```
SELECT userid, firstname, lastname,
BPCHARCMP(firstname, lastname)
FROM users
WHERE BPCHARCMP(firstname, lastname)=0
ORDER BY 1, 2, 3, 4;

+--------+-----------+----------+-----------+
| userid | firstname | lastname | bpcharcmp |
+--------+-----------+----------+-----------+
|     62 | Chase     | Chase    |         0 |
|   4008 | Whitney   | Whitney  |         0 |
|  12516 | Graham    | Graham   |         0 |
|  13570 | Harper    | Harper   |         0 |
|  16712 | Cooper    | Cooper   |         0 |
|  18359 | Chase     | Chase    |         0 |
|  27530 | Bradley   | Bradley  |         0 |
|  31204 | Harding   | Harding  |         0 |
```

```
+--------+-----------+----------+-----------+
```

BTRIM 함수

BTRIM 함수는 선행 및 후행 공백을 제거하거나 옵션으로 지정하는 문자열과 일치하는 선행 및 후행 문자를 제거하여 문자열을 잘라냅니다.

구문

```
BTRIM(string [, trim_chars ] )
```

인수

string

잘라낼 입력 VARCHAR 문자열입니다.

trim_chars

일치시킬 문자가 포함된 VARCHAR 문자열입니다.

반환 타입

BTRIM 함수는 VARCHAR 문자열을 반환합니다.

예시

다음은 문자열 ' abc '에서 선행 및 후행 공백을 잘라내는 예입니다.

```
select '     abc    ' as untrim, btrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    | abc
```

다음은 문자열 'xyzaxyzbxyzcxyz'에서 선행 및 후행 'xyz' 문자열을 제거하는 예입니다. 결과를 보면 선행 및 후행 'xyz'만 제거되었고 문자열 내부에서는 제거되지 않았습니다.

```
select 'xyzaxyzbxyzcxyz' as untrim,
btrim('xyzaxyzbxyzcxyz', 'xyz') as trim;
```

```
     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | axyzbxyzc
```

다음 예제에서는 trim_chars 목록 'tes'의 모든 문자와 일치하는 문자열 'setuphistorycassettes'에서 선행 및 후행 부분을 제거합니다. 입력 문자열의 시작 또는 끝에서 trim_chars 목록에 없는 다른 문자 앞에 오는 모든 t, e 또는 s는 제거됩니다.

```
SELECT btrim('setuphistorycassettes', 'tes');

     btrim
-----------------
 uphistoryca
```

BTTEXT_PATTERN_CMP 함수

BPCHARCMP 함수의 동의어입니다.

세부 정보는 BPCHARCMP 함수 섹션을 참조하세요.

CHAR_LENGTH 함수

LEN 함수의 동의어입니다.

LEN 함수 섹션을 참조하세요.

CHARACTER_LENGTH 함수

LEN 함수의 동의어입니다.

LEN 함수 섹션을 참조하세요.

CHARINDEX 함수

문자열 내에서 지정한 하위 문자열의 위치를 반환합니다.

유사한 함수는 POSITION 함수 및 STRPOS 함수 섹션을 참조하세요.

구문

```
CHARINDEX( substring, string )
```

인수

substring

string 내에서 검색할 하위 문자열입니다.

string

검색할 문자열 또는 열입니다.

반환 타입

INTEGER

CHARINDEX 함수는 하위 문자열의 위치에 해당하는 INTEGER를 반환합니다(0이 아닌 1부터 시작). 이 위치는 바이트가 아닌 문자 수를 기준으로 하기 때문에 멀티바이트 문자도 단일 문자로 계산됩니다. 문자열 내에서 하위 문자열을 찾을 수 없는 경우 CHARINDEX는 0을 반환합니다.

예시

dog이라는 단어 내에서 문자열 fish의 위치를 반환하려면 다음 예제를 사용합니다.

```
SELECT CHARINDEX('fish', 'dog');

+-----------+
| charindex |
+-----------+
|         0 |
+-----------+
```

dogfish이라는 단어 내에서 문자열 fish의 위치를 반환하려면 다음 예제를 사용합니다.

```
SELECT CHARINDEX('fish', 'dogfish');

+-----------+
| charindex |
+-----------+
|         4 |
+-----------+
```

다음 예제에서는 TICKIT 샘플 데이터베이스의 SALES 테이블을 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

SALES 테이블에서 999.00 이상의 수수료를 받는 고유 판매 거래 수를 반환하려면 다음 예제를 사용합니다. 이 명령은 커미션 값의 시작 부분에서 소수점이 4자리 이상인지 확인하여 999.00보다 큰 커미션을 계산합니다.

```
SELECT DISTINCT CHARINDEX('.', commission), COUNT (CHARINDEX('.', commission))
FROM sales
WHERE CHARINDEX('.', commission) > 4
GROUP BY CHARINDEX('.', commission)
ORDER BY 1,2;

+-----------+-------+
| charindex | count |
+-----------+-------+
|         5 |   629 |
+-----------+-------+
```

CHR 함수

CHR 함수는 입력 파라미터에서 지정하는 ASCII 코드 포인트 값과 일치하는 문자를 반환합니다.

구문

```
CHR(number)
```

인수

number

입력 파라미터는 ASCII 코드 포인트 값을 나타내는 INTEGER입니다.

반환 타입

CHAR

ASCII 문자가 입력 값과 일치하면 CHAR 함수가 CHAR 문자열을 반환합니다. 입력 숫자와 ASCII 문자가 일치하지 않으면 NULL을 반환합니다.

예시

ASCII 코드 포인트 0에 해당하는 문자를 반환하려면 다음 예시를 사용합니다. 참고로 CHR 함수는 입력 0에 대해 NULL을 반환합니다.

```
SELECT CHR(0);

+-----+
| chr |
+-----+
|     |
+-----+
```

ASCII 코드 포인트 65에 해당하는 문자를 반환하려면 다음 예제를 사용합니다.

```
SELECT CHR(65);

+-----+
| chr |
+-----+
| A   |
+-----+
```

대문자 A(ASCII 코드 포인트 65)로 시작하는 고유 이벤트 이름을 반환하려면 다음 예제를 사용합니다. 다음 예제에서는 TICKIT 샘플 데이터베이스의 EVENT 테이블을 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

```
SELECT DISTINCT eventname FROM event
WHERE SUBSTRING(eventname, 1, 1)=CHR(65) LIMIT 5;

+-----------------------+
|       eventname       |
+-----------------------+
| A Catered Affair      |
| As You Like It        |
| A Man For All Seasons |
| Alan Jackson          |
| Armando Manzanero     |
+-----------------------+
```

COLLATE 함수

COLLATE 함수는 문자열 열 또는 표현식의 데이터 정렬을 재정의합니다.

데이터베이스 데이터 정렬을 사용하여 테이블을 생성하는 방법에 대한 자세한 내용은 CREATE TABLE 섹션을 참조하세요.

데이터베이스 데이터 정렬을 사용하여 데이터베이스를 생성하는 방법에 대한 자세한 내용은 데이터베이스 생성 섹션을 참조하세요.

구문

```
COLLATE( string, 'case_sensitive' | 'case_insensitive');
```

인수

string

재정의하려는 문자열 열 또는 표현식입니다.

'case_sensitive' | 'case_insensitive'

데이터 정렬 이름의 문자열 상수입니다. Amazon Redshift는 case_sensitive 또는 case_insensitive만 지원합니다.

반환 타입

COLLATE 함수는 첫 번째 입력 표현식 형식에 따라 VARCHAR 또는 CHAR를 반환합니다. 이 함수는 첫 번째 입력 인수의 데이터 정렬만 변경하고 출력 값은 변경하지 않습니다.

예시

T 테이블을 만들고 T 테이블의 col1을 case_sensitive로 정의하려면 다음 예제를 사용합니다

```
CREATE TABLE T ( col1 Varchar(20) COLLATE case_sensitive );

INSERT INTO T VALUES ('john'),('JOHN');
```

첫 번째 쿼리를 실행하면 Amazon Redshift는 john만 반환합니다. COLLATE 함수가 col1에서 실행되면 데이터 정렬이 case_insensitive가 됩니다. 두 번째 쿼리는 john과 JOHN을 모두 반환합니다.

```
SELECT * FROM T WHERE col1 = 'john';

+------+
| col1 |
+------+
| john |
+------+
```

```
SELECT * FROM T WHERE COLLATE(col1, 'case_insensitive') = 'john';

+------+
| col1 |
+------+
| john |
| JOHN |
+------+
```

A 테이블을 만들고 A 테이블의 col1을 case_insensitive로 정의하려면 다음 예제를 사용합니다

```
CREATE TABLE A ( col1 Varchar(20) COLLATE case_insensitive );

INSERT INTO A VALUES ('john'),('JOHN');
```

첫 번째 쿼리를 실행하면 Amazon Redshift는 john과 JOHN을 모두 반환합니다. COLLATE 함수가 col1에서 실행되면 데이터 정렬이 case_sensitive가 됩니다. 두 번째 쿼리는 john만 반환합니다.

```
SELECT * FROM A WHERE col1 = 'john';

+------+
| col1 |
+------+
| john |
| JOHN |
+------+

SELECT * FROM A WHERE COLLATE(col1, 'case_sensitive') = 'john';

+------+
| col1 |
+------+
| john |
+------+
```

CONCAT 함수

CONCAT 함수는 두 표현식을 연결하고 결과 표현식을 반환합니다. 2개 이상의 표현식을 연결하려면 CONCAT 함수를 중첩시켜 사용합니다. 두 표현식 사이의 연결 연산자(||)는 CONCAT 함수와 동일한 결과를 반환합니다.

구문

```
CONCAT ( expression1, expression2 )
```

인수

expression1, expression2

두 인수 모두 고정 길이 문자열, 가변 길이 문자열, 2진 표현식 또는 이러한 입력 중 하나로 평가되는 표현식이 될 수 있습니다.

반환 타입

CONCAT는 표현식을 반환합니다. 표현식의 데이터 유형은 입력 인수와 동일합니다.

입력 표현식의 유형이 다른 경우 Amazon Redshift는 표현식 중 하나의 유형 캐스팅을 암시적으로 시도합니다. 값을 캐스팅할 수 없는 경우 오류가 반환됩니다.

사용 노트

- CONCAT 함수와 연결 연산자 모두 표현식 중 하나 또는 둘 모두 NULL이면 결과도 NULL을 반환합니다.

예시

다음 예에서는 문자열 리터럴 2개를 연결합니다:

```
SELECT CONCAT('December 25, ', '2008');

concat
-------------------
December 25, 2008
(1 row)
```

다음은 CONCAT이 아닌 || 연산자를 사용하여 동일한 결과를 반환하는 예입니다.

```
SELECT 'December 25, '||'2008';

?column?
```

```
------------------
December 25, 2008
(1 row)
```

다음 예에서는 다른 CONCAT 함수 내에 중첩된 CONCAT 함수를 사용하여 세 개의 문자열을 연결합니다.

```
SELECT CONCAT('Thursday, ', CONCAT('December 25, ', '2008'));

concat
----------------------------
Thursday, December 25, 2008
(1 row)
```

NULL을 포함할 수 있는 열을 연결하려면 NULL이 발생할 경우 지정된 값을 반환하는 NVL 및 COALESCE 함수를 사용하세요. 다음은 NVL을 사용하여 NULL 값이 발견될 때마다 0을 반환하는 예입니다.

```
SELECT CONCAT(venuename, CONCAT(' seats ', NVL(venueseats, 0))) AS seating
FROM venue WHERE venuestate = 'NV' OR venuestate = 'NC'
ORDER BY 1
LIMIT 5;

seating
-----------------------------------
Ballys Hotel seats 0
Bank of America Stadium seats 73298
Bellagio Hotel seats 0
Caesars Palace seats 0
Harrahs Hotel seats 0
(5 rows)
```

다음은 VENUE 테이블에서 CITY 값과 STATE 값을 연결하는 쿼리입니다.

```
SELECT CONCAT(venuecity, venuestate)
FROM venue
WHERE venueseats > 75000
ORDER BY venueseats;

concat
------------------
```

```
DenverCO
Kansas CityMO
East RutherfordNJ
LandoverMD
(4 rows)
```

다음은 CONCAT 함수를 중첩시켜 사용하는 쿼리입니다. 이 쿼리는 VENUE 테이블에서 CITY 값과 STATE 값을 연결하지만 쉼표와 공백으로 결과 문자열을 구분합니다.

```
SELECT CONCAT(CONCAT(venuecity,', '),venuestate)
FROM venue
WHERE venueseats > 75000
ORDER BY venueseats;

concat
---------------------
Denver, CO
Kansas City, MO
East Rutherford, NJ
Landover, MD
(4 rows)
```

다음 예에서는 2개의 이진 표현식을 연결합니다. 여기서 abc는 이진 값(616263의 16진수 표현)이고 def는 이진 값(646566의 16진수 표현)입니다. 결과는 이진 값의 16진수 표현으로 자동으로 표시됩니다.

```
SELECT CONCAT('abc'::VARBYTE, 'def'::VARBYTE);

concat
-------------------
616263646566
```

CRC32 함수

CRC32는 오류 감지에 사용되는 함수입니다. 이 함수는 CRC32 알고리즘을 사용하여 원본 데이터와 대상 데이터 간의 변경 사항을 감지합니다. CRC32 함수는 가변 길이 문자열을 32비트 이진 시퀀스인 16진수 값을 텍스트로 표현한 8자 문자열로 변환합니다. 원본 데이터와 대상 데이터 간의 변경 사항을 감지하려면 원본 데이터에서 CRC32 함수를 사용하여 출력을 저장합니다. 그런 다음 대상 데이터에서 CRC32 함수를 사용하여 해당 출력을 원본 데이터의 출력과 비교합니다. 데이터가 수정되지 않은 경우 출력이 동일하고 데이터가 수정된 경우 출력이 달라집니다.

구문

```
CRC32(string)
```

인수

string

CHAR 문자열, VARCHAR 문자열 또는 암시적으로 CHAR 또는 VARCHAR 형식으로 평가되는 표현식입니다.

반환 타입

CRC32 함수는 32비트 이진 시퀀스의 16진수 값을 텍스트로 표현한 8자 문자열을 반환합니다. Amazon Redshift CRC32 함수는 CRC-32C 다항식을 기반으로 합니다.

예시

Amazon Redshift 문자열의 8비트 값을 표시하려면 다음 예제를 사용합니다.

```
SELECT CRC32('Amazon Redshift');

+----------+
|  crc32   |
+----------+
| f2726906 |
+----------+
```

DIFFERENCE 함수

DIFFERENCE 함수는 두 문자열의 미국 Soundex 코드를 비교합니다. 이 함수는 Soundex 코드 간에 일치하는 문자 수를 나타내는 INTEGER를 반환합니다.

Soundex 코드는 4자 길이의 문자열입니다. Soundex 코드는 단어의 철자보다는 단어가 어떻게 들리는지를 나타냅니다. 예를 들어 Smith와 Smyth의 Soundex 코드는 동일합니다.

구문

```
DIFFERENCE(string1, string2)
```

인수

string1

CHAR 문자열, VARCHAR 문자열 또는 암시적으로 CHAR 또는 VARCHAR 형식으로 평가되는 표현식입니다.

string2

CHAR 문자열, VARCHAR 문자열 또는 암시적으로 CHAR 또는 VARCHAR 형식으로 평가되는 표현식입니다.

반환 타입

INTEGER

DIFFERENCE 함수는 두 문자열의 미국 Soundex 코드에서 일치하는 문자의 수를 세는 0~4 사이의 INTEGER 값을 반환합니다. Soundex 코드는 4문자로 구성되므로 문자열의 미국 Soundex 코드 값 중 4문자가 모두 같으면 DIFFERENCE 함수는 4를 반환합니다. 두 문자열 중 하나가 비어 있으면 DIFFERENCE는 0을 반환합니다. 두 문자열 모두 유효한 문자를 포함하지 않으면 1을 반환합니다. DIFFERENCE 함수는 a~z 및 A~Z를 포함하여 영어 알파벳 소문자 또는 대문자 ASCII 문자만 변환합니다. DIFFERENCE는 다른 문자를 무시합니다.

예시

문자열 % 및 @의 Soundex 값을 비교하려면 다음 예제를 사용합니다. 두 문자열 모두 유효한 문자를 포함하지 않으므로 함수는 1을 반환합니다.

```
SELECT DIFFERENCE('%', '@');

+------------+
| difference |
+------------+
|          1 |
+------------+
```

Amazon과 빈 문자열의 Soundex 값을 비교하려면 다음 예제를 사용합니다. 이 함수는 두 문자열 중 하나가 비어 있기 때문에 0을 반환합니다.

```
SELECT DIFFERENCE('Amazon', '');
```

```
+------------+
| difference |
+------------+
|          0 |
+------------+
```

문자열 Amazon 및 Ama의 Soundex 값을 비교하려면 다음 예제를 사용합니다. 이 함수는 문자열의 Soundex 값 중 두 문자가 동일하므로 2를 반환합니다.

```
SELECT DIFFERENCE('Amazon', 'Ama');

+------------+
| difference |
+------------+
|          2 |
+------------+
```

문자열 Amazon 및 +-*/%Amazon의 Soundex 값을 비교하려면 다음 예제를 사용합니다. 이 함수는 문자열의 Soundex 값 4개 문자가 모두 동일하기 때문에 4를 반환합니다. 이 함수는 두 번째 문자열의 유효하지 않은 문자 +-*/%를 무시한다는 점에 유의하세요.

```
SELECT DIFFERENCE('Amazon', '+-*/%Amazon');

+------------+
| difference |
+------------+
|          4 |
+------------+
```

문자열 AC/DC 및 Ay See Dee See의 Soundex 값을 비교하려면 다음 예제를 사용합니다. 이 함수는 문자열의 Soundex 값 4개 문자가 모두 동일하기 때문에 4를 반환합니다.

```
SELECT DIFFERENCE('AC/DC', 'Ay See Dee See');

+------------+
| difference |
+------------+
|          4 |
+------------+
```

INITCAP 함수

지정한 문자열에서 각 단어의 첫 번째 글자를 대문자로 변경합니다. INITCAP은 UTF-8 멀티바이트 문자를 지원하여 문자당 최대 4바이트까지 가능합니다.

구문

```
INITCAP(string)
```

인수

string

CHAR 문자열, VARCHAR 문자열 또는 암시적으로 CHAR 또는 VARCHAR 형식으로 평가되는 표현식입니다.

반환 타입

VARCHAR

사용 노트

INITCAP 함수는 문자열에 속한 각 단어의 첫 글자를 대문자로 변경하고 이후 글자는 소문자로 변경하거나 또는 남겨둡니다. 따라서 공백 문자를 제외하고 어떤 문자가 단어 구분자의 역할을 하는지 알아야 합니다. 단어 구분자 문자는 구두점, 기호, 제어 문자 등 알파벳을 제외한 모든 문자를 가리킵니다. 다음 문자는 모두 단어 구분자입니다.

```
! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [ \ ] ^ _ ` { | } ~
```

그 밖에 탭, 줄 바꿈 문자, 폼 피드, 라인 피드, 캐리지 리턴 등도 단어 구분자에 속합니다.

예시

다음 예제에서는 TICKIT 샘플 데이터베이스의 CATEGORY 및 USERS 테이블 데이터를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

CATDESC 열에 있는 각 단어의 첫 글자를 대문자로 사용하려면 다음 예제를 사용합니다.

```
SELECT catid, catdesc, INITCAP(catdesc)
FROM category
```

```
ORDER BY 1, 2, 3;

+-------+-------------------------------------------
+-----------------------------------------+
| catid |                  catdesc                  |                  initcap
          |
+-------+-------------------------------------------
+-----------------------------------------+
|     1 | Major League Baseball                     | Major League Baseball
          |
|     2 | National Hockey League                    | National Hockey League
          |
|     3 | National Football League                  | National Football League
          |
|     4 | National Basketball Association           | National Basketball Association
          |
|     5 | Major League Soccer                       | Major League Soccer
          |
|     6 | Musical theatre                           | Musical Theatre
          |
|     7 | All non-musical theatre                   | All Non-Musical Theatre
          |
|     8 | All opera and light opera                 | All Opera And Light Opera
          |
|     9 | All rock and pop music concerts           | All Rock And Pop Music Concerts
          |
|    10 | All jazz singers and bands                | All Jazz Singers And Bands
          |
|    11 | All symphony, concerto, and choir concerts | All Symphony, Concerto, And
 Choir Concerts |
+-------+-------------------------------------------
+-----------------------------------------+
```

대문자가 단어의 첫머리가 아닐 때 INITCAP 함수가 대문자를 유지하지 않는다는 것을 보여 주려면 다음 예제를 사용합니다. 예를 들어 MLB 문자열은 Mlb가 됩니다.

```
SELECT INITCAP(catname)
FROM category
ORDER BY catname;

+-----------+
|  initcap  |
+-----------+
```

```
| Classical |
| Jazz      |
| Mlb       |
| Mls       |
| Musicals  |
| Nba       |
| Nfl       |
| Nhl       |
| Opera     |
| Plays     |
| Pop       |
+-----------+
```

공백 이외의 영숫자가 아닌 문자가 단어 구분 기호로 기능한다는 것을 보여 주려면 다음 예제를 사용합니다. 각 문자열의 여러 문자가 대문자로 표시됩니다.

```
SELECT email, INITCAP(email)
FROM users
ORDER BY userid DESC LIMIT 5;

+------------------------------------+------------------------------------+
|               email                |              initcap               |
+------------------------------------+------------------------------------+
| urna.Ut@egetdictumplacerat.edu     | Urna.Ut@Egetdictumplacerat.Edu     |
| nibh.enim@egestas.ca               | Nibh.Enim@Egestas.Ca               |
| in@Donecat.ca                      | In@Donecat.Ca                      |
| sodales@blanditviverraDonec.ca     | Sodales@Blanditviverradonec.Ca     |
| sociis.natoque.penatibus@vitae.org | Sociis.Natoque.Penatibus@Vitae.Org |
+------------------------------------+------------------------------------+
```

LEFT 및 RIGHT 함수

이 두 함수는 문자열의 가장 왼쪽 또는 가장 오른쪽에서 지정한 만큼 문자 수를 반환합니다.

반환되는 문자 수는 바이트가 아닌 문자 수를 기준으로 하기 때문에 멀티바이트 문자도 단일 문자로 계산됩니다.

구문

```
LEFT( string,  integer )

RIGHT( string,  integer )
```

인수

string

CHAR 문자열, VARCHAR 문자열 또는 CHAR 또는 VARCHAR 문자열로 평가되는 모든 표현식입니다.

integer

양의 정수입니다.

반환 타입

VARCHAR

예시

다음 예제에서는 TICKIT 샘플 데이터베이스의 EVENT 테이블 데이터를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

이벤트 ID가 1000에서 1005 사이인 이벤트 이름에서 가장 왼쪽 5자와 가장 오른쪽 5자를 반환하려면 다음 예제를 사용합니다.

```
SELECT eventid, eventname,
LEFT(eventname,5) AS left_5,
RIGHT(eventname,5) AS right_5
FROM event
WHERE eventid BETWEEN 1000 AND 1005
ORDER BY 1;

+---------+----------------+--------+---------+
| eventid |   eventname    | left_5 | right_5 |
+---------+----------------+--------+---------+
|    1000 | Gypsy          | Gypsy  | Gypsy   |
|    1001 | Chicago        | Chica  | icago   |
|    1002 | The King and I | The K  | and I   |
|    1003 | Pal Joey       | Pal J  | Joey    |
|    1004 | Grease         | Greas  | rease   |
|    1005 | Chicago        | Chica  | icago   |
+---------+----------------+--------+---------+
```

LEN 함수

지정된 문자열의 길이를 바이트 수대로 반환합니다.

구문

LEN은 LENGTH 함수, CHAR_LENGTH 함수, CHARACTER_LENGTH 함수 및 TEXTLEN 함수의 동의어입니다.

```
LEN(expression)
```

인수

expression

CHAR 문자열, VARCHAR 문자열, VARBYTE 표현식 또는 암시적으로 CHAR, VARCHAR 또는 VARBYTE 형식으로 평가되는 표현식입니다.

반환 타입

INTEGER

LEN 함수는 입력 문자열의 문자 수를 나타내는 정수를 반환합니다.

입력 문자열이 문자열인 경우 LEN 함수는 바이트 수가 아닌 멀티바이트 문자열의 실제 문자 수를 반환합니다. 예를 들어 VARCHAR(12) 열에 4바이트 중국 문자 3개가 저장되어야 한다고 가정했을 때 LEN 함수는 동일한 문자열에서 3을 반환합니다. 문자열의 길이가 몇 바이트인지 알아보려면 OCTET_LENGTH 함수를 쓰십시오.

사용 노트

표현식이 CHAR 문자열인 경우 후행 공백은 계산되지 않습니다.

표현식이 VARCHAR 문자열인 경우 후행 공백이 계산됩니다.

예시

français 문자열의 바이트 수와 문자 수를 반환하려면 다음 예제를 사용합니다.

```
SELECT OCTET_LENGTH('français'),
LEN('français');

+--------------+-----+
| octet_length | len |
+--------------+-----+
|            9 |   8 |
```

```
+--------------+-----+
```

OCTET_LENGTH 함수를 사용하지 않고 français 문자열에 포함된 바이트 수와 문자 수를 반환하려면 다음 예제를 사용합니다. 자세한 내용은 CAST 함수을 참조하세요.

```
SELECT LEN(CAST('français' AS VARBYTE)) as bytes, LEN('français');

+-------+-----+
| bytes | len |
+-------+-----+
|     9 |   8 |
+-------+-----+
```

후행 공백이 없는 문자열 cat, 후행 공백이 3개인 cat , 길이 6의 CHAR로 캐스팅된 후행 공백이 3개인 cat , 길이 6의 VARCHAR로 캐스팅된 후행 공백이 3개인 cat 에 포함된 문자 수를 반환하려면 다음 예제를 사용합니다. 이 함수는 CHAR 문자열의 경우 후행 공백을 계산하지 않지만 VARCHAR 문자열의 경우 후행 공백을 계산합니다.

```
SELECT LEN('cat'), LEN('cat   '), LEN(CAST('cat   ' AS CHAR(6))) AS len_char,
 LEN(CAST('cat   ' AS VARCHAR(6))) AS len_varchar;

+-----+-----+----------+-------------+
| len | len | len_char | len_varchar |
+-----+-----+----------+-------------+
|   3 |   6 |        3 |           6 |
+-----+-----+----------+-------------+
```

다음 예제에서는 TICKIT 샘플 데이터베이스의 VENUE 테이블 데이터를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

VENUE 테이블에서 가장 긴 10개 장소 이름을 반환하려면 다음 예제를 사용합니다.

```
SELECT venuename, LEN(venuename)
FROM venue
ORDER BY 2 DESC, 1
LIMIT 10;

+----------------------------------------+-----+
|                venuename               | len |
+----------------------------------------+-----+
| Saratoga Springs Performing Arts Center |  39 |
| Lincoln Center for the Performing Arts  |  38 |
```

```
| Nassau Veterans Memorial Coliseum       |  33 |
| Jacksonville Municipal Stadium          |  30 |
| Rangers BallPark in Arlington           |  29 |
| University of Phoenix Stadium           |  29 |
| Circle in the Square Theatre            |  28 |
| Hubert H. Humphrey Metrodome            |  28 |
| Oriole Park at Camden Yards             |  27 |
| Dick's Sporting Goods Park              |  26 |
+-----------------------------------------+-----+
```

LENGTH 함수

LEN 함수의 동의어입니다.

LEN 함수 섹션을 참조하세요.

LOWER 함수

문자열을 소문자로 변환합니다. LOWER는 UTF-8 멀티바이트 문자를 지원하여 문자당 최대 4바이트까지 가능합니다.

구문

```
LOWER(string)
```

인수

string

VARCHAR 문자열 또는 VARCHAR 형식으로 평가되는 표현식입니다.

반환 타입

문자열

LOWER 함수는 입력 문자열과 데이터 형식이 동일한 문자열을 반환합니다. 예를 들어 입력이 CHAR 문자열인 경우 함수는 CHAR 문자열을 반환합니다.

예시

다음 예제에서는 TICKIT 샘플 데이터베이스의 CATEGORY 테이블에 있는 데이터를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

CATNAME 열의 VARCHAR 문자열을 소문자로 변환하려면 다음 예제를 사용합니다.

```
SELECT catname, LOWER(catname) FROM category ORDER BY 1,2;

+-----------+-----------+
|  catname  |   lower   |
+-----------+-----------+
| Classical | classical |
| Jazz      | jazz      |
| MLB       | mlb       |
| MLS       | mls       |
| Musicals  | musicals  |
| NBA       | nba       |
| NFL       | nfl       |
| NHL       | nhl       |
| Opera     | opera     |
| Plays     | plays     |
| Pop       | pop       |
+-----------+-----------+
```

LPAD 및 RPAD 함수

이 두 함수는 지정한 길이에 따라 문자열에 문자를 추가 또는 첨부합니다.

구문

```
LPAD(string1, length, [ string2 ])
```

```
RPAD(string1, length, [ string2 ])
```

인수

string1

CHAR 문자열, VARCHAR 문자열 또는 암시적으로 CHAR 또는 VARCHAR 형식으로 평가되는 표현식입니다.

length

함수의 결과 길이를 정의하는 정수입니다. 문자열의 길이는 바이트가 아닌 문자 수를 기준으로 하기 때문에 멀티바이트 문자도 단일 문자로 계산됩니다. string1이 지정한 길이보다 길면 오른쪽에서 절사됩니다. length가 0이거나 음수면 함수 결과로 빈 문자열이 반환됩니다.

string2

(선택) string1에 추가 또는 첨부되는 1개 이상의 문자입니다. 이 인수를 지정하지 않으면 공백이 사용됩니다.

반환 타입

VARCHAR

예시

다음 예제에서는 TICKIT 샘플 데이터베이스의 EVENT 테이블에 있는 데이터를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

지정된 이벤트 이름 집합을 20자로 자르고 더 짧은 이름 앞에 공백을 추가하려면 다음 예제를 사용합니다.

```
SELECT LPAD(eventname, 20) FROM event
WHERE eventid BETWEEN 1 AND 5 ORDER BY 1;

+----------------------+
|         lpad         |
+----------------------+
|               Salome |
|        Il Trovatore |
|       Boris Godunov |
|     Gotterdammerung |
|La Cenerentola (Cind |
+----------------------+
```

동일한 이벤트 이름 집합을 20자로 줄이되 더 짧은 이름에 0123456789을 추가하려면 다음 예제를 사용합니다.

```
SELECT RPAD(eventname, 20,'0123456789') FROM event
WHERE eventid BETWEEN 1 AND 5 ORDER BY 1;

+----------------------+
|         rpad         |
+----------------------+
| Boris Godunov0123456 |
| Gotterdammerung01234 |
| Il Trovatore01234567 |
```

```
| La Cenerentola (Cind |
| Salome01234567890123 |
+----------------------+
```

ltrim 함수

문자열의 시작 부분부터 문자를 잘라냅니다. 잘라낸 문자 목록에서 문자만 포함하는 가장 긴 문자열을 제거합니다. 잘라내기 문자가 입력 문자열에 나타나지 않으면 잘라내기가 완료된 것입니다.

구문

```
LTRIM( string [, trim_chars] )
```

인수

string

잘라낼 문자열 열, 표현식 또는 문자열 리터럴입니다.

trim_chars

문자열의 처음부터 잘라낼 문자를 나타내는 문자열 열, 표현식 또는 문자열 리터럴입니다. 지정하지 않으면 공백이 잘라내기 문자로 사용됩니다.

반환 타입

LTRIM 함수는 입력 문자열(CHAR 또는 VARCHAR)과 데이터 유형이 동일한 문자열을 반환합니다.

예시

다음은 `listime` 열에서 연도를 잘라내는 예입니다. 문자열 리터럴 `'2008-'`의 잘라내기 문자는 왼쪽부터 잘라낼 문자를 나타냅니다. 잘라내기 문자 `'028-'`을 사용하는 경우에도 동일한 결과를 얻을 수 있습니다.

```
select listid, listtime, ltrim(listtime, '2008-')
from listing
order by 1, 2, 3
limit 10;

listid |      listtime       |     ltrim
-------+---------------------+----------------
     1 | 2008-01-24 06:43:29 | 1-24 06:43:29
     2 | 2008-03-05 12:25:29 | 3-05 12:25:29
```

```
     3 | 2008-11-01 07:35:33 | 11-01 07:35:33
     4 | 2008-05-24 01:18:37 | 5-24 01:18:37
     5 | 2008-05-17 02:29:11 | 5-17 02:29:11
     6 | 2008-08-15 02:08:13 | 15 02:08:13
     7 | 2008-11-15 09:38:15 | 11-15 09:38:15
     8 | 2008-11-09 05:07:30 | 11-09 05:07:30
     9 | 2008-09-09 08:03:36 | 9-09 08:03:36
    10 | 2008-06-17 09:44:54 | 6-17 09:44:54
```

LTRIM은 trim_chars의 문자가 string의 첫 문자이면 모두 제거합니다. 다음은 'C', 'D', 'G' 문자가 VARCHAR 열인 VENUENAME의 첫 문자일 때 각 문자를 잘라내는 예입니다.

```
select venueid, venuename, ltrim(venuename, 'CDG')
from venue
where venuename like '%Park'
order by 2
limit 7;

venueid | venuename                  | btrim
--------+----------------------------+-------------------------
    121 | ATT Park                   | ATT Park
    109 | Citizens Bank Park         | itizens Bank Park
    102 | Comerica Park              | omerica Park
      9 | Dick's Sporting Goods Park | ick's Sporting Goods Park
     97 | Fenway Park                | Fenway Park
    112 | Great American Ball Park   | reat American Ball Park
    114 | Miller Park                | Miller Park
```

다음 예제에서는 venueid 열에서 검색된 잘라내기 문자 2를 사용합니다.

```
select ltrim('2008-01-24 06:43:29', venueid)
from venue where venueid=2;

ltrim
------------------
008-01-24 06:43:29
```

다음 예제에서는 2가 '0' 잘라내기 문자 앞에서 발견되었기 때문에 어떤 문자도 잘라내지 않습니다.

```
select ltrim('2008-01-24 06:43:29', '0');
```

```
ltrim
-------------------
2008-01-24 06:43:29
```

다음 예제에서는 기본 공백 잘라내기 문자를 사용하여 문자열의 시작 부분부터 두 개의 공백을 잘라냅니다.

```
select ltrim('  2008-01-24 06:43:29');

ltrim
-------------------
2008-01-24 06:43:29
```

OCTETINDEX 함수

OCTETINDEX 함수는 문자열 내의 하위 문자열 위치를 바이트 수로 반환합니다.

구문

```
OCTETINDEX(substring, string)
```

인수

substring

CHAR 문자열, VARCHAR 문자열 또는 암시적으로 CHAR 또는 VARCHAR 형식으로 평가되는 표현식입니다.

string

CHAR 문자열, VARCHAR 문자열 또는 암시적으로 CHAR 또는 VARCHAR 형식으로 평가되는 표현식입니다.

반환 타입

INTEGER

OCTETINDEX 함수는 문자열 내 하위 문자열의 위치에 해당하는 INTEGER 값을 바이트 수로 반환하며, 여기서 문자열의 첫 번째 문자는 로 계산됩니다. 문자열에 멀티바이트 문자가 포함되어 있지 않으면 결과는 CHARINDEX 함수의 결과와 같습니다. 문자열에 하위 문자열이 포함되어 있지 않으면 함수는 0을 반환하고 하위 문자열이 비어 있으면 함수는 1을 반환합니다.

예시

q 문자열에서 하위 문자열 Amazon Redshift의 위치를 반환하려면 다음 예제를 사용합니다. 이 예제에서는 하위 문자열이 문자열에 없으므로 0을 반환합니다.

```
SELECT OCTETINDEX('q', 'Amazon Redshift');

+------------+
| octetindex |
+------------+
|          0 |
+------------+
```

문자열 Amazon Redshift에서 비어 있는 하위 문자열의 위치를 반환하려면 다음 예제를 사용합니다. 이 예에서는 하위 문자열이 비어 있으므로 1을 반환합니다.

```
SELECT OCTETINDEX('', 'Amazon Redshift');

+------------+
| octetindex |
+------------+
|          1 |
+------------+
```

Redshift 문자열에서 하위 문자열 Amazon Redshift의 위치를 반환하려면 다음 예제를 사용합니다. 이 예제에서는 하위 문자열이 문자열의 8번째 바이트에서 시작하므로 8을 반환합니다.

```
SELECT OCTETINDEX('Redshift', 'Amazon Redshift');

+------------+
| octetindex |
+------------+
|          8 |
+------------+
```

Redshift 문자열에서 하위 문자열 Amazon Redshift의 위치를 반환하려면 다음 예제를 사용합니다. 이 예제에서는 문자열의 처음 6자가 더블바이트 문자이므로 21을 반환합니다.

```
SELECT OCTETINDEX('Redshift', 'Άμαζον Amazon Redshift');
```

```
+------------+
| octetindex |
+------------+
|         21 |
+------------+
```

OCTET_LENGTH 함수

지정된 문자열의 길이를 바이트 수대로 반환합니다.

구문

```
OCTET_LENGTH(expression)
```

인수

expression

CHAR 문자열, VARCHAR 문자열, VARBYTE 표현식 또는 암시적으로 CHAR, VARCHAR 또는 VARBYTE 형식으로 평가되는 표현식입니다.

반환 타입

INTEGER

OCTET_LENGTH 함수는 입력 문자열의 바이트 수를 나타내는 정수를 반환합니다.

입력 문자열이 문자열인 경우 LEN 함수는 바이트 수가 아닌 멀티바이트 문자열의 실제 문자 수를 반환합니다. 예를 들어 VARCHAR(12) 열에 4바이트 중국 문자 3개가 저장되어야 한다고 가정했을 때 OCTET_LENGTH 함수는 해당 문자열에 대해 12를 반환하고 LEN 함수는 동일한 문자열에 대해 3을 반환합니다.

사용 노트

표현식이 CHAR 문자열인 경우 함수는 CHAR 문자열의 길이를 반환합니다. 예를 들어 CHAR(6) 입력의 출력은 CHAR(6)입니다.

표현식이 VARCHAR 문자열인 경우 후행 공백이 계산됩니다.

예시

후행 공백 3개가 있는 문자열 francais를 CHAR 및 VARCHAR 형식으로 캐스팅할 때 바이트 수를 반환하려면 다음 예제를 사용합니다. 자세한 내용은 CAST 함수을 참조하세요.

```
SELECT OCTET_LENGTH(CAST('francais   ' AS CHAR(15))) AS octet_length_char,
 OCTET_LENGTH(CAST('francais   ' AS VARCHAR(15))) AS octet_length_varchar;

+------------------+----------------------+
| octet_length_char | octet_length_varchar |
+------------------+----------------------+
|                15 |                   11 |
+------------------+----------------------+
```

français 문자열의 바이트 수와 문자 수를 반환하려면 다음 예제를 사용합니다.

```
SELECT OCTET_LENGTH('français'), LEN('français');

+--------------+-----+
| octet_length | len |
+--------------+-----+
|            9 |   8 |
+--------------+-----+
```

문자열 français를 VARBYTE로 캐스팅할 때 바이트 수를 반환하려면 다음 예제를 사용합니다.

```
SELECT OCTET_LENGTH(CAST('français' AS VARBYTE));

+--------------+
| octet_length |
+--------------+
|            9 |
+--------------+
```

POSITION 함수

문자열 내에서 지정한 하위 문자열의 위치를 반환합니다.

유사한 함수는 CHARINDEX 함수 및 STRPOS 함수 섹션을 참조하세요.

구문

```
POSITION(substring IN string )
```

인수

substring

string 내에서 검색할 하위 문자열입니다.

string

검색할 문자열 또는 열입니다.

반환 타입

POSITION 함수는 하위 문자열의 위치에 해당하는 INTEGER를 반환합니다(0이 아닌 1부터 시작). 이 위치는 바이트가 아닌 문자 수를 기준으로 하기 때문에 멀티바이트 문자도 단일 문자로 계산됩니다. 문자열 내에서 하위 문자열이 발견되지 않으면 POSITION이 0을 반환합니다.

예시

dog이라는 단어 내에서 문자열 fish의 위치를 반환하려면 다음 예제를 사용합니다.

```
SELECT POSITION('fish' IN 'dog');

+-----------+
|  position |
+-----------+
|         0 |
+-----------+
```

dogfish이라는 단어 내에서 문자열 fish의 위치를 반환하려면 다음 예제를 사용합니다.

```
SELECT POSITION('fish' IN 'dogfish');

+-----------+
|  position |
+-----------+
|         4 |
+-----------+
```

다음 예제에서는 TICKIT 샘플 데이터베이스의 SALES 테이블을 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

SALES 테이블에서 999.00 이상의 수수료를 받는 고유 판매 거래 수를 반환하려면 다음 예제를 사용합니다. 이 명령은 커미션 값의 시작 부분에서 소수점이 4자리 이상인지 확인하여 999.00보다 큰 커미션을 계산합니다.

```
SELECT DISTINCT POSITION('.' IN commission), COUNT (POSITION('.' IN commission))
FROM sales
WHERE POSITION('.' IN commission) > 4
GROUP BY POSITION('.' IN commission)
ORDER BY 1,2;

+-----------+-------+
|  position | count |
+-----------+-------+
|         5 |   629 |
+-----------+-------+
```

QUOTE_IDENT 함수

QUOTE_IDENT 함수는 지정된 문자열을 선행 큰따옴표와 후행 큰따옴표가 있는 문자열로 반환합니다. 함수 출력은 SQL 문에서 식별자로 사용할 수 있습니다. 함수는 포함된 큰 따옴표를 적절하게 두 배로 늘립니다.

QUOTE_IDENT는 문자열에 식별자가 아닌 문자가 포함되어 있거나 그렇지 않으면 소문자로 변환될 때 유효한 식별자를 생성하는 데 필요한 경우에만 큰따옴표를 추가합니다. 항상 작은 따옴표로 묶인 문자열을 반환하려면 QUOTE_LITERAL을 사용합니다.

구문

```
QUOTE_IDENT(string)
```

인수

string

CHAR 또는 VARCHAR 문자열입니다.

반환 타입

QUOTE_IDENT 함수는 입력 문자열과 동일한 형식의 문자열을 반환합니다.

예시

큰따옴표로 묶은 문자열 "CAT"을 반환하려면 다음 예제를 사용합니다.

```
SELECT QUOTE_IDENT('"CAT"');

+-------------+
| quote_ident |
+-------------+
| """CAT"""   |
+-------------+
```

다음 예제에서는 TICKIT 샘플 데이터베이스의 CATEGORY 테이블에 있는 데이터를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

따옴표로 묶인 CATNAME 열을 반환하려면 다음 예제를 사용합니다.

```
SELECT catid, QUOTE_IDENT(catname)
FROM category
ORDER BY 1,2;

+-------+-------------+
| catid | quote_ident |
+-------+-------------+
|     1 | "MLB"       |
|     2 | "NHL"       |
|     3 | "NFL"       |
|     4 | "NBA"       |
|     5 | "MLS"       |
|     6 | "Musicals"  |
|     7 | "Plays"     |
|     8 | "Opera"     |
|     9 | "Pop"       |
|    10 | "Jazz"      |
|    11 | "Classical" |
+-------+-------------+
```

QUOTE_LITERAL 함수

QUOTE_LITERAL 함수는 지정한 문자열을 SQL 문에서 문자열 리터럴로 사용할 수 있도록 작은 따옴표로 묶인 문자열로 반환합니다. 입력 파라미터가 숫자라고 해도 QUOTE_LITERAL은 이를 문자열로 처리합니다. 작은따옴표와 백슬래시 모두 되풀이하여 두 번 사용됩니다.

구문

```
QUOTE_LITERAL(string)
```

인수

string

CHAR 또는 VARCHAR 문자열입니다.

반환 타입

QUOTE_LITERAL 함수는 입력 문자열(CHAR 또는 VARCHAR)과 데이터 형식이 동일한 문자열을 반환합니다.

예시

작은 따옴표로 묶은 문자열 ''CAT''을 반환하려면 다음 예제를 사용합니다.

```
SELECT QUOTE_LITERAL('''CAT''');

+---------------+
| quote_literal |
+---------------+
| '''CAT'''     |
+---------------+
```

다음 예제에서는 TICKIT 샘플 데이터베이스의 CATEGORY 테이블에 있는 데이터를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

다음은 CATNAME 열을 작은 따옴표로 묶어서 반환하는 예입니다.

```
SELECT catid, QUOTE_LITERAL(catname)
FROM category
ORDER BY 1,2;
```

```
+-------+---------------+
| catid | quote_literal |
+-------+---------------+
|     1 | 'MLB'         |
|     2 | 'NHL'         |
|     3 | 'NFL'         |
|     4 | 'NBA'         |
|     5 | 'MLS'         |
|     6 | 'Musicals'    |
|     7 | 'Plays'       |
|     8 | 'Opera'       |
|     9 | 'Pop'         |
|    10 | 'Jazz'        |
|    11 | 'Classical'   |
+-------+---------------+
```

다음은 CATID 열을 작은 따옴표로 묶어서 반환하는 예입니다.

```
SELECT QUOTE_LITERAL(catid), catname
FROM category
ORDER BY 1,2;

+---------------+-----------+
| quote_literal |  catname  |
+---------------+-----------+
| '1'           | MLB       |
| '10'          | Jazz      |
| '11'          | Classical |
| '2'           | NHL       |
| '3'           | NFL       |
| '4'           | NBA       |
| '5'           | MLS       |
| '6'           | Musicals  |
| '7'           | Plays     |
| '8'           | Opera     |
| '9'           | Pop       |
+---------------+-----------+
```

REGEXP_COUNT 함수

문자열에서 정규 표현식 패턴을 검색한 후 해당 패턴 발생 횟수를 나타내는 정수를 반환합니다. 일치하는 결과가 발견되지 않으면 함수가 0을 반환합니다. 정규 표현식에 관한 자세한 내용은 POSIX 연산자 단원 및 Wikipedia의 정규 표현식을 참조하세요.

구문

```
REGEXP_COUNT( source_string, pattern [, position [, parameters ] ] )
```

인수

source_string

CHAR 또는 VARCHAR 문자열입니다.

패턴

정규 표현식 패턴을 나타내는 UTF-8 문자열 리터럴입니다. 자세한 내용은 POSIX 연산자 단원을 참조하십시오.

position

(선택) 검색을 시작할 source_string 내 위치를 나타내는 양수 INTEGER입니다. 이 위치는 바이트가 아닌 문자 수를 기준으로 하기 때문에 멀티바이트 문자도 단일 문자로 계산됩니다. 기본값은 1입니다. position이 1보다 작으면 검색이 source_string의 첫 문자부터 시작됩니다. position이 source_string의 문자 수보다 크면 결과는 0이 됩니다.

parameters

(선택) 함수가 패턴과 일치하는 방법을 나타내는 하나 이상의 문자열 리터럴입니다. 가능한 값은 다음과 같습니다.

- c - 대/소문자를 구분하여 일치시킵니다. 기본값은 대/소문자 구분 일치를 사용하는 것입니다.
- i - 대/소문자를 구분하지 않고 일치시킵니다.
- p – PCRE(Perl Compatible Regular Expression) 방언으로 패턴을 해석합니다. PCRE에 관한 자세한 내용은 Wikipedia의 펄 호환 정규 표현식을 참조하세요.

반환 타입

INTEGER

예시

3자 시퀀스가 발생하는 횟수를 계산하려면 다음 예제를 사용합니다.

```
SELECT REGEXP_COUNT('abcdefghijklmnopqrstuvwxyz', '[a-z]{3}');
```

```
+--------------+
| regexp_count |
+--------------+
|            8 |
+--------------+
```

대/소문자를 구분하지 않는 일치를 사용하여 문자열 FOX의 발생 횟수를 계산하려면 다음 예제를 사용합니다.

```
SELECT REGEXP_COUNT('the fox', 'FOX', 1, 'i');

+--------------+
| regexp_count |
+--------------+
|            1 |
+--------------+
```

PCRE 방언으로 작성된 패턴을 사용하여 하나 이상의 숫자와 하나의 소문자가 포함된 단어를 찾으려면 다음 예제를 사용합니다. 이 예에서는 ?= 연산자를 사용하는데, 이 연산자는 PCRE에서 특정 앞을 내다보는 의미를 갖습니다. 이 예에서는 대/소문자를 구분하여 일치하는 단어의 발생 횟수를 계산합니다.

```
SELECT REGEXP_COUNT('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'p');

+--------------+
| regexp_count |
+--------------+
|            2 |
+--------------+
```

PCRE 방언으로 작성된 패턴을 사용하여 하나 이상의 숫자와 하나의 소문자가 포함된 단어를 찾으려면 다음 예제를 사용합니다. PCRE에서 특정한 의미를 지닌 ?= 연산자가 사용됩니다. 이 예는 이러한 단어의 발생 횟수를 계산하지만 대/소문자를 구분하지 않는 일치를 사용한다는 점에서 이전 예와 다릅니다.

```
SELECT REGEXP_COUNT('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'ip');
```

```
+--------------+
| regexp_count |
+--------------+
|            3 |
+--------------+
```

다음 예제에서는 TICKIT 샘플 데이터베이스의 USERS 테이블 데이터를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

최상위 도메인 이름이 org 또는 edu인 횟수를 계산하려면 다음 예제를 사용합니다.

```
SELECT email, REGEXP_COUNT(email,'@[^.]*\.(org|edu)') FROM users
ORDER BY userid LIMIT 4;

+----------------------------------------------+--------------+
|                     email                    | regexp_count |
+----------------------------------------------+--------------+
| Etiam.laoreet.libero@sodalesMaurisblandit.edu |            1 |
| Suspendisse.tristique@nonnisiAenean.edu       |            1 |
| amet.faucibus.ut@condimentumegetvolutpat.ca   |            0 |
| sed@lacusUtnec.ca                             |            0 |
+----------------------------------------------+--------------+
```

REGEXP_INSTR 함수

문자열에서 정규 표현식 패턴을 검색하여 일치하는 하위 문자열의 시작 위치 또는 종료 위치를 나타내는 정수를 반환합니다. 일치하는 결과가 발견되지 않으면 함수가 0을 반환합니다. REGEXP_INSTR은 POSITION 함수와 비슷하지만 문자열에서 정규 표현식 패턴을 검색할 수 있습니다. 정규 표현식에 관한 자세한 내용은 POSIX 연산자 단원 및 Wikipedia의 정규 표현식을 참조하세요.

구문

```
REGEXP_INSTR( source_string, pattern [, position [, occurrence] [, option [, parameters
 ] ] ] ] )
```

인수

source_string

열 이름 같이 검색할 문자열 표현식입니다.

패턴

정규 표현식 패턴을 나타내는 UTF-8 문자열 리터럴입니다. 자세한 내용은 POSIX 연산자 단원을 참조하십시오.

position

(선택) 검색을 시작할 source_string 내 위치를 나타내는 양수 INTEGER입니다. 이 위치는 바이트가 아닌 문자 수를 기준으로 하기 때문에 멀티바이트 문자도 단일 문자로 계산됩니다. 기본값은 1입니다. position이 1보다 작으면 검색이 source_string의 첫 문자부터 시작됩니다. position이 source_string의 문자 수보다 크면 결과는 0이 됩니다.

발생

(선택) 사용할 패턴 발생을 나타내는 양의 INTEGER수입니다. REGEXP_INSTR은 첫 번째 일치 항목 *occurrence*-1을 건너뜁니다. 기본값은 1입니다. occurrence가 1보다 작거나 source_string에 있는 문자 수보다 클 경우 검색이 무시되고 결과가 0이 됩니다.

option

(선택) 일치하는 항목의 첫 번째 문자 위치(0)를 반환할지 일치하는 항목의 끝 다음에 나오는 첫 번째 문자의 위치(1)를 반환할지 여부를 나타내는 값입니다. 0이 아닌 값은 1과 같습니다. 기본값은 0입니다.

parameters

(선택) 함수가 패턴과 일치하는 방법을 나타내는 하나 이상의 문자열 리터럴입니다. 가능한 값은 다음과 같습니다.

- c - 대/소문자를 구분하여 일치시킵니다. 기본값은 대/소문자 구분 일치를 사용하는 것입니다.
- i - 대/소문자를 구분하지 않고 일치시킵니다.
- e - 하위 표현식을 사용하여 하위 문자열을 추출합니다.

  패턴에 하위 표현식이 포함되어 있을 경우 REGEXP_INSTR은 패턴의 첫 번째 하위 표현식을 사용하여 하위 문자열과 일치시킵니다. REGEXP_INSTR은 첫 번째 하위 표현식만 고려하며 추가 하위 표현식은 무시됩니다. 패턴에 하위 표현식이 없으면 REGEXP_INSTR이 'e' 파라미터를 무시합니다.
- p – PCRE(Perl Compatible Regular Expression) 방언으로 패턴을 해석합니다. PCRE에 관한 자세한 내용은 Wikipedia의 펄 호환 정규 표현식을 참조하세요.

반환 타입

Integer

예시

다음 예제에서는 TICKIT 샘플 데이터베이스의 USERS 테이블을 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

도메인 이름을 시작하는 @ 문자를 검색하고 첫 번째 일치 항목의 시작 위치를 반환하려면 다음 예제를 사용합니다.

```
SELECT email, REGEXP_INSTR(email, '@[^.]*')
FROM users
ORDER BY userid LIMIT 4;

+----------------------------------------------+--------------+
|                     email                    | regexp_instr |
+----------------------------------------------+--------------+
| Etiam.laoreet.libero@sodalesMaurisblandit.edu |           21 |
| Suspendisse.tristique@nonnisiAenean.edu       |           22 |
| amet.faucibus.ut@condimentumegetvolutpat.ca   |           17 |
| sed@lacusUtnec.ca                             |            4 |
+----------------------------------------------+--------------+
```

Center라는 단어의 변형을 검색하고 첫 번째 일치 항목의 시작 위치를 반환하려면 다음 예제를 사용합니다.

```
SELECT venuename, REGEXP_INSTR(venuename,'[cC]ent(er|re)$')
FROM venue
WHERE REGEXP_INSTR(venuename,'[cC]ent(er|re)$') > 0
ORDER BY venueid LIMIT 4;

+----------------------+--------------+
|       venuename      | regexp_instr |
+----------------------+--------------+
| The Home Depot Center |           16 |
| Izod Center           |            6 |
| Wachovia Center       |           10 |
| Air Canada Centre     |           12 |
+----------------------+--------------+
```

대소문자를 구분하지 않는 일치 논리를 사용하여 문자열 FOX의 첫 번째 발생 위치를 찾으려면 다음 예제를 사용합니다.

```
SELECT REGEXP_INSTR('the fox', 'FOX', 1, 1, 0, 'i');
```

```
+--------------+
| regexp_instr |
+--------------+
|            5 |
+--------------+
```

PCRE 방언으로 작성된 패턴을 사용하여 하나 이상의 숫자와 하나의 소문자가 포함된 단어를 찾으려면 다음 예제를 사용합니다. PCRE에서 특정 미리 보기 의미가 있는 ?= 연산자가 사용됩니다. 이 예에서는 두 번째 단어의 시작 위치를 찾습니다.

```
SELECT REGEXP_INSTR('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'p');

+--------------+
| regexp_instr |
+--------------+
|           21 |
+--------------+
```

PCRE 방언으로 작성된 패턴을 사용하여 하나 이상의 숫자와 하나의 소문자가 포함된 단어를 찾으려면 다음 예제를 사용합니다. PCRE에서 특정 미리 보기 의미가 있는 ?= 연산자가 사용됩니다. 이 예는 두 번째 단어의 시작 위치를 찾지만 대/소문자를 구분하지 않는 일치를 사용한다는 점에서 이전 예와 다릅니다.

```
SELECT REGEXP_INSTR('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'ip');

+--------------+
| regexp_instr |
+--------------+
|           15 |
+--------------+
```

REGEXP_REPLACE 함수

문자열에서 정규 표현식 패턴을 검색한 후 발견되는 모든 패턴을 지정한 문자열로 변경합니다. REGEXP_REPLACE는 REPLACE 함수과 비슷하지만 문자열에서 정규 표현식 패턴을 검색할 수 있습니다. 정규 표현식에 관한 자세한 내용은 POSIX 연산자 단원 및 Wikipedia의 정규 표현식을 참조하세요.

REGEXP_REPLACE는 TRANSLATE 함수 및 REPLACE 함수과 비슷합니다. 단, TRANSLATE는 단일 문자를 여러 차례 변경하고, REPLACE는 전체 문자열 하나를 다른 문자열로 변경하는 반면 REGEXP_REPLACE는 문자열에서 정규 표현식 패턴을 검색할 수 있습니다.

구문

```
REGEXP_REPLACE( source_string, pattern [, replace_string [ , position [, parameters
 ] ] ] )
```

인수

source_string

검색할 열 이름과 같은 CHAR 또는 VARCHAR 문자열 표현식입니다.

패턴

정규 표현식 패턴을 나타내는 UTF-8 문자열 리터럴입니다. 자세한 내용은 POSIX 연산자 단원을 참조하십시오.

replace_string

(선택) 발견되는 패턴을 각각 변경할 CHAR 또는 VARCHAR 문자열 표현식(열 이름 등)입니다. 기본값은 빈 문자열입니다("").

position

(선택) source_string 내에서 검색을 시작할 위치를 나타내는 양의 정수입니다. 이 위치는 바이트가 아닌 문자 수를 기준으로 하기 때문에 멀티바이트 문자도 단일 문자로 계산됩니다. 기본값은 1입니다. position이 1보다 작으면 검색이 source_string의 첫 문자부터 시작됩니다. position이 source_string의 문자 수보다 크면 결과는 source_string이 됩니다.

parameters

(선택) 함수가 패턴과 일치하는 방법을 나타내는 하나 이상의 문자열 리터럴입니다. 가능한 값은 다음과 같습니다.

- c - 대/소문자를 구분하여 일치시킵니다. 기본값은 대/소문자 구분 일치를 사용하는 것입니다.
- i - 대/소문자를 구분하지 않고 일치시킵니다.
- p – PCRE(Perl Compatible Regular Expression) 방언으로 패턴을 해석합니다. PCRE에 관한 자세한 내용은 Wikipedia의 펄 호환 정규 표현식을 참조하세요.

반환 타입

VARCHAR

pattern 또는 replace_string이 NULL이면 결과도 NULL이 됩니다.

예시

대소문자를 구분하지 않는 일치를 사용하여 값 quick brown fox 내에서 문자열 FOX를 모두 바꾸려면 다음 예제를 사용합니다.

```
SELECT REGEXP_REPLACE('the fox', 'FOX', 'quick brown fox', 1, 'i');

+---------------------+
|   regexp_replace    |
+---------------------+
| the quick brown fox |
+---------------------+
```

다음 예에서는 PCRE 방언으로 작성된 패턴을 사용하여 하나 이상의 숫자와 하나의 소문자가 포함된 단어를 찾습니다. PCRE에서 특정 미리 보기 의미가 있는 ?= 연산자가 사용됩니다. 해당 단어가 나타날 때마다 값 [hidden]으로 바꾸려면 다음 예제를 사용합니다.

```
SELECT REGEXP_REPLACE('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'p');

+--------------------------------+
|        regexp_replace          |
+--------------------------------+
| [hidden] plain A1234 [hidden] |
+--------------------------------+
```

다음 예에서는 PCRE 방언으로 작성된 패턴을 사용하여 하나 이상의 숫자와 하나의 소문자가 포함된 단어를 찾습니다. PCRE에서 특정 미리 보기 의미가 있는 ?= 연산자가 사용됩니다. 대소문자를 구분하지 않는 일치를 사용한다는 점에서 이전 예제와 다르지만 이러한 단어의 각 발생을 [hidden] 값으로 바꾸려면 다음 예제를 사용합니다.

```
SELECT REGEXP_REPLACE('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'ip');

+---------------------------------+
```

```
|          regexp_replace          |
+----------------------------------+
| [hidden] plain [hidden] [hidden] |
+----------------------------------+
```

다음 예제에서는 TICKIT 샘플 데이터베이스의 USERS 테이블을 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

이메일 주소에서 @ 및 도메인 주소를 삭제하려면 다음 예제를 사용합니다.

```
SELECT email, REGEXP_REPLACE(email, '@.*\\.(org|gov|com|edu|ca)$')
FROM users
ORDER BY userid LIMIT 4;

+------------------------------------------------+----------------------+
|                     email                      |    regexp_replace    |
+------------------------------------------------+----------------------+
| Etiam.laoreet.libero@sodalesMaurisblandit.edu | Etiam.laoreet.libero  |
| Suspendisse.tristique@nonnisiAenean.edu       | Suspendisse.tristique |
| amet.faucibus.ut@condimentumegetvolutpat.ca   | amet.faucibus.ut      |
| sed@lacusUtnec.ca                             | sed                   |
+------------------------------------------------+----------------------+
```

이메일 주소의 도메인 이름을 값 `internal.company.com`으로 바꾸려면 다음 예제를 사용합니다.

```
SELECT email, REGEXP_REPLACE(email, '@.*\\.[[:alpha:]]{2,3}','@internal.company.com')
FROM users
ORDER BY userid LIMIT 4;

+-----------------------------------------------
+-----------------------------------------+
|                     email                      |               regexp_replace
      |
+-----------------------------------------------
+-----------------------------------------+
| Etiam.laoreet.libero@sodalesMaurisblandit.edu |
 Etiam.laoreet.libero@internal.company.com  |
| Suspendisse.tristique@nonnisiAenean.edu       |
 Suspendisse.tristique@internal.company.com |
| amet.faucibus.ut@condimentumegetvolutpat.ca   | amet.faucibus.ut@internal.company.com
      |
| sed@lacusUtnec.ca                             | sed@internal.company.com
      |
```

```
+-------------------------------------------
+-----------------------------------------+
```

REGEXP_SUBSTR 함수

문자열에서 정규 표현식 패턴을 검색하여 문자를 반환합니다. REGEXP_SUBSTR은 SUBSTRING 함수 함수와 비슷하지만 문자열에서 정규 표현식 패턴을 검색할 수 있습니다. 함수에서 정규 표현식이 문자열의 어떤 문자와도 일치하지 않는 경우 빈 문자열이 반환됩니다. 정규 표현식에 관한 자세한 내용은 POSIX 연산자 단원 및 Wikipedia의 정규 표현식을 참조하세요.

구문

```
REGEXP_SUBSTR( source_string, pattern [, position [, occurrence [, parameters ] ] ] )
```

인수

source_string

검색할 문자열 표현식입니다.

패턴

정규 표현식 패턴을 나타내는 UTF-8 문자열 리터럴입니다. 자세한 내용은 POSIX 연산자 단원을 참조하십시오.

position

source_string 내에서 검색을 시작할 위치를 나타내는 양의 정수입니다. 이 위치는 바이트가 아닌 문자 수를 기준으로 하기 때문에 멀티바이트 문자도 단일 문자로 계산됩니다. 기본 값은 1입니다. position이 1보다 작으면 검색이 source_string의 첫 문자부터 시작됩니다. position이 source_string의 문자 수보다 크면 결과는 빈 문자열("")이 됩니다.

발생

사용할 패턴 발생을 나타내는 양의 정수입니다. REGEXP_SUBSTR은 첫 번째 발생에서 1을 뺀 개수의 일치하는 항목을 건너뜁니다. 기본 값은 1입니다. 발생이 1보다 작거나 source_string에 있는 문자 수보다 클 경우 검색이 무시되고 결과가 NULL이 됩니다.

parameters

함수가 패턴과 일치하는 방법을 나타내는 하나 이상의 문자열 리터럴입니다. 가능한 값은 다음과 같습니다.

- c - 대/소문자를 구분하여 일치시킵니다. 기본값은 대/소문자 구분 일치를 사용하는 것입니다.

- i - 대/소문자를 구분하지 않고 일치시킵니다.
- e - 하위 표현식을 사용하여 하위 문자열을 추출합니다.

  패턴에 하위 표현식이 포함되어 있을 경우 REGEXP_SUBSTR은 패턴의 첫 번째 하위 표현식을 사용하여 하위 문자열과 일치시킵니다. 하위 표현식은 괄호로 묶인 패턴 내 표현식입니다. 예를 들어 `'This is a (\\w+)'` 패턴은 첫 번째 표현식과 뒤에 단어가 오는 `'This is a '` 문자열을 일치시킵니다. e 매개 변수가 있는 REGEXP_SUBSTR은 패턴을 반환하는 대신 하위 표현식 내의 문자열만 반환합니다.

  REGEXP_SUBSTR은 첫 번째 하위 표현식만 고려하며 추가 하위 표현식은 무시됩니다. 패턴에 하위 표현식이 없으면 REGEXP_SUBSTR이 'e' 파라미터를 무시합니다.
- p – PCRE(Perl Compatible Regular Expression) 방언으로 패턴을 해석합니다. PCRE에 관한 자세한 내용은 Wikipedia의 펄 호환 정규 표현식을 참조하세요.

반환 타입

VARCHAR

예시

다음은 @ 문자와 도메인 확장자 사이의 이메일 주소 구간을 반환하는 예입니다. 쿼리된 `users` 데이터는 Amazon Redshift 샘플 데이터에서 가져온 것입니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

```
SELECT email, regexp_substr(email,'@[^.]*')
FROM users
ORDER BY userid LIMIT 4;

                     email                      |      regexp_substr
------------------------------------------------+--------------------------
 Suspendisse.tristique@nonnisiAenean.edu        | @nonnisiAenean
 amet.faucibus.ut@condimentumegetvolutpat.ca    | @condimentumegetvolutpat
 sed@lacusUtnec.ca                              | @lacusUtnec
 Cum@accumsan.com                               | @accumsan
```

다음 예에서는 대/소문자를 구분하지 않는 일치를 사용하여 문자열 FOX의 첫 번째 발생에 해당하는 입력 부분을 반환합니다.

```
SELECT regexp_substr('the fox', 'FOX', 1, 1, 'i');
```

```
 regexp_substr
---------------
 fox
```

다음 예에서는 대/소문자를 구분하지 않는 일치를 사용하여 문자열 FOX의 두 번째 발생에 해당하는 입력 부분을 반환합니다. 두 번째 항목이 없기 때문에 결과값은 NULL(비어 있음)입니다.

```
SELECT regexp_substr('the fox', 'FOX', 1, 2, 'i');

 regexp_substr
---------------

```

다음 예제에서는 소문자로 시작하는 입력의 첫 번째 부분을 반환합니다. 이는 c 파라미터가 없는 동일한 SELECT 문과 기능적으로 동일합니다.

```
SELECT regexp_substr('THE SECRET CODE IS THE LOWERCASE PART OF 1931abc0EZ.', '[a-z]+',
 1, 1, 'c');

 regexp_substr
---------------
 abc
```

다음 예에서는 PCRE 방언으로 작성된 패턴을 사용하여 하나 이상의 숫자와 하나의 소문자가 포함된 단어를 찾습니다. PCRE에서 특정 미리 보기 의미가 있는 ?= 연산자가 사용됩니다. 이 예에서는 두 번째 단어에 해당하는 입력 부분을 반환합니다.

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'p');

 regexp_substr
---------------
 a1234
```

다음 예에서는 PCRE 방언으로 작성된 패턴을 사용하여 하나 이상의 숫자와 하나의 소문자가 포함된 단어를 찾습니다. PCRE에서 특정 미리 보기 의미가 있는 ?= 연산자가 사용됩니다. 이 예는 두 번째 단어에 해당하는 입력 부분을 반환하지만 대/소문자를 구분하지 않는 일치를 사용한다는 점에서 이전 예와 다릅니다.

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'ip');
```

```
 regexp_substr
---------------
 A1234
```

다음 예에서는 하위 표현식을 사용하여 'this is a (\\w+)' 패턴과 일치하는 두 번째 문자열을 찾습니다. 괄호를 친 하위 표현식을 반환합니다.

```
SELECT regexp_substr(
               'This is a cat, this is a dog. This is a mouse.',
               'this is a (\\w+)', 1, 2, 'ie');

 regexp_substr
---------------
 dog
```

REPEAT 함수

문자열을 지정한 횟수만큼 반복합니다. 입력 파라미터가 숫자라고 해도 REPEAT는 이를 문자열로 처리합니다.

REPLICATE 함수의 동의어입니다.

구문

```
REPEAT(string, integer)
```

인수

string

첫 번째 입력 파라미터는 반복할 문자열입니다.

integer

두 번째 파라미터는 문자열을 반복할 횟수를 나타내는 INTEGER입니다.

반환 타입

VARCHAR

예시

다음 예제에서는 TICKIT 샘플 데이터베이스의 CATEGORY 테이블에 있는 데이터를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

CATEGORY 테이블에서 CATID 열의 값을 세 번 반복하려면 다음 예제를 사용합니다.

```
SELECT catid, REPEAT(catid,3)
FROM category
ORDER BY 1,2;

+-------+--------+
| catid | repeat |
+-------+--------+
|     1 |    111 |
|     2 |    222 |
|     3 |    333 |
|     4 |    444 |
|     5 |    555 |
|     6 |    666 |
|     7 |    777 |
|     8 |    888 |
|     9 |    999 |
|    10 | 101010 |
|    11 | 111111 |
+-------+--------+
```

REPLACE 함수

기존 문자열에서 발견되는 모든 문자 집합을 다른 지정 문자로 변경합니다.

REPLACE는 TRANSLATE 함수 및 REGEXP_REPLACE 함수과 비슷합니다. 단, TRANSLATE는 단일 문자를 여러 차례 변경하고, REGEXP_REPLACE는 문자열에서 정규 표현식 패턴을 검색하는 반면 REPLACE는 전체 문자열 하나를 다른 문자열로 변경합니다.

구문

```
REPLACE(string, old_chars, new_chars)
```

인수

string

검색할 CHAR 또는 VARCHAR 문자열입니다.

old_chars

대체할 CHAR 또는 VARCHAR 문자열입니다.

new_chars

old_string을 변경할 새로운 CHAR 또는 VARCHAR 문자열입니다.

반환 타입

VARCHAR

old_chars 또는 new_chars가 NULL이면 결과도 NULL이 됩니다.

예시

다음 예제에서는 TICKIT 샘플 데이터베이스의 CATEGORY 테이블에 있는 데이터를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

CATGROUP 필드에서 문자열 Shows를 Theatre로 변환하려면 다음 예제를 사용합니다.

```
SELECT catid, catgroup, REPLACE(catgroup, 'Shows', 'Theatre')
FROM category
ORDER BY 1,2,3;

+-------+----------+----------+
| catid | catgroup | replace  |
+-------+----------+----------+
|     1 | Sports   | Sports   |
|     2 | Sports   | Sports   |
|     3 | Sports   | Sports   |
|     4 | Sports   | Sports   |
|     5 | Sports   | Sports   |
|     6 | Shows    | Theatre  |
|     7 | Shows    | Theatre  |
|     8 | Shows    | Theatre  |
|     9 | Concerts | Concerts |
```

```
|    10 | Concerts | Concerts |
|    11 | Concerts | Concerts |
+-------+----------+----------+
```

REPLICATE 함수

REPEAT 함수의 동의어입니다.

REPEAT 함수 섹션을 참조하세요.

REVERSE 함수

REVERSE 함수는 문자열에 대해 실행되며, 문자를 역순으로 반환합니다. 예를 들어, `reverse('abcde')`는 edcba를 반환합니다. 이 함수는 문자 데이터 형식 외에 숫자나 날짜 데이터 형식에서도 실행되지만 대부분은 문자열에서 실용적인 값을 갖습니다.

구문

```
REVERSE( expression )
```

인수

expression

문자, 날짜, 타임스탬프, 숫자 데이터 형식 등 문자 반전의 대상이 되는 표현식입니다. 모든 표현식은 묵시적으로 VARCHAR 문자열로 변환됩니다. CHAR 문자열의 후행 공백 입력은 무시됩니다.

반환 타입

VARCHAR

예시

다음 예제에서는 TICKIT 샘플 데이터베이스의 USERS 및 SALES 테이블 데이터를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

USERS 테이블에서 5개의 고유한 도시 이름과 그에 해당하는 반전된 이름을 선택하려면 다음 예제를 사용합니다.

```
SELECT DISTINCT city AS cityname, REVERSE(cityname)
FROM users
ORDER BY city LIMIT 5;
```

```
+----------+----------+
| cityname | reverse  |
+----------+----------+
| Aberdeen | needrebA |
| Abilene  | enelibA  |
| Ada      | adA      |
| Agat     | tagA     |
| Agawam   | mawagA   |
+----------+----------+
```

문자 문자열로 캐스팅된 5개의 판매 ID와 해당 반전된 ID를 선택하려면 다음 예제를 사용합니다.

```
SELECT salesid, REVERSE(salesid)
FROM sales
ORDER BY salesid DESC LIMIT 5;

+---------+---------+
| salesid | reverse |
+---------+---------+
|  172456 |  654271 |
|  172455 |  554271 |
|  172454 |  454271 |
|  172453 |  354271 |
|  172452 |  254271 |
+---------+---------+
```

RTRIM 함수

RTRIM 함수는 문자열 끝부터 지정된 문자 집합을 잘라냅니다. 잘라낸 문자 목록에서 문자만 포함하는 가장 긴 문자열을 제거합니다. 잘라내기 문자가 입력 문자열에 나타나지 않으면 잘라내기가 완료된 것입니다.

구문

```
RTRIM( string, trim_chars )
```

인수

string

잘라낼 문자열 열, 표현식 또는 문자열 리터럴입니다.

trim_chars

문자열의 끝부터 잘라낼 문자를 나타내는 문자열 열, 표현식 또는 문자열 리터럴입니다. 지정하지 않으면 공백이 잘라내기 문자로 사용됩니다.

반환 타입

string 인수와 동일한 데이터 형식의 문자열입니다.

예제

다음은 문자열 ' abc '에서 선행 및 후행 공백을 잘라내는 예입니다.

```
select '     abc    ' as untrim, rtrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    |    abc
```

다음은 문자열 'xyzaxyzbxyzcxyz'에서 후행 'xyz' 문자열을 제거하는 예입니다. 결과를 보면 후행하는 'xyz'만 제거되었고 문자열 내부에서는 제거되지 않았습니다.

```
select 'xyzaxyzbxyzcxyz' as untrim,
rtrim('xyzaxyzbxyzcxyz', 'xyz') as trim;

     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | xyzaxyzbxyzc
```

다음 예제에서는 trim_chars 목록의 모든 문자와 일치하는 문자열 'setuphistorycassettes'에서 후행 부분을 제거합니다. 입력 문자열의 끝에서 trim_chars 목록에 없는 다른 문자 앞에 오는 모든 t, e 또는 s는 제거됩니다.

```
SELECT rtrim('setuphistorycassettes', 'tes');

     rtrim
-----------------
 setuphistoryca
```

다음은 VENUENAME의 끝에서 있는 경우에 한해 문자 'Park'를 잘라내는 예입니다.

```
select venueid, venuename, rtrim(venuename, 'Park')
from venue
order by 1, 2, 3
limit 10;

venueid |         venuename          |          rtrim
--------+----------------------------+-------------------------
      1 | Toyota Park                | Toyota
      2 | Columbus Crew Stadium      | Columbus Crew Stadium
      3 | RFK Stadium                | RFK Stadium
      4 | CommunityAmerica Ballpark  | CommunityAmerica Ballp
      5 | Gillette Stadium           | Gillette Stadium
      6 | New York Giants Stadium    | New York Giants Stadium
      7 | BMO Field                  | BMO Field
      8 | The Home Depot Center      | The Home Depot Cente
      9 | Dick's Sporting Goods Park | Dick's Sporting Goods
     10 | Pizza Hut Park             | Pizza Hut
```

위 예를 보면 P, a, r 또는 k가 VENUENAME의 끝에 있을 경우 RTRIM이 각 문자를 모두 제거한 것을 알 수 있습니다.

SOUNDEX 함수

SOUNDEX 함수는 입력 문자열의 첫 글자와 지정한 문자열의 영어 발음을 나타내는 소리의 3자리 인코딩으로 구성된 미국 Soundex 값을 반환합니다. 예를 들어 Smith 및 Smyth의 Soundex 코드는 동일합니다.

구문

```
SOUNDEX(string)
```

인수

string

American Soundex 코드 값으로 변환하려는 CHAR 또는 VARCHAR 문자열을 지정합니다.

반환 타입

VARCHAR(4)

사용 노트

SOUNDEX 함수는 a~z 및 A~Z를 포함하여 영어 알파벳 소문자와 대문자 ASCII 문자만 변환합니다. SOUNDEX는 다른 문자를 무시합니다. SOUNDEX는 공백으로 구분된 여러 단어의 문자열에 대해 단일 Soundex 값을 반환합니다.

```
SELECT SOUNDEX('AWS Amazon');

+---------+
| soundex |
+---------+
| A252    |
+---------+
```

SOUNDEX는 입력 문자열에 영어가 포함되지 않은 경우 빈 문자열을 반환합니다.

```
SELECT SOUNDEX('+-*/%');

+---------+
| soundex |
+---------+
|         |
+---------+
```

예시

Amazon에 대한 Soundex 값을 반환하려면 다음 예제를 사용합니다.

```
SELECT SOUNDEX('Amazon');

+---------+
| soundex |
+---------+
| A525    |
+---------+
```

smith 및 smyth에 대한 Soundex 값을 반환하려면 다음 예제를 사용합니다. Soundex 값은 동일합니다.

```
SELECT SOUNDEX('smith'), SOUNDEX('smyth');
```

```
+-------+-------+
| smith | smyth |
+-------+-------+
| S530  | S530  |
+-------+-------+
```

SPLIT_PART 함수

지정 구분자를 기준으로 문자열을 분할한 후 지정된 위치에 해당하는 부분을 반환합니다.

구문

```
SPLIT_PART(string, delimiter, position)
```

인수

string

분할할 문자열 열, 표현식 또는 문자열 리터럴입니다. 문자열은 CHAR 또는 VARCHAR가 될 수 있습니다.

delimiter

입력 문자열의 섹션을 나타내는 구분자 문자열입니다.

delimiter가 리터럴이면 작은따옴표로 묶어야 합니다.

position

반환할 문자열 구간의 위치입니다(1부터 시작). 반드시 0보다 큰 정수이어야 합니다. position이 문자열 구간의 수보다 크면 SPLIT_PART가 빈 문자열을 반환합니다. string에서 delimiter를 찾을 수 없는 경우 반환된 값에는 전체 string 또는 빈 값이 될 수 있는 지정된 부분의 내용이 포함됩니다.

반환 타입

string 파라미터와 동일한 CHAR 또는 VARCHAR 문자열입니다.

예시

다음 예제에서는 $ 구분 기호를 사용하여 문자열 리터럴을 여러 부분으로 분할하고 두 번째 부분을 반환합니다.

```
select split_part('abc$def$ghi','$',2)

split_part
----------
def
```

다음 예제에서는 $ 구분 기호를 사용하여 문자열 리터럴을 여러 부분으로 분할합니다. 4 부분을 찾을 수 없기 때문에 빈 문자열이 반환됩니다.

```
select split_part('abc$def$ghi','$',4)

split_part
----------


```

다음 예제에서는 # 구분 기호를 사용하여 문자열 리터럴을 여러 부분으로 분할합니다. 이 구분 기호를 찾을 수 없기 때문에 첫 번째 부분인 전체 문자열이 반환됩니다.

```
select split_part('abc$def$ghi','#',1)

split_part
------------
abc$def$ghi
```

다음은 타임스탬프 필드인 LISTTIME을 년, 월, 일 요소로 분할하는 예입니다.

```
select listtime, split_part(listtime,'-',1) as year,
split_part(listtime,'-',2) as month,
split_part(split_part(listtime,'-',3),' ',1) as day
from listing limit 5;

      listtime       | year | month | day
---------------------+------+-------+------
 2008-03-05 12:25:29 | 2008 | 03    | 05
 2008-09-09 08:03:36 | 2008 | 09    | 09
 2008-09-26 05:43:12 | 2008 | 09    | 26
 2008-10-04 02:00:30 | 2008 | 10    | 04
 2008-01-06 08:33:11 | 2008 | 01    | 06
```

다음은 LISTTIME 타임스탬프 필드를 선택하고, '-' 문자를 기준으로 필드를 분할하여 월(LISTTIME 문자열의 두 번째 구간)을 가져온 다음 각 월의 항목 수를 계산하는 예입니다.

```
select split_part(listtime,'-',2) as month, count(*)
from listing
group by split_part(listtime,'-',2)
order by 1, 2;

 month | count
-------+-------
    01 | 18543
    02 | 16620
    03 | 17594
    04 | 16822
    05 | 17618
    06 | 17158
    07 | 17626
    08 | 17881
    09 | 17378
    10 | 17756
    11 | 12912
    12 | 4589
```

STRPOS 함수

지정한 문자열 내에서 하위 문자열의 위치를 반환합니다.

유사한 함수는 CHARINDEX 함수 및 POSITION 함수 섹션을 참조하세요.

구문

```
STRPOS(string, substring )
```

인수

string

첫 번째 입력 파라미터는 검색 대상인 CHAR 또는 VARCHAR 문자열입니다.

substring

두 번째 파라미터는 string 내에서 검색할 하위 문자열입니다.

반환 타입

INTEGER

STRPOS 함수는 하위 문자열의 위치에 해당하는 INTEGER를 반환합니다(0이 아닌 1부터 시작). 이 위치는 바이트가 아닌 문자 수를 기준으로 하기 때문에 멀티바이트 문자도 단일 문자로 계산됩니다.

사용 노트

문자열 내에서 하위 문자열을 찾을 수 없는 경우 STRPOS는 0을 반환합니다.

```
SELECT STRPOS('dogfish', 'fist');

+--------+
| strpos |
+--------+
|      0 |
+--------+
```

예시

dogfish 내에서 fish의 위치를 표시하려면 다음 예제를 사용합니다.

```
SELECT STRPOS('dogfish', 'fish');

+--------+
| strpos |
+--------+
|      4 |
+--------+
```

다음 예제에서는 TICKIT 샘플 데이터베이스의 SALES 테이블 데이터를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

SALES 테이블에서 COMMISSION이 999.00을 초과하는 판매 거래 수를 반환하려면 다음 예제를 사용합니다.

```
SELECT DISTINCT STRPOS(commission, '.'),
COUNT (STRPOS(commission, '.'))
FROM sales
```

```
WHERE STRPOS(commission, '.') > 4
GROUP BY STRPOS(commission, '.')
ORDER BY 1, 2;

+--------+-------+
| strpos | count |
+--------+-------+
|      5 |   629 |
+--------+-------+
```

STRTOL 함수

지정한 밑의 문자열 표현식을 등가의 정수 값으로 변환합니다. 변환된 값은 부호화 64비트 범위 이내가 되어야 합니다.

구문

```
STRTOL(num_string, base)
```

인수

num_string

변환할 숫자의 문자열 표현식입니다. num_string이 빈 문자열(' ')이거나, 혹은 null 문자('\0')로 시작되면 변환되는 값은 0입니다. num_string이 NULL 값을 포함한 열이면 STRTOL이 NULL을 반환합니다. 문자열은 크기의 제한 없이 공백으로 시작할 수 있으며, 그 뒤에 더하기 '+' 또는 빼기 '-' 기호를 옵션으로 추가하여 양수 또는 음수를 나타냅니다. 기본값은 '+'입니다. 예를 들어 base가 16이라고 가정하면 문자열은 '0x'로 시작할 수 있습니다.

base

2에서 36 사이의 INTEGER입니다.

반환 타입

BIGINT

num_string이 null이면 함수는 NULL을 반환합니다.

예시

문자열과 기본값 쌍을 정수로 변환하려면 다음 예제를 사용합니다.

```
SELECT STRTOL('0xf',16);

+--------+
| strtol |
+--------+
|     15 |
+--------+

SELECT STRTOL('abcd1234',16);

+------------+
|   strtol   |
+------------+
| 2882343476 |
+------------+

SELECT STRTOL('1234567', 10);

+---------+
| strtol  |
+---------+
| 1234567 |
+---------+

SELECT STRTOL('1234567', 8);

+--------+
| strtol |
+--------+
| 342391 |
+--------+

SELECT STRTOL('110101', 2);

+--------+
| strtol |
+--------+
|     53 |
+--------+

SELECT STRTOL('\0', 2);

+--------+
```

```
| strtol |
+--------+
|      0 |
+--------+
```

SUBSTRING 함수

지정된 시작 위치를 기반으로 문자열의 하위 집합을 반환합니다.

입력이 문자열인 경우 추출된 문자의 시작 위치와 수는 바이트가 아닌 문자를 기준으로 하므로 멀티바이트 문자는 단일 문자로 계산됩니다. 입력이 이진 표현식인 경우 시작 위치와 추출된 하위 문자열은 바이트를 기반으로 합니다. 음의 길이는 지정할 수 없지만 음의 시작 위치는 지정 가능합니다.

구문

```
SUBSTRING(character_string FROM start_position [ FOR number_characters ] )
```

```
SUBSTRING(character_string, start_position, number_characters )
```

```
SUBSTRING(binary_expression, start_byte, number_bytes )
```

```
SUBSTRING(binary_expression, start_byte )
```

인수

character_string

검색 대상의 문자열입니다. 문자가 아닌 데이터 형식도 문자열로 처리됩니다.

start_position

문자열 내에서 추출을 시작할 위치이며, 1부터 시작됩니다. start_position은 바이트가 아닌 문자 수를 기준으로 하기 때문에 멀티바이트 문자도 단일 문자로 계산됩니다. 이 수는 음의 값이 될 수 있습니다.

number_characters

추출할 문자 수(하위 문자열의 길이)입니다. number_characters는 바이트가 아닌 문자 수를 기준으로 하기 때문에 멀티바이트 문자도 단일 문자로 계산됩니다. 이 수는 음의 값이 될 수 없습니다.

binary_expression

검색할 데이터 유형 VARBYTE의 binary_expression입니다.

start_byte

1에서 시작하여 추출을 시작할 이진 표현식 내의 위치입니다. 이 수는 음의 값이 될 수 있습니다.

number_bytes

추출할 바이트 수, 즉 하위 문자열의 길이입니다. 이 수는 음의 값이 될 수 없습니다.

반환 타입

입력에 따라 VARCHAR 또는 VARBYTE입니다.

사용 관련 참고 사항

다음은 start_position 및 number_characters를 사용하여 문자열의 다양한 위치에서 하위 문자열을 추출하는 방법에 대한 몇 가지 예입니다.

다음은 6번째 문자부터 4자의 문자열을 반환하는 예입니다.

```
select substring('caterpillar',6,4);
substring
-----------
pill
(1 row)
```

start_position + number_characters가 string의 길이를 초과하면, SUBSTRING이 start_position부터 문자열 끝까지 하위 문자열을 반환합니다. 예:

```
select substring('caterpillar',6,8);
substring
-----------
pillar
(1 row)
```

`start_position`이 0 또는 음수인 경우에는 SUBSTRING 함수가 문자열의 첫 번째 문자부터 `start_position` + `number_characters` -1의 길이를 갖는 하위 문자열을 반환합니다. 예:

```
select substring('caterpillar',-2,6);
substring
```

```
-----------
cat
(1 row)
```

start_position + number_characters -1이 0보다 작거나 같으면 SUBSTRING이 빈 문자열을 반환합니다. 예:

```
select substring('caterpillar',-5,4);
substring
-----------

(1 row)
```

예시

다음은 LISTING 테이블의 LISTTIME 문자열에서 월을 반환하는 예입니다.

```
select listid, listtime,
substring(listtime, 6, 2) as month
from listing
order by 1, 2, 3
limit 10;

 listid |      listtime       | month
--------+---------------------+-------
      1 | 2008-01-24 06:43:29 | 01
      2 | 2008-03-05 12:25:29 | 03
      3 | 2008-11-01 07:35:33 | 11
      4 | 2008-05-24 01:18:37 | 05
      5 | 2008-05-17 02:29:11 | 05
      6 | 2008-08-15 02:08:13 | 08
      7 | 2008-11-15 09:38:15 | 11
      8 | 2008-11-09 05:07:30 | 11
      9 | 2008-09-09 08:03:36 | 09
     10 | 2008-06-17 09:44:54 | 06
(10 rows)
```

다음은 위와 동일하지만 FROM...FOR 옵션을 사용하는 예입니다.

```
select listid, listtime,
substring(listtime from 6 for 2) as month
from listing
```

```
order by 1, 2, 3
limit 10;

 listid |      listtime       | month
--------+---------------------+-------
      1 | 2008-01-24 06:43:29 | 01
      2 | 2008-03-05 12:25:29 | 03
      3 | 2008-11-01 07:35:33 | 11
      4 | 2008-05-24 01:18:37 | 05
      5 | 2008-05-17 02:29:11 | 05
      6 | 2008-08-15 02:08:13 | 08
      7 | 2008-11-15 09:38:15 | 11
      8 | 2008-11-09 05:07:30 | 11
      9 | 2008-09-09 08:03:36 | 09
     10 | 2008-06-17 09:44:54 | 06
(10 rows)
```

멀티바이트 문자가 포함되었을 수도 있는 문자열에서는 접두사를 예측적으로 추출할 때 SUBSTRING 함수를 사용할 수 없습니다. 그 이유는 문자 수가 아닌 바이트 수를 기준으로 멀티바이트 문자열의 길이를 지정해야 하기 때문입니다. 바이트 길이를 기준으로 문자열의 시작 세그먼트를 추출하려면 문자열을 VARCHAR(byte_length)로 변환하여 절사할 수 있습니다. 여기에서 byte_length는 반드시 필요한 길이입니다. 다음은 문자열 'Fourscore and seven'에서 첫 5바이트를 추출하는 예입니다.

```
select cast('Fourscore and seven' as varchar(5));

varchar
-------
Fours
```

다음 예에서는 이진 값 abc의 음수 시작 위치를 보여줍니다. 시작 위치가 -3이므로 이진 값의 시작 부분에서 하위 문자열이 추출됩니다. 결과는 이진 하위 문자열의 16진수 표현으로 자동으로 표시됩니다.

```
select substring('abc'::varbyte, -3);

 substring
-----------
 616263
```

다음 예에서는 이진 값 abc의 시작 위치에 대해 1을 보여줍니다. 길이가 지정되지 않았기 때문에 문자열은 문자열의 시작 위치에서 끝까지 추출됩니다. 결과는 이진 하위 문자열의 16진수 표현으로 자동으로 표시됩니다.

```
select substring('abc'::varbyte, 1);

 substring
-----------
 616263
```

다음 예에서는 이진 값 abc의 시작 위치에 대해 3을 보여줍니다. 길이가 지정되지 않았기 때문에 문자열은 문자열의 시작 위치에서 끝까지 추출됩니다. 결과는 이진 하위 문자열의 16진수 표현으로 자동으로 표시됩니다.

```
select substring('abc'::varbyte, 3);

 substring
-----------
 63
```

다음 예에서는 이진 값 abc의 시작 위치에 대해 2를 보여줍니다. 문자열은 시작 위치에서 위치 10까지 추출되지만 문자열의 끝은 위치 3에 있습니다. 결과는 이진 하위 문자열의 16진수 표현으로 자동으로 표시됩니다.

```
select substring('abc'::varbyte, 2, 10);

 substring
-----------
 6263
```

다음 예에서는 이진 값 abc의 시작 위치에 대해 2를 보여줍니다. 문자열은 1바이트의 시작 위치에서 추출됩니다. 결과는 이진 하위 문자열의 16진수 표현으로 자동으로 표시됩니다.

```
select substring('abc'::varbyte, 2, 1);

 substring
-----------
 62
```

다음 예에서는 입력 문자열 Silva, Ana의 마지막 공백 뒤에 나타나는 첫 번째 이름 Ana를 반환합니다.

```
select reverse(substring(reverse('Silva, Ana'), 1, position(' ' IN reverse('Silva,
 Ana'))))
```

```
 reverse
-----------
 Ana
```

TEXTLEN 함수

LEN 함수의 동의어입니다.

LEN 함수 섹션을 참조하세요.

TRANSLATE 함수

임의의 표현식에서 발견되는 모든 지정 문자를 지정한 대체 문자로 변경합니다. 기존 문자는 characters_to_replace의 문자 위치와 characters_to_substitute 인수에 따라 변환 문자로 매핑됩니다. characters_to_replace 인수에서 지정하는 문자 수가 characters_to_substitute 인수에서 지정하는 문자 수보다 많으면 characters_to_replace 인수의 추가 문자가 반환 값에서 생략됩니다.

TRANSLATE는 REPLACE 함수 및 REGEXP_REPLACE 함수과 비슷합니다. 단, REPLACE는 전체 문자열 하나를 다른 문자열로 변경하고, REGEXP_REPLACE는 문자열에서 정규 표현식 패턴을 검색하는 반면 TRANSLATE는 단일 문자를 여러 차례 변경합니다.

인수가 NULL이면 반환되는 값도 NULL입니다.

구문

```
TRANSLATE( expression, characters_to_replace, characters_to_substitute )
```

인수

expression

변환 대상인 표현식입니다.

characters_to_replace

변경 대상인 문자가 포함된 문자열입니다.

characters_to_substitute

대체할 문자가 포함된 문자열입니다.

반환 타입

VARCHAR

예시

문자열에서 여러 문자를 바꾸려면 다음 예제를 사용합니다.

```
SELECT TRANSLATE('mint tea', 'inea', 'osin');

+-----------+
| translate |
+-----------+
| most tin  |
+-----------+
```

다음 예제에서는 TICKIT 샘플 데이터베이스의 USERS 테이블을 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

열의 모든 값에 대해 at 기호(@)를 마침표로 바꾸려면 다음 예제를 사용합니다.

```
SELECT email, TRANSLATE(email, '@', '.') as obfuscated_email
FROM users LIMIT 10;

+----------------------------------------+----------------------------------------+
|                 email                  |           obfuscated_email             |
+----------------------------------------+----------------------------------------+
| Cum@accumsan.com                       | Cum.accumsan.com                       |
| lorem.ipsum@Vestibulumante.com         | lorem.ipsum.Vestibulumante.com         |
| non.justo.Proin@ametconsectetuer.edu   | non.justo.Proin.ametconsectetuer.edu   |
| non.ante.bibendum@porttitortellus.org  | non.ante.bibendum.porttitortellus.org  |
| eros@blanditatnisi.org                 | eros.blanditatnisi.org                 |
| augue@Donec.ca                         | augue.Donec.ca                         |
| cursus@pedeacurna.edu                  | cursus.pedeacurna.edu                  |
| at@Duis.com                            | at.Duis.com                            |
| quam@facilisisvitaeorci.ca             | quam.facilisisvitaeorci.ca             |
| mi.lorem@nunc.edu                      | mi.lorem.nunc.edu                      |
+----------------------------------------+----------------------------------------+
```

열의 모든 값에 대해 공백을 밑줄로 바꾸고 마침표를 제거하려면 다음 예제를 사용합니다.

```
SELECT city, TRANSLATE(city, ' .', '_')
FROM users
```

```
WHERE city LIKE 'Sain%' OR city LIKE 'St%'
GROUP BY city
ORDER BY city;

+----------------+---------------+
|      city      |   translate   |
+----------------+---------------+
| Saint Albans   | Saint_Albans  |
| Saint Cloud    | Saint_Cloud   |
| Saint Joseph   | Saint_Joseph  |
| Saint Louis    | Saint_Louis   |
| Saint Paul     | Saint_Paul    |
| St. George     | St_George     |
| St. Marys      | St_Marys      |
| St. Petersburg | St_Petersburg |
| Stafford       | Stafford      |
| Stamford       | Stamford      |
| Stanton        | Stanton       |
| Starkville     | Starkville    |
| Statesboro     | Statesboro    |
| Staunton       | Staunton      |
| Steubenville   | Steubenville  |
| Stevens Point  | Stevens_Point |
| Stillwater     | Stillwater    |
| Stockton       | Stockton      |
| Sturgis        | Sturgis       |
+----------------+---------------+
```

TRIM 함수

공백 또는 지정된 문자로 문자열을 자릅니다.

구문

```
TRIM( [ BOTH | LEADING | TRAILING ] [trim_chars FROM ] string )
```

인수

BOTH | LEADING | TRAILING

(선택 사항) 문자를 잘라낼 위치를 지정합니다. 선행 및 후행 문자를 모두 제거하려면 BOTH를 사용하고, 선행 문자만 제거하려면 LEADING을 사용하며, 후행 문자만 제거하려면 TRAILING을 사용합니다. 이 파라미터를 생략하면 선행 및 후행 문자가 모두 잘립니다.

trim_chars

(옵션) 문자열에서 잘라낼 문자입니다. 이 파라미터를 생략하면 공백이 잘립니다.

string

자르기 대상이 되는 문자열입니다.

반환 타입

TRIM 함수는 VARCHAR 또는 CHAR 문자열을 반환합니다. TRIM 함수를 SQL 명령과 함께 사용하면 Amazon Redshift가 함수 결과를 묵시적으로 VARCHAR로 변환합니다. 하지만 SQL 함수의 SELECT 목록에서 TRIM 함수를 사용할 경우에는 Amazon Redshift가 함수 결과를 묵시적으로 변환하지 못하기 때문에 데이터 형식의 불일치 오류를 피하려면 명시적으로 변환해야만 합니다. 명시적 변환에 대한 자세한 내용은 CAST 함수 및 CONVERT 함수 함수 섹션을 참조하세요.

예시

문자열 dog 에서 선행 및 후행 공백을 잘라내려면 다음 예제를 사용합니다.

```
SELECT TRIM('    dog    ');

+-------+
| btrim |
+-------+
| dog   |
+-------+
```

문자열 dog 에서 선행 및 후행 공백을 잘라내려면 다음 예제를 사용합니다.

```
SELECT TRIM(BOTH FROM '    dog    ');

+-------+
| btrim |
+-------+
| dog   |
+-------+
```

문자열 "dog"에서 선행 큰따옴표를 제거하려면 다음 예제를 사용합니다.

```
SELECT TRIM(LEADING '"' FROM'"dog"');
```

```
+-------+
| ltrim |
+-------+
| dog"  |
+-------+
```

문자열 "dog"에서 후행 큰따옴표를 제거하려면 다음 예제를 사용합니다.

```
SELECT TRIM(TRAILING '"' FROM'"dog"');

+-------+
| rtrim |
+-------+
| "dog  |
+-------+
```

TRIM은 trim_chars의 문자가 string의 첫 문자 또는 끝 문자이면 모두 제거합니다. 다음 예제에서는 VARCHAR열인 VENUENAME의 시작 또는 끝에 나타나는 문자 'C', 'D' 및 'G'를 잘라냅니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

```
SELECT venueid, venuename, TRIM('CDG' FROM venuename)
FROM venue
WHERE venuename LIKE '%Park'
ORDER BY 2
LIMIT 7;

+---------+----------------------------+----------------------------+
| venueid |         venuename          |           btrim            |
+---------+----------------------------+----------------------------+
|     121 | AT&T Park                  | AT&T Park                  |
|     109 | Citizens Bank Park         | itizens Bank Park          |
|     102 | Comerica Park              | omerica Park               |
|       9 | Dick's Sporting Goods Park | ick's Sporting Goods Park  |
|      97 | Fenway Park                | Fenway Park                |
|     112 | Great American Ball Park   | reat American Ball Park    |
|     114 | Miller Park                | Miller Park                |
+---------+----------------------------+----------------------------+
```

UPPER 함수

문자열을 소문자로 변환합니다. UPPER는 UTF-8 멀티바이트 문자를 지원하여 문자당 최대 4바이트까지 가능합니다.

구문

```
UPPER(string)
```

인수

string

입력 파라미터는 VARCHAR 문자열 또는 VARCHAR로 암시적으로 변환될 수 있는 CHAR와 같은 기타 데이터 형식입니다.

반환 타입

UPPER 함수는 입력 문자열과 데이터 형식이 동일한 문자열을 반환합니다. 예를 들어 입력이 VARCHAR 문자열인 경우 함수는 VARCHAR 문자열을 반환합니다.

예시

다음 예제에서는 TICKIT 샘플 데이터베이스의 CATEGORY 테이블에 있는 데이터를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

CATNAME 필드를 대문자로 변환하려면 다음을 사용합니다.

```
SELECT catname, UPPER(catname)
FROM category
ORDER BY 1,2;

+-----------+-----------+
|  catname  |   upper   |
+-----------+-----------+
| Classical | CLASSICAL |
| Jazz      | JAZZ      |
| MLB       | MLB       |
| MLS       | MLS       |
| Musicals  | MUSICALS  |
| NBA       | NBA       |
| NFL       | NFL       |
| NHL       | NHL       |
| Opera     | OPERA     |
| Plays     | PLAYS     |
| Pop       | POP       |
```

```
+-----------+-----------+
```

SUPER 형식 정보 함수

다음에서 Amazon Redshift가 SUPER 데이터 형식의 입력에서 동적 정보를 파생하도록 지원하는 SQL용 형식 정보 함수에 대한 설명을 찾아볼 수 있습니다.

주제

DECIMAL_PRECISION 함수

저장할 최대 총 소수점 이하 자릿수의 전체 자릿수를 확인합니다. 이 숫자에는 소수점의 왼쪽 및 오른쪽 숫자가 모두 포함됩니다. 전체 자릿수 범위는 1~38이며 기본값은 38입니다.

구문

```
DECIMAL_PRECISION(super_expression)
```

인수

super_expression

SUPER 표현식 또는 열입니다.

반환 타입

INTEGER

예시

테이블 t에 DECIMAL_PRECISION 함수를 적용하려면 다음 예제를 사용합니다.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (3.14159);

SELECT DECIMAL_PRECISION(s) FROM t;

+-------------------+
| decimal_precision |
+-------------------+
|                 6 |
+-------------------+
```

DECIMAL_SCALE 함수

소수점 오른쪽에 저장할 소수점 이하 자릿수를 확인합니다. 소수 자릿수 범위는 0~전체 자릿수이며 기본값은 0입니다.

구문

```
DECIMAL_SCALE(super_expression)
```

인수

super_expression

SUPER 표현식 또는 열입니다.

반환 타입

INTEGER

예시

테이블 t에 DECIMAL_SCALE 함수를 적용하려면 다음 예제를 사용합니다.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (3.14159);

SELECT DECIMAL_SCALE(s) FROM t;

+---------------+
| decimal_scale |
+---------------+
|             5 |
+---------------+
```

IS_ARRAY 함수

변수가 배열인지 확인합니다. 변수가 배열인 경우 함수는 `true`를 반환합니다. 함수에 빈 배열도 포함됩니다. 그렇지 않으면 함수는 null을 포함한 다른 모든 값에 대해 `false`를 반환합니다.

구문

```
IS_ARRAY(super_expression)
```

인수

super_expression

SUPER 표현식 또는 열입니다.

반환 타입

BOOLEAN

예시

IS_ARRAY 함수를 사용하여 `[1,2]`가 배열인지 확인하려면 다음 예제를 사용합니다.

```
SELECT IS_ARRAY(JSON_PARSE('[1,2]'));

+----------+
| is_array |
+----------+
| true     |
+----------+
```

IS_BIGINT 함수

값이 BIGINT인지 여부를 확인합니다. IS_BIGINT 함수는 64비트 범위에서 소수 자릿수 0의 숫자에 대해 true를 반환합니다. 그렇지 않으면 함수는 null 및 부동 소수점 숫자를 포함한 다른 모든 값에 대해 false를 반환합니다.

IS_BIGINT 함수는 IS_INTEGER의 상위 집합입니다.

구문

```
IS_BIGINT(super_expression)
```

인수

super_expression

SUPER 표현식 또는 열입니다.

반환 타입

BOOLEAN

예시

IS_BIGINT 함수를 사용하여 5가 BIGINT인지 확인하려면 다음 예제를 사용합니다.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (5);

SELECT s, IS_BIGINT(s) FROM t;
```

```
+---+-----------+
| s | is_bigint |
+---+-----------+
| 5 | true      |
+---+-----------+
```

IS_BOOLEAN 함수

값이 BOOLEAN인지 여부를 확인합니다. IS_BOOLEAN 함수는 상수 JSON 부울에 대해 `true`를 반환합니다. 이 함수는 null을 포함한 다른 모든 값에 대해 `false`를 반환합니다.

구문

```
IS_BOOLEAN(super_expression)
```

인수

super_expression

SUPER 표현식 또는 열입니다.

반환 타입

BOOLEAN

예시

IS_BOOLEAN 함수를 사용하여 TRUE가 BOOLEAN인지 확인하려면 다음 예제를 사용합니다.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (TRUE);

SELECT s, IS_BOOLEAN(s) FROM t;

+------+------------+
|  s   | is_boolean |
+------+------------+
| true | true       |
+------+------------+
```

IS_CHAR 함수

값이 CHAR인지 여부를 확인합니다. IS_CHAR 함수는 ASCII 문자만 있는 문자열에 대해 `true`를 반환합니다. CHAR 형식은 ASCII 형식의 문자만 저장할 수 있기 때문입니다. 이 함수는 다른 모든 값에 대해 `false`를 반환합니다.

구문

```
IS_CHAR(super_expression)
```

인수

super_expression

SUPER 표현식 또는 열입니다.

반환 타입

BOOLEAN

예시

IS_CHAR 함수를 사용하여 t가 CHAR인지 확인하려면 다음 예제를 사용합니다.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES ('t');

SELECT s, IS_CHAR(s) FROM t;

+-----+---------+
|  s  | is_char |
+-----+---------+
| "t" | true    |
+-----+---------+
```

IS_DECIMAL 함수

값이 DECIMAL인지 여부를 확인합니다. IS_DECIMAL 함수는 부동 소수점이 아닌 숫자에 대해 `true`를 반환합니다. 이 함수는 null을 포함한 다른 모든 값에 대해 `false`를 반환합니다.

IS_DECIMAL 함수는 IS_BIGINT의 상위 집합입니다.

구문

```
IS_DECIMAL(super_expression)
```

인수

super_expression

SUPER 표현식 또는 열입니다.

반환 타입

BOOLEAN

예시

IS_DECIMAL 함수를 사용하여 1.22가 DECIMAL인지 확인하려면 다음 예제를 사용합니다.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (1.22);

SELECT s, IS_DECIMAL(s) FROM t;

+------+------------+
|  s   | is_decimal |
+------+------------+
| 1.22 | true       |
+------+------------+
```

IS_FLOAT 함수

값이 부동 소수점 숫자인지 확인합니다. IS_FLOAT 함수는 부동 소수점 숫자(FLOAT4 및 FLOAT8)에 대해 true를 반환합니다. 이 함수는 다른 모든 값에 대해 false를 반환합니다.

IS_DECIMAL 집합과 IS_FLOAT 집합은 분리되어 있습니다.

구문

```
IS_FLOAT(super_expression)
```

인수

super_expression

SUPER 표현식 또는 열입니다.

반환 타입

BOOLEAN

예시

IS_FLOAT 함수를 사용하여 2.22::FLOAT가 FLOAT인지 확인하려면 다음 예제를 사용합니다.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES(2.22::FLOAT);

SELECT s, IS_FLOAT(s) FROM t;

+---------+----------+
|    s    | is_float |
+---------+----------+
| 2.22e+0 | true     |
+---------+----------+
```

IS_INTEGER 함수

32비트 범위에서 소수 자릿수 0의 숫자에 대해 `true`를 반환하고 다른 모든 숫자(null 및 부동 소수점 숫자 포함)에 대해 `false`를 반환합니다.

IS_INTEGER 함수는 IS_SMALLINT 함수의 상위 집합입니다.

구문

```
IS_INTEGER(super_expression)
```

인수

super_expression

SUPER 표현식 또는 열입니다.

반환 타입

BOOLEAN

예시

IS_INTEGER 함수를 사용하여 5가 INTEGER인지 확인하려면 다음 예제를 사용합니다.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (5);

SELECT s, IS_INTEGER(s) FROM t;

+---+------------+
| s | is_integer |
+---+------------+
| 5 | true       |
+---+------------+
```

IS_OBJECT 함수

변수가 객체인지 확인합니다. IS_OBJECT 함수는 빈 객체를 포함한 객체에 대해 `true`를 반환합니다. 이 함수는 null을 포함한 다른 모든 값에 대해 `false`를 반환합니다.

구문

```
IS_OBJECT(super_expression)
```

인수

super_expression

SUPER 표현식 또는 열입니다.

반환 타입

BOOLEAN

예시

IS_OBJECT 함수를 사용하여 `{"name": "Joe"}`가 객체인지 확인하려면 다음 예제를 사용합니다.

```
CREATE TABLE t(s super);

INSERT INTO t VALUES (JSON_PARSE('{"name": "Joe"}'));

SELECT s, IS_OBJECT(s) FROM t;

+----------------+-----------+
|       s        | is_object |
+----------------+-----------+
| {"name":"Joe"} | true      |
+----------------+-----------+
```

IS_SCALAR 함수

변수가 스칼라인지 확인합니다. IS_SCALAR 함수는 배열이나 객체가 아닌 모든 값에 대해 `true`를 반환합니다. 이 함수는 null을 포함한 다른 모든 값에 대해 `false`를 반환합니다.

IS_ARRAY, IS_OBJECT 및 IS_SCALAR 집합은 null을 제외한 모든 값을 포함합니다.

구문

```
IS_SCALAR(super_expression)
```

인수

super_expression

SUPER 표현식 또는 열입니다.

반환 타입

BOOLEAN

예시

IS_SCALAR 함수를 사용하여 `{"name": "Joe"}`가 스칼라인지 확인하려면 다음 예제를 사용합니다.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (JSON_PARSE('{"name": "Joe"}'));
```

```
SELECT s, IS_SCALAR(s.name) FROM t;

+----------------+-----------+
|       s        | is_scalar |
+----------------+-----------+
| {"name":"Joe"} | true      |
+----------------+-----------+
```

IS_SMALLINT 함수

변수가 SMALLINT인지 확인합니다. IS_SMALLINT 함수는 16비트 범위에서 소수 자릿수 0의 숫자에 대해 `true`를 반환합니다. 함수는 null 및 부동 소수점 숫자를 포함한 다른 값에 대해 `false`를 반환합니다.

구문

```
IS_SMALLINT(super_expression)
```

인수

super_expression

SUPER 표현식 또는 열입니다.

반환

BOOLEAN

예시

IS_SMALLINT 함수를 사용하여 5가 SMALLINT인지 확인하려면 다음 예제를 사용합니다.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (5);

SELECT s, IS_SMALLINT(s) FROM t;

+---+-------------+
| s | is_smallint |
```

```
+---+-------------+
| 5 | true        |
+---+-------------+
```

IS_VARCHAR 함수

변수가 VARCHAR인지 확인합니다. IS_VARCHAR 함수는 모든 문자열에 대해 `true`를 반환합니다. 이 함수는 다른 모든 값에 대해 `false`를 반환합니다.

IS_VARCHAR 함수는 IS_CHAR 함수의 상위 집합입니다.

구문

```
IS_VARCHAR(super_expression)
```

인수

super_expression

SUPER 표현식 또는 열입니다.

반환 타입

BOOLEAN

예시

IS_VARCHAR 함수를 사용하여 abc가 VARCHAR인지 확인하려면 다음 예제를 사용합니다.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES ('abc');

SELECT s, IS_VARCHAR(s) FROM t;

+-------+------------+
|   s   | is_varchar |
+-------+------------+
| "abc" | true       |
+-------+------------+
```

JSON_SIZE 함수

JSON_SIZE 함수는 문자열로 직렬화되는 경우 지정된 SUPER 표현식의 바이트 수를 반환합니다.

구문

```
JSON_SIZE(super_expression)
```

인수

super_expression

SUPER 상수 또는 표현식입니다.

반환 타입

INTEGER

JSON_SIZE 함수는 입력 문자열의 바이트 수를 나타내는 INTEGER를 반환합니다. 이 값은 문자 수와 다릅니다. 예를 들어 검은색 점인 UTF-8 문자 #는 1자이긴 하지만 크기는 3바이트입니다.

사용 노트

JSON_SIZE(x)는 OCTET_LENGTH(JSON_SERIALIZE)와 기능적으로 동일합니다. 그러나 제공된 SUPER 표현식이 시스템이 직렬화될 때의 VARCHAR 제한을 초과할 경우 JSON_SERIALIZE에서 오류를 반환합니다. JSON_SIZE에는 이러한 제한이 없습니다.

예시

문자열로 직렬화된 SUPER 값의 길이를 반환하려면 다음 예제를 사용합니다.

```
SELECT JSON_SIZE(JSON_PARSE('[10001,10002,"#"]'));

+-----------+
| json_size |
+-----------+
|        19 |
+-----------+
```

제공된 SUPER 표현식은 17자이지만, #은 3바이트 문자이므로 JSON_SIZE는 19를 반환합니다.

JSON_TYPEOF 함수

JSON_TYPEOF 스칼라 함수는 SUPER 값의 동적 형식에 따라 부울, 숫자, 문자열, 객체, 배열 또는 null 값이 있는 VARCHAR를 반환합니다.

구문

```
JSON_TYPEOF(super_expression)
```

인수

super_expression

SUPER 표현식 또는 열입니다.

반환 타입

VARCHAR

예시

JSON_TYPEOF 함수를 사용하여 [1,2] 배열의 JSON 형식을 확인하려면 다음 예제를 사용합니다.

```
SELECT JSON_TYPEOF(ARRAY(1,2));

+-------------+
| json_typeof |
+-------------+
| array       |
+-------------+
```

JSON_TYPEOF 함수를 사용하여 {"name":"Joe"} 객체의 JSON 형식을 확인하려면 다음 예제를 사용합니다.

```
SELECT JSON_TYPEOF(JSON_PARSE('{"name":"Joe"}'));

+-------------+
| json_typeof |
+-------------+
| object      |
+-------------+
```

SIZE

SUPER 형식 상수 또는 표현식의 이진 메모리 내 크기를 INTEGER로 반환합니다.

구문

```
SIZE(super_expression)
```

인수

super_expression

SUPER 형식 상수 또는 표현식입니다.

반환 타입

INTEGER

예시

SIZE를 사용하여 여러 SUPER 형식 표현식의 메모리 내 크기를 가져오려면 다음 예제를 사용합니다.

```
CREATE TABLE test_super_size(a SUPER);

INSERT INTO test_super_size
VALUES
  (null),
  (TRUE),
  (JSON_PARSE('[0,1,2,3]')),
  (JSON_PARSE('{"a":0,"b":1,"c":2,"d":3}'))
;

SELECT a, SIZE(a)
FROM test_super_size
ORDER BY 2, 1;

+---------------------------+------+
|             a             | size |
+---------------------------+------+
| true                      |    4 |
| NULL                      |    4 |
| [0,1,2,3]                 |   23 |
| {"a":0,"b":1,"c":2,"d":3} |   52 |
```

```
+--------------------------+------+
```

VARBYTE 함수 및 연산자

VARBYTE 데이터 유형을 지원하는 Amazon Redshift 함수 및 연산자는 다음과 같습니다.

- VARBYTE 연산자
- FROM_HEX
- FROM_VARBYTE
- GETBIT
- TO_HEX
- TO_VARBYTE
- CONCAT
- LEN
- LENGTH 함수
- OCTET_LENGTH
- SUBSTRING 함수

VARBYTE 연산자

다음 표에는 VARBYTE 연산자가 나열되어 있습니다. 연산자는 데이터 유형 VARBYTE의 이진 값으로 작동합니다. 입력이 하나 또는 둘 다 null이면 결과는 null입니다.

지원되는 연산자

연산자	설명	반환 타입
<	보다 작음	BOOLEAN
<=	작거나 같음	BOOLEAN
=	같음	BOOLEAN
>	보다 큼	BOOLEAN
>=	크거나 같음	BOOLEAN

연산자	설명	반환 타입
!= 또는 <>	같지 않음	BOOLEAN
||	연결	VARBYTE
+	연결	VARBYTE
~	Bitwise not	VARBYTE
&	비트 논리곱	VARBYTE
|	Bitwise or	VARBYTE
#	Bitwise xor	VARBYTE

예시

다음 예에서 'a'::VARBYTE의 값은 61이고 'b'::VARBYTE의 값은 62입니다. ::는 문자열을 VARBYTE 데이터 형식으로 캐스팅합니다. 데이터 형식 캐스팅에 대한 자세한 내용은 CAST 단원을 참조하세요.

< 연산자를 사용하여 'a'가 'b'보다 작은지 비교하려면 다음 예제를 사용합니다.

```
SELECT 'a'::VARBYTE < 'b'::VARBYTE AS less_than;

+-----------+
| less_than |
+-----------+
| true      |
+-----------+
```

= 연산자를 사용하여 'a'가 'b'랑 같은지 비교하려면 다음 예제를 사용합니다.

```
SELECT 'a'::VARBYTE = 'b'::VARBYTE AS equal;

+-------+
| equal |
+-------+
| false |
```

```
+-------+
```

|| 연산자를 사용하여 두 이진 값을 연결하려면 다음 예제를 사용합니다.

```
SELECT 'a'::VARBYTE || 'b'::VARBYTE AS concat;

+--------+
| concat |
+--------+
|   6162 |
+--------+
```

+ 연산자를 사용하여 두 이진 값을 연결하려면 다음 예제를 사용합니다.

```
SELECT 'a'::VARBYTE + 'b'::VARBYTE AS concat;

+--------+
| concat |
+--------+
|   6162 |
+--------+
```

FROM_VARBYTE 함수를 사용하여 입력 이진 값의 각 비트를 무효화하려면 다음 예제를 사용합니다. 문자열''a'는 01100001로 평가됩니다. 자세한 내용은 FROM_VARBYTE 단원을 참조하십시오.

```
SELECT FROM_VARBYTE(~'a'::VARBYTE, 'binary');

+--------------+
| from_varbyte |
+--------------+
|     10011110 |
+--------------+
```

두 입력 이진 값에 & 연산자를 적용하려면 다음 예제를 사용합니다. 문자열 'a'는 01100001로 평가되고 'b'는 01100010로 평가됩니다.

```
SELECT FROM_VARBYTE('a'::VARBYTE & 'b'::VARBYTE, 'binary');

+--------------+
| from_varbyte |
+--------------+
```

```
|     01100000 |
+--------------+
```

FROM_HEX 함수

FROM_HEX는 16진수를 2진수 값으로 변환합니다.

구문

```
FROM_HEX(hex_string)
```

인수

hex_string

변환할 데이터 형식 VARCHAR 또는 TEXT의 16진수 문자열입니다. 형식은 리터럴 값이 되어야 합니다.

반환 타입

VARBYTE

예시

'6162'의 16진수 표현을 이진 값으로 변환하려면 다음 예제를 사용합니다. 결과는 이진 값의 16진수 표현으로 자동으로 표시됩니다.

```
SELECT FROM_HEX('6162');

+----------+
| from_hex |
+----------+
|     6162 |
+----------+
```

FROM_VARBYTE 함수

FROM_VARBYTE는 이진 값을 지정된 형식의 문자열로 변환합니다.

구문

```
FROM_VARBYTE(binary_value, format)
```

인수

binary_value

데이터 형식 VARBYTE의 이진 값입니다.

format

반환된 문자열의 형식입니다. 대/소문자를 구분하지 않는 유효한 값은 hex, binary, utf8(utf-8 및 utf_8도 가능함) 및 base64입니다.

반환 타입

VARCHAR

예시

이진 값 'ab'를 16진수로 변환하려면 다음 예제를 사용합니다.

```
SELECT FROM_VARBYTE('ab', 'hex');

+--------------+
| from_varbyte |
+--------------+
|         6162 |
+--------------+
```

'4d'의 이진 표현을 반환하려면 다음 예제를 사용합니다. '4d'의 이진 표현은 문자열 01001101입니다.

```
SELECT FROM_VARBYTE(FROM_HEX('4d'), 'binary');

+--------------+
| from_varbyte |
+--------------+
|     01001101 |
+--------------+
```

GETBIT 함수

GETBIT는 지정된 인덱스에서 이진 값의 비트 값을 반환합니다.

구문

```
GETBIT(binary_value, index)
```

인수

binary_value

데이터 형식 VARBYTE의 이진 값입니다.

인덱스를 구축하고 배포할 것입니다

반환되는 이진 값의 비트 인덱스 번호입니다. 이진 값은 맨 오른쪽 비트(최하위 비트)에서 맨 왼쪽 비트(최상위 비트)로 인덱싱되는 0부터 시작하는 비트 배열입니다.

반환 타입

INTEGER

예시

이진 값 from_hex('4d')의 인덱스 2에 있는 비트를 반환하려면 다음 예제를 사용합니다. '4d'의 이진 표현은 01001101입니다.

```
SELECT GETBIT(FROM_HEX('4d'), 2);

+--------+
| getbit |
+--------+
|      1 |
+--------+
```

from_hex('4d')가 반환하는 이진 값의 8개 인덱스 위치에서 비트를 반환하려면 다음 예제를 사용합니다. '4d'의 이진 표현은 01001101입니다.

```
SELECT GETBIT(FROM_HEX('4d'), 7), GETBIT(FROM_HEX('4d'), 6),
  GETBIT(FROM_HEX('4d'), 5), GETBIT(FROM_HEX('4d'), 4),
  GETBIT(FROM_HEX('4d'), 3), GETBIT(FROM_HEX('4d'), 2),
  GETBIT(FROM_HEX('4d'), 1), GETBIT(FROM_HEX('4d'), 0);

+--------+--------+--------+--------+--------+--------+--------+--------+
| getbit | getbit | getbit | getbit | getbit | getbit | getbit | getbit |
```

```
+--------+--------+--------+--------+--------+--------+--------+--------+
|      0 |      1 |      0 |      0 |      1 |      1 |      0 |      1 |
+--------+--------+--------+--------+--------+--------+--------+--------+
```

TO_HEX 함수

TO_HEX는 숫자 또는 이진 값을 16진수 표현으로 변환합니다.

구문

```
TO_HEX(value)
```

인수

USD 상당

변환할 숫자 또는 이진 값(VARBYTE)입니다.

반환 타입

VARCHAR

예시

숫자를 16진수 표현으로 변환하려면 다음 예제를 사용합니다.

```
SELECT TO_HEX(2147676847);

+----------+
|  to_hex  |
+----------+
| 8002f2af |
+----------+
```

'abc'의 VARBYTE 표현을 16진수로 변환하려면 다음 예제를 사용합니다.

```
SELECT TO_HEX('abc'::VARBYTE);

+--------+
| to_hex |
+--------+
| 616263 |
```

```
+--------+
```

테이블을 만들려면 16진수에 'abc'의 VARBYTE 표현을 삽입하고 값이 있는 열을 선택하면 됩니다. 다음 예제를 사용합니다.

```
CREATE TABLE t (vc VARCHAR);
INSERT INTO t SELECT TO_HEX('abc'::VARBYTE);
SELECT vc FROM t;

+--------+
|   vc   |
+--------+
| 616263 |
+--------+
```

VARBYTE 값을 VARCHAR로 캐스팅할 때 형식이 UTF-8임을 표시하려면 다음 예제를 사용합니다.

```
CREATE TABLE t (vc VARCHAR);
INSERT INTO t SELECT 'abc'::VARBYTE::VARCHAR;

SELECT vc FROM t;

+-----+
| vc  |
+-----+
| abc |
+-----+
```

TO_VARBYTE 함수

TO_VARBYTE는 지정된 형식의 문자열을 이진 값으로 변환합니다.

구문

```
TO_VARBYTE(string, format)
```

인수

string

CHAR 또는 VARCHAR 문자열입니다.

format

입력 문자열의 형식입니다. 대/소문자를 구분하지 않는 유효한 값은 hex, binary, utf8(utf-8 및 utf_8도 가능함) 및 base64입니다.

반환 타입

VARBYTE

예시

16진수 6162를 이진 값으로 변환하려면 다음 예제를 사용합니다. 결과는 이진 값의 16진수 표현으로 자동으로 표시됩니다.

```
SELECT TO_VARBYTE('6162', 'hex');

+------------+
| to_varbyte |
+------------+
|       6162 |
+------------+
```

4d의 이진 표현을 반환하려면 다음 예제를 사용합니다. '4d'의 이진 표현은 01001101입니다.

```
SELECT TO_VARBYTE('01001101', 'binary');

+------------+
| to_varbyte |
+------------+
|         4d |
+------------+
```

UTF-8의 문자열 'a'를 이진 값으로 변환하려면 다음 예제를 사용합니다. 결과는 이진 값의 16진수 표현으로 자동으로 표시됩니다.

```
SELECT TO_VARBYTE('a', 'utf8');

+------------+
| to_varbyte |
+------------+
```

```
|         61 |
+------------+
```

16진수 문자열 '4'를 이진 값으로 변환하려면 다음 예제를 사용합니다. 16진수 문자열 길이가 홀수이면 0이 앞에 추가되어 유효한 16진수를 구성합니다.

```
SELECT TO_VARBYTE('4', 'hex');

+------------+
| to_varbyte |
+------------+
|         04 |
+------------+
```

윈도 함수

창 함수를 사용하면 사용자가 분석 비즈니스 쿼리를 보다 효율적으로 생성할 수 있습니다. 창 함수는 결과 집합의 파티션, 즉 "창"에서 실행되어 해당 창에 속하는 모든 행에 대한 값을 반환합니다. 이와는 반대로 창이 없는 함수는 결과 집합의 모든 행에 대해 계산을 실행합니다. 그 밖에도 결과 행을 집계하는 그룹 함수와 달리 창 함수에서는 테이블 표현식의 모든 행이 그대로 유지됩니다.

반환 값은 해당 창에 속한 행 집합의 값을 사용하여 계산됩니다. 창은 테이블의 각 행마다 추가 속성을 계산하는 데 사용되는 행 집합을 정의합니다. 창은 창 명세(OVER 절)를 사용하여 정의되며, 다음과 같이 세 가지 주요 개념을 근거로 합니다.

- 창 파티션 - 행 그룹을 형성합니다(PARTITION 절).
- 창 순서 지정 - 각 파티션의 행 순서 또는 시퀀스를 정의합니다(ORDER BY 절).
- 창 프레임 - 행 집합을 제한하기 위해 각 행마다 정의됩니다(ROWS 명세).

창 함수는 최종 ORDER BY 절을 제외하고 쿼리에서 실행되는 마지막 연산 집합입니다. 창 함수를 처리할 때는 그 전에 모든 조인을 비롯한 WHERE, GROUP BY 및 HAVING 절까지 모두 완료됩니다. 따라서 창 함수는 선택 목록 또는 ORDER BY 절에만 나타날 수 있습니다. 다른 프레임 절이 있는 단일 쿼리 내에서 여러 윈도 함수를 사용할 수 있습니다. CASE 등의 다른 스칼라 표현식에서 윈도 함수를 사용할 수도 있습니다.

창 함수는 중첩할 수 없습니다. 예를 들어 SUM 집계 함수는 SUM 창 함수 내에 표시될 수 있지만 창 함수 SUM은 다른 창 함수 SUM 내에 표시될 수 없습니다. 창 함수가 다른 창 함수에 중첩되어 있으므로 다음 사항이 지원되지 않습니다.

```
SELECT SUM(SUM(selectcol) OVER (PARTITION BY ordercol)) OVER (Partition by ordercol)
 FROM t;
```

창 함수 구문 요약

Window 함수는 다음과 같은 표준 구문을 따릅니다.

```
function (expression) OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list [ frame_clause ] ] )
```

여기서 함수는 이 섹션에서 설명하는 함수 중 하나입니다.

expr_list는 다음과 같습니다.

```
expression | column_name [, expr_list ]
```

order_list는 다음과 같습니다.

```
expression | column_name [ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ]
[, order_list ]
```

frame_clause는 다음과 같습니다.

```
ROWS
{ UNBOUNDED PRECEDING | unsigned_value PRECEDING | CURRENT ROW } |

{ BETWEEN
{ UNBOUNDED PRECEDING | unsigned_value { PRECEDING | FOLLOWING } | CURRENT ROW}
AND
{ UNBOUNDED FOLLOWING | unsigned_value { PRECEDING | FOLLOWING } | CURRENT ROW }}
```

인수

함수

창 함수 자세한 내용은 각 함수에 대한 설명을 참조하십시오.

OVER

창 명세를 정의하는 절입니다. OVER 절은 창 함수에서 필수 인수로서 창 함수와 다른 SQL 함수를 구분하는 역할을 합니다.

PARTITION BY expr_list

(옵션) PARTITION BY 절은 결과 집합을 여러 파티션으로 분할한다는 점에서 GROUP BY 절과 매우 유사합니다. 파티션 절이 존재하는 경우에는 함수가 각 파티션의 행에 대해 계산됩니다. 반대로 파티션 절을 지정하지 않으면 전체 테이블이 단일 파티션으로 구성되어 함수가 해당하는 전체 테이블에 대해서 계산됩니다.

DENSE_RANK, NTILE, RANK, ROW_NUMBER 같은 순위 함수에서는 결과 집합의 모든 행을 전역적으로 비교해야 합니다. 이때 PARTITION BY 절을 사용하면 쿼리 옵티마이저가 워크로드를 파티션에 따라 다수의 조각으로 분산시키기 때문에 각 집계를 병렬 방식으로 실행할 수 있습니다. PARTITION BY 절을 사용하지 않으면 단일 조각에서 직렬 방식으로 집계를 실행해야 하기 때문에 특히 대용량의 클러스터에서는 성능에 매우 부정적인 영향을 끼치게 됩니다.

Amazon Redshift는 PARTITION BY 절에서 문자열 리터럴을 지원하지 않습니다.

ORDER BY order_list

(옵션) 윈도 함수는 ORDER BY의 순서 명세에 따라 정렬된 각 파티션의 행에 적용됩니다. 이 ORDER BY 절은 frame_clause의 ORDER BY 절과 구분되어 전혀 관련이 없습니다. 이러한 ORDER BY 절은 PARTITION BY 절 없이도 사용할 수 있습니다.

순위 함수에서는 ORDER BY 절이 순위 값의 기준을 식별하는 역할을 합니다. 집계 함수에서는 각 프레임에 대한 집계 함수 계산 이전에 파티션 행의 순서를 지정해야 합니다. 윈도 함수 형식에 대한 자세한 내용은 윈도 함수 섹션을 참조하세요.

order list에는 열 식별자, 또는 열 식별자로 평가되는 표현식이 필요합니다. 열 이름 대신에 상수나 상수 표현식을 사용할 수도 없습니다.

NULLS 값은 자체 그룹으로 처리되어 NULLS FIRST 또는 NULLS LAST 옵션에 따라 정렬 후 순위가 결정됩니다. 기본적으로 NULL 값은 ASC 순서에서는 마지막에 정렬 후 순위가 결정되며, DESC 순서에서는 처음에 정렬 후 순위가 결정됩니다.

Amazon Redshift는 ORDER BY 절에서 문자열 리터럴을 지원하지 않습니다.

ORDER BY 절을 생략하면 행의 순서는 비확정적입니다.

> **Note**
>
> Amazon Redshift와 같은 병렬 시스템에서는 ORDER BY 절이 데이터의 전체 순서를 고유하게 지정하지 않으면 행의 순서는 비확정적입니다. 다시 말해 ORDER BY 표현식에서 중복 값이 산출되면(부분 순서 지정) Amazon Redshift를 실행할 때마다 해당하는 행의 반환 순서가 달라질 수 있습니다. 그러면 창 함수 역시 예상하지 못하거나 일관적이지 못한 결과를 반환하게 됩니다. 자세한 내용은 창 함수 데이터에 대한 고유 순서 지정 단원을 참조하십시오.

column_name

파티션으로 분할하거나 순서를 지정할 때 기준이 되는 열의 이름입니다.

ASC | DESC

표현식의 정렬 순서를 정의하는 옵션으로서 각각 다음과 같은 의미를 갖습니다.

- ASC: 오름차순(예: 숫자 값의 경우 낮은 값에서 높은 값 순, 문자열의 경우 'A'에서 'Z'의 순. 지정된 옵션이 없는 경우에는 데이터가 기본적으로 오름차순으로 정렬됩니다.
- DESC: 내림차순(숫자 값의 경우 높은 값에서 낮은 값 순, 문자열의 경우 'Z'에서 'A'의 순).

NULLS FIRST | NULLS LAST

NULLS의 순서를 NULL 값 이외의 값 이전에 결정할지, 혹은 이후에 결정할지 지정하는 옵션입니다. 기본적으로 ASC 순서에서는 마지막에 정렬 후 순위가 결정되며, DESC 순서에서는 처음에 정렬 후 순위가 결정됩니다.

frame_clause

집계 함수에서 프레임 절은 ORDER BY를 사용하여 함수의 창에 포함되는 행 집합을 추가적으로 정제하는 역할을 합니다. 이를 통해 순서가 지정된 결과 내에 행 집합을 추가하거나 제거할 수 있습니다. ROWS 키워드와 관련 지정자로 구성됩니다.

프레임 절은 순위 함수에 적용되지 않습니다. 또한 집계 함수의 OVER 절에 ORDER BY 절이 사용되지 않는 경우 프레임 절이 필요하지 않습니다. 집계 함수에서 ORDER BY 절이 사용되면 명시적인 프레임 절이 필요합니다.

ORDER BY 절을 지정하지 않으면 묵시적 프레임이 무제한이기 때문에 ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING과 다름 없습니다.

ROWS

이 절은 현재 행에서 물리적 오프셋을 지정하여 창 프레임을 정의합니다.

이 절은 현재 창 또는 파티션에서 현재 행의 값이 결합되는 행을 지정합니다. 행의 위치는 인수를 사용하여 지정하며, 현재 행 앞 또는 뒤가 될 수 있습니다. 모든 창 프레임에서 기준점은 현재 행입니다. 각 행은 창 프레임이 파티션에서 밀려 앞으로 이동하면서 번갈아 현재 행이 됩니다.

프레임은 다음과 같이 현재 행까지 포함하여 단일 행 집합이 되거나,

```
{UNBOUNDED PRECEDING | offset PRECEDING | CURRENT ROW}
```

혹은 다음과 같이 두 경계 사이의 행 집합이 될 수도 있습니다.

```
BETWEEN
{ UNBOUNDED PRECEDING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
AND
{ UNBOUNDED FOLLOWING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
```

UNBOUNDED PRECEDING은 파티션의 첫 행에서 창이 시작된다는 것을 나타내고, offset PRECEDING은 오프셋 값에 해당하는 행의 수만큼 현재 행 앞에서 창이 시작된다는 것을 나타냅니다. 기본값은 UNBOUNDED PRECEDING입니다.

CURRENT ROW는 창이 현재 행에서 시작하거나 끝난다는 것을 나타냅니다.

UNBOUNDED FOLLOWING은 파티션의 마지막 행에서 창이 끝나는 것을 나타내고, offset FOLLOWING은 오프셋 값에 해당하는 행의 수만큼 현재 행 뒤에서 창이 끝난다는 것을 나타냅니다.

offset은 현재 행 앞 또는 뒤로 물리적인 행의 수를 의미합니다. 이 경우에는 offset이 양의 숫자 값으로 평가되는 상수여야 합니다. 예를 들어 5 FOLLOWING일 때는 현재 행 뒤로 5개 행을 지나 프레임이 종료됩니다.

BETWEEN을 지정하지 않으면 묵시적이지만 프레임 경계가 현재 행으로 결정됩니다. 예를 들어 `ROWS 5 PRECEDING`은 `ROWS BETWEEN 5 PRECEDING AND CURRENT ROW`와 같습니다. 또한 `ROWS UNBOUNDED FOLLOWING`은 `ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING`과 같습니다.

> **Note**
>
> 시작 경계가 종료 경계보다 크게 프레임을 지정할 수는 없습니다. 예를 들어 다음과 같은 프레임은 지정할 수 없습니다.
>
> ```
> between 5 following and 5 preceding
> ```

```
between current row and 2 preceding
between 3 following and current row
```

창 함수 데이터에 대한 고유 순서 지정

윈도 함수의 ORDER BY 절이 데이터의 전체 순서를 고유하게 지정하지 않으면 행의 순서는 비확정적입니다. 다시 말해 ORDER BY 표현식에서 중복 값이 산출되면(부분 순서 지정) 여러 차례 실행할 때마다 해당 행의 반환 순서가 달라질 수 있습니다. 이 경우 윈도 함수 역시 예상하지 못하거나 일관적이지 못한 결과를 반환하게 됩니다.

예를 들어 다음 쿼리는 여러 실행에 대해 다른 결과를 반환합니다. 이러한 다른 결과는 `order by dateid`가 SUM 윈도 함수 데이터의 고유한 순서를 생성하지 않기 때문에 발생합니다.

```
select dateid, pricepaid,
sum(pricepaid) over(order by dateid rows unbounded preceding) as sumpaid
from sales
group by dateid, pricepaid;

dateid | pricepaid |   sumpaid
--------+-----------+-------------
1827 |   1730.00 |     1730.00
1827 |    708.00 |     2438.00
1827 |    234.00 |     2672.00
...

select dateid, pricepaid,
sum(pricepaid) over(order by dateid rows unbounded preceding) as sumpaid
from sales
group by dateid, pricepaid;

dateid | pricepaid |   sumpaid
--------+-----------+-------------
1827 |    234.00 |      234.00
1827 |    472.00 |      706.00
1827 |    347.00 |     1053.00
...
```

이 경우에는 두 번째 ORDER BY 열을 윈도 함수에 추가하여 문제를 해결할 수 있습니다.

```
select dateid, pricepaid,
```

```
sum(pricepaid) over(order by dateid, pricepaid rows unbounded preceding) as sumpaid
from sales
group by dateid, pricepaid;

dateid | pricepaid | sumpaid
--------+-----------+---------
1827 |    234.00 |  234.00
1827 |    337.00 |  571.00
1827 |    347.00 |  918.00
...
```

지원되는 함수

Amazon Redshift는 집계와 순위, 두 가지 형식의 윈도 함수를 지원합니다.

다음은 지원되는 집계 함수입니다.

- AVG 창 함수
- COUNT 창 함수
- CUME_DIST 창 함수
- DENSE_RANK 창 함수
- FIRST_VALUE 창 함수
- LAG 창 함수
- LAST_VALUE 창 함수
- LEAD 창 함수
- LISTAGG 창 함수
- MAX 창 함수
- MEDIAN 창 함수
- MIN 창 함수
- NTH_VALUE 창 함수
- PERCENTILE_CONT 창 함수
- PERCENTILE_DISC 창 함수
- RATIO_TO_REPORT 창 함수
- STDDEV_SAMP 및 STDDEV_POP 창 함수(STDDEV_SAMP 및 STDDEV는 동의어)
- SUM 창 함수

- VAR_SAMP 및 VAR_POP 창 함수(VAR_SAMP와 VARIANCE는 동의어)

다음은 지원되는 순위 함수입니다.

- DENSE_RANK 창 함수
- NTILE 창 함수
- PERCENT_RANK 창 함수
- RANK 창 함수
- ROW_NUMBER 창 함수

창 함수 예제를 위한 샘플 테이블

각 함수 설명과 함께 특정 창 함수 예제를 찾을 수도 있습니다. 일부 예는 다음과 같은 11개의 행이 포함된 WINSALES라는 테이블을 사용합니다.

SALESID	DATEID	SELLERID	BUYERID	QTY	QTY_SHIPPED
30001	30001	3	B	10	10
10001	10001	1	C	10	10
10005	10005	1	A	30	
40001	40001	4	A	40	
10006	10006	1	C	10	
20001	20001	2	B	20	20
40005	40005	4	A	10	10
20002	20002	2	C	20	20
30003	30003	3	B	15	
30004	30004	3	B	20	
30007	30007	3	C	30	

다음은 WINSALES 샘플 테이블을 생성하여 채우는 스크립트입니다.

```
CREATE TABLE winsales(
  salesid int,
  dateid date,
  sellerid int,
  buyerid char(10),
  qty int,
  qty_shipped int);

INSERT INTO winsales VALUES
  (30001, '8/2/2003', 3, 'b', 10, 10),
  (10001, '12/24/2003', 1, 'c', 10, 10),
  (10005, '12/24/2003', 1, 'a', 30, null),
  (40001, '1/9/2004', 4, 'a', 40, null),
  (10006, '1/18/2004', 1, 'c', 10, null),
  (20001, '2/12/2004', 2, 'b', 20, 20),
  (40005, '2/12/2004', 4, 'a', 10, 10),
  (20002, '2/16/2004', 2, 'c', 20, 20),
  (30003, '4/18/2004', 3, 'b', 15, null),
  (30004, '4/18/2004', 3, 'b', 20, null),
  (30007, '9/7/2004', 3, 'c', 30, null);
```

AVG 창 함수

AVG 창 함수는 입력 표현식 값의 평균(산술 평균)을 반환합니다. AVG 함수는 숫자 값을 사용하고 NULL 값을 무시합니다.

구문

```
AVG ( [ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

인수

표현식

함수가 실행되는 대상 열 또는 표현식입니다.

ALL

인수가 ALL일 때는 함수가 표현식의 모든 중복 값을 그대로 유지한 채 행의 수를 계산합니다. ALL이 기본값입니다. DISTINCT는 지원되지 않습니다.

OVER

집계 함수의 창 절을 지정합니다. OVER 절은 창 집계 함수와 일반적인 집합 집계 함수를 구분하는 역할을 합니다.

PARTITION BY expr_list

하나 이상의 표현식과 관련하여 AVG 함수의 창을 정의합니다.

ORDER BY order_list

각 파티션의 행을 정렬합니다. PARTITION BY를 지정하지 않으면 ORDER BY가 전체 테이블을 사용합니다.

frame_clause

집계 함수에서 ORDER BY 절이 사용되면 명시적인 프레임 절이 필요합니다. 프레임 절은 순서가 지정된 결과 내에 행 집합을 추가하거나 제거함으로써 함수의 창에 포함되는 행 집합을 정제하는 역할을 하며, ROWS 키워드와 관련 지정자로 구성됩니다. 창 함수 구문 요약 섹션을 참조하세요.

데이터 타입

AVG 함수에서 지원되는 인수 형식은 SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL, REAL, DOUBLE PRECISION입니다.

AVG 함수에서 지원되는 반환 형식은 다음과 같습니다.

- SMALLINT 또는 INTEGER 인수일 때 BIGINT
- BIGINT 인수일 때 NUMERIC
- 부동 소수점 인수일 때 DOUBLE PRECISION

예시

다음 예에서는 날짜를 기준으로 판매된 수량의 이동 평균을 계산한 후 날짜 ID와 판매 ID에 따라 결과 순서를 지정합니다.

```
select salesid, dateid, sellerid, qty,
avg(qty) over
```

```
(order by dateid, salesid rows unbounded preceding) as avg
from winsales
order by 2,1;

salesid |   dateid   | sellerid | qty | avg
---------+------------+----------+-----+-----
30001 | 2003-08-02 |        3 |  10 |  10
10001 | 2003-12-24 |        1 |  10 |  10
10005 | 2003-12-24 |        1 |  30 |  16
40001 | 2004-01-09 |        4 |  40 |  22
10006 | 2004-01-18 |        1 |  10 |  20
20001 | 2004-02-12 |        2 |  20 |  20
40005 | 2004-02-12 |        4 |  10 |  18
20002 | 2004-02-16 |        2 |  20 |  18
30003 | 2004-04-18 |        3 |  15 |  18
30004 | 2004-04-18 |        3 |  20 |  18
30007 | 2004-09-07 |        3 |  30 |  19
(11 rows)
```

요청 데이터에 대한 설명은 창 함수 예제를 위한 샘플 테이블 섹션을 참조하세요.

COUNT 창 함수

COUNT 창 함수는 표현식에서 정의하는 행의 수를 계산합니다.

COUNT 함수는 2가지 변형이 있습니다. COUNT(*)는 NULL 값의 유무에 상관없이 대상 테이블에서 모든 행의 수를 계산합니다. COUNT(expression)는 특정 열 또는 표현식에서 NULL을 제외한 값이 포함된 행의 수를 계산합니다.

구문

```
COUNT ( * | [ ALL ] expression) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

인수

표현식

함수가 실행되는 대상 열 또는 표현식입니다.

ALL

인수가 ALL일 때는 함수가 표현식의 모든 중복 값을 그대로 유지한 채 행의 수를 계산합니다. ALL이 기본값입니다. DISTINCT는 지원되지 않습니다.

OVER

집계 함수의 창 절을 지정합니다. OVER 절은 창 집계 함수와 일반적인 집합 집계 함수를 구분하는 역할을 합니다.

PARTITION BY expr_list

하나 이상의 표현식과 관련하여 COUNT 함수의 창을 정의합니다.

ORDER BY order_list

각 파티션의 행을 정렬합니다. PARTITION BY를 지정하지 않으면 ORDER BY가 전체 테이블을 사용합니다.

frame_clause

집계 함수에서 ORDER BY 절이 사용되면 명시적인 프레임 절이 필요합니다. 프레임 절은 순서가 지정된 결과 내에 행 집합을 추가하거나 제거함으로써 함수의 창에 포함되는 행 집합을 정제하는 역할을 하며, ROWS 키워드와 관련 지정자로 구성됩니다. 창 함수 구문 요약 섹션을 참조하세요.

데이터 타입

COUNT 함수는 모든 인수 데이터 형식을 지원합니다.

COUNT 함수에서 지원되는 반환 형식은 BIGINT입니다.

예시

다음 예에서는 데이터 윈도의 시작부터 판매 ID, 수량 및 모든 행의 수를 보여줍니다.

```
select salesid, qty,
count(*) over (order by salesid rows unbounded preceding) as count
from winsales
order by salesid;

salesid | qty | count
---------+-----+-----
10001 |  10 |   1
```

```
10005 |  30 |   2
10006 |  10 |   3
20001 |  20 |   4
20002 |  20 |   5
30001 |  10 |   6
30003 |  15 |   7
30004 |  20 |   8
30007 |  30 |   9
40001 |  40 |   10
40005 |  10 |   11
(11 rows)
```

요청 데이터에 대한 설명은 창 함수 예제를 위한 샘플 테이블 섹션을 참조하세요.

다음 예에서는 데이터 윈도의 시작부터 판매 ID, 수량 및 null이 아닌 행의 수를 보여줍니다. WINSALES 테이블의 QTY_SHIPPED 열에는 일부 NULL이 포함되어 있습니다.

```
select salesid, qty, qty_shipped,
count(qty_shipped)
over (order by salesid rows unbounded preceding) as count
from winsales
order by salesid;

salesid | qty | qty_shipped | count
---------+-----+-------------+-------
10001 |  10 |          10 |   1
10005 |  30 |             |   1
10006 |  10 |             |   1
20001 |  20 |          20 |   2
20002 |  20 |          20 |   3
30001 |  10 |          10 |   4
30003 |  15 |             |   4
30004 |  20 |             |   4
30007 |  30 |             |   4
40001 |  40 |             |   4
40005 |  10 |          10 |   5
(11 rows)
```

CUME_DIST 창 함수

창 또는 파티션에 속하는 값의 누적 분포를 계산합니다. 오름차순을 가정했을 때 누적 분포는 다음과 같은 공식으로 결정됩니다.

```
count of rows with values <= x / count of rows in the window or partition
```

여기에서 x는 ORDER BY 절에서 지정하는 열의 현재 행 값과 동일합니다. 다음은 위와 같은 공식의 사용을 나타내는 데이터 세트입니다.

```
Row# Value   Calculation    CUME_DIST
1        2500    (1)/(5)    0.2
2        2600    (2)/(5)    0.4
3        2800    (3)/(5)    0.6
4        2900    (4)/(5)    0.8
5        3100    (5)/(5)    1.0
```

반환 값의 범위는 0부터 1까지입니다(1 포함).

구문

```
CUME_DIST ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
)
```

인수

OVER

창 파티션을 지정하는 절입니다. OVER 절에는 창 프레임 명세가 포함될 수 없습니다.

PARTITION BY partition_expression

선택 사항. OVER 절에서 각 그룹의 레코드 범위를 설정하는 표현식입니다.

ORDER BY order_list

누적 분포를 계산하기 위한 표현식입니다. 이 표현식은 숫자 데이터 형식을 갖거나, 혹은 묵시적으로 1로 변환될 수 있어야 합니다. 즉 ORDER BY가 생략되면 모든 행의 반환 값은 1입니다.

ORDER BY에서 고유한 순서를 지정하지 않으면 행의 순서는 비확정적입니다. 자세한 내용은 창 함수 데이터에 대한 고유 순서 지정 단원을 참조하십시오.

반환 타입

FLOAT8

예시

다음은 각 판매자의 수량 누적 분포를 계산하는 예입니다.

```
select sellerid, qty, cume_dist()
over (partition by sellerid order by qty)
from winsales;

sellerid   qty    cume_dist
------------------------------------------------
1          10.00    0.33
1          10.64    0.67
1          30.37    1
3          10.04    0.25
3          15.15    0.5
3          20.75    0.75
3          30.55    1
2          20.09    0.5
2          20.12    1
4          10.12    0.5
4          40.23    1
```

요청 데이터에 대한 설명은 창 함수 예제를 위한 샘플 테이블 섹션을 참조하세요.

DENSE_RANK 창 함수

DENSE_RANK 창 함수는 OVER 절의 ORDER BY 표현식을 기준으로 값 그룹에 속한 값의 순위를 결정합니다. PARTITION BY 절(옵션)이 존재하면 각 행 그룹의 순위가 재설정됩니다. 순위 기준 값이 같은 행은 순위도 동일하게 결정됩니다. DENSE_RANK 함수는 한 가지 측면에서 RANK와 다릅니다. 즉 2개 이상의 행에서 순위가 동일하면 순위 값의 순서에서도 빈 자리가 없습니다. 예를 들어 두 행의 순위가 1로 결정되면 다음 순위는 2입니다.

순위 함수에서는 동일한 쿼리라고 해도 PARTITION BY 절과 ORDER BY 절을 다르게 사용할 수 있습니다.

구문

```
DENSE_RANK() OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

인수

( )

함수에 인수가 없지만 빈 괄호가 필요합니다.

OVER

DENSE_RANK 함수의 창 절입니다.

PARTITION BY expr_list

(선택) 창을 정의하는 하나 이상의 표현식입니다.

ORDER BY order_list

(선택) 순위 값의 기준이 되는 표현식입니다. PARTITION BY를 지정하지 않으면 ORDER BY가 전체 테이블을 사용합니다. 즉 ORDER BY가 생략되면 모든 행의 반환 값은 1입니다.

ORDER BY에서 고유한 순서를 지정하지 않으면 행의 순서는 비확정적입니다. 자세한 내용은 창 함수 데이터에 대한 고유 순서 지정 단원을 참조하십시오.

반환 타입

BIGINT

예시

다음 예제에서는 창 함수에 대한 샘플 테이블을 사용합니다. 자세한 내용은 창 함수 예제를 위한 샘플 테이블 단원을 참조하십시오.

다음 예에서는 판매 수량으로 테이블을 정렬하고 각 행에 밀집 순위와 정규 순위를 모두 할당합니다. 결과는 창 함수 결과를 적용한 후에 정렬됩니다.

```
SELECT salesid, qty,
DENSE_RANK() OVER(ORDER BY qty DESC) AS d_rnk,
RANK() OVER(ORDER BY qty DESC) AS rnk
FROM winsales
ORDER BY 2,1;

+---------+-----+-------+-----+
| salesid | qty | d_rnk | rnk |
+---------+-----+-------+-----+
|   10001 |  10 |     5 |   8 |
```

```
|   10006 |  10 |     5 |   8 |
|   30001 |  10 |     5 |   8 |
|   40005 |  10 |     5 |   8 |
|   30003 |  15 |     4 |   7 |
|   20001 |  20 |     3 |   4 |
|   20002 |  20 |     3 |   4 |
|   30004 |  20 |     3 |   4 |
|   10005 |  30 |     2 |   2 |
|   30007 |  30 |     2 |   2 |
|   40001 |  40 |     1 |   1 |
+---------+-----+-------+-----+
```

동일한 쿼리에서 DENSE_RANK와 RANK 함수를 함께 사용하여 같은 행 집합에 할당되는 순위의 차이를 기록합니다.

다음 예제에서는 sellerid를 기준으로 테이블을 분할하고, 각 분할을 수량별로 정렬하고, 각 행에 고밀도 순위를 할당합니다. 결과는 창 함수 결과를 적용한 후에 정렬됩니다.

```
SELECT salesid, sellerid, qty,
DENSE_RANK() OVER(PARTITION BY sellerid ORDER BY qty DESC) AS d_rnk
FROM winsales
ORDER BY 2,3,1;

+---------+----------+-----+-------+
| salesid | sellerid | qty | d_rnk |
+---------+----------+-----+-------+
|   10001 |        1 |  10 |     2 |
|   10006 |        1 |  10 |     2 |
|   10005 |        1 |  30 |     1 |
|   20001 |        2 |  20 |     1 |
|   20002 |        2 |  20 |     1 |
|   30001 |        3 |  10 |     4 |
|   30003 |        3 |  15 |     3 |
|   30004 |        3 |  20 |     2 |
|   30007 |        3 |  30 |     1 |
|   40005 |        4 |  10 |     2 |
|   40001 |        4 |  40 |     1 |
+---------+----------+-----+-------+
```

마지막 예제를 성공적으로 사용하려면 다음 명령을 사용하여 WINSALES 테이블에 행을 삽입합니다. 이 행은 다른 행과 동일한 buyerid, sellerid 및 qtysold를 가집니다. 이렇게 하면 마지막 예제에서 두 행이 동점이 되므로 DENSE_RANK와 RANK 함수의 차이점이 표시됩니다.

```
INSERT INTO winsales VALUES(30009, '2/2/2003', 3, 'b', 20, NULL);
```

다음 예제에서는 buyerid 및 sellerid를 기준으로 테이블을 분할하고, 각 분할을 수량별로 정렬하고, 각 행에 밀도 순위와 일반 순위를 모두 할당합니다. 결과는 창 함수가 적용된 후 정렬됩니다.

```
SELECT salesid, sellerid, qty, buyerid,
DENSE_RANK() OVER(PARTITION BY buyerid, sellerid ORDER BY qty DESC) AS d_rnk,
RANK() OVER (PARTITION BY buyerid, sellerid ORDER BY qty DESC) AS rnk
FROM winsales
ORDER BY rnk;

+---------+----------+-----+---------+-------+-----+
| salesid | sellerid | qty | buyerid | d_rnk | rnk |
+---------+----------+-----+---------+-------+-----+
|   20001 |        2 |  20 | b       |     1 |   1 |
|   30007 |        3 |  30 | c       |     1 |   1 |
|   10006 |        1 |  10 | c       |     1 |   1 |
|   10005 |        1 |  30 | a       |     1 |   1 |
|   20002 |        2 |  20 | c       |     1 |   1 |
|   30009 |        3 |  20 | b       |     1 |   1 |
|   40001 |        4 |  40 | a       |     1 |   1 |
|   30004 |        3 |  20 | b       |     1 |   1 |
|   10001 |        1 |  10 | c       |     1 |   1 |
|   40005 |        4 |  10 | a       |     2 |   2 |
|   30003 |        3 |  15 | b       |     2 |   3 |
|   30001 |        3 |  10 | b       |     3 |   4 |
+---------+----------+-----+---------+-------+-----+
```

FIRST_VALUE 창 함수

행 집합의 순서가 지정되었다고 가정할 때 FIRST VALUE 함수는 창 프레임의 첫 번째 행과 관련하여 지정된 표현식의 값을 반환합니다.

프레임의 마지막 행 선택에 대한 자세한 내용은 LAST_VALUE 창 함수 섹션을 참조하세요.

구문

```
FIRST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

인수

expression

함수가 실행되는 대상 열 또는 표현식입니다.

IGNORE NULLS

FIRST_VALUE에서 이 옵션을 사용하면 프레임에서 NULL이 아닌 첫 번째 값을 반환합니다(또는 모든 값이 NULL이면 NULL을 반환합니다).

RESPECT NULLS

Amazon Redshift에서 사용할 행을 결정할 때 null 값을 포함시켜야 한다는 것을 의미합니다. RESPECT NULLS는 IGNORE NULLS를 지정하지 않은 경우 기본적으로 지원됩니다.

OVER

함수에서 창 절을 삽입합니다.

PARTITION BY expr_list

하나 이상의 표현식과 관련하여 함수의 창을 정의합니다.

ORDER BY order_list

각 파티션의 행을 정렬합니다. PARTITION BY 절을 지정하지 않으면 ORDER BY가 전체 테이블을 정렬합니다. ORDER BY 절을 지정하면 frame_clause 역시 지정해야 합니다.

FIRST_VALUE 함수의 결과는 데이터 순서에 따라 결정됩니다. 다음과 같은 경우 함수 결과는 비확정적입니다.

- ORDER BY 절이 지정되지 않고 파티션에 다른 표현식 값 2개가 포함된 경우
- 표현식이 ORDER BY 목록에서는 동일한 값이지만 다른 값으로 평가되는 경우

frame_clause

집계 함수에서 ORDER BY 절이 사용되면 명시적인 프레임 절이 필요합니다. 프레임 절은 순서가 지정된 결과에 행 집합을 추가하거나 제거함으로써 함수의 창에 포함되는 행 집합을 정제하는 역할을 하며, ROWS 키워드와 관련 지정자로 구성됩니다. 창 함수 구문 요약 섹션을 참조하세요.

반환 타입

이 두 함수는 기본 Amazon Redshift 데이터 형식을 사용하는 표현식을 지원합니다. 반환 형식은 expression 데이터 형식과 동일합니다.

예시

다음 예시에서는 샘플 TICKIT 데이터의 VENUE 테이블을 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

다음은 VENUE 테이블에서 각 장소의 좌석 수용 능력을 반환하는 예로서 함수 결과의 순서(내림차순)는 좌석 수용 능력에 따라 지정됩니다. FIRST_VALUE 함수는 프레임에서 첫 번째 행에 해당하는 장소의 이름을 선택할 때 사용됩니다. 이 경우에는 좌석 수가 가장 많은 행이 여기에 해당합니다. 결과가 주를 기준으로 분할되어 있으므로 VENUESTATE 값이 바뀌면 첫 번째 값도 새롭게 선택됩니다. 여기에서는 창 프레임의 경계가 없기 때문에 각 파티션의 행마다 선택되는 첫 번째 값이 동일합니다.

California를 예로 들면, `Qualcomm Stadium`의 좌석 수(`70561`)가 가장 높기 때문에 이 장소의 이름이 `CA` 파티션의 모든 행에 대한 첫 번째 값에 해당합니다.

```
select venuestate, venueseats, venuename,
first_value(venuename)
over(partition by venuestate
order by venueseats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venueseats >0)
order by venuestate;

venuestate | venueseats |           venuename            |         first_value
-----------+------------+--------------------------------
+-------------------------------
CA         |      70561 | Qualcomm Stadium               | Qualcomm Stadium
CA         |      69843 | Monster Park                   | Qualcomm Stadium
CA         |      63026 | McAfee Coliseum                | Qualcomm Stadium
CA         |      56000 | Dodger Stadium                 | Qualcomm Stadium
CA         |      45050 | Angel Stadium of Anaheim       | Qualcomm Stadium
CA         |      42445 | PETCO Park                     | Qualcomm Stadium
CA         |      41503 | AT&T Park                      | Qualcomm Stadium
CA         |      22000 | Shoreline Amphitheatre         | Qualcomm Stadium
CO         |      76125 | INVESCO Field                  | INVESCO Field
CO         |      50445 | Coors Field                    | INVESCO Field
DC         |      41888 | Nationals Park                 | Nationals Park
FL         |      74916 | Dolphin Stadium                | Dolphin Stadium
FL         |      73800 | Jacksonville Municipal Stadium | Dolphin Stadium
FL         |      65647 | Raymond James Stadium          | Dolphin Stadium
FL         |      36048 | Tropicana Field                | Dolphin Stadium
...
```

다음은 IGNORE NULLS 옵션을 사용하는 예로서 새로운 행을 VENUE 테이블에 추가합니다.

```
insert into venue values(2000,null,'Stanford','CA',90000);
```

위의 새로운 행에서는 VENUENAME 열에 NULL 값이 포함되어 있습니다. 이제 이번 단원 앞에서 실행했던 FIRST_VALUE 쿼리를 반복합니다.

```
select venuestate, venueseats, venuename,
first_value(venuename)
over(partition by venuestate
order by venueseats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venueseats >0)
order by venuestate;

venuestate | venueseats |         venuename          | first_value
-----------+------------+----------------------------+-------------
CA         |      90000 | NULL                       | NULL
CA         |      70561 | Qualcomm Stadium           | NULL
CA         |      69843 | Monster Park               | NULL
...
```

새로운 행의 VENUESEATS 값(90000)이 가장 높지만 VENUENAME이 NULL 값이기 때문에 FIRST_VALUE 함수는 CA 파티션에 대해 NULL을 반환합니다. 이렇게 함수 평가에서 행을 무시하려면 IGNORE NULLS 옵션을 아래와 같이 함수 인수에 추가하면 됩니다.

```
select venuestate, venueseats, venuename,
first_value(venuename) ignore nulls
over(partition by venuestate
order by venueseats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venuestate='CA')
order by venuestate;

venuestate | venueseats |         venuename          |   first_value
------------+------------+----------------------------+------------------
CA         |      90000 | NULL                       | Qualcomm Stadium
CA         |      70561 | Qualcomm Stadium           | Qualcomm Stadium
CA         |      69843 | Monster Park               | Qualcomm Stadium
...
```

LAG 창 함수

LAG 창 함수는 파티션에서 현재 행 위(앞)의 지정 오프셋에 위치한 행의 값을 반환합니다.

구문

```
LAG (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

인수

value_expr

함수가 실행되는 대상 열 또는 표현식입니다.

Offset

현재 행 앞으로 값을 반환할 행이 위치한 수를 지정하는 파라미터(옵션)입니다. 이 오프셋은 상수 정수 혹은 정수로 평가되는 표현식이 될 수 있습니다. 오프셋을 지정하지 않으면 Amazon Redshift가 기본값으로 1을 사용합니다. 오프셋이 0이면 현재 행을 나타냅니다.

IGNORE NULLS

Amazon Redshift에서 사용할 행을 결정할 때 null 값을 건너뛰어야 한는 것을 의미하는 명세(옵션)입니다. IGNORE NULLS를 지정하지 않으면 NULL 값이 포함됩니다.

> **Note**
>
> NVL 또는 COALESCE 표현식을 사용하여 NULL 값을 다른 값으로 변경할 수도 있습니다. 자세한 내용은 NVL 및 COALESCE 함수 단원을 참조하십시오.

RESPECT NULLS

Amazon Redshift에서 사용할 행을 결정할 때 null 값을 포함시켜야 한다는 것을 의미합니다. RESPECT NULLS는 IGNORE NULLS를 지정하지 않은 경우 기본적으로 지원됩니다.

OVER

창 파티션 및 순서를 지정합니다. OVER 절에는 창 프레임 명세가 포함될 수 없습니다.

PARTITION BY window_partition

OVER 절에서 각 그룹의 레코드 범위를 설정하는 인수(옵션)입니다.

ORDER BY window_ordering

각 파티션의 행을 정렬합니다.

LAG 윈도 함수는 Amazon Redshift 데이터 형식을 모두 사용하는 표현식을 지원합니다. 반환 형식은 value_expr 형식과 동일합니다.

예시

다음은 구매자 ID가 3인 구매자에게 팔린 티켓 수량과 구매자 3이 티켓을 구입한 시간을 나타내는 예입니다. 쿼리가 구매자 3의 각 판매 수량을 이전 판매 수량과 비교할 수 있도록 각 판매 수량에 대한 이전 판매 수량을 반환합니다. 2008년 1월 16일 이전에는 구매 기록이 없기 때문에 이전 판매 수량의 첫 번째 값은 NULL입니다.

```
select buyerid, saletime, qtysold,
lag(qtysold,1) over (order by buyerid, saletime) as prev_qtysold
from sales where buyerid = 3 order by buyerid, saletime;

buyerid |      saletime       | qtysold | prev_qtysold
---------+---------------------+---------+--------------
3 | 2008-01-16 01:06:09 |       1 |
3 | 2008-01-28 02:10:01 |       1 |            1
3 | 2008-03-12 10:39:53 |       1 |            1
3 | 2008-03-13 02:56:07 |       1 |            1
3 | 2008-03-29 08:21:39 |       2 |            1
3 | 2008-04-27 02:39:01 |       1 |            2
3 | 2008-08-16 07:04:37 |       2 |            1
3 | 2008-08-22 11:45:26 |       2 |            2
3 | 2008-09-12 09:11:25 |       1 |            2
3 | 2008-10-01 06:22:37 |       1 |            1
3 | 2008-10-20 01:55:51 |       2 |            1
3 | 2008-10-28 01:30:40 |       1 |            2
(12 rows)
```

LAST_VALUE 창 함수

행 집합의 순서가 지정되었다고 가정할 때 LAST VALUE 함수는 프레임의 마지막 행과 관련하여 표현식의 값을 반환합니다.

프레임의 첫 번째 행 선택에 대한 자세한 내용은 FIRST_VALUE 창 함수 섹션을 참조하세요.

구문

```
LAST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

인수

expression

함수가 실행되는 대상 열 또는 표현식입니다.

IGNORE NULLS

프레임에서 NULL이 아닌 마지막 값을 반환합니다(또는 모든 값이 NULL이면 NULL을 반환합니다).

RESPECT NULLS

Amazon Redshift에서 사용할 행을 결정할 때 null 값을 포함시켜야 한다는 것을 의미합니다. RESPECT NULLS는 IGNORE NULLS를 지정하지 않은 경우 기본적으로 지원됩니다.

OVER

함수에서 창 절을 삽입합니다.

PARTITION BY expr_list

하나 이상의 표현식과 관련하여 함수의 창을 정의합니다.

ORDER BY order_list

각 파티션의 행을 정렬합니다. PARTITION BY 절을 지정하지 않으면 ORDER BY가 전체 테이블을 정렬합니다. ORDER BY 절을 지정하면 frame_clause 역시 지정해야 합니다.

결과는 데이터 순서에 따라 달라집니다. 다음과 같은 경우 함수 결과는 비확정적입니다.

- ORDER BY 절이 지정되지 않고 파티션에 다른 표현식 값 2개가 포함된 경우
- 표현식이 ORDER BY 목록에서는 동일한 값이지만 다른 값으로 평가되는 경우

frame_clause

집계 함수에서 ORDER BY 절이 사용되면 명시적인 프레임 절이 필요합니다. 프레임 절은 순서가 지정된 결과에 행 집합을 추가하거나 제거함으로써 함수의 창에 포함되는 행 집합을 정제하는 역할을 하며, ROWS 키워드와 관련 지정자로 구성됩니다. 창 함수 구문 요약 섹션을 참조하세요.

반환 타입

이 두 함수는 기본 Amazon Redshift 데이터 형식을 사용하는 표현식을 지원합니다. 반환 형식은 expression 데이터 형식과 동일합니다.

예시

다음 예시에서는 샘플 TICKIT 데이터의 VENUE 테이블을 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

다음은 VENUE 테이블에서 각 장소의 좌석 수용 능력을 반환하는 예로서 함수 결과의 순서(내림차순)는 좌석 수용 능력에 따라 지정됩니다. LAST_VALUE 함수는 프레임에서 마지막 행에 해당하는 장소의 이름을 선택할 때 사용됩니다. 이 경우에는 좌석 수가 가장 적은 행이 여기에 해당합니다. 결과가 주를 기준으로 분할되어 있으므로 VENUESTATE 값이 바뀌면 마지막 값도 새롭게 선택됩니다. 여기에서는 창 프레임의 경계가 없기 때문에 각 파티션의 행마다 선택되는 마지막 값이 동일합니다.

California를 보면, 파티션의 모든 행에 대해서 좌석 수(Shoreline Amphitheatre)가 가장 적은 22000가 반환됩니다.

```
select venuestate, venueseats, venuename,
last_value(venuename)
over(partition by venuestate
order by venueseats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venueseats >0)
order by venuestate;

venuestate | venueseats |           venuename            |          last_value
-----------+------------+--------------------------------
+-----------------------------
CA         |      70561 | Qualcomm Stadium               | Shoreline Amphitheatre
CA         |      69843 | Monster Park                   | Shoreline Amphitheatre
CA         |      63026 | McAfee Coliseum                | Shoreline Amphitheatre
CA         |      56000 | Dodger Stadium                 | Shoreline Amphitheatre
CA         |      45050 | Angel Stadium of Anaheim       | Shoreline Amphitheatre
CA         |      42445 | PETCO Park                     | Shoreline Amphitheatre
CA         |      41503 | AT&T Park                      | Shoreline Amphitheatre
CA         |      22000 | Shoreline Amphitheatre         | Shoreline Amphitheatre
CO         |      76125 | INVESCO Field                  | Coors Field
CO         |      50445 | Coors Field                    | Coors Field
DC         |      41888 | Nationals Park                 | Nationals Park
FL         |      74916 | Dolphin Stadium                | Tropicana Field
FL         |      73800 | Jacksonville Municipal Stadium | Tropicana Field
```

```
FL         |      65647 | Raymond James Stadium          | Tropicana Field
FL         |      36048 | Tropicana Field                | Tropicana Field
...
```

LEAD 창 함수

LEAD 창 함수는 파티션에서 현재 행 아래(뒤)의 지정 오프셋에 위치한 행의 값을 반환합니다.

구문

```
LEAD (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

인수

value_expr

함수가 실행되는 대상 열 또는 표현식입니다.

Offset

현재 행 아래로 값을 반환할 행이 위치한 수를 지정하는 파라미터(옵션)입니다. 이 오프셋은 상수 정수 혹은 정수로 평가되는 표현식이 될 수 있습니다. 오프셋을 지정하지 않으면 Amazon Redshift가 기본값으로 1을 사용합니다. 오프셋이 0이면 현재 행을 나타냅니다.

IGNORE NULLS

Amazon Redshift에서 사용할 행을 결정할 때 null 값을 건너뛰어야 한는 것을 의미하는 명세(옵션)입니다. IGNORE NULLS를 지정하지 않으면 NULL 값이 포함됩니다.

> **Note**
>
> NVL 또는 COALESCE 표현식을 사용하여 NULL 값을 다른 값으로 변경할 수도 있습니다. 자세한 내용은 NVL 및 COALESCE 함수 단원을 참조하십시오.

RESPECT NULLS

Amazon Redshift에서 사용할 행을 결정할 때 null 값을 포함시켜야 한다는 것을 의미합니다. RESPECT NULLS는 IGNORE NULLS를 지정하지 않은 경우 기본적으로 지원됩니다.

OVER

창 파티션 및 순서를 지정합니다. OVER 절에는 창 프레임 명세가 포함될 수 없습니다.

PARTITION BY window_partition

OVER 절에서 각 그룹의 레코드 범위를 설정하는 인수(옵션)입니다.

ORDER BY window_ordering

각 파티션의 행을 정렬합니다.

LEAD 윈도 함수는 Amazon Redshift 데이터 형식을 모두 사용하는 표현식을 지원합니다. 반환 형식은 value_expr 형식과 동일합니다.

예시

다음은 SALES 테이블의 이벤트에 대해 2008년 1월 1일과 동년 1월 2일에 판매된 티켓 수수료와 후속 티켓 판매 수수료를 나타낸 예입니다. 다음 예제에서는 TICKIT 샘플 데이터베이스를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

```
SELECT eventid, commission, saletime, LEAD(commission, 1) over ( ORDER BY saletime ) AS
 next_comm
FROM sales
WHERE saletime BETWEEN '2008-01-09 00:00:00' AND '2008-01-10 12:59:59'
LIMIT 10;

+---------+------------+---------------------+-----------+
| eventid | commission |      saletime       | next_comm |
+---------+------------+---------------------+-----------+
|    1664 |       13.2 | 2008-01-09 01:00:21 |      69.6 |
|     184 |       69.6 | 2008-01-09 01:00:36 |     116.1 |
|    6870 |      116.1 | 2008-01-09 01:02:37 |      11.1 |
|    3718 |       11.1 | 2008-01-09 01:05:19 |     205.5 |
|    6772 |      205.5 | 2008-01-09 01:14:04 |      38.4 |
|    3074 |       38.4 | 2008-01-09 01:26:50 |     209.4 |
|    5254 |      209.4 | 2008-01-09 01:29:16 |      26.4 |
|    3724 |       26.4 | 2008-01-09 01:40:09 |      57.6 |
|    5303 |       57.6 | 2008-01-09 01:40:21 |      51.6 |
|    3678 |       51.6 | 2008-01-09 01:42:54 |      43.8 |
+---------+------------+---------------------+-----------+
```

다음 예제에서는 SALES 테이블에 있는 이벤트의 커미션과 동일한 이벤트의 후속 티켓 판매에 대해 지불한 커미션의 최대 차이를 보여줍니다. 이 예에서는 GROUP BY 절과 함께 LEAD를 사용하는 방법을

보여줍니다. 집계 절에서는 창 함수를 사용할 수 없으므로 이 예제에서는 하위 쿼리를 사용합니다. 다음 예제에서는 TICKIT 샘플 데이터베이스를 사용합니다. 자세한 내용은 샘플 데이터베이스 단원을 참조하십시오.

```
SELECT eventid, eventname, max(next_comm_diff) as max_commission_difference
FROM
(
    SELECT sales.eventid, eventname, commission - LEAD(commission, 1) over (ORDER BY
 sales.eventid, saletime) AS next_comm_diff
    FROM sales JOIN event ON sales.eventid = event.eventid
)
GROUP BY eventid, eventname
ORDER BY eventid

LIMIT 10

| eventid | eventname                  | max_commission_difference |
+---------+----------------------------+---------------------------+
| 1       | Gotterdammerung            | 7.95                      |
| 2       | Boris Godunov              | 227.85                    |
| 3       | Salome                     | 1350.9                    |
| 4       | La Cenerentola (Cinderella) | 790.05                    |
| 5       | Il Trovatore               | 214.05                    |
| 6       | L Elisir d Amore           | 510.9                     |
| 7       | Doctor Atomic              | 180.6                     |
| 9       | The Fly                    | 147                       |
| 10      | Rigoletto                  | 186.6                     |
+---------+----------------------------+---------------------------+
```

LISTAGG 창 함수

LISTAGG 창 함수는 ORDER BY 표현식에 따라 쿼리 내 각 그룹의 행 순서를 지정한 다음, 값을 연결하여 문자열 하나로 만듭니다.

구문

```
LISTAGG( [DISTINCT] expression [, 'delimiter' ] )
[ WITHIN GROUP (ORDER BY order_list) ]
OVER ( [PARTITION BY partition_expression] )
```

인수

DISTINCT

(선택 사항) 연결하기 전에 지정된 표현식에서 중복 값을 없애는 절입니다. 후행 공백은 무시되므로 문자열 'a'와 'a '를 중복으로 간주합니다. LISTAGG는 발생한 첫 번째 값을 사용합니다. 자세한 내용은 후행 공백의 중요성 단원을 참조하십시오.

aggregate_expression

집계할 값을 제공하는 모든 유효 표현식(열 이름 등)입니다. NULL 값과 빈 문자열은 무시됩니다.

delimiter

(선택 사항) 연결된 값을 구분하는 문자열 상수입니다. 기본값은 NULL입니다.

WITHIN GROUP (ORDER BY order_list)

(선택 사항) 집계된 값의 정렬 순서를 지정하는 절입니다. ORDER BY 절에서 고유한 순서를 지정한 경우에 한해 확정적입니다. 기본값은 모든 행을 집계한 후 단일 값을 반환하는 것입니다.

OVER

창 파티션을 지정하는 절입니다. OVER 절에는 창 순서 또는 창 프레임 명세가 포함될 수 없습니다.

PARTITION BY partition_expression

(선택 사항) OVER 절에서 각 그룹의 레코드 범위를 설정합니다.

반환

VARCHAR(최대). 결과 집합이 최대 VARCHAR 크기(64K – 1, 즉 65535)보다 클 경우에는 LISTAGG가 다음과 같은 오류를 반환합니다.

```
Invalid operation: Result size exceeds LISTAGG limit
```

예시

아래 예들에서는 WINSALES 테이블을 사용합니다. 요청 데이터에 대한 설명은 창 함수 예제를 위한 샘플 테이블 섹션을 참조하세요.

다음은 판매자 ID에 따라 순서를 지정하여 판매자 ID 목록을 반환하는 예입니다.

```
select listagg(sellerid)
within group (order by sellerid)
```

```
over() from winsales;

  listagg
------------
 11122333344
...
...
 11122333344
 11122333344
   (11 rows)
```

다음은 날짜에 따라 순서를 지정하여 구매자 B의 판매자 ID 목록을 반환하는 예입니다.

```
select listagg(sellerid)
within group (order by dateid)
over () as seller
from winsales
where buyerid = 'b' ;

  seller
---------
    3233
    3233
    3233
    3233
```

다음은 구매자 B의 판매 날짜 목록을 쉼표로 구분하여 반환하는 예입니다.

```
select listagg(dateid,',')
within group (order by sellerid desc,salesid asc)
over () as dates
from winsales
where buyerid  = 'b';

             dates
-----------------------------------------
2003-08-02,2004-04-18,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-04-18,2004-02-12
```

다음 예에서는 DISTINCT를 사용하여 구매자 B의 고유한 판매 날짜 목록을 반환합니다.

```
select listagg(distinct dateid,',')
within group (order by sellerid desc,salesid asc)
over () as dates
from winsales
where buyerid  = 'b';

           dates
-------------------------------
2003-08-02,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-02-12
```

다음은 각 구매자 ID마다 판매 ID 목록을 쉼표로 구분하여 반환하는 예입니다.

```
select buyerid,
listagg(salesid,',')
within group (order by salesid)
over (partition by buyerid) as sales_id
from winsales
order by buyerid;

+---------+------------------------+
| buyerid |        sales_id        |
+---------+------------------------+
| a       | 10005,40001,40005      |
| a       | 10005,40001,40005      |
| a       | 10005,40001,40005      |
| b       | 20001,30001,30003,30004 |
| b       | 20001,30001,30003,30004 |
| b       | 20001,30001,30003,30004 |
| b       | 20001,30001,30003,30004 |
| c       | 10001,10006,20002,30007 |
| c       | 10001,10006,20002,30007 |
| c       | 10001,10006,20002,30007 |
| c       | 10001,10006,20002,30007 |
+---------+------------------------+
```

MAX 창 함수

MAX 창 함수는 입력 표현식의 최댓값을 반환합니다. MAX 함수는 숫자 값을 사용하고 NULL 값을 무시합니다.

구문

```
MAX ( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

인수

표현식

함수가 실행되는 대상 열 또는 표현식입니다.

ALL

인수가 ALL일 때는 함수가 표현식의 모든 중복 값을 그대로 유지합니다. ALL이 기본값입니다. DISTINCT는 지원되지 않습니다.

OVER

집계 함수의 창 절을 지정하는 절입니다. OVER 절은 창 집계 함수와 일반적인 집합 집계 함수를 구분하는 역할을 합니다.

PARTITION BY expr_list

하나 이상의 표현식과 관련하여 MAX 함수의 창을 정의합니다.

ORDER BY order_list

각 파티션의 행을 정렬합니다. PARTITION BY를 지정하지 않으면 ORDER BY가 전체 테이블을 사용합니다.

frame_clause

집계 함수에서 ORDER BY 절이 사용되면 명시적인 프레임 절이 필요합니다. 프레임 절은 순서가 지정된 결과 내에 행 집합을 추가하거나 제거함으로써 함수의 창에 포함되는 행 집합을 정제하는 역할을 하며, ROWS 키워드와 관련 지정자로 구성됩니다. 창 함수 구문 요약 섹션을 참조하세요.

데이터 타입

입력값으로 모든 데이터 형식을 지원합니다. expression과 동일한 데이터 형식을 반환합니다.

예시

다음 예에서는 데이터 윈도의 시작부터 판매 ID, 수량 및 최대 수량을 보여줍니다.

```
select salesid, qty,
max(qty) over (order by salesid rows unbounded preceding) as max
from winsales
order by salesid;

salesid | qty | max
---------+-----+-----
10001 |  10 |  10
10005 |  30 |  30
10006 |  10 |  30
20001 |  20 |  30
20002 |  20 |  30
30001 |  10 |  30
30003 |  15 |  30
30004 |  20 |  30
30007 |  30 |  30
40001 |  40 |  40
40005 |  10 |  40
(11 rows)
```

요청 데이터에 대한 설명은 창 함수 예제를 위한 샘플 테이블 섹션을 참조하세요.

다음 예에서는 제한적 프레임 내에서 판매 ID, 수량 및 최대 수량을 보여줍니다.

```
select salesid, qty,
max(qty) over (order by salesid rows between 2 preceding and 1 preceding) as max
from winsales
order by salesid;

salesid | qty | max
---------+-----+-----
10001 |  10 |
10005 |  30 |  10
10006 |  10 |  30
20001 |  20 |  30
20002 |  20 |  20
30001 |  10 |  20
30003 |  15 |  20
30004 |  20 |  15
30007 |  30 |  20
40001 |  40 |  30
40005 |  10 |  40
```

```
(11 rows)
```

MEDIAN 창 함수

창 또는 파티션에서 값의 범위에 대한 중간 값을 계산합니다. 범위 내 NULL 값은 무시됩니다.

MEDIAN은 연속 분포 모델을 가정하는 역분포 함수입니다.

구문

```
MEDIAN ( median_expression )
OVER ( [ PARTITION BY partition_expression ] )
```

인수

median_expression

중간을 결정할 값을 제공하는 표현식(열 이름 등)입니다. 이 표현식은 숫자 또는 날짜/시간 데이터 형식을 갖거나, 혹은 묵시적으로 1로 변환될 수 있어야 합니다.

OVER

창 파티션을 지정하는 절입니다. OVER 절에는 창 순서 또는 창 프레임 명세가 포함될 수 없습니다.

PARTITION BY partition_expression

선택 사항. OVER 절에서 각 그룹의 레코드 범위를 설정하는 표현식입니다.

데이터 타입

반환 형식은 median_expression의 형식에 따라 결정됩니다. 다음 표는 각 median_expression 데이터 형식에 따른 반환 형식을 나타낸 것입니다.

입력 형식	반환 유형
INT2, INT4, INT8, NUMERIC, DECIMAL	DECIMAL
FLOAT, DOUBLE	DOUBLE
날짜	날짜

사용 노트

median_expression 인수가 최대 정밀도가 38자리로 정의된 DECIMAL 데이터 형식인 경우에는 MEDIAN이 부정확한 결과 또는 오류를 반환합니다. MEDIAN 함수의 반환 값이 38자리를 초과하면 정밀도가 손실될 수도 있기 때문에 알맞은 자리 수로 결과가 잘립니다. 보간 도중 중간 결과가 최대 정밀도를 초과하면 수치 오버플로우가 발생하고 함수는 오류를 반환합니다. 이러한 상황을 방지하려면 정밀도가 낮은 데이터 형식을 사용하거나, 혹은 median_expression 인수를 낮은 정밀도로 변환합니다.

예를 들어 DECIMAL 인수가 포함된 SUM 함수는 38자리의 기본 정밀도를 반환합니다. 함수 결과의 비율은 인수 비율과 동일합니다. 따라서 예를 들어 DECIMAL(5,2) 열의 SUM은 DECIMAL(38,2) 데이터 형식을 반환합니다.

다음은 MEDIAN 함수의 median_expression 인수에 SUM 함수를 사용한 예입니다. PRICEPAID 열의 데이터 형식이 DECIMAL(8,2)이므로 SUM 함수는 DECIMAL(38,2)을 반환합니다.

```
select salesid, sum(pricepaid), median(sum(pricepaid))
over() from sales where salesid < 10 group by salesid;
```

잠재적 정밀도 손실이나 오버플로우 오류를 방지하려면 다음 예와 같이 함수 결과를 정밀도가 낮은 DECIMAL 데이터 형식으로 변환하는 것이 좋습니다.

```
select salesid, sum(pricepaid), median(sum(pricepaid)::decimal(30,2))
over() from sales where salesid < 10 group by salesid;
```

예시

다음은 각 판매자의 중간 판매 수량을 계산하는 예입니다.

```
select sellerid, qty, median(qty)
over (partition by sellerid)
from winsales
order by sellerid;


sellerid qty median
---------------------------
1  10 10.0
1  10 10.0
1  30 10.0
2  20 20.0
2  20 20.0
3  10 17.5
```

```
3  15 17.5
3  20 17.5
3  30 17.5
4  10 25.0
4  40 25.0
```

요청 데이터에 대한 설명은 창 함수 예제를 위한 샘플 테이블 섹션을 참조하세요.

MIN 창 함수

MIN 창 함수는 입력 표현식의 최솟값을 반환합니다. MIN 함수는 숫자 값을 사용하고 NULL 값을 무시합니다.

구문

```
MIN ( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

인수

표현식

함수가 실행되는 대상 열 또는 표현식입니다.

ALL

인수가 ALL일 때는 함수가 표현식의 모든 중복 값을 그대로 유지합니다. ALL이 기본값입니다. DISTINCT는 지원되지 않습니다.

OVER

집계 함수의 창 절을 지정합니다. OVER 절은 창 집계 함수와 일반적인 집합 집계 함수를 구분하는 역할을 합니다.

PARTITION BY expr_list

하나 이상의 표현식과 관련하여 MIN 함수의 창을 정의합니다.

ORDER BY order_list

각 파티션의 행을 정렬합니다. PARTITION BY를 지정하지 않으면 ORDER BY가 전체 테이블을 사용합니다.

frame_clause

집계 함수에서 ORDER BY 절이 사용되면 명시적인 프레임 절이 필요합니다. 프레임 절은 순서가 지정된 결과 내에 행 집합을 추가하거나 제거함으로써 함수의 창에 포함되는 행 집합을 정제하는 역할을 하며, ROWS 키워드와 관련 지정자로 구성됩니다. 창 함수 구문 요약 섹션을 참조하세요.

데이터 타입

입력값으로 모든 데이터 형식을 지원합니다. expression과 동일한 데이터 형식을 반환합니다.

예시

다음 예에서는 데이터 윈도의 시작부터 판매 ID, 수량 및 최소 수량을 보여줍니다.

```
select salesid, qty,
min(qty) over
(order by salesid rows unbounded preceding)
from winsales
order by salesid;

salesid | qty | min
---------+-----+-----
10001 |  10 |  10
10005 |  30 |  10
10006 |  10 |  10
20001 |  20 |  10
20002 |  20 |  10
30001 |  10 |  10
30003 |  15 |  10
30004 |  20 |  10
30007 |  30 |  10
40001 |  40 |  10
40005 |  10 |  10
(11 rows)
```

요청 데이터에 대한 설명은 창 함수 예제를 위한 샘플 테이블 섹션을 참조하세요.

다음 예에서는 제한적 프레임 내에서 판매 ID, 수량 및 최소 수량을 보여줍니다.

```
select salesid, qty,
min(qty) over
```

```
(order by salesid rows between 2 preceding and 1 preceding) as min
from winsales
order by salesid;

salesid | qty | min
---------+-----+-----
10001 |  10 |
10005 |  30 |  10
10006 |  10 |  10
20001 |  20 |  10
20002 |  20 |  10
30001 |  10 |  20
30003 |  15 |  10
30004 |  20 |  10
30007 |  30 |  15
40001 |  40 |  20
40005 |  10 |  30
(11 rows)
```

NTH_VALUE 창 함수

NTH_VALUE 창 함수는 창의 첫 번째 행과 관련하여 창 프레임에서 지정된 행의 표현식 값을 반환합니다.

구문

```
NTH_VALUE (expr, offset)
[ IGNORE NULLS | RESPECT NULLS ]
OVER
( [ PARTITION BY window_partition ]
[ ORDER BY window_ordering
                        frame_clause ] )
```

인수

expr

함수가 실행되는 대상 열 또는 표현식입니다.

Offset

창에서 표현식을 반환할 첫 번째 행에 대한 행 번호를 결정합니다. offset은 상수 또는 0보다 큰 양의 정수이어야 하는 표현식이 될 수 있습니다.

IGNORE NULLS

Amazon Redshift에서 사용할 행을 결정할 때 null 값을 건너뛰어야 한는 것을 의미하는 명세(옵션)입니다. IGNORE NULLS를 지정하지 않으면 NULL 값이 포함됩니다.

RESPECT NULLS

Amazon Redshift에서 사용할 행을 결정할 때 null 값을 포함시켜야 한다는 것을 의미합니다. RESPECT NULLS는 IGNORE NULLS를 지정하지 않은 경우 기본적으로 지원됩니다.

OVER

창 파티션, 순서 및 창 프레임을 지정합니다.

PARTITION BY window_partition

OVER 절에서 각 그룹의 레코드 범위를 설정합니다.

ORDER BY window_ordering

각 파티션의 행을 정렬합니다. ORDER BY를 생략하면 기본 프레임은 파티션에 속한 모든 행으로 구성됩니다.

frame_clause

집계 함수에서 ORDER BY 절이 사용되면 명시적인 프레임 절이 필요합니다. 프레임 절은 순서가 지정된 결과에 행 집합을 추가하거나 제거함으로써 함수의 창에 포함되는 행 집합을 정제하는 역할을 하며, ROWS 키워드와 관련 지정자로 구성됩니다. 창 함수 구문 요약 섹션을 참조하세요.

NTH_VALUE 윈도 함수는 Amazon Redshift 데이터 형식을 모두 사용하는 표현식을 지원합니다. 반환 형식은 expr 형식과 동일합니다.

예시

다음은 California, Florida 및 New York 주에서 세 번째로 가장 큰 장소의 좌석 수를 동일한 주에서 나머지 장소의 좌석 수와 비교하는 예입니다.

```
select venuestate, venuename, venueseats,
nth_value(venueseats, 3)
ignore nulls
over(partition by venuestate order by venueseats desc
rows between unbounded preceding and unbounded following)
as third_most_seats
```

```
from (select * from venue where venueseats > 0 and
venuestate in('CA', 'FL', 'NY'))
order by venuestate;

venuestate |           venuename            | venueseats | third_most_seats
-----------+--------------------------------+------------+------------------
CA         | Qualcomm Stadium               |      70561 |            63026
CA         | Monster Park                   |      69843 |            63026
CA         | McAfee Coliseum                |      63026 |            63026
CA         | Dodger Stadium                 |      56000 |            63026
CA         | Angel Stadium of Anaheim       |      45050 |            63026
CA         | PETCO Park                     |      42445 |            63026
CA         | AT&T Park                      |      41503 |            63026
CA         | Shoreline Amphitheatre         |      22000 |            63026
FL         | Dolphin Stadium                |      74916 |            65647
FL         | Jacksonville Municipal Stadium |      73800 |            65647
FL         | Raymond James Stadium          |      65647 |            65647
FL         | Tropicana Field                |      36048 |            65647
NY         | Ralph Wilson Stadium           |      73967 |            20000
NY         | Yankee Stadium                 |      52325 |            20000
NY         | Madison Square Garden          |      20000 |            20000
(15 rows)
```

NTILE 창 함수

NTILE 창 함수는 파티션에서 순서가 지정된 행을 최대한 같은 크기의 순위 그룹 수로 지정 분할한 후 임의의 행이 해당하는 그룹을 반환합니다.

구문

```
NTILE (expr)
OVER (
[ PARTITION BY expression_list ]
[ ORDER BY order_list ]
)
```

인수

expr

순위 그룹 수이며, 각 파티션마다 0보다 큰 양의 정수가 되어야 합니다. expr 인수가 NULL 값을 허용해서는 안 됩니다.

OVER

창 파티션 및 순서를 지정하는 절입니다. OVER 절에는 창 프레임 명세가 포함될 수 없습니다.

PARTITION BY window_partition

선택 사항. OVER 절에서 각 그룹의 레코드 범위입니다.

ORDER BY window_ordering

선택 사항. 각 파티션의 행을 정렬하는 표현식입니다. ORDER BY 절을 생략할 경우 순위 결정 방식은 동일합니다.

ORDER BY에서 고유한 순서를 지정하지 않으면 행의 순서는 비확정적입니다. 자세한 내용은 창 함수 데이터에 대한 고유 순서 지정 단원을 참조하십시오.

반환 타입

BIGINT

예시

다음은 2008년 8월 26일 Hamlet 공연 티켓 가격을 4개 순위 그룹으로 구분하는 예입니다. 결과 집합에서는 17개 행이 1순위부터 4순위까지 거의 균일하게 분할됩니다.

```
select eventname, caldate, pricepaid, ntile(4)
over(order by pricepaid desc) from sales, event, date
where sales.eventid=event.eventid and event.dateid=date.dateid and eventname='Hamlet'
and caldate='2008-08-26'
order by 4;

eventname |  caldate   | pricepaid | ntile
----------+------------+-----------+-------
Hamlet    | 2008-08-26 |   1883.00 |     1
Hamlet    | 2008-08-26 |   1065.00 |     1
Hamlet    | 2008-08-26 |    589.00 |     1
Hamlet    | 2008-08-26 |    530.00 |     1
Hamlet    | 2008-08-26 |    472.00 |     1
Hamlet    | 2008-08-26 |    460.00 |     2
Hamlet    | 2008-08-26 |    355.00 |     2
Hamlet    | 2008-08-26 |    334.00 |     2
Hamlet    | 2008-08-26 |    296.00 |     2
Hamlet    | 2008-08-26 |    230.00 |     3
```

```
Hamlet    | 2008-08-26 |    216.00 |     3
Hamlet    | 2008-08-26 |    212.00 |     3
Hamlet    | 2008-08-26 |    106.00 |     3
Hamlet    | 2008-08-26 |    100.00 |     4
Hamlet    | 2008-08-26 |     94.00 |     4
Hamlet    | 2008-08-26 |     53.00 |     4
Hamlet    | 2008-08-26 |     25.00 |     4
(17 rows)
```

PERCENT_RANK 창 함수

임의의 행의 백분율 순위를 계산합니다. 백분율 순위를 구하는 공식은 다음과 같습니다.

`(x - 1) / (the number of rows in the window or partition - 1)`

여기에서 x는 현재 행의 순위입니다. 다음은 위와 같은 공식의 사용을 나타내는 데이터 세트입니다.

```
Row# Value Rank Calculation PERCENT_RANK
1 15 1 (1-1)/(7-1) 0.0000
2 20 2 (2-1)/(7-1) 0.1666
3 20 2 (2-1)/(7-1) 0.1666
4 20 2 (2-1)/(7-1) 0.1666
5 30 5 (5-1)/(7-1) 0.6666
6 30 5 (5-1)/(7-1) 0.6666
7 40 7 (7-1)/(7-1) 1.0000
```

반환 값의 범위는 0부터 1까지입니다(0과 1 포함). 모든 집합에서 첫 번째 행은 PERCENT_RANK가 0입니다.

구문

```
PERCENT_RANK ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
)
```

인수

( )

함수에 인수가 없지만 빈 괄호가 필요합니다.

OVER

창 파티션을 지정하는 절입니다. OVER 절에는 창 프레임 명세가 포함될 수 없습니다.

PARTITION BY partition_expression

선택 사항. OVER 절에서 각 그룹의 레코드 범위를 설정하는 표현식입니다.

ORDER BY order_list

선택 사항. 백분율 순위를 계산하기 위한 표현식입니다. 이 표현식은 숫자 데이터 형식을 갖거나, 혹은 묵시적으로 1로 변환될 수 있어야 합니다. 즉 ORDER BY가 생략되면 모든 행의 반환 값은 0입니다.

ORDER BY에서 고유한 순서를 지정하지 않으면 행의 순서는 비확정적입니다. 자세한 내용은 창 함수 데이터에 대한 고유 순서 지정 단원을 참조하십시오.

반환 타입

FLOAT8

예시

다음은 각 판매자의 판매 수량에 대한 백분율 순위를 계산하는 예입니다.

```
select sellerid, qty, percent_rank()
over (partition by sellerid order by qty)
from winsales;

sellerid qty  percent_rank
----------------------------------------
1  10.00  0.0
1  10.64  0.5
1  30.37  1.0
3  10.04  0.0
3  15.15  0.33
3  20.75  0.67
3  30.55  1.0
2  20.09  0.0
2  20.12  1.0
4  10.12  0.0
4  40.23  1.0
```

요청 데이터에 대한 설명은 창 함수 예제를 위한 샘플 테이블 섹션을 참조하세요.

PERCENTILE_CONT 창 함수

PERCENTILE_CONT는 연속 분포 모델을 가정하는 역분포 함수입니다. 백분위 값과 정렬 명세를 가지며, 정렬 명세와 관련하여 지정된 백분위 값에 해당하는 보간 값을 반환합니다.

PERCENTILE_CONT는 순서가 지정된 값 사이의 선형 보간을 계산합니다. 이 함수는 집계 그룹에서 백분위 값(P)과 NULL을 제외한 행들의 번호(N)를 사용하여 정렬 명세에 따라 행의 순서를 지정한 후 행 번호를 계산합니다. 행 번호(RN)를 계산하는 공식은 RN = (1+ (P*(N-1))입니다. 이 집계 함수의 최종 결과는 행 번호가 CRN = CEILING(RN)과 FRN = FLOOR(RN)인 행의 값 사이 선형 보간을 통해 계산됩니다.

최종 결과는 다음과 같습니다.

(CRN = FRN = RN)일 때 결과는 (value of expression from row at RN)입니다.

그렇지 않다면 다음 결과가 표시됩니다.

(CRN - RN) * (value of expression for row at FRN) + (RN - FRN) * (value of expression for row at CRN).

PARTITION 절은 OVER 절에서만 지정할 수 있습니다. 각 행마다 PARTITION을 지정하면 PERCENTILE_CONT가 임의의 파티션에 속한 값 집합 중에서 지정한 백분위에 해당하는 값을 반환합니다.

구문

```
PERCENTILE_CONT ( percentile )
WITHIN GROUP (ORDER BY expr)
OVER (  [ PARTITION BY expr_list ]  )
```

인수

Percentile

0과 1 사이의 숫자 상수입니다. 이 계산에서 Null 값은 무시됩니다.

WITHIN GROUP (ORDER BY expr)

숫자 또는 날짜/시간 값을 지정하여 백분위를 정렬 및 계산합니다.

OVER

창 파티션을 지정합니다. OVER 절에는 창 순서 또는 창 프레임 명세가 포함될 수 없습니다.

PARTITION BY expr

OVER 절에서 각 그룹의 레코드 범위를 설정하는 인수(옵션)입니다.

반환

반환 형식은 WITHIN GROUP 절에서 ORDER BY 표현식의 데이터 형식에 따라 결정됩니다. 다음 표는 ORDER BY 표현식의 데이터 형식에 따른 반환 형식을 나타낸 것입니다.

입력 형식	반환 유형
INT2, INT4, INT8, NUMERIC, DECIMAL	DECIMAL
FLOAT, DOUBLE	DOUBLE
날짜	날짜
TIMESTAMP	TIMESTAMP

사용 노트

ORDER BY 표현식이 최대 정밀도가 38자리로 정의된 DECIMAL 데이터 형식인 경우에는 PERCENTILE_CONT가 부정확한 결과 또는 오류를 반환합니다. PERCENTILE_CONT 함수의 반환 값이 38자리를 초과하면 정밀도가 손실될 수도 있기 때문에 알맞은 자리 수로 결과가 잘립니다. 보간 도중 중간 결과가 최대 정밀도를 초과하면 수치 오버플로우가 발생하고 함수는 오류를 반환합니다. 이러한 상황을 방지하려면 정밀도가 낮은 데이터 형식을 사용하거나, 혹은 ORDER BY 표현식을 낮은 정밀도로 변환합니다.

예를 들어 DECIMAL 인수가 포함된 SUM 함수는 38자리의 기본 정밀도를 반환합니다. 함수 결과의 비율은 인수 비율과 동일합니다. 따라서 예를 들어 DECIMAL(5,2) 열의 SUM은 DECIMAL(38,2) 데이터 형식을 반환합니다.

다음은 PERCENTILE_CONT 함수의 ORDER BY 절에서 SUM 함수를 사용한 예입니다. PRICEPAID 열의 데이터 형식이 DECIMAL(8,2)이므로 SUM 함수는 DECIMAL(38,2)을 반환합니다.

```
select salesid, sum(pricepaid), percentile_cont(0.6)
```

```
within group (order by sum(pricepaid) desc) over()
from sales where salesid < 10 group by salesid;
```

잠재적 정밀도 손실이나 오버플로우 오류를 방지하려면 다음 예와 같이 함수 결과를 정밀도가 낮은 DECIMAL 데이터 형식으로 변환하는 것이 좋습니다.

```
select salesid, sum(pricepaid), percentile_cont(0.6)
within group (order by sum(pricepaid)::decimal(30,2) desc) over()
from sales where salesid < 10 group by salesid;
```

예시

아래 예들에서는 WINSALES 테이블을 사용합니다. 요청 데이터에 대한 설명은 창 함수 예제를 위한 샘플 테이블 섹션을 참조하세요.

```
select sellerid, qty, percentile_cont(0.5)
within group (order by qty)
over() as median from winsales;

 sellerid | qty | median
----------+-----+--------
        1 |  10 |   20.0
        1 |  10 |   20.0
        3 |  10 |   20.0
        4 |  10 |   20.0
        3 |  15 |   20.0
        2 |  20 |   20.0
        3 |  20 |   20.0
        2 |  20 |   20.0
        3 |  30 |   20.0
        1 |  30 |   20.0
        4 |  40 |   20.0
(11 rows)
```

```
select sellerid, qty, percentile_cont(0.5)
within group (order by qty)
over(partition by sellerid) as median from winsales;

 sellerid | qty | median
----------+-----+--------
        2 |  20 |   20.0
```

```
        2 |  20 |   20.0
        4 |  10 |   25.0
        4 |  40 |   25.0
        1 |  10 |   10.0
        1 |  10 |   10.0
        1 |  30 |   10.0
        3 |  10 |   17.5
        3 |  15 |   17.5
        3 |  20 |   17.5
        3 |  30 |   17.5
(11 rows)
```

다음은 Washington 주에 거주하는 판매자의 티켓 판매에 대한 PERCENTILE_CONT와 PERCENTILE_DISC를 계산하는 예입니다.

```
SELECT sellerid, state, sum(qtysold*pricepaid) sales,
percentile_cont(0.6) within group (order by sum(qtysold*pricepaid::decimal(14,2) )
 desc) over(),
percentile_disc(0.6) within group (order by sum(qtysold*pricepaid::decimal(14,2) )
 desc) over()
from sales s, users u
where s.sellerid = u.userid and state = 'WA' and sellerid < 1000
group by sellerid, state;

 sellerid | state |  sales  | percentile_cont | percentile_disc
----------+-------+---------+-----------------+-----------------
      127 | WA    | 6076.00 |         2044.20 |         1531.00
      787 | WA    | 6035.00 |         2044.20 |         1531.00
      381 | WA    | 5881.00 |         2044.20 |         1531.00
      777 | WA    | 2814.00 |         2044.20 |         1531.00
       33 | WA    | 1531.00 |         2044.20 |         1531.00
      800 | WA    | 1476.00 |         2044.20 |         1531.00
        1 | WA    | 1177.00 |         2044.20 |         1531.00
(7 rows)
```

PERCENTILE_DISC 창 함수

PERCENTILE_DISC는 이산 분포 모델을 가정하는 역분포 함수로서 백분위 값과 정렬 명세를 가지며, 지정된 집합에서 요소를 반환합니다.

임의의 백분위 값을 P라고 할 때, PERCENTILE_DISC는 ORDER BY 절의 표현식 값을 정렬한 후 동일한 정렬 명세와 관련하여 가장 작지만 P보다는 크거나 같은 누적 분포 값을 반환합니다.

PARTITION 절은 OVER 절에서만 지정할 수 있습니다.

구문

```
PERCENTILE_DISC ( percentile )
WITHIN GROUP (ORDER BY expr)
OVER (  [ PARTITION BY expr_list ]  )
```

인수

Percentile

0과 1 사이의 숫자 상수입니다. 이 계산에서 Null 값은 무시됩니다.

WITHIN GROUP (ORDER BY expr)

숫자 또는 날짜/시간 값을 지정하여 백분위를 정렬 및 계산합니다.

OVER

창 파티션을 지정합니다. OVER 절에는 창 순서 또는 창 프레임 명세가 포함될 수 없습니다.

PARTITION BY expr

OVER 절에서 각 그룹의 레코드 범위를 설정하는 인수(옵션)입니다.

반환

WITHIN GROUP 절의 ORDER BY 표현식과 동일한 데이터 형식

예시

아래 예에서는 WINSALES 테이블을 사용합니다. 요청 데이터에 대한 설명은 창 함수 예제를 위한 샘플 테이블 섹션을 참조하세요.

```
SELECT sellerid, qty, PERCENTILE_DISC(0.5)
WITHIN GROUP (ORDER BY qty)
OVER() AS MEDIAN FROM winsales;

+----------+-----+--------+
| sellerid | qty | median |
+----------+-----+--------+
| 3        | 10  | 20     |
| 1        | 10  | 20     |
```

```
| 1         | 10  | 20     |
| 4         | 10  | 20     |
| 3         | 15  | 20     |
| 2         | 20  | 20     |
| 2         | 20  | 20     |
| 3         | 20  | 20     |
| 1         | 30  | 20     |
| 3         | 30  | 20     |
| 4         | 40  | 20     |
+----------+-----+--------+

SELECT sellerid, qty, PERCENTILE_DISC(0.5)
WITHIN GROUP (ORDER BY qty)
OVER(PARTITION BY sellerid) AS MEDIAN FROM winsales;

+----------+-----+--------+
| sellerid | qty | median |
+----------+-----+--------+
| 4         | 10  | 10     |
| 4         | 40  | 10     |
| 3         | 10  | 15     |
| 3         | 15  | 15     |
| 3         | 20  | 15     |
| 3         | 30  | 15     |
| 2         | 20  | 20     |
| 2         | 20  | 20     |
| 1         | 10  | 10     |
| 1         | 10  | 10     |
| 1         | 30  | 10     |
+----------+-----+--------+
```

판매자 ID별로 구분한 수량에 대한 PERCENTILE_DISC(0.25) 및 PERCENTILE_DISC(0.75)를 찾으려면 다음 예를 사용하세요.

```
SELECT sellerid, qty, PERCENTILE_DISC(0.25)
WITHIN GROUP (ORDER BY qty)
OVER(PARTITION BY sellerid) AS quartile1 FROM winsales;

+----------+-----+-----------+
| sellerid | qty | quartile1 |
+----------+-----+-----------+
| 4         | 10  | 10        |
| 4         | 40  | 10        |
```

```
| 2         | 20  | 20         |
| 2         | 20  | 20         |
| 3         | 10  | 10         |
| 3         | 15  | 10         |
| 3         | 20  | 10         |
| 3         | 30  | 10         |
| 1         | 10  | 10         |
| 1         | 10  | 10         |
| 1         | 30  | 10         |
+-----------+-----+------------+

SELECT sellerid, qty, PERCENTILE_DISC(0.75)
WITHIN GROUP (ORDER BY qty)
OVER(PARTITION BY sellerid) AS quartile3 FROM winsales;

+-----------+-----+------------+
| sellerid | qty | quartile3 |
+-----------+-----+------------+
| 3         | 10  | 20         |
| 3         | 15  | 20         |
| 3         | 20  | 20         |
| 3         | 30  | 20         |
| 4         | 10  | 40         |
| 4         | 40  | 40         |
| 2         | 20  | 20         |
| 2         | 20  | 20         |
| 1         | 10  | 30         |
| 1         | 10  | 30         |
| 1         | 30  | 30         |
+-----------+-----+------------+
```

RANK 창 함수

RANK 창 함수는 OVER 절의 ORDER BY 표현식을 기준으로 값 그룹에 속한 값의 순위를 결정합니다. PARTITION BY 절(옵션)이 존재하면 각 행 그룹의 순위가 재설정됩니다. 순위 기준 값이 같은 행은 순위도 동일하게 결정됩니다. Amazon Redshift는 순위가 동일한 행의 수를 동일한 순위에 추가하여 다음 순위를 계산하기 때문에 순위가 연속된 수가 아닐 수도 있습니다. 예를 들어 두 행의 순위가 1로 결정되면 다음 순위는 3입니다.

RANK는 한 가지 측면에서 DENSE_RANK 창 함수와 다릅니다. 즉 DENSE_RANK에서는 2개 이상의 행에서 순위가 동일하면 순위 값의 순서에서도 빈 자리가 없습니다. 예를 들어 두 행의 순위가 1로 결정되면 다음 순위는 2입니다.

순위 함수에서는 동일한 쿼리라고 해도 PARTITION BY 절과 ORDER BY 절을 다르게 사용할 수 있습니다.

구문

```
RANK () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

인수

( )

함수에 인수가 없지만 빈 괄호가 필요합니다.

OVER

RANK 함수의 창 절입니다.

PARTITION BY expr_list

선택 사항. 창을 정의하는 하나 이상의 표현식입니다.

ORDER BY order_list

선택 사항. 순위 값의 기준이 되는 열을 정의합니다. PARTITION BY를 지정하지 않으면 ORDER BY가 전체 테이블을 사용합니다. 즉 ORDER BY가 생략되면 모든 행의 반환 값은 1입니다.

ORDER BY에서 고유한 순서를 지정하지 않으면 행의 순서는 비확정적입니다. 자세한 내용은 창 함수 데이터에 대한 고유 순서 지정 단원을 참조하십시오.

반환 타입

INTEGER

예시

다음 예에서는 판매 수량에 따라 테이블의 순서(기본 오름차순)를 지정한 후 각 행마다 순위를 할당합니다. 순위 값 1은 가장 높은 순위의 값입니다. 결과는 창 함수 결과를 적용한 후에 정렬됩니다:

```
select salesid, qty,
rank() over (order by qty) as rnk
```

```
from winsales
order by 2,1;

salesid | qty | rnk
--------+-----+-----
10001 |  10 |  1
10006 |  10 |  1
30001 |  10 |  1
40005 |  10 |  1
30003 |  15 |  5
20001 |  20 |  6
20002 |  20 |  6
30004 |  20 |  6
10005 |  30 |  9
30007 |  30 |  9
40001 |  40 |  11
(11 rows)
```

이번 예에서는 외부 ORDER BY 절에 열 2와 1이 포함되어 Amazon Redshift가 쿼리를 실행할 때마다 일관적으로 정렬된 결과를 반환할 수 있습니다. 예를 들어 판매 ID가 10001과 10006인 행은 QTY 및 RNK 값이 동일합니다. 이때 열 1에 따라 최종 결과 집합의 순서를 지정하면 10001 행이 항상 10006 행보다 앞에 위치할 수 있습니다. 요청 데이터에 대한 설명은 창 함수 예제를 위한 샘플 테이블 섹션을 참조하세요.

다음 예에서는 창 함수(`order by qty desc`)의 순서가 반전됩니다. 여기에서는 최고 순위 값이 가장 큰 QTY 값에 적용됩니다.

```
select salesid, qty,
rank() over (order by qty desc) as rank
from winsales
order by 2,1;

 salesid | qty | rank
---------+-----+-----
   10001 |  10 |   8
   10006 |  10 |   8
   30001 |  10 |   8
   40005 |  10 |   8
   30003 |  15 |   7
   20001 |  20 |   4
   20002 |  20 |   4
   30004 |  20 |   4
   10005 |  30 |   2
```

```
   30007 |  30 |   2
   40001 |  40 |   1
(11 rows)
```

요청 데이터에 대한 설명은 창 함수 예제를 위한 샘플 테이블 섹션을 참조하세요.

다음 예에서는 SELLERID를 기준으로 테이블을 분할하여 수량에 따라 각 파티션의 순서(내림차순)를 지정한 후 순위를 각 행에 할당합니다. 결과는 창 함수 결과를 적용한 후에 정렬됩니다.

```
select salesid, sellerid, qty, rank() over
(partition by sellerid
order by qty desc) as rank
from winsales
order by 2,3,1;

salesid | sellerid | qty | rank
--------+----------+-----+-----
  10001 |        1 |  10 |  2
  10006 |        1 |  10 |  2
  10005 |        1 |  30 |  1
  20001 |        2 |  20 |  1
  20002 |        2 |  20 |  1
  30001 |        3 |  10 |  4
  30003 |        3 |  15 |  3
  30004 |        3 |  20 |  2
  30007 |        3 |  30 |  1
  40005 |        4 |  10 |  2
  40001 |        4 |  40 |  1
(11 rows)
```

RATIO_TO_REPORT 창 함수

창 또는 파티션에서 값의 합에 대한 임의의 값 비율을 계산합니다. 값의 비율을 구하는 공식은 다음과 같습니다.

`value of` ratio_expression `argument for the current row / sum of` ratio_expression `argument for the window or partition`

다음은 위와 같은 공식의 사용을 나타내는 데이터 세트입니다.

```
Row# Value Calculation RATIO_TO_REPORT
1 2500 (2500)/(13900) 0.1798
```

```
2 2600 (2600)/(13900) 0.1870
3 2800 (2800)/(13900) 0.2014
4 2900 (2900)/(13900) 0.2086
5 3100 (3100)/(13900) 0.2230
```

반환 값의 범위는 0부터 1까지입니다(0과 1 포함). ratio_expression이 NULL이면 반환 값은 NULL입니다. partition_expression의 값이 고유한 경우 함수는 해당 값에 대해 1을 반환합니다.

구문

```
RATIO_TO_REPORT ( ratio_expression )
OVER ( [ PARTITION BY partition_expression ] )
```

인수

ratio_expression

비율을 결정할 값을 제공하는 표현식(열 이름 등)입니다. 이 표현식은 숫자 데이터 형식을 갖거나, 혹은 묵시적으로 1로 변환될 수 있어야 합니다.

그 외에 다른 분석 함수는 ratio_expression에서 사용할 수 없습니다.

OVER

창 파티션을 지정하는 절입니다. OVER 절에는 창 순서 또는 창 프레임 명세가 포함될 수 없습니다.

PARTITION BY partition_expression

선택 사항. OVER 절에서 각 그룹의 레코드 범위를 설정하는 표현식입니다.

반환 타입

FLOAT8

예시

아래 예에서는 WINSALES 테이블을 사용합니다. WINSALES 테이블 생성 방법에 대한 자세한 내용은 창 함수 예제를 위한 샘플 테이블 섹션을 참조하세요.

다음 예에서는 모든 셀러 수량의 합계에 대한 셀러 수량의 각 행의 보고서 대비 비율 값을 계산합니다.

```
select sellerid, qty, ratio_to_report(qty)
over()
```

```
from winsales
order by sellerid;

sellerid  qty    ratio_to_report
----------------------------------------
1         30     0.13953488372093023
1         10     0.046511627906976744
1         10     0.046511627906976744
2         20     0.09302325581395349
2         20     0.09302325581395349
3         30     0.13953488372093023
3         20     0.09302325581395349
3         15     0.06976744186046512
3         10     0.046511627906976744
4         10     0.046511627906976744
4         40     0.18604651162790697
```

다음은 파티션별로 각 판매자의 판매 수량에 대한 비율을 계산하는 예입니다.

```
select sellerid, qty, ratio_to_report(qty)
over(partition by sellerid)
from winsales;

sellerid   qty    ratio_to_report
-----------------------------------------
2          20     0.5
2          20     0.5
4          40     0.8
4          10     0.2
1          10     0.2
1          30     0.6
1          10     0.2
3          10     0.13333333333333333
3          15     0.2
3          20     0.26666666666666666
3          30     0.4
```

ROW_NUMBER 창 함수

OVER 절의 ORDER BY 표현식을 기준으로 행 그룹 내에서 1부터 현재 행의 서수를 할당합니다. PARTITION BY 절(옵션)이 존재하면 각 행 그룹의 서수가 재설정됩니다. ORDER BY 표현식 값이 동일한 행이라고 해도 비확정적으로 다른 행 번호를 받습니다.

구문

```
ROW_NUMBER() OVER(
  [ PARTITION BY expr_list ]
  [ ORDER BY order_list ]
)
```

인수

( )

함수에 인수가 없지만 빈 괄호가 필요합니다.

OVER

ROW_NUMBER 함수에 대한 창 함수 절입니다.

PARTITION BY expr_list

선택 사항. 결과를 행 집합으로 나누는 하나 이상의 열 표현식입니다.

ORDER BY order_list

선택 사항. 집합 내 행의 순서를 정의하는 하나 이상의 열 표현식입니다. PARTITION BY를 지정하지 않으면 ORDER BY가 전체 테이블을 사용합니다.

ORDER BY가 고유한 순서를 지정하지 않거나 생략되면 행의 순서는 비확정적입니다. 자세한 내용은 창 함수 데이터에 대한 고유 순서 지정 단원을 참조하십시오.

반환 타입

BIGINT

예시

다음 예제에서는 WINSALES 테이블을 사용합니다. WINSALES 테이블에 대한 설명은 창 함수 예제를 위한 샘플 테이블 섹션을 참조하세요.

다음 예제에서는 테이블을 QTY(오름차순)로 정렬한 다음 각 행에 행 번호를 할당합니다. 결과는 창 함수 결과를 적용한 후에 정렬됩니다.

```
SELECT salesid, sellerid, qty,
ROW_NUMBER() OVER(
   ORDER BY qty ASC) AS row
FROM winsales
```

```
ORDER BY 4,1;

salesid   sellerid   qty   row
---------+----------+-----+-----
   30001 |        3 |  10 |   1
   10001 |        1 |  10 |   2
   10006 |        1 |  10 |   3
   40005 |        4 |  10 |   4
   30003 |        3 |  15 |   5
   20001 |        2 |  20 |   6
   20002 |        2 |  20 |   7
   30004 |        3 |  20 |   8
   10005 |        1 |  30 |   9
   30007 |        3 |  30 |  10
   40001 |        4 |  40 |  11
```

다음은 SELLERID를 기준으로 테이블을 분할하여 수량에 따라 각 파티션의 순서(오름차순)를 지정한 후 행 번호를 각 행에 할당하는 예입니다. 결과는 창 함수 결과를 적용한 후에 정렬됩니다.

```
SELECT salesid, sellerid, qty,
ROW_NUMBER() OVER(
  PARTITION BY sellerid
  ORDER BY qty ASC) AS row_by_seller
FROM winsales
ORDER BY 2,4;

 salesid | sellerid | qty | row_by_seller
---------+----------+-----+-----
   10001 |        1 |  10 |   1
   10006 |        1 |  10 |   2
   10005 |        1 |  30 |   3
   20001 |        2 |  20 |   1
   20002 |        2 |  20 |   2
   30001 |        3 |  10 |   1
   30003 |        3 |  15 |   2
   30004 |        3 |  20 |   3
   30007 |        3 |  30 |   4
   40005 |        4 |  10 |   1
   40001 |        4 |  40 |   2
```

다음 예제에서는 선택 절을 사용하지 않을 때의 결과를 보여줍니다.

```
SELECT salesid, sellerid, qty, ROW_NUMBER() OVER() AS row
```

```
FROM winsales
ORDER BY 4,1;

salesid   sellerid   qty   row
---------+----------+-----+-----
   30001 |        3 |  10 |   1
   10001 |        1 |  10 |   2
   10005 |        1 |  30 |   3
   40001 |        4 |  40 |   4
   10006 |        1 |  10 |   5
   20001 |        2 |  20 |   6
   40005 |        4 |  10 |   7
   20002 |        2 |  20 |   8
   30003 |        3 |  15 |   9
   30004 |        3 |  20 |  10
   30007 |        3 |  30 |  11
```

STDDEV_SAMP 및 STDDEV_POP 창 함수

STDDEV_SAMP 및 STDDEV_POP 창 함수는 숫자 값(정수, 소수 또는 부동 소수점) 집합의 표본 표준 편차와 모 표준 편차를 반환합니다. 또한 STDDEV_SAMP 및 STDDEV_POP 함수 섹션도 참조하세요.

STDDEV_SAMP와 STDDEV는 동일한 함수이기 때문에 동의어나 마찬가지입니다.

구문

```
STDDEV_SAMP | STDDEV | STDDEV_POP
( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

인수

표현식

함수가 실행되는 대상 열 또는 표현식입니다.

ALL

인수가 ALL일 때는 함수가 표현식의 모든 중복 값을 그대로 유지합니다. ALL이 기본값입니다. DISTINCT는 지원되지 않습니다.

OVER

집계 함수의 창 절을 지정합니다. OVER 절은 창 집계 함수와 일반적인 집합 집계 함수를 구분하는 역할을 합니다.

PARTITION BY expr_list

하나 이상의 표현식과 관련하여 함수의 창을 정의합니다.

ORDER BY order_list

각 파티션의 행을 정렬합니다. PARTITION BY를 지정하지 않으면 ORDER BY가 전체 테이블을 사용합니다.

frame_clause

집계 함수에서 ORDER BY 절이 사용되면 명시적인 프레임 절이 필요합니다. 프레임 절은 순서가 지정된 결과 내에 행 집합을 추가하거나 제거함으로써 함수의 창에 포함되는 행 집합을 정제하는 역할을 하며, ROWS 키워드와 관련 지정자로 구성됩니다. 창 함수 구문 요약 섹션을 참조하세요.

데이터 타입

STDDEV 함수에서 지원되는 인수 형식은 SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL, REAL, DOUBLE PRECISION입니다.

표현식의 데이터 형식과 상관없이 STDDEV 함수의 반환 형식은 배정밀도 숫자입니다.

예시

다음은 STDDEV_POP 및 VAR_POP 함수를 창 함수로 사용하는 방법을 나타낸 예입니다. 쿼리가 SALES 테이블의 PRICEPAID 값에 대한 모 분산과 모 표준 편차를 계산합니다.

```
select salesid, dateid, pricepaid,
round(stddev_pop(pricepaid) over
(order by dateid, salesid rows unbounded preceding)) as stddevpop,
round(var_pop(pricepaid) over
(order by dateid, salesid rows unbounded preceding)) as varpop
from sales
order by 2,1;

salesid | dateid | pricepaid | stddevpop | varpop
--------+--------+-----------+-----------+---------
  33095 |   1827 |    234.00 |         0 |       0
  65082 |   1827 |    472.00 |       119 |   14161
```

```
  88268 |   1827 |    836.00 |       248 |   61283
  97197 |   1827 |    708.00 |       230 |   53019
 110328 |   1827 |    347.00 |       223 |   49845
 110917 |   1827 |    337.00 |       215 |   46159
 150314 |   1827 |    688.00 |       211 |   44414
 157751 |   1827 |   1730.00 |       447 |  199679
 165890 |   1827 |   4192.00 |      1185 | 1403323
...
```

표본 표준 편차 및 분산 함수 역시 같은 방식으로 사용할 수 있습니다.

SUM 창 함수

SUM 창 함수는 입력 열 또는 표현식 값의 합을 반환합니다. SUM 함수는 숫자 값을 사용하고 NULL 값을 무시합니다.

구문

```
SUM ( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

인수

표현식

함수가 실행되는 대상 열 또는 표현식입니다.

ALL

인수가 ALL일 때는 함수가 표현식의 모든 중복 값을 그대로 유지합니다. ALL이 기본값입니다. DISTINCT는 지원되지 않습니다.

OVER

집계 함수의 창 절을 지정합니다. OVER 절은 창 집계 함수와 일반적인 집합 집계 함수를 구분하는 역할을 합니다.

PARTITION BY expr_list

하나 이상의 표현식과 관련하여 SUM 함수의 창을 정의합니다.

ORDER BY order_list

각 파티션의 행을 정렬합니다. PARTITION BY를 지정하지 않으면 ORDER BY가 전체 테이블을 사용합니다.

frame_clause

집계 함수에서 ORDER BY 절이 사용되면 명시적인 프레임 절이 필요합니다. 프레임 절은 순서가 지정된 결과 내에 행 집합을 추가하거나 제거함으로써 함수의 창에 포함되는 행 집합을 정제하는 역할을 하며, ROWS 키워드와 관련 지정자로 구성됩니다. 창 함수 구문 요약 섹션을 참조하세요.

데이터 타입

SUM 함수에서 지원되는 인수 형식은 SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL, REAL, DOUBLE PRECISION입니다.

SUM 함수에서 지원되는 반환 형식은 다음과 같습니다.

- SMALLINT 또는 INTEGER 인수일 때 BIGINT
- BIGINT 인수일 때 NUMERIC
- 부동 소수점 인수일 때 DOUBLE PRECISION

예시

다음 예에서는 날짜 및 판매 ID에 따라 순서가 지정된 판매 수량의 누적(롤링) 합을 생성합니다.

```
select salesid, dateid, sellerid, qty,
sum(qty) over (order by dateid, salesid rows unbounded preceding) as sum
from winsales
order by 2,1;

salesid |   dateid   | sellerid | qty | sum
---------+------------+----------+-----+-----
30001 | 2003-08-02 |        3 |  10 |  10
10001 | 2003-12-24 |        1 |  10 |  20
10005 | 2003-12-24 |        1 |  30 |  50
40001 | 2004-01-09 |        4 |  40 |  90
10006 | 2004-01-18 |        1 |  10 | 100
20001 | 2004-02-12 |        2 |  20 | 120
40005 | 2004-02-12 |        4 |  10 | 130
20002 | 2004-02-16 |        2 |  20 | 150
```

```
30003 | 2004-04-18 |        3 |  15 | 165
30004 | 2004-04-18 |        3 |  20 | 185
30007 | 2004-09-07 |        3 |  30 | 215
(11 rows)
```

요청 데이터에 대한 설명은 창 함수 예제를 위한 샘플 테이블 섹션을 참조하세요.

다음 예에서는 날짜별 판매 수량에 대한 누적(롤링) 합을 생성하고 판매자 ID를 기준으로 그 결과를 분할한 다음 파티션 내에서 날짜 및 판매 ID에 따라 결과의 순서를 지정합니다.

```
select salesid, dateid, sellerid, qty,
sum(qty) over (partition by sellerid
order by dateid, salesid rows unbounded preceding) as sum
from winsales
order by 2,1;

salesid |   dateid   | sellerid | qty | sum
---------+------------+----------+-----+-----
30001 | 2003-08-02 |        3 |  10 |  10
10001 | 2003-12-24 |        1 |  10 |  10
10005 | 2003-12-24 |        1 |  30 |  40
40001 | 2004-01-09 |        4 |  40 |  40
10006 | 2004-01-18 |        1 |  10 |  50
20001 | 2004-02-12 |        2 |  20 |  20
40005 | 2004-02-12 |        4 |  10 |  50
20002 | 2004-02-16 |        2 |  20 |  40
30003 | 2004-04-18 |        3 |  15 |  25
30004 | 2004-04-18 |        3 |  20 |  45
30007 | 2004-09-07 |        3 |  30 |  75
(11 rows)
```

다음 예에서는 결과 집합의 모든 행에 SELLERID 및 SALESID 열을 기준으로 순서대로 번호를 지정합니다.

```
select salesid, sellerid, qty,
sum(1) over (order by sellerid, salesid rows unbounded preceding) as rownum
from winsales
order by 2,1;

salesid | sellerid |  qty | rownum
--------+----------+------+--------
10001 |        1 |   10 |     1
10005 |        1 |   30 |     2
```

```
10006 |        1 |   10 |     3
20001 |        2 |   20 |     4
20002 |        2 |   20 |     5
30001 |        3 |   10 |     6
30003 |        3 |   15 |     7
30004 |        3 |   20 |     8
30007 |        3 |   30 |     9
40001 |        4 |   40 |    10
40005 |        4 |   10 |    11
(11 rows)
```

요청 데이터에 대한 설명은 창 함수 예제를 위한 샘플 테이블 섹션을 참조하세요.

다음 예에서는 결과 집합의 모든 행에 순차적으로 번호를 매기고, 결과를 SELLERID로 분할하고, 파티션 내에서 SELLERID 및 SALESID로 결과를 정렬합니다.

```
select salesid, sellerid, qty,
sum(1) over (partition by sellerid
order by sellerid, salesid rows unbounded preceding) as rownum
from winsales
order by 2,1;

salesid | sellerid | qty | rownum
---------+----------+-----+--------
10001 |        1 |  10 |      1
10005 |        1 |  30 |      2
10006 |        1 |  10 |      3
20001 |        2 |  20 |      1
20002 |        2 |  20 |      2
30001 |        3 |  10 |      1
30003 |        3 |  15 |      2
30004 |        3 |  20 |      3
30007 |        3 |  30 |      4
40001 |        4 |  40 |      1
40005 |        4 |  10 |      2
(11 rows)
```

VAR_SAMP 및 VAR_POP 창 함수

VAR_SAMP 및 VAR_POP 창 함수는 숫자 값(정수, 소수 또는 부동 소수점) 집합의 표본 분산과 모 분산을 반환합니다. 또한 VAR_SAMP 및 VAR_POP 함수 섹션도 참조하세요.

VAR_SAMP 및 VARIANCE는 동일한 함수이기 때문에 동의어나 마찬가지입니다.

구문

```
VAR_SAMP | VARIANCE | VAR_POP
( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

인수

표현식

함수가 실행되는 대상 열 또는 표현식입니다.

ALL

인수가 ALL일 때는 함수가 표현식의 모든 중복 값을 그대로 유지합니다. ALL이 기본값입니다. DISTINCT는 지원되지 않습니다.

OVER

집계 함수의 창 절을 지정합니다. OVER 절은 창 집계 함수와 일반적인 집합 집계 함수를 구분하는 역할을 합니다.

PARTITION BY expr_list

하나 이상의 표현식과 관련하여 함수의 창을 정의합니다.

ORDER BY order_list

각 파티션의 행을 정렬합니다. PARTITION BY를 지정하지 않으면 ORDER BY가 전체 테이블을 사용합니다.

frame_clause

집계 함수에서 ORDER BY 절이 사용되면 명시적인 프레임 절이 필요합니다. 프레임 절은 순서가 지정된 결과 내에 행 집합을 추가하거나 제거함으로써 함수의 창에 포함되는 행 집합을 정제하는 역할을 하며, ROWS 키워드와 관련 지정자로 구성됩니다. 창 함수 구문 요약 섹션을 참조하세요.

데이터 타입

VARIANCE 함수에서 지원되는 인수 형식은 SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL, REAL, DOUBLE PRECISION입니다.

표현식의 데이터 형식과 상관없이 VARIANCE 함수의 반환 형식은 배정밀도 숫자입니다.

시스템 관리 함수

주제

Amazon Redshift는 여러 시스템 관리 함수를 지원합니다.

CHANGE_QUERY_PRIORITY

CHANGE_QUERY_PRIORITY를 통해 수퍼유저는 워크로드 관리(WLM)에서 실행 또는 대기 중인 쿼리의 우선 순위를 수정할 수 있습니다.

이 기능을 통해 수퍼유저는 시스템 내 모든 쿼리의 우선 순위를 즉시 변경할 수 있습니다. 하나의 쿼리, 사용자 또는 세션만 `CRITICAL` 우선 순위로 실행할 수 있습니다.

구문

```
CHANGE_QUERY_PRIORITY(query_id, priority)
```

인수

query_id

우선 순위가 변경된 쿼리의 쿼리 식별자입니다. `INTEGER` 값이 필요합니다.

우선순위

쿼리에 지정할 새로운 우선 순위입니다. 이 인수는 값이`CRITICAL`, `HIGHEST`, `HIGH`, `NORMAL`, `LOW` 또는 `LOWEST`인 문자열이어야 합니다.

반환 유형

없음

예시

STV_WLM_QUERY_STATE 시스템 테이블에 있는 query_priority 열을 표시하려면 다음 예제를 사용합니다.

```
SELECT query, service_class, query_priority, state
FROM stv_wlm_query_state WHERE service_class = 101;

+-------+---------------+----------------+---------+
| query | service_class | query_priority |  state  |
+-------+---------------+----------------+---------+
|  1076 |           101 | Lowest         | Running |
|  1075 |           101 | Lowest         | Running |
+-------+---------------+----------------+---------+
```

우선 순위를 CRITICAL로 변경하기 위해 change_query_priority 함수를 실행한 슈퍼유저가 얻은 결과를 표시하려면 다음 예제를 사용합니다.

```
SELECT CHANGE_QUERY_PRIORITY(1076, 'Critical');

+-------------------------------------------------------------------------------+
|                            change_query_priority                              |
+-------------------------------------------------------------------------------+
| Succeeded to change query priority. Priority changed from Lowest to Critical. |
+-------------------------------------------------------------------------------+
```

CHANGE_SESSION_PRIORITY

CHANGE_SESSION_PRIORITY를 통해 수퍼유저는 시스템 내 모든 세션의 우선 순위를 즉시 변경할 수 있습니다. 하나의 세션, 사용자 또는 쿼리만 CRITICAL 우선 순위로 실행할 수 있습니다.

구문

```
CHANGE_SESSION_PRIORITY(pid, priority)
```

인수

pid

우선 순위가 변경된 세션의 프로세스 식별자입니다. -1 값은 현재 세션을 참조합니다. INTEGER 값이 필요합니다.

우선순위

세션에 지정할 새로운 우선 순위입니다. 이 인수는 값이CRITICAL, HIGHEST, HIGH, NORMAL, LOW 또는 LOWEST인 문자열이어야 합니다.

반환 타입

없음

예시

현재 세션을 처리하는 서버 프로세스의 프로세스 식별자를 반환하려면 다음 예제를 사용합니다.

```
SELECT pg_backend_pid();

+----------------+
| pg_backend_pid |
+----------------+
|          30311 |
+----------------+
```

이 예제에서는 현재 세션의 우선 순위는 LOWEST로 변경됩니다.

```
SELECT CHANGE_SESSION_PRIORITY(30311, 'Lowest');

+-------------------------------------------------------------------------------------
+
|                                change_session_priority
 |
+-------------------------------------------------------------------------------------
+
| Succeeded to change session priority. Changed session (pid:30311) priority to lowest.
 |
+-------------------------------------------------------------------------------------
+
```

이 예제에서는 현재 세션의 우선 순위는 HIGH로 변경됩니다.

```
SELECT CHANGE_SESSION_PRIORITY(-1, 'High');

+-----------------------------------------------------------------------------------------------------
+
|                                    change_session_priority
           |
+-----------------------------------------------------------------------------------------------------
+
| Succeeded to change session priority. Changed session (pid:30311) priority from
 lowest to high. |
+-----------------------------------------------------------------------------------------------------
+
```

세션 우선 순위를 변경하는 저장 프로시저를 만들려면 다음 예제를 사용합니다. 이 저장 프로시저를 실행할 수 있는 권한은 데이터베이스 사용자 test_user에게 부여됩니다.

```
CREATE OR REPLACE PROCEDURE sp_priority_low(pid IN int, result OUT varchar)
AS $$
BEGIN
  SELECT CHANGE_SESSION_PRIORITY(pid, 'low') into result;
END;
$$ LANGUAGE plpgsql
SECURITY DEFINER;
GRANT EXECUTE ON PROCEDURE sp_priority_low(int) TO test_user;
```

그러면 test_user라는 이름의 데이터베이스 사용자는 해당 프로시저를 호출합니다.

```
CALL sp_priority_low(pg_backend_pid());

+-------------------------------------------------------+
|                        result                         |
+-------------------------------------------------------+
| Success. Change session (pid:13155) priority to low. |
+-------------------------------------------------------+
```

CHANGE_USER_PRIORITY

CHANGE_QUERY_PRIORITY를 통해 수퍼유저는 사용자가 발행한, 워크로드 관리(WLM)에서 실행 또는 대기 중인 모든 쿼리의 우선 순위를 수정할 수 있습니다. 하나의 사용자, 세션 또는 쿼리만 CRITICAL 우선 순위로 실행할 수 있습니다.

구문

```
CHANGE_USER_PRIORITY(user_name, priority)
```

인수

user_name

쿼리 우선 순위가 변경된 데이터베이스 사용자 이름입니다.

우선순위

user_name이 발행한 모든 쿼리에 지정할 새로운 우선 순위입니다. 이 인수는 값이 CRITICAL, HIGHEST, HIGH, NORMAL, LOW, LOWEST 또는 RESET인 문자열이어야 합니다. 수퍼유저만 우선 순위를 CRITICAL로 변경할 수 있습니다. 우선 순위를 RESET으로 변경하면 user_name에 대한 우선 순위 설정이 제거됩니다.

반환 타입

없음

예시

사용자 analysis_user에 대한 우선 순위를 LOWEST로 변경하려면 다음 예제를 사용합니다.

```
SELECT CHANGE_USER_PRIORITY('analysis_user', 'lowest');

+-------------------------------------------------------------------------------------+
|                                change_user_priority                                 |
+-------------------------------------------------------------------------------------+
| Succeeded to change user priority. Changed user (analysis_user) priority to lowest. |
+-------------------------------------------------------------------------------------+
```

우선 순위를 LOW로 변경하려면 다음 예제를 사용합니다.

```
SELECT CHANGE_USER_PRIORITY('analysis_user', 'low');

+-------------------------------------------------------------------------------------------------------
+
|                                    change_user_priority
        |
+-------------------------------------------------------------------------------------------------------
+
```

```
| Succeeded to change user priority. Changed user (analysis_user) priority from Lowest
 to low. |
+-----------------------------------------------------------------------------------------
+
```

우선 순위를 재설정하려면 다음 예제를 사용합니다.

```
SELECT CHANGE_USER_PRIORITY('analysis_user', 'reset');

+-----------------------------------------------------+
|                change_user_priority                 |
+-----------------------------------------------------+
| Succeeded to reset priority for user (analysis_user). |
+-----------------------------------------------------+
```

CURRENT_SETTING

CURRENT_SETTING은 지정한 구성 파라미터의 현재 값을 반환합니다.

이 함수는 SET 명령과 동일합니다.

구문

```
current_setting('parameter')
```

다음 문은 지정한 세션 컨텍스트 변수의 현재 값을 반환합니다.

```
current_setting('variable_name')
current_setting('variable_name'[, error_if_undefined])
```

인수

파라미터

설정할 파라미터 값. 구성 파라미터 목록은 구성 참조 섹션을 참조하세요.

variable_name

표시할 변수의 이름입니다. 세션 컨텍스트 변수의 문자열 상수여야 합니다.

error_if_undefined

(선택) 변수 이름이 없는 경우 동작을 지정하는 선택적 부울 값입니다. error_if_undefined가 기본값인 TRUE로 설정되면 Amazon Redshift에서 오류가 발생합니다. error_if_undefined가 FALSE로 설

정되면 Amazon Redshift에서 NULL이 반환됩니다. Amazon Redshift는 세션 컨텍스트 변수에 대해서만 error_if_undefined 파라미터를 지원합니다. 입력이 구성 파라미터인 경우에는 사용할 수 없습니다.

반환 타입

CHAR 또는 VARCHAR 문자열을 반환합니다.

예시

query_group 파라미터의 현재 설정을 반환하려면 다음 예제를 사용합니다.

```
SELECT CURRENT_SETTING('query_group');

+-----------------+
| current_setting |
+-----------------+
| unset           |
+-----------------+
```

app_context.user_id 변수의 현재 설정을 반환하려면 다음 예제를 사용합니다.

```
SELECT CURRENT_SETTING('app_context.user_id', FALSE);
```

PG_CANCEL_BACKEND

쿼리를 취소합니다. PG_CANCEL_BACKEND는 기능 면에서 CANCEL 명령과 동일합니다. 사용자가 현재 실행하고 있는 쿼리를 취소할 수 있습니다. 수퍼유저는 어떠한 쿼리든 취소할 수 있습니다.

구문

```
pg_cancel_backend( pid )
```

인수

pid

취소할 쿼리의 프로세스 ID(PID)입니다. 쿼리 ID를 지정해서는 쿼리를 취소할 수 없습니다. 반드시 쿼리의 프로세스 ID를 지정해야 합니다. INTEGER 값이 필요합니다.

반환 타입

없음

사용 노트

다수의 세션에서 동일한 테이블에 대해 쿼리를 실행하면서 잠금 현상이 발생하는 경우에는 PG_TERMINATE_BACKEND 함수를 사용해 세션 중 하나를 종료할 수 있습니다. 그러면 종료된 세션에서 실행 중이던 트랜잭션이 모든 잠금을 강제로 해제하여 트랜잭션을 롤백시킵니다. PG__LOCKS 카탈로그 테이블에 대해 쿼리를 실행하여 현재 잠금 상태를 확인합니다. 쿼리가 트랜잭션 블록(BEGIN ... END) 내부여서 취소할 수 없는 경우에는 PG_TERMINATE_BACKEND 함수를 사용하여 쿼리가 실행 중인 세션을 종료할 수 있습니다.

예시

현재 실행 중인 쿼리를 취소하려면 먼저 취소하려는 쿼리에 대한 프로세스 ID를 검색하십시오. 현재 실행 중인 모든 쿼리의 프로세스 ID를 확인하려면 다음 명령을 실행합니다.

```
SELECT pid, TRIM(starttime) AS start,
duration, TRIM(user_name) AS user,
SUBSTRING(query,1,40) AS querytxt
FROM stv_recents
WHERE status = 'Running';

+-----+-----------------------+----------+--------+-----------------------------+
| pid |       starttime       | duration |  user  |          querytxt           |
+-----+-----------------------+----------+--------+-----------------------------+
| 802 | 2013-10-14 09:19:03.55 |      132 | dwuser | select venuename from venue |
| 834 | 2013-10-14 08:33:49.47 |  1250414 | dwuser | select * from listing;      |
| 964 | 2013-10-14 08:30:43.29 |   326179 | dwuser | select sellerid from sales  |
+-----+-----------------------+----------+--------+-----------------------------+
```

프로세스 ID 802로 쿼리를 취소하려면 다음 예제를 사용합니다.

```
SELECT PG_CANCEL_BACKEND(802);
```

PG_TERMINATE_BACKEND

세션을 종료합니다. 사용자가 소유하고 있는 세션을 종료할 수 있으며, 그 밖에 수퍼유저라면 모든 세션을 종료할 수 있습니다.

구문

```
pg_terminate_backend( pid )
```

인수

pid

종료할 세션의 프로세스 ID입니다. INTEGER 값이 필요합니다.

반환 타입

없음

사용 노트

동시 접속 제한에 가깝게 도달하면 PG_TERMINATE_BACKEND를 사용하여 유휴 세션을 종료하고 접속할 수 있는 여유 세션을 확보할 수 있습니다. 자세한 내용은 Amazon Redshift의 제한을 참조하세요.

다수의 세션에서 동일한 테이블에 대해 쿼리를 실행하면서 잠금 현상이 발생하는 경우에는 PG_TERMINATE_BACKEND 함수를 사용해 세션 중 하나를 종료할 수 있습니다. 그러면 종료된 세션에서 실행 중이던 트랜잭션이 모든 잠금을 강제로 해제하여 트랜잭션을 롤백시킵니다. PG_LOCKS 카탈로그 테이블에 대해 쿼리를 실행하여 현재 잠금 상태를 확인합니다.

쿼리가 트랜잭션 블록(BEGIN … END) 내부에 있지 않으면 CANCEL 명령 또는 PG_CANCEL_BACKEND 함수를 사용해 쿼리를 취소할 수 있습니다.

예시

SVV_TRANSACTIONS 테이블에 대해 쿼리를 실행하여 현재 트랜잭션에 적용되어 있는 모든 잠금을 표시하려면 다음 예제를 사용합니다.

```
SELECT * FROM svv_transactions;

+----------+--------+-------+------+--------------------+-----------------
+----------------------+----------+---------+
| txn_owner | txn_db |  xid  | pid  |      txn_start      |    lock_mode    |
 lockable_object_type | relation | granted |
+----------+--------+-------+------+--------------------+-----------------
+----------------------+----------+---------+
```

```
| rsuser    | dev    | 96178 | 8585 | 2017-04-12 20:13:07 | AccessShareLock | relation
            |    51940 | true    |
| rsuser    | dev    | 96178 | 8585 | 2017-04-12 20:13:07 | AccessShareLock | relation
            |    52000 | true    |
| rsuser    | dev    | 96178 | 8585 | 2017-04-12 20:13:07 | AccessShareLock | relation
            |   108623 | true    |
| rsuser    | dev    | 96178 | 8585 | 2017-04-12 20:13:07 | ExclusiveLock   |
 transactionid        |          | true    |
+-----------+--------+-------+------+---------------------+-----------------
+----------------------+----------+---------+
```

잠금이 설정된 세션을 종료하려면 다음 예제를 사용합니다.

```
SELECT PG_TERMINATE_BACKEND(8585);
```

REBOOT_CLUSTER

클러스터에 대한 연결을 닫지 않고 Amazon Redshift 클러스터를 재부팅합니다. 이 명령을 실행하려면 데이터베이스 슈퍼 사용자여야 합니다.

이 소프트 재부팅이 완료된 후 Amazon Redshift 클러스터는 사용자 애플리케이션에 오류를 반환하고 사용자 애플리케이션은 소프트 재부팅으로 인해 중단된 트랜잭션이나 쿼리를 다시 제출해야 합니다.

구문

```
SELECT REBOOT_CLUSTER();
```

SET_CONFIG

구성 파라미터를 새로운 값으로 설정합니다.

이 함수는 SQL의 SET 명령과 동일합니다.

구문

```
SET_CONFIG('parameter', 'new_value' , is_local)
```

다음 문은 세션 컨텍스트 변수를 새로운 설정으로 설정합니다.

```
set_config('variable_name', 'new_value' , is_local)
```

인수

파라미터

설정할 파라미터입니다.

variable_name

설정할 변수의 이름입니다.

새 값

파라미터의 새 값입니다.

is_local

true이면 파라미터 값이 현재 트랜잭션에만 적용됩니다. 유효 값은 `true` 또는 `1`과 `false` 또는 `0`입니다.

반환 타입

CHAR 또는 VARCHAR 문자열을 반환합니다.

예시

현재 트랜잭션에서만 `query_group` 파리미터 값을 `test`로 설정하려면 다음 예제를 사용합니다.

```
SELECT SET_CONFIG('query_group', 'test', true);

+------------+
| set_config |
+------------+
| test       |
+------------+
```

세션 컨텍스트 변수를 설정하려면 다음 예제를 사용합니다.

```
SELECT SET_CONFIG(‘app.username’, ‘cuddy’, FALSE);
```

시스템 정보 함수

Amazon Redshift는 다양한 시스템 정보 함수를 지원합니다.

주제

- CURRENT_AWS_ACCOUNT
- CURRENT_DATABASE
- CURRENT_NAMESPACE
- CURRENT_SCHEMA
- CURRENT_SCHEMAS
- CURRENT_SESSION_ARN
- CURRENT_USER
- CURRENT_USER_ID
- DEFAULT_IAM_ROLE
- GET_MOUNTED_ROLE
- HAS_ASSUMEROLE_PRIVILEGE
- HAS_DATABASE_PRIVILEGE
- HAS_SCHEMA_PRIVILEGE
- HAS_TABLE_PRIVILEGE
- LAST_USER_QUERY_ID
- PG_BACKEND_PID
- PG_GET_COLS
- PG_GET_GRANTEE_BY_IAM_ROLE
- PG_GET_IAM_ROLE_BY_USER
- PG_GET_LATE_BINDING_VIEW_COLS
- PG_GET_SESSION_ROLES
- PG_LAST_COPY_COUNT
- PG_LAST_COPY_ID
- PG_LAST_UNLOAD_ID
- PG_LAST_QUERY_ID
- PG_LAST_UNLOAD_COUNT
- SLICE_NUM 함수
- USER
- ROLE_IS_MEMBER_OF
- USER_IS_MEMBER_OF

- version

CURRENT_AWS_ACCOUNT

쿼리를 제출한 Amazon Redshift 클러스터와 연결된 AWS 계정을 반환합니다.

구문

```
current_aws_account
```

반환 타입

정수를 반환합니다.

예제

다음은 현재 데이터베이스 이름을 반환하는 쿼리입니다.

```
select user, current_aws_account;
current_user | current_account
-------------+---------------
dwuser       | 987654321

(1 row)
```

CURRENT_DATABASE

현재 연결 중인 데이터베이스 이름을 반환합니다.

구문

```
current_database()
```

반환 타입

CHAR 또는 VARCHAR 문자열을 반환합니다.

예제

다음은 현재 데이터베이스 이름을 반환하는 쿼리입니다.

```
select current_database();
```

```
current_database
------------------
tickit
(1 row)
```

CURRENT_NAMESPACE

현재 Amazon Redshift 클러스터의 클러스터 네임스페이스를 반환합니다. Amazon Redshift 클러스터 네임스페이스는 Amazon Redshift 클러스터의 고유 ID입니다.

구문

```
current_namespace
```

반환 타입

CHAR 또는 VARCHAR 문자열을 반환합니다.

예제

다음은 현재 네임스페이스 이름을 반환하는 쿼리입니다.

```
select user, current_namespace;
current_user | current_namespace
-------------+--------------------------------------
dwuser       | 86b5169f-01dc-4a6f-9fbb-e2e24359e9a8

(1 row)
```

CURRENT_SCHEMA

검색 경로 앞에 있는 스키마 이름을 반환합니다. 이 스키마는 모든 테이블에, 또는 대상 스키마를 지정하지 않고 생성되는 다른 이름의 객체에 사용됩니다.

구문

> **Note**
>
> CURRENT_SCHEMA는 리더 노드 함수이기 때문에 사용자 생성 테이블, STL 또는 STV 시스템 테이블, SVV 또는 SVL 시스템 뷰를 참조하는 경우에는 오류를 반환합니다.

```
current_schema()
```

반환 타입

CURRENT_SCHEMA는 CHAR 또는 VARCHAR 문자열을 반환합니다.

예시

다음은 현재 스키마를 반환하는 쿼리입니다.

```
select current_schema();

current_schema
---------------
public
(1 row)
```

CURRENT_SCHEMAS

현재 검색 경로에서 모든 스키마의 이름 배열을 반환합니다. 현재 검색 경로는 search_path 파라미터에서 정의합니다.

구문

> **Note**
>
> CURRENT_SCHEMA는 리더 노드 함수이기 때문에 사용자 생성 테이블, STL 또는 STV 시스템 테이블, SVV 또는 SVL 시스템 뷰를 참조하는 경우에는 오류를 반환합니다.

```
current_schemas(include_implicit)
```

인수

include_implicit

true이면 검색 경로에 묵시적으로 포함된 시스템 스키마까지 모두 추가하도록 지정합니다. 유효한 값은 `true` 및 `false`입니다. 일반적으로 `true`일 때는 현재 스키마 외에 pg_catalog 스키마까지 반환합니다.

반환 타입

CHAR 또는 VARCHAR 문자열을 반환합니다.

예시

다음은 현재 검색 경로에서 묵시적으로 포함된 시스템 스키마를 제외하고 스키마 이름을 반환하는 예입니다.

```
select current_schemas(false);

current_schemas
-----------------
{public}
(1 row)
```

다음은 현재 검색 경로에서 묵시적으로 포함된 시스템 스키마까지 추가하여 스키마 이름을 반환하는 예입니다.

```
select current_schemas(true);

current_schemas
---------------------
{pg_catalog,public}
(1 row)
```

CURRENT_SESSION_ARN

현재 권한이 부여된 글로벌 사용자의 ARN을 반환합니다. 글로벌 사용자는 Redshift 계정, 클러스터, Serverless 작업 그룹에서 동일한 자격 증명으로 존재합니다. 글로벌 사용자는 IAM Identity Center를 통해 로그인하거나 IAM 기반 세션 인증을 통해 로그인합니다. 데이터 레이크 사용자는 글로벌 AWS 사용자입니다.

이 기능은 일반적으로 다중 언어 AWS Glue 뷰를 사용하는 상황에서 사용됩니다. IAM Identity Center 및 Redshift를 사용한 자격 증명 관리에 대한 자세한 내용은 Redshift를 IAM Identity Center와 연결하여 사용자에게 Single Sign-On 경험을 제공합니다를 참조하세요. 다중 언어 Glue 뷰에 대한 자세한 내용은 AWS Glue 데이터 카탈로그에서 뷰 생성을 참조하세요.

구문

```
current_session_arn()
```

반환 타입

전역적으로 인증된 사용자의 VARCHAR 문자열 또는 null 값을 반환합니다.

사용 노트

로컬 사용자는 지원되지 않으며 null 응답이 반환됩니다.

예제

다음은 현재 세션 ARN 이름을 반환하는 쿼리입니다.

```
SELECT current_session_arn();

current_session_arn
--------------
arn:aws:iam::123456789012:user/user
(1 row)
```

CURRENT_USER

검사 권한에 따라 데이터베이스의 현재 "유효" 사용자 이름을 반환합니다. 일반적으로 이 사용자 이름은 세션 사용자와 동일하지만 경우에 따라 수퍼유저에 의해 변경될 수 있습니다.

> **Note**
>
> CURRENT_USER를 호출할 때는 후행 괄호를 사용하지 마십시오.

구문

```
current_user
```

반환 타입

CURRENT_USER는 NAME 데이터 유형을 반환하며 CHAR 또는 VARCHAR 문자열로 캐스팅할 수 있습니다.

사용 노트

CREATE_PROCEDURE 명령의 SECURITY DEFINER 옵션을 사용하여 저장 프로시저를 생성한 경우, 저장 프로시저 내에서 CURRENT_USER 함수를 호출하면 Amazon Redshift는 저장 프로시저 소유자의 사용자 이름을 반환합니다.

예제

다음은 현재 데이터베이스 사용자의 이름을 반환하는 쿼리입니다.

```
select current_user;

current_user
--------------
dwuser
(1 row)
```

CURRENT_USER_ID

현재 세션에 로그인되어 있는 Amazon Redshift 사용자의 고유 식별자를 반환합니다.

구문

```
CURRENT_USER_ID
```

반환 타입

CURRENT_USER_ID 함수는 정수를 반환합니다.

예시

다음은 이 세션의 현재 사용자 이름 및 ID를 반환하는 예입니다.

```
select user, current_user_id;

 current_user | current_user_id
--------------+-----------------
   dwuser     |               1
(1 row)
```

DEFAULT_IAM_ROLE

현재 Amazon Redshift 클러스터와 연결된 기본 IAM 역할을 반환합니다. 연결된 기본 IAM 역할이 없는 경우 함수는 없음을 반환합니다.

구문

```
select default_iam_role();
```

반환 타입

VARCHAR 문자열을 반환합니다.

예제

다음 예에서는 현재 지정된 Amazon Redshift 클러스터와 연결된 기본 IAM 역할을 반환합니다.

```
select default_iam_role();
              default_iam_role
-----------------------------------------------
 arn:aws:iam::123456789012:role/myRedshiftRole
(1 row)
```

GET_MOUNTED_ROLE

다중 언어 AWS Glue 뷰의 일부로 호출하면 Lake Formation 스키마 또는 데이터베이스를 마운트하는 데 사용되는 IAM 역할을 반환할 수 있습니다. 다중 언어란 SQL이 Amazon EMR 및 Redshift와 같은 여러 쿼리 엔진에서 지원된다는 의미입니다. 다중 언어 Glue 뷰에 대한 자세한 내용은 AWS Glue 데이터 카탈로그에서 뷰 생성을 참조하세요.

구문

```
get_mounted_role()
```

반환 타입

VARCHAR 문자열 또는 null 값을 반환합니다.

사용 노트

이 함수는 외부 Lake Formation 뷰 이외의 모든 사용 사례에 대해 null을 반환합니다.

예제

다음 쿼리는 Lake Formation 리소스를 마운트할 자격 증명을 반환합니다.

```
CREATE EXTERNAL PROTECTED VIEW external_schema.remote_view AS
SELECT mycol, get_mounted_role() FROM external_schema.remote_table;

mycol | get_mounted_role
----------------------------
1       arn:aws:iam::123456789012:role/salesrole
(1 row)
```

HAS_ASSUMEROLE_PRIVILEGE

지정된 사용자가 지정된 명령을 실행할 권한이 있는 지정된 IAM 역할을 가지고 있으면 Boolean `true(t)`를 반환합니다. 사용자가 지정된 명령을 실행할 권한이 있는 지정된 IAM 역할을 가지고 있지 않으면 함수가 `false(f)`를 반환합니다. 권한에 대한 자세한 내용은 GRANT 섹션을 참조하세요.

구문

```
has_assumerole_privilege( [ user, ] iam_role_arn, cmd_type)
```

인수

사용자

IAM 역할 권한을 확인할 사용자의 이름입니다. 기본적으로 현재 사용자를 검사합니다. 슈퍼 사용자와 사용자가 이 함수를 사용할 수 있습니다. 그러나 사용자는 자신의 권한만 볼 수 있습니다.

iam_role_arn

명령 권한이 부여된 IAM 역할입니다.

cmd_type

액세스 권한이 부여된 명령입니다. 유효한 값은 다음과 같습니다.

- COPY
- UNLOAD
- EXTERNAL FUNCTION
- CREATE MODEL

반환 타입

BOOLEAN

예제

다음 쿼리는 사용자 reg_user1이 COPY 명령을 실행할 수 있는 Redshift-S3-Read 역할에 대한 권한을 가지고 있는지 확인합니다.

```
select has_assumerole_privilege('reg_user1', 'arn:aws:iam::123456789012:role/Redshift-S3-Read', 'copy');
```

```
has_assumerole_privilege
------------------------
true
(1 row)
```

HAS_DATABASE_PRIVILEGE

사용자가 지정한 데이터베이스에 대한 특정 권한을 가지고 있으면 true를 반환합니다. 권한에 대한 자세한 내용은 GRANT 섹션을 참조하세요.

구문

> **Note**
>
> CURRENT_SCHEMA는 리더 노드 함수이기 때문에 사용자 생성 테이블, STL 또는 STV 시스템 테이블, SVV 또는 SVL 시스템 뷰를 참조하는 경우에는 오류를 반환합니다.

```
has_database_privilege( [ user, ] database, privilege)
```

인수

사용자

데이터베이스 권한 유무를 확인할 사용자의 이름입니다. 기본적으로 현재 사용자를 검사합니다.

데이터베이스

권한과 연결되어 있는 데이터베이스입니다.

privilege

확인할 권한입니다. 유효한 값은 다음과 같습니다.

- CREATE
- 객체
- TEMP

반환 타입

CHAR 또는 VARCHAR 문자열을 반환합니다.

예제

다음은 GUEST 사용자가 TICKIT 데이터베이스에 대한 TEMP 권한을 가지고 있는지 확인하는 쿼리입니다.

```
select has_database_privilege('guest', 'tickit', 'temp');

has_database_privilege
------------------------
true
(1 row)
```

HAS_SCHEMA_PRIVILEGE

사용자가 지정한 스키마에 대한 특정 권한을 가지고 있으면 `true`를 반환합니다. 권한에 대한 자세한 내용은 GRANT 섹션을 참조하세요.

구문

> **Note**
>
> CURRENT_SCHEMA는 리더 노드 함수이기 때문에 사용자 생성 테이블, STL 또는 STV 시스템 테이블, SVV 또는 SVL 시스템 뷰를 참조하는 경우에는 오류를 반환합니다.

```
has_schema_privilege( [ user, ] schema, privilege)
```

인수

사용자

스키마 권한 유무를 확인할 사용자 이름입니다. 기본적으로 현재 사용자를 검사합니다.

스키마

권한과 연결되어 있는 스키마입니다.

privilege

확인할 권한입니다. 유효한 값은 다음과 같습니다.

- CREATE
- 객체

반환 타입

CHAR 또는 VARCHAR 문자열을 반환합니다.

예제

다음은 GUEST 사용자가 PUBLIC 스키마에 대한 CREATE 권한을 가지고 있는지 확인하는 쿼리입니다.

```
select has_schema_privilege('guest', 'public', 'create');

has_schema_privilege
----------------------
true
(1 row)
```

HAS_TABLE_PRIVILEGE

사용자에게 지정된 테이블에 대한 지정된 권한이 있으면 `true`을 반환하고 그렇지 않으면 `false`을 반환합니다.

구문

> **Note**
>
> CURRENT_SCHEMA는 리더 노드 함수이기 때문에 사용자 생성 테이블, STL 또는 STV 시스템 테이블, SVV 또는 SVL 시스템 뷰를 참조하는 경우에는 오류를 반환합니다. 권한에 대한 자세한 내용은 GRANT 섹션을 참조하세요.

```
has_table_privilege( [ user, ] table, privilege)
```

인수

사용자

테이블 권한 유무를 확인할 사용자 이름입니다. 기본적으로 현재 사용자를 검사합니다.

table

권한과 연결되어 있는 테이블입니다.

privilege

검사할 권한입니다. 유효한 값은 다음과 같습니다.

- SELECT
- INSERT
- UPDATE
- DELETE
- DROP
- REFERENCES

반환 타입

BOOLEAN

예시

다음은 GUEST 사용자가 LISTING 테이블에 대한 SELECT 권한을 가지고 있지 않은지 확인하는 쿼리입니다.

```
select has_table_privilege('guest', 'listing', 'select');

has_table_privilege
---------------------
false
```

다음 쿼리는 pg_tables 및 pg_user 카탈로그 테이블의 출력을 사용하여 선택, 삽입, 업데이트 및 삭제를 비롯한 테이블 권한을 나열합니다. 이는 샘플일 뿐입니다. 데이터베이스에서 스키마 이름과 테이블 이름을 지정해야 할 수 있습니다. 자세한 내용은 카탈로그 테이블 쿼리 단원을 참조하십시오.

```
SELECT
     tablename
     ,usename
     ,HAS_TABLE_PRIVILEGE(users.usename, tablename, 'select') AS sel
     ,HAS_TABLE_PRIVILEGE(users.usename, tablename, 'insert') AS ins
     ,HAS_TABLE_PRIVILEGE(users.usename, tablename, 'update') AS upd
     ,HAS_TABLE_PRIVILEGE(users.usename, tablename, 'delete') AS del
FROM
(SELECT * from pg_tables
WHERE schemaname = 'public' and tablename in ('event','listing')) as tables
,(SELECT * FROM pg_user) AS users;

tablename | usename   |  sel   |  ins  |  upd  | del
----------+-----------+--------+-------+-------+-------
event     |  john     |  true  | true  | true  | true
event     |  sally    |  false | false | false | false
event     |  elsa     |  false | false | false | false
listing   |  john     |  true  | true  | true  | true
listing   |  sally    |  false | false | false | false
listing   |  elsa     |  false | false | false | false
```

이전 쿼리에는 교차 조인도 포함됩니다. 자세한 내용은 JOIN 예 단원을 참조하십시오. public 스키마에 없는 테이블을 쿼리하려면 쿼리하기 전에 WHERE 절에서 schemaname 조건을 제거하고 다음 예제를 사용합니다.

```
SET SEARCH_PATH to 'schema_name';
```

LAST_USER_QUERY_ID

현재 세션에서 가장 최근에 완료된 사용자 쿼리의 쿼리 ID를 반환합니다. 현재 세션에서 쿼리가 실행되지 않았다면 last_user_query_id가 -1을 반환합니다. 이 함수는 리더 노드에서만 실행되는 쿼리에 대해서는 쿼리 ID를 반환하지 않습니다. 자세한 내용은 리더 노드 전용 함수 단원을 참조하십시오.

구문

```
last_user_query_id()
```

반환 타입

정수를 반환합니다.

예제

다음 쿼리는 현재 세션에서 사용자가 실행하여 완료된 마지막 쿼리의 ID를 반환합니다.

```
select last_user_query_id();
```

결과는 다음과 같습니다.

```
last_user_query_id
-----------------------
    5437
(1 row)
```

다음 쿼리는 현재 세션에서 사용자가 실행하여 가장 최근에 완료된 쿼리의 쿼리 ID 및 텍스트를 반환합니다.

```
select query_id, query_text from sys_query_history where query_id =
 last_user_query_id();
```

결과는 다음과 같습니다.

```
 query_id, query_text
---------
+-----------------------------------------------------------------------------------------------
 5556975 | select last_user_query_id() limit 100 --RequestID=<unique request ID>;
 TraceID=<unique trace ID>
```

PG_BACKEND_PID

현재 세션을 처리하고 있는 서버 프로세스의 프로세스 ID(PID)를 반환합니다.

> **Note**
>
> PID는 전역적 고유성이 없으므로 시간이 지난 뒤에도 재사용할 수 있습니다.

구문

```
pg_backend_pid()
```

반환 타입

정수를 반환합니다.

예제

PG_BACKEND_PID와 로그 테이블의 상관관계를 통해 현재 세션에 대한 정보를 가져올 수 있습니다. 예를 들어 다음은 현재 세션에서 완료된 쿼리에 대해 쿼리 ID와 쿼리 텍스트 일부를 반환하는 쿼리입니다.

```
select query, substring(text,1,40)
from stl_querytext
where pid =  PG_BACKEND_PID()
order by query desc;

 query |                substring
-------+----------------------------------------
 14831 | select query, substring(text,1,40) from
 14827 | select query, substring(path,0,80) as pa
 14826 | copy category from 's3://dw-tickit/manif
 14825 | Count rows in target table
 14824 | unload ('select * from category') to 's3
(5 rows)
```

PG_BACKEND_PID와 다음 로그 테이블에 있는 pid 열의 상관관계를 살펴볼 수 있습니다(괄호 안은 예외임).

- STL_CONNECTION_LOG

- STL_DDLTEXT
- STL_ERROR
- STL_QUERY
- STL_QUERYTEXT
- STL_SESSIONS (포함)
- STL_TR_CONFLICT
- STL_UTILITYTEXT
- STV_ACTIVE_CURSORS
- STV_INFLIGHT
- STV_LOCKS(lock_owner_pid)
- STV_RECENTS(process_id)

PG_GET_COLS

테이블 또는 보기 정의에 대한 열 메타데이터를 반환합니다.

구문

```
pg_get_cols('name')
```

인수

이름

Amazon Redshift 테이블 또는 뷰의 이름입니다. 자세한 내용은 이름 및 식별자 단원을 참조하십시오.

반환 타입

VARCHAR

사용 노트

PG_GET_COLS 함수는 테이블 또는 보기 정의에 있는 각 열에 대해 하나의 행을 반환합니다. 행에는 스키마 이름, 관계 이름, 열 이름, 데이터 형식 및 열 번호가 있는 쉼표로 분리된 목록이 포함되어 있습니다. SQL 결과의 형식 지정은 사용된 SQL 클라이언트에 따라 달라집니다.

예시

다음 예제는 연결된 데이터베이스 dev에서 사용자가 만든 스키마 public에 이름이 지정된 SALES_VW 뷰와 스키마 mytickit1에 이름이 지정된 sales 테이블 테이블에 대한 결과를 반환합니다.

다음 예는 SALES_VW라는 뷰에 대한 열 메타데이터를 반환합니다.

```
select pg_get_cols('sales_vw');

pg_get_cols
---------------------------------------------------------------
(public,sales_vw,salesid,integer,1)
(public,sales_vw,listid,integer,2)
(public,sales_vw,sellerid,integer,3)
(public,sales_vw,buyerid,integer,4)
(public,sales_vw,eventid,integer,5)
(public,sales_vw,dateid,smallint,6)
(public,sales_vw,qtysold,smallint,7)
(public,sales_vw,pricepaid,"numeric(8,2)",8)
(public,sales_vw,commission,"numeric(8,2)",9)
(public,sales_vw,saletime,"timestamp without time zone",10)
```

다음 예는 테이블 형식의 SALES_VW 뷰에 대한 열 메타데이터를 반환합니다.

```
select * from pg_get_cols('sales_vw')
cols(view_schema name, view_name name, col_name name, col_type varchar, col_num int);

view_schema | view_name | col_name   | col_type                    | col_num
------------+-----------+------------+-----------------------------+--------
public      | sales_vw  | salesid    | integer                     |       1
public      | sales_vw  | listid     | integer                     |       2
public      | sales_vw  | sellerid   | integer                     |       3
public      | sales_vw  | buyerid    | integer                     |       4
public      | sales_vw  | eventid    | integer                     |       5
public      | sales_vw  | dateid     | smallint                    |       6
public      | sales_vw  | qtysold    | smallint                    |       7
public      | sales_vw  | pricepaid  | numeric(8,2)                |       8
public      | sales_vw  | commission | numeric(8,2)                |       9
public      | sales_vw  | saletime   | timestamp without time zone |      10
```

다음 예는 테이블 형식의 SALES 스키마 mytickit1에 있는 테이블의 열 메타데이터를 반환합니다.

```
select * from pg_get_cols('"mytickit1"."sales"')
cols(view_schema name, view_name name, col_name name, col_type varchar, col_num int);

view_schema | view_name | col_name   | col_type                    | col_num
------------+-----------+------------+-----------------------------+--------
mytickit1   | sales     | salesid    | integer                     |       1
mytickit1   | sales     | listid     | integer                     |       2
mytickit1   | sales     | sellerid   | integer                     |       3
mytickit1   | sales     | buyerid    | integer                     |       4
mytickit1   | sales     | eventid    | integer                     |       5
mytickit1   | sales     | dateid     | smallint                    |       6
mytickit1   | sales     | qtysold    | smallint                    |       7
mytickit1   | sales     | pricepaid  | numeric(8,2)                |       8
mytickit1   | sales     | commission | numeric(8,2)                |       9
mytickit1   | sales     | saletime   | timestamp without time zone |      10
```

PG_GET_GRANTEE_BY_IAM_ROLE

지정된 IAM 역할이 부여된 모든 사용자와 그룹을 반환합니다.

구문

```
pg_get_grantee_by_iam_role('iam_role_arn')
```

인수

iam_role_arn

이 역할이 부여된 사용자와 그룹을 반환할 IAM 역할입니다.

반환 타입

VARCHAR

사용 노트

PG_GET_GRANTEE_BY_IAM_ROLE 함수는 각 사용자 또는 그룹에 대해 하나의 행을 반환합니다. 각 행에는 피부여자 이름, 피부여자 유형 및 부여된 권한이 포함됩니다. 피부여자 유형에 가능한 값은 퍼블릭의 경우 p, 사용자의 경우 u, 그룹의 경우 g입니다.

슈퍼 사용자만 이 함수를 사용할 수 있습니다.

예제

다음 예는 group1 및 reg_user1에게 IAM 역할 Redshift-S3-Write가 부여되었음을 나타냅니다. group_1의 사용자는 COPY 작업에 대해서만 역할을 지정할 수 있고, 사용자 reg_user1은 UNLOAD 작업을 수행하는 역할만 지정할 수 있습니다.

```
select pg_get_grantee_by_iam_role('arn:aws:iam::123456789012:role/Redshift-S3-Write');
```

```
  pg_get_grantee_by_iam_role
-----------------------------
 (group_1,g,COPY)
 (reg_user1,u,UNLOAD)
```

PG_GET_GRANTEE_BY_IAM_ROLE 함수의 다음 예는 결과 형식을 테이블로 지정합니다.

```
select grantee, grantee_type, cmd_type FROM
 pg_get_grantee_by_iam_role('arn:aws:iam::123456789012:role/Redshift-S3-Write')
 res_grantee(grantee text, grantee_type text, cmd_type text) ORDER BY 1,2,3;
```

```
  grantee  | grantee_type | cmd_type
-----------+--------------+----------
 group_1   | g            | COPY
 reg_user1 | u            | UNLOAD
```

PG_GET_IAM_ROLE_BY_USER

사용자에게 부여된 모든 IAM 역할 및 명령 권한을 반환합니다.

구문

```
pg_get_iam_role_by_user('name')
```

인수

이름

IAM 역할을 반환할 사용자의 이름입니다.

반환 타입

VARCHAR

사용 노트

PG_GET_IAM_ROLE_BY_USER 함수는 각 역할 및 명령 권한 집합에 대해 하나의 행을 반환합니다. 행에는 사용자 이름, IAM 역할 및 명령이 들어 있는 쉼표로 분리된 목록이 있습니다.

결과에서 값 `default`는 사용자가 표시된 명령을 수행하기 위해 사용 가능한 역할을 지정할 수 있음을 나타냅니다.

슈퍼 사용자만 이 함수를 사용할 수 있습니다.

예제

다음 예는 사용자 `reg_user1`이 COPY 작업을 수행하기 위해 사용 가능한 IAM 역할을 지정할 수 있음을 나타냅니다. 사용자는 UNLOAD 작업에 대해 `Redshift-S3-Write` 역할을 지정할 수도 있습니다.

```
select pg_get_iam_role_by_user('reg_user1');
```

```
                            pg_get_iam_role_by_user
-------------------------------------------------------------------------------
 (reg_user1,default,COPY)
 (reg_user1,arn:aws:iam::123456789012:role/Redshift-S3-Write,COPY|UNLOAD)
```

PG_GET_IAM_ROLE_BY_USER 함수의 다음 예는 결과 형식을 테이블로 지정합니다.

```
select username, iam_role, cmd FROM pg_get_iam_role_by_user('reg_user1')
 res_iam_role(username text, iam_role text, cmd text);
```

```
 username  |                    iam_role                     | cmd
-----------+-------------------------------------------------+------
 reg_user1 | default                                         | None
 reg_user1 | arn:aws:iam::123456789012:role/Redshift-S3-Read | COPY
```

PG_GET_LATE_BINDING_VIEW_COLS

데이터베이스에 있는 모든 Late Binding 보기에 대한 열 메타데이터를 반환합니다. 자세한 내용은 Late Binding 보기 섹션을 참조하세요.

구문

```
pg_get_late_binding_view_cols()
```

반환 타입

VARCHAR

사용 노트

PG_GET_LATE_BINDING_VIEW_COLS 함수는 Late Binding 보기에 있는 각 열에 대해 하나의 행을 반환합니다. 행에는 스키마 이름, 관계 이름, 열 이름, 데이터 형식 및 열 번호가 있는 쉼표로 분리된 목록이 포함되어 있습니다.

예제

다음 예는 모든 Late Binding 보기에 대한 열 메타데이터를 반환합니다.

```
select pg_get_late_binding_view_cols();

pg_get_late_binding_view_cols
------------------------------------------------------------
(public,myevent,eventname,"character varying(200)",1)
(public,sales_lbv,salesid,integer,1)
(public,sales_lbv,listid,integer,2)
(public,sales_lbv,sellerid,integer,3)
(public,sales_lbv,buyerid,integer,4)
(public,sales_lbv,eventid,integer,5)
(public,sales_lbv,dateid,smallint,6)
(public,sales_lbv,qtysold,smallint,7)
(public,sales_lbv,pricepaid,"numeric(8,2)",8)
(public,sales_lbv,commission,"numeric(8,2)",9)
(public,sales_lbv,saletime,"timestamp without time zone",10)
(public,event_lbv,eventid,integer,1)
(public,event_lbv,venueid,smallint,2)
(public,event_lbv,catid,smallint,3)
(public,event_lbv,dateid,smallint,4)
(public,event_lbv,eventname,"character varying(200)",5)
(public,event_lbv,starttime,"timestamp without time zone",6)
```

다음 예는 테이블 형식의 모든 Late Binding 보기에 대한 열 메타데이터를 반환합니다.

```
select * from pg_get_late_binding_view_cols() cols(view_schema name, view_name name,
 col_name name, col_type varchar, col_num int);
view_schema | view_name | col_name   | col_type                     | col_num
------------+-----------+------------+------------------------------+--------
public      | sales_lbv | salesid    | integer                      |       1
public      | sales_lbv | listid     | integer                      |       2
public      | sales_lbv | sellerid   | integer                      |       3
public      | sales_lbv | buyerid    | integer                      |       4
public      | sales_lbv | eventid    | integer                      |       5
public      | sales_lbv | dateid     | smallint                     |       6
public      | sales_lbv | qtysold    | smallint                     |       7
public      | sales_lbv | pricepaid  | numeric(8,2)                 |       8
public      | sales_lbv | commission | numeric(8,2)                 |       9
public      | sales_lbv | saletime   | timestamp without time zone  |      10
public      | event_lbv | eventid    | integer                      |       1
public      | event_lbv | venueid    | smallint                     |       2
public      | event_lbv | catid      | smallint                     |       3
public      | event_lbv | dateid     | smallint                     |       4
public      | event_lbv | eventname  | character varying(200)       |       5
public      | event_lbv | starttime  | timestamp without time zone  |       6
```

PG_GET_SESSION_ROLES

현재 로그인한 사용자의 세션 역할을 반환합니다. 사용자의 세션 역할은 로그인한 사용자에 대해 ID 제공업체(iDP)가 정의한 그룹입니다. 예를 들어 Microsoft Azure Active Directory(Azure AD)와 같은 ID 제공업체(idP)는 사용자 로그인 프로세스 중에 사용자의 신원을 확인하고 사용자가 속한 모든 외부 그룹을 제공합니다. 이러한 외부 그룹은 Amazon Redshift 역할로 변환되며 현재 세션 중에 사용할 수 있습니다. 이러한 역할을 세션 역할이라고 합니다. 관리자는 다른 Amazon Redshift 역할과 유사하게 세션 역할에 권한을 부여할 수 있습니다. 역할 사용에 대한 자세한 내용은 역할 기반 액세스 제어(RBAC)를 참조하세요. ID 제공업체(iDP)로 ID를 관리하는 방법에 대한 자세한 내용은 Amazon Redshift 관리 가이드에서 Amazon Redshift용 기본 IdP(ID 공급자) 페더레이션을 참조하세요

Amazon Redshift 카탈로그에 정의된 역할을 보려면 SVV_ROLES 시스템 뷰를 쿼리합니다.

구문

```
pg_get_session_roles()
```

반환 타입

두 개의 값으로 구성된 행 집합입니다. 첫 번째 값은 콜론(:)으로 구분된 두 부분으로 구성되며, 이 두 부분에는 idp-namespace:role-name이 포함되어 있습니다. idp-namespace는 ID 제공업체(iDP)의 네임스페이스입니다. role-name은 ID 제공업체(iDP)의 외부 그룹 이름입니다. 두 번째 값에는 역할 식별자인 role-id가 포함됩니다.

사용 노트

PG_GET_SESSION_ROLES 함수는 반환된 각 세션 역할에 대해 하나의 행을 반환합니다.

예시

다음 예에서는 Azure Active Directory IdP에서 각 역할에 대해 하나의 행을 반환합니다. 반환된 열은 열 name 및 roleid를 사용하여 sess_roles로 캐스팅됩니다. 각 name은 Azure Active Directory 네임스페이스와 Azure Active Directory의 그룹 이름으로 구성됩니다.

```
SELECT * FROM pg_get_session_roles() AS sess_roles(name name, roleid integer);

name                  roleid
-------------------------------
my_aad:test_group_1   106204
my_aad:test_group_2   106205
my_aad:test_group_3   106206
my_aad:test_group_4   106207
my_aad:test_group_5   106208
```

다음 예제에서는 현재 로그인한 IAM 사용자가 멤버인 각 IAM 그룹에 대해 하나의 행을 반환합니다. 반환된 열은 열 name 및 roleid를 사용하여 sess_roles로 캐스팅됩니다. 각 name은 IAM 네임스페이스 및 IAM 그룹 이름으로 구성됩니다.

```
SELECT * FROM pg_get_session_roles() AS sess_roles(name name, roleid integer);

name                  roleid
-------------------------------
IAM:myGroup           110332
```

PG_LAST_COPY_COUNT

현재 세션에서 마지막 COPY 명령으로 로드된 행의 수를 반환합니다. PG_LAST_COPY_COUNT는 마지막 COPY ID로 업데이트됩니다. 이 ID는 비록 로드가 중단되었더라도 로드 프로세스를 시작한 마지

막 COPY의 쿼리 ID이기도 합니다. 쿼리 ID와 COPY ID는 COPY 명령이 로드 프로세스를 시작할 때 업데이트됩니다.

구문 오류 또는 부족한 권한으로 인해 COPY 명령이 중단되면 COPY ID는 업데이트되지 않으며, PG_LAST_COPY_COUNT가 이전 COPY 수를 반환합니다. 현재 세션에서 COPY 명령이 실행되지 않았거나, 혹은 마지막 COPY가 로드 도중 중단되면 PG_LAST_COPY_COUNT가 0을 반환합니다. 자세한 내용은 PG_LAST_COPY_ID 단원을 참조하십시오.

구문

```
pg_last_copy_count()
```

반환 타입

BIGINT를 반환합니다.

예제

다음은 현재 세션에서 마지막 COPY 명령으로 로드된 행의 수를 반환하는 쿼리입니다.

```
select pg_last_copy_count();

pg_last_copy_count
--------------------
             192497
(1 row)
```

PG_LAST_COPY_ID

현재 세션에서 가장 최근에 완료된 COPY 명령의 쿼리 ID를 반환합니다. 현재 세션에서 COPY 명령이 실행되지 않았다면 PG_LAST_COPY_ID가 -1을 반환합니다.

PG_LAST_COPY_ID 값은 COPY 명령이 로드 프로세스를 시작할 때 업데이트됩니다. 잘못된 로드 데이터로 인해 COPY 명령이 중단되더라도 COPY ID는 업데이트됩니다. 따라서 STL_LOAD_ERRORS 테이블에 대한 쿼리를 실행할 때 PG_LAST_COPY_ID를 사용할 수 있습니다. 단, COPY 트랜잭션이 롤백되면 COPY ID는 업데이트되지 않습니다.

로그 프로세스가 시작되기 전에 구문 오류, 액세스 오류, 잘못된 자격 증명, 부족한 권한 등으로 인한 오류가 발생하여 COPY 명령이 중단되면 COPY ID가 업데이트되지 않습니다. 성공적으로 연결된 후 데

이터 로드 이전에 시작되는 압축 분석 단계에서 COPY 명령이 중단되어도 COPY ID가 업데이트되지 않습니다.

구문

```
pg_last_copy_id()
```

반환 타입

정수를 반환합니다.

예제

다음은 현재 세션에서 마지막 COPY 명령의 쿼리 ID를 반환하는 쿼리입니다.

```
select pg_last_copy_id();

pg_last_copy_id
---------------
          5437
(1 row)
```

다음은 STL_LOAD_ERRORS를 STL_LOADERROR_DETAIL로 조인하여 현재 세션에서 가장 최근 로드 도중 발생한 오류 정보를 표시하는 쿼리입니다.

```
select d.query, substring(d.filename,14,20),
d.line_number as line,
substring(d.value,1,16) as value,
substring(le.err_reason,1,48) as err_reason
from stl_loaderror_detail d, stl_load_errors le
where d.query = le.query
and d.query = pg_last_copy_id();

 query |    substring      | line |  value   |                  err_reason
-------+-------------------+------+----------
+-------------------------------------------------
    558| allusers_pipe.txt |  251 | 251      | String contains invalid or unsupported
 UTF8 code
    558| allusers_pipe.txt |  251 | ZRU29FGR | String contains invalid or unsupported
 UTF8 code
    558| allusers_pipe.txt |  251 | Kaitlin  | String contains invalid or unsupported
 UTF8 code
```

```
   558| allusers_pipe.txt |  251 | Walter   | String contains invalid or unsupported
UTF8 code
```

PG_LAST_UNLOAD_ID

현재 세션에서 가장 최근에 완료된 UNLOAD 명령의 쿼리 ID를 반환합니다. 현재 세션에서 UNLOAD 명령이 실행되지 않았다면 PG_LAST_UNLOAD_ID가 -1을 반환합니다.

PG_LAST_UNLOAD_ID 값은 UNLOAD 명령이 로드 프로세스를 시작할 때 업데이트됩니다. 잘못된 로드 데이터로 인해 UNLOAD 명령이 중단되더라도 UNLOAD ID가 업데이트됩니다. 따라서 추가 조사에 UNLOAD ID를 사용할 수 있습니다. 단, UNLOAD 트랜잭션이 롤백되면 UNLOAD ID는 업데이트되지 않습니다.

로그 프로세스가 시작되기 전에 구문 오류, 액세스 오류, 잘못된 자격 증명, 부족한 권한 등으로 인한 오류가 발생하여 UNLOAD 명령이 중단되면 UNLOAD ID가 업데이트되지 않습니다.

구문

```
PG_LAST_UNLOAD_ID()
```

반환 타입

정수를 반환합니다.

예제

다음은 현재 세션에서 마지막 UNLOAD 명령의 쿼리 ID를 반환하는 쿼리입니다.

```
select PG_LAST_UNLOAD_ID();

PG_LAST_UNLOAD_ID
---------------
          5437
(1 row)
```

PG_LAST_QUERY_ID

현재 세션에서 가장 최근에 완료된 쿼리의 쿼리 ID를 반환합니다. 현재 세션에서 쿼리가 실행되지 않았다면 PG_LAST_QUERY_ID가 -1을 반환합니다. PG_LAST_QUERY_ID는 리더 노드에서만 실행되는 쿼리에 대해서는 쿼리 ID를 반환하지 않습니다. 자세한 내용은 리더 노드 전용 함수 단원을 참조하십시오.

구문

```
pg_last_query_id()
```

반환 타입

정수를 반환합니다.

예제

다음은 현재 세션에서 완료된 마지막 쿼리의 ID를 반환하는 쿼리입니다.

```
select pg_last_query_id();
```

결과는 다음과 같습니다.

```
pg_last_query_id
----------------
           5437
(1 row)
```

다음은 현재 세션에서 가장 최근에 완료된 쿼리의 쿼리 ID 및 텍스트를 반환하는 쿼리입니다.

```
select query, trim(querytxt) as sqlquery
from stl_query
where query = pg_last_query_id();
```

결과는 다음과 같습니다.

```
query | sqlquery
------+-------------------------------------------------
 5437 | select name, loadtime from stl_file_scan where loadtime > 1000000;
(1 rows)
```

PG_LAST_UNLOAD_COUNT

현재 세션에서 완료된 마지막 UNLOAD 명령에 의해 언로드된 행 수를 반환합니다. PG_LAST_UNLOAD_COUNT는 작업이 중단되었더라도 마지막 UNLOAD의 쿼리 ID로 업데이트됩니다. 쿼리 ID는 UNLOAD 완료 시 업데이트됩니다. 구문 오류 또는 부족한 권한으로 인해 UNLOAD 명령이 중단되면 PG_LAST_UNLOAD_COUNT가 이전 UNLOAD 수를 반환합니다. 현재

세션에서 UNLOAD 명령이 완료되지 않았거나, 혹은 마지막 UNLOAD가 언로드 작업 도중 중단되면 PG_LAST_UNLOAD_COUNT가 0을 반환합니다.

구문

```
pg_last_unload_count()
```

반환 타입

BIGINT를 반환합니다.

예제

다음은 현재 세션에서 마지막 UNLOAD 명령으로 언로드된 행의 수를 반환하는 쿼리입니다.

```
select pg_last_unload_count();

pg_last_unload_count
--------------------
              192497
(1 row)
```

SLICE_NUM 함수

클러스터에서 행 데이터가 위치한 조각 번호에 해당하는 정수를 반환합니다. SLICE_NUM은 파라미터가 없습니다.

구문

```
SLICE_NUM()
```

반환 타입

SLICE_NUM 함수는 정수를 반환합니다.

예시

다음은 EVENTS 테이블에서 처음 10개 EVENT 행의 데이터가 포함된 조각을 표시하는 예입니다.

```
select distinct eventid, slice_num() from event order by eventid limit 10;

 eventid | slice_num
```

```
---------+-----------
       1 |         1
       2 |         2
       3 |         3
       4 |         0
       5 |         1
       6 |         2
       7 |         3
       8 |         0
       9 |         1
      10 |         2
(10 rows)
```

다음은 코드(10000)를 반환하는 예로서 FROM 문을 제외한 쿼리가 리더 노드에서 실행되는 것을 나타냅니다.

```
select slice_num();
slice_num
-----------
10000
(1 row)
```

USER

CURRENT_USER의 동의어입니다. CURRENT_USER 섹션을 참조하세요.

ROLE_IS_MEMBER_OF

역할이 다른 역할의 멤버인 경우 true를 반환합니다. 슈퍼 사용자는 모든 역할의 멤버십을 확인할 수 있습니다. ACCESS SYSTEM TABLE 권한이 있는 일반 사용자는 모든 사용자의 멤버십을 확인할 수 있습니다. 그렇지 않으면 일반 사용자는 액세스 권한을 가지고 있는 역할만 확인할 수 있습니다. 제공된 역할이 없거나 현재 사용자가 역할에 액세스할 수 없는 경우 Amazon Redshift에 오류가 발생합니다.

구문

```
role_is_member_of( role_name,  granted_role_name)
```

인수

role_name

역할의 이름.

granted_role_name

부여된 역할의 이름.

반환 타입

BOOLEAN을 반환합니다.

예제

다음 쿼리는 역할이 role1 또는 role2의 구성원이 아님을 확인합니다.

```
SELECT role_is_member_of('role1', 'role2');

 role_is_member_of
-------------------
             False
```

USER_IS_MEMBER_OF

사용자가 역할이나 그룹의 구성원인 경우 true를 반환합니다. 슈퍼 사용자는 모든 사용자의 멤버십을 확인할 수 있습니다. sys:secadmin 또는 sys:superuser 역할의 멤버인 일반 사용자는 모든 사용자의 멤버십을 확인할 수 있습니다. 그렇지 않으면 일반 사용자는 자신만 확인할 수 있습니다. 제공된 자격 증명이 없거나 현재 사용자가 역할에 액세스할 수 없는 경우 Amazon Redshift는 오류를 전송합니다.

구문

```
user_is_member_of( user_name,  role_name | group_name)
```

인수

user_name

사용자의 이름입니다.

role_name

역할의 이름.

group_name

그룹 이름입니다.

반환 타입

BOOLEAN을 반환합니다.

예제

다음 쿼리는 사용자가 role1의 멤버가 아니라는 것을 확인하는 쿼리입니다.

```
SELECT user_is_member_of('reguser', 'role1');

 user_is_member_of
-------------------
           False
```

version

VERSION 함수는 현재 설치된 릴리스에 대한 정보와 함께 마지막에 특정 Amazon Redshift 버전 정보를 반환합니다.

> **Note**
>
> CURRENT_SCHEMA는 리더 노드 함수이기 때문에 사용자 생성 테이블, STL 또는 STV 시스템 테이블, SVV 또는 SVL 시스템 뷰를 참조하는 경우에는 오류를 반환합니다.

구문

```
VERSION()
```

반환 타입

CHAR 또는 VARCHAR 문자열을 반환합니다.

예시

다음 예제에서는 현재 클러스터의 클러스터 버전 정보를 보여줍니다.

```
select version();
```

```
 version
 ----------------------------------------------------------------------------------------------------
```

```
PostgreSQL 8.0.2 on i686-pc-linux-gnu, compiled by GCC gcc (GCC) 3.4.2 20041017 (Red
Hat 3.4.2-6.fc3), Redshift 1.0.12103
```

여기서 1.0.12103은 클러스터 버전 번호입니다.

> **Note**
>
> 클러스터가 최신 클러스터 버전으로 업데이트되도록 설정하려면 유지 관리 기간을 조정합니다.

예약어

다음은 Amazon Redshift 예약어 목록입니다. 이러한 예약어는 구분된 식별자(큰따옴표)와 함께 사용할 수 있습니다.

> **Note**
>
> START 및 CONNECT는 예약어가 아니지만, 쿼리에서 START 및 CONNECT를 테이블 별칭으로 사용하는 경우에는 런타임 시 오류가 발생하지 않도록 구분된 식별자 또는 AS를 사용하세요.

자세한 내용은 이름 및 식별자 단원을 참조하십시오.

```
AES128
AES256
ALL
ALLOWOVERWRITE
ANALYSE
ANALYZE
AND
ANY
ARRAY
AS
ASC
AUTHORIZATION
AZ64
BACKUP
BETWEEN
BINARY
```

```
BLANKSASNULL
BOTH
BYTEDICT
BZIP2
CASE
CAST
CHECK
COLLATE
COLUMN
CONSTRAINT
CREATE
CREDENTIALS
CROSS
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_USER
CURRENT_USER_ID
DEFAULT
DEFERRABLE
DEFLATE
DEFRAG
DELTA
DELTA32K
DESC
DISABLE
DISTINCT
DO
ELSE
EMPTYASNULL
ENABLE
ENCODE
ENCRYPT
ENCRYPTION
END
EXCEPT
EXPLICIT
FALSE
FOR
FOREIGN
FREEZE
FROM
FULL
GLOBALDICT256
```

```
GLOBALDICT64K
GRANT
GROUP
GZIP
HAVING
IDENTITY
IGNORE
ILIKE
IN
INITIALLY
INNER
INTERSECT
INTERVAL
INTO
IS
ISNULL
JOIN
LEADING
LEFT
LIKE
LIMIT
LOCALTIME
LOCALTIMESTAMP
LUN
LUNS
LZO
LZOP
MINUS
MOSTLY16
MOSTLY32
MOSTLY8
NATURAL
NEW
NOT
NOTNULL
NULL
NULLS
OFF
OFFLINE
OFFSET
OID
OLD
ON
ONLY
```

```
OPEN
OR
ORDER
OUTER
OVERLAPS
PARALLEL
PARTITION
PERCENT
PERMISSIONS
PIVOT
PLACING
PRIMARY
RAW
READRATIO
RECOVER
REFERENCES
REJECTLOG
RESORT
RESPECT
RESTORE
RIGHT
SELECT
SESSION_USER
SIMILAR
SNAPSHOT
SOME
SYSDATE
SYSTEM
TABLE
TAG
TDES
TEXT255
TEXT32K
THEN
TIMESTAMP
TO
TOP
TRAILING
TRUE
TRUNCATECOLUMNS
UNION
UNIQUE
UNNEST
UNPIVOT
```

```
USER
USING
VERBOSE
WALLET
WHEN
WHERE
WITH
WITHOUT
```

시스템 테이블 및 뷰 참조

Amazon Redshift는 시스템 작동 방식에 대한 정보가 저장되어 있는 시스템 테이블 및 뷰가 많습니다. 이러한 시스템 테이블 및 뷰 역시 다른 데이터베이스 테이블에 대한 쿼리와 동일한 방법으로 쿼리를 실행할 수 있습니다. 이번 단원에서는 몇 가지 시스템 테이블 쿼리 샘플을 살펴보면서 다음과 같은 주제를 가지고 설명하겠습니다.

- 다양한 유형의 시스템 테이블 및 뷰 생성
- 이러한 테이블에서 얻을 수 있는 정보 유형
- Amazon Redshift 시스템 테이블을 카탈로그 테이블에 조인하는 방법
- 점점 증가하는 시스템 테이블 로그 파일의 관리 방법

일부 시스템 테이블은 진단 목적으로 AWS 직원만 사용할 수 있습니다. 다음 단원에서는 시스템 관리자 또는 데이터베이스 사용자가 유용한 정보를 얻기 위해 쿼리를 실행할 수 있는 시스템 테이블에 대해서 얘기하겠습니다.

> **Note**
>
> 시스템 테이블은 자동 또는 수동 클러스터 백업(스냅샷)에 포함되지 않습니다. STL 시스템 뷰는 7일간의 로그 기록을 보존합니다. 따라서 로그 보존에는 별도의 고객 조치가 필요하지 않습니다. 따라서 로그 데이터를 7일 이상 보관하고 싶다면 정기적으로 다른 테이블에 복사하거나 Amazon S3으로 언로드해야 합니다.

주제

- 시스템 모니터링(프로비저닝만 해당)
- 시스템 카탈로그 테이블

시스템 테이블 및 뷰의 유형

다음과 같은 여러 유형의 시스템 테이블과 뷰가 있습니다.

- SVV 뷰에는 임시 STV 테이블에 대한 참조와 함께 데이터베이스 객체에 대한 정보가 포함됩니다.
- SYS 뷰는 프로비저닝된 클러스터 및 서버리스 작업 그룹의 쿼리 및 워크로드 리소스 사용량을 모니터링하는 데 사용됩니다.
- STL 뷰는 디스크에 영구 저장되어 시스템 이력을 알려주는 로그에서 생성됩니다.
- STV 테이블은 현재 시스템 데이터의 스냅샷을 포함하는 가상 시스템 테이블입니다. 이 두 테이블은 일시적인 인메모리 데이터를 기반으로 하기 때문에 디스크 기반 로그나 정규 테이블에 영구 저장되지는 않습니다.
- SVCS 뷰는 동시성 확장 클러스터와 기본 클러스터 모두의 쿼리에 대한 세부 정보를 제공합니다.
- SVL 뷰는 기본 클러스터의 쿼리에 대한 세부 정보를 제공합니다.

시스템 테이블과 뷰는 정규 테이블과 동일한 일관성 모델을 사용하지 않습니다. 특히 STV 테이블과 SVV 뷰에 대한 쿼리를 실행할 때는 이러한 문제를 알고 있어야 합니다. 예를 들어 일반 테이블 t1에 열 c1이 있다고 가정할 때 다음과 같은 쿼리는 아무런 행도 반환하지 않을 수 있습니다.

```
select * from t1
where c1 > (select max(c1) from t1)
```

하지만 시스템 테이블에 대한 다음 쿼리는 분명히 행을 반환하게 됩니다.

```
select * from stv_exec_state
where currenttime > (select max(currenttime) from stv_exec_state)
```

위 쿼리가 행을 반환할 수 있는 이유는 currenttime이 일시적이고, 쿼리에서 두 참조가 평가 시 동일한 값을 반환하지 않기 때문입니다.

반면에 다음 쿼리는 아무런 행도 반환하지 않습니다.

```
select * from stv_exec_state
where currenttime = (select max(currenttime) from stv_exec_state)
```

시스템 테이블 및 뷰에 있는 데이터의 가시성

시스템 테이블 및 보기에 저장되는 데이터의 가시성은 사용자 가시성과 수퍼유저 가시성이라는 두 가지 등급이 있습니다.

수퍼유저 가시성 카테고리에 속하는 테이블의 데이터는 수퍼유저 권한이 있는 사용자만 볼 수 있습니다. 일반 사용자는 사용자 가시성 테이블의 데이터를 볼 수 있습니다. 일반 사용자에게 수퍼유저 가시성 테이블에 대한 액세스를 제공하려면 해당 테이블에 대한 SELECT 권한을 일반 사용자에게 부여합니다. 자세한 내용은 GRANT 단원을 참조하십시오.

기본적으로 대부분의 사용자 가시성 테이블에서 일반 사용자는 다른 사용자가 생성한 행을 볼 수 없습니다. 일반 사용자에게 SYSLOG ACCESS UNRESTRICTED가 부여된 경우 해당 사용자는 다른 사용자가 생성한 행을 포함하여 사용자 가시성 테이블에 있는 모든 행을 볼 수 있습니다. 자세한 내용은 ALTER USER 또는 사용자 생성을 참조하세요. SVV_TRANSACTIONS의 모든 행은 모든 사용자에게 표시됩니다. 데이터 가시성에 대한 자세한 내용은 AWS re:Post 기술 자료 문서 Amazon Redshift 데이터베이스의 일반 사용자에게 내 클러스터의 다른 사용자가 제공한 시스템 테이블 데이터를 볼 수 있는 권한을 허용하려면 어떻게 해야 합니까?를 참조하세요.

메타데이터 보기의 경우 Amazon Redshift는 SYSLOG ACCESS UNRESTRICTED가 부여된 사용자에게 가시성을 허용하지 않습니다.

> **Note**
>
> 사용자에게 시스템 테이블에 대한 무제한 액세스를 제공하면 다른 사용자가 생성한 데이터에 대한 가시성이 사용자에게 제공됩니다. 예를 들어, STL_QUERY 및 STL_QUERY_TEXT에는 INSERT, UPDATE 및 DELETE 문의 전체 텍스트가 포함되며, 여기에는 사용자가 생성한 민감한 데이터가 포함될 수 있습니다.

수퍼유저는 모든 테이블의 모든 행을 볼 수 있습니다. 일반 사용자에게 수퍼유저 가시성 테이블에 대한 액세스를 제공하려면 해당 테이블에 대한 GRANT SELECT 권한을 일반 사용자에게 부여합니다.

시스템 생성 쿼리 필터링

일반적으로 쿼리 관련 시스템 테이블 및 뷰(SVL_QUERY_SUMMARY, SVL_QLOG 등)에는 Amazon Redshift에서 데이터베이스 상태를 모니터링하는 데 사용하는 자동 생성 문이 다수 포함되어 있습니다. 이러한 시스템 생성 쿼리도 수퍼유저에게 노출되지만 사용할 일은 거의 없습니다. 따라서 `userid` 열을 사용하는 시스템 테이블이나 시스템 뷰에서 선택할 때 이러한 쿼리를 필터링하려면 WHERE 절에 `userid > 1` 조건을 추가하면 됩니다. 예시:

```
select * from svl_query_summary where userid > 1
```

프로비저닝 전용 쿼리를 SYS 모니터링 뷰 쿼리로 마이그레이션

프로비저닝된 클러스터에서 Amazon Redshift Serverless로 마이그레이션

프로비저닝된 클러스터를 Amazon Redshift Serverless로 마이그레이션하는 경우 프로비저닝된 클러스터의 데이터만 저장하는 다음과 같은 시스템 뷰를 사용하는 쿼리가 있을 수 있습니다.

- 모든 STL 뷰
- 모든 STV 뷰
- 모든 SVCS 뷰
- 모든 SVL 뷰
- 일부 SVV 뷰
  - Amazon Redshift Serverless에서 지원되지 않는 SVV 뷰의 전체 목록은 Amazon Redshift 관리 안내서의 Amazon Redshift Serverless로 쿼리 및 워크로드 모니터링 하단에 있는 목록을 참조하세요.

쿼리를 계속 사용하려면 SYS 모니터링 뷰에서 정의되었으며 프로비저닝 전용 뷰의 열에 해당하는 열을 사용하도록 쿼리를 다시 구성하세요. 프로비저닝 전용 뷰와 SYS 모니터링 뷰 간의 매핑 관계는 SYS 모니터링 뷰로 마이그레이션하기 위한 시스템 뷰 매핑 섹션을 참조하세요.

프로비저닝된 클러스터에 머물면서 쿼리 업데이트

Amazon Redshift Serverless로 마이그레이션하지 않는 경우에도 기존 쿼리를 업데이트하고 싶을 수 있습니다. SYS 모니터링 뷰는 사용 편의성과 복잡성 감소를 위해 설계되어 효과적인 모니터링 및 문제 해결을 위한 완전한 지표를 제공합니다. 여러 개의 프로비저닝 전용 뷰의 정보를 통합하는 SYS_QUERY_HISTORY 및 SYS_QUERY_DETAIL 등의 SYS 뷰를 사용하면 쿼리를 간소화할 수 있습니다.

SYS 모니터링 뷰를 사용하여 쿼리 식별자 추적 개선

SYS_QUERY_HISTORY 및 SYS_QUERY_DETAIL과 같은 SYS 모니터링 뷰에는 사용자 쿼리에 대한 식별자가 들어 있는 query_id 열이 포함됩니다. 마찬가지로 STL_QUERY 및 SVL_QLOG와 같은 프로비저닝 전용 뷰에는 쿼리 열이 포함되며 쿼리 열에도 쿼리 식별자가 들어 있습니다. 하지만 SYS 시스템 뷰에 기록된 쿼리 식별자는 프로비저닝 전용 뷰에 기록된 쿼리 식별자와 다릅니다.

SYS 뷰의 query_id 열 값과 프로비저닝 전용 뷰의 쿼리 열 값 간의 차이는 다음과 같습니다.

- SYS 뷰에서 query_id 열은 사용자가 제출한 쿼리를 원래 형식으로 기록합니다. Amazon Redshift 최적화 프로그램은 성능 향상을 위해 이러한 쿼리를 하위 쿼리로 분류할 수 있지만, 실행하는 단일 쿼리에는 SYS_QUERY_HISTORY에 여전히 행이 하나뿐입니다. 개별 하위 쿼리는 SYS_QUERY_DETAIL에서 확인할 수 있습니다.
- 프로비저닝 전용 뷰에서는 쿼리 열에 하위 쿼리 수준의 쿼리가 기록됩니다. Amazon Redshift 최적화 프로그램이 원래 쿼리를 여러 하위 쿼리로 다시 작성하는 경우, 실행하는 단일 쿼리에 대해 쿼리 식별자 값이 서로 다른 여러 행이 STL_QUERY에 생성됩니다.

모니터링 및 진단 쿼리를 프로비저닝 전용 뷰에서 SYS 뷰로 마이그레이션할 때는 이러한 차이를 고려하여 적절하게 쿼리를 편집하세요. Amazon Redshift의 쿼리 처리 방식에 대한 자세한 내용은 쿼리 계획 및 실행 워크플로우 섹션을 참조하세요.

예제

Amazon Redshift가 프로비저닝 전용 뷰와 SYS 모니터링 뷰에서 어떻게 다르게 쿼리를 기록하는지 보여주는 예는 다음 샘플 쿼리를 참조하세요. 이 쿼리는 Amazon Redshift에서 실행할 때와 같이 작성된 것입니다.

```
SELECT
  s_name
  , COUNT(*) AS numwait
FROM
  supplier,
  lineitem l1,
  orders,
  nation
WHERE    s_suppkey = l1.l_suppkey
         AND o_orderkey = l1.l_orderkey
         AND o_orderstatus = 'F'
         AND l1.l_receiptdate > l1.l_commitdate
         AND EXISTS (SELECT
                       *
                     FROM
                       lineitem l2
                     WHERE  l2.l_orderkey = l1.l_orderkey
                            AND l2.l_suppkey <> l1.l_suppkey )
         AND NOT EXISTS (SELECT
                           *
```

```
                          FROM
                            lineitem l3
                          WHERE  l3.l_orderkey = l1.l_orderkey
                                 AND l3.l_suppkey <> l1.l_suppkey
                                 AND l3.l_receiptdate > l3.l_commitdate )
         AND s_nationkey = n_nationkey
         AND n_name = 'UNITED STATES'
GROUP BY
  s_name
ORDER BY
  numwait DESC
  , s_name LIMIT 100;
```

눈에 보이지는 않지만 Amazon Redshift 쿼리 최적화 프로그램은 사용자가 제출한 위 쿼리를 5개의 하위 쿼리로 다시 작성합니다.

첫 번째 하위 쿼리는 임시 테이블을 생성하여 하위 쿼리를 구체화합니다.

```
CREATE TEMP TABLE volt_tt_606590308b512(l_orderkey
                                       , l_suppkey
                                       , s_name   ) AS SELECT
                                                       l1.l_orderkey
                                                       , l1.l_suppkey
                                                       , public.supplier.s_name
                                                     FROM
                                                       public.lineitem AS l1,
                                                       public.nation,
                                                       public.orders,
                                                       public.supplier
                                                     WHERE  l1.l_commitdate <
 l1.l_receiptdate
                                                            AND l1.l_orderkey =
 public.orders.o_orderkey
                                                            AND l1.l_suppkey =
 public.supplier.s_suppkey
                                                            AND public.nation.n_name
 = 'UNITED STATES'::CHAR(8)
                                                            AND
 public.nation.n_nationkey = public.supplier.s_nationkey
                                                            AND
 public.orders.o_orderstatus = 'F'::CHAR(1);
```

두 번째 하위 쿼리는 임시 테이블에서 통계를 수집합니다.

```
padb_fetch_sample: select count(*) from volt_tt_606590308b512;
```

세 번째 하위 쿼리는 위에서 만든 임시 테이블을 참조하여 다른 하위 쿼리를 구체화하기 위해 또 다른 임시 테이블을 만듭니다.

```
CREATE TEMP TABLE volt_tt_606590308c2ef(l_orderkey
                                       , l_suppkey) AS (SELECT

 volt_tt_606590308b512.l_orderkey
                                                          ,
 volt_tt_606590308b512.l_suppkey
                                                        FROM
                                                          public.lineitem AS l2,
                                                          volt_tt_606590308b512
                                                        WHERE  l2.l_suppkey <>
 volt_tt_606590308b512.l_suppkey
                                                               AND l2.l_orderkey =
 volt_tt_606590308b512.l_orderkey)
                                                               EXCEPT distinct (SELECT
 volt_tt_606590308b512.l_orderkey, volt_tt_606590308b512.l_suppkey
                                                               FROM public.lineitem AS
 l3, volt_tt_606590308b512
                                                               WHERE l3.l_commitdate <
 l3.l_receiptdate
                                                                 AND l3.l_suppkey <>
 volt_tt_606590308b512.l_suppkey
                                                                 AND l3.l_orderkey =
 volt_tt_606590308b512.l_orderkey);
```

네 번째 하위 쿼리도 임시 테이블의 통계를 수집합니다.

```
padb_fetch_sample: select count(*) from volt_tt_606590308c2ef
```

마지막 하위 쿼리는 위에서 만든 임시 테이블을 사용하여 출력을 생성합니다.

```
SELECT
  volt_tt_606590308b512.s_name AS s_name
  , COUNT(*) AS numwait
FROM
  volt_tt_606590308b512,
  volt_tt_606590308c2ef
```

```
WHERE    volt_tt_606590308b512.l_orderkey = volt_tt_606590308c2ef.l_orderkey
         AND volt_tt_606590308b512.l_suppkey = volt_tt_606590308c2ef.l_suppkey
GROUP BY
  1
ORDER BY
  2 DESC
  , 1 ASC LIMIT 100;
```

프로비저닝 전용 시스템 뷰 STL_QUERY에서 Amazon Redshift는 다음과 같이 하위 쿼리 수준에서 5개 행을 기록합니다.

```
SELECT userid, xid, pid, query, querytxt::varchar(100);
FROM stl_query
WHERE xid = 48237350
ORDER BY xid, starttime;

 userid |   xid    |    pid     |  query   |
    querytxt
--------+----------+------------+----------
+-----------------------------------------------------------------------------------------------------
    101 | 48237350 | 1073840810 | 12058151 | CREATE TEMP TABLE
 volt_tt_606590308b512(l_orderkey, l_suppkey, s_name) AS SELECT l1.l_orderkey, l1.l
    101 | 48237350 | 1073840810 | 12058152 | padb_fetch_sample: select count(*) from
 volt_tt_606590308b512
    101 | 48237350 | 1073840810 | 12058156 | CREATE TEMP TABLE
 volt_tt_606590308c2ef(l_orderkey, l_suppkey) AS (SELECT volt_tt_606590308b512.l_or
    101 | 48237350 | 1073840810 | 12058168 | padb_fetch_sample: select count(*) from
 volt_tt_606590308c2ef
    101 | 48237350 | 1073840810 | 12058170 | SELECT s_name , COUNT(*) AS numwait FROM
 supplier, lineitem l1, orders, nation WHERE s_suppkey = l1.
(5 rows)
```

SYS 모니터링 뷰 SYS_QUERY_HISTORY에서 Amazon Redshift는 다음과 같이 쿼리를 기록합니다.

```
SELECT user_id, transaction_id, session_id, query_id, query_text::varchar(100)
FROM sys_query_history
WHERE transaction_id = 48237350
ORDER BY start_time;

 user_id | transaction_id | session_id | query_id |
          query_text
---------+----------------+------------+----------
+-----------------------------------------------------------------------------------------------------
```

```
     101 |       48237350 | 1073840810 | 12058149 | SELECT s_name , COUNT(*) AS numwait
 FROM supplier, lineitem l1, orders, nation WHERE s_suppkey = l1.
```

SYS_QUERY_DETAIL에서는 SYS_QUERY_HISTORY_HISTORY의 query_id 값을 사용하여 하위 쿼리 수준의 세부 정보를 찾을 수 있습니다. child_query_sequence 열에는 하위 쿼리가 실행되는 순서가 표시됩니다. SYS_QUERY_DETAIL의 열에 대한 자세한 내용은 SYS_QUERY_DETAIL 섹션을 참조하세요.

```
select user_id,
       query_id,
       child_query_sequence,
       stream_id,
       segment_id,
       step_id,
       start_time,
       end_time,
       duration,
       blocks_read,
       blocks_write,
       local_read_io,
       remote_read_io,
       data_skewness,
       time_skewness,
       is_active,
       spilled_block_local_disk,
       spilled_block_remote_disk
from sys_query_detail
where query_id = 12058149
      and step_id = -1
order by query_id,
         child_query_sequence,
         stream_id,
         segment_id,
         step_id;

 user_id | query_id | child_query_sequence | stream_id | segment_id | step_id |
       start_time         |          end_time          | duration | blocks_read |
 blocks_write | local_read_io | remote_read_io | data_skewness | time_skewness |
 is_active | spilled_block_local_disk | spilled_block_remote_disk
---------+----------+----------------------+-----------+------------+---------
+----------------------------+----------------------------+----------+-------------
+--------------+---------------+----------------+---------------+--------------
+-----------+---------------------------+---------------------------
```

```
    101 | 12058149 |                    1 |         0 |          0 |      -1 |
2023-09-27 15:40:38.512415 | 2023-09-27 15:40:38.533333 |    20918 |           0 |
        0 |             0 |              0 |             0 |           44 | f
|                         0 |                           0
    101 | 12058149 |                    1 |         1 |          1 |      -1 |
2023-09-27 15:40:39.931437 | 2023-09-27 15:40:39.972826 |    41389 |          12 |
        0 |            12 |              0 |             0 |           77 | f
|                         0 |                           0
    101 | 12058149 |                    1 |         2 |          2 |      -1 |
2023-09-27 15:40:40.584412 | 2023-09-27 15:40:40.613982 |    29570 |          32 |
        0 |            32 |              0 |             0 |           25 | f
|                         0 |                           0
    101 | 12058149 |                    1 |         2 |          3 |      -1 |
2023-09-27 15:40:40.582038 | 2023-09-27 15:40:40.615758 |    33720 |           0 |
        0 |             0 |              0 |             0 |            1 | f
|                         0 |                           0
    101 | 12058149 |                    1 |         3 |          4 |      -1 |
2023-09-27 15:40:46.668766 | 2023-09-27 15:40:46.705456 |    36690 |          24 |
        0 |            15 |              0 |             0 |           17 | f
|                         0 |                           0
    101 | 12058149 |                    1 |         4 |          5 |      -1 |
2023-09-27 15:40:46.707209 | 2023-09-27 15:40:46.709176 |     1967 |           0 |
        0 |             0 |              0 |             0 |           18 | f
|                         0 |                           0
    101 | 12058149 |                    1 |         4 |          6 |      -1 |
2023-09-27 15:40:46.70656  | 2023-09-27 15:40:46.71289  |     6330 |           0 |
        0 |             0 |              0 |             0 |            0 | f
|                         0 |                           0
    101 | 12058149 |                    1 |         5 |          7 |      -1 |
2023-09-27 15:40:46.71405  | 2023-09-27 15:40:46.714343 |      293 |           0 |
        0 |             0 |              0 |             0 |            0 | f
|                         0 |                           0
    101 | 12058149 |                    2 |         0 |          0 |      -1 |
2023-09-27 15:40:52.083907 | 2023-09-27 15:40:52.087854 |     3947 |           0 |
        0 |             0 |              0 |             0 |           35 | f
|                         0 |                           0
    101 | 12058149 |                    2 |         1 |          1 |      -1 |
2023-09-27 15:40:52.089632 | 2023-09-27 15:40:52.091129 |     1497 |           0 |
        0 |             0 |              0 |             0 |           11 | f
|                         0 |                           0
    101 | 12058149 |                    2 |         1 |          2 |      -1 |
2023-09-27 15:40:52.089008 | 2023-09-27 15:40:52.091306 |     2298 |           0 |
        0 |             0 |              0 |             0 |            0 | f
|                         0 |                           0
```

```
    101 | 12058149 |                    3 |         0 |          0 |      -1 |
2023-09-27 15:40:56.882013 | 2023-09-27 15:40:56.897282 |    15269 |           0 |
        0 |             0 |              0 |             0 |           29 | f
|                         0 |                           0
    101 | 12058149 |                    3 |         1 |          1 |      -1 |
2023-09-27 15:40:59.718554 | 2023-09-27 15:40:59.722789 |     4235 |           0 |
        0 |             0 |              0 |             0 |           13 | f
|                         0 |                           0
    101 | 12058149 |                    3 |         2 |          2 |      -1 |
2023-09-27 15:40:59.800382 | 2023-09-27 15:40:59.807388 |     7006 |           0 |
        0 |             0 |              0 |             0 |           58 | f
|                         0 |                           0
    101 | 12058149 |                    3 |         3 |          3 |      -1 |
2023-09-27 15:41:06.488685 | 2023-09-27 15:41:06.493825 |     5140 |           0 |
        0 |             0 |              0 |             0 |           56 | f
|                         0 |                           0
    101 | 12058149 |                    3 |         3 |          4 |      -1 |
2023-09-27 15:41:06.486206 | 2023-09-27 15:41:06.497756 |    11550 |           0 |
        0 |             0 |              0 |             0 |            2 | f
|                         0 |                           0
    101 | 12058149 |                    3 |         4 |          5 |      -1 |
2023-09-27 15:41:06.499201 | 2023-09-27 15:41:06.500851 |     1650 |           0 |
        0 |             0 |              0 |             0 |           15 | f
|                         0 |                           0
    101 | 12058149 |                    3 |         4 |          6 |      -1 |
2023-09-27 15:41:06.498609 | 2023-09-27 15:41:06.500949 |     2340 |           0 |
        0 |             0 |              0 |             0 |            0 | f
|                         0 |                           0
    101 | 12058149 |                    3 |         5 |          7 |      -1 |
2023-09-27 15:41:06.502945 | 2023-09-27 15:41:06.503282 |      337 |           0 |
        0 |             0 |              0 |             0 |            0 | f
|                         0 |                           0
    101 | 12058149 |                    4 |         0 |          0 |      -1 |
2023-09-27 15:41:06.62899  | 2023-09-27 15:41:06.631452 |     2462 |           0 |
        0 |             0 |              0 |             0 |           22 | f
|                         0 |                           0
    101 | 12058149 |                    4 |         1 |          1 |      -1 |
2023-09-27 15:41:06.632313 | 2023-09-27 15:41:06.63391  |     1597 |           0 |
        0 |             0 |              0 |             0 |           20 | f
|                         0 |                           0
    101 | 12058149 |                    4 |         1 |          2 |      -1 |
2023-09-27 15:41:06.631726 | 2023-09-27 15:41:06.633813 |     2087 |           0 |
        0 |             0 |              0 |             0 |            0 | f
|                         0 |                           0
```

```
    101 | 12058149 |                    5 |         0 |          0 |      -1 |
 2023-09-27 15:41:12.571974 | 2023-09-27 15:41:12.584234 |    12260 |           0 |
         0 |             0 |              0 |             0 |           39 | f
 |                         0 |                            0
    101 | 12058149 |                    5 |         0 |          1 |      -1 |
 2023-09-27 15:41:12.569815 | 2023-09-27 15:41:12.585391 |    15576 |           0 |
         0 |             0 |              0 |             0 |            4 | f
 |                         0 |                            0
    101 | 12058149 |                    5 |         1 |          2 |      -1 |
 2023-09-27 15:41:13.758513 | 2023-09-27 15:41:13.76401  |     5497 |           0 |
         0 |             0 |              0 |             0 |           39 | f
 |                         0 |                            0
    101 | 12058149 |                    5 |         1 |          3 |      -1 |
 2023-09-27 15:41:13.749    | 2023-09-27 15:41:13.772987 |    23987 |           0 |
         0 |             0 |              0 |             0 |           32 | f
 |                         0 |                            0
    101 | 12058149 |                    5 |         2 |          4 |      -1 |
 2023-09-27 15:41:13.799526 | 2023-09-27 15:41:13.813506 |    13980 |           0 |
         0 |             0 |              0 |             0 |           62 | f
 |                         0 |                            0
    101 | 12058149 |                    5 |         2 |          5 |      -1 |
 2023-09-27 15:41:13.798823 | 2023-09-27 15:41:13.813651 |    14828 |           0 |
         0 |             0 |              0 |             0 |            0 | f
 |                         0 |                            0
(28 rows)
```

시스템 테이블 쿼리, 프로세스 및 세션 ID

시스템 테이블에 표시되는 쿼리, 프로세스 및 세션 ID를 분석할 때 다음 사항에 유의하세요.

- `query_id` 및 `query`와 같은 열의 쿼리 ID 값은 시간이 지남에 따라 재사용될 수 있습니다.
- `process_id`, `pid`, `session_id`와 같은 열의 프로세스 ID 또는 세션 ID 값은 시간이 지남에 따라 재사용될 수 있습니다.
- `transaction_id` 및 `xid`와 같은 열의 트랜잭션 ID 값은 고유합니다.

SVV 메타데이터 뷰

SVV 뷰는 데이터베이스 객체에 대한 정보를 포함하는 Amazon Redshift의 시스템 뷰입니다. 사용자 권한 또는 테이블 이름과 같은 정보는 저장하지만 사용자가 만든 관계와 조인하기 위한 것은 아닙니다.

> **Note**
>
> Amazon Redshift는 어떤 이유로든 데이터베이스 응답이 실패할 경우 오류가 아닌 경고를 보고합니다. Amazon Redshift는 데이터 공유에서 객체를 쿼리할 때 오류 메시지를 보내지 않습니다.

주제

- SVV_GEOMETRY_COLUMNS
- SVV_IAM_PRIVILEGES
- SVV_IDENTITY_PROVIDERS
- SVV_INTEGRATION
- SVV_INTEGRATION_TABLE_STATE
- SVV_INTERLEAVED_COLUMNS
- SVV_LANGUAGE_PRIVILEGES
- SVV_MASKING_POLICY
- SVV_ML_MODEL_INFO
- SVV_ML_MODEL_PRIVILEGES
- SVV_MV_DEPENDENCY
- SVV_MV_INFO
- SVV_QUERY_INFLIGHT
- SVV_QUERY_STATE
- SVV_REDSHIFT_COLUMNS
- SVV_REDSHIFT_DATABASES
- SVV_REDSHIFT_FUNCTIONS
- SVV_REDSHIFT_SCHEMA_QUOTA
- SVV_REDSHIFT_SCHEMAS
- SVV_REDSHIFT_TABLES
- SVV_RELATION_PRIVILEGES
- SVV_RLS_APPLIED_POLICY
- SVV_RLS_ATTACHED_POLICY
- SVV_RLS_POLICY
- SVV_RLS_RELATION
- SVV_ROLE_GRANTS
- SVV_ROLES
- SVV_SCHEMA_PRIVILEGES
- SVV_SCHEMA_QUOTA_STATE

- SVV_SYSTEM_PRIVILEGES
- SVV_TABLE_INFO
- SVV_TABLES
- SVV_TRANSACTIONS
- SVV_USER_GRANTS
- SVV_USER_INFO
- SVV_VACUUM_PROGRESS
- SVV_VACUUM_SUMMARY

SVV_ACTIVE_CURSORS

SVV_ACTIVE_CURSORS는 현재 열려있는 커서의 세부 정보를 표시합니다. 자세한 내용은 DECLARE 단원을 참조하십시오.

SVV_ACTIVE_CURSORS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오. 사용자는 자신이 연 커서만 볼 수 있습니다. 수퍼유저는 모든 커서를 볼 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	커서를 만든 사용자의 ID입니다.
cursor_name	varchar(128)	커서의 이름입니다.
transaction_id	bigint(128)	트랜잭션의 ID입니다.
session_id	정수	활성 커서가 있는 프로세스의 ID입니다.
declare_time	타임스탬프	커서가 선언된 시간입니다.
total_bytes	bigint	커서 결과 집합의 크기(바이트)입니다.

열 명칭	데이터 유형	설명
total_rows	bigint	커서 결과 집합의 행 수입니다.
fetched_rows	bigint	현재 커서 결과 집합에서 가져온 행 수입니다.
cursor_storage_limit_used_percent	정수	현재 커서가 사용 중인 디스크 공간의 백분율입니다.

SVV_ALL_COLUMNS

SVV_ALL_COLUMNS를 사용하여 SVV_REDSHIFT_COLUMNS에 표시된 대로 Amazon Redshift 테이블의 열 조합과 모든 외부 테이블의 모든 외부 열 통합 목록을 봅니다. Amazon Redshift 열에 대한 자세한 내용은 SVV_REDSHIFT_COLUMNS 섹션을 참조하세요.

SVV_ALL_COLUMNS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
database_name	varchar(128)	데이터베이스의 이름입니다.
schema_name	varchar(128)	스키마의 이름입니다.
table_name	varchar(128)	테이블의 이름
column_name	varchar(128)	열의 이름입니다.
ordinal_position	정수	테이블의 열 위치
column_default	varchar(4000)	열의 기본값
is_nullable	varchar(3)	열의 NULL 허용 여부를 나타내는 값 가능한 값은 yes와 no입니다.

열 명칭	데이터 유형	설명
data_type	varchar(128)	열의 데이터 형식.
character_maximum_length	정수	int
numeric_precision	정수	숫자 정밀도
numeric_scale	정수	숫자 스케일
remarks	varchar(256)	설명.

샘플 쿼리

다음 예에서는 SVV_ALL_COLUMNS의 출력을 반환합니다.

```
SELECT *
FROM svv_all_columns
WHERE database_name = 'tickit_db'
    AND TABLE_NAME = 'tickit_sales_redshift'
ORDER BY COLUMN_NAME,
    SCHEMA_NAME
LIMIT 5;

 database_name | schema_name |     table_name        | column_name | ordinal_position
 | column_default | is_nullable | data_type | character_maximum_length |
 numeric_precision | numeric_scale | remarks
 --------------+-------------+-----------------------+-------------+-----------------
+----------------+-------------+-----------+-------------------------
+------------------+---------------+---------
   tickit_db   |    public   | tickit_sales_redshift |    buyerid  |        4         |
               |      NO     |  integer  |                           |         32
  |       0       |
   tickit_db   |    public   | tickit_sales_redshift | commission  |        9         |
               |     YES     |  numeric  |                           |          8
  | 2       |
   tickit_db   |    public   | tickit_sales_redshift |    dateid   |        7         |
               |      NO     |  smallint |                           |         16
  |       0       |
   tickit_db   |    public   | tickit_sales_redshift |   eventid   |        5         |
               |      NO     |  integer  |                           |         32
  |       0       |
```

```
   tickit_db   |    public   | tickit_sales_redshift |    listid   |        2         |
               |      NO     |  integer  |                         |         32
  |       0       |
```

SVV_ALL_SCHEMAS

SVV_ALL_SCHEMAS를 사용하여 SVV_REDSHIFT_SCHEMAS에 표시된 Amazon Redshift 스키마의 조합과 모든 데이터베이스의 모든 외부 스키마의 통합 목록을 봅니다. Amazon Redshift 스키마에 대한 자세한 내용은 SVV_REDSHIFT_SCHEMAS 섹션을 참조하세요.

SVV_ALL_SCHEMAS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
database_name	varchar(128)	스키마가 존재하는 데이터베이스의 이름입니다.
schema_name	varchar(128)	스키마의 이름입니다.
schema_owner	정수	스키마 소유자의 사용자 ID입니다. 사용자 ID에 대한 자세한 내용은 PG_USER_INFO 단원을 참조하세요.
schema_type	varchar(128)	스키마의 형식입니다. 가능한 값은 external, local 및 shared 스키마입니다.
schema_acl	varchar(128)	스키마에 대해 지정된 사용자 또는 사용자 그룹에 대한 권한을 정의하는 문자열입니다.
source_database	varchar(128)	외부 스키마의 원본 데이터베이스의 이름입니다.

열 명칭	데이터 유형	설명
schema_option	varchar(256)	스키마의 옵션입니다. 외부 스키마 속성입니다.

샘플 쿼리

다음 예에서는 SVV_ALL_SCHEMAS의 출력을 반환합니다.

```
SELECT *
FROM svv_all_schemas
WHERE database_name = 'tickit_db'
ORDER BY database_name,
    SCHEMA_NAME;


 database_name |    schema_name     | schema_owner | schema_type | schema_acl |
 source_database | schema_option
---------------+--------------------+--------------+-------------+------------
+-----------------+---------------
   tickit_db   |       public       |       1      |   shared    |            |
         |
```

SVV_ALL_TABLES

SVV_ALL_TABLES를 사용하여 SVV_REDSHIFT_TABLES에 표시된 대로 Amazon Redshift 테이블의 조합과 모든 외부 스키마의 모든 외부 테이블의 통합 목록을 봅니다. Amazon Redshift 테이블에 대한 자세한 내용은 SVV_REDSHIFT_TABLES 섹션을 참조하세요.

SVV_ALL_TABLES는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
database_name	varchar(128)	테이블이 속한 데이터베이스의 이름입니다.

열 명칭	데이터 유형	설명
schema_name	varchar(128)	테이블의 스키마 이름
table_name	varchar(128)	테이블의 이름
table_acl	varchar(128)	테이블에 대해 지정된 사용자 또는 사용자 그룹에 대한 권한을 정의하는 문자열입니다.
table_type	varchar(128)	테이블의 형식입니다. 가능한 값은 view, base table, external table 및 shared table입니다.
remarks	varchar(256)	설명.

샘플 쿼리

다음 예에서는 SVV_ALL_TABLES의 출력을 반환합니다.

```
SELECT *
FROM svv_all_tables
WHERE database_name = 'tickit_db'
ORDER BY TABLE_NAME,
    SCHEMA_NAME
LIMIT 5;

 database_name | schema_name |        table_name        | table_type | table_acl |
 remarks
---------------+-------------+--------------------------+------------+-----------
+---------
   tickit_db   |    public   | tickit_category_redshift |    TABLE   |           |
   tickit_db   |    public   |   tickit_date_redshift   |    TABLE   |           |
   tickit_db   |    public   |   tickit_event_redshift  |    TABLE   |           |
   tickit_db   |    public   | tickit_listing_redshift  |    TABLE   |           |
   tickit_db   |    public   |   tickit_sales_redshift  |    TABLE   |           |
```

table_acl 값이 null인 경우 해당 테이블에 액세스 권한이 명시적으로 부여되지 않은 것입니다.

SVV_ALTER_TABLE_RECOMMENDATIONS

테이블에 대한 현재 Amazon Redshift Advisor 권장 사항을 기록합니다. 이 뷰는 자동 최적화에 대해 정의되었는지 여부에 관계없이 모든 테이블에 대한 권장 사항을 보여줍니다. 자동 최적화를 위해 테이블이 정의되어 있는지 보려면 SVV_TABLE_INFO 섹션을 참조하세요. 현재 세션의 데이터베이스에 표시되는 테이블에 대한 항목만 나타납니다. Amazon Redshift 또는 사용자에 의해 적용된 권장 사항은 더 이상 뷰에 표시되지 않습니다.

SVV_ALTER_TABLE_RECOMMENDATIONS는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
형식	character (30)	권장 사항의 유형입니다. 가능한 값은 distkey와 sortkey입니다.
데이터베이스	character (128)	데이터베이스 이름입니다.
table_id	정수	테이블 식별자입니다.
group_id	정수	권장 사항 집합의 그룹 번호입니다. 최대 이점을 얻으려면 그룹의 권장 사항을 모두 적용해야 합니다. 가능한 값은 정렬 키 권장 사항의 경우 -1이고, 배포 키 권장 사항의 경우 0보다 큰 숫자입니다.
ddl	character (1024)	권장 사항을 적용하기 위해 실행해야 하는 SQL 문입니다.
auto_eligible	character (1)	이 값은 권장 사항이 Amazon Redshift를 자동으로 실행할 수 있는지 여부를 나타냅니다. 이 값이 `t`이면 true이고 `f`이면 false입니다.

샘플 쿼리

다음 예에서 결과의 행은 배포 키 및 정렬 키에 대한 권장 사항을 보여줍니다. 또한 행은 권장 사항이 Amazon Redshift에서 자동으로 적용할 수 있는지 여부를 보여줍니다.

```
select type, database, table_id, group_id, ddl, auto_eligible
from svv_alter_table_recommendations;
```

```
type      | database | table_id | group_id | ddl

                  | auto_eligible
diststyle | db0      | 117884   | 2        | ALTER TABLE "sch"."dp21235_tbl_1" ALTER
DISTSTYLE KEY DISTKEY "c0"
                     | f
diststyle | db0      | 117892   | 2        | ALTER TABLE "sch"."dp21235_tbl_1" ALTER
DISTSTYLE KEY DISTKEY "c0"
                     | f
diststyle | db0      | 117885   | 1        | ALTER TABLE "sch"."catalog_returns"
ALTER DISTSTYLE KEY DISTKEY "cr_sold_date_sk", ALTER COMPOUND SORTKEY
("cr_sold_date_sk","cr_returned_time_sk") | t
sortkey   | db0      | 117890   | -1       | ALTER TABLE "sch"."customer_addresses"
ALTER COMPOUND SORTKEY ("ca_address_sk")
                      | t
```

SVV_ATTACHED_MASKING_POLICY

SVV_ATTACHED_MASKING_POLICY를 사용하여 현재 연결된 데이터베이스에 정책이 연결된 모든 관계 및 역할/사용자를 봅니다.

`sys:secadmin` 역할이 부여된 슈퍼 사용자와 사용자만 SVV_ATTACHED_MASKING_POLICY를 볼 수 있습니다. 일반 사용자에게는 0개의 행이 표시됩니다.

테이블 열

열 명칭	데이터 유형	설명
policy_name	텍스트	테이블에 연결할 마스킹 정책 이름입니다.
schema_name	텍스트	정책이 연결될 테이블의 스키마입니다.
table_name	텍스트	정책이 연결될 테이블의 이름입니다.

열 명칭	데이터 유형	설명
table_type	텍스트	정책이 연결될 테이블의 형식입니다.
grantor	텍스트	정책을 연결한 사용자의 이름입니다.
grantee	텍스트	정책이 연결된 사용자/역할의 이름입니다.
grantee_type	텍스트	피부여자의 유형입니다. 역할, 사용자 또는 퍼블릭일 수 있습니다.
priority	int	연결된 정책의 우선 순위입니다.
input_columns	텍스트	연결된 정책의 입력 열 속성입니다.
output_columns	텍스트	연결된 정책의 출력 열 속성입니다.

내부 함수

SVV_ATTACHED_MASKING_POLICY는 다음과 같은 내부 함수를 지원합니다.

mask_get_policy_for_role_on_column

지정된 열/역할 쌍에 적용되는 가장 높은 우선 순위 정책을 가져옵니다.

구문

```
mask_get_policy_for_role_on_column
                        (relschema,
                        relname,
                        colname,
                        rolename);
```

파라미터

relschema

정책이 있는 스키마의 이름입니다.

relname

정책이 있는 테이블의 이름입니다.

colname

정책이 연결된 열의 이름입니다.

rolename

정책이 연결된 역할의 이름입니다.

mask_get_policy_for_user_on_column

지정된 열/사용자 쌍에 적용되는 가장 높은 우선 순위 정책을 가져옵니다.

구문

```
mask_get_policy_for_user_on_column
                        (relschema,
                        relname,
                        colname,
                        username);
```

파라미터

relschema

정책이 있는 스키마의 이름입니다.

relname

정책이 있는 테이블의 이름입니다.

colname

정책이 연결된 열의 이름입니다.

rolename

정책이 연결된 사용자의 이름입니다.

SVV_COLUMNS

Late Binding 보기를 포함하여 로컬 및 외부 테이블과 보기의 열에 대한 카탈로그 정보를 보려면 SVV_COLUMNS를 사용합니다.

SVV_COLUMNS는 모든 사용자가 볼 수 있습니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

SVV_COLUMNS 보기는 시스템 카탈로그 테이블(PG 접두사가 포함된 테이블)의 테이블 메타데이터와 SVV_EXTERNAL_COLUMNS 시스템 보기를 결합합니다. 시스템 카탈로그 테이블은 Amazon Redshift 데이터베이스 테이블을 설명합니다. SVV_EXTERNAL_COLUMNS는 Amazon Redshift Spectrum과 사용되는 외부 테이블을 설명합니다.

모든 사용자는 시스템 카탈로그 테이블의 모든 행을 볼 수 있습니다. 기본 사용자는 자신에게 액세스 권한이 있는 외부 테이블에 한해 SVV_EXTERNAL_COLUMNS 보기에서 열 정의를 볼 수 있습니다. 기본 사용자도 시스템 카탈로그 테이블에서 테이블 메타데이터를 볼 수는 있지만, 데이터를 선택할 수 있는 테이블은 본인이 소유한 사용자 정의 테이블이나 액세스 권한이 있는 사용자 정의 테이블뿐입니다.

테이블 열

열 명칭	데이터 유형	설명
table_catalog	텍스트	테이블이 속한 카탈로그 이름
table_schema	텍스트	테이블의 스키마 이름
table_name	텍스트	테이블의 이름
column_name	텍스트	열 이름.
ordinal_position	int	테이블의 열 위치
column_default	텍스트	열의 기본값
is_nullable	텍스트	열의 NULL 허용 여부를 나타내는 값
data_type	텍스트	열의 데이터 형식.
character_maximum_length	int	int

열 명칭	데이터 유형	설명
numeric_precision	int	숫자 정밀도 data_type 열이 숫자인 경우 이 열은 전체 값의 유효 자릿수를 반환합니다.
numeric_precision_radix	int	숫자 정밀도 기수 data_type 열이 숫자인 경우 이 열은 numeric_precision 및 numeric_scale 열의 기본값을 반환합니다.
numeric_scale	int	숫자 스케일 data_type 열이 숫자인 경우 이 열은 소수점 값의 유효 자릿수를 반환합니다.
datetime_precision	int	날짜/시간 정밀도
interval_type	텍스트	간격 유형
interval_precision	텍스트	간격 정밀도
character_set_catalog	텍스트	문자 세트 카탈로그
character_set_schema	텍스트	문자 세트 스키마
character_set_name	텍스트	문자 세트 이름
collation_catalog	텍스트	콜레이션 카탈로그
collation_schema	텍스트	콜레이션 스키마
collation_name	텍스트	콜레이션 이름
domain_name	텍스트	도메인 이름.
remarks	텍스트	설명.

SVV_COLUMN_PRIVILEGES

현재 데이터베이스의 사용자, 롤 및 그룹에 명시적으로 부여된 열 권한을 보려면 SVV_COLUMN_PRIVILEGES를 사용합니다.

SVV_COLUMN_PRIVILEGES는 다음 사용자에게 표시됩니다.

- 슈퍼 사용자
- ACCESS SYSTEM TABLE 권한이 있는 사용자

다른 사용자는 액세스할 수 있거나 소유한 ID만 볼 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
namespace_name	텍스트	지정된 관계가 존재하는 네임스페이스의 이름입니다.
relation_name	텍스트	관계의 이름입니다.
column_name	텍스트	열 이름.
privilege_type	텍스트	권한의 유형입니다. 가능한 값은 SELECT 또는 UPDATE입니다.
identity_id	정수	자격 증명의 ID입니다. 가능한 값은 사용자 ID, 역할 ID 또는 그룹 ID입니다.
identity_name	텍스트	자격 증명의 이름입니다.
identity_type	텍스트	자격 증명의 유형입니다. 가능한 값은 사용자, 역할, 그룹 또는 퍼블릭입니다.

샘플 쿼리

다음 예에서는 SVV_COLUMN_PRIVILEGES의 결과를 표시합니다.

```
SELECT
 namespace_name,relation_name,COLUMN_NAME,privilege_type,identity_name,identity_type
FROM svv_column_privileges WHERE relation_name = 'lineitem';

 namespace_name | relation_name | column_name | privilege_type | identity_name |
 identity_type
----------------+---------------+-------------+----------------+---------------
+----------------
    public      |   lineitem    | l_orderkey  |     SELECT     |    reguser    |
 user
    public      |   lineitem    | l_orderkey  |     SELECT     |     role1     |
 role
    public      |   lineitem    | l_partkey   |     SELECT     |    reguser    |
 user
    public      |   lineitem    | l_partkey   |     SELECT     |     role1     |
 role
```

SVV_COPY_JOB_INTEGRATIONS

SVV_COPY_JOB_INTEGRATIONS을 사용하여 S3 이벤트 통합의 세부 정보를 봅니다.

이 뷰에는 만들어진 S3 이벤트 통합이 포함되어 있습니다.

SVV_COPY_JOB_INTEGRATIONS은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
job_owner	정수	작업 소유자의 식별자입니다.
channel_arn	name	통합 식별자입니다.
버킷	텍스트	통합에 연결된 Amazon S3 버킷의 이름입니다.
channel state	name	통합 상태입니다. 유효한 값: `Pending`, `Established` , 및 `Inactive`

열 명칭	데이터 유형	설명
db_name	name	종속 객체의 데이터베이스 이름입니다.
job_name	텍스트	작업의 이름입니다.
job_state	정수	작업의 상태입니다. 유효한 값: 활성인 경우 0, 보류 중인 경우 1

다음 예시에서는 현재 데이터베이스에 대한 S3 통합을 반환합니다.

```
SELECT * FROM SVV_COPY_JOB_INTEGRATIONS WHERE db_name = pg_catalog.current_database();
```

SVV_DATABASE_PRIVILEGES

SVV_DATABASE_PRIVILEGES를 사용하여 Amazon Redshift 클러스터의 사용자, 역할 및 그룹에 명시적으로 부여된 데이터베이스 권한을 확인합니다.

SVV_DATABASE_PRIVILEGES는 다음 사용자에게 표시됩니다.

- 슈퍼 사용자
- ACCESS SYSTEM TABLE 권한이 있는 사용자

다른 사용자는 액세스할 수 있거나 소유한 ID만 볼 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
database_name	텍스트	데이터베이스의 이름입니다.
privilege_type	텍스트	권한의 유형입니다. privilege_scope가 데이터베이스인 권한의 경우 가능한 값은 USAGE, CREATE, TEMPORARY, TEMP, ALTER입니다. 데이

열 명칭	데이터 유형	설명
		터베이스 이외의 privilege_scope 값의 경우 가능한 값에는 권한 범위에서 사용할 수 있는 모든 권한 유형이 포함됩니다.
identity_id	정수	자격 증명의 ID입니다. 가능한 값은 사용자 ID, 역할 ID 또는 그룹 ID입니다.
identity_name	텍스트	자격 증명의 이름입니다.
identity_type	텍스트	자격 증명의 유형입니다. 가능한 값은 사용자, 역할, 그룹 또는 퍼블릭입니다.
admin_option	boolean	사용자가 다른 사용자 및 역할에 권한을 부여할 수 있는지 여부를 나타내는 값. 역할 및 그룹 자격 증명 유형에 대해서는 항상 false입니다.
privilege_scope	텍스트	privilege_type에 지정된 권한의 범위입니다. 가능한 값은 다음과 같습니다. • 데이터베이스 • SCHEMAS • TABLES • FUNCTIONS • LANGUAGES 범위 지정 권한에 관한 자세한 내용은 범위가 지정된 권한 섹션으로 이동해 참조하시기 바랍니다.

샘플 쿼리

다음 예에서는 SVV_DATABASE_PRIVILEGES의 결과를 표시합니다.

```
SELECT database_name,privilege_type,identity_name,identity_type,admin_option FROM
 svv_database_privileges
WHERE database_name = 'test_db';

 database_name | privilege_type | identity_name | identity_type | admin_option
```

```
--------------+----------------+--------------+--------------+-------------
     test_db   |     CREATE     |     reguser  |      user    |     False
     test_db   |     CREATE     |      role1   |      role    |     False
     test_db   |     TEMP       |      public  |      public  |     False
     test_db   |     TEMP       |      role1   |      role    |     False
```

SVV_DATASHARE_PRIVILEGES

SVV_DATASHARE_PRIVILEGES를 사용하여 Amazon Redshift 클러스터의 사용자, 역할 및 그룹에 명시적으로 부여된 datashare 권한을 확인합니다.

SVV_DATASHARE_PRIVILEGES는 다음 사용자에게 표시됩니다.

- 슈퍼 사용자
- ACCESS SYSTEM TABLE 권한이 있는 사용자

다른 사용자는 액세스할 수 있거나 소유한 ID만 볼 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
datashare_name	텍스트	datashare의 이름입니다.
privilege_type	텍스트	권한의 유형입니다. 가능한 값은 ALTER 또는 SHARE입니다.
identity_id	정수	자격 증명의 ID입니다. 가능한 값은 사용자 ID, 역할 ID 또는 그룹 ID입니다.
identity_name	텍스트	자격 증명의 이름입니다.
identity_type	텍스트	자격 증명의 유형입니다. 가능한 값은 사용자, 역할, 그룹 또는 퍼블릭입니다.
admin_option	boolean	사용자가 다른 사용자 및 역할에 권한을 부여할 수 있는지 여부를 나타내는 값. 역할 및 그룹 자격 증명 유형에 대해서는 항상 false입니다.

샘플 쿼리

다음 예에서는 SVV_DATASHARE_PRIVILEGES의 결과를 표시합니다.

```
SELECT datashare_name,privilege_type,identity_name,identity_type,admin_option FROM
 svv_datashare_privileges
WHERE datashare_name = 'demo_share';

 datashare_name | privilege_type |  identity_name | identity_type | admin_option
----------------+----------------+----------------+---------------+--------------
   demo_share   |     ALTER      |    superuser   |     user      |   False
   demo_share   |     ALTER      |    reguser     |     user      |   False
```

SVV_DATASHARES

SVV_DATASHARES를 사용하여 클러스터에서 생성된 datashare 및 클러스터와 공유된 datashare의 목록을 봅니다.

SVV_DATASHARES는 다음 사용자에게 표시됩니다.

- 슈퍼 사용자
- 데이터 공유 소유자
- 데이터 공유에 대한 ALTER 또는 USAGE 권한이 있는 사용자

다른 사용자는 행을 볼 수 없습니다. ALTER 및 USAGE 권한에 대한 자세한 내용은 GRANT 섹션을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
share_name	varchar(128)	datashare의 이름입니다.
share_id	정수	datashare의 ID입니다.
share_owner	정수	datashare의 소유자입니다.
source_database	varchar(128)	이 datashare의 원본 데이터베이스입니다.

열 명칭	데이터 유형	설명
consumer_database	varchar(128)	이 datashare에서 생성되는 소비자 데이터베이스입니다.
share_type	varchar(8)	datashare의 형식입니다. 가능한 값은 INBOUND와 OUTBOUND입니다.
createdate	시간대 미포함 TIMESTAMP	datashare가 생성된 날짜입니다.
is_publicaccessible	boolean	공개적으로 액세스 가능한 클러스터에 datashare를 공유할 수 있는지 여부를 지정하는 속성입니다.
share_acl	varchar(256)	datashare에 대해 지정된 사용자 또는 사용자 그룹에 대한 권한을 정의하는 문자열입니다.
producer_account	varchar(16)	datashare 생산자 계정의 ID입니다.
producer_namespace	varchar(64)	datashare 생산자 클러스터의 고유 클러스터 식별자입니다.
managed_by	varchar(64)	datashare를 관리하는 AWS 서비스를 지정하는 속성입니다.

사용 노트

추가 메타데이터 검색 – `share_owner` 열에 반환된 정수를 사용하면 SVL_USER_INFO의 `usesysid`와 조인하여 데이터 공유 소유자에 관한 데이터를 가져올 수 있습니다. 여기에는 이름 및 추가 속성이 포함됩니다.

샘플 쿼리

다음 예에서는 SVV_DATASHARES에 대한 출력을 반환합니다.

```
SELECT share_owner, source_database, share_type, is_publicaccessible
FROM svv_datashares
WHERE share_name LIKE 'tickit_datashare%'
AND source_database = 'dev';

  share_owner | source_database | share_type  | is_publicaccessible
--------------+-----------------+-------------+---------------------
     100      |      dev        |   OUTBOUND  |        True
(1 rows)
```

다음 예에서는 아웃바운드 datashare에 대한 SVV_DATASHARES의 출력을 반환합니다.

```
SELECT share_name, share_owner, btrim(source_database), btrim(consumer_database),
 share_type, is_publicaccessible, share_acl, btrim(producer_account),
 btrim(producer_namespace), btrim(managed_by) FROM svv_datashares WHERE share_type =
 'OUTBOUND';

   share_name   | share_owner | source_database | consumer_database | share_type |
 is_publicaccessible | share_acl | producer_account|         producer_namespace
   | managed_by
----------------+-------------+-----------------+-------------------
+------------+---------------------+-----------+-----------------
+--------------------------------------+------------
    salesshare  |      1      |       dev       |                   |  OUTBOUND  |
   True         |           |   123456789012  | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d |

 marketingshare |      1      |       dev       |                   |  OUTBOUND  |
   True         |           |   123456789012  | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d |
```

다음 예에서는 인바운드 datashare에 대한 SVV_DATASHARES의 출력을 반환합니다.

```
SELECT share_name, share_owner, btrim(source_database), btrim(consumer_database),
 share_type, is_publicaccessible, share_acl, btrim(producer_account),
 btrim(producer_namespace), btrim(managed_by) FROM svv_datashares WHERE share_type =
 'INBOUND';

  share_name    | share_owner | source_database | consumer_database | share_type |
 is_publicaccessible | share_acl | producer_account |         producer_namespace
    | managed_by
```

```
---------------+-------------+-----------------+-------------------
+------------+--------------------+-----------+-----------------
+--------------------------------------+------------
  salesshare    |             |                 |                    |  INBOUND   |
   False         |           |  123456789012    | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d
 |
 marketingshare |             |                 |                    |  INBOUND   |
  False         |           |  123456789012    | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d |
 ADX
```

SVV_DATASHARE_CONSUMERS

SVV_DATASHARE_CONSUMERS를 사용하여 클러스터에서 생성된 datashare에 대한 소비자 목록을 봅니다.

SVV_DATASHARE_CONSUMERS는 다음 사용자에게 표시됩니다.

- 슈퍼 사용자
- 데이터 공유 소유자
- 데이터 공유에 대한 ALTER 또는 USAGE 권한이 있는 사용자

다른 사용자는 행을 볼 수 없습니다. ALTER 및 USAGE 권한에 대한 자세한 내용은 GRANT 섹션을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
share_name	varchar(128)	datashare의 이름입니다.
consumer_account	varchar(16)	datashare 소비자의 계정 ID입니다.
consumer_namespace	varchar(64)	datashare 소비자 클러스터의 고유 클러스터 식별자입니다.
share_date	시간대 미포함 TIMESTAMP	datashare가 공유된 날짜입니다.

샘플 쿼리

다음 예에서는 SVV_DATASHARE_CONSUMERS의 출력을 반환합니다.

```
SELECT count(*)
FROM svv_datashare_consumers
WHERE share_name LIKE 'tickit_datashare%';

1
```

SVV_DATASHARE_OBJECTS

SVV_DATASHARE_OBJECTS를 사용하여 클러스터에서 생성되었거나 클러스터와 공유된 모든 datashare의 객체 목록을 봅니다.

SVV_DATASHARE_OBJECTS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

데이터 공유 목록을 보는 방법에 대한 자세한 내용은 SVV_DATASHARES를 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
share_type	varchar(8)	지정된 datashare의 형식입니다. 가능한 값은 OUTBOUND와 INBOUND입니다.
share_name	varchar(128)	datashare의 이름입니다.
object_type	varchar(64)	지정된 객체의 형식입니다. 가능한 값은 schema, table, view, late binding view, materialized view 및 function입니다.
객체 이름	varchar(512)	객체의 이름. 객체 이름은 schema1.t1과 같은 스키마 이름을 포함하도록 확장됩니다.

열 명칭	데이터 유형	설명
producer_account	varchar(16)	datashare 생산자 계정의 ID입니다.
producer_namespace	varchar(64)	datashare 생산자 클러스터의 고유 클러스터 식별자입니다.
include_new	boolean	지정된 스키마에서 생성된 향후 테이블, 뷰 또는 SQL 사용자 정의 함수(UDF)를 datashare에 추가할지 여부를 지정하는 속성입니다. 이 파라미터는 OUTBOUND datashare와 datashare의 스키마 형식에만 관련이 있습니다.

샘플 쿼리

다음 예에서는 SVV_DATASHARE_OBJECTS에 대한 출력을 반환합니다.

```
SELECT share_type,
    btrim(share_name)::varchar(16) AS share_name,
    object_type,
    object_name
FROM svv_datashare_objects
WHERE share_name LIKE 'tickit_datashare%'
AND object_name LIKE '%tickit%'
ORDER BY object_name
LIMIT 5;

 share_type |     share_name     | object_type |          object_name
------------+--------------------+-------------+--------------------------------
 OUTBOUND   |  tickit_datashare  |    table    |  public.tickit_category_redshift
 OUTBOUND   |  tickit_datashare  |    table    |  public.tickit_date_redshift
 OUTBOUND   |  tickit_datashare  |    table    |  public.tickit_event_redshift
 OUTBOUND   |  tickit_datashare  |    table    |  public.tickit_listing_redshift
 OUTBOUND   |  tickit_datashare  |    table    |  public.tickit_sales_redshift
```

```
SELECT * FROM SVV_DATASHARE_OBJECTS WHERE share_name like 'sales%';

share_type | share_name | object_type | object_name  | producer_account |
 producer_namespace          | include_new
-----------+------------+-------------+--------------+-----------------
+--------------------------------------+-------------
 OUTBOUND  | salesshare | schema      | public       | 123456789012     |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d |      t
 OUTBOUND  | salesshare | table       | public.sales | 123456789012     |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d |
```

SVV_DEFAULT_PRIVILEGES

SVV_DEFAULT_PRIVILEGES를 사용하여 Amazon Redshift 클러스터에서 사용자가 액세스할 수 있는 기본 권한을 볼 수 있습니다.

SVV_DEFAULT_PRIVILEGES는 다음 사용자에게 표시됩니다.

- 슈퍼 사용자
- ACCESS SYSTEM TABLE 권한이 있는 사용자

다른 사용자는 자신에게 부여된 기본 권한만 볼 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
schema_name	텍스트	스키마의 이름입니다.
object_type	텍스트	객체 유형입니다. 가능한 값은 RELATION, FUNCTION 또는 PROCEDURE입니다.
owner_id	정수	소유자 ID입니다. 가능한 값은 사용자 ID입니다.
owner_name	텍스트	소유자 이름입니다.
owner_type	텍스트	소유자 유형입니다. 가능한 값은 사용자입니다.

열 명칭	데이터 유형	설명
privilege_type	텍스트	권한 유형입니다. 가능한 값은 INSERT, SELECT, UPDATE, DELETE, RULE, REFERENCES TRIGGER, DROP 및 EXECUTE입니다.
grantee_id	정수	피부여자 ID입니다. 가능한 값은 사용자 ID, 역할 ID 및 그룹 ID입니다.
grantee_type	텍스트	피부여자 유형입니다. 가능한 값은 사용자, 역할 및 퍼블릭입니다.
admin_option	boolean	사용자가 다른 사용자 및 역할에 권한을 부여할 수 있는지 여부를 나타내는 값입니다. 역할 및 그룹 유형에 대해서는 항상 false입니다.

샘플 쿼리

다음 예에서는 SVV_DEFAULT_PRIVILEGES의 출력을 반환합니다.

```
SELECT * from svv_default_privileges;

 schema_name |    object_type    | owner_id | owner_name | owner_type | privilege_type
 | grantee_id | grantee_name | grantee_type | admin_option
-------------+-------------------+--------- +------------+------------+----------------
+------------+--------------+--------------+-------------+
   public    |     RELATION      |    106   |     u1     |    user    |     UPDATE
 |     107    |      u2      |     user     |      f      |
   public    |     RELATION      |    106   |     u1     |    user    |     SELECT
 |     107    |      u2      |     user     |      f      |
```

SVV_DISKUSAGE

Amazon Redshift는 STV_TBL_PERM 테이블과 STV_BLOCKLIST 테이블을 조인하여 SVV_DISKUSAGE 시스템 뷰를 생성합니다. SVV_DISKUSAGE 뷰에는 데이터베이스의 테이블에 대한 데이터 할당에 관한 정보가 포함되어 있습니다.

다음 예에 나온 것처럼 SVV_DISKUSAGE에 집계 쿼리를 사용하여 데이터베이스, 테이블, 조각 또는 열마다 할당되는 디스크 블록의 수를 결정합니다. 각 데이터 블록이 1MB씩 사용합니다. 또한 STV_PARTITIONS를 사용하여 디스크 사용률에 대한 요약 정보를 볼 수 있습니다.

SVV_DISKUSAGE는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> 이 보기는 프로비저닝된 클러스터를 쿼리할 때만 사용할 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
db_id	정수	데이터베이스 ID.
name	character (72)	테이블 이름
slice	정수	테이블에 할당되는 데이터 조각.
col	정수	0부터 시작되는 열 인덱스. 생성하는 모든 테이블에는 INSERT_XID, DELETE_XID, ROW_ID(OID)라는 3개의 숨겨진 열이 추가되어 있습니다. 3개의 사용자 정의 열이 있는 테이블에는 6개의 실제 열이 포함되며, 사용자 정의 열은 내부적으로 0, 1, 2로 번호 지정됩니다. 이 예에서 INSERT_XID, DELETE_XID 및 ROW_ID 열은 각각 3, 4, 5로 번호 지정됩니다.
tbl	정수	테이블 ID.
blocknum	정수	데이터 블록의 ID.
num_values	정수	블록에 포함된 값의 수.
minvalue	bigint	블록에 포함된 최솟값.
maxvalue	bigint	블록에 포함된 최댓값.
sb_pos	정수	디스크 상의 슈퍼 블록 위치에 대한 내부 식별자.

열 명칭	데이터 유형	설명
pinned	정수	블록이 사전 로드의 일환으로 메모리에 고정되어 있는지 여부. 0 = false, 1 = true. 기본값은 false입니다.
on_disk	정수	블록이 디스크에 자동으로 저장되는지 여부. 0 = false, 1 = true. 기본값은 false입니다.
수정됨	정수	블록이 수정되었는지 여부. 0 = false, 1 = true. 기본값은 false입니다.
hdr_modified	정수	블록 헤더가 수정되었는지 여부. 0 = false, 1 = true. 기본값은 false입니다.
unsorted	정수	블록이 정렬되지 않았는지 여부. 0 = false, 1 = true. 기본값은 true입니다.
tombstone	정수	내부용.
preferred_diskno	정수	디스크에 결함이 발생하지 않은 경우, 블록이 있어야 할 디스크 번호. 디스크가 고쳐지면 블록은 이 디스크로 다시 이동합니다.
임시	정수	블록에 임시 테이블 또는 중간 쿼리 결과 같은 임시 데이터가 포함되는지 여부. 0 = false, 1 = true. 기본값은 false입니다.
newblock	정수	블록이 새것인지(true) 또는 디스크에 한번도 커밋되지 않았는지(false) 여부. 0 = false, 1 = true.

샘플 쿼리

SVV_DISKUSAGE는 할당된 디스크 블록당 하나의 행을 포함하므로 모든 행을 선택하는 쿼리는 매우 많은 수의 행을 반환할 수 있습니다. SVV_DISKUSAGE에는 집계 쿼리만을 사용하는 것이 좋습니다.

USERS 테이블의 열 6(EMAIL 열)에 할당된 블록의 최대 개수를 반환합니다.

```
select db_id, trim(name) as tablename, max(blocknum)
from svv_diskusage
where name='users' and col=6
group by db_id, name;
```

```
db_id  | tablename | max
--------+-----------+-----
175857 | users     |   2
(1 row)
```

다음 쿼리는 SALESNEW라고 하는 10열짜리 큰 테이블의 모든 열에 대해 비슷한 결과를 반환합니다. (열 10부터 12까지의 마지막 3행은 숨겨진 메타데이터 열을 위한 것입니다.)

```
select db_id, trim(name) as tablename, col, tbl, max(blocknum)
from svv_diskusage
where name='salesnew'
group by db_id, name, col, tbl
order by db_id, name, col, tbl;

db_id  | tablename  | col |  tbl   | max
--------+------------+-----+--------+-----
175857 | salesnew   |   0 | 187605 | 154
175857 | salesnew   |   1 | 187605 | 154
175857 | salesnew   |   2 | 187605 | 154
175857 | salesnew   |   3 | 187605 | 154
175857 | salesnew   |   4 | 187605 | 154
175857 | salesnew   |   5 | 187605 |  79
175857 | salesnew   |   6 | 187605 |  79
175857 | salesnew   |   7 | 187605 | 302
175857 | salesnew   |   8 | 187605 | 302
175857 | salesnew   |   9 | 187605 | 302
175857 | salesnew   |  10 | 187605 |   3
175857 | salesnew   |  11 | 187605 |   2
175857 | salesnew   |  12 | 187605 | 296
(13 rows)
```

SVV_EXTERNAL_COLUMNS

외부 테이블에 있는 열의 세부 정보를 보려면 SVV_EXTERNAL_COLUMNS를 사용합니다. 사용자가 액세스할 수 있는 연결되지 않은 데이터베이스의 테이블에서 모든 열에 대한 세부 정보를 보려면 데이터베이스 간 쿼리에도 SVV_EXTERNAL_COLUMNS를 사용합니다.

SVV_EXTERNAL_COLUMNS는 모든 사용자가 볼 수 있습니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
redshift_database_name	텍스트	로컬 Amazon Redshift 데이터베이스의 이름입니다.
schemaname	텍스트	외부 테이블의 Amazon Redshift 외부 스키마 이름.
tablename	텍스트	외부 테이블 이름.
columnname	텍스트	열 이름.
external_type	텍스트	열의 데이터 형식.
columnnum	정수	외부 열 번호(1부터 시작).
part_key	정수	열이 파티션 키인 경우, 키의 순서. 열이 파티션이 아닌 경우, 값은 **0**입니다.
is_nullable	텍스트	열이 NULL 허용인지 여부를 정의합니다. 일부 값은 `true`, `false` 또는 정보를 나타내지 않는 빈 문자열 " "입니다.

SVV_EXTERNAL_DATABASES

외부 데이터베이스에 대한 세부 정보를 보려면 SVV_EXTERNAL_DATABASES를 사용합니다.

SVV_EXTERNAL_DATABASES는 모든 사용자가 볼 수 있습니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
eskind	정수	데이터베이스의 외부 카탈로그 형식. **1**은 데이터 카탈로그를 나타내고, **2**는 Hive 메타스토어를 나타냅니다.
esoptions	텍스트	데이터베이스가 상주하는 카탈로그의 세부 정보.
databasename	텍스트	외부 카탈로그에 있는 데이터베이스의 이름.
location	텍스트	데이터베이스 위치
parameters	텍스트	데이터베이스 파라미터

SVV_EXTERNAL_PARTITIONS

외부 테이블에 있는 파티션의 세부 정보를 보려면 SVV_EXTERNAL_PARTITIONS를 사용합니다.

SVV_EXTERNAL_PARTITIONS는 모든 사용자가 볼 수 있습니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
schemaname	텍스트	파티션이 지정된 외부 테이블의 Amazon Redshift 외부 스키마 이름.
tablename	텍스트	외부 테이블 이름.
values	텍스트	파티션의 값입니다.

열 명칭	데이터 유형	설명
location	텍스트	파티션 위치. 열 크기는 128자로 제한됩니다. 더 긴 값은 잘립니다.
input_format	텍스트	입력 형식.
output_format	텍스트	출력 형식.
serialization_lib	텍스트	직렬화 라이브러리.
serde_parameters	텍스트	SerDe parameters.
compressed	정수	파티션이 압축되어 있는지를 나타내는 값. **1**은 압축되었음을 나타내고 **0**은 압축되지 않았음을 나타냅니다.
parameters	텍스트	파티션 속성.

SVV_EXTERNAL_SCHEMAS

SVV_EXTERNAL_SCHEMAS는 외부 스키마에 대한 정보를 확인하는 데 사용됩니다. 자세한 내용은 CREATE EXTERNAL SCHEMA 단원을 참조하십시오.

SVV_EXTERNAL_SCHEMAS는 모든 사용자가 볼 수 있습니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
esoid	oid	외부 스키마 ID입니다.
eskind	smallint	외부 스키마에 대한 외부 카탈로그 형식: 1은 데이터 카탈로그, 2는 Hive 메타스토어, 3은 Aurora PostgreSQL 또는 Amazon RDS PostgreSQL에 대한 페더레이션 쿼리, 4는 로컬 Amazon Redshift 데이터베이스에 대한 스키마, 5는 원격 Amazon Redshift 데이터베이스에 대한 스키마, 6은 시스템

열 명칭	데이터 유형	설명
		테이블에 대한 스키마, 8은 원격 MySQL 데이터베이스에 대한 스키마, 9는 Amazon Kinesis 데이터 스트림에 대한 스키마, 10은 Apache Kafka 데이터 스트림용 Amazon 관리형 스트리밍을 나타냅니다.
schemaname	name	외부 스키마 이름입니다.
esowner	정수	외부 스키마 소유자의 사용자 ID입니다.
databasename	텍스트	외부 데이터베이스 이름입니다.
esoptions	텍스트	외부 스키마 옵션입니다.

예제

다음은 외부 스키마에 대한 세부 정보를 나타내는 예입니다.

```
select * from svv_external_schemas;

esoid  | eskind | schemaname | esowner | databasename | esoptions

-------+--------+------------+---------+--------------
+-------------------------------------------------------------
100133 |      1 | spectrum   |     100 | redshift     |
 {"IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
```

SVV_EXTERNAL_TABLES

외부 테이블에 대한 세부 정보를 보려면 SVV_EXTERNAL_TABLES를 사용합니다. 자세한 내용은 CREATE EXTERNAL SCHEMA 섹션을 참조하세요. 사용자가 액세스할 수 있는 연결되지 않은 데이터베이스의 모든 테이블에 대한 메타데이터를 보려면 데이터베이스 간 쿼리에도 SVV_EXTERNAL_TABLES를 사용합니다.

SVV_EXTERNAL_TABLES는 모든 사용자가 볼 수 있습니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
redshift_database_name	텍스트	로컬 Amazon Redshift 데이터베이스의 이름입니다.
schemaname	텍스트	외부 테이블의 Amazon Redshift 외부 스키마 이름.
tablename	텍스트	외부 테이블 이름.
tabletype	텍스트	테이블 유형. 일부 값은 TABLE, VIEW, MATERIALIZED VIEW 또는 정보를 나타내지 않는 빈 문자열 " "입니다.
location	텍스트	테이블의 위치
input_format	텍스트	입력 형식
output_format	텍스트	출력 형식.
serialization_lib	텍스트	직렬화 라이브러리.
serde_parameters	텍스트	SerDe parameters.
compressed	정수	테이블이 압축되어 있는지를 나타내는 값. **1**은 압축되었음을 나타내고 **0**은 압축되지 않았음을 나타냅니다.
parameters	텍스트	테이블 속성.

예제

다음 예에서는 연합 쿼리에서 사용하는 외부 스키마의 조건자가 있는 svv_external_tables에 대한 세부 정보를 보여 줍니다.

```
select schemaname, tablename from svv_external_tables where schemaname = 'apg_tpch';
schemaname  | tablename
------------+-----------
apg_tpch    | customer
apg_tpch    | lineitem
apg_tpch    | nation
apg_tpch    | orders
apg_tpch    | part
apg_tpch    | partsupp
apg_tpch    | region
apg_tpch    | supplier
(8 rows)
```

SVV_FUNCTION_PRIVILEGES

SVV_FUNCTION_PRIVILEGES를 사용하여 현재 데이터베이스의 사용자, 롤 및 그룹에 명시적으로 부여된 함수 권한을 볼 수 있습니다.

SVV_FUNCTION_PRIVILEGES는 다음 사용자에게 표시됩니다.

- 슈퍼 사용자
- ACCESS SYSTEM TABLE 권한이 있는 사용자

다른 사용자는 액세스할 수 있거나 소유한 ID만 볼 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
namespace_name	텍스트	지정된 함수가 존재하는 네임스페이스의 이름입니다.
function_name	텍스트	함수의 이름입니다.
argument_type	텍스트	함수의 입력 인수 형식을 나타내는 문자열입니다.

열 명칭	데이터 유형	설명
privilege_type	텍스트	권한의 유형입니다. 가능한 값은 EXECUTE입니다.
identity_id	정수	자격 증명의 ID입니다. 가능한 값은 사용자 ID, 역할 ID 또는 그룹 ID입니다.
identity_name	텍스트	자격 증명의 이름입니다.
identity_type	텍스트	자격 증명의 유형입니다. 가능한 값은 사용자, 역할, 그룹 또는 퍼블릭입니다.
admin_option	boolean	사용자가 다른 사용자 및 역할에 권한을 부여할 수 있는지 여부를 나타내는 값. 역할 및 그룹 자격 증명 유형에 대해서는 항상 false입니다.

샘플 쿼리

다음 예에서는 SV_FUNCTION_PRIVILEGES의 결과를 표시합니다.

```
SELECT
 namespace_name,function_name,argument_types,privilege_type,identity_name,identity_type,admin_o
 FROM svv_function_privileges
WHERE identity_name IN ('role1', 'reguser');

 namespace_name | function_name |       argument_types       | privilege_type |
 identity_name | identity_type | admin_option
----------------+---------------+----------------------------+----------------
+----------------+---------------+-------------
    public      | test_func1    | integer                    |    EXECUTE     |
 role1     |     role      |  False
    public      | test_func2    | integer, character varying |    EXECUTE     |
 reguser    |     user      |  False
```

SVV_GEOGRAPHY_COLUMNS

데이터 웨어하우스의 GEOGRAPHY 열 목록을 보려면 SVV_GEOGRAPHY_COLUMNS를 사용하세요. 이 열 목록에는 데이터 공유의 열이 포함됩니다.

SVV_GEOGRAPHY_COLUMNS는 모든 사용자가 볼 수 있습니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
f_table_catalog	varchar(128)	GEOGRAPHY 열을 포함하는 테이블이 있는 데이터베이스의 이름입니다.
f_table_schema	varchar(128)	GEOGRAPHY 열을 포함하는 테이블이 있는 스키마의 이름입니다.
f_table_name	varchar(128)	GEOGRAPHY 열을 포함하는 테이블의 이름입니다.
f_geography_column	varchar(128)	GEOGRAPHY 열의 이름입니다.
coord_dimension	정수	GEOGRAPHY 데이터의 차원 수입니다.
srid	정수	GEOGRAPHY 데이터의 공간 참조 시스템 식별자(SRID)입니다.
type	varchar(128)	공간 지리 데이터 유형 이름입니다.

샘플 쿼리

다음 예에서는 SVV_GEOGRAPHY_COLUMNS의 결과를 표시합니다.

```
SELECT * FROM svv_geography_columns;

f_table_catalog  | f_table_schema  | f_table_name  | f_geography_column  |
 coord_dimension | srid |  type
-----------------+-----------------+--------------+---------------------
+-----------------+------+-------------
```

```
dev              | public          | spatial_test  | test_geography      | 2
    | 0    | GEOGRAPHY
```

SVV_GEOMETRY_COLUMNS

데이터 웨어하우스의 GEOMETRY 열 목록을 보려면 SVV_GEOMETRY_COLUMNS를 사용하세요. 이 열 목록에는 데이터 공유의 열이 포함됩니다.

SVV_GEOMETRY_COLUMNS는 모든 사용자가 볼 수 있습니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
f_table_catalog	varchar(128)	GEOMETRY 열을 포함하는 테이블이 있는 데이터베이스의 이름입니다.
f_table_schema	varchar(128)	GEOMETRY 열을 포함하는 테이블이 있는 스키마의 이름입니다.
f_table_name	varchar(128)	GEOMETRY 열을 포함하는 테이블의 이름입니다.
f_geometry_column	varchar(128)	GEOMETRY 열의 이름입니다.
coord_dimension	정수	GEOMETRY 데이터의 차원 수입니다.
srid	정수	GEOMETRY 열의 공간 참조 시스템 식별자(SRID)입니다.
type	varchar(128)	공간 지오메트리 유형의 이름입니다.

샘플 쿼리

다음 예에서는 SVV_GEOMETRY_COLUMNS의 결과를 표시합니다.

```
SELECT * FROM svv_geometry_columns;

f_table_catalog  | f_table_schema  | f_table_name  | f_geometry_column   |
 coord_dimension | srid |  type
-----------------+-----------------+--------------+--------------------
+-----------------+------+-------------
dev              | public          | accomodations | shape               | 2
    | 0    | GEOMETRY
dev              | public          | zipcode       | wkb_geometry        | 2
    | 0    | GEOMETRY
```

SVV_IAM_PRIVILEGES

SVV_IAM_PRIVILEGES를 사용하여 사용자, 역할 및 그룹에 명시적으로 부여된 IAM 권한을 봅니다.

SVV_IAM_PRIVILEGES는 다음 사용자에게 표시됩니다.

- 슈퍼 사용자
- ACCESS SYSTEM TABLE 권한이 있는 사용자

다른 사용자는 액세스할 수 있는 항목만 볼 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
iam_arn	텍스트	네임스페이스의 이름입니다.
commandtype	텍스트	권한 유형입니다. 가능한 값은 COPY, UNLOAD, CREATE MODEL 또는 EXTERNAL FUNCTION입니다.
identity_id	정수	아이덴티티 ID입니다. 가능한 값은 사용자 ID, 역할 ID 또는 그룹 ID입니다.
identity_name	텍스트	아이덴티티 이름입니다.

열 명칭	데이터 유형	설명
identity_type	텍스트	자격 증명 유형입니다. 가능한 값은 사용자, 역할, 그룹 또는 퍼블릭입니다.

샘플 쿼리

다음 예제에서는 SVV_IAM_PRIVILEGES를 보여줍니다.

```
SELECT * from SVV_IAM_PRIVILEGES ORDER BY IDENTITY_ID;
       iam_arn        | command_type | identity_id | identity_name | identity_type
----------------------+--------------+-------------+---------------+---------------
 default-aws-iam-role | COPY         |           0 | public        | public
 default-aws-iam-role | UNLOAD       |           0 | public        | public
 default-aws-iam-role | CREATE MODEL |           0 | public        | public
 default-aws-iam-role | EXFUNC       |           0 | public        | public
 default-aws-iam-role | COPY         |         106 | u1            | user
 default-aws-iam-role | UNLOAD       |         106 | u1            | user
 default-aws-iam-role | CREATE MODEL |         106 | u1            | user
 default-aws-iam-role | EXFUNC       |         106 | u1            | user
 default-aws-iam-role | COPY         |      118413 | r1            | role
 default-aws-iam-role | UNLOAD       |      118413 | r1            | role
 default-aws-iam-role | CREATE MODEL |      118413 | r1            | role
 default-aws-iam-role | EXFUNC       |      118413 | r1            | role
(12 rows)
```

SVV_IDENTITY_PROVIDERS

SVV_IDENTITY_PROVIDERS 보기는 자격 증명 공급자의 이름과 추가 속성을 반환합니다. 자격 증명 공급자를 생성하는 방법에 대한 자세한 내용은 CREATE IDENTITY PROVIDER 섹션을 참조하세요.

SVV_IDENTITY_PROVIDERS는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
uid	정수	등록된 자격 증명 공급자의 고유 ID입니다.
name	텍스트	자격 증명 공급자 이름입니다.
type	텍스트	자격 증명 공급자 유형입니다.
instanceid	텍스트	동일한 유형의 인스턴스 간의 고유 차별화 요소입니다.
namespc	텍스트	자격 증명 네임스페이스 접두사입니다.
params	텍스트	자격 증명 공급자에 대한 파라미터가 있는 JSON 객체입니다.
enabled	bool	자격 증명 공급자가 활성화되어 있는지 여부를 나타냅니다.

샘플 쿼리

자격 증면 공급자 속성을 보려면 자격 증명 공급자를 만든 후 다음과 같은 쿼리를 실행합니다.

```
SELECT name, type, instanceid, namespc, params, enabled
FROM svv_identity_providers
ORDER BY 1;
```

샘플 출력에는 파라미터 설명이 포함되어 있습니다.

```
      name       | type  |             instanceid             | namespc |

                                      params

                    | enabled
```

```
------------------+-------+--------------------------------------+---------
+-----------------------------------------------------------------------------------------------------------------
+---------
 rs5517_azure_idp | azure | e40d4bb2-7670-44ae-bfb8-5db013221d73 | abc      |
 {"issuer":"https://login.microsoftonline.com/e40d4bb2-7670-44ae-bfb8-5db013221d73/
v2.0", "client_id":"871c010f-5e61-4fb1-83ac-98610a7e9110", "client_secret":,
 "audience":["https://analysis.windows.net/powerbi/connector/AmazonRedshift", "https://
analysis.windows.net/powerbi/connector/AWSRDS"]} | t
(1 row)
```

SVV_INTEGRATION

SVV_INTEGRATION은 통합 구성에 대한 세부 정보를 표시합니다.

SVV_INTEGRATION은 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

제로 ETL 통합에 대한 자세한 내용은 제로 ETL 통합 작업을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
integration_id	character (128)	통합과 관련된 식별자입니다.
target_database	character (128)	통합 데이터를 수신하는 Amazon Redshift의 데이터베이스입니다.
source	character (128)	통합을 위한 소스 데이터입니다. 가능한 유형에는 `MySQL`, `PostgreSQL` 및 `S3_EVENT_NOTIFICATIONS` 이 포함됩니다.
state	character (128)	통합 상태입니다. 가능한 값은 `PendingDbConnectState` , `SchemaDiscoveryState` , `CdcRefreshState` , `ErrorState` 입니다.
current_lag	bigint	통합의 소스와 대상 간의 현재 지연 시간(밀리초)입니다.

열 명칭	데이터 유형	설명
last_replicated_checkpoint	character (128)	마지막으로 복제된 체크포인트입니다.
total_tables_replicated	정수	현재 복제된 상태의 총 테이블 수입니다.
total_tables_failed	정수	현재 실패 상태의 총 테이블 수입니다.
creation_time	타임스탬프	통합이 생성된 시점의 시각(UTC)입니다. 통합에서 대상 데이터베이스가 생성되는 시각으로 정의됩니다.
refresh_interval	정수	제로 ETL 소스에서 대상 데이터베이스로 데이터를 새로고침하는 대략적인 시간 간격입니다.
source_database	character (128)	소스 데이터베이스의 이름입니다.

샘플 쿼리

다음 SQL 명령은 현재 정의된 통합을 표시합니다.

```
select * from svv_integration;

           integration_id              | target_database | source |      state
   | current_lag |      last_replicated_checkpoint     | total_tables_replicated |
 total_tables_failed |       creation_time       |  refresh_interval  | source_database
-------------------------------------+-----------------+--------+-----------------
+-------------+--------------------------------------+-------------------------
+---------------------+--------------------------+--------------------
+----------------
  99108e72-1cfd-414f-8cc0-0216acefac77 |     perfdb      |  MySQL | CdcRefreshState |
  56606106  | {"txn_seq":9834,"txn_id":126597515} |            152          |
 0         | 2023-09-19 21:05:27.520299|      720           + mysourceetl
```

SVV_INTEGRATION_TABLE_STATE

SVV_INTEGRATION_TABLE_STATE는 테이블 수준 통합 정보에 대한 세부 정보를 표시합니다.

SVV_INTEGRATION_TABLE_STATE는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

자세한 내용은 제로 ETL 통합 작업을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
integration_id	character(128)	통합과 관련된 식별자입니다.
target_database	character(128)	Amazon Redshift 데이터베이스의 이름입니다.
schema_name	character(128)	Amazon Redshift 스키마의 이름입니다.
table_name	character(128)	테이블의 이름
table_state	character(128)	테이블 상태입니다. 가능한 값은 `Synced`, `Failed`, `Deleted`, `ResyncRequired` 및 `ResyncInitiated` 입니다.
table_last_replicated_checkpoint	character(128)	현재 동기화된 로그 좌표입니다.
reason	character(256)	마지막 상태 전환의 이유입니다. 일반적인 이유는 테이블에 지원되지 않는 데이터 형식이 있거나 테이블에 프라이머리 키가 없는 경우입니다. 일반적인 문제를 해결하는 방법에 대해 자세히 알아보려면 Amazon Redshift에서 제로 ETL 통합 문제 해결을 참조하세요.
last_updated_timestamp	시간대 미포함 TIMESTAMP	테이블이 마지막으로 업데이트된 시간(UTC)입니다.
table_rows	bigint	테이블에 포함된 행의 총 수입니다.
table_size	bigint	테이블의 크기입니다(MB).

샘플 쿼리

다음 SQL 명령은 통합 로그의 열을 표시합니다.

```
select * from svv_integration_table_state;

          integration_id              | target_database | schema_name |     table_name
   | Table_state  |table_last_replicated_checkpoint | reason | last_updated_timestamp
   |table_rows  | table_size
-------------------------------------+-----------------+-------------
+------------------+-------------+---------------------------------+--------
+---------------------------+-----------+----------
 4798e675-8f9f-4686-b05f-92c538e19629 |  sample_test2   |    sample   |
 SampleTestChannel |    Synced    |   {"txn_seq":3,"txn_id":3122}   |        |
 2023-05-12 12:40:30.656625 | 2          |   16
```

SVV_INTERLEAVED_COLUMNS

SVV_INTERLEAVED_COLUMNS 뷰를 사용하면 인터리브 정렬 키를 사용하는 테이블을 VACUUM REINDEX를 사용하여 다시 인덱싱해야 하는지 결정하는 데 도움이 됩니다. VACUUM 실행 주기와 VACUUM REINDEX 실행 시점에 대한 자세한 내용은 Vacuum 시간 최소화 섹션을 참조하세요.

SVV_INTERLEAVED_COLUMNS는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
tbl	정수	테이블 ID.
col	정수	0부터 시작되는 열 인덱스.
interleaved_skew	numeric 9,2)	테이블의 인터리브 정렬 키 열에 존재하는 스큐의 양을 나타내는 비율. 값 1.00은 스큐가 없음을 나타내고, 이보다 큰 값은 스큐가 더 많음을 나타냅니다. 스큐가 큰 테이블은 VACUUM REINDEX 명령을 사용하여 다시 인덱싱해야 합니다.

열 명칭	데이터 유형	설명
last_reindex	타임스탬프	지정된 테이블에 대해 마지막 VACUUM REINDEX가 실행된 시간. 테이블의 인덱스를 재지정한 적이 없는 경우 또는 기본 시스템 로그 테이블 STL_VACUUM이 마지막 인덱스 재지정 이후 교체된 경우 이 값은 NULL입니다.

샘플 쿼리

다시 인덱싱해야 할 테이블을 식별하려면 다음 쿼리를 실행합니다.

```
select tbl as tbl_id, stv_tbl_perm.name as table_name,
col, interleaved_skew, last_reindex
from svv_interleaved_columns, stv_tbl_perm
where svv_interleaved_columns.tbl = stv_tbl_perm.id
and interleaved_skew is not null;

 tbl_id | table_name | col | interleaved_skew | last_reindex
--------+------------+-----+------------------+---------------------
 100068 | lineorder  |   0 |             3.65 | 2015-04-22 22:05:45
 100068 | lineorder  |   1 |             2.65 | 2015-04-22 22:05:45
 100072 | customer   |   0 |             1.65 | 2015-04-22 22:05:45
 100072 | lineorder  |   1 |             1.00 | 2015-04-22 22:05:45
(4 rows)
```

SVV_LANGUAGE_PRIVILEGES

현재 데이터베이스의 사용자, 역할 및 그룹에 명시적으로 부여된 언어 권한을 보려면 SVV_LANGUAGE_PRIVILEGES를 사용합니다.

SVV_LANGUAGE_PRIVILEGES는 다음 사용자에게 표시됩니다.

- 슈퍼 사용자
- ACCESS SYSTEM TABLE 권한이 있는 사용자

다른 사용자는 액세스할 수 있거나 소유한 ID만 볼 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
language_name	텍스트	언어의 이름입니다.
privilege_type	텍스트	권한의 유형입니다. 가능한 값은 USAGE입니다.
identity_id	정수	자격 증명의 ID입니다. 가능한 값은 사용자 ID, 역할 ID 또는 그룹 ID입니다.
identity_name	텍스트	자격 증명의 이름입니다.
identity_type	텍스트	자격 증명의 유형입니다. 가능한 값은 사용자, 역할, 그룹 또는 퍼블릭입니다.
admin_option	boolean	사용자가 다른 사용자 및 역할에 권한을 부여할 수 있는지 여부를 나타내는 값. 역할 및 그룹 자격 증명 유형에 대해서는 항상 false입니다.

샘플 쿼리

다음 예에서는 SVV_LANGUAGE_PRIVILEGES의 결과를 표시합니다.

```
SELECT language_name,privilege_type,identity_name,identity_type,admin_option FROM
 svv_language_privileges
WHERE identity_name IN ('role1', 'reguser');

 language_name | privilege_type | identity_name | identity_type | admin_option
---------------+----------------+---------------+---------------+--------------
   exfunc      |     USAGE      |    reguser    |     user      |    False
   exfunc      |     USAGE      |     role1     |     role      |    False
   plpythonu   |     USAGE      |    reguser    |     user      |    False
```

SVV_MASKING_POLICY

SVV_MASKING_POLICY를 사용하여 클러스터에 생성된 모든 마스킹 정책을 볼 수 있습니다.

sys:secadmin 역할이 부여된 슈퍼 사용자와 사용자만 SVV_MASKING_POLICY를 볼 수 있습니다. 일반 사용자에게는 0개의 행이 표시됩니다.

테이블 열

열 명칭	데이터 유형	설명
policy_database	텍스트	마스킹 정책이 생성된 데이터베이스의 이름입니다.
policy_name	텍스트	마스킹 정책의 이름입니다.
input_columns	텍스트	CREATE POLICY 문의 WITH 절에 제공된 속성입니다.
policy_expression	텍스트	정책에 사용된 마스킹 표현식입니다.
policy_modified_by	텍스트	정책을 마지막으로 수정한 사용자의 이름입니다.
policy_modified_time	타임스탬프	정책이 생성되거나 마지막으로 수정된 시간의 타임스탬프입니다.

SVV_ML_MODEL_INFO

기계 학습 모델의 현재 상태에 대한 상태 정보입니다.

STV_ML_MODEL_INFO는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
database_name	char(128)	모델의 데이터베이스입니다.

열 명칭	데이터 유형	설명
schema_name	char(128)	모델의 스키마.
user_name	char(128)	모델의 소유자입니다.
model_name	char(128)	모델의 이름입니다.
life_cycle	char(20)	모델의 수명 주기 상태입니다.
is_refreshable	정수	훈련 쿼리의 원래 테이블과 열이 여전히 존재하고 사용자에게 여전히 권한이 있는 경우 모델을 새로 고칠 수 있는지 여부의 모델 상태입니다. 가능한 값은 1(새로 고칠 수 있음)과 0(새로 고칠 수 없음)입니다.
model_state	char(128)	모델의 현재 상태입니다.

샘플 쿼리

다음 쿼리는 기계 학습 모델의 현재 상태를 표시합니다.

```
SELECT schema_name, model_name, model_state
FROM svv_ml_model_info;

 schema_name |        model_name             |             model_state
-------------+-------------------------------+-----------------------------------
 public      | customer_churn_auto_model     | Train Model On SageMaker In Progress
 public      | customer_churn_xgboost_model | Model is Ready
(2 row)
```

SVV_ML_MODEL_PRIVILEGES

클러스터의 사용자, 역할 및 그룹에 명시적으로 부여된 기계 학습 모델 권한을 보려면 SVV_ML_MODEL_PRIVILEGES를 사용합니다.

SVV_ML_MODEL_PRIVILEGES는 다음 사용자에게 표시됩니다.

- 슈퍼 사용자
- ACCESS SYSTEM TABLE 권한이 있는 사용자

다른 사용자는 액세스할 수 있거나 소유한 ID만 볼 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
namespace_name	텍스트	지정된 기계 학습 모델이 존재하는 네임스페이스의 이름입니다.
model_name	텍스트	기계 학습 모델의 이름입니다.
model_version	정수	모델의 버전 번호입니다.
privilege_type	텍스트	권한의 유형입니다. 가능한 값은 EXECUTE입니다.
identity_id	정수	자격 증명의 ID입니다. 가능한 값은 사용자 ID, 역할 ID 또는 그룹 ID입니다.
identity_name	텍스트	자격 증명의 이름입니다.
identity_type	텍스트	자격 증명의 유형입니다. 가능한 값은 사용자, 역할, 그룹 또는 퍼블릭입니다.
admin_option	boolean	사용자가 다른 사용자 및 역할에 권한을 부여할 수 있는지 여부를 나타내는 값. 역할 및 그룹 자격 증명 유형에 대해서는 항상 false입니다.

샘플 쿼리

다음 예에서는 SVV_ML_MODEL_PRIVILEGES의 결과를 표시합니다.

```
SELECT
 namespace_name,model_name,model_version,privilege_type,identity_name,identity_type,admin_optio
 FROM svv_ml_model_privileges
WHERE model_name = 'test_model';
```

```
 namespace_name | model_name | model_version | privilege_type |  identity_name |
 identity_type | admin_option
----------------+------------+---------------+----------------+----------------
+---------------+-------------
      public    | test_model |       1       |    EXECUTE     |     reguser    |
 user       |    False
      public    | test_model |       1       |    EXECUTE     |     role1      |
 role       |    False
```

SVV_MV_DEPENDENCY

SVV_MV_DEPENDENCY 테이블은 Amazon Redshift 내의 다른 구체화된 뷰에 대한 구체화된 뷰의 종속성을 보여줍니다.

구체화된 뷰에 대한 자세한 내용은 Amazon Redshift의 구체화된 뷰 섹션을 참조하세요.

SVV_MV_DEPENDENCY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
database_name	char(128)	지정한 구체화된 뷰를 포함하는 데이터베이스입니다.
schema_name	char(128)	구체화된 보기의 스키마입니다.
name	char(128)	구체화된 보기의 이름입니다.
dependent_database_name	char(128)	이 구체화된 뷰가 종속된 구체화된 뷰 데이터베이스입니다.
dependent_schema_name	char(128)	이 구체화된 뷰가 종속된 구체화된 뷰 스키마입니다.
dependent_name	char(128)	이 구체화된 뷰가 종속된 구체화된 뷰의 이름입니다.

샘플 쿼리

다음 쿼리는 구체화된 뷰 mv_over_foo가 정의에서 구체화된 뷰 mv_foo를 종속성으로 사용함을 나타내는 출력 행을 반환합니다.

```
CREATE SCHEMA test_ivm_setup;
CREATE TABLE test_ivm_setup.foo(a INT);
CREATE MATERIALIZED VIEW test_ivm_setup.mv_foo AS SELECT * FROM test_ivm_setup.foo;
CREATE MATERIALIZED VIEW test_ivm_setup.mv_over_foo AS SELECT * FROM
 test_ivm_setup.mv_foo;

SELECT * FROM svv_mv_dependency;

 database_name | schema_name          | name        | dependent_database_name |
 dependent_schema_name     | dependent_name
---------------+----------------------+-------------+-------------------------
+--------------------------+----------
 dev           | test_ivm_setup       | mv_over_foo |                     dev |
 test_ivm_setup            | mv_foo
```

SVV_MV_INFO

SVV_MV_INFO 테이블에는 모든 구체화된 뷰의 행, 데이터가 기간이 지났는지 여부 및 상태 정보가 포함되어 있습니다.

구체화된 뷰에 대한 자세한 내용은 Amazon Redshift의 구체화된 뷰 섹션을 참조하세요.

SVV_MV_INFO는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
database_name	char(128)	구체화된 보기를 포함하는 데이터베이스입니다.
schema_name	char(128)	데이터베이스의 스키마입니다.
user_name	char(128)	구체화된 보기를 소유한 사용자입니다.

열 명칭	데이터 유형	설명
name	char(128)	구체화된 보기 이름입니다.
is_stale	char(1)	t는 구체화된 보기가 오래 되었음을 나타냅니다. 오래된 구체화된 보기는 기본 테이블이 업데이트 되었지만 구체화된 보기가 새로 고쳐지지 않은 것입니다. 마지막 재시작 이후 새로 고침을 실행하지 않은 경우 이 정보가 정확하지 않을 수 있습니다.
state	정수	구체화된 보기의 상태는 다음과 같습니다. • 0 - 새로 고칠 때 구체화된 뷰가 완전히 다시 계산됩니다. • 1 - 구체화된 뷰가 증분적입니다. • 101 - 삭제된 열로 인해 구체화된 뷰를 새로 고칠 수 없습니다. 이 제약 조건은 구체화된 보기에서 열이 사용되지 않는 경우에도 적용됩니다. • 102 - 변경된 열 형식으로 인해 구체화된 뷰를 새로 고칠 수 없습니다. 이 제약 조건은 구체화된 보기에서 열이 사용되지 않는 경우에도 적용됩니다. • 103 - 이름이 변경된 테이블로 인해 구체화된 뷰를 새로 고칠 수 없습니다. • 104 - 이름이 변경된 열로 인해 구체화된 뷰를 새로 고칠 수 없습니다. 이 제약 조건은 구체화된 보기에서 열이 사용되지 않는 경우에도 적용됩니다. • 105 - 이름이 변경된 스키마로 인해 구체화된 뷰를 새로 고칠 수 없습니다.
autorewrite	char(1)	t는 구체화된 뷰가 쿼리를 자동으로 다시 작성할 수 있음을 나타냅니다.
autorefresh	char(1)	t는 구체화된 뷰가 자동으로 새로 고쳐질 수 있음을 나타냅니다.

샘플 쿼리

모든 구체화된 보기의 상태를 보려면 다음 쿼리를 실행합니다.

```
select * from svv_mv_info;
```

위 쿼리는 다음과 같은 샘플 출력을 반환합니다.

```
database_name |       schema_name       | user_name |   name   |  is_stale | state |
 autorefresh | autorewrite
--------------+-------------------------+-----------+----------+-----------+-------
+-------------+----------------
 dev          | test_ivm_setup          | catch-22  | mv       |   f       |     1 |
       1 |            0
 dev          | test_ivm_setup          | lotr      | old_mv   |   t       |     1 |
       0 |            1
```

SVV_QUERY_INFLIGHT

데이터베이스에서 현재 실행 중인 쿼리를 확인하려면 SVV_QUERY_INFLIGHT 뷰를 사용합니다. 이 뷰는 STV_INFLIGHT을 STL_QUERYTEXT에 조인합니다. SVV_QUERY_INFLIGHT는 리더 노드 전용 쿼리를 표시하지 않습니다. 자세한 내용은 리더 노드 전용 함수 단원을 참조하십시오.

SVV_QUERY_INFLIGHT는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> 이 보기는 프로비저닝된 클러스터를 쿼리할 때만 사용할 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.

열 명칭	데이터 유형	설명
slice	정수	쿼리가 실행 중인 조각.
쿼리	정수	쿼리 ID. 다양한 다른 시스템 테이블 및 보기를 조인하는 데 사용할 수 있습니다.
pid	정수	프로세스 ID. 한 세션의 모든 쿼리는 동일 프로세스에서 실행되므로 동일 세션에서 일련의 쿼리를 실행하는 경우, 이 값은 항상 같은 값을 유지합니다. 이 열을 사용하여 STL_ERROR 테이블에 조인할 수 있습니다.
starttime	타임스탬프	쿼리가 시작된 시간입니다.
suspended	정수	쿼리가 일시 중지되었는지 여부. **0** = false; **1** = true.
텍스트	character(200)	200자씩 증가하는 쿼리 텍스트.
SEQUENCE	정수	쿼리 문 세그먼트의 시퀀스 번호.

샘플 쿼리

아래의 샘플 출력은 SVV_QUERY_INFLIGHT 쿼리 자체와 테이블에서 3개의 행으로 나뉜 쿼리 428이라는 현재 실행 중인 2개의 쿼리를 보여 줍니다. (starttime 및 statement 열은 이 샘플 출력에서 잘려 있습니다.)

```
select slice, query, pid, starttime, suspended, trim(text) as statement, sequence
from svv_query_inflight
order by query, sequence;

slice|query| pid  |      starttime       |suspended| statement | sequence
-----+-----+------+----------------------+---------+-----------+---------
1012 | 428 | 1658 | 2012-04-10 13:53:... |       0 | select ...|    0
1012 | 428 | 1658 | 2012-04-10 13:53:... |       0 | enueid ...|    1
1012 | 428 | 1658 | 2012-04-10 13:53:... |       0 | atname,...|    2
1012 | 429 | 1608 | 2012-04-10 13:53:... |       0 | select ...|    0
(4 rows)
```

SVV_QUERY_STATE

SVV_QUERY_STATE를 사용하여 현재 실행 중인 쿼리의 런타임에 관한 정보를 확인합니다.

SVV_QUERY_STATE 뷰에는 STV_EXEC_STATE 테이블의 데이터 하위 집합이 포함됩니다.

SVV_QUERY_STATE는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_DETAIL에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

> **Note**
>
> 이 보기는 프로비저닝된 클러스터를 쿼리할 때만 사용할 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 다양한 다른 시스템 테이블 및 보기를 조인하는 데 사용할 수 있습니다.
seg	정수	실행 중인 쿼리 세그먼트의 수. 하나의 쿼리는 여러 세그먼트로 구성되며, 각각의 세그먼트는 하나 이상의 단계로 구성됩니다. 쿼리 세그먼트는 병렬로 실행될 수 있습니다. 각 세그먼트는 단일 프로세스에서 실행됩니다.
step	정수	실행 중인 쿼리 단계의 수. 단계는 최소 쿼리 런타임 단위입니다. 각 단계는 테이블 스캔, 결과 반환 또는 데이터 정렬 같은 개별적 작업 단위를 나타냅니다.
maxtime	interval	이 단계가 실행되기 위한 최대 시간(마이크로초)입니다.

열 명칭	데이터 유형	설명
avgtime	interval	이 단계가 실행되기 위한 평균 시간(마이크로초)입니다.
rows	bigint	실행 중인 단계에 의해 만들어진 행의 수.
bytes	bigint	실행 중인 단계에 의해 만들어진 바이트의 수.
cpu	bigint	내부용.
memory	bigint	내부용.
rate_row	double precision	쿼리가 시작된 이후의 초당 행 속도로서 행을 합산해 쿼리가 시작된 때부터 현재 시간까지의 초로 나누어 계산합니다.
rate_byte	double precision	쿼리가 시작된 이후의 초당 바이트 속도로서 바이트를 합산해 쿼리가 시작된 때부터 현재 시간까지의 초로 나누어 계산합니다.
레이블	character(25)	쿼리 레이블: `scan` 또는 `sort`와 같은 단계의 이름.
is_diskbased	character(1)	쿼리의 이 단계가 디스크 기반 작업으로 실행 중인지 여부: true(**t**) 또는 false(**f**). 해시, 정렬, 집계 단계 같은 특정 단계만 디스크로 갈 수 있습니다. 많은 단계 형식은 항상 메모리에서 수행됩니다.
workmem	bigint	쿼리 단계에 할당된 작업 메모리의 양(바이트)입니다.
num_parts	정수	해시 단계 도중 해시 테이블이 분할되는 파티션의 수. 이 열의 양수는 해시 단계가 디스크 기반 작업으로 실행되었음을 뜻하지 않습니다. 해시 단계가 디스크 기반인지 확인하려면 IS_DISKBASED 열의 값을 확인합니다.
is_rrscan	character(1)	true(**t**)인 경우, 단계에서 범위 제한 스캔이 사용되었음을 나타냅니다. 기본값은 false(**f**)입니다.
is_delayed_scan	character(1)	true(**t**)인 경우, 단계에서 지연된 스캔이 사용되었음을 나타냅니다. 기본값은 false(**f**)입니다.

샘플 쿼리

단계별 쿼리 처리 시간 확인

다음 쿼리는 쿼리 ID 279인 쿼리의 각 단계가 실행되는 데 걸린 시간과 Amazon Redshift가 처리한 데이터 행의 수를 보여줍니다.

```
select query, seg, step, maxtime, avgtime, rows, label
from svv_query_state
where query = 279
order by query, seg, step;
```

이 쿼리는 다음 샘플 출력에서 보듯 쿼리 279에 대한 처리 정보를 검색합니다.

```
query |   seg   | step | maxtime | avgtime |  rows   | label
------+---------+------+---------+---------+---------+-------------------
  279 |       3 |    0 | 1658054 | 1645711 | 1405360 | scan
  279 |       3 |    1 | 1658072 | 1645809 |       0 | project
  279 |       3 |    2 | 1658074 | 1645812 | 1405434 | insert
  279 |       3 |    3 | 1658080 | 1645816 | 1405437 | distribute
  279 |       4 |    0 | 1677443 | 1666189 | 1268431 | scan
  279 |       4 |    1 | 1677446 | 1666192 | 1268434 | insert
  279 |       4 |    2 | 1677451 | 1666195 |       0 | aggr
(7 rows)
```

디스크에서 현재 실행 중인 활성 쿼리가 있는지 확인

다음 쿼리는 디스크에서 현재 실행 중인 활성 쿼리가 있는지 보여 줍니다.

```
select query, label, is_diskbased from svv_query_state
where is_diskbased = 't';
```

이 샘플 출력은 디스크에서 현재 실행 중인 모든 활성 쿼리를 보여 줍니다.

```
 query | label        | is_diskbased
-------+--------------+--------------
1025   | hash tbl=142 |      t
(1 row)
```

SVV_REDSHIFT_COLUMNS

사용자가 액세스할 수 있는 모든 열의 목록을 보려면 SVV_REDSHIFT_COLUMNS를 사용합니다. 이 열 집합에는 클러스터의 열과 원격 클러스터에서 제공하는 datashare의 열이 포함됩니다.

SVV_REDSHIFT_COLUMNS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
database_name	varchar(128)	열을 포함하는 테이블이 있는 데이터베이스의 이름입니다.
schema_name	varchar(128)	테이블의 스키마 이름입니다.
table_name	varchar(128)	테이블의 이름
column_name	varchar(128)	열의 이름입니다.
ordinal_position	정수	테이블의 열 위치
data_type	varchar(32)	열의 데이터 형식.
column_default	varchar(4000)	열의 기본값
is_nullable	varchar(3)	열의 Null 허용 여부를 정의하는 값입니다. 가능한 값은 yes, no 및 " "(정보를 나타내지 않는 빈 문자열)입니다.
인코딩	varchar(128)	열의 인코딩 형식입니다.
distkey	boolean	이 열이 테이블의 배포 키이면 true이고 그렇지 않으면 false인 값입니다.
sortkey	정수	정렬 키에서 열의 순서를 지정하는 값입니다. 테이블이 복합 정렬 키를 사용하는 경우에는 정렬 키에 포함된 모든 열이 양의 값으로 정렬

열 명칭	데이터 유형	설명
		키에서 자신의 위치를 나타냅니다. 테이블이 인터리브 정렬 키를 사용하는 경우 정렬 키의 일부인 각 열에는 교대로 양수 또는 음수 값이 있습니다. 여기서 절대값은 정렬 키에서 열의 위치를 나타냅니다. sortkey가 0이면 정렬 키에 포함되지 않는 열입니다.
column_acl	varchar(128)	열에 대해 지정된 사용자 또는 사용자 그룹에 대한 권한을 정의하는 문자열입니다.
remarks	varchar(256)	설명.

샘플 쿼리

다음 예에서는 SVV_REDSHIFT_COLUMNS의 출력을 반환합니다.

```
SELECT *
FROM svv_redshift_columns
WHERE database_name = 'tickit_db'
    AND TABLE_NAME = 'tickit_sales_redshift'
ORDER BY COLUMN_NAME,
    TABLE_NAME,
    database_name
LIMIT 5;

database_name | schema_name |       table_name      | column_name | ordinal_position |
 data_type | column_default | is_nullable | encoding | distkey | sortkey | column_acl
 | remarks
--------------+-------------+-----------------------+-------------+------------------
+-----------+----------------+-------------+----------+---------+---------
+-------------+--------
```

```
  tickit_db  |   public    | tickit_sales_redshift |   buyerid   |        4         |
integer  |               |      NO     |   az64   |  False  |    0    |              |
  tickit_db  |   public    | tickit_sales_redshift |  commission |        9         |
numeric  |      (8,2)    |     YES     |   az64   |  False  |    0    |              |
  tickit_db  |   public    | tickit_sales_redshift |    dateid   |        6         |
smallint |               |      NO     |   none   |  False  |    1    |              |
  tickit_db  |   public    | tickit_sales_redshift |   eventid   |        5         |
integer  |               |      NO     |   az64   |  False  |    0    |       |
  tickit_db  |   public    | tickit_sales_redshift |   listid    |        2         |
integer  |               |      NO     |   az64   |  True   |    0    |              |
```

SVV_REDSHIFT_DATABASES

SVV_REDSHIFT_DATABASES를 사용하여 사용자가 액세스할 수 있는 모든 데이터베이스 목록을 봅니다. 여기에는 클러스터의 데이터베이스와 원격 클러스터에서 제공하는 datashare에서 생성된 데이터베이스가 포함됩니다.

SVV_REDSHIFT_DATABASES는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
database_name	varchar(128)	데이터베이스의 이름입니다.
database_owner	정수	데이터베이스 소유자 사용자 ID입니다.
database_type	varchar(32)	데이터베이스의 형식입니다. 가능한 형식은 로컬 또는 공유 데이터베이스입니다.
database_acl	varchar(128)	이 정보는 내부 전용입니다.
database_options	varchar(128)	데이터베이스의 속성입니다.
database_isolation_level	varchar(128)	데이터베이스의 격리 수준입니다. 가능한 값은 Snapshot

열 명칭	데이터 유형	설명
		Isolation 및 Serializable 입니다.

샘플 쿼리

다음 예에서는 SVV_REDSHIFT_DATABASES의 출력을 반환합니다.

```
select database_name, database_owner, database_type, database_options,
 database_isolation_level
from  svv_redshift_databases;


database_name | database_owner | database_type | database_options |
 database_isolation_level
--------------+----------------+---------------+------------------+------------------
   dev        |  1             | local         | NULL             | Serializable
```

SVV_REDSHIFT_FUNCTIONS

사용자가 액세스할 수 있는 모든 함수의 목록을 보려면 SVV_REDSHIFT_FUNCTIONS를 사용합니다. 이 함수 집합에는 클러스터의 함수와 원격 클러스터에서 제공하는 datashare의 함수가 포함됩니다.

SVV_REDSHIFT_FUNCTIONS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
database_name	varchar(128)	이러한 함수가 있는 클러스터가 존재하는 데이터베이스의 이름입니다.
schema_name	varchar(128)	지정된 함수를 지정하는 스키마의 이름입니다.

열 명칭	데이터 유형	설명
function_name	varchar(128)	지정된 함수의 이름입니다.
function_type	varchar(128)	함수의 형식입니다. 가능한 값은 regular function, aggregate function 및 stored procedure입니다.
argument_type	varchar(512)	함수의 입력 인수 형식을 나타내는 문자열입니다.
result_type	varchar(128)	함수의 반환 값의 데이터 형식입니다.

샘플 쿼리

다음 예에서는 SVV_REDSHIFT_FUNCTIONS의 출력을 반환합니다.

```
SELECT *
FROM svv_redshift_functions
WHERE database_name = 'tickit_db'
    AND SCHEMA_NAME = 'public'
ORDER BY function_name
LIMIT 5;

database_name | schema_name |      function_name    |  function_type   |
 argument_type  | result_type
--------------+-------------+-----------------------+------------------
+------------------+-------------
   tickit_db  |    public   |     shared_function   | REGULAR FUNCTION | integer,
 integer |   integer
```

SVV_REDSHIFT_SCHEMA_QUOTA

데이터베이스에서 각 스키마에 대한 할당량 및 현재 디스크 사용량을 표시합니다.

SVV_REDSHIFT_SCHEMA_QUOTA는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 뷰는 프로비저닝된 클러스터 또는 Redshift 서버리스 작업 그룹을 쿼리할 때 사용할 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
database_name	character(128)	스키마를 포함하는 데이터베이스입니다.
schema_name	character(128)	스키마의 이름입니다.
schema_owner	정수	스키마 소유자의 내부 사용자 ID입니다.
할당량	정수	스키마에서 사용할 수 있는 디스크 공간(MB)입니다.
disk_usage	정수	스키마에서 현재 사용 중인 디스크 공간(MB)입니다.

샘플 쿼리

다음 예에서는 `sales_schema`라는 스키마에 대한 할당량 및 현재 디스크 사용량을 표시합니다.

```
SELECT TRIM(SCHEMA_NAME) "schema_name", QUOTA, disk_usage FROM
 svv_redshift_schema_quota
WHERE SCHEMA_NAME = 'sales_schema';


schema_name   | quota | disk_usage
--------------+-------+------------
sales_schema  | 2048  | 30
```

SVV_REDSHIFT_SCHEMAS

사용자가 액세스할 수 있는 모든 스키마의 목록을 보려면 SVV_REDSHIFT_SCHEMAS를 사용합니다. 이 스키마 집합에는 클러스터의 스키마와 원격 클러스터에서 제공하는 datashare의 스키마가 포함됩니다.

SVV_REDSHIFT_SCHEMAS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
database_name	varchar(128)	지정된 스키마가 존재하는 데이터베이스의 이름입니다.
schema_name	varchar(128)	네임스페이스 또는 스키마 이름입니다.
schema_owner	정수	스키마 소유자의 내부 사용자 ID입니다.
schema_type	varchar(16)	스키마의 형식입니다. 가능한 값은 shared 및 local 스키마입니다.
schema_acl	varchar(128)	스키마에 대해 지정된 사용자 또는 사용자 그룹에 대한 권한을 정의하는 문자열입니다.
schema_option	varchar(128)	스키마의 옵션입니다.

샘플 쿼리

다음 예에서는 SVV_REDSHIFT_SCHEMAS의 출력을 반환합니다.

```
SELECT *
FROM svv_redshift_schemas
WHERE database_name = 'tickit_db'
ORDER BY database_name,
    SCHEMA_NAME;

database_name |    schema_name     | schema_owner | schema_type | schema_acl |
 schema_option
```

```
-------------+--------------------+-------------+------------+-----------
+---------------
   tickit_db  |       public       |       1     |    shared  |           |
```

SVV_REDSHIFT_TABLES

사용자가 액세스할 수 있는 모든 테이블의 목록을 보려면 SVV_REDSHIFT_TABLES를 사용합니다. 이 테이블 집합에는 클러스터의 테이블과 원격 클러스터에서 제공하는 datashare의 테이블이 포함됩니다.

SVV_REDSHIFT_TABLES는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
database_name	varchar(128)	지정된 테이블이 존재하는 데이터베이스의 이름입니다.
schema_name	varchar(128)	테이블의 스키마 이름
table_name	varchar(128)	테이블의 이름
table_type	varchar(128)	테이블 유형. 가능한 값은 view와 table입니다.
table_acl	varchar(128)	테이블에 대해 지정된 사용자 또는 사용자 그룹에 대한 권한을 정의하는 문자열입니다.
remarks	varchar(128)	설명.
table_owner	varchar(128)	테이블의 소유자입니다.

샘플 쿼리

다음 예에서는 SVV_REDSHIFT_TABLES의 출력을 반환합니다.

```
SELECT *
FROM svv_redshift_tables
WHERE database_name = 'tickit_db' AND TABLE_NAME LIKE 'tickit_%'
ORDER BY database_name,
TABLE_NAME;

database_name | schema_name |         table_name       | table_type | table_acl |
 remarks | table_owner
--------------+-------------+--------------------------+------------+-----------
+---------+-----------
   tickit_db  |    public   | tickit_category_redshift |    TABLE   |           |
   +
   tickit_db  |    public   |   tickit_date_redshift   |    TABLE   |           |
   +
   tickit_db  |    public   |   tickit_event_redshift  |    TABLE   |           |
   +
   tickit_db  |    public   |  tickit_listing_redshift |    TABLE   |           |
   +
   tickit_db  |    public   |   tickit_sales_redshift  |    TABLE   |           |
   +
   tickit_db  |    public   |   tickit_users_redshift  |    TABLE   |           |
   +
   tickit_db  |    public   |   tickit_venue_redshift  |    TABLE   |           |
```

table_acl 값이 null인 경우 해당 테이블에 액세스 권한이 명시적으로 부여되지 않은 것입니다.

SVV_RELATION_PRIVILEGES

현재 데이터베이스의 사용자, 롤 및 그룹에 명시적으로 부여된 관계 (테이블 및 보기) 권한을 보려면 SVV_RELATION_PRIVILEGES를 사용합니다.

SVV_RELATION_PRIVILEGES는 다음 사용자에게 표시됩니다.

- 슈퍼 사용자
- SYSLOG ACCESS UNRESTRICTED 권한이 있는 사용자

다른 사용자는 액세스할 수 있거나 소유한 ID만 볼 수 있습니다. 데이터 가시성에 대한 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 섹션을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
namespace_name	텍스트	지정된 관계가 존재하는 네임스페이스의 이름입니다.
relation_name	텍스트	관계의 이름입니다.
privilege_type	텍스트	권한의 유형입니다. 가능한 값은 INSERT, SELECT, UPDATE, DELETE, REFERENCES 또는 DROP입니다.
identity_id	정수	자격 증명의 ID입니다. 가능한 값은 사용자 ID, 역할 ID 또는 그룹 ID입니다.
identity_name	텍스트	자격 증명의 이름입니다.
identity_type	텍스트	자격 증명의 유형입니다. 가능한 값은 사용자, 역할, 그룹 또는 퍼블릭입니다.
admin_option	boolean	사용자가 다른 사용자 및 역할에 권한을 부여할 수 있는지 여부를 나타내는 값. 역할 및 그룹 자격 증명 유형에 대해서는 항상 false입니다.

샘플 쿼리

다음 예에서는 SVV_RELATION_PRIVILEGES.

```
SELECT
 namespace_name,relation_name,privilege_type,identity_name,identity_type,admin_option
 FROM svv_relation_privileges
WHERE relation_name = 'orders' AND privilege_type = 'SELECT';

 namespace_name | relation_name | privilege_type |  identity_name | identity_type |
 admin_option
----------------+---------------+----------------+----------------+---------------
+--------------
     public     |    orders     |     SELECT     |    reguser     |     user      |
 False
```

```
    public     |    orders     |     SELECT     |     role1      |     role      |
False
```

SVV_RLS_APPLIED_POLICY

SVV_RLS_APPLIED_POLICY를 사용하여 RLS 보호 관계를 참조하는 쿼리에서 RLS 정책의 적용을 추적합니다.

SVV_RLS_APPLIED_POLICY는 다음 사용자에게 표시됩니다.

- 슈퍼 사용자
- `sys:operator` 역할이 있는 사용자
- ACCESS SYSTEM TABLE 권한이 있는 사용자

sys:secadmin에는 이 시스템 권한이 부여되지 않습니다.

테이블 열

열 명칭	데이터 유형	설명
사용자 이름	텍스트	쿼리를 실행한 사용자의 이름입니다.
쿼리	정수	쿼리의 ID입니다.
xid	long	트랜잭션의 컨텍스트입니다.
pid	정수	쿼리를 실행하는 리더 프로세스입니다.
recordtime	시간	쿼리가 기록된 시간입니다.
명령	char(1)	RLS 정책이 적용된 명령입니다. 가능한 값은 k(알 수 없음), s(선택). u(업데이트), i(삽입), y(유틸리티) 및 d(삭제)입니다.
datname	텍스트	행 수준 보안 정책이 연결된 관계의 데이터베이스 이름입니다.
relschema	텍스트	행 수준 보안 정책이 연결된 관계의 스키마 이름입니다.
relname	텍스트	행 수준 보안 정책이 연결된 관계의 이름입니다.

열 명칭	데이터 유형	설명
polname	텍스트	관계에 연결된 행 수준 보안 정책의 이름입니다.
poldefault	char(1)	관계에 연결된 행 수준 보안 정책의 기본 설정입니다. 가능한 값은 기본 false 정책이 적용된 경우 false를 나타내는 f와 기본 true 정책이 적용된 경우 true를 나타내는 t입니다.

샘플 쿼리

다음 예는 SVV_RLS_APPLIED_POLICY의 결과를 보여줍니다. SVV_RLS_APPLIED_POLICY를 쿼리하려면 시스템 테이블 액세스 권한이 있어야 합니다.

```
-- Check what RLS policies were applied to the run query.
SELECT username, command, datname, relschema, relname, polname, poldefault
FROM svv_rls_applied_policy
WHERE datname = CURRENT_DATABASE() AND query = PG_LAST_QUERY_ID();

 username | command |  datname  | relschema |          relname          |      polname
  | poldefault
----------+---------+-----------+-----------+---------------------------
+-----------------+------------
   molly  |    s    | tickit_db |   public  | tickit_category_redshift |
 policy_concerts |
```

SVV_RLS_ATTACHED_POLICY

SVV_RLS_ATTACHED_POLICY를 사용하여 현재 연결된 데이터베이스에 하나 이상의 행 수준 보안 정책이 연결된 모든 관계 및 사용자의 목록을 봅니다.

sys:secadmin 역할이 부여된 사용자만 이 보기를 쿼리할 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
relschema	텍스트	행 수준 보안 정책이 연결된 관계의 스키마 이름입니다.

열 명칭	데이터 유형	설명
relname	텍스트	행 수준 보안 정책이 연결된 관계의 이름입니다.
relkind	텍스트	테이블과 같은 객체의 유형입니다.
polname	텍스트	관계에 연결된 행 수준 보안 정책의 이름입니다.
grantor	텍스트	이 정책을 연결한 사용자의 이름입니다.
grantee	텍스트	이 정책이 연결된 사용자 또는 역할의 이름입니다.
granteekind	텍스트	피부여자의 유형입니다. 가능한 값은 사용자 또는 역할입니다.
is_pol_on	boolean	테이블에서 행 수준 보안 정책이 설정되어 있는지 아니면 해제되어 있는지 여부를 나타내는 파라미터입니다. 가능한 값은 true와 false입니다.
is_rls_on	boolean	테이블에서 행 수준 보안이 설정되어 있는지 아니면 해제되어 있는지 여부를 나타내는 파라미터입니다. 가능한 값은 true와 false입니다.
rls_conjunction_type	character (3)	관계가 RLS 정책을 and로 결합하는지, or로 결합하는지를 나타내는 파라미터입니다.

샘플 쿼리

다음 예는 SVV_RLS_ATTACHED_POLICY의 결과를 보여줍니다.

```
--Inspect the policy in SVV_RLS_ATTACHED_POLICY
SELECT * FROM svv_rls_attached_policy;

 relschema |        relname           | relkind |     polname     | grantor | grantee
 | granteekind | is_pol_on | is_rls_on | rls_conjuntion_type
-----------+--------------------------+---------+-----------------+---------+----------
+-------------+-----------+-----------+---------------------
 public    | tickit_category_redshift |  table  | policy_concerts |   bob   |  analyst
 |    role     |    True   |    True   |      and
```

```
public    | tickit_category_redshift |  table  | policy_concerts |   bob   |  dbadmin
|    role     |    True   |    True   |      and
```

SVV_RLS_POLICY

SVV_RLS_POLICY를 사용하여, Amazon Redshift 클러스터에 생성된 모든 행 수준 보안 정책의 목록을 봅니다.

SVV_RLS_POLICY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
poldb	텍스트	행 수준 보안 정책이 생성된 데이터베이스의 이름입니다.
polname	텍스트	행 수준 보안 정책의 이름입니다.
polalias	텍스트	정책 정의에 사용된 테이블 별칭입니다.
polatts	텍스트	정책 정의에 제공된 속성입니다.
polqual	텍스트	CREATE POLICY 문의 USING 절에 제공된 정책 조건입니다.
polenabled	boolean	정책이 전역적으로 설정되어 있는지 여부를 나타냅니다.
polmodifiedby	텍스트	가장 최근에 정책을 생성하거나 수정한 사용자의 이름입니다.
polmodifiedtime	타임스탬프	정책이 생성되거나 마지막으로 수정된 시간의 타임스탬프입니다.

샘플 쿼리

다음 예는 SVV_RLS_POLICY의 결과를 보여줍니다.

```
-- Create some policies.
CREATE RLS POLICY pol1 WITH (a int) AS t USING ( t.a IS NOT NULL );
```

```
CREATE RLS POLICY pol2 WITH (c varchar(10)) AS t USING ( c LIKE '%public%');

-- Inspect the policy in SVV_RLS_POLICY
SELECT * FROM svv_rls_policy;

 poldb | polname | polalias |                    polatts                     |
         polqual               | polenabled | polmodifiedby |   polmodifiedtime
-------+---------+----------+------------------------------------------------
+--------------------------------------+------------+--------------
+---------------------
 my_db | pol1    | t        | [{"colname":"a","type":"integer"}]             |
 "t"."a" IS NOT NULL                  | t          | policy_admin  | 2022-02-11
 14:40:49
 my_db | pol2    | t        | [{"colname":"c","type":"character varying(10)"}] |
 "t"."c" LIKE CAST('%public%' AS TEXT) | t          | policy_admin  | 2022-02-11
 14:41:28
```

SVV_RLS_RELATION

RLS로 보호되는 모든 관계의 목록을 보려면 SVV_RLS_RELATION을 사용합니다.

SVV_RLS_RELATION은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
datname	텍스트	관계가 포함된 데이터베이스의 이름입니다.
relschema	텍스트	관계가 포함된 스키마의 이름입니다.
relname	텍스트	관계의 이름입니다.
relkind	텍스트	테이블, 보기 등, 관계의 유형입니다.
is_rls_on	boolean	관계가 RLS로 보호되는지 여부를 나타내는 파라미터입니다.
is_rls_datashare_on	boolean	관계가 데이터 공유에 대해 RLS로 보호되는지 여부를 나타내는 파라미터입니다.

열 명칭	데이터 유형	설명
rls_conjunction_type	character(3)	관계가 RLS 정책을 and로 결합하는지, or로 결합하는지를 나타내는 파라미터입니다.
rls_datashare_conjunction_type	character(3)	관계가 데이터 공유를 통해 RLS 정책을 and로 결합하는지, or로 결합하는지를 나타내는 파라미터입니다.

샘플 쿼리

다음 예는 SVV_RLS_RELATION의 결과를 보여줍니다.

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON FOR DATASHARES;


--Inspect RLS state on the relations using SVV_RLS_RELATION.
SELECT datname, relschema, relname, relkind, is_rls_on, is_rls_datashare_on FROM
 svv_rls_relation ORDER BY relname;

  datname  | relschema |        relname           | relkind | is_rls_on |
 is_rls_datashare_on | rls_conjunction_type | rls_datashare_conjunction_type
-----------+-----------+--------------------------+---------+-----------
+---------------------+----------------------+--------------------------------
 tickit_db |   public  | tickit_category_redshift |  table  |      t    |           t
       |          and         |              and
(1 row)
```

SVV_ROLE_GRANTS

클러스터에서 명시적으로 역할이 부여된 역할 목록을 보려면 SVV_ROLE_GRANTS를 사용합니다.

SVV_ROLE_GRANTS는 다음 사용자에게 표시됩니다.

- 슈퍼 사용자
- ACCESS SYSTEM TABLE 권한이 있는 사용자

다른 사용자는 액세스할 수 있거나 소유한 ID만 볼 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
role_id	정수	역할의 ID입니다.
role_name	텍스트	역할의 이름.
granted_role_id	정수	부여된 역할의 ID입니다.
granted_role_name	텍스트	부여된 역할의 이름입니다.

샘플 쿼리

다음 예에서는 SVV_ROLE_GRANTS의 출력을 반환합니다.

```
GRANT ROLE role1 TO ROLE role2;
GRANT ROLE role2 TO ROLE role3;

SELECT role_name, granted_role_name FROM svv_role_grants;

 role_name |  granted_role_name
-----------+--------------------
   role2   |      role1
   role3   |      role2
(2 rows)
```

SVV_ROLES

SVV_ROLES를 사용하여 역할 정보를 봅니다.

이 테이블은 모든 사용자에게 표시됩니다.

테이블 열

열 명칭	데이터 유형	설명
role_id	정수	역할 ID입니다.
role_name	텍스트	역할의 이름.

열 명칭	데이터 유형	설명
role_owner	텍스트	역할 소유자의 이름입니다.
external_id	텍스트	서드 파티 자격 증명 공급자의 역할의 고유 식별자입니다.

샘플 쿼리

다음 예에서는 SVV_ROLES의 출력을 반환합니다.

```
SELECT role_name,role_owner FROM svv_roles WHERE role_name IN ('role1', 'role2');

 role_name | role_owner
-----------+------------
   role1   | superuser
   role2   | superuser
```

SVV_SCHEMA_PRIVILEGES

SVV_SCHEMA_PRIVILEGES를 사용하여 현재 데이터베이스의 사용자, 롤 및 그룹에 명시적으로 부여된 스키마 권한을 봅니다.

SVV_SCHEMA_PRIVILEGES는 다음 사용자에게 표시됩니다.

- 슈퍼 사용자
- ACCESS SYSTEM TABLE 권한이 있는 사용자

다른 사용자는 액세스할 수 있거나 소유한 ID만 볼 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
namespace_name	텍스트	지정된 스키마가 존재하는 네임스페이스입니다.
privilege_type	텍스트	권한의 유형입니다. privilege_scope가 스키마인 권한의 경우 가능한 값

열 명칭	데이터 유형	설명
		은 CREATE, USAGE, ALTER입니다. 스키마 이외의 privilege_scope 값의 경우 가능한 값에는 권한 범위에서 사용할 수 있는 모든 권한 유형이 포함됩니다.
identity_id	정수	자격 증명의 ID입니다. 가능한 값은 사용자 ID, 역할 ID 또는 그룹 ID입니다.
identity_name	텍스트	자격 증명의 이름입니다.
identity_type	텍스트	자격 증명의 유형입니다. 가능한 값은 사용자, 역할, 그룹 또는 퍼블릭입니다.
admin_option	boolean	사용자가 다른 사용자 및 역할에 권한을 부여할 수 있는지 여부를 나타내는 값. 역할 및 그룹 자격 증명 유형에 대해서는 항상 false입니다.
privilege_scope	텍스트	privilege_type에 지정된 권한의 범위입니다. 가능한 값은 다음과 같습니다. • 스키마 • TABLES • FUNCTIONS 범위 지정 권한에 관한 자세한 내용은 범위가 지정된 권한 섹션으로 이동해 참조하시기 바랍니다.

샘플 쿼리

다음 예에서는 SVV_SCHEMA_PRIVILEGES의 결과를 표시합니다.

```
SELECT namespace_name,privilege_type,identity_name,identity_type,admin_option FROM
 svv_schema_privileges
WHERE namespace_name = 'test_schema1';

 namespace_name | privilege_type |  identity_name | identity_type | admin_option
----------------+----------------+----------------+---------------+--------------
 test_schema1   |    USAGE       |     reguser    |     user      |   False
 test_schema1   |    USAGE       |     role1      |     role      |   False
```

SVV_SCHEMA_QUOTA_STATE

각 스키마에 대한 할당량 및 현재 디스크 사용량을 표시합니다.

일반 사용자는 USAGE 권한이 있는 스키마에 대한 정보를 볼 수 있습니다. 수퍼유저는 현재 데이터베이스의 모든 스키마에 대한 정보를 볼 수 있습니다.

SVV_SCHEMA_QUOTA_STATE는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> 이 보기는 프로비저닝된 클러스터를 쿼리할 때만 사용할 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
schema_id	정수	네임스페이스 또는 스키마 ID입니다.
schema_name	character(128)	네임스페이스 또는 스키마 이름입니다.
schema_owner	정수	스키마 소유자의 내부 사용자 ID입니다.
할당량	정수	스키마에서 사용할 수 있는 디스크 공간(MB)입니다.

열 명칭	데이터 유형	설명
disk_usage	정수	스키마에서 현재 사용 중인 디스크 공간(MB)입니다.
disk_usage_pct	double precision	구성된 할당량 중 스키마에서 현재 사용하고 있는 디스크 공간 백분율입니다.

샘플 쿼리

다음 예에서는 스키마에 대한 할당량 및 현재 디스크 사용량을 표시합니다.

```
SELECT TRIM(SCHEMA_NAME) "schema_name", QUOTA, disk_usage, disk_usage_pct FROM
 svv_schema_quota_state
WHERE SCHEMA_NAME = 'sales_schema';
schema_name   | quota | disk_usage | disk_usage_pct
--------------+-------+------------+----------------
sales_schema  | 2048  | 30         | 1.46
(1 row)
```

SVV_SYSTEM_PRIVILEGES

SVV_SYSTEM_PRIVILEGES는 다음 사용자에게 표시됩니다.

- 슈퍼 사용자
- ACCESS SYSTEM TABLE 권한이 있는 사용자

다른 사용자는 액세스할 수 있거나 소유한 ID만 볼 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
system_privilege	텍스트	권한 세트의 이름입니다.

열 명칭	데이터 유형	설명
identity_id	정수	자격 증명의 ID입니다. 가능한 값은 사용자 ID 또는 역할 ID입니다.
identity_name	텍스트	자격 증명의 이름입니다.
identity_type	텍스트	자격 증명의 유형입니다. 가능한 값은 사용자 또는 역할입니다.

샘플 쿼리

다음 예에서는 지정된 정책의 세부 정보를 표시합니다.

```
SELECT system_privilege,identity_name,identity_type FROM svv_system_privileges
WHERE system_privilege = 'ALTER TABLE' AND identity_name = 'sys:superuser';

 system_privilege | identity_name | identity_type
------------------+---------------+---------------
   ALTER TABLE    | sys:superuser |     role
```

SVV_TABLE_INFO

데이터베이스에 있는 테이블에 대한 요약 정보를 보여 줍니다. 이 뷰는 시스템 테이블을 필터링하고 사용자 정의 테이블만을 보여 줍니다.

SVV_TABLE_INFO 보기를 사용하여 쿼리 성능에 영향을 미칠 수 있는 테이블 설계 문제를 진단하고 해결할 수 있습니다. 여기에는 압축 인코딩, 배포 키, 정렬 스타일, 데이터 배포 스큐, 테이블 크기, 통계 문제 등이 포함됩니다. SVV_TABLE_INFO 보기는 빈 테이블에 대한 어떠한 정보도 반환하지 않습니다.

SVV_TABLE_INFO 보기는 STV_BLOCKLIST, STV_NODE_STORAGE_CAPACITY, STV_TBL_PERM 및 STV_SLICES 시스템 테이블과 PG_DATABASE, PG_ATTRIBUTE, PG_CLASS, PG_NAMESPACE 및 PG_TYPE 카탈로그 테이블의 정보를 요약합니다.

SVV_TABLE_INFO는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오. 사용자가 보기를 쿼리할 수 있도록 하려면 사용자에게 SVV_TABLE_INFO에 대한 SELECT 권한을 부여합니다.

테이블 열

열 명칭	데이터 유형	설명
database	텍스트	데이터베이스 이름.
schema	텍스트	스키마 이름
table_id	oid	테이블 ID
table	텍스트	테이블 이름.
encoded	텍스트	열에 압축 인코딩이 정의되어 있는지를 나타내는 값.
diststyle	텍스트	키 분산이 정의된 경우, 분산 스타일 또는 분산 키 열. 가능한 값은 EVEN, KEY(*column*), ALL, AUTO(ALL) , AUTO(EVEN) 및 AUTO(KEY(*column*))입니다.
sortkey1	텍스트	정렬 키가 정의된 경우, 정렬 키의 첫 번째 열. 가능한 값은 *column*, AUTO(SORT KEY) 및 AUTO(SORT KEY(*column*))입니다.
max_varchar	정수	VARCHAR 데이터 형식을 사용하는 가장 큰 열의 크기.
sortkey1_enc	character(32)	정렬 키가 정의된 경우, 정렬 키 첫 번째 열의 압축 인코딩.
sortkey_num	정수	정렬 키로 정의된 열의 수.
size	bigint	테이블의 크기(1MB 데이터 블록 단위).

열 명칭	데이터 유형	설명
pct_used	numeric(10,4)	테이블이 사용하는 가용 공간의 백분율.
empty	bigint	내부용. 이 열은 사용되지 않으며 향후 릴리스에서 삭제됩니다.
unsorted	numeric(5,2)	테이블에서 정렬되지 않은 행의 백분율.
stats_off	numeric(5,2)	테이블의 통계가 얼마나 부실한지 나타내는 숫자. 0은 최신이고 100은 오래된 상태입니다.
tbl_rows	numeric(38,0)	테이블에 포함된 행의 총 수입니다. 이 값에는 삭제 표시만 되어있고 아직 정리되지 않은 행이 포함됩니다.
skew_sortkey1	numeric(19,2)	정렬 키가 정의된 경우, 가장 큰 비정렬 키 열의 크기 대 정렬 키 첫 번째 열의 크기의 비율. 이 값을 사용하여 정렬 키의 효율성을 평가합니다.
skew_rows	numeric(19,2)	행이 가장 많은 조각의 행 수 대 행이 가장 적은 조각의 행 수의 비율.
estimated_visible_rows	numeric(38,0)	테이블의 추정 행입니다. 이 값에 삭제 표시된 행은 포함되지 않습니다.

열 명칭	데이터 유형	설명

risk_event	텍스트	테이블에 대한 위험 정보입니다. 이 필드는 다음과 같이 여러 부분으로 구분됩니다.
		
		`risk_type |xid|timestamp`
		
		• `risk_type` , 여기서 1은 `COPY command with the EXPLICIT_IDS option`이 실행되었음을 나타냅니다. Amazon Redshift는 더 이상 테이블에서 IDENTITY 열의 고유성을 확인하지 않습니다. 자세한 내용은 EXPLICIT_IDS 단원을 참조하십시오.
		• 트랜잭션 ID인 `xid`는 위험을 초래한 것입니다.
		• `timestamp` 는 COPY 명령이 실행된 시간입니다.
		
		다음 예제는 필드에서 이러한 값을 보여 줍니다.
		
		`1|1107|2019-06-22 07:16:11.292952`

vacuum_sort_benefit	numeric(12,2)	vacuum sort를 실행할 때 스캔 쿼리 성능에 대해 예상되는 최대 향상율입니다.
create_time	시간대 미포함 TIMESTAMP	테이블이 생성된 타임스탬프입니다.

샘플 쿼리

다음 예는 데이터베이스 내 모든 사용자 정의 테이블의 인코딩, 분산 스타일, 정렬 및 데이터 스큐를 보여 줍니다. 여기서 "table"은 예약된 단어이므로 큰따옴표로 묶어야 합니다.

```
select "table", encoded, diststyle, sortkey1, skew_sortkey1, skew_rows
from svv_table_info
order by 1;

table          | encoded | diststyle       | sortkey1     | skew_sortkey1 | skew_rows
---------------+---------+-----------------+--------------+---------------+----------
category       | N       | EVEN            |              |               |
date           | N       | ALL             | dateid       |          1.00 |
event          | Y       | KEY(eventid)    | dateid       |          1.00 |      1.02
listing        | Y       | KEY(listid)     | dateid       |          1.00 |      1.01
sales          | Y       | KEY(listid)     | dateid       |          1.00 |      1.02
users          | Y       | KEY(userid)     | userid       |          1.00 |      1.01
venue          | N       | ALL             | venueid      |          1.00 |
(7 rows)
```

SVV_TABLES

SVV_TABLES는 로컬 및 외부 카탈로그에서 테이블을 확인하는 데 사용됩니다.

SVV_TABLES는 모든 사용자가 볼 수 있습니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
table_catalog	텍스트	테이블이 속한 카탈로그 이름
table_schema	텍스트	테이블의 스키마 이름
table_name	텍스트	테이블의 이름
table_type	텍스트	테이블 유형. 가능한 값은 view, external table 및 base table입니다.

열 명칭	데이터 유형	설명
remarks	텍스트	설명.

SVV_TRANSACTIONS

데이터베이스의 테이블을 현재 잠그고 있는 트랜잭션에 대한 레코드 정보. 열려 있는 트랜잭션을 식별하고 경합 문제를 잠그려면 SVV_TRANSACTIONS 뷰를 사용합니다. 자세한 내용은 동시 쓰기 작업 관리 및 LOCK 섹션을 참조하세요.

SVV_TRANSACTIONS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
txn_owner	텍스트	트랜잭션 소유자의 이름.
txn_db	텍스트	bigint
xid	bigint	트랜잭션 ID.
pid	정수	잠금에 연결된 프로세스 ID.
txn_start	타임스탬프	트랜잭션의 시작 시간.
lock_mode	텍스트	이 프로세스에 의해 유지 또는 요청된 잠금 모드의 이름. `lock_mode` 가 `Exclusive Lock` 이고 `granted`가 true(t)인 경우, 이 트랜잭션 ID는 열린 트랜잭션입니다.
lockable_object_type	텍스트	잠금을 요청 또는 유지하는 객체의 형식으로서 테이블인 경우에는 `relation`, 트랜잭션인

열 명칭	데이터 유형	설명
		경우에는 transactionid 입니다.
관계	정수	잠금을 획득한 테이블(관계)의 테이블 ID. lockable_object_type 이 transactionid 인 경우, 이 값은 NULL입니다.
granted	boolean	잠금이 부여되었는지(t) 또는 보류 중인지(f)를 나타내는 값.

샘플 쿼리

다음 명령은 모든 활성 트랜잭션 및 각 트랜잭션이 요청한 잠금을 보여 줍니다.

```
select * from svv_transactions;

 txn_
 lockable_
 owner | txn_db |  xid   |  pid  |         txn_start          |      lock_mode      |
 object_type    | relation | granted
-------+--------+--------+-------+----------------------------+---------------------
+----------------+----------+---------
 root  | dev    | 438484 | 22223 | 2016-03-02 18:42:18.862254 | AccessShareLock     |
 relation       |   100068 | t
 root  | dev    | 438484 | 22223 | 2016-03-02 18:42:18.862254 | ExclusiveLock       |
 transactionid  |          | t
 root  | tickit | 438490 | 22277 | 2016-03-02 18:42:48.084037 | AccessShareLock     |
 relation       |    50860 | t
 root  | tickit | 438490 | 22277 | 2016-03-02 18:42:48.084037 | AccessShareLock     |
 relation       |    52310 | t
 root  | tickit | 438490 | 22277 | 2016-03-02 18:42:48.084037 | ExclusiveLock       |
 transactionid  |          | t
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | AccessExclusiveLock |
 relation       |   100068 | f
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | RowExclusiveLock    |
 relation       |    16688 | t
```

```
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | AccessShareLock     |
 relation       |   100064 | t
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | AccessExclusiveLock |
 relation       |   100166 | t
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | AccessExclusiveLock |
 relation       |   100171 | t
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | AccessExclusiveLock |
 relation       |   100190 | t
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | ExclusiveLock       |
 transactionid  |          | t
(12 rows)

(12 rows)
```

SVV_USER_GRANTS

클러스터에서 명시적으로 역할이 부여된 사용자 목록을 보려면 SVV_USER_GRANTS를 사용합니다.

SVV_USER_GRANTS는 다음 사용자에게 표시됩니다.

- 슈퍼 사용자
- ACCESS SYSTEM TABLE 권한이 있는 사용자

다른 사용자는 자신에게 명시적으로 부여된 역할만 볼 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	사용자의 사용자 ID입니다.
user_name	텍스트	사용자의 이름입니다.
role_id	정수	부여된 역할에 대한 역할 ID입니다.
role_name	텍스트	부여된 역할의 역할 이름입니다.
admin_option	boolean	사용자가 다른 사용자와 역할에 역할을 부여할 수 있는지 여부를 나타내는 값.

샘플 쿼리

다음 쿼리는 사용자에게 역할을 부여하고 명시적으로 역할이 부여된 사용자 목록을 표시합니다.

```
GRANT ROLE role1 TO reguser;
GRANT ROLE role2 TO reguser;
GRANT ROLE role1 TO superuser;
GRANT ROLE role2 TO superuser;

SELECT user_name,role_name,admin_option FROM svv_user_grants;

 user_name | role_name | admin_option
-----------+-----------+--------------
 superuser |  role1    | False
 reguser   |  role1    | False
 superuser |  role2    | False
  reguser  |  role2    | False
```

SVV_USER_INFO

SVV_USER_INFO 뷰를 사용하여 Amazon Redshift 데이터베이스 사용자에 대한 데이터를 검색할 수 있습니다.

SVV_USER_INFO는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_name	텍스트	역할에 대한 사용자 이름입니다.
user_id	정수	사용자의 사용자 ID입니다.
createdb	boolean	사용자가 데이터베이스를 생성할 수 있는 권한이 있는지 여부를 나타내는 값
수퍼유저	boolean	사용자가 수퍼유저인지 여부를 나타내는 값

열 명칭	데이터 유형	설명
catalog_update	boolean	사용자가 시스템 카탈로그를 업데이트할 수 있는지 여부를 나타내는 값
connection_limit	텍스트	사용자가 열 수 있는 연결 수
syslog_access	텍스트	사용자가 시스템 로그에 액세스할 수 있는 권한이 있는지 여부를 나타내는 값 RESTRICTED 및 UNRESTRICTED 의 두 가지 값이 가능합니다. RESTRICTED 이면 수퍼유저가 아닌 사용자가 본인의 레코드를 볼 수 있고, UNRESTRICTED 이면 수퍼유저가 아닌 사용자가 SELECT 권한이 있는 시스템 뷰 및 테이블에 있는 모든 레코드를 볼 수 있습니다.
last_ddl_timestamp	타임스탬프	사용자가 데이터 정의 언어(DDL) create 문을 마지막으로 실행한 타임스탬프
session_timeout	정수	세션이 시간 초과되기 전에 비활성 또는 유휴 상태로 유지되는 최대 시간(초)입니다. 0은 시간 제한이 설정되지 않았음을 나타냅니다. 클러스터의 유휴 또는 비활성 시간 제한 설정에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 Amazon Redshift의 할당량 및 제한 섹션을 참조하세요.
external_user_id	텍스트	서드 파티 자격 증명 공급자 사용자의 고유 식별자입니다.

샘플 쿼리

다음 명령은 SVV_USER_INFO에서 사용자 정보를 검색합니다.

```
SELECT * FROM SVV_USER_INFO;
```

SVV_VACUUM_PROGRESS

이 뷰는 현재 진행 중인 vacuum 작업 완료까지의 예상 시간을 반환합니다.

SVV_VACUUM_PROGRESS는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_VACUUM_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

SVV_VACUUM_SUMMARY에 대한 자세한 내용은 SVV_VACUUM_SUMMARY 섹션을 참조하세요.

SVL_VACUUM_PERCENTAGE에 대한 자세한 내용은 SVL_VACUUM_PERCENTAGE 섹션을 참조하세요.

> **Note**
>
> 이 보기는 프로비저닝된 클러스터를 쿼리할 때만 사용할 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
table_name	텍스트	현재 vacuum 중인 테이블 또는 진행 중인 작업이 없는 경우, 마지막으로 vacuum된 테이블의 이름.
status	텍스트	vacuum 작업의 일환으로 수행되고 있는 현재 활동의 설명: • 초기화 • 정렬 • 병합 • 삭제 • Select • Failed • 완료 • 건너뜀 • INTERLEAVED SORTKEY 순서 구축
time_remaining_estimate	텍스트	현재 vacuum 작업이 완료되기까지 남은 예상 시간(분/초). 예: **5m 10s** vacuum이 첫 번째 정렬 작업을 완료할 때까지는 예상 시간이 반환되지 않습니다. 진행 중인 vacuum이 없는 경우,

열 명칭	데이터 유형	설명
		STATS 열과 비어 있는 TIME_REMAINING_ESTIMATE 열에 마지막으로 수행된 vacuum이 **Completed** 로 표시됩니다. 예상 시간은 vacuum이 진행됨에 따라 일반적으로 더 정확해집니다.

샘플 쿼리

몇 분 간격으로 실행된 다음의 쿼리들은 SALESNEW라는 이름의 큰 테이블이 vacuum되고 있음을 보여 줍니다.

```
select * from svv_vacuum_progress;

table_name    |            status              | time_remaining_estimate
--------------+--------------------------------+-------------------------
salesnew      |  Vacuum: initialize salesnew  |
(1 row)
...
select * from svv_vacuum_progress;

table_name   |         status         | time_remaining_estimate
-------------+------------------------+-------------------------
salesnew     |  Vacuum salesnew sort  | 33m 21s
(1 row)
```

다음 쿼리는 현재 진행 중인 vacuum 작업이 없음을 보여 줍니다. vacuum될 마지막 테이블은 SALES 테이블이었습니다.

```
select * from svv_vacuum_progress;

table_name   |  status  | time_remaining_estimate
-------------+----------+-------------------------
  sales      | Complete |
(1 row)
```

SVV_VACUUM_SUMMARY

SVV_VACUUM_SUMMARY 뷰는 STL_VACUUM, STL_QUERY 및 STV_TBL_PERM 테이블을 조인하여 시스템에 의해 기록되는 vacuum 작업에 대한 정보를 요약합니다. 이 뷰는 vacuum 트랜잭션마다 테이블당 하나의 행을 반환합니다. 이 뷰는 작업 경과 시간, 생성된 정렬 파티션의 수, 필요한 병합 증분의 수, 행에 있는 델타 및 작업 수행 전후의 블록 수를 기록합니다.

SVV_VACUUM_SUMMARY는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_VACUUM_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

SVV_VACUUM_PROGRESS에 대한 자세한 내용은 SVV_VACUUM_PROGRESS 섹션을 참조하세요.

SVL_VACUUM_PERCENTAGE에 대한 자세한 내용은 SVL_VACUUM_PERCENTAGE 섹션을 참조하세요.

> **Note**
>
> 이 보기는 프로비저닝된 클러스터를 쿼리할 때만 사용할 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
table_name	텍스트	vacuum된 테이블의 이름.
xid	bigint	VACUUM 작업의 트랜잭션 ID.
sort_partitions	bigint	vacuum 작업의 정렬 단계 도중 생성되는 정렬된 파티션의 수.
merge_increments	bigint	vacuum 작업의 병합 단계를 완료하는 데 필요한 병합 증분의 수.
경과 시간	bigint	vacuum 작업의 경과된 런타임(마이크로초).

열 명칭	데이터 유형	설명
row_delta	bigint	vacuum 전과 후의 테이블의 총 행 수의 차이.
sortedrow_delta	bigint	vacuum 전과 후의 정렬된 테이블 행 수의 차이.
block_delta	정수	vacuum 전과 후의 테이블 블록 수의 차이.
max_merge_partitions	정수	이 열은 성능 분석에 사용되며 병합 단계 반복당 테이블에 대해 vacuum이 처리할 수 있는 최대 파티션 수를 나타냅니다. (Vacuum은 정렬되지 않은 리전을 하나 이상의 정렬된 파티션으로 정렬합니다. 테이블의 열 수와 현재 Amazon Redshift 구성에 따라 병합 단계는 단일 병합 반복에서 최대 파티션 수를 처리할 수 있습니다. 병합 단계는 정렬된 파티션 수가 최대 병합 파티션 수를 초과해도 작동하지만 더 많은 병합 반복이 필요합니다.)

샘플 쿼리

다음 쿼리는 세 가지 테이블에서의 vacuum 작업에 대한 통계를 반환합니다. SALES 테이블은 두 번 vacuum되었습니다.

```
select table_name, xid, sort_partitions as parts, merge_increments as merges,
elapsed_time, row_delta, sortedrow_delta as sorted_delta, block_delta
from svv_vacuum_summary
order by xid;

table_  | xid  |parts|merges| elapsed_ | row_    | sorted_ | block_
name    |      |     |      | time     | delta   | delta   | delta
--------+------+-----+------+----------+---------+---------+--------
users   | 2985 |   1 |    1 | 61919653 |       0 |   49990 |      20
category| 3982 |   1 |    1 | 24136484 |       0 |      11 |       0
sales   | 3992 |   2 |    1 | 71736163 |       0 | 1207192 |      32
sales   | 4000 |   1 |    1 | 15363010 | -851648 | -851648 |    -140
(4 rows)
```

SYS 모니터링 뷰

모니터링 뷰는 프로비저닝된 클러스터 및 서버리스 작업 그룹의 쿼리 및 워크로드 리소스 사용량을 모니터링하는 데 사용되는 Amazon Redshift Serverless의 시스템 뷰입니다. 이러한 보기는 `pg_catalog` 스키마에 있습니다. 이러한 보기에서 제공하는 정보를 표시하려면 SQL SELECT 문을 실행합니다.

달리 명시되지 않는 한, 이러한 보기는 Amazon Redshift 클러스터 및 Amazon Redshift Serverless 작업 그룹에서 사용할 수 있습니다.

SYS_SERVERLESS_USAGE는 Amazon Redshift Serverless의 사용량 데이터만 수집합니다.

주제

- SYS_LOAD_DETAIL
- SYS_LOAD_ERROR_DETAIL
- SYS_LOAD_HISTORY
- SYS_MV_REFRESH_HISTORY
- SYS_MV_STATE
- SYS_PROCEDURE_CALL
- SYS_PROCEDURE_MESSAGES
- SYS_QUERY_DETAIL
- SYS_QUERY_EXPLAIN
- SYS_QUERY_HISTORY
- SYS_QUERY_TEXT
- SYS_RESTORE_LOG
- SYS_RESTORE_STATE
- SYS_SCHEMA_QUOTA_VIOLATIONS
- SYS_SERVERLESS_USAGE
- SYS_SESSION_HISTORY
- SYS_SPATIAL_SIMPLIFY
- SYS_STREAM_SCAN_ERRORS
- SYS_STREAM_SCAN_STATES
- SYS_TRANSACTION_HISTORY
- SYS_UDF_LOG
- SYS_UNLOAD_DETAIL
- SYS_UNLOAD_HISTORY
- SYS_USERLOG
- SYS_VACUUM_HISTORY

SYS_ANALYZE_COMPRESSION_HISTORY

COPY 또는 ANALYZE COMPRESSION 명령을 수행하는 동안 압축 분석 작업에 대한 세부 정보를 기록합니다.

SYS_ANALYZE_COMPRESSION_HISTORY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	항목을 생성한 사용자의 ID입니다.
start_time	타임스탬프	압축 분석 작업을 시작한 시간입니다.
transaction_id	bigint	압축 분석 작업의 트랜잭션 ID입니다.
table_id	정수	분석한 테이블의 테이블 ID입니다.
table_name	character(128)	분석한 테이블의 이름입니다.
column_position	정수	압축 인코딩을 확인하기 위해 분석한 테이블의 열 인덱스입니다.
old_encoding	character(15)	압축 분석 이전의 인코딩 유형입니다.
new_encoding	character(15)	압축 분석 이후의 인코딩 유형입니다.
mode	character(14)	가능한 값은 다음과 같습니다. PRESET new_encoding 이 열 데이터 형식을 기준으로 Amazon Redshift COPY 명령에 의해 결정되도록 지정합니다. 데이터가 샘플링되지 않습니다. ON new_encoding 이 샘플 데이터의 분석을 기준으로 Amazon Redshift COPY 명령에 의해 결정되도록 지정합니다.

열 명칭	데이터 유형	설명
		ANALYZE ONLY new_encoding 이 샘플 데이터의 분석을 기준으로 Amazon Redshift ANALYZE COMPRESSION 명령에 의해 결정되도록 지정합니다. 그러나 분석된 열의 인코딩 유형은 변경되지 않습니다.

샘플 쿼리

다음 예에서는 동일한 세션에서 실행된 마지막 COPY 명령으로 lineitem 테이블에 대한 압축 분석의 세부 정보를 검사합니다.

```
select transaction_id, table_id, btrim(table_name) as table_name, column_position,
 old_encoding, new_encoding, mode
from sys_analyze_compression_history
where transaction_id = (select transaction_id from sys_query_history where query_id =
 pg_last_copy_id()) order by column_position;

 transaction_id  |  table_id   | table_name | column_position |  old_encoding   |
 new_encoding   |      mode
-----------------+-------------+------------+-----------------+-----------------
+-----------------+-------------
      8196       |   248126    | lineitem   |        0        | mostly32        |
 mostly32        | ON
      8196       |   248126    | lineitem   |        1        | mostly32        | lzo
           | ON
      8196       |   248126    | lineitem   |        2        | lzo             |
 delta32k        | ON
      8196       |   248126    | lineitem   |        3        | delta           | delta
           | ON
      8196       |   248126    | lineitem   |        4        | bytedict        |
 bytedict        | ON
      8196       |   248126    | lineitem   |        5        | mostly32        |
 mostly32        | ON
      8196       |   248126    | lineitem   |        6        | delta           | delta
           | ON
      8196       |   248126    | lineitem   |        7        | delta           | delta
           | ON
```

```
      8196      |   248126    | lineitem   |        8        | lzo             | zstd
           | ON
      8196      |   248126    | lineitem   |        9        | runlength       | zstd
           | ON
      8196      |   248126    | lineitem   |       10        | delta           | lzo
           | ON
      8196      |   248126    | lineitem   |       11        | delta           | delta
           | ON
      8196      |   248126    | lineitem   |       12        | delta           | delta
           | ON
      8196      |   248126    | lineitem   |       13        | bytedict        | zstd
           | ON
      8196      |   248126    | lineitem   |       14        | bytedict        | zstd
           | ON
      8196      |   248126    | lineitem   |       15        | text255         | zstd
           | ON
(16 rows)
```

SYS_ANALYZE_HISTORY

ANALYZE 작업에 대한 세부 정보를 로깅합니다.

SYS_ANALYZE_HISTORY는 수퍼유저에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	항목을 생성한 사용자의 ID입니다.
transaction_id	long	트랜잭션 ID.
query_id	long	SYS_QUERY_HISTORY의 쿼리 식별자입니다.
database_name	char(30)	데이터베이스의 이름입니다.
table_name	char(30)	테이블의 이름
table_id	정수	테이블의 ID입니다.

열 명칭	데이터 유형	설명
is_automatic	char(1)	작업에 Amazon Redshift ANALYZE 작업이 기본적으로 포함되어 있는 경우 이 값은 참(t)입니다. ANALYZE 명령이 명시적으로 실행된 경우 이 값은 거짓(f)입니다.
status	char(15)	ANALYZE 명령의 결과입니다. 가능한 값은 Full, Skipped, PredicateColumn입니다.
start_time	타임스탬프	ANALYZE 작업이 실행되기 시작한 시간(UTC)입니다.
end_time	타임스탬프	ANALYZE 작업이 실행을 마친 시간(UTC)입니다.
rows	double	테이블에 포함된 행의 총 수입니다.
modified_rows	double	마지막 ANALYZE 작업 이후 수정된 행의 총 수입니다.
analyze_threshold_percent	정수	analyze_threshold_percent 파라미터의 값입니다.
last_analyze_time	타임스탬프	이전에 테이블을 분석한 시간(UTC)입니다.

샘플 쿼리

```
 user_id | transaction_id | database_name | schema_name |      table_name     |
 table_id | is_automatic | Status |         start_time         |          end_time
     | rows | modified_rows | analyze_threshold_percent |  last_analyze_time
---------+----------------+---------------+-------------+---------------------
+----------+--------------+--------+----------------------------+----------
+-----------------+------+--------------+--------------------------
+---------------------
     101 |           8006 |           dev |      public | test_table_562bf8dc
 |   110427 |            f |   Full | 2023-09-21 18:33:08.504646 | 2023-09-21
 18:33:24.296498 |    5 |             5 |                          0 | 2000-01-01
 00:00:00
```

SYS_APPLIED_MASKING_POLICY_LOG

SYS_APPLIED_MASKING_POLICY_LOG를 사용하여 DDM으로 보호되는 관계를 참조하는 쿼리에서 동적 데이터 마스킹 정책의 적용을 추적합니다.

SYS_APPLIED_MASKING_POLICY_LOG는 다음 사용자에게 표시됩니다.

- 슈퍼 사용자
- `sys:operator` 역할이 있는 사용자
- ACCESS SYSTEM TABLE 권한이 있는 사용자

일반 사용자에게는 0개의 행이 표시됩니다.

참고로 SYS_APPLIED_MASKING_POLICY_LOG는 `sys:secadmin` 역할을 가진 사용자에게는 표시되지 않습니다.

동적 데이터 마스킹에 대한 자세한 내용은 동적 데이터 마스킹 섹션을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
policy_name	텍스트	마스킹 정책의 이름입니다.
user_id	텍스트	쿼리를 실행한 사용자의 ID입니다.
record_time	타임스탬프	시스템 뷰 항목이 기록된 시간입니다.
session_id	int	프로세스 ID.
transaction_id	long	트랜잭션 ID.
query_id	int	쿼리 ID.
database_name	텍스트	쿼리가 실행된 데이터베이스의 이름입니다.

열 명칭	데이터 유형	설명
relation_name	텍스트	마스킹 정책이 적용된 테이블의 이름입니다.
schema_name	텍스트	테이블이 있는 스키마의 이름입니다.
attachment_id	long	연결된 마스킹 정책의 ID입니다.
relation_kind	텍스트	마스킹 정책이 적용된 관계의 유형입니다. 가능한 값은 TABLE, VIEW, LATE BINDING VIEW 및 MATERIALIZED VIEW 입니다.

샘플 쿼리

다음 예에서는 mask_credit_card_full 마스킹 정책이 credit_db.public.credit_cards 테이블에 연결된 것을 보여줍니다.

```
select policy_name, database_name, relation_name, schema_name, relation_kind
from sys_applied_masking_policy_log;

policy_name           | database_name | relation_name | schema_name | relation_kind
----------------------+---------------+---------------+-------------+---------------
mask_credit_card_full | credit_db     | credit_cards  | public      | table

(1 row)
```

SYS_AUTO_TABLE_OPTIMIZATION

자동 최적화를 위해 정의된 테이블에 Amazon Redshift에서 수행한 자동화된 작업을 기록합니다.

SYS_AUTO_TABLE_OPTIMIZATION은 슈퍼유저에게만 표시됩니다. 자세한 내용은 [시스템 테이블 및 뷰에 있는 데이터의 가시성] 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
transaction_id	long	트랜잭션 식별자입니다.
session_id	int	alter 명령을 실행한 프로세스의 세션 식별자입니다.
table_id	int	테이블 식별자입니다.
alter_table_type	character (32)	권장 사항의 유형입니다. 가능한 값은 `distkey`, `sortkey` 및 `encode`입니다.
status	character (128)	권장 사항의 완료 상태입니다. 가능한 값은 `Start`, `Complete`, `Skipped`, `Abort`, `Checkpoint` , `Failed`입니다.
event_time	타임스탬프	상태 열의 타임스탬프입니다.
alter_from	character (200)	권장 사항을 적용하기 전에 테이블의 이전 배포 스타일과 정렬 키입니다. 값은 200자 단위로 잘립니다.

샘플 쿼리

다음 예에서 결과의 행은 Amazon Redshift에서 수행한 작업을 보여줍니다.

```
SELECT table_id, alter_table_type, status, event_time, alter_from
FROM SYS_AUTO_TABLE_OPTIMIZATION;

 table_id |  alter_table_type   |                        status
 |         event_time          |  alter_from
----------+---------------------+-------------------------------------------------------
+-----------------------------+-----------------
   118082 | sortkey             | Start
 | 2020-08-22 19:42:20.727049  |
   118078 | sortkey             | Start
 | 2020-08-22 19:43:54.728819  |
```

```
  118082 | sortkey             | Start
| 2020-08-22 19:42:52.690264  |
  118072 | sortkey             | Start
| 2020-08-22 19:44:14.793572  |
  118082 | sortkey             | Failed
| 2020-08-22 19:42:20.728917  |
  118078 | sortkey             | Complete
| 2020-08-22 19:43:54.792705  |  SORTKEY: None;
  118086 | sortkey             | Complete
| 2020-08-22 19:42:00.72635   |  SORTKEY: None;
  118082 | sortkey             | Complete
| 2020-08-22 19:43:34.728144  |  SORTKEY: None;
  118072 | sortkey             | Skipped:Retry exceeds the maximum limit for a table.
| 2020-08-22 19:44:46.706155  |
  118086 | sortkey             | Start
| 2020-08-22 19:42:00.685255  |
  118082 | sortkey             | Start
| 2020-08-22 19:43:34.69531   |
  118072 | sortkey             | Start
| 2020-08-22 19:44:46.703331  |
  118082 | sortkey             | Checkpoint: progress 14.755079%
| 2020-08-22 19:42:52.692828  |
  118072 | sortkey             | Failed
| 2020-08-22 19:44:14.796071  |
  116723 | sortkey             | Abort:This table is not AUTO.
| 2020-10-28 05:12:58.479233  |
  110203 | distkey             | Abort:This table is not AUTO.
| 2020-10-28 05:45:54.67259   |
```

SYS_CHILD_QUERY_TEXT

하위 쿼리의 SQL 텍스트를 반환합니다.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	쿼리를 제출한 사용자의 식별자입니다.
query_id	bigint	사용자 쿼리 ID

열 명칭	데이터 유형	설명
child_query_sequence	정수	재작성된 사용자 쿼리의 순서입니다(1로 시작).
SEQUENCE	정수	이 쿼리 조각의 시퀀스 번호입니다.
텍스트	character (200)	SQL 쿼리 텍스트의 첫 200자입니다.

샘플 쿼리

다음 예에서 결과의 행은 Amazon Redshift에서 수행한 작업을 보여줍니다.

```
SELECT * from sys_child_query_text where query_id = '34487366' order by
 child_query_sequence asc, sequence asc;

user_id | query_id | child_query_sequence | sequence | text
--------|----------|----------------------|----------|-----------------------------------------
100     | 34899339 |   1                  |    0     |  /* RQEV2-aY6ZZ1ZpQK */\nwith
 venue as (\n    select venueid,\n            venuename,\n            venuestate\n
 from venue\n), event as (\n    select eventid,\n            venueid,\n            date

100     | 34899339 |   1                  |    1     |  id,\n            eventname\n
  from event\n    where eventname like '3 Doors Down'\n), users as (\n    select userid
\n    from users\n), sales as (\n    select salesid,\n            pricepaid,
100     | 34899339 |   1                  |    2     |  \n            eventid,\n
       buyerid\n    from sales\n)\nselect e.eventname,\n        v.venuename,\n
 count(distinct(u.userid)) as unique_customers,\n        sum(s.pricepaid) as total_sal

100     | 34899339 |   1                  |    3     |  es\nfrom venue as v inner join
 event e on v.venueid = e.venueid\ninner join sales s on e.eventid = s.eventid inner
 join users u on s.buyerid = u.userid\ngroup by 1,2\norder by 4 desc limit 100
```

SYS_CONNECTION_LOG

인증 시도 횟수와 연결 및 차단 정보를 기록합니다.

SYS_CONNECTION_LOG는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
이벤트	character(50)	연결 또는 인증 이벤트
record_time	타임스탬프	이벤트 발생 시간
원격 호스트	character(45)	원격 호스트의 이름 또는 IP 주소
remote_port	character(32)	원격 호스트의 포트 번호
session_id	정수	쿼리 문과 연결된 프로세스 ID
database_name	character(50)	데이터베이스 이름.
user_name	character(50)	사용자 이름
auth_method	character(32)	인증 방법
기간	정수	연결 지속 시간(마이크로초)
ssl_version	character(50)	SSL(Secure Sockets Layer) 버전
ssl_cipher	character(128)	SSL 암호
mtu	정수	최대 전송 단위(MTU)
ssl_compression	character(64)	SSL 압축 유형
ssl_expansion	character(64)	SSL 확장 유형
iam_auth_guid	character(36)	CloudTrail 요청에 대한 IAM 인증 ID입니다.

열 명칭	데이터 유형	설명
application_name	character(250)	세션에서 애플리케이션의 초기 이름 또는 업데이트된 이름입니다.
driver_version	character(64)	서드 파티 SQL 클라이언트 도구에서 Amazon Redshift 클러스터에 연결하는 ODBC 또는 JDBC 드라이버 버전입니다.
os_version	character(64)	Amazon Redshift 클러스터에 연결하는 클라이언트 시스템에 있는 운영 체제의 버전입니다.
plugin_name	character(32)	Amazon Redshift 클러스터에 연결하는 데 사용되는 플러그인의 이름입니다.
protocol_version	정수	Amazon Redshift 드라이버가 서버와의 연결을 설정할 때 사용하는 내부 프로토콜 버전입니다. 프로토콜 버전은 드라이버와 서버 간에 협상됩니다. 버전은 사용 가능한 기능을 설명합니다. 유효한 값으로는 다음이 포함됩니다. • 0(BASE_SERVER_PROTOCOL_VERSION) • 1(EXTENDED_RESULT_METADATA_SERVER_PROTOCOL_VERSION) – 쿼리당 왕복을 저장하기 위해 서버는 추가 결과 집합 메타데이터 정보를 보냅니다. • 2(BINARY_PROTOCOL_VERSION) – 결과 집합의 데이터 유형에 따라 서버는 데이터를 이진 형식으로 보냅니다. • 3(EXTENDED2_RESULT_METADATA_SERVER_PROTOCOL_VERSION) – 서버가 열의 대/소문자 구분(데이터 정렬) 정보를 보냅니다.
global_session_id	character(36)	현재 세션에 대한 전역적으로 고유한 식별자입니다. 세션 ID는 노드 오류가 다시 시작해도 유지됩니다.

샘플 쿼리

열려있는 연결 세부 정보를 보려면 다음과 같이 쿼리를 실행합니다.

```
select record_time, user_name, database_name, remote_host, remote_port
from sys_connection_log
```

```
where event = 'initiating session'
and session_id not in
(select session_id from sys_connection_log
where event = 'disconnecting session')
order by 1 desc;

record_time         | user_name   | database_name   | remote_host   | remote_port

--------------------+-------------+-----------------+---------------
+---------------------------------
2014-11-06 20:30:06 | rdsdb       | dev             | [local]       |

2014-11-06 20:29:37 | test001     | test            | 10.49.42.138  | 11111

2014-11-05 20:30:29 | rdsdb       | dev             | 10.49.42.138  | 33333

2014-11-05 20:28:35 | rdsdb       | dev             | [local]       |
(4 rows)
```

다음은 실패한 인증 시도와 성공한 연결 및 차단을 나타낸 예입니다.

```
select event, record_time, remote_host, user_name
from sys_connection_log order by record_time;

            event      |         record_time        |  remote_host  | user_name

-----------------------+----------------------------+---------------+---------
authentication failure | 2012-10-25 14:41:56.96391  | 10.49.42.138  | john

authenticated          | 2012-10-25 14:42:10.87613  | 10.49.42.138  | john

initiating session     | 2012-10-25 14:42:10.87638  | 10.49.42.138  | john

disconnecting session  | 2012-10-25 14:42:19.95992  | 10.49.42.138  | john

(4 rows)
```

다음 예에서는 ODBC 드라이버의 버전, 클라이언트 시스템의 운영 체제 및 Amazon Redshift 클러스터에 연결하는 데 사용되는 플러그인을 보여줍니다. 이 예에서 사용되는 플러그인은 로그인 이름과 암호를 사용하는 표준 ODBC 드라이버 인증을 위한 것입니다.

```
select driver_version, os_version, plugin_name from sys_connection_log;
```

```
driver_version                          |  os_version                       |
 plugin_name
----------------------------------------+-----------------------------------
+--------------------
Amazon Redshift ODBC Driver 1.4.15.0001 | Darwin 18.7.0 x86_64              | none
Amazon Redshift ODBC Driver 1.4.15.0001 | Linux 4.15.0-101-generic x86_64   | none
```

다음 예에서는 클라이언트 시스템의 운영 체제 버전, 드라이버 버전 및 프로토콜 버전을 보여줍니다.

```
select os_version, driver_version, protocol_version from sys_connection_log;

os_version                      |  driver_version              | protocol_version
--------------------------------+------------------------------+--------------------
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
```

SYS_COPY_JOB

SYS_COPY_JOB을 사용하여 COPY JOB 명령의 세부 정보를 봅니다.

이 보기에는 생성된 COPY JOB 명령이 포함되어 있습니다.

SYS_COPY_JOB은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
job_id	bigint	복사 작업 식별자입니다.
job_name	character(128)	복사 작업의 이름입니다.
iam_role	character(128)	COPY 문에 지정된 IAM 역할입니다.
job_text	character(256)	COPY 문의 파라미터입니다.

열 명칭	데이터 유형	설명
is_auto	정수	COPY JOB이 Amazon Redshift에 의해 자동으로 실행되는지를 나타냅니다. 1은 true를 나타내고 0은 false를 나타냅니다.
on_error_suspend	정수	이 정보는 내부 전용입니다.

SYS_COPY_JOB_DETAIL

SYS_COPY_JOB_DETAIL을 사용하여 COPY JOB 명령의 세부 정보를 봅니다.

이 보기에는 생성된 COPY JOB 명령이 포함되어 있습니다. COPY JOB이 파일을 로드하려고 하는데 로드하지 못하면 향후 자동 COPY JOB 시도에서 해당 파일을 건너뜁니다.

SYS_COPY_JOB_DETAIL은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	작업을 소유한 사용자의 식별자입니다.
database_name	텍스트	작업이 포함된 데이터베이스의 이름입니다.
job_id	정수	복사 작업 식별자입니다.
job_name	텍스트	복사 작업의 이름입니다.
file_location	텍스트	항목의 Amazon S3 버킷 이름입니다.

열 명칭	데이터 유형	설명
file_name	텍스트	항목의 Amazon S3 키 이름입니다.
file_size	bigint	항목의 크기입니다(바이트).
file_etag	텍스트	항목의 Amazon S3 엔터티 태그(ETag)입니다.
job_text	character(256)	COPY 문의 파라미터입니다.
modification_time	타임스탬프	수집을 시작하여 S3 객체가 만들어지거나 수정된 시간입니다(UTC). S3 나열 모드에서는 객체 수정 시간입니다. 알림 모드의 경우 알림 이벤트가 발생한 시간입니다.
enqueue_time	타임스탬프	항목이 수집 대기열에 추가된 시간입니다(UTC).
status	char(1)	항목의 상태입니다. 유효한 값: 보류 중인 수집의 경우 P, 수집된 경우 I, 수집 오류의 경우 E, 알 수 없음의 경우 U(일종의 예상치 못한 상태를 나타냄)입니다.

다음 예시에서는 수집된 항목에 대해 행 하나를 반환합니다.

```
SELECT * FROM SYS_COPY_JOB_DETAIL WHERE status ilike '%ingested%' limit 1;


user_id | 100
database_name | dev
job_name | many_job_4_3
job_id | 110702
file_location | saral-sqs-system4623202051-0
```

```
file_name | frenzy-9/4623202051/file_0_107
file_size | 11302
file_etag | 51b2d78ac5b5aecf4ee6f8374815ad19
modification_time | 2024-07-15 20:43:14
enqueue_time | 2024-07-15 20:44:24
status | Ingested
```

SYS_COPY_JOB_INFO

SYS_COPY_JOB_INFO를 사용하여 COPY JOB에 대해 로깅된 메시지를 봅니다.

이 뷰에는 실행된 COPY JOB의 오류에 대한 정보가 포함되어 있습니다.

SYS_COPY_JOB_INFO는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
job_id	bigint	복사 작업 식별자입니다.
job_name	character(128)	복사 작업의 이름입니다.
database_name	character(128)	데이터베이스의 이름입니다.
record_time	타임스탬프	메시지가 로깅된 시간입니다 (UTC).
message	character(512)	COPY JOB에 대해 로깅된 이벤트의 메시지입니다.

SYS_COPY_REPLACEMENTS

ACCEPTINVCHARS 옵션과 함께 COPY 명령을 실행하여 잘못된 UTF-8 문자를 대체했을 때 기록되는 로그를 표시합니다. 적어도 1개 이상 대체가 필요했던 각 노드 조각에서 첫 번째 100개 행마다 로그 항목이 SYS_COPY_REPLACEMENTS에 추가됩니다.

이 뷰를 사용하여 서버리스 작업 그룹 및 프로비저닝된 클러스터에 대한 정보를 볼 수 있습니다.

SYS_COPY_REPLACEMENTS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	쿼리를 생성한 사용자의 ID입니다.
query_id	bigint	쿼리 ID. 다른 시스템 테이블 및 뷰를 조인하는 데 사용되는 열입니다.
table_id	정수	테이블 ID입니다.
file_name	character (256)	COPY 명령에 대한 입력 파일의 전체 경로입니다.
column_name	character (127)	잘못된 UTF-8 문자가 포함된 첫 번째 필드입니다.
line_number	bigint	입력 데이터 파일에 잘못된 UTF-8 문자가 포함된 줄 번호입니다. -1은 열 형식 데이터 파일에서 복사하는 경우와 같이 줄 번호를 사용할 수 없음을 나타냅니다.
raw_line	character (1024)	잘못된 UTF-8 문자가 포함된 원시 로드 데이터입니다.

샘플 쿼리

다음은 가장 최근 COPY 작업에서 대체된 파일을 반환하는 예입니다.

```
select query_idp, table_id, file_name, line_number, colname
from sys_copy_replacements
where query = pg_last_copy_id();


 query_id | table_id |   file_name                                                |
 line_number | column_name
```

```
 ---------+----------+-------------------------------------------------------
+-------------+--------
    96    |    26    | s3://DOC-EXAMPLE-BUCKET/allusers_pipe.txt             |
 123 | city
    96    |    26    | s3://DOC-EXAMPLE-BUCKET/allusers_pipe.txt             |
 456 | city
    96    |    26    | s3://DOC-EXAMPLE-BUCKET/allusers_pipe.txt             |
 789 | city
    96    |    26    | s3://DOC-EXAMPLE-BUCKET/allusers_pipe.txt             |
 012 | city
    96    |    26    | s3://DOC-EXAMPLE-BUCKET/allusers_pipe.txt             |
 119 | city
...
```

SYS_DATASHARE_CHANGE_LOG

생산자 클러스터와 소비자 클러스터 모두에서 datashare의 변경 내용을 추적하기 위한 통합 뷰를 기록합니다.

SYS_DATASHARE_CHANGE_LOG는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	작업을 수행하는 사용자의 ID입니다.
user_name	varchar(128)	작업을 수행하는 사용자의 이름입니다.
session_id	정수	세션의 ID입니다.
transaction_id	bigint	트랜잭션의 ID입니다.
share_id	정수	영향을 받는 datashare의 ID입니다.

열 명칭	데이터 유형	설명
share_name	varchar(128)	datashare의 이름입니다.
source_database_id	정수	datashare가 속한 데이터베이스의 ID입니다.
source_database_name	varchar(128)	datashare가 속한 데이터베이스의 이름입니다.
consumer_database_id	정수	datashare에서 가져온 데이터베이스의 ID입니다.
consumer_database_name	varchar(128)	datashare에서 가져온 데이터베이스의 이름입니다.
arn	varchar(192)	가져온 데이터베이스를 뒷받침하는 리소스의 ARN입니다.
record_time	타임스탬프	작업의 타임스탬프입니다.
작업	varchar(128)	실행 중인 작업입니다. 가능한 값은 CREATE DATASHARE, DROP DATASHARE, GRANT ALTER, REVOKE ALTER, GRANT SHARE, REVOKE SHARE, ALTER ADD, ALTER REMOVE, ALTER SET, GRANT USAGE, REVOKE USAGE, CREATE DATABASE, GRANT 또는 REVOKE USAGE(공유 데이터베이스 대상), DROP SHARED DATABASE, ALTER SHARED DATABASE입니다.
status	정수	작업의 상태입니다. 가능한 값은 SUCCESS와 ERROR-ERROR CODE입니다.
share_object_type	varchar(64)	datashare에서 추가되거나 제거된 데이터베이스 객체의 유형입니다. 가능한 값은 schema, table, column, function 및 view입니다. 생산자 클러스터에 대한 필드입니다.

열 명칭	데이터 유형	설명
share_object_id	정수	datashare에서 추가되거나 제거된 데이터베이스 객체의 ID입니다. 생산자 클러스터에 대한 필드입니다.
share_object_name	varchar(128)	datashare에서 추가되거나 제거된 데이터베이스 객체의 이름입니다. 생산자 클러스터에 대한 필드입니다.
target_user_type	varchar(16)	권한이 부여된 사용자 또는 그룹의 유형입니다. 생산자 및 소비자 클러스터 모두에 대한 필드입니다.
target_user_id	정수	권한이 부여된 사용자 또는 그룹의 ID입니다. 생산자 및 소비자 클러스터 모두에 대한 필드입니다.
target_user_name	varchar(128)	권한이 부여된 사용자 또는 그룹의 이름입니다. 생산자 및 소비자 클러스터 모두에 대한 필드입니다.
consumer_account	varchar(16)	데이터 소비자의 계정 ID입니다. 생산자 클러스터에 대한 필드입니다.
consumer_namespace	varchar(64)	데이터 소비자 계정의 네임스페이스입니다. 생산자 클러스터에 대한 필드입니다.
producer_account	varchar(16)	datashare가 속한 생산자 계정의 계정 ID입니다. 소비자 클러스터에 대한 필드입니다.
producer_namespace	varchar(64)	datashare가 속한 제품 계정의 네임스페이스입니다. 소비자 클러스터에 대한 필드입니다.
attribute_name	varchar(64)	datashare 또는 공유 데이터베이스의 속성 이름입니다.
attribute_value	varchar(128)	datashare 또는 공유 데이터베이스의 속성 값입니다.
message	varchar(512)	작업이 실패할 경우 오류 메시지입니다.

샘플 쿼리

다음 예에서는 SYS_DATASHARE_CHANGE_LOG 뷰를 보여줍니다.

```
SELECT DISTINCT action
FROM sys_datashare_change_log
WHERE share_object_name LIKE 'tickit%';

         action
 ----------------------
  "ALTER DATASHARE ADD"
```

SYS_DATASHARE_CROSS_REGION_USAGE

SYS_DATASHARE_CROSS_REGION_USAGE 뷰를 사용하면 교차 리전 데이터 공유 쿼리로 인해 발생한 교차 리전 데이터 전송 사용량을 요약하여 확인할 수 있습니다. SYS_DATASHARE_CROSS_REGION_USAGE는 세그먼트 수준에서 세부 정보를 집계합니다.

SYS_DATASHARE_CROSS_REGION_USAGE는 수퍼유저에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
query_id	정수	쿼리의 ID입니다. 이 값을 사용하여 다른 시스템 테이블 및 뷰를 조인합니다.
child_query_sequence	정수	재작성된 사용자 쿼리의 순서입니다(1로 시작).
segment_id	bigint	세그먼트의 번호입니다. 하나의 쿼리는 여러 세그먼트로 구성되며, 각각의 세그먼트는 하나 이상의 단계로 구성됩니다.
start_time	시간	데이터 전송이 시작된 UTC 시간입니다.
end_time	시간	데이터 전송이 종료된 UTC 시간입니다.

열 명칭	데이터 유형	설명
transferred_data	bigint	생산자 리전에서 소비자 리전으로 전송된 데이터의 바이트 수입니다.
source_region	char(25)	쿼리가 데이터를 전송한 생산자 리전입니다.

샘플 쿼리

다음 예에서는 SYS_DATASHARE_CROSS_REGION_USAGE 뷰를 보여줍니다.

```
SELECT query_id, segment_id, transferred_data, source_region
from sys_datashare_cross_region_usage
where query_id = pg_last_query_id()
order by query_id, segment_id;

  query_id | segment_id | transferred_data | source_region
-----------+------------+------------------+---------------
    200048 |          2 |          4194304 |    us-west-1
    200048 |          2 |          4194304 |    us-east-2
```

SYS_DATASHARE_USAGE_CONSUMER

datashare의 활동과 사용량을 기록합니다. 이 뷰는 소비자 클러스터에만 관련이 있습니다.

SYS_DATASHARE_USAGE_CONSUMER는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	요청을 발행하는 사용자의 ID입니다.
session_id	정수	쿼리를 실행하는 리더 프로세스의 ID입니다.

열 명칭	데이터 유형	설명
transaction_id	bigint	현재 트랜잭션의 컨텍스트입니다.
request_id	varchar(50)	요청된 API 호출의 고유 ID입니다.
request_type	varchar(25)	생산자 클러스터에 대한 요청 형식입니다.
transaction_uid	varchar(50)	트랜잭션의 고유 ID입니다.
record_time	타임스탬프	작업이 기록되는 시간입니다.
status	정수	요청된 API 호출의 상태입니다.
error_message	varchar(512)	오류에 대한 메시지입니다.

샘플 쿼리

다음 예에서는 SYS_DATASHARE_USAGE_CONSUMER 뷰를 보여줍니다.

```
SELECT request_type, status, trim(error) AS error
FROM sys_datashare_usage_consumer

  request_type  | status | error_message
----------------+--------+---------------
 "GET RELATION" |   0    |
```

SYS_DATASHARE_USAGE_PRODUCER

datashare의 활동과 사용량을 기록합니다. 이 뷰는 생산자 클러스터에만 관련이 있습니다.

SYS_DATASHARE_USAGE_PRODUCER는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
share_id	정수	datashare의 객체 ID(OID)입니다.
share_name	varchar(128)	datashare의 이름입니다.
request_id	varchar(50)	요청된 API 호출의 고유 ID입니다.
request_type	varchar(25)	생산자 클러스터에 대한 요청 형식입니다.
object_type	varchar(64)	datashare에서 공유되는 객체의 형식입니다. 가능한 값은 schema, table, column, function 및 view입니다.
object_oid	정수	datashare에서 공유되는 객체의 ID입니다.
객체 이름	varchar(128)	datashare에서 공유되는 객체의 이름입니다.
consumer_account	varchar(16)	datashare가 공유되는 소비자 계정의 계정입니다.
consumer_namespace	varchar(64)	datashare가 공유되는 소비자 계정의 네임스페이스입니다.
consumer_transaction_uid	varchar(50)	소비자 클러스터에 있는 문의 고유 트랜잭션 ID입니다.
record_time	타임스탬프	작업이 기록되는 시간입니다.

열 명칭	데이터 유형	설명
status	정수	datashare의 상태입니다.
error_message	varchar(512)	오류에 대한 메시지입니다.
consumer_region	char(64)	소비자 클러스터가 속한 리전입니다.

샘플 쿼리

다음 예에서는 SYS_DATASHARE_USAGE_PRODUCER 뷰를 보여줍니다.

```
SELECT DISTINCT
FROM sys_datashare_usage_producer
WHERE object_name LIKE 'tickit%';

   request_type
 ------------------
   "GET RELATION"
```

SYS_EXTERNAL_QUERY_DETAIL

SYS_EXTERNAL_QUERY_DETAIL로 세그먼트 수준에서 쿼리에 대한 세부 정보를 봅니다. 각 행은 처리된 행 수, 처리된 바이트 수, Amazon S3에 있는 외부 테이블의 파티션 정보와 같은 세부 정보가 포함된 특정 WLM 쿼리의 세그먼트를 나타냅니다. 이 보기의 각 행은 SYS_QUERY_DETAIL 보기에도 해당 항목이 있습니다. 단, 이 보기에는 외부 쿼리 처리와 관련된 세부 정보가 있습니다.

SYS_EXTERNAL_QUERY_DETAIL은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	쿼리를 제출한 사용자의 식별자입니다.
query_id	bigint	외부 쿼리의 쿼리 식별자입니다.
transaction_id	bigint	트랜잭션 식별자입니다.
child_query_sequence	정수	재작성된 사용자 쿼리의 순서입니다. segment_id와 유사하게 0으로 시작합니다.
segment_id	정수	쿼리 세그먼트의 세그먼트 식별자입니다.
source_type	character(32)	쿼리의 데이터 원본 유형은 Redshift Spectrum의 경우 S3, 연합 쿼리의 경우 PG일 수 있습니다.
start_time	타임스탬프	쿼리가 시작된 시간입니다.
end_time	타임스탬프	쿼리가 완료된 시간입니다.
duration	bigint	쿼리에 소요된 시간(마이크로초)입니다.
total_partitions	정수	Amazon S3 쿼리에 필요한 파티션 수입니다.
qualified_partitions	정수	Amazon S3 쿼리가 스캔한 파티션 수입니다.
scanned_files	bigint	스캔한 Amazon S3 파일 수입니다.

열 명칭	데이터 유형	설명
returned_rows	bigint	Amazon S3 쿼리에 대해 스캔된 행 수 또는 연합 쿼리에 대해 반환된 행 수입니다.
returned_bytes	bigint	Amazon S3 쿼리에 대해 스캔된 바이트 수 또는 연합 쿼리에 대해 반환된 바이트 수입니다.
file_format	텍스트	Amazon S3 파일의 파일 형식입니다.
file_location	텍스트	외부 테이블의 Amazon S3 위치입니다.
external_query_text	텍스트	연합 쿼리에 대한 세그먼트 수준 쿼리 텍스트입니다.
warning_message	character(4000)	쿼리가 실행될 때 표시되는 경고 메시지입니다.
table_name	character(136)	작동 중인 단계의 테이블 이름입니다.
is_recursive	character(1)	하위 폴더에 대한 반복 스캔이 있는지를 나타냅니다.
is_nested	character(1)	중첩된 열 데이터 유형에 액세스할지를 나타냅니다.
s3list_time	bigint	파일 목록의 기간(밀리초)입니다.
get_partition_time	long	AWS Glue Data Catalog 및 Apache Hive에서 지정된 외부 객체의 파티션을 나열하고 자격을 부여하는 데 소요된 시간입니다.

샘플 쿼리

다음 쿼리는 외부 쿼리 세부 정보를 보여줍니다.

```
SELECT query_id,
       segment_id,
       start_time,
       end_time,
       total_partitions,
       qualified_partitions,
       scanned_files,
       returned_rows,
       returned_bytes,
       trim(external_query_text) query_text,
       trim(file_location) file_location
FROM sys_external_query_detail
ORDER BY query_id, start_time DESC
LIMIT 2;
```

샘플 출력은 다음과 같습니다.

```
 query_id | segment_id |         start_time         |          end_time
   | total_partitions | qualified_partitions | scanned_files | returned_rows |
 returned_bytes | query_text | file_location
----------+------------+----------------------------+----------------------------
+------------------+----------------------+---------------+---------------
+----------------+------------+---------------
   763251 |          0 | 2022-02-15 22:32:23.312448 | 2022-02-15 22:32:24.036023 |
          3 |                    3 |             3 |         38203 |        2683414 |
          |
   763254 |          0 | 2022-02-15 22:32:40.17103  | 2022-02-15 22:32:40.839313 |
          3 |                    3 |             3 |         38203 |        2683414 |
          |
```

SYS_EXTERNAL_QUERY_ERROR

시스템 뷰 SYS_EXTERNAL_QUERY_ERROR를 쿼리하여 적색편이 스펙트럼 스캔 오류에 대한 정보를 얻을 수 있습니다. SYS_EXTERNAL_QUERY_ERROR는 기록된 오류의 샘플을 표시합니다. 기본값은 쿼리당 10개 항목입니다.

SYS_EXTERNAL_QUERY_ERROR는 모든 사용자가 볼 수 있습니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	이 행을 생성한 사용자의 ID입니다.
query_id	bigint	이 행을 생성한 쿼리의 ID입니다.
file_location	char(256)	쿼리되는 데이터의 위치입니다.
rowid	char(2100)	파일의 오류 위치입니다. `rowid` 부분은 `:`(콜론)으로 구분되며 기타 부분은 향후에 추가할 수 있습니다. `row_offset :row_group :row_id` row_offset은 파일 내 행의 오프셋(바이트)이며 지원되지 않는 파일 형식의 경우 `-1`입니다. 테이블은 row_groups로 구분되며 각 그룹에는 고유한 row_ids를 가진 행이 있습니다.
column_name	char(127)	쿼리에서 반환되는 열의 이름입니다.
original_value	char(1024)	쿼리되는 원래 값입니다.
modified_value	char(1024)	쿼리에 지정된 데이터 처리 구성 옵션에 따라 반환되는 수정된 값입니다.
트리거	char(128)	쿼리에 지정된 데이터 처리 옵션입니다.
작업	char(128)	쿼리에 지정된 데이터 처리 옵션과 연결된 작업입니다.
action_value	char(128)	쿼리에 지정된 데이터 처리 옵션과 연결된 작업 파라미터의 값입니다.

열 명칭	데이터 유형	설명
error_code	정수	쿼리에 지정된 데이터 처리 옵션의 결과 코드입니다.

샘플 쿼리

다음 쿼리에서는 데이터 처리 작업이 수행된 행의 목록을 반환합니다.

```
SELECT * FROM sys_external_query_error;
```

쿼리는 다음과 비슷한 결과를 반환합니다.

```
  user_id   query_id  file_location                                rowid
column_name           original_value             modified_value       trigger
action               action_value                 error_code
    100     1574007   s3://spectrum-uddh/league/spi_global_rankings.0:0
league_name           Barclays Premier League    Barclays Premier Lea UNSPECIFIED
TRUNCATE                                          156
    100     1574007   s3://spectrum-uddh/league/spi_global_rankings.0:0
league_nspi           34595                      32767                UNSPECIFIED
OVERFLOW_VALUE                                    199
    100     1574007   s3://spectrum-uddh/league/spi_global_rankings.0:1
league_nspi           34151                      32767                UNSPECIFIED
OVERFLOW_VALUE                                    199
    100     1574007   s3://spectrum-uddh/league/spi_global_rankings.0:2
league_name           Barclays Premier League    Barclays Premier Lea UNSPECIFIED
TRUNCATE                                          156
    100     1574007   s3://spectrum-uddh/league/spi_global_rankings.0:2
league_nspi           33223                      32767                UNSPECIFIED
OVERFLOW_VALUE                                    199
    100     1574007   s3://spectrum-uddh/league/spi_global_rankings.0:3
league_name           Barclays Premier League    Barclays Premier Lea UNSPECIFIED
TRUNCATE                                          156
    100     1574007   s3://spectrum-uddh/league/spi_global_rankings.0:3
league_nspi           32808                      32767                UNSPECIFIED
OVERFLOW_VALUE                                    199
    100     1574007   s3://spectrum-uddh/league/spi_global_rankings.0:4
league_nspi           32790                      32767                UNSPECIFIED
OVERFLOW_VALUE                                    199
```

```
    100     1574007   s3://spectrum-uddh/league/spi_global_rankings.0:5
league_name           Spanish Primera Division   Spanish Primera Divi UNSPECIFIED
TRUNCATE                                          156
    100     1574007   s3://spectrum-uddh/league/spi_global_rankings.0:6
league_name           Spanish Primera Division   Spanish Primera Divi UNSPECIFIED
TRUNCATE                                          156
```

SYS_INTEGRATION_ACTIVITY

SYS_INTEGRATION_ACTIVITY는 완료된 통합 실행에 대한 세부 정보를 표시합니다.

SYS_INTEGRATION_ACTIVITY는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

제로 ETL 통합에 대한 자세한 내용은 Amazon Redshift 관리 안내서의 제로 ETL 통합 작업을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
integration_id	character (128)	통합과 관련된 식별자입니다.
target_database	character (128)	통합 데이터를 수신하는 Amazon Redshift의 데이터베이스입니다.
source	character (128)	통합을 위한 소스 데이터입니다. 가능한 유형에는 MySQL 및 PostgreSQL 이 포함됩니다.
checkpoint_name	character (128)	binlog 좌표를 복제하는 체크포인트 이름입니다.
checkpoint_type	character (16)	체크포인트 유형입니다. 가능한 값은 snapshot, cdc가 포함되어 있습니다.
checkpoint_bytes	bigint	이 체크포인트의 바이트 수입니다.
last_commit_timestamp	타임스탬프	이 체크포인트에서 마지막으로 커밋된 타임스탬프입니다.

열 명칭	데이터 유형	설명
수정된 테이블	정수	체크포인트에서 수정된 테이블 수입니다.
integration_start_time	시간	이 체크포인트에 대한 통합이 시작된 시간(UTC)입니다.
integration_end_time	시간	이 체크포인트에 대한 통합이 종료된 시간(UTC)입니다.

샘플 쿼리

다음 SQL 명령은 통합 로그를 표시합니다.

```
select * from sys_integration_activity;

          integration_id              | target_database | source |
   checkpoint_name                  | checkpoint_type  | checkpoint_bytes |
 last_commit_timestamp   | modified_tables |   integration_start_time   |
 integration_end_time
-------------------------------------+-----------------+--------
+--------------------------------------------+------------------+------------------
+-------------------------+-----------------+----------------------------
+----------------------------
 76b15917-afae-4447-b7fd-08e2a5acce7b |   demo1         | MySQL  | checkpoints/
checkpoint_3_241_3_510.json     |        cdc       |        762       | 2023-05-10
 23:00:14.201 |         1       | 2023-05-10 23:00:45.054265 | 2023-05-10
 23:00:46.339826
 76b15917-afae-4447-b7fd-08e2a5acce7b |   demo1         | MySQL  | checkpoints/
checkpoint_3_16329_3_17839.json |        cdc       |       13488      | 2023-05-11
 01:33:57.411 |         2       | 2023-05-11 02:19:09.440121 | 2023-05-11
 02:19:16.090492
 76b15917-afae-4447-b7fd-08e2a5acce7b |   demo1         | MySQL  | checkpoints/
checkpoint_3_5103_3_5532.json   |        cdc       |        1657      | 2023-05-10
 23:13:14.205 |         2       | 2023-05-10 23:13:23.545487 | 2023-05-10
 23:13:25.652144
```

SYS_INTEGRATION_TABLE_ACTIVITY

SYS_INTEGRATION_TABLE_ACTIVITY는 제로 ETL 통합의 삽입, 삭제 및 업데이트 활동에 대한 세부 정보를 표시합니다. 완료된 수집마다 행이 하나씩 추가됩니다.

슈퍼 사용자는 이 테이블의 모든 행을 볼 수 있습니다.

자세한 내용은 제로 ETL 통합을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
integration_id	character (128)	통합과 관련된 식별자입니다.
checkpoint_name	character (128)	체크포인트의 이름입니다.
target_database	character (128)	Amazon Redshift 데이터베이스의 이름입니다.
schema_name	character (128)	Amazon Redshift 스키마의 이름입니다.
table_name	character (128)	테이블의 이름
table_id	정수	테이블의 식별자입니다.
record_time	타임스탬프	이 변경이 완료된 시간입니다(UTC).
transaction_id	bigint	트랜잭션 식별자입니다.
inserted_rows	bigint	수집에 의해 삽입된 행 수입니다.
deleted_rows	bigint	수집에서 삭제한 행 수입니다.
updated_rows	bigint	수집에 의해 업데이트된 행 수입니다.
바이트_수집	bigint	수집된 바이트 수입니다.

샘플 쿼리

다음 SQL 명령은 통합의 활동을 표시합니다.

```
select * from sys_integration_table_activity;

          integration_id              | checkpoint_name | target_database | schema_name
 |     table_name    | table_id     | record_time                | transaction_id  |
 inserted_rows  | deleted_rows | updated_rows | bytes_ingested
-------------------------------------+-----------------+-----------------
+------------+------------------+-------------+--------------------------
+-----------------+----------------+-------------+-------------+---------------
 4798e675-8f9f-4686-b05f-92c538e19629 |                 | sample_test2    |    sample
 | SampleTestChannel |  111276      | 2023-05-12 12:40:30.656625 | 7736            |  2
             | 0            | 0            | 125
```

SYS_INTEGRATION_TABLE_STATE_CHANGE

SYS_INTEGRATION_TABLE_STATE_CHANGE는 통합에 대한 테이블 상태 변경 로그를 자세히 표시합니다.

슈퍼 사용자는 이 테이블의 모든 행을 볼 수 있습니다.

자세한 내용은 제로 ETL 통합 작업을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
integration_id	character (128)	통합과 관련된 식별자입니다.
database_name	character (128)	Amazon Redshift 데이터베이스의 이름입니다.
schema_name	character (128)	Amazon Redshift 스키마의 이름입니다.
table_name	character (128)	테이블의 이름
new_state	character (128)	테이블 상태입니다. 가능한 값은 `Synced`, `ResyncRequired` , `ResyncInitiated` , `Deleted`, `Failed`, `ResyncDeleted` 입니다.

열 명칭	데이터 유형	설명
table_last_replicated_checkpoint	character (128)	현재 동기화된 로그 좌표입니다.
state_change_reason	character (256)	마지막 상태 전환의 이유입니다.
record_time	타임스탬프	기록이 업데이트된 시각(UTC)입니다.

샘플 쿼리

다음 SQL 명령은 통합 로그를 표시합니다.

```
select * from sys_integration_table_state_change;

            integration_id            | database_name | schema_name | table_name
 | new_state |  table_last_replicated_checkpoint   | state_change_reason |
 record_time
--------------------------------------+---------------+-------------+------------
+-----------+--------------------------------------+---------------------
+----------------------------
 99108e72-1cfd-414f-8cc0-0216acefac77 | perfdb        | sbtest80t3s | sbtest79   |
 Synced    | {"txn_seq":9834,"txn_id":126597515} |                     | 2023-09-20
 19:39:50.087868
 99108e72-1cfd-414f-8cc0-0216acefac77 | perfdb        | sbtest80t3s | sbtest56   |
 Synced    | {"txn_seq":9834,"txn_id":126597515} |                     | 2023-09-20
 19:39:45.54005
 99108e72-1cfd-414f-8cc0-0216acefac77 | perfdb        | sbtest80t3s | sbtest50   |
 Synced    | {"txn_seq":9834,"txn_id":126597515} |                     | 2023-09-20
 19:40:20.362504
 99108e72-1cfd-414f-8cc0-0216acefac77 | perfdb        | sbtest80t3s | sbtest18   |
 Synced    | {"txn_seq":9834,"txn_id":126597515} |                     | 2023-09-20
 19:40:32.544084
 99108e72-1cfd-414f-8cc0-0216acefac77 | perfdb        | sbtest40t3s | sbtest23   |
 Synced    | {"txn_seq":9834,"txn_id":126597515} |                     | 2023-09-20
 15:49:05.186209
```

SYS_LOAD_DETAIL

데이터 로드를 추적하거나 문제를 해결하기 위한 정보를 반환합니다.

이 뷰는 데이터베이스 테이블에 로드되는 각 데이터 파일의 진행 상황을 기록합니다.

이 뷰는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	항목을 생성한 사용자의 ID.
query_id	정수	쿼리 ID.
file_name	character(256)	로드할 파일 이름입니다.
bytes_scanned	정수	Amazon S3의 파일에서 스캔한 바이트 수입니다.
lines_scanned	정수	로드 파일에서 스캔되는 라인 수. 이 라인 수는 실제로 로드되는 행의 수와 다를 수도 있습니다. 예를 들어 로드 작업은 COPY 명령의 MAXERROR 옵션에 따라 스캔 도중에도 다수의 불량 레코드를 허용할 수도 있기 때문입니다.
record_time	타임스탬프	이 항목이 마지막으로 업데이트된 시간
splits_scanned	이 파일의 분할 수입니다.	이 파일의 분할 수입니다.
start_time	타임스탬프	이 파일 처리가 시작된 시간입니다.
end_time	타임스탬프	이 파일 처리가 종료된 시간입니다.

샘플 쿼리

다음은 마지막 COPY 작업의 세부 정보를 반환하는 예입니다.

```
select query_id, trim(file_name) as file, record_time
from sys_load_detail
where query_id = pg_last_copy_id();
```

```
 query_id |               file                |          record_time
----------+-----------------------------------+----------------------------
 28554    | s3://dw-tickit/category_pipe.txt | 2013-11-01 17:14:52.648486
(1 row)
```

다음은 TICKIT 데이터베이스에 새로운 테이블 로드 항목을 저장하는 쿼리입니다.

```
select query_id, trim(file_name), record_time
from sys_load_detail
where file_name like '%tickit%' order by query_id;

 query_id |           btrim           |          record_time
----------+---------------------------+----------------------------
 22475    | tickit/allusers_pipe.txt | 2013-02-08 20:58:23.274186
 22478    | tickit/venue_pipe.txt    | 2013-02-08 20:58:25.070604
 22480    | tickit/category_pipe.txt | 2013-02-08 20:58:27.333472
 22482    | tickit/date2008_pipe.txt | 2013-02-08 20:58:28.608305
 22485    | tickit/allevents_pipe.txt| 2013-02-08 20:58:29.99489
 22487    | tickit/listings_pipe.txt | 2013-02-08 20:58:37.632939
 22593    | tickit/allusers_pipe.txt | 2013-02-08 21:04:08.400491
 22596    | tickit/venue_pipe.txt    | 2013-02-08 21:04:10.056055
 22598    | tickit/category_pipe.txt | 2013-02-08 21:04:11.465049
 22600    | tickit/date2008_pipe.txt | 2013-02-08 21:04:12.461502
 22603    | tickit/allevents_pipe.txt| 2013-02-08 21:04:14.785124
 22605    | tickit/listings_pipe.txt | 2013-02-08 21:04:20.170594

(12 rows)
```

레코드가 이 시스템 뷰의 로그 파일에 작성된다고 해서 저장 트랜잭션(containing transaction)에서 로드가 성공적으로 커밋되었다는 것을 의미하지는 않습니다. 로드 커밋 여부를 확인하려면 STL_UTILITYTEXT 뷰에 대한 쿼리를 실행하여 COPY 트랜잭션과 일치하는 COMMIT 레코드를 찾습니다. 예를 들어 다음 쿼리는 STL_UTILITYTEXT에 대한 하위 쿼리를 기준으로 SYS_LOAD_DETAIL 테이블과 STL_QUERY 테이블을 조인합니다.

```
select l.query_id,rtrim(l.file_name),q.xid
from sys_load_detail l, stl_query q
where l.query_id=q.query
and exists
(select xid from stl_utilitytext where xid=q.xid and rtrim("text")='COMMIT');

 query_id |           rtrim           |  xid
----------+---------------------------+-------
```

```
 22600    | tickit/date2008_pipe.txt  | 68311
 22480    | tickit/category_pipe.txt  | 68066
  7508    | allusers_pipe.txt         | 23365
  7552    | category_pipe.txt         | 23415
  7576    | allevents_pipe.txt        | 23429
  7516    | venue_pipe.txt            | 23390
  7604    | listings_pipe.txt         | 23445
 22596    | tickit/venue_pipe.txt     | 68309
 22605    | tickit/listings_pipe.txt  | 68316
 22593    | tickit/allusers_pipe.txt  | 68305
 22485    | tickit/allevents_pipe.txt | 68071
  7561    | allevents_pipe.txt        | 23429
  7541    | category_pipe.txt         | 23415
  7558    | date2008_pipe.txt         | 23428
 22478    | tickit/venue_pipe.txt     | 68065
   526    | date2008_pipe.txt         |  2572
  7466    | allusers_pipe.txt         | 23365
 22482    | tickit/date2008_pipe.txt  | 68067
 22598    | tickit/category_pipe.txt  | 68310
 22603    | tickit/allevents_pipe.txt | 68315
 22475    | tickit/allusers_pipe.txt  | 68061
   547    | date2008_pipe.txt         |  2572
 22487    | tickit/listings_pipe.txt  | 68072
  7531    | venue_pipe.txt            | 23390
  7583    | listings_pipe.txt         | 23445
(25 rows)
```

SYS_LOAD_ERROR_DETAIL

SYS_LOAD_ERROR_DETAIL을 사용하여 COPY 명령 오류의 세부 정보를 봅니다. 각 행은 COPY 명령을 나타냅니다. 여기에는 실행 중인 COPY 명령과 완료된 COPY 명령이 모두 포함됩니다.

SYS_LOAD_ERROR_DETAIL은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	사본을 제출한 사용자의 식별자입니다.

열 명칭	데이터 유형	설명
query_id	bigint	사본의 쿼리 식별자입니다.
transaction_id	bigint	트랜잭션 식별자입니다.
session_id	정수	사본을 실행하는 프로세스의 프로세스 식별자입니다.
database_name	character(64)	복사가 실행되었을 때 사용자가 연결된 데이터베이스의 이름입니다.
table_id	정수	테이블 식별자입니다.
start_time	타임스탬프	복사가 시작된 시간(UTC)입니다.
file_name	character(256)	로드할 입력 파일의 전체 경로입니다.
line_number	bigint	오류가 있는 로드 파일의 줄 번호입니다. JSON 파일을 로드할 때 오류가 발생한 JSON 객체의 마지막 라인 번호입니다.
column_name	character(127)	오류가 발생한 필드입니다.
column_type	character(10)	오류가 있는 필드의 데이터 유형입니다.
column_length	character(10)	열 길이(해당되는 경우)입니다. 이 필드는 데이터 형식에 길이 제한이 있을 때 채워집니다. 예를 들어 열의 데이터 유형이 "character(3)"인 경우 이 열에 값 "3"이 저장됩니다.
position	정수	필드의 오류 위치입니다.

열 명칭	데이터 유형	설명
error_code	정수	오류 코드입니다.
error_message	character(512)	오류에 대한 설명입니다.

샘플 쿼리

다음 쿼리는 특정 쿼리에 대한 copy 명령의 로드 오류 세부 정보를 보여줍니다.

```
SELECT query_id,
       table_id,
       start_time,
       trim(file_name) AS file_name,
       trim(column_name) AS column_name,
       trim(column_type) AS column_type,
       trim(error_message) AS error_message
FROM sys_load_error_detail
WHERE query_id = 762949
ORDER BY start_time
LIMIT 10;
```

샘플 출력은 다음과 같습니다.

```
 query_id | table_id |         start_time         |               file_name
      | column_name | column_type |                 error_message
----------+----------+----------------------------
+----------------------------------------+-------------+-------------
+-----------------------------------------------
   762949 |   137885 | 2022-02-15 22:14:46.759151 | s3://load-test/copyfail/
wrong_format_000 | id          | int4        | Invalid digit, Value 'a', Pos 0, Type:
 Integer
   762949 |   137885 | 2022-02-15 22:14:46.759151 | s3://load-test/copyfail/
wrong_format_001 | id          | int4        | Invalid digit, Value 'a', Pos 0, Type:
 Integer
```

SYS_LOAD_HISTORY

SYS_LOAD_HISTORY를 사용하여 COPY 명령의 세부 정보를 봅니다. 각 행은 일부 필드에 대한 누적 통계가 있는 COPY 명령을 나타냅니다. 여기에는 실행 중인 COPY 명령과 완료된 COPY 명령이 모두 포함됩니다.

SYS_LOAD_HISTORY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	사본을 제출한 사용자의 식별자입니다.
query_id	bigint	사본의 쿼리 식별자입니다.
transaction_id	bigint	트랜잭션 식별자입니다.
session_id	정수	사본을 실행하는 프로세스의 프로세스 식별자입니다.
database_name	텍스트	작업이 실행되었을 때 사용자가 연결된 데이터베이스의 이름입니다.
status	텍스트	복사본의 상태입니다. 유효한 값은 `running`, `completed` , `aborted`입니다.
table_name	텍스트	복사할 테이블의 이름입니다.
start_time	타임스탬프	복사가 시작된 시간입니다.
end_time	타임스탬프	복사가 완료된 시간입니다.
duration	bigint	COPY 명령에 소요된 시간(마이크로초)입니다.

열 명칭	데이터 유형	설명
data_source	텍스트	복사할 파일 입력의 Amazon S3 위치입니다.
file_format	텍스트	소스 파일 형식입니다. 형식으로는 csv, txt, json, avro, orc, parquet 등이 있습니다.
loaded_rows	bigint	테이블에 복사된 행 수입니다.
loaded_bytes	bigint	테이블에 복사된 바이트 수입니다.
source_file_count	정수	소스 파일의 파일 수입니다.
source_file_bytes	bigint	소스 파일의 바이트 수입니다.
file_count_scanned	정수	Amazon S3에서 스캔한 파일 수입니다.
file_bytes_scanned	bigint	Amazon S3의 파일에서 스캔한 바이트 수입니다.
error_count	bigint	오류 수입니다.
copy_job_id	bigint	복사 작업 식별자입니다. 0은 작업 식별자가 없음을 나타냅니다.

샘플 쿼리

다음 쿼리는 특정 copy 명령의 로드된 행, 바이트 수, 테이블 및 데이터 소스를 보여줍니다.

```
SELECT query_id,
       table_name,
       data_source,
       loaded_rows,
       loaded_bytes
FROM sys_load_history
```

```
WHERE query_id IN (6389,490791,441663,74374,72297)
ORDER BY query_id,
         data_source DESC;
```

샘플 출력은 다음과 같습니다.

```
 query_id |    table_name    |                                data_source
              | loaded_rows | loaded_bytes
----------+------------------
+-------------------------------------------------------------------------+-------------
+---------------
     6389 | store_returns    | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/
store_returns/    |   287999764 | 1196240296158
    72297 | web_site         | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/
web_site/         |          54 |         43808
    74374 | ship_mode        | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/
ship_mode/        |          20 |          1320
   441663 | income_band      | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/
income_band/      |          20 |          2152
   490791 | customer_address | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/
customer_address/ |     6000000 |     722924305
```

다음 쿼리는 copy 명령의 로드된 행, 바이트 수, 테이블 및 데이터 소스를 보여줍니다.

```
SELECT query_id,
       table_name,
       data_source,
       loaded_rows,
       loaded_bytes
FROM sys_load_history
ORDER BY query_id DESC
LIMIT 10;
```

샘플 출력은 다음과 같습니다.

```
 query_id |       table_name       |                                  data_source
                          | loaded_rows |  loaded_bytes
----------+------------------------
+-------------------------------------------------------------------------
+-------------+-----------------
   491058 | web_site               | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/web_site/               |          54 |           43808
```

```
   490947 | web_sales              | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/web_sales/             |   720000376 |  22971988122819
   490923 | web_returns            | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/web_returns/           |    71997522 |     96597496325
   490918 | web_page               | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/web_page/              |        3000 |            1320
   490907 | warehouse              | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/warehouse/             |          20 |            1320
   490902 | time_dim               | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/time_dim/              |       86400 |            1320
   490876 | store_sales            | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/store_sales/           |  2879987999 | 151666241887933
   490870 | store_returns          | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/store_returns/         |   287999764 |   1196405607941
   490865 | store                  | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/store/                 |        1002 |          365507
```

다음 쿼리는 copy 명령의 일별 로드된 행 및 바이트 수를 보여줍니다.

```
SELECT date_trunc('day',start_time) AS exec_day,
       SUM(loaded_rows) AS loaded_rows,
       SUM(loaded_bytes) AS loaded_bytes
FROM sys_load_history
GROUP BY exec_day
ORDER BY exec_day DESC;
```

샘플 출력은 다음과 같습니다.

```
      exec_day       | loaded_rows |   loaded_bytes
---------------------+-------------+------------------
 2022-01-20 00:00:00 |  6347386005 |  258329473070606
 2022-01-19 00:00:00 | 19042158015 |  775198502204572
 2022-01-18 00:00:00 | 38084316030 | 1550294469446883
 2022-01-17 00:00:00 | 25389544020 | 1033271084791724
 2022-01-16 00:00:00 | 19042158015 |  775222736252792
 2022-01-15 00:00:00 | 19834245387 |  798122849155598
 2022-01-14 00:00:00 | 75376544688 | 3077040926571384
```

SYS_MV_REFRESH_HISTORY

결과에는 모든 구체화된 뷰의 새로 고침 기록에 대한 정보가 포함됩니다. 결과에는 수동 또는 자동과 같은 새로 고침 유형과 가장 최근의 새로 고침 상태가 포함됩니다.

SYS_MV_REFRESH_HISTORY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	새로 고침을 제출한 사용자의 식별자입니다.
session_id	정수	구체화된 뷰 새로 고침을 실행하는 프로세스의 프로세스 식별자입니다.
transaction_id	bigint	트랜잭션 식별자입니다.
database_name	char(128)	구체화된 보기를 포함하는 데이터베이스입니다.
schema_name	char(128)	구체화된 보기의 스키마입니다.
mv_id	bigint	구체화된 뷰의 객체 ID입니다.
mv_name	char(128)	구체화된 보기 이름입니다.
refresh_type	char(32)	새로 고침 유형(예: 수동 또는 자동)
status	텍스트	새로 고침의 상태입니다. 상태에 대한 자세한 내용은 SVL_MV_REFRESH_STATUS의 상태 열을 참조하세요.
start_time	타임스탬프	새로 고침의 시작 시간입니다.
end_time	타임스탬프	새로 고침의 종료 시간입니다.

열 명칭	데이터 유형	설명
duration	bigint	구체화된 뷰를 새로 고치는 데 걸린 시간(마이크로초)입니다.

샘플 쿼리

다음 쿼리는 구체화된 뷰에 대한 새로 고침 기록을 보여줍니다.

```
SELECT user_id,
     session_id,
     transaction_id,
     database_name,
     schema_name,
     mv_id,
     mv_name,
     refresh_type,
     status,
     start_time,
     end_time,
     duration
     from sys_mv_refresh_history;
```

이 쿼리는 다음과 같은 샘플 출력을 반환합니다.

```
 user_id | session_id | transaction_id | database_name | schema_name                |
 mv_id  |  mv_name           |  refresh_type  |  status
         |  start_time                |  end_time                  |  duration
---------+------------+----------------+--------------
+----------------------------+--------+--------------------+----------------
+-------------------------------------------------+---------------------------
+----------------------------+-----------
       1 | 1073815659 |          15066 | dev           | test_stl_mv_refresh_schema |
 203762 | mv_incremental     | Manual         | MV was already updated
         | 2023-10-26 15:59:20.952179 | 2023-10-26 15:59:20.952866 |      687
       1 | 1073815659 |          15068 | dev           | test_stl_mv_refresh_schema |
 203771 | mv_nonincremental  | Manual         | MV was already updated
         | 2023-10-26 15:59:21.008049 | 2023-10-26 15:59:21.008658 |      609
       1 | 1073815659 |          15070 | dev           | test_stl_mv_refresh_schema |
 203779 | mv_refresh_error   | Manual         | MV was already updated
         | 2023-10-26 15:59:21.064252 | 2023-10-26 15:59:21.064885 |      633
```

```
      1 | 1073815659 |          15074 | dev           | test_stl_mv_refresh_schema
 | 203762 | mv_incremental     | Manual         | Refresh successfully updated MV
 incrementally   | 2023-10-26 15:59:29.693329 | 2023-10-26 15:59:43.482842 | 13789513
      1 | 1073815659 |          15076 | dev           | test_stl_mv_refresh_schema |
 203771 | mv_nonincremental  | Manual         | Refresh successfully recomputed MV from
 scratch | 2023-10-26 15:59:43.550184 | 2023-10-26 15:59:47.880833 |  4330649
      1 | 1073815659 |          15078 | dev           | test_stl_mv_refresh_schema |
 203779 | mv_refresh_error   | Manual         | Refresh failed due to an internal error
        | 2023-10-26 15:59:47.949052 | 2023-10-26 15:59:52.494681 |  4545629
(6 rows)
```

SYS_MV_STATE

결과에는 모든 구체화된 뷰의 상태에 대한 정보가 포함됩니다. 여기에는 기본 테이블 정보, 스키마 속성 및 열 삭제와 같은 최근 이벤트에 대한 정보가 포함됩니다.

SYS_MV_STATE는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	bigint	이벤트를 만든 사용자의 ID입니다.
transaction_id	bigint	이벤트의 트랜잭션 ID입니다.
database_name	char(128)	구체화된 보기를 포함하는 데이터베이스입니다.
event_desc	char(500)	상태 변경을 유발한 이벤트입니다. 값 예로는 다음이 포함됩니다. • 열 유형이 변경되었습니다. • 열이 삭제되었습니다. • 열 이름이 바뀌었습니다.

열 명칭	데이터 유형	설명
		• 스키마 이름이 변경되었습니다. • 작은 테이블 변환 • TRUNCATE • Vacuum 참고로 이 열은 다른 값도 가질 수 있습니다.
start_time	타임스탬프	이벤트의 시작 시간입니다.
base_table_database_name	char(128)	기본 테이블의 데이터베이스 이름입니다.
base_table_schema	char(128)	기본 테이블의 스키마입니다.
base_table_name	char(128)	기본 테이블의 이름입니다.
mv_schema	char(128)	구체화된 보기의 스키마입니다.
mv_name	char(128)	구체화된 보기의 이름입니다.
state	character(32)	구체화된 뷰의 변경된 상태는 다음과 같습니다. • 다시 계산 • 새로 고칠 수 없음

샘플 쿼리

다음 쿼리는 구체화된 뷰 상태를 보여줍니다.

```
select * from sys_mv_state;
```

이 쿼리는 다음과 같은 샘플 출력을 반환합니다.

```
 user_id | transaction_id | database_name | event_desc                  | start_time
              | base_table_database_name | base_table_schema | base_table_name     |
 mv_schema  | mv_name                    | state
---------+----------------+---------------+-----------------------------
+--------------------------+-------------------------+------------------
+---------------------+-------------+---------------------------+-------------
 106     | 12720          | tickit_db     | TRUNCATE                    | 2023-07-26
 14:59:12.788268 | tickit_db                | mv_schema         | test_table_95d6d861 |
 mv_schema   | materialized_view_a1f3f862 | Recompute
 106     | 12724          | tickit_db     | ALTER TABLE ALTER DISTSTYLE | 2023-07-26
 14:59:51.409014 | tickit_db                | mv_schema         | test_table_58102435 |
 mv_schema   | materialized_view_ca746631 | Recompute
 106     | 12720          | tickit_db     | Column was renamed          | 2023-07-26
 14:59:12.822928 | tickit_db                | mv_schema         | test_table_95d6d861 |
 mv_schema   | materialized_view_5750a8d4 | Unrefreshable
 106     | 12727          | tickit_db     | Table was renamed           | 2023-07-26
 15:00:08.051244 | tickit_db                | mv_schema         | test_table_95d6d861 |
 mv_schema   | materialized_view_5750a8d4 | Unrefreshable
 106     | 12720          | tickit_db     | Column was renamed          | 2023-07-26
 14:59:12.857755 | tickit_db                | mv_schema         | test_table_95d6d861 |
 mv_schema   | materialized_view_5750a8d4 | Unrefreshable
 106     | 12727          | tickit_db     | Table was renamed           | 2023-07-26
 15:00:08.051358 | tickit_db                | mv_schema         | test_table_95d6d861 |
 mv_schema   | materialized_view_5ef0d754 | Unrefreshable
 106     | 12720          | tickit_db     | TRUNCATE                    | 2023-07-26
 14:59:12.788159 | tickit_db                | mv_schema         | test_table_95d6d861 |
 mv_schema   | materialized_view_5750a8d4 | Recompute
 106     | 12720          | tickit_db     | Column was renamed          | 2023-07-26
 14:59:12.857799 | tickit_db                | mv_schema         | test_table_95d6d861 |
 mv_schema   | materialized_view_a1f3f862 | Unrefreshable
 106     | 12720          | tickit_db     | TRUNCATE                    | 2023-07-26
 14:59:12.788327 | tickit_db                | mv_schema         | test_table_95d6d861 |
 mv_schema   | materialized_view_5ef0d754 | Recompute
 106     | 12727          | tickit_db     | ALTER TABLE ALTER SORTKEY   | 2023-07-26
 15:00:08.006235 | tickit_db                | mv_schema         | test_table_58102435 |
 mv_schema   | materialized_view_ca746631 | Recompute
 106     | 12720          | tickit_db     | Column was renamed          | 2023-07-26
 14:59:12.82297  | tickit_db                | mv_schema         | test_table_95d6d861 |
 mv_schema   | materialized_view_a1f3f862 | Unrefreshable
 106     | 12727          | tickit_db     | Table was renamed           | 2023-07-26
 15:00:08.051321 | tickit_db                | mv_schema         | test_table_95d6d861 |
 mv_schema   | materialized_view_a1f3f862 | Unrefreshable
```

SYS_PROCEDURE_CALL

SYS_PROCEDURE_CALL 뷰를 사용하면 시작 시간, 종료 시간, 저장 프로시저 호출의 상태, 중첩된 저장 프로시저 호출의 호출 계층 구조 등 저장 프로시저 호출에 대한 정보를 가져올 수 있습니다. 각 저장 프로시저 호출은 쿼리 ID를 수신합니다.

SYS_PROCEDURE_CALL은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
session_user_id	정수	세션 생성자이자 최상위 저장 프로시저 호출의 호출자인 사용자의 식별자입니다.
security_user_id	정수	저장 프로시저 내에서 문을 실행하는 데 사용된 권한을 소유한 사용자의 식별자입니다. 저장 프로시저가 DEFINER인 경우 저장 프로시저의 소유자 user_id입니다.
query_id	정수	프로시저 호출의 쿼리 식별자입니다.
query_text	char(4,000)	저장 프로시저 호출 쿼리의 텍스트입니다.
start_time	타임스탬프	쿼리 실행이 시작된 시간(UTC)입니다. 타임스탬프는 소수 초 단위로 6자리의 정밀도를 사용합니다. 예: 2009-06-12 11:29:19.131358.
end_time	타임스탬프	쿼리 실행이 종료된 시간(UTC)입니다. 예를 들어 타임스탬

열 명칭	데이터 유형	설명
		프는 소수 초 단위로 6자리의 정밀도를 사용합니다. 예: 2009-06-12 11:29:19.131358.
status	char(10)	저장 프로시저 호출의 상태입니다. 저장 프로시저가 시스템에 의해 중지되거나 사용자가 취소한 경우 값이 취소됩니다. 저장 프로시저 호출이 완료될 때까지 실행되면 값은 성공입니다.
caller_procedure_query_id	정수	저장 프로시저 호출이 다른 프로시저 호출에 의해 호출된 경우, 이 열에 외부 호출의 쿼리 ID가 포함됩니다. 그렇지 않은 경우 이 필드는 NULL입니다.

샘플 쿼리

다음 쿼리는 중첩된 저장 프로시저 호출의 계층 구조를 반환합니다.

```
select query_id, datediff(seconds, start_time, end_time) as elapsed_time, status,
 trim(query_text) as call, caller_procedure_query_id from sys_procedure_call;
```

샘플 출력은 다음과 같습니다.

```
 query_id | elapsed_time | status  |                        call                        |
 caller_procedure_query_id
----------+--------------+---------+----------------------------------------------------
+---------------------------
     3087 |           18 | success | CALL proc_bd906c98c45443ffa165e9552056902d(1)    |
          3085
     3085 |           18 | success | CALL proc_bd906c98c45443ffa165e9552056902d_2(1); |

(2 rows)
```

SYS_PROCEDURE_MESSAGES

SYS_PROCEDURE_MESSAGES는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
transaction_id	bigint	트랜잭션 식별자입니다.
query_id	정수	저장 프로시저 호출의 쿼리 식별자입니다.
record_time	타임스탬프	메시지가 생성된 시간(UTC)입니다.
log_level	char(10)	생성된 메시지의 로그 수준입니다. 가능한 값은 LOG, INFO, NOTICE, WARNING, EXCEPTION입니다.
message	char(1024)	생성된 메시지의 텍스트입니다.
line_number	정수	생성된 메시지의 줄 번호입니다.

샘플 쿼리

다음 쿼리는 SYS_PROCEDURE_MESSAGES의 샘플 출력을 보여줍니다.

```
select transaction_id, query_id, record_time, log_level, trim(message), line_number
 from sys_procedure_messages;
```

```
transaction_id | query_id |        record_time         | log_level |           btrim
        | line_number
```

```
--------------+----------+---------------------------+-----------
+--------------------------+-------------
     25267     |   80562  | 2023-07-17 14:38:31.910136 |   NOTICE  |
 test_notice_msg_b9f1e749  |     8
     25267     |   80562  | 2023-07-17 14:38:31.910002 |    LOG    |
 test_log_msg_833c7420     |     6
     25267     |   80562  | 2023-07-17 14:38:31.910111 |    INFO   |
 test_info_msg_651373d9    |     7
     25267     |   80562  | 2023-07-17 14:38:31.910154 |   WARNING |
 test_warning_msg_831c5747 |     9
(4 rows)
```

SYS_QUERY_DETAIL

SYS_QUERY_DETAIL을 사용하여 다양한 지표 수준에서 쿼리에 대한 세부 정보를 봅니다. 각 행은 지정된 지표 수준에서 특정 WLM 쿼리에 대한 세부 정보를 나타냅니다. 이 보기에는 DDL, DML 및 유틸리티 명령(예: 복사 및 언로드)과 같은 다양한 유형의 쿼리가 포함되어 있습니다. 일부 열은 쿼리 유형에 따라 관련이 없을 수 있습니다. 예를 들어, external_scanned_bytes는 내부 테이블과 관련이 없습니다.

SYS_QUERY_DETAIL은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	쿼리를 제출한 사용자의 식별자입니다.
query_id	bigint	쿼리 식별자입니다.
child_query_sequence	정수	재작성된 사용자 쿼리의 순서입니다(1로 시작).
stream_id	정수	쿼리 스트림의 스트림 식별자입니다.

열 명칭	데이터 유형	설명
segment_id	정수	쿼리 실행 세그먼트의 세그먼트 식별자입니다.
step_id	정수	세그먼트의 단계 식별자입니다.
step_name	텍스트	세그먼트의 단계 이름입니다. 가능한 값은 `aggregate`, `broadcast`, `delete`, `distribute`, `hash`, `hashjoin`, `insert`, `limit`, `merge`, `nestloop`, `parse`, `return`, `save`, `scan`, `sort`, `sortlimit`, `unique` 및 `window`입니다.
table_id	정수	영구 테이블 스캔을 위한 테이블 식별자입니다.
table_name	character(136)	작동 중인 단계의 테이블 이름입니다.
is_rrscan	character	단계가 스캔 단계인지 여부를 나타내는 값입니다. True(t)는 범위 제한 스캔이 사용되었음을 나타냅니다.
start_time	타임스탬프	쿼리 단계가 시작된 시간입니다. 이 필드는 `metrics_level` 열의 값과 관계없이 세그먼트 수준에서 기록됩니다.
end_time	타임스탬프	쿼리 단계가 완료된 시간입니다. 이 필드는 `metrics_level` 열의 값과 관계없이 세그먼트 수준에서 기록됩니다.

열 명칭	데이터 유형	설명
duration	bigint	단계에 소요된 시간(마이크로초)입니다. 이 필드는 `metrics_level` 열의 값과 관계없이 세그먼트 수준에서 기록됩니다.
알림	텍스트	알림 이벤트에 대한 설명입니다.
input_bytes	bigint	현재 단계의 입력 바이트입니다.
input_rows	bigint	현재 단계의 입력 바이트입니다.
output_bytes	bigint	현재 단계의 출력 바이트입니다.
output_rows	bigint	현재 단계의 출력 행입니다.
blocks_read	bigint	단계에서 읽은 블록 수입니다.
blocks_write	bigint	단계에서 작성한 블록 수입니다.
local_read_IO	bigint	로컬 디스크 캐시의 블록 읽기 수입니다.
remote_read_IO	bigint	원격에서 읽은 블록 수입니다.
source	텍스트	스캔한 데이터베이스 객체의 형식입니다. 이 열은 행의 step_name 값이 `scan`인 경우에만 값을 갖습니다.

열 명칭	데이터 유형	설명
data_skewness	정수	모든 단계 간의 출력 행 분포의 비대칭도입니다. 0%에서 100% 사이의 숫자입니다. 숫자가 클수록 분포가 더 불균형하다는 뜻입니다.
time_skewness	정수	모든 단계 간의 실행 시간 분포의 비대칭도입니다. 0%에서 100% 사이의 숫자입니다. 숫자가 클수록 분포가 더 불균형하다는 뜻입니다.
is_active	character	단계 수준에서 쿼리의 상태입니다. 가능한 값은 단계가 현재 실행 중임을 나타내는 't'이거나 단계가 실행을 완료했음을 나타내는 'f'입니다.
spilled_block_local_disk	bigint	로컬 디스크로 유출된 블록 수입니다.
spilled_block_remote_disk	bigint	Amazon Simple Storage Service에 유출된 블록 수입니다.
step_attribute	character(64)	관련 단계에 대한 정보가 들어 있습니다. 스캔 단계에서 가능한 값: `multi-dimensional` .

열 명칭	데이터 유형	설명
metrics_level	character(64)	쿼리의 지표 수준입니다. 가능한 값은 다음과 같습니다. • child query • 스트림 • segment • 단계 •
plan_parent_id	정수	계획 노드의 상위 노드 식별자입니다. 상위 노드에는 여러 하위 노드가 있을 수 있습니다. 예를 들어 병합 조인은 조인된 테이블에서 스캔의 상위 노드입니다.
plan_node_id	정수	계획 노드 식별자로, 쿼리에서 1개 이상의 단계에 매핑합니다.

사용 노트

SYS_QUERY_DETAIL에는 단계, 스트림, 세그먼트 및 하위 쿼리 수준의 지표가 포함될 수 있습니다. metrics_level 열을 참조하는 것 외에도 다음 표에 따라 step_id, segment_id 및 stream_id 필드를 참조하여 지정된 행에 표시되는 지표 수준을 확인할 수 있습니다.

지표 수준	stream_id 값	segment_id 값	step_id 값
child query	-1	-1	-1
스트림	유효한 단계 값	-1	-1
segment	유효한 단계 값	유효한 단계 값	-1
단계	유효한 단계 값	유효한 단계 값	유효한 단계 값

샘플 쿼리

다음 예에서는 SYS_QUERY_DETAIL의 출력을 반환합니다.

다음 쿼리는 단계 이름, input_bytes, output_bytes, input_rows, output_rows 등 단계 수준의 쿼리 메타데이터 세부 정보를 보여줍니다.

```
SELECT query_id,
       child_query_sequence,
       stream_id,
       segment_id,
       step_id,
       trim(step_name) AS step_name,
       duration,
       input_bytes,
       output_bytes,
       input_rows,
       output_rows
FROM sys_query_detail
WHERE query_id IN (193929)
ORDER BY query_id,
         stream_id,
         segment_id,
         step_id DESC;
```

샘플 출력은 다음과 같습니다.

```
 query_id | child_query_sequence | stream_id | segment_id | step_id | step_name  |
 duration     | input_bytes | output_bytes | input_rows | output_rows
----------+----------------------+-----------+------------+---------+------------
+-----------------+-------------+--------------+------------+-------------
   193929 |                    2 |         0 |          0 |       3 | hash       |
      37144 |           0 |      9350272 |          0 |      292196
   193929 |                    5 |         0 |          0 |       3 | hash       |
       9492 |           0 |        23360 |          0 |        1460
   193929 |                    1 |         0 |          0 |       3 | hash       |
      46809 |           0 |      9350272 |          0 |      292196
   193929 |                    4 |         0 |          0 |       2 | return     |
       7685 |           0 |          896 |          0 |         112
   193929 |                    1 |         0 |          0 |       2 | project    |
      46809 |           0 |            0 |          0 |      292196
   193929 |                    2 |         0 |          0 |       2 | project    |
      37144 |           0 |            0 |          0 |      292196
```

```
193929 |                   5 |         0 |          0 |       2 | project    |
    9492 |           0 |           0 |          0 |        1460
193929 |                   3 |         0 |          0 |       2 | return     |
   11033 |           0 |       14336 |          0 |         112
193929 |                   2 |         0 |          0 |       1 | project    |
   37144 |           0 |           0 |          0 |      292196
193929 |                   1 |         0 |          0 |       1 | project    |
   46809 |           0 |           0 |          0 |      292196
193929 |                   5 |         0 |          0 |       1 | project    |
    9492 |           0 |           0 |          0 |        1460
193929 |                   3 |         0 |          0 |       1 | aggregate  |
   11033 |           0 |      201488 |          0 |          14
193929 |                   4 |         0 |          0 |       1 | aggregate  |
    7685 |           0 |       28784 |          0 |          14
193929 |                   5 |         0 |          0 |       0 | scan       |
    9492 |           0 |       23360 |     292196 |        1460
193929 |                   4 |         0 |          0 |       0 | scan       |
    7685 |           0 |        1344 |        112 |         112
193929 |                   2 |         0 |          0 |       0 | scan       |
   37144 |           0 |     7304900 |     292196 |      292196
193929 |                   3 |         0 |          0 |       0 | scan       |
   11033 |           0 |       13440 |        112 |         112
193929 |                   1 |         0 |          0 |       0 | scan       |
   46809 |           0 |     7304900 |     292196 |      292196
193929 |                   5 |         0 |          0 |      -1 |            |
    9492 |       12288 |           0 |          0 |         0
193929 |                   1 |         0 |          0 |      -1 |            |
   46809 |       16384 |           0 |          0 |         0
193929 |                   2 |         0 |          0 |      -1 |            |
   37144 |       16384 |           0 |          0 |         0
193929 |                   4 |         0 |          0 |      -1 |            |
    7685 |       28672 |           0 |          0 |         0
193929 |                   3 |         0 |          0 |      -1 |            |
   11033 |      114688 |           0 |          0 |         0
```

데이터베이스의 테이블을 가장 많이 사용된 것부터 가장 적게 사용된 것까지 순서대로 보려면 다음 예제를 사용하세요. *sample_data_dev*를 사용자 데이터베이스로 바꿉니다. 이 쿼리는 클러스터가 생성된 시점부터 쿼리를 계산하지만, 데이터 웨어하우스에 공간이 부족한 경우 시스템 뷰 데이터는 저장되지 않습니다.

```
SELECT table_name, COUNT (DISTINCT query_id)
FROM SYS_QUERY_DETAIL
WHERE table_name LIKE 'sample_data_dev%'
```

```
GROUP BY table_name
ORDER BY COUNT(*) DESC;

+--------------------------------+-------+
|           table_name           | count |
+--------------------------------+-------+
| sample_data_dev.tickit.venue   |     4 |
| sample_data_dev.myunload1.venue |     3 |
| sample_data_dev.tickit.listing  |     1 |
| sample_data_dev.tickit.category |     1 |
| sample_data_dev.tickit.users    |     1 |
| sample_data_dev.tickit.date     |     1 |
| sample_data_dev.tickit.sales    |     1 |
| sample_data_dev.tickit.event    |     1 |
+--------------------------------+-------+
```

다음 예시에서는 단일 WLM 쿼리에 대한 다양한 지표 수준을 보여줍니다.

```
SELECT query_id, child_query_sequence, stream_id, segment_id, step_id, step_name,
 start_time, end_time, metrics_level
FROM sys_query_detail
WHERE query_id = 1553 AND step_id = -1
ORDER BY stream_id, segment_id, step_id;

 query_id | child_query_sequence | stream_id | segment_id | step_id | step_name |
   start_time         |          end_time          | metrics_level
----------+----------------------+-----------+------------+---------+-----------
+----------------------------+----------------------------+---------------
     1553 |                    1 |        -1 |         -1 |      -1 |           |
 2024-10-17 02:28:49.814721 | 2024-10-17 02:28:49.847838 | child query
     1553 |                    1 |         0 |         -1 |      -1 |           |
 2024-10-17 02:28:49.814721 | 2024-10-17 02:28:49.835609 | stream
     1553 |                    1 |         0 |          0 |      -1 |           |
 2024-10-17 02:28:49.824677 | 2024-10-17 02:28:49.830372 | segment
     1553 |                    1 |         1 |         -1 |      -1 |           |
 2024-10-17 02:28:49.835624 | 2024-10-17 02:28:49.845773 | stream
     1553 |                    1 |         1 |          1 |      -1 |           |
 2024-10-17 02:28:49.84088  | 2024-10-17 02:28:49.842388 | segment
     1553 |                    1 |         1 |          2 |      -1 |           |
 2024-10-17 02:28:49.835926 | 2024-10-17 02:28:49.844396 | segment
     1553 |                    1 |         2 |         -1 |      -1 |           |
 2024-10-17 02:28:49.846949 | 2024-10-17 02:28:49.847838 | stream
```

```
     1553 |                    1 |         2 |          3 |      -1 |          |
 2024-10-17 02:28:49.847013 | 2024-10-17 02:28:49.847485 | segment
(8 rows)
```

SYS_QUERY_EXPLAIN

실행을 목적으로 제출된 쿼리의 EXPLAIN 계획을 표시합니다.

SYS_QUERY_EXPLAIN은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	쿼리를 제출한 사용자의 식별자입니다.
query_id	bigint	쿼리 식별자입니다. 자세한 쿼리 정보는 SYS_QUERY_HISTORY에 저장됩니다.
child_query_sequence	정수	재작성된 사용자 쿼리의 순서입니다(1로 시작).
plan_node_id	정수	계획 노드 식별자로, 쿼리에서 1개 이상의 단계에 매핑합니다.
plan_parent_id	정수	계획 노드의 상위 노드 식별자입니다. 상위 노드에는 여러 하위 노드가 있을 수 있습니다. 예를 들어 병합 조인은 조인된 테이블에서 스캔의 상위 노드입니다.
plan_node	character(400)	EXPLAIN 출력의 노드 텍스트. 컴퓨팅 노드에서 실행을 의미하는 계획 노드에는 EXPLAIN

열 명칭	데이터 유형	설명
		출력 시 XN 접두사가 첨부됩니다.
node_info	character(400)	계획 노드의 한정자 및 필터 정보. 예를 들어 조인 조건과 WHERE 절 제한이 이 열에 포함됩니다.

샘플 쿼리

다음 예시는 단일 쿼리의 EXPLAIN 계획입니다.

```
SELECT * FROM sys_query_explain WHERE query_id = 612635 ORDER_BY plan_node_id;

 userid | query_id | child_query_sequence | plan_node_id | plan_parent_id |


          plan_node

                                          |

                                                                plan_info



--------+----------+----------------------+--------------+----------------
+-----------------------------------------------------------------------------------------------------
+-----------------------------------------------------------------------------------------------------
    100 |   612635 |                    1 |            1 |              0 | XN Limit
 (cost=3604047533041.00..3604047533041.25 rows=100 width=20)


                                            |
```

```
   100 |   612635 |                    1 |           2 |             1 |   ->  XN
Merge  (cost=3604047533041.00..3604047533148.02 rows=42809 width=20)

                                          | Merge Key: sum(b.totalprice)

   100 |   612635 |                    1 |           3 |             2 |         ->
XN Network  (cost=3604047533041.00..3604047533148.02 rows=42809 width=20)

                                        |   Send to leader

   100 |   612635 |                    1 |           4 |             3 |
  ->  XN Sort  (cost=3604047533041.00..3604047533148.02 rows=42809 width=20)

                                        | Sort Key: sum(b.totalprice)

   100 |   612635 |                    1 |           5 |             4 |
           ->  XN HashAggregate  (cost=2604047529640.76..2604047529747.78 rows=42809
width=20)

                                             |
```

```
   100 |   612635 |                    1 |           6 |             5 |
                ->  XN Hash Join DS_DIST_NONE  (cost=15104956.16..2602364653507.34
rows=336575226684 width=20)


                                                      | Hash Cond: (("outer".listid =
"inner".listid) AND ("outer".sellerid = "inner".sellerid))




   100 |   612635 |                    1 |           7 |             6 |
                           ->  XN Seq Scan on listing b  (cost=0.00..7884677.12
rows=788467712 width=24)

                                                       |




   100 |   612635 |                    1 |           8 |             6 |
                   ->  XN Hash  (cost=7063797.76..7063797.76 rows=706379776 width=8)

                                        |




   100 |   612635 |                    1 |           9 |             8 |
                               ->  XN Seq Scan on sales a  (cost=0.00..7063797.76
rows=706379776 width=8)

                                                  |
```

```
(9 rows)
```

SYS_QUERY_HISTORY

SYS_QUERY_HISTORY를 사용하여 사용자 쿼리의 세부 정보를 봅니다. 각 행은 일부 필드에 대한 누적 통계가 있는 사용자 쿼리를 나타냅니다. 이 보기에는 데이터 정의 언어(DDL), 데이터 조작 언어(DML), 복사, 언로드 및 Amazon Redshift Spectrum과 같은 다양한 유형의 쿼리가 포함되어 있습니다. 여기에는 실행 중인 쿼리와 완료된 쿼리가 모두 포함됩니다.

SYS_QUERY_HISTORY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	쿼리를 제출한 사용자의 식별자입니다.
query_id	bigint	쿼리 식별자입니다.
query_label	character(320)	쿼리의 짧은 이름입니다.
transaction_id	bigint	트랜잭션 식별자입니다.
session_id	정수	쿼리를 실행하는 프로세스의 프로세스 식별자입니다.
database_name	character(128)	쿼리가 실행되었을 때 사용자가 연결된 데이터베이스의 이름.
query_type	character(32)	SELECT, INSERT, UPDATE, UNLOAD COPY, COMMAND, DDL, UTILITY, CTAS, OTHER 등의 쿼리 유형입니다.

열 명칭	데이터 유형	설명
status	character(10)	쿼리의 상태입니다. 유효한 값: 계획 중, 대기됨, 실행 중, 반환 중, 실패, 취소됨 및 성공.
result_cache_hit	불	쿼리가 결과 캐시와 일치하는지를 나타냅니다.
start_time	타임스탬프	쿼리가 시작된 시간입니다.
end_time	타임스탬프	쿼리가 완료된 시간입니다.
경과 시간	bigint	쿼리에 소요된 총 시간(마이크로초)입니다.
queue_time	bigint	서비스 클래스 쿼리 대기열에 소요된 총 시간(마이크로초)입니다.
execution_time	bigint	서비스 클래스에서 실행 중인 총 시간(마이크로초)입니다.
error_message	character(512)	쿼리가 실패한 이유입니다.
returned_rows	bigint	클라이언트에게 반환되는 행 수입니다.
returned_bytes	bigint	클라이언트에게 반환되는 바이트 수입니다.
query_text	character(4000)	쿼리 문자열. 이 문자열은 잘릴 수 있습니다.
redshift_version	character(256)	쿼리가 실행된 Amazon Redshift 버전입니다.
usage_limit	character(150)	쿼리에서 도달한 사용량 제한 ID의 목록입니다.

열 명칭	데이터 유형	설명
compute_type	varchar(32)	쿼리가 기본 클러스터에서 실행되는지 아니면 동시성 확장 클러스터에서 실행되는지를 나타냅니다. 가능한 값은 `primary`(기본 클러스터에서 쿼리 실행), `secondary` (보조 클러스터에서 쿼리 실행) 또는 `primary-scale` (동시성 클러스터에서 쿼리 실행)입니다. 이는 프로비전된 클러스터에만 적용됩니다.
compile_time	bigint	쿼리 컴파일에 소요된 총시간(마이크로초)입니다.
planning_time	bigint	쿼리 계획에 소요된 총시간(마이크로초)입니다.
lock_wait_time	bigint	관계 잠금을 기다리는 데 소요된 총시간(마이크로초)입니다.
service_class_id	정수	서비스 클래스 ID입니다. 서비스 클래스 ID의 목록을 보려면 WLM 서비스 클래스 ID 섹션으로 이동하세요. 이 열은 프로비저닝된 클러스터에서 실행되는 쿼리에만 사용됩니다. Redshift Serverless에서 실행되는 쿼리의 경우, 이 열에는 -1이 포함됩니다.

열 명칭	데이터 유형	설명
service_class_name	character(64)	서비스 클래스 이름입니다. 이 열은 프로비저닝된 클러스터에서 실행되는 쿼리에만 사용됩니다. Amazon Redshift Redshift Serverless에서 실행되는 쿼리의 경우. 이 열이 비어 있습니다.
query_priority	character(20)	쿼리가 실행된 대기열의 우선 순위. 가능한 값은 다음과 같습니다. • NULL • 가장 낮음 • low • 정상 • high • 가장 높음 NULL은 해당 쿼리에 쿼리 우선 순위가 지원되지 않음을 의미합니다. 이 열은 프로비저닝된 클러스터에서 실행되는 쿼리에만 사용됩니다. Redshift Serverless에서 실행되는 쿼리의 경우. 이 열이 비어 있습니다.

열 명칭	데이터 유형	설명
short_query_accelerated	character(10)	단기 쿼리 가속화(SQA)를 사용하여 쿼리를 가속했는지 여부. 가능한 값은 다음과 같습니다. • true • false • NULL 이 열은 프로비저닝된 클러스터에서 실행되는 쿼리에만 사용됩니다. Redshift Serverless에서 실행되는 쿼리의 경우. 이 열이 비어 있습니다.
user_query_hash	character(40)	쿼리 리터럴을 포함하여 쿼리에서 생성된 쿼리 해시입니다. 동일한 쿼리 텍스트가 포함된 반복 쿼리는 user_query_hash 값이 동일합니다.
generic_query_hash	character(40)	쿼리 리터럴을 제외한 쿼리에서 생성된 쿼리 해시입니다. 동일한 쿼리 텍스트가 포함되지만 다른 쿼리 리터럴이 있는 반복 쿼리는 generic_query_hash 값이 동일합니다.
query_hash_version	정수	쿼리에서 생성된 쿼리 해시의 버전 번호입니다.

샘플 쿼리

다음 쿼리는 실행 중인 쿼리와 대기 중인 쿼리를 반환합니다.

```
SELECT user_id,
```

```
       query_id,
       transaction_id,
       session_id,
       status,
       trim(database_name) AS database_name,
       start_time,
       end_time,
       result_cache_hit,
       elapsed_time,
       queue_time,
       execution_time
FROM sys_query_history
WHERE status IN ('running','queued')
ORDER BY start_time;
```

샘플 출력은 다음과 같습니다.

```
 user_id | query_id | transaction_id | session_id | status  | database_name |
 start_time         |          end_time          | result_cache_hit | elapsed_time |
 queue_time | execution_time
---------+----------+----------------+------------+---------+---------------
+---------------------------+---------------------------+-----------------
+--------------+------------+----------------
     101 |   760705 |         852337 | 1073832321 | running | tpcds_1t      |
 2022-02-15 19:03:19.67849 | 2022-02-15 19:03:19.739811 | f                |
 61321 |          0 |              0
```

다음 쿼리는 특정 쿼리의 쿼리 시작 시간, 종료 시간, 대기열 시간, 경과 시간, 계획 시간 및 기타 메타데이터를 반환합니다.

```
SELECT user_id,
       query_id,
       transaction_id,
       session_id,
       status,
       trim(database_name) AS database_name,
       start_time,
       end_time,
       result_cache_hit,
       elapsed_time,
       queue_time,
       execution_time,
       planning_time,
```

```
       trim(query_text) as query_text
FROM sys_query_history
WHERE query_id = 3093;
```

샘플 출력은 다음과 같습니다.

```
user_id | query_id | transaction_id | session_id |   status   | database_name |
   start_time         |          end_time          | result_cache_hit | elapsed_time |
 queue_time | execution_time | planning_time | query_text
--------+----------+----------------+------------+------------
+---------------+----------------------------+----------------------------
+------------------+--------------+------------+----------------+--------------
+-------------------------------------
    106 |     3093 |          11759 | 1073750146 | success    | dev           |
 2023-03-16 16:53:17.840214 | 2023-03-16 16:53:18.106588 | f                |
 266374 |          0 |         105725 |        136589 | select count(*) from item;
```

다음은 가장 최근 SELECT 쿼리 10개를 나열하는 쿼리입니다.

```
SELECT query_id,
       transaction_id,
       session_id,
       start_time,
       elapsed_time,
       queue_time,
       execution_time,
       returned_rows,
       returned_bytes
FROM sys_query_history
WHERE query_type = 'SELECT'
ORDER BY start_time DESC limit 10;
```

샘플 출력은 다음과 같습니다.

```
 query_id | transaction_id | session_id |         start_time         | elapsed_time |
 queue_time | execution_time | returned_rows | returned_bytes
----------+----------------+------------+----------------------------+--------------
+------------+----------------+---------------+----------------
   526532 |          61093 | 1073840313 | 2022-02-09 04:43:24.149603 |       520571 |
        0 |         481293 |             1 |           3794
```

```
   526520 |          60850 | 1073840313 | 2022-02-09 04:38:27.24875  |       635957 |
        0 |         596601 |          1 |           3679
   526508 |          60803 | 1073840313 | 2022-02-09 04:37:51.118835 |       563882 |
        0 |         503135 |          5 |          17216
   526505 |          60763 | 1073840313 | 2022-02-09 04:36:48.636224 |       649337 |
        0 |         589823 |          1 |            652
   526478 |          60730 | 1073840313 | 2022-02-09 04:36:11.741471 |     14611321 |
        0 |       14544058 |          0 |              0
   526467 |          60636 | 1073840313 | 2022-02-09 04:34:11.91463  |     16711367 |
        0 |       16633767 |          1 |            575
   511617 |         617946 | 1074009948 | 2022-01-20 06:21:54.44481  |      9937090 |
        0 |        9899271 |        100 |          12500
   511603 |         617941 | 1074259415 | 2022-01-20 06:21:45.71744  |      8065081 |
        0 |        7582500 |        100 |           8889
   511595 |         617935 | 1074128320 | 2022-01-20 06:21:44.030876 |      1051270 |
        0 |        1014879 |          1 |             72
   511584 |         617931 | 1074030019 | 2022-01-20 06:21:42.764088 |       609033 |
        0 |         485887 |        100 |           8438
```

다음 쿼리는 일별 선택 쿼리 수와 평균 쿼리 경과 시간을 보여줍니다.

```
SELECT date_trunc('day',start_time) AS exec_day,
       status,
       COUNT(*) AS query_cnt,
       AVG(datediff (microsecond,start_time,end_time)) AS elapsed_avg
FROM sys_query_history
WHERE query_type = 'SELECT'
AND start_time >= '2022-01-14'
AND start_time <= '2022-01-18'
GROUP BY exec_day,
         status
ORDER BY exec_day,
         status;
```

샘플 출력은 다음과 같습니다.

```
      exec_day       | status  | query_cnt | elapsed_avg
---------------------+---------+-----------+------------
 2022-01-14 00:00:00 | success |      5253 |  56608048
 2022-01-15 00:00:00 | success |      7004 |  56995017
 2022-01-16 00:00:00 | success |      5253 |  57016363
 2022-01-17 00:00:00 | success |      5309 |  55236784
```

```
 2022-01-18 00:00:00 | success |      8092 |  54355124
```

다음 쿼리는 일별 쿼리 경과 시간 성능을 보여줍니다.

```
SELECT distinct date_trunc('day',start_time) AS exec_day,
       query_count.cnt AS query_count,
       Percentile_cont(0.5) within group(ORDER BY elapsed_time) OVER (PARTITION BY
 exec_day) AS P50_runtime,
       Percentile_cont(0.8) within group(ORDER BY elapsed_time) OVER (PARTITION BY
 exec_day) AS P80_runtime,
       Percentile_cont(0.9) within group(ORDER BY elapsed_time) OVER (PARTITION BY
 exec_day) AS P90_runtime,
       Percentile_cont(0.99) within group(ORDER BY elapsed_time) OVER (PARTITION BY
 exec_day) AS P99_runtime,
       Percentile_cont(1.0) within group(ORDER BY elapsed_time) OVER (PARTITION BY
 exec_day) AS max_runtime
FROM sys_query_history
LEFT JOIN (SELECT  date_trunc('day',start_time) AS day, count(*) cnt
           FROM sys_query_history
           WHERE query_type = 'SELECT'
           GROUP by 1) query_count
ON date_trunc('day',start_time) = query_count.day
WHERE query_type = 'SELECT'
ORDER BY exec_day;
```

샘플 출력은 다음과 같습니다.

```
      exec_day       | query_count | p50_runtime | p80_runtime | p90_runtime |
 p99_runtime  | max_runtime
--------------------+-------------+-------------+-------------+-------------
+-------------+-------------
 2022-01-14 00:00:00 |        5253 |  16816922.0 |  69525096.0 | 158524917.8 |
 486322477.52 | 1582078873.0
 2022-01-15 00:00:00 |        7004 |  15896130.5 |  71058707.0 | 164314568.9 |
 500331542.07 | 1696344792.0
 2022-01-16 00:00:00 |        5253 |  15750451.0 |  72037082.2 | 159513733.4 |
 480372059.24 | 1594793766.0
 2022-01-17 00:00:00 |        5309 |  15394513.0 |  68881393.2 | 160254700.0 |
 493372245.84 | 1521758640.0
 2022-01-18 00:00:00 |        8092 |  15575286.5 |  68485955.4 | 154559572.5 |
 463552685.39 | 1542783444.0
```

```
 2022-01-19 00:00:00 |        5860 |  16648747.0 |  72470482.6 | 166485138.2 |
 492038228.67 | 1693483241.0
 2022-01-20 00:00:00 |        1751 |  15422072.0 |  69686381.0 | 162315385.0 |
 497066615.00 | 1439319739.0
 2022-02-09 00:00:00 |          13 |   6382812.0 |  17616161.6 |  21197988.4 |
 23021343.84 |   23168439.0
```

다음 쿼리는 쿼리 유형 분포를 보여줍니다.

```
SELECT query_type,
       COUNT(*) AS query_count
FROM sys_query_history
GROUP BY query_type
ORDER BY query_count DESC;
```

샘플 출력은 다음과 같습니다.

```
 query_type | query_count
------------+-------------
 UTILITY    |      134486
 SELECT     |       38537
 DDL        |        4832
 OTHER      |         768
 LOAD       |         768
 CTAS       |         748
 COMMAND    |          92
```

다음 예제에서는 여러 쿼리 간의 쿼리 해시 결과의 차이를 보여 줍니다. 다음 쿼리를 관찰합니다.

```
CREATE TABLE test_table (col1 INT);

INSERT INTO test_table VALUES (1),(2);

SELECT * FROM test_table;

SELECT * FROM test_table;

SELECT col1 FROM test_table;

SELECT * FROM test_table WHERE col1=1;

SELECT * FROM test_table WHERE col1=2;
```

```
SELECT query_id, TRIM(user_query_hash) AS user_query_hash, TRIM(generic_query_hash)
 AS generic_query_hash, TRIM(query_text) AS text FROM sys_query_history ORDER BY
 start_time
DESC LIMIT 10;
```

다음은 출력 샘플입니다.

```
query_id | user_query_hash | generic_query_hash | text
---------+-----------------+--------------------+----------
24723049 | oPuFtjEPLTs=    | oPuFtjEPLTs=       | select query_id,
 trim(user_query_hash) as user_query_hash, trim(generic_query_hash) as
 generic_query_hash, query_hash_version, trim(query_text) as text from
 sys_query_history order by start_time\r\ndesc limit 20
24723045 | Gw2Kwdd8m2I=    | IwfRu8/XAKI=       | select * from test_table where col1=2
 limit 100
24723041 | LNw2vx0GDXo=    | IwfRu8/XAKI=       | select * from test_table where col1=1
 limit 100
24723036 | H+qep/c82Y8=    | H+qep/c82Y8=       | select col1 from test_table limit 100
24723033 | H+qep/c82Y8=    | H+qep/c82Y8=       | select * from test_table limit 100
24723029 | H+qep/c82Y8=    | H+qep/c82Y8=       | select * from test_table limit 100
24723023 | 50sirx9E1hU=    | uO36Z1a/QYs=       | insert into test_table values (1),(2)
24723021 | YSVnlivZHeo=    | YSVnlivZHeo=       | create table test_table (col1 int)
```

`SELECT * FROM test_table;` 및 `SELECT col1 FROM test_table;`에는 열이 하나뿐이므로 동일한 user_query_hash 값이 포함됩니다. `SELECT * FROM test_table WHERE col1=1;` 및 `SELECT * FROM test_table WHERE col1=2;`에는 user_query_hash 값이 다르지만 두 쿼리가 쿼리 리터럴 1과 2 외부에서 동일하므로 generic_query_hash 값은 동일합니다.

SYS_QUERY_TEXT

SYS_QUERY_TEXT를 사용하여 모든 쿼리의 쿼리 텍스트를 볼 수 있습니다. 각 행은 시퀀스 번호 0으로 시작하는 최대 4,000자의 쿼리 텍스트를 나타냅니다. 쿼리 명령문이 4,000자를 초과하는 경우 각 행의 시퀀스 번호를 증가시켜 명령문에 대한 추가 행이 로깅됩니다. 이 뷰는 DDL, 유틸리티, Amazon Redshift 쿼리 및 리더 노드 전용 쿼리와 같은 모든 사용자 쿼리 텍스트를 로깅합니다.

SYS_QUERY_TEXT는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	쿼리를 제출한 사용자의 식별자입니다.
query_id	bigint	쿼리 식별자입니다.
transaction_id	bigint	명령문과 연결된 트랜잭션의 ID입니다.
session_id	정수	쿼리를 실행하는 세션의 프로세스 식별자입니다.
start_time	타임스탬프	쿼리가 시작하는 시간입니다.
SEQUENCE	정수	단일 명령문에 4,000자 이상이 포함된 경우, 해당 명령문에 대해 추가 행이 로깅됩니다. 시퀀스 0이 첫 번째 행이고 1이 두 번째 행이 되는 방식입니다.
텍스트	character(4000)	4,000자 단위로 늘어나는 SQL 쿼리의 텍스트입니다. 이 필드에는 백슬래시(\) 및 줄 바꿈(\n) 등의 특수 문자가 포함될 수 있습니다.

샘플 쿼리

다음 쿼리는 실행 중인 쿼리와 대기 중인 쿼리를 반환합니다.

```
SELECT user_id,
 query_id,
 transaction_id,
 session_id, start_time,
 sequence, trim(text) as text from sys_query_text
 ORDER BY sequence;
```

샘플 출력은 다음과 같습니다.

```
 user_id | query_id | transaction_id | session_id |        start_time          |
 sequence |                                                        text
--------+----------+----------------+------------+----------------------------
+----------
+-----------------------------------------------------------------------------------------------------------------
   100  |     4    |       1396     | 1073750220 | 2023-04-28 16:44:55.887184 |
   0    | SELECT trim(text) as text, sequence FROM sys_query_text WHERE query_id =
 pg_last_query_id() AND user_id > 1 AND start
_time > '2023-04-28 16:44:55.922705+00:00'::timestamp order by sequence;
```

다음 쿼리는 데이터베이스의 그룹에서 부여되거나 해지된 권한을 반환합니다.

```
SELECT
    SPLIT_PART(text, ' ', 1) as grantrevoke,
    SPLIT_PART((SUBSTRING(text, STRPOS(UPPER(text), 'GROUP'))), ' ', 2) as group,
    SPLIT_PART((SUBSTRING(text, STRPOS(UPPER(text), ' '))), 'ON', 1) as type,
    SPLIT_PART((SUBSTRING(text, STRPOS(UPPER(text), 'ON'))), ' ', 2) || ' ' ||
 SPLIT_PART((SUBSTRING(text, STRPOS(UPPER(text), 'ON'))), ' ', 3) as entity
FROM SYS_QUERY_TEXT
WHERE (text LIKE 'GRANT%' OR text LIKE 'REVOKE%') AND text LIKE '%GROUP%';

+-------------+----------+--------+----------+
| grantrevoke |  group   |  type  |  entity  |
+-------------+----------+--------+----------+
| GRANT       | bi_group | SELECT | TABLE t1 |
| GRANT       | bi_group | SELECT | TABLE t1 |
| GRANT       | bi_group | SELECT | TABLE t1 |
| GRANT       | bi_group | USAGE  | TABLE t1 |
| GRANT       | bi_group | SELECT | TABLE t1 |
| GRANT       | bi_group | SELECT | TABLE t1 |
+-------------+----------+--------+----------+
```

SYS_RESTORE_LOG

RA3 노드로의 클래식 크기 조정 중에 클러스터 내 각 테이블의 마이그레이션 진행 상황을 모니터링하려면 SYS_RESTORE_LOG를 사용합니다. 이는 크기 조정 작업 중 데이터 마이그레이션의 과거 처리량을 캡처합니다. RA3 노드로의 클래식 크기 조정에 대한 자세한 내용은 클래식 크기 조정을 참조하세요.

SYS_RESTORE_LOG는 슈퍼 사용자에게만 표시됩니다.

테이블 열

열 명칭	데이터 유형	설명
event_time	타임스탬프	로그 항목이 기록되는 시기를 나타내는 타임스탬프입니다.
database_name	char(128)	데이터베이스의 이름입니다.
schema_name	char(128)	스키마의 이름입니다.
table_name	char(128)	테이블의 이름
table_id	정수	테이블의 ID입니다.
작업	char(128)	입력 시 취해진 작업입니다. 값에는 마이그레이션 시작됨, 체크포인트, 재개됨, 완료됨, 취소됨, 재설정 등이 포함될 수 있습니다.
table_size	long	테이블의 크기입니다.
total_data_processed	long	이 시점까지 처리된 테이블의 데이터 크기(MB 입니다.
delta_data_processed	long	마지막 event_time 업데이트 이후 처리된 데이터 크기(MB)입니다. 이를 통해 이전에 기록된 시간 간격 이후 처리된 데이터의 양을 확인할 수 있습니다. 이 값을 table_size와 비교하여 데이터 처리 속도를 파악할 수 있습니다.
message	char(512)	작업 열의 값에 대한 자세한 설명입니다.
redistribution_type	char(32)	테이블의 재배포 유형입니다. KEY 변환 또는 EVEN 재분배

열 명칭	데이터 유형	설명
		작업 중 하나입니다. 배포 스타일에 대한 자세한 내용은 배포 스타일을 참조하세요.

샘플 쿼리

다음 쿼리는 SYS_RESTORE_LOG를 사용하여 데이터 처리 처리량을 계산합니다.

```
SELECT
    ROUND(sum(delta_data_processed) / 1024.0, 2) as data_processed_gb,
    ROUND(datediff(sec, min(event_time), max(event_time)) / 3600.0, 2) as duration_hr,
    ROUND(data_processed_gb/duration_hr, 2) as throughput_gb_per_hr
from sys_restore_log;
```

샘플 출력은 다음과 같습니다.

```
 data_processed_gb | duration_hr | throughput_gb_per_hr
-------------------+-------------+----------------------
              0.91 |        8.37 |                 0.11
(1 row)
```

모든 재배포 유형을 보여주는 다음 쿼리입니다.

```
SELECT * from sys_restore_log ORDER BY event_time;
```

```
 database_name |     schema_name      |      table_name      | table_id |
 action             | total_data_processed | delta_data_processed |         event_time
        | table_size | message |   redistribution_type
---------------+----------------------+----------------------+----------
+----------------------------+----------------------+----------------------
+----------------------------+------------+---------+--------------------------
 dev           | schemaaaa877096d844d | customer_key         |   106424 |
 Redistribution started      |                    0 |                      | 2024-01-05
 02:18:00.744977 |        325 |         | Restore Distkey Table
 dev           | schemaaaa877096d844d | dp30907_t2_autokey   |   106430 |
 Redistribution started      |                    0 |                      | 2024-01-05
 02:18:02.756675 |         90 |         | Restore Distkey Table
```

```
 dev           | schemaaaa877096d844d | dp30907_t2_autokey   |   106430 |
 Redistribution completed    |                   90 |                   90 | 2024-01-05
 02:23:30.643718 |         90 |         | Restore Distkey Table
 dev           | schemaaaa877096d844d | customer_key         |   106424 |
 Redistribution completed    |                  325 |                  325 | 2024-01-05
 02:23:45.998249 |        325 |         | Restore Distkey Table
 dev           | schemaaaa877096d844d | dp30907_t1_even      |   106428 |
 Redistribution started      |                    0 |                      | 2024-01-05
 02:23:46.083849 |         30 |         | Rebalance Disteven Table
 dev           | schemaaaa877096d844d | dp30907_t5_auto_even |   106436 |
 Redistribution started      |                    0 |                      | 2024-01-05
 02:23:46.855728 |         45 |         | Rebalance Disteven Table
 dev           | schemaaaa877096d844d | dp30907_t5_auto_even |   106436 |
 Redistribution completed    |                   45 |                   45 | 2024-01-05
 02:24:16.343029 |         45 |         | Rebalance Disteven Table
 dev           | schemaaaa877096d844d | dp30907_t1_even      |   106428 |
 Redistribution completed    |                   30 |                   30 | 2024-01-05
 02:24:20.584703 |         30 |         | Rebalance Disteven Table
 dev           | schemaefd028a2a48a4c | customer_even        |   130512 |
 Redistribution started      |                    0 |                      | 2024-01-05
 04:54:55.641741 |        190 |         | Restore Disteven Table
 dev           | schemaefd028a2a48a4c | customer_even        |   130512 |
 Redistribution checkpointed |     29.4342113157737 |     29.4342113157737 | 2024-01-05
 04:55:04.770696 |        190 |         | Restore Disteven Table
(8 rows)
```

SYS_RESTORE_STATE

클래식 크기 조정 중에 각 테이블의 마이그레이션 진행 상황을 모니터링하려면 SYS_RESTORE_STATE를 사용합니다. 이는 구체적으로 대상 노드 유형이 RA3일 때 적용됩니다. RA3 노드로의 클래식 크기 조정에 대한 자세한 내용은 클래식 크기 조정을 참조하세요.

SYS_RESTORE_STATE는 수퍼유저만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	쿼리를 제출한 사용자의 식별자입니다.

열 명칭	데이터 유형	설명
database_name	char(64)	테이블의 데이터베이스 이름입니다.
schema_id	정수	테이블의 스키마 ID입니다.
table_id	정수	테이블의 ID입니다.
table_name	char(128)	테이블의 이름
redistribution_status	char(128)	테이블의 재배포 진행 상태입니다. 가능한 값은 `Completed` , `In progress` 및 `Pending`입니다.
percentage_redistributed	float	테이블의 재배포 진행률 백분율입니다. 가능한 값은 0에서 100%까지입니다. 예를 들어, 값이 25이면 데이터의 25%가 재배포된다는 의미입니다.
redistribution_type	char(32)	테이블의 재배포 유형입니다. KEY 변환 또는 EVEN 재분배 작업 중 하나입니다. 배포 스타일에 대한 자세한 내용은 배포 스타일을 참조하세요.

샘플 쿼리

다음 쿼리는 실행 중인 쿼리와 대기 중인 쿼리를 반환합니다.

```
SELECT * FROM sys_restore_state;
```

샘플 출력은 다음과 같습니다.

```
 userid | database_name | schema_id | table_id |   table_name   | redistribution_status
 | precentage_redistributed |   redistribution_type
```

```
--------+--------------+-----------+----------+----------------
+-----------------------+--------------------------+------------------------
    1   |     test1     |   124865  |  124878  | customer_key_4 |         Pending
 |      0                |  Rebalance Disteven Table
    1   |      dev      |   124865  |  124874  | customer_key_3 |         Pending
 |      0                |  Rebalance Disteven Table
    1   |      dev      |   124865  |  124870  | customer_key_2 |        Completed
 |     100               |  Rebalance Disteven Table
    1   |      dev      |   124865  |  124866  | customer_key_1 |       In progress
 |     13.52             |  Restore Distkey Table
```

다음은 데이터 처리 상태를 보여줍니다.

```
SELECT
    redistribution_status, ROUND(SUM(block_count) / 1024.0, 2) AS total_size_gb
FROM sys_restore_state sys inner join stv_tbl_perm stv
    on sys.table_id = stv.id
GROUP BY sys.redistribution_status;
```

샘플 출력은 다음과 같습니다.

```
 redistribution_status | total_size_gb
-----------------------+---------------
 Completed             |          0.07
 Pending               |          0.71
 In progress           |          0.20
(3 rows)
```

SYS_SCHEMA_QUOTA_VIOLATIONS

스키마 할당량을 초과한 경우 발생, 트랜잭션 ID 및 기타 유용한 정보를 기록합니다. 이 시스템 테이블은 STL_SCHEMA_QUOTA_VIOLATIONS의 변환입니다.

r_SYS_SCHEMA_QUOTA_VIOLATIONS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
owner_id	정수	스키마 소유자의 ID입니다.
user_id	정수	항목을 생성한 사용자의 ID입니다.
transaction_id	bigint	문과 연결된 트랜잭션 ID
session_id	정수	문과 연결된 프로세스 ID
schema_id	정수	네임스페이스 또는 스키마 ID입니다.
schema_name	character (128)	네임스페이스 또는 스키마 이름입니다.
할당량	정수	스키마에서 사용할 수 있는 디스크 공간(MB)입니다.
disk_usage	정수	스키마에서 현재 사용 중인 디스크 공간(MB)입니다.
record_time	시간대 미포함 TIMESTAMP	위반이 발생한 시간입니다.

샘플 쿼리

다음은 할당량 위반의 결과를 보여 주는 쿼리입니다.

```
SELECT user_id, TRIM(schema_name) "schema_name", quota, disk_usage, record_time FROM
sys_schema_quota_violations WHERE SCHEMA_NAME = 'sales_schema' ORDER BY timestamp DESC;
```

이 쿼리는 지정된 스키마에 대해 다음 샘플 출력을 반환합니다.

```
user_id| schema_name  | quota | disk_usage | record_time
-------+--------------+-------+------------+----------------------------
104    | sales_schema | 2048  | 2798       | 2020-04-20 20:09:25.494723
```

```
(1 row)
```

SYS_SERVERLESS_USAGE

SYS_SERVERLESS_USAGE를 사용하여 리소스의 Amazon Redshift Serverless 사용에 대한 세부 정보를 봅니다. 이 시스템 보기는 프로비저닝된 Amazon Redshift 클러스터에는 적용되지 않습니다.

이 보기에는 쿼리를 처리하는 데 사용되는 컴퓨팅 용량과 1분 단위로 사용되는 Amazon Redshift 관리형 스토리지의 양을 포함한 서버리스 사용량 요약이 포함되어 있습니다. 컴퓨팅 용량은 Redshift 처리 단위(RPU)로 측정되며 실행하는 워크로드에 대해 초당 RPU(초) 단위로 측정됩니다. RPU는 데이터 웨어하우스에 로드되거나, Amazon S3 데이터 레이크에서 쿼리되거나, 연합 쿼리를 사용하여 운영 데이터베이스에서 액세스되는 데이터에 대한 쿼리를 처리하는 데 사용됩니다. Amazon Redshift Serverless는 7일 동안 SYS_SERVERLESS_USAGE에 정보를 보관합니다.

컴퓨팅 비용 청구에 대한 예시는 Amazon Redshift Serverless에 대한 청구를 참조하세요.

SYS_SERVERLESS_USAGE는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
start_time	타임스탬프	간격이 시작된 시간입니다.
end_time	타임스탬프	간격이 완료된 시간입니다.
compute_seconds	double precision	이 시간 간격 동안 사용된 누적 컴퓨팅 단위(RPU) 초입니다. 이 값은 계정에 할당된 기본 RPU 용량을 나타냅니다.
compute_capacity	double precision	이 시간 간격 동안 할당된 평균 컴퓨팅 단위(Redshift 처리 단위 또는 RPU) 수입니다. compute_capacity 값은 동적으로 변경될 수 있습니다.

열 명칭	데이터 유형	설명
data_storage	정수	이 시간 간격 동안 사용된 평균 데이터 스토리지 공간(MB)입니다. 사용된 데이터 스토리지는 데이터가 데이터베이스에서 로드되거나 삭제될 때 동적으로 변경될 수 있습니다.
cross_region_transferred_data	정수	이 시간 간격 동안 리전 간 데이터 공유를 위해 전송된 누적 데이터(바이트)입니다.
charged_seconds	정수	이 시간 간격 동안 청구된 누적 컴퓨팅 단위(RPU) 초입니다. 트랜잭션이 종료된 후에 계산되므로 트랜잭션이 실행되는 동안 0이 될 수 있습니다. charge_seconds를 사용하여 Amazon Redshift Serverless 작업 그룹의 비용을 계산합니다. 이 값은 Amazon Redshift Serverless 작업 그룹에 할당된 RPU 용량을 설명합니다.

사용 노트

- compute_seconds가 0이지만 charged_seconds가 0보다 크거나 그 반대의 경우가 있습니다. 이는 데이터가 시스템 뷰에 기록되는 방식에 따른 정상적인 동작입니다. 서버리스 사용량 세부 정보를 보다 정확하게 표시하려면 데이터를 집계하는 것이 좋습니다.

예제

charge_seconds를 쿼리하여 시간 간격에 사용된 RPU 시간에 대한 총 요금을 얻으려면 다음 쿼리를 실행합니다.

```
select trunc(start_time) "Day",
(sum(charged_seconds)/3600::double precision) * <Price for 1 RPU> as cost_incurred
from sys_serverless_usage
group by 1
order by 1
```

간격 동안 유휴 시간이 있을 수 있습니다. 유휴 시간은 사용된 RPU에 추가되지 않습니다.

SYS_SESSION_HISTORY

SYS_SESSION_HISTORY를 사용하여 현재 활성 세션 및 세션 기록에 대한 정보를 볼 수 있습니다.

SYS_SESSION_HISTORY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	char(50)	항목을 생성한 사용자의 식별자입니다.
session_id	정수	문과 연결된 세션의 ID입니다.
database_name	char(50)	데이터베이스의 이름입니다.
status	char	세션의 상태입니다. 가능한 값은 `active`, `timed out` 및 `closed`입니다.
session_timeout	정수	세션이 시간 초과되기 전에 비활성 또는 유휴 상태로 유지되는 최대 시간(초)입니다. 0은 시간 제한이 설정되지 않았음을 나타냅니다.
start_time	타임스탬프	연결이 설정된 시점의 타임스탬프입니다.
end_time	타임스탬프	연결이 중지된 시점의 타임스탬프입니다.

예제

다음은 SYS_SESSION_HISTORY의 샘플 출력입니다.

```
select * from sys_session_history;
 user_id | session_id |   database_name  | status | session_timeout |
 start_time         |          end_time
---------+------------+------------------+--------+-----------------
+----------------------------+----------------------------
       1 | 1073971370 | dev              | closed |        0        | 2023-07-17
 15:50:12.030104 | 2023-07-17 15:50:12.123218
       1 | 1073979694 | dev              | closed |        0        | 2023-07-17
 15:50:24.117947 | 2023-07-17 15:50:24.131859
       1 | 1073873049 | dev              | closed |        0        | 2023-07-17
 15:49:29.067398 | 2023-07-17 15:49:29.070294
       1 | 1073873086 | database18127a4a | closed |        0        | 2023-07-17
 15:49:29.119018 | 2023-07-17 15:49:29.125925
       1 | 1073832112 | dev              | closed |        0        | 2023-07-17
 15:49:29.164688 | 2023-07-17 15:49:29.179631
       1 | 1073987697 | dev              | closed |        0        | 2023-07-17
 15:49:29.26749  | 2023-07-17 15:49:29.273034
       1 | 1073922429 | dev              | closed |        0        | 2023-07-17
 15:49:33.35315  | 2023-07-17 15:49:33.367499
       1 | 1073766783 | dev              | closed |        0        | 2023-07-17
 15:49:45.38237  | 2023-07-17 15:49:45.396902
       1 | 1073807506 | dev              | active |        0        | 2023-07-17
 15:51:48        |
```

SYS_SPATIAL_SIMPLIFY

COPY 명령을 사용하여 단순화된 공간 지오메트리 객체에 대한 정보를 가져오기 위해 시스템 뷰 SYS_SPATIAL_SIMPLIFY를 쿼리할 수 있습니다. shapefile에서 COPY를 사용할 때 SIMPLIFY `tolerance`, SIMPLIFY AUTO 및 SIMPLIFY AUTO `max_tolerance` 수집 옵션을 지정할 수 있습니다. 단순화 결과는 SYS_SPATIAL_SIMPLIFY 시스템 뷰에 요약되어 있습니다.

SIMPLIFY AUTO `max_tolerance`가 설정되면 이 뷰에는 최대 크기를 초과한 각 지오메트리에 대한 행이 포함됩니다. SIMPLIFY `tolerance`가 설정되면 전체 COPY 작업에 대한 하나의 행이 저장됩니다. 이 행은 COPY 쿼리 ID와 지정된 단순화 허용치를 참조합니다.

shapefile 로드에 대한 자세한 내용은 Amazon Redshift에 shapefile 로드를 참조하세요.

SYS_SPATIAL_SIMPLIFY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
query_id	bigint	이 행을 생성한 쿼리(COPY 명령)의 ID입니다.
line_number	bigint	COPY SIMPLIFY AUTO 옵션이 지정되면 이 값은 shapefile에서 단순화된 레코드의 레코드 번호입니다.
maximum_tolerance	double precision	COPY 명령에 지정된 거리 허용치 값입니다. 이는 SIMPLIFY AUTO 옵션을 사용하는 최대 허용치 값이거나 SIMPLIFY 옵션을 사용하는 고정 허용치 값입니다.
initial_size	bigint	단순화 전 GEOMETRY 데이터 값의 크기(바이트)입니다.
simplified	char(1)	COPY SIMPLIFY AUTO 옵션이 지정된 경우 지오메트리가 성공적으로 단순화되면 t이고, 그렇지 않으면 f입니다. 주어진 최대 허용치로 단순화한 후에도 지오메트리의 크기가 여전히 최대 지오메트리 크기보다 큰 경우 지오메트리가 성공적으로 단순화되지 않을 수 있습니다.
final_size	bigint	COPY SIMPLIFY AUTO 옵션이 지정된 경우 단순화 후 지오메트리의 크기(바이트)입니다.
final_tolerance	double precision	단순화를 위해 선택한 최종 허용 오차입니다.

샘플 쿼리

다음 쿼리는 COPY가 단순화한 레코드 목록을 반환합니다.

```
SELECT * FROM sys_spatial_simplify;
```

```
 query_id | line_number | maximum_tolerance | initial_size | simplified | final_size |
  final_tolerance
----------+-------------+-------------------+--------------+------------+------------
+----------------------
    20    |     1184704 |                -1 |      1513736 | t          |    1008808 |
  1.276386653895e-05
    20    |     1664115 |                -1 |      1233456 | t          |    1023584 |
 6.11707814796635e-06
```

SYS_STREAM_SCAN_ERRORS

스트리밍 수집을 통해 로드된 레코드의 오류를 기록합니다.

SYS_STREAM_SCAN_ERRORS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
external_schema_name	character (128)	Kinesis 스트림 또는 Amazon MSK 주제 스키마의 이름입니다. 대소문자를 구분합니다.
stream_name	character (255)	스트림 또는 주제의 이름입니다. 대소문자를 구분합니다.
mv_name	character (128)	구체화된 뷰의 이름입니다. 없는 경우 비어 있습니다. 대소문자를 구분합니다.
transaction_id	bigint	트랜잭션 ID.
query_id	bigint	쿼리 ID.

열 명칭	데이터 유형	설명
stream_timestamp_type	character (1)	스트림 타임스탬프의 유형입니다. 대소문자를 구분합니다.
stream_timestamp	시간대 미포함 TIMESTAM	레코드가 도착한 시간입니다.
record_time	시간대 미포함 TIMESTAM	오류 메시지가 기록된 시간입니다.
partition_id	character (128)	파티션/샤드 ID입니다. 대소문자를 구분합니다.
position	character (128)	레코드의 위치입니다. Kinesis의 시퀀스 번호 또는 Amazon MSK의 오프셋에 해당합니다. 대소문자를 구분합니다.
error_code	정수	오류 코드입니다.
error_reason	character (128)	오류 이유입니다. 대소문자를 구분합니다.

SYS_STREAM_SCAN_STATES

스트리밍 수집을 통해 로드된 레코드의 스캔 상태를 기록합니다.

SYS_STREAM_SCAN_STATES는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
external_schema_name	character (128)	외부 스키마 이름입니다. 대소문자를 구분합니다.
stream_name	character (255)	스트림 이름입니다. 대소문자를 구분합니다.
mv_name	character (128)	구체화된 뷰의 이름입니다. 없는 경우 비어 있습니다. 대소문자를 구분합니다.
transaction_id	bigint	트랜잭션 ID.
query_id	bigint	쿼리 ID.
record_time	시간대 미포함 TIMESTAM	데이터가 기록된 시간입니다.
partition_id	character (128)	파티션 또는 샤드 ID입니다. 대소문자를 구분합니다.
latest_position	character (128)	배치에서 읽은 마지막 레코드의 위치입니다. Kinesis의 시퀀스 번호 또는 Amazon MSK의 오프셋에 해당합니다. 대소문자를 구분합니다.
scanned_rows	bigint	배치에서 스캔된 레코드 수입니다.
skipped_rows	bigint	배치에서 건너뛰었던 레코드 수입니다.
scanned_bytes	bigint	배치에서 스캔된 바이트 수입니다.

열 명칭	데이터 유형	설명
stream_record_time_min	시간대 미포함 TIMESTAM	배치에서 가장 먼저 도착한 레코드의 Kinesis 또는 Amazon MSK 도착 시간입니다.
stream_record_time_max	시간대 미포함 TIMESTAM	배치에서 가장 나중에 도착한 레코드의 Kinesis 또는 Amazon MSK 도착 시간입니다.

다음 쿼리는 특정 쿼리에 대한 스트림 및 주제 데이터를 보여줍니다.

```
select
 query_id,mv_name::varchar,external_schema_name::varchar,stream_name::varchar,sum(scanned_rows)
 total_records,
sum(scanned_bytes) total_bytes from sys_stream_scan_states where query in
 (5401180,8601939) group by 1,2,3,4;

  query_id  |    mv_name     | external_schema_name |   stream_name    | total_records |
  total_bytes
------------+----------------+----------------------+------------------+---------------
+----------------
 5401180    | kinesistest    | kinesis              | kinesisstream    |    1493255696 |
 3209006490704
 8601939    | msktest        | msk                  | mskstream        |      14677023 |
   31056580668
(2 rows)
```

SYS_TRANSACTION_HISTORY

쿼리를 추적할 때 SYS_TRANSACTION_HISTORY를 사용하면 트랜잭션의 세부 정보를 볼 수 있습니다.

SYS_TRANSACTION_HISTORY는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	항목을 생성한 사용자의 ID.
transaction_id	bigint	트랜잭션의 ID입니다.
isolation_level	텍스트	트랜잭션의 격리 수준입니다. 가능한 값은 `Serializable` 및 `Snapshot Isolation` 입니다.
status	텍스트	트랜잭션의 상태입니다. 가능한 상태는 `committed` 및 `rolledback` 입니다.
transaction_start_time	타임스탬프	트랜잭션의 시작 시간입니다.
commit_start_time	타임스탬프	커밋의 시작 시간입니다.
commit_end_time	타임스탬프	커밋의 종료 시간입니다.
blocks_committed	bigint	이번 커밋에 포함하여 작성해야 했던 블록의 수입니다.
undo_transaction_id	bigint	트랜잭션이 실행 취소된 경우 실행 취소 트랜잭션의 ID입니다. 그렇지 않으면 이 값은 -1입니다.

샘플 쿼리

```
select * from sys_transaction_history order by transaction_start_time desc;

 user_id | transaction_id | isolation_level |   status   |   transaction_start_time
   |     commit_start_time      |      commit_end_time       | blocks_committed |
 undo_transaction_id
```

```
---------+----------------+-----------------+------------+----------------------------
+----------------------------+----------------------------+------------------
+---------------------
     100 |           1310 | Serializable    | committed  | 2023-08-27 21:03:11.822205 |
 2023-08-28 21:03:11.825287 | 2023-08-28 21:03:11.854883 |               17 |
       -1
     101 |           1345 | Serializable    | committed  | 2023-08-27 21:03:12.000278 |
 2023-08-28 21:03:12.003736 | 2023-08-28 21:03:12.030061 |               17 |
       -1
     102 |           1367 | Serializable    | committed  | 2023-08-27 21:03:12.1532   |
 2023-08-28 21:03:12.156124 | 2023-08-28 21:03:12.186468 |               17 |
       -1
     100 |           1370 | Serializable    | committed  | 2023-08-27 21:03:12.199316 |
 2023-08-28 21:03:12.204854 | 2023-08-28 21:03:12.238186 |               24 |
       -1
     100 |           1408 | Serializable    | committed  | 2023-08-27 21:03:53.891107 |
 2023-08-28 21:03:53.894825 | 2023-08-28 21:03:53.927465 |               17 |
       -1
     100 |           1409 | Serializable    | rolledback | 2023-08-27 21:03:53.936431 |
 2000-01-01 00:00:00        | 2023-08-28 21:04:08.712532 |                0 |
      1409
     101 |           1415 | Serializable    | committed  | 2023-08-27 21:04:24.283188 |
 2023-08-28 21:04:24.289196 | 2023-08-28 21:04:24.374318 |               25 |
       -1
     101 |           1416 | Serializable    | committed  | 2023-08-27 21:04:24.38818  |
 2023-08-28 21:04:24.391688 | 2023-08-28 21:04:24.415135 |               17 |
       -1
     100 |           1417 | Serializable    | rolledback | 2023-08-27 21:04:24.424252 |
 2000-01-01 00:00:00        | 2023-08-28 21:04:28.354826 |                0 |
      1417
     101 |           1418 | Serializable    | rolledback | 2023-08-27 21:04:24.425195 |
 2000-01-01 00:00:00        | 2023-08-28 21:04:28.680355 |                0 |
      1418
     100 |           1420 | Serializable    | committed  | 2023-08-27 21:04:28.697607 |
 2023-08-28 21:04:28.702374 | 2023-08-28 21:04:28.735541 |               23 |
       -1
     101 |           1421 | Serializable    | committed  | 2023-08-27 21:04:28.744854 |
 2023-08-28 21:04:28.749344 | 2023-08-28 21:04:28.779029 |               23 |
       -1
     101 |           1423 | Serializable    | committed  | 2023-08-27 21:04:28.78942  |
 2023-08-28 21:04:28.791556 | 2023-08-28 21:04:28.817485 |               16 |
       -1
```

```
    100 |           1430 | Serializable    | committed  | 2023-08-27 21:04:28.917788 |
2023-08-28 21:04:28.919993 | 2023-08-28 21:04:28.944812 |               16 |
       -1
    102 |           1494 | Serializable    | committed  | 2023-08-27 21:04:37.029058 |
2023-08-28 21:04:37.033137 | 2023-08-28 21:04:37.062001 |               16 |
       -1
```

SYS_UDF_LOG

사용자 정의 함수(UDF)를 실행하는 중에 생성되는 시스템 정의 오류 및 경고 메시지를 기록합니다.

SYS_UDF_LOG는 수퍼유저에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
query_id	bigint	쿼리 식별자입니다.
function_name	텍스트	사용자 정의 함수의 이름입니다.
record_time	타임스탬프	기록이 생성된 시간입니다.
SEQUENCE	정수	단일 로그 메시지의 시퀀스입니다.
message	텍스트	로그 메시지 텍스트입니다.

샘플 쿼리

다음 예는 UDF가 시스템 정의 오류를 처리하는 방법을 보여 줍니다. 첫 번째 블록은 인수의 역을 반환하는 UDF 함수의 정의를 보여 줍니다. 함수를 실행하고 인수로 0을 제공하면 함수가 오류를 반환합니다. 마지막 문은 SYS_UDF_LOG에 로깅된 오류 메시지를 반환합니다.

```
-- Create a function to find the inverse of a number.
CREATE OR REPLACE FUNCTION f_udf_inv(a int)
```

```
RETURNS float

IMMUTABLE AS $$return 1/a

$$ LANGUAGE plpythonu;

-- Run the function with 0 to create an error.
Select f_udf_inv(0);

-- Query SYS_UDF_LOG to view the message.
Select query_id, record_time, message::varchar from sys_udf_log;


query_id    |    record_time              |                 message
----------+----------------------------
+-------------------------------------------------------
2211        | 2023-08-23 15:53:11.360538 |  ZeroDivisionError: integer division or
 modulo by zero line 2, in f_udf_inv\n return 1/a\n
```

다음 예는 UDF에 로깅 및 경고 메시지를 추가하므로 0으로 나누기 연산은 오류 메시지와 함께 중단되는 대신 경고 메시지를 생성합니다.

```
-- Create a function to find the inverse of a number and log a warning if you input 0.
CREATE OR REPLACE FUNCTION f_udf_inv_log(a int)
  RETURNS float IMMUTABLE
 AS $$
  import logging
  logger = logging.getLogger() #get root logger
  if a==0:
    logger.warning('You attempted to divide by zero.\nReturning zero instead of error.
\n')
    return 0
  else:
     return 1/a
$$ LANGUAGE plpythonu;

-- Run the function with 0 to trigger the warning.
Select f_udf_inv_log(0);

-- Query SYS_UDF_LOG to view the message.
Select query_id, record_time, message::varchar from sys_udf_log;

 query_id |        record_time         |                                    message
```

```
----------+----------------------------
+-------------------------------------------------------------------------------
     0    | 2023-08-23 16:10:48.833503 | WARNING: You attempted to divide by zero.
\nReturning zero instead of error.\n
```

SYS_UNLOAD_DETAIL

SYS_UNLOAD_DETAIL을 사용하여 UNLOAD 작업의 세부 정보를 봅니다. UNLOAD 문에서 생성되는 파일마다 행 1개를 기록합니다. 예를 들어 UNLOAD를 실행하여 파일 12개가 생성된다면 SYS_UNLOAD_DETAIL에 포함되는 행의 수도 12개입니다.

SYS_UNLOAD_DETAIL은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	항목을 생성한 사용자의 식별자입니다.
query_id	정수	UNLOAD 명령의 쿼리 식별자입니다.
session_id	정수	쿼리 문과 연결된 프로세스 ID입니다.
transaction_id	bigint	쿼리 문과 연결된 트랜잭션의 ID입니다.
file_name	character (1280)	파일의 전체 Amazon S3 객체 경로.
start_time	타임스탬프	트랜잭션이 시작한 시간입니다.
end_time	타임스탬프	트랜잭션이 완료된 시간입니다.

열 명칭	데이터 유형	설명
line_count	bigint	파일로 언로드되는 라인(행)의 수입니다.
transfer_size	bigint	전송되는 바이트 수입니다.
file_format	character (10)	언로드된 파일의 파일 형식입니다.

샘플 쿼리

다음 쿼리는 unload 명령의 형식, 행 및 파일 수를 비롯하여 언로드된 쿼리에 대한 세부 정보를 보여줍니다.

```
select query_id, substring(file_name, 0, 50), transfer_size, file_format from
 sys_unload_detail;
```

샘플 출력은 다음과 같습니다.

```
 query_id |                     substring                                | transfer_size
 | file_format
----------+----------------------------------------------------------------
+---------------+-------------
     9272 | s3://amzn-s3-demo-bucket/my_unload_doc_venue0000_part_00.gz  |
 395886 | Text
     9272 | s3://amzn-s3-demo-bucket/my_unload_doc_venue0001_part_00.gz  |
 406444 | Text
     9272 | s3://amzn-s3-demo-bucket/my_unload_doc_venue0002_part_00.gz  |
 409431 | Text
     9272 | s3://amzn-s3-demo-bucket/my_unload_doc_venue0003_part_00.gz  |
 403051 | Text
     9272 | s3://amzn-s3-demo-bucket/my_unload_doc_venue0004_part_00.gz  |
 413592 | Text
     9272 | s3://amzn-s3-demo-bucket/my_unload_doc_venue0005_part_00.gz  |
 395689 | Text
(6 rows)
```

SYS_UNLOAD_HISTORY

SYS_UNLOAD_HISTORY를 사용하여 UNLOAD 명령의 세부 정보를 봅니다. 각 행은 일부 필드에 대한 누적 통계가 있는 UNLOAD 명령을 나타냅니다. 여기에는 실행 중인 UNLOAD 명령과 완료된 UNLOAD 명령이 모두 포함됩니다.

SYS_UNLOAD_HISTORY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	언로드를 제출한 사용자의 식별자입니다.
query_id	bigint	UNLOAD 명령의 쿼리 식별자입니다.
transaction_id	bigint	트랜잭션 식별자입니다.
session_id	정수	언로드를 실행하는 프로세스의 프로세스 식별자입니다.
database_name	텍스트	작업이 실행되었을 때 사용자가 연결된 데이터베이스의 이름입니다.
status	텍스트	UNLOAD 명령의 상태입니다. 유효한 값에는 `running`, `completed` , `aborted` 및 `unknown` 등이 있습니다.
start_time	타임스탬프	언로드가 시작된 시간입니다.
end_time	타임스탬프	언로드가 완료된 시간입니다.
duration	bigint	UNLOAD 명령에 소요된 시간(마이크로초)입니다.

열 명칭	데이터 유형	설명
file_format	텍스트	출력 파일의 파일 형식입니다.
compression_type	텍스트	압축 유형입니다.
unloaded_location	텍스트	언로드된 파일의 Amazon S3 위치입니다.
unloaded_rows	bigint	행 수입니다.
unloaded_files_count	bigint	출력 파일의 파일 수입니다.
unloaded_files_size	bigint	출력 파일의 파일 크기입니다.
error_message	텍스트	UNLOAD 명령의 오류 메시지입니다.

샘플 쿼리

다음 쿼리는 unload 명령의 형식, 행 및 파일 수를 비롯하여 언로드된 쿼리에 대한 세부 정보를 보여줍니다.

```
SELECT query_id,
       file_format,
       start_time,
       duration,
       unloaded_rows,
       unloaded_files_count
FROM sys_unload_history
ORDER BY query_id,
file_format limit 100;
```

샘플 출력은 다음과 같습니다.

```
 query_id | file_format |         start_time         | duration | unloaded_rows |
 unloaded_files_count
----------+-------------+----------------------------+----------+---------------
+----------------------
```

```
  527067 | Text        | 2022-02-09 05:18:35.844452 |  5932478 |           10 |
           1
```

SYS_USERLOG

다음과 같이 데이터베이스 사용자의 변경 사항에 대한 세부 정보를 기록합니다.

- 사용자 생성
- 사용자 삭제
- 사용자 변경(이름 변경)
- 사용자 변경(속성 변경)

이 뷰를 쿼리하여 서버리스 작업 그룹 및 프로비저닝된 클러스터에 대한 정보를 볼 수 있습니다.

SYS_USERLOG는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	언로드를 제출한 사용자의 식별자입니다.
user_name	character(50)	변경 사항이 적용되는 사용자의 이름입니다.
original_user_name	character(50)	이름 변경 작업의 원래 사용자 이름입니다. 이 필드는 다른 모든 작업에 대해 비어 있습니다.
작업	character(10)	발생한 작업입니다. 유효한 값은 변경, 생성, 삭제, 이름 바꾸기입니다.
has_create_db_privs	정수	true(값 1)이면 사용자에게 데이터베이스 생성 권한이 있는 것을 의미합니다.

열 명칭	데이터 유형	설명
is_superuser	정수	true(값 1)이면 사용자가 시스템 카탈로그를 업데이트할 수 있습니다.
has_update_catalog_privs	정수	true(값 1)이면 사용자가 시스템 카탈로그를 업데이트할 수 있습니다.
password_expiration	타임스탬프	암호 만료 날짜입니다.
session_id	정수	프로세스 ID.
transaction_id	bigint	트랜잭션 ID.
record_time	타임스탬프	쿼리가 시작된 시간(UTC 기준)입니다.

샘플 쿼리

다음은 사용자 작업 4개를 실행한 후 SYS_USERLOG 뷰에 대한 쿼리를 실행하는 예입니다.

```
CREATE USER userlog1 password 'Userlog1';
ALTER USER userlog1 createdb createuser;
ALTER USER userlog1 rename  to userlog2;
DROP user userlog2;

SELECT user_id, user_name, original_user_name, action, has_create_db_privs,
 is_superuser from SYS_USERLOG order by record_time desc;
```

```
user_id |  user_name | original_user_name |  action | has_create_db_privs |
 is_superuser
--------+------------+--------------------+---------+---------------------+------------
    108 |  userlog2  |                    | drop    |                   1 |   1
    108 |  userlog2  |     userlog1       | rename  |                   1 |   1
    108 |  userlog1  |                    | alter   |                   1 |   1
    108 |  userlog1  |                    | create  |                   0 |   0
 (4 rows)
```

SYS_VACUUM_HISTORY

SYS_VACUUM_HISTORY를 사용하여 Vacuum 쿼리의 세부 정보를 볼 수 있습니다. 명령에 대한 자세한 내용은 VACUUM을 참조하세요.

SYS_VACUUM_HISTORY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
user_id	정수	쿼리를 시작한 사용자의 ID입니다.
transaction_id	long	VACUUM 문의 트랜잭션 ID.
query_id	long	VACUUM 문의 쿼리 식별자입니다. 이 테이블을 SYS_QUERY_DETAIL 뷰로 조인하면 임의의 VACUUM 트랜잭션에서 실행된 SQL 문을 개별적으로 확인할 수 있습니다. 전체 데이터베이스를 정리하는 경우에는 각 테이블이 별도의 트랜잭션으로 정리됩니다. 자동화된 VACUUM 작업의 경우 이 값은 null입니다.
database_name	텍스트	데이터베이스의 이름입니다.
schema_name	텍스트	스키마의 이름입니다.
table_name	텍스트	테이블의 이름
table_id	정수	테이블의 ID입니다.

열 명칭	데이터 유형	설명
vacuum_type	character	VACUUM 작업의 유형입니다. 가능한 값은 다음과 같습니다. • **Delete** • **Sort** • **Reindex** • **Recluster** • **Full** vacuum 형식에 대한 자세한 내용은 VACUUM을 참조하세요.
is_automatic	boolean	자동 vacuum인 경우 `true`입니다. 그렇지 않을 경우 `false`입니다.
status	character	vacuum 작업의 일환으로 수행되고 있는 현재 활동의 설명: • 초기화 • 정렬 • 병합 • 삭제 • Select • Failed • 완료 • 건너뜀 • INTERLEAVED SORTKEY 순서 구축
start_time	타임스탬프	vacuum 작업이 시작된 시간입니다.

열 명칭	데이터 유형	설명
end_time	타임스탬프	vacuum 작업이 종료된 시간입니다. 작업이 진행 중인 경우 이 필드는 비어 있습니다.
record_time	타임스탬프	SYS_VACUUM_HISTORY에 vacuum 작업이 기록된 시간입니다.
기간	정수	vacuum 작업의 시작과 종료 사이의 시간(마이크로초)입니다. vacuum 작업이 진행 중인 경우 이 필드는 비어 있습니다.
rows_before_vacuum	bigint	테이블의 실제 행 수 + 삭제되었지만 아직 디스크에 저장되어 있는(정리 대기 중인) 모든 행.
size_before_vacuum	정수	vacuum 작업이 시작되기 전 테이블의 크기(MB)입니다.
reclaimable_rows	bigint	vacuum 작업이 시작하기 전에 회수할 것으로 예상되는 행 수입니다.
reclaimed_rows	bigint	vacuum 작업으로 회수된 행 수입니다.
reclaimed_blocks	bigint	vacuum 작업으로 회수한 블록 수입니다.
sortedrows_before_vacuum	정수	vacuum 작업이 시작되기 전에 테이블에서 정렬된 행의 수입니다.

열 명칭	데이터 유형	설명
sortedrows_after_vacuum	정수	vacuum 작업이 완료된 후 테이블에서 정렬된 행의 추가 개수입니다. `sortedrows_before_vacuum` 에는 다음과 같이 계산된 행은 포함되지 않습니다.

SYS 모니터링 뷰로 마이그레이션하기 위한 시스템 뷰 매핑

Amazon Redshift 프로비저닝 클러스터를 Amazon Redshift Serverless로 마이그레이션할 때 모니터링 또는 진단 쿼리는 프로비저닝된 클러스터에서만 사용할 수 있는 시스템 뷰를 참조할 수 있습니다. SYS 모니터링 뷰를 사용하도록 쿼리를 업데이트할 수 있습니다. 이 페이지는 쿼리를 업데이트할 때 참조할 수 있도록 프로비저닝 전용 뷰와 SYS 뷰 매핑을 제공합니다.

주제

- SYS_QUERY_HISTORY
- SYS_QUERY_DETAIL
- SYS_RESTORE_LOG
- SYS_RESTORE_STATE
- SYS_TRANSACTION_HISTORY
- SYS_QUERY_TEXT
- SYS_CONNECTION_LOG
- SYS_SESSION_HISTORY
- SYS_LOAD_DETAIL
- SYS_LOAD_HISTORY
- SYS_LOAD_ERROR_DETAIL
- SYS_UNLOAD_HISTORY
- SYS_UNLOAD_DETAIL
- SYS_COPY_REPLACEMENTS
- SYS_DATASHARE_USAGE_CONSUMER
- SYS_DATASHARE_USAGE_PRODUCER

- SYS_DATASHARE_CROSS_REGION_USAGE
- SYS_DATASHARE_CHANGE_LOG
- SYS_EXTERNAL_QUERY_DETAIL
- SYS_EXTERNAL_QUERY_ERROR
- SYS_VACUUM_HISTORY
- SYS_ANALYZE_HISTORY
- SYS_ANALYZE_COMPRESSION_HISTORY
- SYS_MV_REFRESH_HISTORY
- SYS_MV_STATE
- SYS_PROCEDURE_CALL
- SYS_PROCEDURE_MESSAGES
- SYS_UDF_LOG
- SYS_USERLOG
- SYS_SCHEMA_QUOTA_VIOLATIONS
- SYS_SPATIAL_SIMPLIFY

SYS_QUERY_HISTORY

다음 테이블의 열 중 일부 또는 전부는 SYS_QUERY_HISTORY에도 정의되어 있습니다.

- STL_DDLTEXT
- STL_ERROR
- STL_QUERY
- STL_UTILITYTEXT
- STL_WLM_QUERY
- STV_INFLIGHT
- STV_RECENTS
- STV_WLM_QUERY_STATE
- SVL_COMPILE
- SVL_MULTI_STATEMENT_VIOLATIONS
- SVL_QLOG

- SVL_QUERY_QUEUE_INFO
- SVL_STATEMENTTEXT
- SVL_TERMINATE

SYS_QUERY_DETAIL

다음 테이블의 열 중 일부 또는 전부는 SYS_QUERY_DETAIL에도 정의되어 있습니다.

- STL_AGGR
- STL_ALERT_EVENT_LOG
- STL_BCAST
- STL_DELETE
- STL_DIST
- STL_EXPLAIN
- STL_HASH
- STL_HASHJOIN
- STL_INSERT
- STL_LIMIT
- STL_MERGE
- STL_MERGEJOIN
- STL_NESTLOOP
- STL_PARSE
- STL_PLAN_INFO
- STL_PROJECT
- STL_QUERY_METRICS
- STL_RETURN
- STL_SAVE
- STL_SCAN
- STL_SORT
- STL_STREAM_SEGS
- STL_UNIQUE

- STL_WINDOW
- STV_EXEC_STATE
- STV_QUERY_METRICS
- SVCS_QUERY_SUMMARY
- SVL_QUERY_METRICS
- SVL_QUERY_METRICS_SUMMARY
- SVL_QUERY_REPORT
- SVL_QUERY_SUMMARY
- SVV_QUERY_STATE

SYS_RESTORE_LOG

다음 테이블의 열 중 일부 또는 전부는 SYS_RESTORE_LOG에도 정의되어 있습니다.

- SVL_RESTORE_ALTER_TABLE_PROGRESS

SYS_RESTORE_STATE

다음 테이블의 열 중 일부 또는 전부는 SYS_RESTORE_STATE에도 정의되어 있습니다.

- STV_XRESTORE_ALTER_QUEUE_STATE

SYS_TRANSACTION_HISTORY

다음 테이블의 열 중 일부 또는 전부는 SYS_TRANSACTION_HISTORY에도 정의되어 있습니다.

- STL_COMMIT_STATS
- STL_TR_CONFLICT
- STL_UNDONE

SYS_QUERY_TEXT

다음 테이블의 열 중 일부 또는 전부는 SYS_QUERY_TEXT에도 정의되어 있습니다.

- STL_QUERYTEXT

SYS_CONNECTION_LOG

다음 테이블의 열 중 일부 또는 전부는 SYS_CONNECTION_LOG에도 정의되어 있습니다.

- STL_CONNECTION_LOG

SYS_SESSION_HISTORY

다음 테이블의 열 중 일부 또는 전부는 SYS_SESSION_HISTORY에도 정의되어 있습니다.

- STL_SESSIONS
- STL_RESTARTED_SESSIONS
- STV_SESSIONS

SYS_LOAD_DETAIL

다음 테이블의 열 중 일부 또는 전부는 SYS_LOAD_DETAIL에도 정의되어 있습니다.

- STL_LOAD_COMMITS

SYS_LOAD_HISTORY

다음 테이블의 열 중 일부 또는 전부는 SYS_LOAD_HISTORY에도 정의되어 있습니다.

- STL_LOAD_COMMITS

SYS_LOAD_ERROR_DETAIL

다음 테이블의 열 중 일부 또는 전부는 SYS_LOAD_ERROR_DETAIL에도 정의되어 있습니다.

- STL_LOADERROR_DETAIL
- STL_LOAD_ERRORS

SYS_UNLOAD_HISTORY

다음 테이블의 열 중 일부 또는 전부는 SYS_UNLOAD_HISTORY에도 정의되어 있습니다.

- STL_UNLOAD_LOG

SYS_UNLOAD_DETAIL

다음 테이블의 열 중 일부 또는 전부는 SYS_UNLOAD_DETAIL에도 정의되어 있습니다.

- STL_UNLOAD_LOG

SYS_COPY_REPLACEMENTS

다음 테이블의 열 중 일부 또는 전부는 SYS_COPY_REPLACEMENTS에도 정의되어 있습니다.

- STL_REPLACEMENTS

SYS_DATASHARE_USAGE_CONSUMER

다음 테이블의 열 중 일부 또는 전부는 SYS_DATASHARE_USAGE_CONSUMER에도 정의되어 있습니다.

- SVL_DATASHARE_USAGE_CONSUMER

SYS_DATASHARE_USAGE_PRODUCER

다음 테이블의 열 중 일부 또는 전부는 SYS_DATASHARE_USAGE_PRODUCER에도 정의되어 있습니다.

- SVL_DATASHARE_USAGE_PRODUCER

SYS_DATASHARE_CROSS_REGION_USAGE

다음 테이블의 열 중 일부 또는 전부는 SYS_DATASHARE_CROSS_REGION_USAGE에도 정의되어 있습니다.

- SVL_DATASHARE_CROSS_REGION_USAGE

SYS_DATASHARE_CHANGE_LOG

다음 테이블의 열 중 일부 또는 전부는 SYS_DATASHARE_CHANGE_LOG에도 정의되어 있습니다.

- SVL_DATASHARE_CHANGE_LOG

SYS_EXTERNAL_QUERY_DETAIL

다음 테이블의 열 중 일부 또는 전부는 SYS_EXTERNAL_QUERY_DETAIL에도 정의되어 있습니다.

- SVL_FEDERATED_QUERY
- SVL_S3LIST
- SVL_S3QUERY
- SVL_S3QUERY_SUMMARY

SYS_EXTERNAL_QUERY_ERROR

다음 테이블의 열 중 일부 또는 전부는 SYS_EXTERNAL_QUERY_ERROR에도 정의되어 있습니다.

- SVL_SPECTRUM_SCAN_ERROR

SYS_VACUUM_HISTORY

다음 테이블의 열 중 일부 또는 전부는 SYS_VACUUM_HISTORY에도 정의되어 있습니다.

- STL_VACUUM
- SVL_VACUUM_PERCENTAGE
- SVV_VACUUM_PROGRESS
- SVV_VACUUM_SUMMARY

SYS_ANALYZE_HISTORY

다음 테이블의 열 중 일부 또는 전부는 SYS_ANALYZE_HISTORY에도 정의되어 있습니다.

- STL_ANALYZE

SYS_ANALYZE_COMPRESSION_HISTORY

다음 테이블의 열 중 일부 또는 전부는 SYS_ANALYZE_COMPRESSION_HISTORY에도 정의되어 있습니다.

- STL_ANALYZE_COMPRESSION

SYS_MV_REFRESH_HISTORY

다음 테이블의 열 중 일부 또는 전부는 SYS_MV_REFRESH_HISTORY에도 정의되어 있습니다.

- SVL_MV_REFRESH_STATUS

SYS_MV_STATE

다음 테이블의 열 중 일부 또는 전부는 SYS_MV_STATE에도 정의되어 있습니다.

- STL_MV_STATE

SYS_PROCEDURE_CALL

다음 테이블의 열 중 일부 또는 전부는 SYS_PROCEDURE_CALL에도 정의되어 있습니다.

- SVL_STORED_PROC_CALL

SYS_PROCEDURE_MESSAGES

다음 테이블의 열 중 일부 또는 전부는 SYS_PROCEDURE_MESSAGES에도 정의되어 있습니다.

- SVL_STORED_PROC_MESSAGES

SYS_UDF_LOG

다음 테이블의 열 중 일부 또는 전부는 SYS_UDF_LOG에도 정의되어 있습니다.

- SVL_UDF_LOG

SYS_USERLOG

다음 테이블의 열 중 일부 또는 전부는 SYS_USERLOG에도 정의되어 있습니다.

- STL_USERLOG

SYS_SCHEMA_QUOTA_VIOLATIONS

다음 테이블의 열 중 일부 또는 전부는 SYS_SCHEMA_QUOTA_VIOLATIONS에도 정의되어 있습니다.

- STL_SCHEMA_QUOTA_VIOLATIONS

SYS_SPATIAL_SIMPLIFY

다음 테이블의 열 중 일부 또는 전부는 SYS_SPATIAL_SIMPLIFY에도 정의되어 있습니다.

- SVL_SPATIAL_SIMPLIFY

시스템 모니터링(프로비저닝만 해당)

프로비저닝된 클러스터에서 다음과 같은 시스템 테이블과 뷰를 쿼리할 수 있습니다. STL 및 STV 테이블 및 뷰에는 몇몇 시스템 테이블에서 찾을 수 있는 데이터 하위 집합이 포함됩니다. 여기에서는 이들 테이블에서 찾을 수 있는 일반적으로 쿼리되는 데이터에 대한 보다 빠르고 쉬운 액세스를 제공합니다.

SVCS 뷰는 동시성 확장 클러스터와 기본 클러스터 모두의 쿼리에 대한 세부 정보를 제공합니다. SVL 뷰는 SVL_STATEMENTTEXT를 제외하고 기본 클러스터에서 실행되는 쿼리에 대해서만 정보를 제공합니다. SVL_STATEMENTTEXT는 기본 클러스터뿐만 아니라 동시성 규모 조정 클러스터에서 실행되는 쿼리에 대한 정보를 포함할 수 있습니다.

주제

- 로깅을 위한 STL 뷰
- 스냅샷 데이터를 위한 STV 테이블
- 기본 및 동시성 조정 클러스터에 대한 SVCS 뷰
- 기본 클러스터의 SVL 뷰

로깅을 위한 STL 뷰

STL 시스템 뷰는 Amazon Redshift 로그 파일에서 생성되어 시스템 이력 정보를 제공합니다.

로그 파일은 데이터 웨어하우스 클러스터의 모든 노드에 상주합니다. STL 뷰가 이러한 로그 파일에서 정보를 가져와 시스템 관리자가 사용할 수 있는 뷰로 형식을 변경합니다.

로그 보존 - STL 시스템 뷰는 7일간의 로그 기록을 보존합니다. 로그 보존은 모든 클러스터 크기 및 노드 유형에 대해 보장되며 클러스터 워크로드의 변화에 영향을 받지 않습니다. 또한 로그 보존은 클러스터가 일시 중지된 경우와 같은 클러스터 상태의 영향을 받지 않습니다. 클러스터가 새 클러스터인 경우에만 7일 미만의 로그 기록이 있습니다. 로그 보존을 위해 별도의 조치를 취할 필요는 없지만 로그 데이터를 7일 이상 보관하고 싶다면 정기적으로 다른 테이블에 복사하거나 Amazon S3로 언로드해야 합니다.

주제

- STL_LOAD_ERRORS
- STL_LOADERROR_DETAIL
- STL_MERGE
- STL_MERGEJOIN
- STL_MV_STATE
- STL_NESTLOOP
- STL_PARSE
- STL_PLAN_INFO
- STL_PROJECT
- STL_QUERY
- STL_QUERY_METRICS
- STL_QUERYTEXT
- STL_REPLACEMENTS
- STL_RESTARTED_SESSIONS
- STL_RETURN
- STL_S3CLIENT
- STL_S3CLIENT_ERROR
- STL_SAVE
- STL_SCAN
- STL_SCHEMA_QUOTA_VIOLATIONS
- STL_SESSIONS
- STL_SORT
- STL_SSHCLIENT_ERROR
- STL_STREAM_SEGS
- STL_TR_CONFLICT
- STL_UNDONE
- STL_UNIQUE
- STL_UNLOAD_LOG
- STL_USAGE_CONTROL
- STL_USERLOG

- STL_UTILITYTEXT
- STL_VACUUM
- STL_WINDOW
- STL_WLM_ERROR
- STL_WLM_RULE_ACTION
- STL_WLM_QUERY

STL_AGGR

쿼리의 집계 실행 단계를 분석합니다. 이 단계는 집계 함수와 GROUP BY 절을 실행하는 동안 발생합니다.

STL_AGGR은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_AGGR에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호

열 명칭	데이터 유형	설명
step	정수	실행된 쿼리 단계입니다.
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
tasknum	정수	단계 실행을 위해 할당된 쿼리 태스크 프로세스 수.
rows	bigint	처리된 총 행 수.
bytes	bigint	단계에서 출력되는 모든 행의 크기(바이트)
slots	정수	해시 버킷 수
occupied	정수	레코드가 저장되는 슬롯 수
maxlength	정수	가장 큰 슬롯의 크기
tbl	정수	테이블 ID.
is_diskbased	character(1)	true(t)인 경우 쿼리가 디스크 기반 작업으로 실행된 것을 의미합니다. false(f)인 경우 쿼리가 메모리에서 실행된 것을 의미합니다.
workmem	bigint	단계에 할당되는 유효 메모리의 바이트 수
type	character(6)	단계 유형. 유효한 값은 다음과 같습니다. • HASHED. 단계가 정렬 없이 분류된 집계를 사용한 것을 의미합니다. • PLAIN. 단계가 분류되지 않은 스칼라 집계를 사용한 것을 의미합니다. • SORTED. 단계가 정렬과 함께 분류된 집계를 사용한 것을 의미합니다.

열 명칭	데이터 유형	설명
크기 조정	정수	이 정보는 내부 전용입니다.
flushable	정수	이 정보는 내부 전용입니다.

샘플 쿼리

SLICE 1이고, TBL 239일 때 집계 실행 단계에 대한 정보를 반환합니다.

```
select query, segment, bytes, slots, occupied, maxlength, is_diskbased, workmem, type
from stl_aggr where slice=1 and tbl=239
order by rows
limit 10;
```

```
 query | segment | bytes |  slots  | occupied | maxlength | is_diskbased |  workmem  |
 type
-------+---------+-------+---------+----------+-----------+--------------+-----------
+--------
   562 |       1 |     0 | 4194304 |        0 |         0 | f            | 383385600 |
 HASHED
   616 |       1 |     0 | 4194304 |        0 |         0 | f            | 383385600 |
 HASHED
   546 |       1 |     0 | 4194304 |        0 |         0 | f            | 383385600 |
 HASHED
   547 |       0 |     8 |       0 |        0 |         0 | f            |         0 |
 PLAIN
   685 |       1 |    32 | 4194304 |        1 |         0 | f            | 383385600 |
 HASHED
   652 |       0 |     8 |       0 |        0 |         0 | f            |         0 |
 PLAIN
   680 |       0 |     8 |       0 |        0 |         0 | f            |         0 |
 PLAIN
   658 |       0 |     8 |       0 |        0 |         0 | f            |         0 |
 PLAIN
   686 |       0 |     8 |       0 |        0 |         0 | f            |         0 |
 PLAIN
   695 |       1 |    32 | 4194304 |        1 |         0 | f            | 383385600 |
 HASHED
(10 rows)
```

STL_ALERT_EVENT_LOG

쿼리 옵티마이저에서 성능 문제를 야기할 수 있는 조건이 식별되면 알림 메시지가 기록됩니다. STL_ALERT_EVENT_LOG 뷰는 쿼리 성능을 높일 수 있는 방법을 찾는 데 사용됩니다.

하나의 쿼리는 여러 세그먼트로 구성되며, 각각의 세그먼트는 하나 이상의 단계로 구성됩니다. 자세한 내용은 쿼리 처리 단원을 참조하십시오.

STL_ALERT_EVENT_LOG는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_ALERT_EVENT_LOG에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호
step	정수	실행된 쿼리 단계입니다.
pid	정수	쿼리 문 및 조각과 연결된 프로세스 ID. 동일한 쿼리가 다수의 조각에서 실행되는 경우에는 다수의 PID를 가질 수 있습니다.

열 명칭	데이터 유형	설명
xid	bigint	문에 연결된 트랜잭션 ID.
이벤트	character (1024)	알림 이벤트에 대한 설명
solution	character (1024)	권장 솔루션
event_time	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .

사용 노트

STL_ALERT_EVENT_LOG는 쿼리에서 잠재적 문제를 식별한 후 쿼리 성능 튜닝의 모범 사례에 따라 데이터베이스 설계를 최적화하여 쿼리를 재작성하는 데 사용할 수 있습니다. STL_ALERT_EVENT_LOG는 다음과 같은 알림을 기록합니다.

- 통계 누락

  통계가 누락된 경우에는 데이터 로드 또는 중요한 업데이트에 이어 ANALYZE를 실행한 다음 COPY 작업에 STATUPDATE를 사용하십시오. 자세한 내용은 Amazon Redshift 쿼리 설계 모범 사례 단원을 참조하십시오.

- 중첩 루프

  중첩 루프는 일반적으로 데카르트 곱입니다. 이때는 쿼리를 평가하여 참여하는 테이블이 모두 효율적으로 조인되었는지 확인하십시오.

- 선택의 폭이 제한적인 필터

  반환되는 행과 스캔되는 행의 비율은 0.05미만입니다. 스캔되는 행은 `rows_pre_user_filter` 값이고, 반환되는 행은 STL_SCAN 시스템 뷰의 rows 값입니다. 이 알림은 쿼리가 결과 집합을 확인하기 위해 비정상적으로 많은 수의 행을 스캔하고 있다는 것을 의미합니다. 이러한 문제는 정렬 키가 누락되었거나 잘못되었을 때 발생할 수 있습니다. 자세한 내용은 정렬 키 단원을 참조하십시오.

- 지나치게 많은 고스트 행

스캔 작업이 정리가 아닌 삭제된 것으로 표시되었거나, 혹은 커밋되지 않고 삽입된 비교적 다수의 행을 건너뛰었습니다. 자세한 내용은 테이블 Vacuum 단원을 참조하십시오.

- 다수의 행 분산

  100만 개가 넘는 행이 해시 조인 또는 집계를 위해 재분산되었습니다. 자세한 내용은 쿼리 최적화를 위한 데이터 배포 단원을 참조하십시오.

- 다수의 행 브로드캐스팅

  100만 개가 넘는 행이 해시 조인을 위해 브로드캐스팅되었습니다. 자세한 내용은 쿼리 최적화를 위한 데이터 배포 단원을 참조하십시오.

- 직렬 실행

  쿼리 계획에서 DS_DIST_ALL_INNER 재분산 스타일이 지정되면서 내부 테이블 전체가 단일 조각으로 재분산되었기 때문에 직렬 실행을 피할 수 없습니다. 자세한 내용은 쿼리 최적화를 위한 데이터 배포 단원을 참조하십시오.

샘플 쿼리

다음은 4개 쿼리의 알림 이벤트를 나타낸 쿼리입니다.

```
SELECT query, substring(event,0,25) as event,
substring(solution,0,25) as solution,
trim(event_time) as event_time from stl_alert_event_log order by query;

 query |             event             |          solution            |     event_time

-------+-------------------------------+-------------------------------
+---------------------
  6567 | Missing query planner statist | Run the ANALYZE command      | 2014-01-03
 18:20:58
  7450 | Scanned a large number of del | Run the VACUUM command to rec| 2014-01-03
 21:19:31
  8406 | Nested Loop Join in the query | Review the join predicates to| 2014-01-04
 00:34:22
 29512 | Very selective query filter:r | Review the choice of sort key| 2014-01-06
 22:00:00

(4 rows)
```

STL_ANALYZE

ANALYZE 작업에 대한 세부 정보를 기록합니다.

STL_ANALYZE는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_ANALYZE_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID입니다.
xid	long	트랜잭션 ID.
데이터베이스	char(30)	데이터베이스 이름입니다.
table_id	정수	테이블 ID입니다.
status	char(15)	ANALYZE 명령의 결과입니다. 가능한 값은 `Full`, `Skipped` 및 `PredicateColumn` 입니다.
rows	double	테이블에 포함된 행의 총 수입니다.
modified_rows	double	마지막 ANALYZE 작업 이후 수정된 행의 총 수입니다.
threshold_percent	정수	`analyze_threshold_percent` 파라미터의 값입니다.
is_auto	char(1)	작업에 Amazon Redshift 분석 작업이 기본적으로 포함되어 있는 경우 이 값은 tru(`t`)입니다. ANALYZE 명령이 명시적으로 실행된 경우 이 값은 false(`f`)입니다.
starttime	타임스탬프	ANALYZE 작업이 실행되기 시작한 시간(UTC)입니다.

열 명칭	데이터 유형	설명
endtime	타임스탬프	ANALYZE 작업이 실행을 마친 시간(UTC)입니다.
prevtime	타임스탬프	이전에 테이블을 분석한 시간(UTC)입니다.
num_predicate_cols	정수	테이블의 현재 조건자 열 수입니다.
num_new_predicate_cols	정수	이전 ANALYZE 작업 이후 테이블의 새로운 조건자 열 수입니다.
is_background	character(1)	자동 ANALYZE 작업에 의해 분석이 수행된 경우 이 값은 true(t)입니다. 그 외의 경우 이 값은 false(f)입니다.
auto_analyze_phase	character(100)	내부용으로 예약되어 있습니다.
schema_name	char(128)	테이블의 스키마 이름
table_name	char(136)	테이블의 이름

샘플 쿼리

다음은 STV_TBL_PERM을 조인하여 테이블 이름과 실행 세부 정보를 표시하는 예입니다.

```
select distinct a.xid, trim(t.name) as name, a.status, a.rows, a.modified_rows,
 a.starttime, a.endtime
from stl_analyze a
join stv_tbl_perm t  on t.id=a.table_id
where name = 'users'
order by starttime;

xid    | name  | status          | rows  | modified_rows | starttime           |
 endtime
-------+-------+-----------------+-------+---------------+---------------------
+---------------------
  1582 | users | Full            | 49990 |         49990 | 2016-09-22 22:02:23 |
 2016-09-22 22:02:28
```

```
244287 | users | Full            | 24992 |         74988 | 2016-10-04 22:50:58 |
 2016-10-04 22:51:01
244712 | users | Full            | 49984 |         24992 | 2016-10-04 22:56:07 |
 2016-10-04 22:56:07
245071 | users | Skipped         | 49984 |             0 | 2016-10-04 22:58:17 |
 2016-10-04 22:58:17
245439 | users | Skipped         | 49984 |          1982 | 2016-10-04 23:00:13 |
 2016-10-04 23:00:13
(5 rows)
```

STL_ANALYZE_COMPRESSION

COPY 또는 ANALYZE COMPRESSION 명령을 수행하는 동안 압축 분석 작업에 대한 세부 정보를 기록합니다.

STL_ANALYZE_COMPRESSION은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰
인 SYS_ANALYZE_COMPRESSION_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID입니다.
start_time	타임스탬프	압축 분석 작업을 시작한 시간입니다.
xid	bigint	압축 분석 작업의 트랜잭션 ID입니다.
tbl	정수	분석한 테이블의 테이블 ID입니다.
tablename	character(128)	분석한 테이블의 이름입니다.
col	정수	압축 인코딩을 확인하기 위해 분석한 테이블의 열 인덱스입니다.

열 명칭	데이터 유형	설명
old_encoding	character(15)	압축 분석 이전의 인코딩 유형입니다.
new_encoding	character(15)	압축 분석 이후의 인코딩 유형입니다.
mode	character(14)	가능한 값은 다음과 같습니다. PRESET new_encoding 이 열 데이터 형식을 기준으로 Amazon Redshift COPY 명령에 의해 결정되도록 지정합니다. 데이터가 샘플링되지 않습니다. ON new_encoding 이 샘플 데이터의 분석을 기준으로 Amazon Redshift COPY 명령에 의해 결정되도록 지정합니다. ANALYZE ONLY new_encoding 이 샘플 데이터의 분석을 기준으로 Amazon Redshift ANALYZE COMPRESSION 명령에 의해 결정되도록 지정합니다. 그러나 분석된 열의 인코딩 유형은 변경되지 않습니다.
best_compression_encoding	character(15)	최적의 압축률을 제공하는 인코딩 유형입니다.
recommended_bytes	character(15)	새 인코딩을 채택하여 사용하는 바이트입니다.
best_compression_bytes	character(15)	최적의 압축 인코딩을 채택하여 사용하는 바이트입니다.
ndv	bigint	샘플 행에 있는 고유한 값 수입니다.

샘플 쿼리

다음 예에서는 동일한 세션에서 실행된 마지막 COPY 명령으로 lineitem 테이블에 대한 압축 분석의 세부 정보를 검사합니다.

```
select xid, tbl, btrim(tablename) as tablename, col, old_encoding, new_encoding,
 best_compression_encoding, mode
from stl_analyze_compression
where xid = (select xid from stl_query where query = pg_last_copy_id()) order by col;

 xid  |  tbl   | tablename | col |  old_encoding   |  new_encoding   |
 best_compression_encoding |      mode
------+--------+-----------+-----+-----------------+-----------------
+---------------------------+----------------
 5308 | 158961 | $lineitem |   0 | mostly32        | az64            | delta
          | ON
 5308 | 158961 | $lineitem |   1 | mostly32        | az64            | az64
          | ON
 5308 | 158961 | $lineitem |   2 | lzo             | az64            | az64
          | ON
 5308 | 158961 | $lineitem |   3 | delta           | az64            | az64
          | ON
 5308 | 158961 | $lineitem |   4 | bytedict        | az64            | bytedict
          | ON
 5308 | 158961 | $lineitem |   5 | mostly32        | az64            | az64
          | ON
 5308 | 158961 | $lineitem |   6 | delta           | az64            | az64
          | ON
 5308 | 158961 | $lineitem |   7 | delta           | az64            | az64
          | ON
 5308 | 158961 | $lineitem |   8 | lzo             | lzo             | lzo
          | ON
 5308 | 158961 | $lineitem |   9 | runlength       | runlength       | runlength
          | ON
 5308 | 158961 | $lineitem |  10 | delta           | az64            | az64
          | ON
 5308 | 158961 | $lineitem |  11 | delta           | az64            | az64
          | ON
 5308 | 158961 | $lineitem |  12 | delta           | az64            | az64
          | ON
 5308 | 158961 | $lineitem |  13 | bytedict        | bytedict        | bytedict
          | ON
 5308 | 158961 | $lineitem |  14 | bytedict        | bytedict        | bytedict
          | ON
```

```
 5308 | 158961 | $lineitem |  15 | text255         | text255         | text255
          | ON
(16 rows)
```

STL_BCAST

데이터를 브로드캐스팅하는 쿼리 단계를 실행하면서 네트워크 활동에 대한 정보를 기록합니다. 네트워크 트래픽은 임의 조각의 특정 단계에서 네트워크를 통해 전송되는 행, 바이트 및 패킷 수를 기준으로 수집됩니다. 단계의 지속 시간은 로그 기록이 시작되는 시간부터 종료되는 시간까지입니다.

쿼리에서 브로드캐스팅 단계를 식별하려면 SVL_QUERY_SUMMARY 뷰에서 bcast 레이블을 찾거나, 혹은 EXPLAIN 명령을 실행하여 bcast가 포함된 단계 속성을 찾으면 됩니다.

STL_BCAST는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_BCAST에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호
step	정수	실행된 쿼리 단계입니다.

열 명칭	데이터 유형	설명
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .
tasknum	정수	단계 실행을 위해 할당된 쿼리 태스크 프로세스 수.
rows	bigint	처리된 총 행 수.
bytes	bigint	단계에서 출력되는 모든 행의 크기(바이트)
packets	정수	네트워크를 통해 전송되는 패킷 총 수

샘플 쿼리

다음은 패킷이 1개 이상이고, 시작 시간부터 종료 시간까지 차이가 1초 이상인 쿼리의 브로드캐스팅 정보를 반환하는 예입니다.

```
select query, slice, step, rows, bytes, packets, datediff(seconds, starttime, endtime)
from stl_bcast
where packets>0 and datediff(seconds, starttime, endtime)>0;
```

```
 query | slice | step | rows | bytes | packets | date_diff
-------+-------+------+------+-------+---------+-----------
   453 |     2 |    5 |    1 |   264 |       1 |         1
   798 |     2 |    5 |    1 |   264 |       1 |         1
  1408 |     2 |    5 |    1 |   264 |       1 |         1
  2993 |     0 |    5 |    1 |   264 |       1 |         1
  5045 |     3 |    5 |    1 |   264 |       1 |         1
  8073 |     3 |    5 |    1 |   264 |       1 |         1
  8163 |     3 |    5 |    1 |   264 |       1 |         1
  9212 |     1 |    5 |    1 |   264 |       1 |         1
  9873 |     1 |    5 |    1 |   264 |       1 |         1
(9 rows)
```

STL_COMMIT_STATS

다양한 커밋 단계의 타이밍과 커밋된 블록 수를 포함하여 커밋 성능에 대한 지표를 제공합니다. STL_COMMIT_STATS에 대한 쿼리를 실행하여 커밋 시 사용된 트랜잭션 구간과 발생한 대기열의 크기를 확인합니다.

STL_COMMIT_STATS는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_TRANSACTION_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
xid	bigint	커밋 중인 트랜잭션 ID
node	integer	노드 번호. -1은 리더 노드입니다.
startqueue	timestamp	커밋을 위한 대기열의 시작
startwork	timestamp	커밋 시작
endflush	timestamp	더티 블록 플러시 단계의 끝
endstage	timestamp	메타데이터 스테이징 단계의 끝
endlocal	timestamp	로컬 커밋 단계의 끝
startglobal	timestamp	전역 단계의 시작
endtime	timestamp	커밋의 끝
queuelen	bigint	커밋 대기열에서 이 트랜잭션 앞에 있었던 트랙잭션의 수
permblocks	bigint	이번 커밋 시점을 기준으로 기존 영구 블록의 수
newblocks	bigint	이번 커밋 시점을 기준으로 새로운 영구 블록의 수
dirtyblocks	bigint	이번 커밋에 포함하여 작성해야 했던 블록의 수

열 명칭	데이터 유형	설명
headers	bigint	이번 커밋에 포함하여 작성해야 했던 블록 헤더의 수
numxids	integer	활성 DML 트랜잭션의 수.
oldestxid	bigint	가장 오래된 활성 DML 트랜잭션의 XID.
extwritelatency	bigint	이 정보는 내부 전용입니다.
metadatawritten	int	이 정보는 내부 전용입니다.
tombstonedblocks	bigint	이 정보는 내부 전용입니다.
tossedblocks	bigint	이 정보는 내부 전용입니다.
batched_by	bigint	이 정보는 내부 전용입니다.

샘플 쿼리

```
select node, datediff(ms,startqueue,startwork) as queue_time,
datediff(ms, startwork, endtime) as commit_time, queuelen
from stl_commit_stats
where xid = 2574
order by node;

node | queue_time   | commit_time | queuelen
-----+--------------+-------------+---------
  -1 |            0 |         617 |        0
   0 | 444950725641 |         616 |        0
   1 | 444950725636 |         616 |        0
```

STL_CONNECTION_LOG

인증 시도 횟수와 연결 및 차단 정보를 기록합니다.

STL_CONNECTION_LOG는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_CONNECTION_LOG에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
이벤트	character(50)	연결 또는 인증 이벤트
recordtime	타임스탬프	이벤트 발생 시간
remotehost	character(45)	원격 호스트의 이름 또는 IP 주소
remoteport	character(32)	원격 호스트의 포트 번호
pid	정수	쿼리 문과 연결된 프로세스 ID
dbname	character(50)	데이터베이스 이름.
사용자 이름	character(50)	사용자 이름.
authmethod	character(32)	인증 방법
기간	정수	연결 지속 시간(마이크로초)
sslversion	character(50)	SSL(Secure Sockets Layer) 버전
sslcipher	character(128)	SSL 암호
mtu	정수	최대 전송 단위(MTU)
sslcompression	character(64)	SSL 압축 유형
sslexpansion	character(64)	SSL 확장 유형

열 명칭	데이터 유형	설명
iamauthguid	character(36)	CloudTrail 요청에 대한 IAM 인증 ID입니다.
application_name	character(250)	세션에서 애플리케이션의 초기 이름 또는 업데이트된 이름입니다.
os_version	character(64)	Amazon Redshift 클러스터에 연결하는 클라이언트 시스템에 있는 운영 체제의 버전입니다.
driver_version	character(64)	서드 파티 SQL 클라이언트 도구에서 Amazon Redshift 클러스터에 연결하는 ODBC 또는 JDBC 드라이버 버전입니다.
plugin_name	character(32)	Amazon Redshift 클러스터에 연결하는 데 사용되는 플러그인의 이름입니다.
protocol_version	정수	Amazon Redshift 드라이버가 서버와의 연결을 설정할 때 사용하는 내부 프로토콜 버전입니다. 프로토콜 버전은 드라이버와 서버 간에 협상됩니다. 버전은 사용 가능한 기능을 설명합니다. 유효한 값으로는 다음이 포함됩니다. • 0(BASE_SERVER_PROTOCOL_VERSION) • 1(EXTENDED_RESULT_METADATA_SERVER_PROTOCOL_VERSION) – 쿼리당 왕복을 저장하기 위해 서버는 추가 결과 집합 메타데이터 정보를 보냅니다. • 2(BINARY_PROTOCOL_VERSION) – 결과 집합의 데이터 유형에 따라 서버는 데이터를 이진 형식으로 보냅니다. • 3(EXTENDED2_RESULT_METADATA_SERVER_PROTOCOL_VERSION) – 서버가 열의 대/소문자 구분(데이터 정렬) 정보를 보냅니다.
sessionid	character(36)	현재 세션에 대한 전역적으로 고유한 식별자입니다. 세션 ID는 노드 오류가 다시 시작해도 유지됩니다.
압축	character(16)	연결에 사용 중인 압축 알고리즘입니다.

샘플 쿼리

열려있는 연결 세부 정보를 보려면 다음과 같이 쿼리를 실행합니다.

```
select recordtime, username, dbname, remotehost, remoteport
from stl_connection_log
where event = 'initiating session'
and pid not in
(select pid from stl_connection_log
where event = 'disconnecting session')
order by 1 desc;

recordtime          | username    | dbname     | remotehost    | remoteport

--------------------+-------------+------------+---------------+------------
2014-11-06 20:30:06 | rdsdb       | dev        | [local]       |

2014-11-06 20:29:37 | test001     | test       | 10.49.42.138  | 11111

2014-11-05 20:30:29 | rdsdb       | dev        | 10.49.42.138  | 33333

2014-11-05 20:28:35 | rdsdb       | dev        | [local]       |
(4 rows)
```

다음은 실패한 인증 시도와 성공한 연결 및 차단을 나타낸 예입니다.

```
select event, recordtime, remotehost, username
from stl_connection_log order by recordtime;

            event      |         recordtime        |  remotehost   | username

-----------------------+---------------------------+---------------+---------
authentication failure | 2012-10-25 14:41:56.96391 | 10.49.42.138 | john

authenticated          | 2012-10-25 14:42:10.87613 | 10.49.42.138 | john

initiating session     | 2012-10-25 14:42:10.87638 | 10.49.42.138 | john

disconnecting session  | 2012-10-25 14:42:19.95992 | 10.49.42.138 | john

(4 rows)
```

다음 예에서는 ODBC 드라이버의 버전, 클라이언트 시스템의 운영 체제 및 Amazon Redshift 클러스터에 연결하는 데 사용되는 플러그인을 보여줍니다. 이 예에서 사용되는 플러그인은 로그인 이름과 암호를 사용하는 표준 ODBC 드라이버 인증을 위한 것입니다.

```
select driver_version, os_version, plugin_name from stl_connection_log;

driver_version                          |  os_version                        |
 plugin_name
----------------------------------------+------------------------------------
+--------------------
Amazon Redshift ODBC Driver 1.4.15.0001 | Darwin 18.7.0 x86_64               | none
Amazon Redshift ODBC Driver 1.4.15.0001 | Linux 4.15.0-101-generic x86_64    | none
```

다음 예에서는 클라이언트 시스템의 운영 체제 버전, 드라이버 버전 및 프로토콜 버전을 보여줍니다.

```
select os_version, driver_version, protocol_version from stl_connection_log;

os_version                      |  driver_version              | protocol_version
--------------------------------+------------------------------+--------------------
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
```

STL_DDLTEXT

다음과 같이 시스템에서 실행된 DDL 문을 수집합니다.

이러한 DDL 문에는 다음 쿼리와 객체가 포함됩니다.

- CREATE SCHEMA, TABLE, VIEW
- DROP SCHEMA, TABLE, VIEW
- ALTER SCHEMA, TABLE

STL_QUERYTEXTSTL_UTILITYTEXT 및 도 참조하십시오.SVL_STATEMENTTEXT 이 보기들은 시스템에서 SQL 명령을 실행한 시간표를 제공하기 때문에 문제 해결 뿐만 아니라 모든 시스템 활동에 대한 감사 추적을 생성하는 데도 유용합니다.

STARTTIME 열과 ENDTIME 열은 일정 시간 동안 기록된 문을 확인하는 데 사용됩니다. 긴 SQL 텍스트 블록은 200개 문자의 길이로 구분되며, SEQUENCE 열에서 단일 문에 속하는 텍스트 조각을 식별할 수 있습니다.

STL_DDLTEXT는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
xid	bigint	문에 연결된 트랜잭션 ID.
pid	정수	쿼리 문과 연결된 프로세스 ID
레이블	character(320)	쿼리 실행에 사용되는 파일의 이름 또는 SET QUERY_GROUP 명령을 사용하여 정의되는 레이블. 쿼리가 파일 기반이 아니거나 QUERY_GROUP 파라미터가 설정되지 않은 경우, 이 필드의 값은 공백입니다.
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
SEQUENCE	정수	단일 문에 200자 이상이 포함된 경우, 해당 문에 대해 추가 행이 기록됩니다. 시퀀스 0이 첫 번째 행이고 1이 두 번째 행이 되는 방식입니다.
텍스트	character(200)	200자씩 증가하는 SQL 텍스트. 이 필드에는 백슬래시(\\) 및 줄 바꿈(\n) 등의 특수 문자가 포함될 수 있습니다.

샘플 쿼리

다음 쿼리는 이전에 실행된 DDL 문을 포함하는 레코드를 반환합니다.

```
select xid, starttime, sequence, substring(text,1,40) as text
from stl_ddltext order by xid desc, sequence;
```

다음은 CREATE TABLE 문 4개를 보여주는 출력 샘플입니다. DDL 문은 가독성을 위해 잘린 text 열에 나타납니다.

```
 xid  |         starttime          | sequence |                  text
------+----------------------------+----------
+----------------------------------------
 1806 | 2013-10-23 00:11:14.709851 |        0 | CREATE TABLE supplier ( s_suppkey int4
 N
 1806 | 2013-10-23 00:11:14.709851 |        1 |  s_comment varchar(101) NOT NULL )
 1805 | 2013-10-23 00:11:14.496153 |        0 | CREATE TABLE region ( r_regionkey int4
 N
 1804 | 2013-10-23 00:11:14.285986 |        0 | CREATE TABLE partsupp ( ps_partkey int8
 1803 | 2013-10-23 00:11:14.056901 |        0 | CREATE TABLE part ( p_partkey int8 NOT
 N
 1803 | 2013-10-23 00:11:14.056901 |        1 | ner char(10) NOT NULL , p_retailprice
 nu
(6 rows)
```

저장된 SQL 재구성

다음 SQL은 STL_DDLTEXT의 text 열에 저장된 행을 나열합니다. 행은 xid 및 sequence로 정렬됩니다. 원래 SQL이 200자 이상의 여러 행인 경우 STL_DDLTEXT는 sequence별로 여러 행을 포함할 수 있습니다.

```
SELECT xid, sequence, LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text)
 END, '') WITHIN GROUP (ORDER BY sequence) as query_statement
FROM stl_ddltext GROUP BY xid, sequence ORDER BY xid, sequence;
```

```
xid     |  sequence | query_statement
--------+-----------+-----------------
7886671    0          create external schema schema_spectrum_uddh\nfrom data catalog
\ndatabase 'spectrum_db_uddh'\niam_role ''\ncreate external database if not exists;
```

```
7886752    0          CREATE EXTERNAL TABLE schema_spectrum_uddh.soccer_league\n(\n
  league_rank smallint,\n  prev_rank   smallint,\n  club_name   varchar(15),\n
 league_name varchar(20),\n  league_off  decimal(6,2),\n  le
7886752    1          ague_def  decimal(6,2),\n  league_spi  decimal(6,2),\n
 league_nspi smallint\n)\nROW FORMAT DELIMITED \n    FIELDS TERMINATED BY ',' \n
 LINES TERMINATED BY '\\n\\l'\nstored as textfile\nLOCATION 's
7886752    2          3://mybucket-spectrum-uddh/'\ntable properties
 ('skip.header.line.count'='1');
...
```

STL_DDLTEXT의 `text` 열에 저장된 SQL을 재구성하려면 다음 SQL 문을 실행합니다. 그렇게 하면 `text` 열에 있는 하나 이상의 세그먼트에서 DDL 문이 함께 포함됩니다. 재구성된 SQL을 실행하기 전에 SQL 클라이언트에서 모든 (\n) 특수 문자를 줄 바꿈으로 바꿉니다. 다음 SELECT 문의 결과는 `query_statement` 필드에서 SQL을 재구성하기 위해 세 개의 행을 순서대로 조합합니다.

```
SELECT LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END) WITHIN
 GROUP (ORDER BY sequence) as query_statement
FROM stl_ddltext GROUP BY xid, endtime order by xid, endtime;
```

```
query_statement
--------------
create external schema schema_spectrum_uddh\nfrom data catalog\ndatabase
 'spectrum_db_uddh'\niam_role ''\ncreate external database if not exists;
CREATE EXTERNAL TABLE schema_spectrum_uddh.soccer_league\n(\n  league_rank smallint,
\n  prev_rank   smallint,\n  club_name   varchar(15),\n  league_name varchar(20),\n
 league_off  decimal(6,2),\n  league_def  decimal(6,2),\n  league_spi  decimal(6,2),
\n  league_nspi smallint\n)\nROW FORMAT DELIMITED \n    FIELDS TERMINATED BY ',' \n
  LINES TERMINATED BY '\\n\\l'\nstored as textfile\nLOCATION 's3://mybucket-spectrum-
uddh/'\ntable properties ('skip.header.line.count'='1');
```

STL_DELETE

쿼리의 삭제 실행 단계를 분석합니다.

STL_DELETE는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_DELETE에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호
step	정수	실행된 쿼리 단계입니다.
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
tasknum	정수	단계 실행을 위해 할당된 쿼리 태스크 프로세스 수.
rows	bigint	처리된 총 행 수.
tbl	정수	테이블 ID.

샘플 쿼리

다음은 STL_DELETE에 행을 생성할 목적으로 EVENT 테이블에 행을 삽입한 후 삭제하는 예입니다.

먼저 EVENT 테이블에 행을 삽입한 후 실제로 삽입되었는지 확인합니다.

```
insert into event(eventid,venueid,catid,dateid,eventname)
values ((select max(eventid)+1 from event),95,9,1857,'Lollapalooza');
```

```
select * from event
where eventname='Lollapalooza'
order by eventid;
```

```
 eventid | venueid | catid | dateid |  eventname   |      starttime
---------+---------+-------+--------+--------------+---------------------
    4274 |     102 |     9 |   1965 | Lollapalooza | 2008-05-01 19:00:00
    4684 |     114 |     9 |   2105 | Lollapalooza | 2008-10-06 14:00:00
    5673 |     128 |     9 |   1973 | Lollapalooza | 2008-05-01 15:00:00
    5740 |      51 |     9 |   1933 | Lollapalooza | 2008-04-17 15:00:00
    5856 |     119 |     9 |   1831 | Lollapalooza | 2008-01-05 14:00:00
    6040 |     126 |     9 |   2145 | Lollapalooza | 2008-11-15 15:00:00
    7972 |      92 |     9 |   2026 | Lollapalooza | 2008-07-19 19:30:00
    8046 |      65 |     9 |   1840 | Lollapalooza | 2008-01-14 15:00:00
    8518 |      48 |     9 |   1904 | Lollapalooza | 2008-03-19 15:00:00
    8799 |      95 |     9 |   1857 | Lollapalooza |
(10 rows)
```

이제 EVENT 테이블에 추가한 행을 삭제하고, 실제로 삭제되었는지 확인합니다.

```
delete from event
where eventname='Lollapalooza' and eventid=(select max(eventid) from event);
```

```
select * from event
where eventname='Lollapalooza'
order by eventid;
```

```
 eventid | venueid | catid | dateid |  eventname   |      starttime
---------+---------+-------+--------+--------------+---------------------
```

```
    4274 |     102 |     9 |   1965 | Lollapalooza | 2008-05-01 19:00:00
    4684 |     114 |     9 |   2105 | Lollapalooza | 2008-10-06 14:00:00
    5673 |     128 |     9 |   1973 | Lollapalooza | 2008-05-01 15:00:00
    5740 |      51 |     9 |   1933 | Lollapalooza | 2008-04-17 15:00:00
    5856 |     119 |     9 |   1831 | Lollapalooza | 2008-01-05 14:00:00
    6040 |     126 |     9 |   2145 | Lollapalooza | 2008-11-15 15:00:00
    7972 |      92 |     9 |   2026 | Lollapalooza | 2008-07-19 19:30:00
    8046 |      65 |     9 |   1840 | Lollapalooza | 2008-01-14 15:00:00
    8518 |      48 |     9 |   1904 | Lollapalooza | 2008-03-19 15:00:00
(9 rows)
```

그런 다음 stl_delete에 대해 쿼리를 실행하여 삭제 실행 단계를 확인합니다. 이번 예에서는 쿼리가 300개가 넘는 행을 반환하였으며, 아래 출력은 표시를 위해 줄인 것입니다.

```
select query, slice, segment, step, tasknum, rows, tbl from stl_delete order by query;
```

```
 query | slice | segment | step | tasknum | rows |  tbl
-------+-------+---------+------+---------+------+--------
     7 |     0 |       0 |    1 |       0 |    0 | 100000
     7 |     1 |       0 |    1 |       0 |    0 | 100000
     8 |     0 |       0 |    1 |       2 |    0 | 100001
     8 |     1 |       0 |    1 |       2 |    0 | 100001
     9 |     0 |       0 |    1 |       4 |    0 | 100002
     9 |     1 |       0 |    1 |       4 |    0 | 100002
    10 |     0 |       0 |    1 |       6 |    0 | 100003
    10 |     1 |       0 |    1 |       6 |    0 | 100003
    11 |     0 |       0 |    1 |       8 |    0 | 100253
    11 |     1 |       0 |    1 |       8 |    0 | 100253
    12 |     0 |       0 |    1 |       0 |    0 | 100255
    12 |     1 |       0 |    1 |       0 |    0 | 100255
    13 |     0 |       0 |    1 |       2 |    0 | 100257
    13 |     1 |       0 |    1 |       2 |    0 | 100257
    14 |     0 |       0 |    1 |       4 |    0 | 100259
    14 |     1 |       0 |    1 |       4 |    0 | 100259
    ...
```

STL_DISK_FULL_DIAG

디스크가 가득 찬 경우에 기록되는 오류에 대한 로그 정보입니다.

STL_DISK_FULL_DIAG는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명		
currenttime	bigint	2000년 1월 1일 이후 오류가 생성된 날짜와 시간입니다(밀리초).		
node_num	bigint	노드에 대한 식별자입니다.		
query_id	bigint	오류를 발생한 쿼리의 식별자입니다.		
temp_blocks	bigint	쿼리에서 생성되는 임시 블록의 수입니다.		

샘플 쿼리

다음 예에서는 디스크 가득 참 오류가 있을 때 저장되는 데이터에 대한 세부 정보를 반환합니다.

```
select * from stl_disk_full_diag
```

다음은 currenttime을 타임스탬프로 변환하는 예입니다.

```
select '2000-01-01'::timestamp + (currenttime/1000000.0)* interval '1 second' as
 currenttime,node_num,query_id,temp_blocks from pg_catalog.stl_disk_full_diag;
```

```
        currenttime         | node_num | query_id | temp_blocks
----------------------------+----------+----------+-------------
 2019-05-18 19:19:18.609338 |        0 |   569399 |       70982
 2019-05-18 19:37:44.755548 |        0 |   569580 |       70982
 2019-05-20 13:37:20.566916 |        0 |   597424 |       70869
```

STL_DIST

데이터를 분산시키는 쿼리 단계를 실행하면서 네트워크 활동에 대한 정보를 기록합니다. 네트워크 트래픽은 임의 조각의 특정 단계에서 네트워크를 통해 전송되는 행, 바이트 및 패킷 수를 기준으로 수집됩니다. 단계의 지속 시간은 로그 기록이 시작되는 시간부터 종료되는 시간까지입니다.

쿼리에서 분산 단계를 식별하려면 QUERY_SUMMARY 뷰에서 dist 레이블을 찾거나, 혹은 EXPLAIN 명령을 실행하여 dist가 포함된 단계 속성을 찾으면 됩니다.

STL_DIST는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_DIST에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호
step	정수	실행된 쿼리 단계입니다.
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .

열 명칭	데이터 유형	설명
tasknum	정수	단계 실행을 위해 할당된 쿼리 태스크 프로세스 수.
rows	bigint	처리된 총 행 수.
bytes	bigint	단계에서 출력되는 모든 행의 크기(바이트)
packets	정수	네트워크를 통해 전송되는 패킷 총 수

샘플 쿼리

다음은 패킷이 1개 이상이고, 지속 시간이 0보다 큰 쿼리의 분산 정보를 반환하는 예입니다.

```
select query, slice, step, rows, bytes, packets,
datediff(seconds, starttime, endtime) as duration
from stl_dist
where packets>0 and datediff(seconds, starttime, endtime)>0
order by query
limit 10;
```

```
 query  | slice | step |  rows  |  bytes  | packets | duration
--------+-------+------+--------+---------+---------+----------
    567 |     1 |    4 |  49990 | 6249564 |     707 |         1
    630 |     0 |    5 |   8798 |  408404 |      46 |         2
    645 |     1 |    4 |   8798 |  408404 |      46 |         1
    651 |     1 |    5 | 192497 | 9226320 |    1039 |         6
    669 |     1 |    4 | 192497 | 9226320 |    1039 |         4
    675 |     1 |    5 |   3766 |  194656 |      22 |         1
    696 |     0 |    4 |   3766 |  194656 |      22 |         1
    705 |     0 |    4 |    930 |   44400 |       5 |         1
 111525 |     0 |    3 |     68 |   17408 |       2 |         1
(9 rows)
```

STL_ERROR

Amazon Redshift 데이터베이스 엔진에서 발생한 내부 처리 오류를 기록합니다. STL_ERROR는 SQL 오류 또는 메시지를 기록하지는 않습니다. STL_ERROR에 기록되는 정보는 몇 가지 오류 문제를 해결하는 데 유용합니다. AWS 지원 엔지니어가 문제 해결 도중에 이 정보를 물을 수도 있습니다.

STL_ERROR는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

Copy 명령을 사용하여 데이터를 불러올 때 발생할 수 있는 오류 코드 목록은 로드 오류 참조 섹션을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
포함	character(12)	예외 발생 프로세스
recordtime	타임스탬프	오류 발생 시간
pid	정수	프로세스 ID. STL_QUERY 테이블에는 프로세스 ID와 완료된 쿼리의 고유 쿼리 ID가 저장됩니다.
errcode	정수	오류 카테고리에 해당하는 오류 코드
파일	character(90)	오류가 발생한 소스 파일 이름
linenum	정수	소스 파일에서 오류가 발생한 라인 번호
context	character(100)	오류 원인
오류	character(512)	오류 메시지.

샘플 쿼리

다음은 STL_ERROR에서 오류 정보를 가져오는 예입니다.

```
select process, errcode, linenum as line,
trim(error) as err
```

```
from stl_error;

   process    | errcode | line |                                 err
--------------+---------+------
+-------------------------------------------------------------------
 padbmaster   |    8001 |  194 | Path prefix: s3://redshift-downloads/testnulls/
venue.txt*
 padbmaster   |    8001 |  529 | Listing bucket=redshift-downloads prefix=tests/
category-csv-quotes
 padbmaster   |       2 |  190 | database "template0" is not currently accepting
 connections
 padbmaster   |      32 | 1956 | pq_flush: could not send data to client: Broken pipe
(4 rows)
```

STL_EXPLAIN

실행을 목적으로 제출된 쿼리의 EXPLAIN 계획을 표시합니다.

STL_EXPLAIN은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_EXPLAIN에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_EXPLAIN을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.

열 명칭	데이터 유형	설명
nodeid	정수	계획 노드 식별자. 쿼리 실행 시 이 식별자를 통해 노드가 1개 이상의 단계로 매핑됩니다.
parentid	정수	상위 노드를 나타내는 계획 노드 식별자. 상위 노드 1개에는 하위 노드가 다수 종속됩니다. 예를 들어 병합 조인은 조인 테이블에서 스캔에 대한 상위 노드입니다.
plannode	character(400)	EXPLAIN 출력의 노드 텍스트. 컴퓨팅 노드에서 실행을 의미하는 계획 노드에는 EXPLAIN 출력 시 **XN** 접두사가 첨부됩니다.
정보	character(400)	계획 노드의 한정자 및 필터 정보. 예를 들어 조인 조건과 WHERE 절 제한이 이 열에 포함됩니다.

샘플 쿼리

집계 조인 쿼리의 EXPLAIN 출력이 다음과 같다고 가정하겠습니다.

```
explain select avg(datediff(day, listtime, saletime)) as avgwait
from sales, listing where sales.listid = listing.listid;
                                QUERY PLAN

------------------------------------------------------------------------------
 XN Aggregate  (cost=6350.30..6350.31 rows=1 width=16)
  ->  XN Hash Join DS_DIST_NONE  (cost=47.08..6340.89 rows=3766 width=16)
        Hash Cond: ("outer".listid = "inner".listid)
        -> XN Seq Scan on listing  (cost=0.00..1924.97 rows=192497 width=12)
        -> XN Hash  (cost=37.66..37.66 rows=3766 width=12)
              -> XN Seq Scan on sales  (cost=0.00..37.66 rows=3766 width=12)
(6 rows)
```

이 쿼리를 실행하고, 쿼리 ID가 10이라면 STL_EXPLAIN 테이블을 사용하여 EXPLAIN 명령이 반환하는 것과 동일한 유형의 정보를 볼 수 있습니다.

```
select query,nodeid,parentid,substring(plannode from 1 for 30),
substring(info from 1 for 20) from stl_explain
where query=10 order by 1,2;

query| nodeid |parentid|           substring            |    substring
```

```
-----+--------+--------+--------------------------------+-------------------
10   |      1 |      0 |XN Aggregate  (cost=6717.61..6  |
10   |      2 |      1 |  -> XN Merge Join DS_DIST_NO   | Merge Cond:("outer"
10   |      3 |      2 |       -> XN Seq Scan on lis    |
10   |      4 |      2 |       -> XN Seq Scan on sal    |
(4 rows)
```

다음과 같은 쿼리를 가정합니다.

```
select event.eventid, sum(pricepaid)
from event, sales
where event.eventid=sales.eventid
group by event.eventid order by 2 desc;

eventid |   sum
--------+----------
    289 | 51846.00
   7895 | 51049.00
   1602 | 50301.00
    851 | 49956.00
   7315 | 49823.00
...
```

이 쿼리의 ID가 15라면 다음과 같은 시스템 보기 쿼리에서 이전에 완료한 계획 노드가 반환됩니다. 이때 노드 순서가 반전되면서 실제 실행 순서를 표시합니다.

```
select query,nodeid,parentid,substring(plannode from 1 for 56)
from stl_explain where query=15 order by 1, 2 desc;

query|nodeid|parentid|                              substring
-----+------+--------+-----------------------------------------------------------
15   |    8 |      7 |                                    -> XN Seq Scan on eve
15   |    7 |      5 |                              -> XN Hash(cost=87.98..87.9
15   |    6 |      5 |                              -> XN Seq Scan on sales(cos
15   |    5 |      4 |                         -> XN Hash Join DS_DIST_OUTER(cos
15   |    4 |      3 |                  -> XN HashAggregate(cost=862286577.07..
15   |    3 |      2 |           -> XN Sort(cost=1000862287175.47..10008622871
15   |    2 |      1 |  -> XN Network(cost=1000862287175.47..1000862287197.
15   |    1 |      0 |XN Merge(cost=1000862287175.47..1000862287197.46 rows=87
(8 rows)
```

다음은 창 함수가 포함된 모든 쿼리 계획의 쿼리 ID를 가져오는 쿼리입니다.

```
select query, trim(plannode) from stl_explain
where plannode like '%Window%';

query|                                        btrim
-----+-------------------------------------------------------------------------
26   | -> XN Window(cost=1000985348268.57..1000985351256.98 rows=170 width=33)
27   | -> XN Window(cost=1000985348268.57..1000985351256.98 rows=170 width=33)
(2 rows)
```

STL_FILE_SCAN

COPY 명령을 사용하여 데이터를 로드하면서 Amazon Redshift가 읽은 파일을 반환합니다.

이 뷰에 대해 쿼리를 실행하면 데이터 로드 오류를 해결하는 데 도움이 될 수 있습니다. 특히 STL_FILE_SCAN은 병렬 데이터 로드 시 문제를 찾아내는 데 효과적입니다. 병렬 데이터 로드는 단일 COPY 명령으로 다수의 파일을 로드하기 때문입니다.

STL_FILE_SCAN은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_FILE_SCAN에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_LOAD_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.

열 명칭	데이터 유형	설명
name	character(90)	로드된 파일의 전체 경로 및 이름
lines	bigint	파일에서 읽어온 라인 수
bytes	bigint	파일에서 읽어온 바이트 수
loadtime	bigint	파일을 로드하는 데 걸린 시간(마이크로초)
curtime	Timestamp	Amazon Redshift가 파일 처리를 시작한 시간의 타임스탬프.
is_partial	정수	true(1)인 경우 COPY 작업 중에 입력 파일이 범위로 분할됨을 나타내는 값입니다. 이 값이 false(0)이면 입력 파일이 분할되지 않습니다.
start_offset	bigint	COPY 작업 중에 입력 파일이 분할되는 경우 분할의 오프셋 값(바이트 단위)을 나타내는 값입니다. 파일이 분할되지 않은 경우 이 값은 0입니다.

샘플 쿼리

다음은 Amazon Redshift가 읽어오는 데 1,000,000마이크로초 이상이 걸린 모든 파일의 이름과 로드 시간을 가져오는 쿼리입니다.

```
select trim(name)as name, loadtime from stl_file_scan
where loadtime > 1000000;
```

위 쿼리는 다음과 같은 예 출력을 반환합니다.

```
           name             | loadtime
----------------------------+----------
 listings_pipe.txt          |  9458354
 allusers_pipe.txt          |  2963761
 allevents_pipe.txt         |  1409135
 tickit/listings_pipe.txt   |  7071087
 tickit/allevents_pipe.txt  |  1237364
 tickit/allusers_pipe.txt   |  2535138
 listings_pipe.txt          |  6706370
 allusers_pipe.txt          |  3579461
```

```
 allevents_pipe.txt         |  1313195
 tickit/allusers_pipe.txt   |  3236060
 tickit/listings_pipe.txt   |  4980108
(11 rows)
```

STL_HASH

쿼리의 해시 실행 단계를 분석합니다.

STL_HASH는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_HASH에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호
step	정수	실행된 쿼리 단계입니다.
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .

열 명칭	데이터 유형	설명
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .
tasknum	정수	단계 실행을 위해 할당된 쿼리 태스크 프로세스 수.
rows	bigint	처리된 총 행 수.
bytes	bigint	단계에서 출력되는 모든 행의 크기(바이트)
slots	정수	해시 버킷의 총 수
occupied	정수	레코드가 저장되는 슬롯의 총 수
maxlength	정수	가장 큰 슬롯의 크기
tbl	정수	테이블 ID.
is_diskbased	character(1)	true(t)인 경우 쿼리가 디스크 기반 작업으로 수행된 것을 의미합니다. false(f)인 경우 쿼리가 메모리에서 수행된 것을 의미합니다.
workmem	bigint	단계에 할당된 유효 메모리 바이트의 총 수
num_parts	정수	해시 단계 도중 해시 테이블이 분할된 파티션의 총 수.
est_rows	bigint	해시 처리될 것으로 예상되는 행의 수
num_blocks_permitted	정수	이 정보는 내부 전용입니다.
크기 조정	정수	이 정보는 내부 전용입니다.
checksum	bigint	이 정보는 내부 전용입니다.
runtime_filter_size	정수	실행 시간 필터의 크기(바이트).

열 명칭	데이터 유형	설명
max_runtime_filter_size	정수	실행 시간 필터의 최대 크기(바이트).

샘플 쿼리

다음은 쿼리 720의 해시에 사용된 파티션의 수에 대한 정보를 반환하는 예로서 디스크에서 실행된 단계가 하나도 없다는 것을 나타내고 있습니다.

```
select slice, rows, bytes, occupied, workmem, num_parts, est_rows,
 num_blocks_permitted, is_diskbased
from stl_hash
where query=720 and segment=5
order by slice;
```

```
 slice | rows | bytes  | occupied | workmem  | num_parts | est_rows |
 num_blocks_permitted | is_diskbased
-------+------+--------+----------+----------+-----------+----------
+----------------------+--------------
     0 |  145 | 585800 |        1 | 88866816 |        16 |        1 |
 52                   f
     1 |    0 |      0 |        0 |        0 |        16 |        1 |
 52                   f
(2 rows)
```

STL_HASHJOIN

쿼리의 해시 조인 실행 단계를 분석합니다.

STL_HASHJOIN은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_HASHJOIN에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에

대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호
step	정수	실행된 쿼리 단계입니다.
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .
tasknum	정수	단계 실행을 위해 할당된 쿼리 태스크 프로세스 수.
rows	bigint	처리된 총 행 수.
tbl	정수	테이블 ID.
num_parts	정수	해시 단계 도중 해시 테이블이 분할된 파티션의 총 수.
join_type	정수	단계별 조인 유형: • 0. 쿼리가 내부 조인을 사용했습니다. • 1. 쿼리가 왼쪽 외부 조인을 사용했습니다. • 2. 쿼리가 전체 외부 조인을 사용했습니다.

열 명칭	데이터 유형	설명
		• 3. 쿼리가 오른쪽 외부 조인을 사용했습니다. • 4. 쿼리가 UNION 연산자를 사용했습니다. • 5. 쿼리가 IN 조건을 사용했습니다. • 6. 이 정보는 내부 전용입니다. • 7. 이 정보는 내부 전용입니다. • 8. 이 정보는 내부 전용입니다. • 9. 이 정보는 내부 전용입니다. • 10. 이 정보는 내부 전용입니다. • 11. 이 정보는 내부 전용입니다. • 12. 이 정보는 내부 전용입니다.
hash_looped	character(1)	이 정보는 내부 전용입니다.
switched_parts	character(1)	build(또는 외부) 측과 probe(또는 내부) 측의 전환 여부를 나타냅니다.
used_prefetching	character(1)	이 정보는 내부 전용입니다.
hash_segment	정수	해당 해시 단계의 세그먼트입니다.
hash_step	정수	해당 해시 단계의 단계 수입니다.
checksum	bigint	이 정보는 내부 전용입니다.
배포	정수	이 정보는 내부 전용입니다.

샘플 쿼리

다음은 쿼리 720의 해시 조인에 사용되는 파티션 수를 반환하는 예입니다.

```
select query, slice, tbl, num_parts
from stl_hashjoin
```

```
where query=720 limit 10;
```

```
 query | slice | tbl | num_parts
-------+-------+-----+-----------
   720 |     0 | 243 |         1
   720 |     1 | 243 |         1
(2 rows)
```

STL_INSERT

쿼리의 삽입 실행 단계를 분석합니다.

STL_INSERT는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_INSERT에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호
step	정수	실행된 쿼리 단계입니다.

열 명칭	데이터 유형	설명
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .
tasknum	정수	단계 실행을 위해 할당된 쿼리 태스크 프로세스 수.
rows	bigint	처리된 총 행 수.
tbl	정수	테이블 ID.
inserted_mega_value	character(1)	이 정보는 내부 전용입니다. 이 정보는 지정된 삽입 단계에서 큰 값을 삽입했는지를 보여줍니다. 큰 값은 여러 블록에 저장됩니다. 블록 크기는 기본적으로 1MB이며, 큰 값은 기본 설정에서 1MB보다 큽니다.

샘플 쿼리

다음은 가장 최근 쿼리의 삽입 실행 단계를 반환하는 예입니다.

```
select slice, segment, step, tasknum, rows, tbl
from stl_insert
where query=pg_last_query_id();
```

```
 slice | segment | step | tasknum | rows  |  tbl
-------+---------+------+---------+-------+--------
     0 |       2 |    2 |      15 | 24958 | 100548
     1 |       2 |    2 |      15 | 25032 | 100548
(2 rows)
```

STL_LIMIT

LIMIT 절을 SELECT 쿼리에서 사용할 때 발생하는 실행 단계를 분석합니다.

STL_LIMIT는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_LIMIT에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호
step	정수	실행된 쿼리 단계입니다.
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
tasknum	정수	단계 실행을 위해 할당된 쿼리 태스크 프로세스 수.
rows	bigint	처리된 총 행 수.

열 명칭	데이터 유형	설명
checksum	bigint	이 정보는 내부 전용입니다.

샘플 쿼리

다음 예에서는 STL_LIMIT에 행을 생성할 목적으로 먼저 VENUE 테이블에 대해 LIMIT 절을 사용하여 다음 쿼리를 실행합니다.

```
select * from venue
order by 1
limit 10;
```

```
 venueid |         venuename          |    venuecity    | venuestate | venueseats
---------+----------------------------+-----------------+------------+------------
       1 | Toyota Park                | Bridgeview      | IL         |          0
       2 | Columbus Crew Stadium      | Columbus        | OH         |          0
       3 | RFK Stadium                | Washington      | DC         |          0
       4 | CommunityAmerica Ballpark  | Kansas City     | KS         |          0
       5 | Gillette Stadium           | Foxborough      | MA         |      68756
       6 | New York Giants Stadium    | East Rutherford | NJ         |      80242
       7 | BMO Field                  | Toronto         | ON         |          0
       8 | The Home Depot Center      | Carson          | CA         |          0
       9 | Dick's Sporting Goods Park | Commerce City   | CO         |          0
      10 | Pizza Hut Park             | Frisco          | TX         |          0
(10 rows)
```

그러고 나서 다음 쿼리를 실행하여 VENUE 테이블에 대해 마지막으로 실행했던 쿼리의 쿼리 ID를 확인합니다.

```
select max(query)
from stl_query;
```

```
  max
--------
 127128
(1 row)
```

옵션이지만 다음 쿼리를 실행하여 쿼리 ID가 앞에서 실행한 LIMIT 쿼리와 일치하는지 확인할 수도 있습니다.

```
select query, trim(querytxt)
from stl_query
where query=127128;
```

```
 query  |                  btrim
--------+-----------------------------------------
 127128 | select * from venue order by 1 limit 10;
(1 row)
```

마지막으로 다음 쿼리를 실행하여 LIMIT 쿼리에 대한 정보를 STL_LIMIT 테이블에서 반환합니다.

```
select slice, segment, step, starttime, endtime, tasknum
from stl_limit
where query=127128
order by starttime, endtime;
```

```
  slice | segment | step |         starttime          |          endtime           |
 tasknum
 -------+---------+------+----------------------------+----------------------------
+---------
      1 |       1 |    3 | 2013-09-06 22:56:43.608114 | 2013-09-06 22:56:43.609383 |
   15
      0 |       1 |    3 | 2013-09-06 22:56:43.608708 | 2013-09-06 22:56:43.609521 |
   15
  10000 |       2 |    2 | 2013-09-06 22:56:43.612506 | 2013-09-06 22:56:43.612668 |
    0
(3 rows)
```

STL_LOAD_COMMITS

데이터 로드를 추적하거나 문제를 해결하기 위한 정보를 반환합니다.

이 뷰는 데이터베이스 테이블에 로드되는 각 데이터 파일의 진행 상황을 기록합니다.

STL_LOAD_COMMITS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_LOAD_COMMITS에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_LOAD_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	이 항목에서 로드되는 조각
name	character(256)	시스템 정의 값
filename	character(256)	추적 중인 파일 이름
byte_offset	정수	이 정보는 내부 전용입니다.
lines_scanned	정수	로드 파일에서 스캔되는 라인 수. 이 라인 수는 실제로 로드되는 행의 수와 다를 수도 있습니다. 예를 들어 로드 작업은 COPY 명령의 MAXERROR 옵션에 따라 스캔 도중에도 다수의 불량 레코드를 허용할 수도 있기 때문입니다.
오류	정수	이 정보는 내부 전용입니다.
curtime	타임스탬프	이 항목이 마지막으로 업데이트된 시간
status	정수	이 정보는 내부 전용입니다.
file_format	character(16)	로드 파일의 형식입니다. 가능한 값은 다음과 같습니다.

열 명칭	데이터 유형	설명
		• Avro • JSON • ORC • PARQUET • 텍스트
is_partial	정수	true(1)인 경우 COPY 작업 중에 입력 파일이 범위로 분할됨을 나타내는 값입니다. 이 값이 false(0)이면 입력 파일이 분할되지 않습니다.
start_offset	bigint	COPY 작업 중에 입력 파일이 분할되는 경우 분할의 오프셋 값(바이트 단위)을 나타내는 값입니다. 각 파일 분할은 해당 start_offset 값과 함께 별도의 레코드로 기록됩니다. 파일이 분할되지 않은 경우 이 값은 0입니다.
copy_job_id	bigint	복사 작업 식별자입니다. 0은 작업 식별자가 없음을 나타냅니다.

샘플 쿼리

다음은 마지막 COPY 작업의 세부 정보를 반환하는 예입니다.

```
select query, trim(filename) as file, curtime as updated
from stl_load_commits
where query = pg_last_copy_id();

 query |               file                |          updated
-------+-----------------------------------+----------------------------
 28554 | s3://dw-tickit/category_pipe.txt | 2013-11-01 17:14:52.648486
(1 row)
```

다음은 TICKIT 데이터베이스에 새로운 테이블 로드 항목을 저장하는 쿼리입니다.

```
select query, trim(filename), curtime
from stl_load_commits
where filename like '%tickit%' order by query;
```

```
 query |           btrim           |          curtime
-------+---------------------------+---------------------------
 22475 | tickit/allusers_pipe.txt  | 2013-02-08 20:58:23.274186
 22478 | tickit/venue_pipe.txt     | 2013-02-08 20:58:25.070604
 22480 | tickit/category_pipe.txt  | 2013-02-08 20:58:27.333472
 22482 | tickit/date2008_pipe.txt  | 2013-02-08 20:58:28.608305
 22485 | tickit/allevents_pipe.txt | 2013-02-08 20:58:29.99489
 22487 | tickit/listings_pipe.txt  | 2013-02-08 20:58:37.632939
 22593 | tickit/allusers_pipe.txt  | 2013-02-08 21:04:08.400491
 22596 | tickit/venue_pipe.txt     | 2013-02-08 21:04:10.056055
 22598 | tickit/category_pipe.txt  | 2013-02-08 21:04:11.465049
 22600 | tickit/date2008_pipe.txt  | 2013-02-08 21:04:12.461502
 22603 | tickit/allevents_pipe.txt | 2013-02-08 21:04:14.785124
 22605 | tickit/listings_pipe.txt  | 2013-02-08 21:04:20.170594

(12 rows)
```

레코드가 이 시스템 뷰의 로그 파일에 작성된다고 해서 저장 트랜잭션(containing transaction)에서 로드가 성공적으로 커밋되었다는 것을 의미하지는 않습니다. 로드 커밋 여부를 확인하려면 STL_UTILITYTEXT 뷰에 대한 쿼리를 실행하여 COPY 트랜잭션과 일치하는 COMMIT 레코드를 찾습니다. 예를 들어 다음 쿼리는 STL_UTILITYTEXT에 대한 하위 쿼리를 기준으로 STL_LOAD_COMMITS 테이블과 STL_QUERY 테이블을 조인합니다.

```
select l.query,rtrim(l.filename),q.xid
from stl_load_commits l, stl_query q
where l.query=q.query
and exists
(select xid from stl_utilitytext where xid=q.xid and rtrim("text")='COMMIT');

 query |           rtrim           |  xid
-------+---------------------------+-------
 22600 | tickit/date2008_pipe.txt  | 68311
 22480 | tickit/category_pipe.txt  | 68066
  7508 | allusers_pipe.txt         | 23365
  7552 | category_pipe.txt         | 23415
  7576 | allevents_pipe.txt        | 23429
  7516 | venue_pipe.txt            | 23390
  7604 | listings_pipe.txt         | 23445
 22596 | tickit/venue_pipe.txt     | 68309
 22605 | tickit/listings_pipe.txt  | 68316
 22593 | tickit/allusers_pipe.txt  | 68305
 22485 | tickit/allevents_pipe.txt | 68071
```

```
  7561 | allevents_pipe.txt        | 23429
  7541 | category_pipe.txt         | 23415
  7558 | date2008_pipe.txt         | 23428
 22478 | tickit/venue_pipe.txt     | 68065
   526 | date2008_pipe.txt         |  2572
  7466 | allusers_pipe.txt         | 23365
 22482 | tickit/date2008_pipe.txt  | 68067
 22598 | tickit/category_pipe.txt  | 68310
 22603 | tickit/allevents_pipe.txt | 68315
 22475 | tickit/allusers_pipe.txt  | 68061
   547 | date2008_pipe.txt         |  2572
 22487 | tickit/listings_pipe.txt  | 68072
  7531 | venue_pipe.txt            | 23390
  7583 | listings_pipe.txt         | 23445
(25 rows)
```

다음 예에서는 is_partial 및 start_offset 열 값을 강조 표시합니다.

```
-- Single large file copy without scan range
SELECT count(*) FROM stl_load_commits WHERE query = pg_last_copy_id();
1

-- Single large uncompressed, delimited file copy with scan range
SELECT count(*) FROM stl_load_commits WHERE query = pg_last_copy_id();
16

-- Scan range offset logging in the file at 64MB boundary.
SELECT start_offset FROM stl_load_commits
WHERE query = pg_last_copy_id() ORDER BY start_offset;
0
67108864
134217728
201326592
268435456
335544320
402653184
469762048
536870912
603979776
671088640
738197504
805306368
872415232
```

```
939524096
1006632960
```

STL_LOAD_ERRORS

모든 Amazon Redshift 로드 오류에 대한 레코드를 표시합니다.

STL_LOAD_ERRORS에는 모든 Amazon Redshift 로드 오류에 대한 이력이 저장됩니다. 가능한 로드 오류와 설명을 나타낸 전체 목록은 로드 오류 참조 섹션을 참조하세요.

구문 분석 오류가 발생한 경우 STL_LOAD_ERRORS에 대한 쿼리를 실행하여 오류에 대한 일반 정보를 확인한 이후에도 정확한 데이터 행과 열 등을 포함하여 그 밖의 세부 정보를 알고 싶다면 STL_LOADERROR_DETAIL에 대한 쿼리를 실행합니다.

STL_LOAD_ERRORS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_LOAD_ERRORS에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_LOAD_ERROR_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
slice	정수	오류가 발생한 조각
tbl	정수	테이블 ID.
starttime	타임스탬프	로드 시작 시간(UTC)

열 명칭	데이터 유형	설명
session	정수	로드 실행 세션의 세션 ID
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
filename	character(256)	로드할 입력 파일의 전체 경로
line_number	bigint	로드 파일에서 오류가 발생한 라인 번호 JSON을 통한 COPY 작업일 때는 오류가 발생한 JSON 객체의 마지막 라인 번호입니다.
colname	character(127)	오류가 발생한 필드
type	character(10)	필드의 데이터 형식입니다.
col_length	character(10)	열 길이(해당되는 경우). 이 필드는 데이터 형식에 길이 제한이 있을 때 채워집니다. 예를 들어 열의 데이터 형식이 "character(3)"일 때는 이 열에 "3"의 값이 저장됩니다.
position	정수	필드의 오류 위치
raw_line	character(1024)	오류가 포함된 원시 로드 데이터. 로드 데이터의 멀티바이트 문자는 마침표로 대체됩니다.
raw_field_value	char(1024)	"colname" 필드에서 구문 분석 오류의 원인이 되는 사전 구문 분석 값
err_code	정수	오류 코드.
err_reason	character(100)	오류 설명
is_partial	정수	true(1)인 경우 COPY 작업 중에 입력 파일이 범위로 분할됨을 나타내는 값입니다. 이 값이 false(0)이면 입력 파일이 분할되지 않습니다.

열 명칭	데이터 유형	설명
start_offset	bigint	COPY 작업 중에 입력 파일이 분할되는 경우 분할의 오프셋 값(바이트 단위)을 나타내는 값입니다. 파일의 줄 번호를 알 수 없는 경우 줄 번호는 -1입니다. 파일이 분할되지 않은 경우 이 값은 0입니다.
copy_job_id	bigint	복사 작업 식별자입니다. 0은 작업 식별자가 없음을 나타냅니다.

샘플 쿼리

다음은 STL_LOAD_ERRORS를 STL_LOADERROR_DETAIL로 조인하여 가장 최근 로드 도중 발생한 오류 정보를 표시하는 쿼리입니다.

```
select d.query, substring(d.filename,14,20),
d.line_number as line,
substring(d.value,1,16) as value,
substring(le.err_reason,1,48) as err_reason
from stl_loaderror_detail d, stl_load_errors le
where d.query = le.query
and d.query = pg_last_copy_id();

 query |    substring      | line |  value   |              err_reason
-------+-------------------+------+----------+-----------------------------
    558| allusers_pipe.txt |  251 | 251      | String contains invalid or
                                               unsupported UTF8 code
    558| allusers_pipe.txt |  251 | ZRU29FGR | String contains invalid or
                                               unsupported UTF8 code
    558| allusers_pipe.txt |  251 | Kaitlin  | String contains invalid or
                                               unsupported UTF8 code
    558| allusers_pipe.txt |  251 | Walter   | String contains invalid or
                                               unsupported UTF8 code
```

다음은 STL_LOAD_ERRORS와 STV_TBL_PERM을 함께 사용하여 새로운 뷰를 생성한 다음 이 뷰에서 EVENT 테이블로 데이터를 로드할 때 발생한 오류를 확인하는 예입니다.

```
create view loadview as
(select distinct tbl, trim(name) as table_name, query, starttime,
trim(filename) as input, line_number, colname, err_code,
```

```
trim(err_reason) as reason
from stl_load_errors sl, stv_tbl_perm sp
where sl.tbl = sp.id);
```

다음은 EVENT 테이블에 로드하는 도중 마지막으로 발생한 오류를 실제로 반환하는 쿼리입니다.

```
select table_name, query, line_number, colname, starttime,
trim(reason) as error
from loadview
where table_name ='event'
order by line_number limit 1;
```

위 쿼리는 EVENT 테이블에서 발생한 마지막 로드 오류를 반환합니다. 발생한 로드 오류가 없는 경우에는 쿼리가 0개의 행을 반환합니다. 이번 예에서는 쿼리가 단일 오류를 반환합니다.

```
 table_name | query | line_number | colname | error | starttime
------+-----+----+----+-------------------------------------------------------------
+----------------------
event | 309 |  0 |  5 | Error in Timestamp value or format [%Y-%m-%d %H:%M:%S] |
 2014-04-22 15:12:44

(1 row)
```

병렬 처리를 용이하게 하기 위해 COPY 명령이 압축되지 않은 텍스트로 구분된 큰 파일 데이터를 자동으로 분할하는 경우 line_number, is_partial 및 start_offset 열은 분할과 관련된 정보를 표시합니다. (원본 파일의 줄 번호가 없을 경우 줄 번호를 알 수 없습니다.)

```
--scan ranges information
SELECT line_number, POSITION, btrim(raw_line), btrim(raw_field_value),
btrim(err_reason), is_partial, start_offset FROM stl_load_errors
WHERE query = pg_last_copy_id();

--result
-1,51,"1008771|13463413|463414|2|28.00|38520.72|0.06|0.07|NO|1998-08-30|1998-09-25|
1998-09-04|TAKE BACK RETURN|RAIL|ans cajole sly","NO","Char length exceeds DDL
 length",1,67108864
```

STL_LOADERROR_DETAIL

COPY 명령을 사용해 테이블에 로드할 때 발생한 데이터 구문 분석 오류의 로그를 표시합니다. 디스크 공간을 절약할 수 있도록 각 로드 작업마다 노드 조각 1개당 최대 20개의 오류가 기록됩니다.

구문 분석 오류는 Amazon Redshift가 데이터를 테이블에 로드하면서 데이터 행의 필드 구문을 분석할 수 없을 때 발생합니다. 예를 들어 테이블 열에는 정수 데이터 형식이 필요한데 데이터 파일의 해당 필드에는 문자열이 포함되어 있으면 구문 분석 오류가 발생합니다.

구문 분석 오류가 발생한 경우 STL_LOAD_ERRORS에 대한 쿼리를 실행하여 오류에 대한 일반 정보를 확인한 이후에도 정확한 데이터 행과 열 등을 포함하여 그 밖의 세부 정보를 알고 싶다면 STL_LOADERROR_DETAIL에 대한 쿼리를 실행하십시오.

STL_LOADERROR_DETAIL 뷰에는 구문 분석 오류가 발생한 열을 포함하여 그 전의 모든 데이터 열이 포함됩니다. VALUE 필드에서는 정확히 오류까지 구문 분석된 열을 포함하여 이 열에서 실제로 구문 분석된 데이터 값을 확인할 수 있습니다.

이 뷰는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_LOADERROR_DETAIL에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_LOAD_ERROR_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
slice	정수	오류가 발생한 조각
session	정수	로드 실행 세션의 세션 ID
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
filename	character(256)	로드할 입력 파일의 전체 경로

열 명칭	데이터 유형	설명
line_number	bigint	로드 파일에서 오류가 발생한 라인 번호
field	정수	오류가 발생한 필드
colname	character(1024)	열 이름.
값	character(1024)	구문 분석된 필드의 데이터 값. (잘릴 수도 있음) 로드 데이터의 멀티바이트 문자는 마침표로 대체됩니다.
is_null	정수	구문 분석된 값의 NULL 여부
type	character(10)	필드의 데이터 형식입니다.
col_length	character(10)	열 길이(해당되는 경우). 이 필드는 데이터 형식에 길이 제한이 있을 때 채워집니다. 예를 들어 열의 데이터 형식이 "character(3)"일 때는 이 열에 "3"의 값이 저장됩니다.

샘플 쿼리

다음은 STL_LOAD_ERRORS를 STL_LOADERROR_DETAIL로 조인하여 테이블 ID가 100133인 EVENT 테이블에 데이터를 로드할 때 발생한 구문 분석 오류의 세부 정보를 표시하는 쿼리입니다.

```
select d.query, d.line_number, d.value,
le.raw_line, le.err_reason
from stl_loaderror_detail d, stl_load_errors le
where
d.query = le.query
and tbl = 100133;
```

다음은 오류가 발생한 열을 포함하여 성공적으로 로드된 열을 나타낸 샘플 출력입니다. 이 예에서는 정수가 필요한 필드에서 문자열이 잘못 구분 분석되어 세 번째 열에서 오류가 발생하기 전에 성공적으로 로드된 열이 2개입니다. 필드에 필요했던 형식은 정수이기 때문에 비초기화 데이터(uninitialized data)인 문자열 "aaa"를 NULL로 분석하고 구문 분석 오류를 생성하였습니다. 예를 보면 원시 값과 구문 분석된 값, 그리고 오류 이유가 출력되어 있습니다.

```
query  | line_number | value | raw_line | err_reason
-------+-------------+-------+----------+----------------
```

```
4      |      3      |  1201 |  1201    | Invalid digit
4      |      3      |   126 |   126    | Invalid digit
4      |      3      |       |   aaa    | Invalid digit
(3 rows)
```

쿼리를 실행하여 STL_LOAD_ERRORS와 STL_LOADERROR_DETAIL을 조인하면 데이터 행의 각 열마다 오류 이유가 출력되며, 이는 단순히 해당 행에서 오류가 발생하였다는 것을 의미합니다. 결과에서 마지막 행이 실제로 구문 분석 오류가 발생한 열입니다.

STL_MERGE

쿼리의 병합 실행 단계를 분석합니다. 이 단계는 병렬 작업(정렬, 조인 등) 결과가 후속 처리를 위해 병합될 때 발생합니다.

STL_MERGE는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_MERGE에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호

열 명칭	데이터 유형	설명
step	정수	실행된 쿼리 단계입니다.
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .
tasknum	정수	단계 실행을 위해 할당된 쿼리 태스크 프로세스 수.
rows	bigint	처리된 총 행 수.

샘플 쿼리

다음은 병합 실행 결과 10개를 반환하는 예입니다.

```
select query, step, starttime, endtime, tasknum, rows
from stl_merge
limit 10;
```

```
 query | step |       starttime     |        endtime      | tasknum | rows
-------+------+---------------------+---------------------+---------+------
     9 |    0 | 2013-08-12 20:08:14 | 2013-08-12 20:08:14 |       0 |    0
    12 |    0 | 2013-08-12 20:09:10 | 2013-08-12 20:09:10 |       0 |    0
    15 |    0 | 2013-08-12 20:10:24 | 2013-08-12 20:10:24 |       0 |    0
    20 |    0 | 2013-08-12 20:11:27 | 2013-08-12 20:11:27 |       0 |    0
    26 |    0 | 2013-08-12 20:12:28 | 2013-08-12 20:12:28 |       0 |    0
    32 |    0 | 2013-08-12 20:14:33 | 2013-08-12 20:14:33 |       0 |    0
    38 |    0 | 2013-08-12 20:16:43 | 2013-08-12 20:16:43 |       0 |    0
    44 |    0 | 2013-08-12 20:17:05 | 2013-08-12 20:17:05 |       0 |    0
    50 |    0 | 2013-08-12 20:18:48 | 2013-08-12 20:18:48 |       0 |    0
    56 |    0 | 2013-08-12 20:20:48 | 2013-08-12 20:20:48 |       0 |    0
(10 rows)
```

STL_MERGEJOIN

쿼리의 병합 조인 실행 단계를 분석합니다.

STL_MERGEJOIN은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_MERGEJOIN에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호
step	정수	실행된 쿼리 단계입니다.
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .

열 명칭	데이터 유형	설명
tasknum	정수	단계 실행을 위해 할당된 쿼리 태스크 프로세스 수.
rows	bigint	처리된 총 행 수.
tbl	정수	테이블 ID. 병합 조인에 사용된 내부 테이블의 ID입니다.
checksum	bigint	이 정보는 내부 전용입니다.

샘플 쿼리

다음은 가장 최근 쿼리의 병합 조인 결과를 반환하는 예입니다.

```
select sum(s.qtysold), e.eventname
from event e, listing l, sales s
where e.eventid=l.eventid
and l.listid= s.listid
group by e.eventname;

select * from stl_mergejoin where query=pg_last_query_id();
```

```
 userid | query | slice | segment | step |         starttime   |          endtime    |
 tasknum | rows | tbl
--------+-------+-------+---------+------+---------------------+---------------------
+---------+------+-----
    100 | 27399 |     3 |       4 |    4 | 2013-10-02 16:30:41 | 2013-10-02 16:30:41 |
     19 |43428 | 240
    100 | 27399 |     0 |       4 |    4 | 2013-10-02 16:30:41 | 2013-10-02 16:30:41 |
     19 |43159 | 240
    100 | 27399 |     2 |       4 |    4 | 2013-10-02 16:30:41 | 2013-10-02 16:30:41 |
     19 |42778 | 240
    100 | 27399 |     1 |       4 |    4 | 2013-10-02 16:30:41 | 2013-10-02 16:30:41 |
     19 |43091 | 240
```

STL_MV_STATE

STL_MV_STATE 뷰에는 구체화된 뷰의 모든 상태 전환에 대한 행이 포함되어 있습니다.

구체화된 뷰에 대한 자세한 내용은 Amazon Redshift의 구체화된 뷰 섹션을 참조하세요.

STL_MV_STATE는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_MV_STATE에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	bigint	이벤트를 만든 사용자의 ID입니다.
starttime	타임스탬프	이벤트의 시작 시간입니다.
xid	bigint	이벤트의 트랜잭션 ID입니다.
event_desc	char(500)	상태 변경을 유발한 이벤트입니다. 다음은 값의 예입니다. • 열 유형이 변경되었습니다. • 열이 삭제되었습니다. • 열 이름이 바뀌었습니다. • 스키마 이름이 변경되었습니다. • 작은 테이블 변환 • TRUNCATE • Vacuum 참고로 이 열은 다른 값도 가질 수 있습니다.
db_name	char(128)	구체화된 보기를 포함하는 데이터베이스입니다.
base_table_schema	char(128)	기본 테이블의 스키마입니다.
base_table_name	char(128)	기본 테이블의 이름입니다.

열 명칭	데이터 유형	설명
mv_schema	char(128)	구체화된 보기의 스키마입니다.
mv_name	char(128)	구체화된 보기의 이름입니다.
state	character(32)	구체화된 보기의 변경된 상태는 다음과 같습니다. • 다시 계산 • 새로 고칠 수 없음

다음 표에 `event_desc`와 `state` 조합의 예가 나와 있습니다.

```
          event_desc     |     state
-------------------------+---------------
 TRUNCATE                | Recompute
 TRUNCATE                | Recompute
 Small table conversion  | Recompute
 Vacuum                  | Recompute
 Column was renamed      | Unrefreshable
 Column was dropped      | Unrefreshable
 Table was renamed       | Unrefreshable
 Column type was changed | Unrefreshable
 Schema name was changed | Unrefreshable
```

샘플 쿼리

구체화된 보기의 상태 전환 로그를 보려면 다음 쿼리를 실행합니다.

```
select * from stl_mv_state;
```

위 쿼리는 다음과 같은 샘플 출력을 반환합니다.

```
 userid |         starttime          | xid  |            event_desc       | db_name |
 base_table_schema    |   base_table_name    |      mv_schema       | mv_name       |
  state
--------+----------------------------+------+-----------------------------+---------
+---------------------+---------------------+---------------------+--------------
+---------------
```

```
   138 | 2020-02-14 02:21:25.578885 | 5180 | TRUNCATE                    | dev     |
public               | mv_base_table        | public               | mv_test       |
Recompute
   138 | 2020-02-14 02:21:56.846774 | 5275 | Column was dropped          | dev     |
                     | mv_base_table        | public               | mv_test       |
Unrefreshable
   100 | 2020-02-13 22:09:53.041228 | 1794 | Column was renamed          | dev     |
                     | mv_base_table        | public               | mv_test       |
Unrefreshable
     1 | 2020-02-13 22:10:23.630914 | 1893 | ALTER TABLE ALTER SORTKEY   | dev     |
public               | mv_base_table_sorted | public               | mv_test       |
Recompute
     1 | 2020-02-17 22:57:22.497989 | 8455 | ALTER TABLE ALTER DISTSTYLE | dev     |
public               | mv_base_table        | public               | mv_test       |
Recompute
   173 | 2020-02-17 22:57:23.591434 | 8504 | Table was renamed           | dev     |
                     | mv_base_table        | public               | mv_test       |
Unrefreshable
   173 | 2020-02-17 22:57:27.229423 | 8592 | Column type was changed     | dev     |
                     | mv_base_table        | public               | mv_test       |
Unrefreshable
   197 | 2020-02-17 22:59:06.212569 | 9668 | TRUNCATE                    | dev     |
schemaf796e415850f4f | mv_base_table        | schemaf796e415850f4f | mv_test       |
Recompute
   138 | 2020-02-14 02:21:55.705655 | 5226 | Column was renamed          | dev     |
                     | mv_base_table        | public               | mv_test       |
Unrefreshable
     1 | 2020-02-14 02:22:26.292434 | 5325 | ALTER TABLE ALTER SORTKEY   | dev     |
public               | mv_base_table_sorted | public               | mv_test       |
Recompute
```

STL_NESTLOOP

쿼리의 중첩 루프 조인 실행 단계를 분석합니다.

STL_NESTLOOP는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_NESTLOOP에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호
step	정수	실행된 쿼리 단계입니다.
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
tasknum	정수	단계 실행을 위해 할당된 쿼리 태스크 프로세스 수.
rows	bigint	처리된 총 행 수.
tbl	정수	테이블 ID.
checksum	bigint	이 정보는 내부 전용입니다.

샘플 쿼리

다음 쿼리는 CATEGORY 테이블을 조인하지 않아서 부분적으로 데카르트 곱을 산출하지만 권장하지는 않습니다. 그럼에도 불구하고 여기에 표시하는 이유는 중첩 루프를 설명하기 위해서입니다.

```
select count(event.eventname), event.eventname, category.catname, date.caldate
from event, category, date
where event.dateid = date.dateid
group by event.eventname, category.catname, date.caldate;
```

다음은 위 쿼리의 결과를 STL_NESTLOOP 뷰에 표시하는 쿼리입니다.

```
select query, slice, segment as seg, step,
datediff(msec, starttime, endtime) as duration, tasknum, rows, tbl
from stl_nestloop
where query = pg_last_query_id();
```

```
 query | slice | seg | step | duration | tasknum | rows  | tbl
-------+-------+-----+------+----------+---------+-------+-----
  6028 |     0 |   4 |    5 |       41 |      22 | 24277 | 240
  6028 |     1 |   4 |    5 |       26 |      23 | 24189 | 240
  6028 |     3 |   4 |    5 |       25 |      23 | 24376 | 240
  6028 |     2 |   4 |    5 |       54 |      22 | 23936 | 240
```

STL_PARSE

로딩을 위한 문자열을 이진 값으로 구문 분석하는 쿼리 단계를 분석합니다.

STL_PARSE는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_PARSE에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호
step	정수	실행된 쿼리 단계입니다.
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .
tasknum	정수	단계 실행을 위해 할당된 쿼리 태스크 프로세스 수.
rows	bigint	처리된 총 행 수.

샘플 쿼리

다음은 문자열을 이진 값으로 구문 분석했을 때 조각 1과 세그먼트 0에 대한 모든 쿼리 단계 결과를 반환하는 예입니다.

```
select query, step, starttime, endtime, tasknum, rows
from stl_parse
where slice=1 and segment=0;
```

```
 query | step |     starttime       |        endtime      | tasknum |  rows
-------+------+---------------------+---------------------+---------+--------
   669 |    1 | 2013-08-12 22:35:13 | 2013-08-12 22:35:17 |      32 | 192497
   696 |    1 | 2013-08-12 22:35:49 | 2013-08-12 22:35:49 |      32 |      0
```

```
   525 |    1 | 2013-08-12 22:32:03 | 2013-08-12 22:32:03 |      13 |  49990
   585 |    1 | 2013-08-12 22:33:18 | 2013-08-12 22:33:19 |      13 |    202
   621 |    1 | 2013-08-12 22:34:03 | 2013-08-12 22:34:03 |      27 |    365
   651 |    1 | 2013-08-12 22:34:47 | 2013-08-12 22:34:53 |      35 | 192497
   590 |    1 | 2013-08-12 22:33:28 | 2013-08-12 22:33:28 |      19 |      0
   599 |    1 | 2013-08-12 22:33:39 | 2013-08-12 22:33:39 |      31 |     11
   675 |    1 | 2013-08-12 22:35:26 | 2013-08-12 22:35:27 |      38 |   3766
   567 |    1 | 2013-08-12 22:32:47 | 2013-08-12 22:32:48 |      23 |  49990
   630 |    1 | 2013-08-12 22:34:17 | 2013-08-12 22:34:17 |      36 |      0
   572 |    1 | 2013-08-12 22:33:04 | 2013-08-12 22:33:04 |      29 |      0
   645 |    1 | 2013-08-12 22:34:37 | 2013-08-12 22:34:38 |      29 |   8798
   604 |    1 | 2013-08-12 22:33:47 | 2013-08-12 22:33:47 |      37 |      0
(14 rows)
```

STL_PLAN_INFO

STL_PLAN_INFO 뷰는 쿼리의 EXPLAIN 출력을 행 집합으로 표시하는 데 사용됩니다. 이 테이블은 쿼리 계획을 살펴볼 수 있는 대체 방법이기도 합니다.

STL_PLAN_INFO는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_PLAN_INFO에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.

열 명칭	데이터 유형	설명
nodeid	정수	계획 노드 식별자. 쿼리 실행 시 이 식별자를 통해 노드가 1개 이상의 단계로 매핑됩니다.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호
step	정수	쿼리 단계를 식별할 수 있는 번호
locus	정수	단계가 실행되는 위치로 컴퓨터 노드이면 0이고, 리더 노드이면 1입니다.
plannode	정수	계획 노드의 열거 값. plannode 열의 열거형 값은 다음 테이블을 참조하십시오. (STL_EXPLAIN 테이블의 PLANNODE 열에는 계획 노드 텍스트가 포함됩니다)
startupcost	double precision	이 단계의 첫 번째 행을 반환하는 상대적 예상 비용
totalcost	double precision	단계를 실행하는 상대적 예상 비용
rows	bigint	단계에서 산출될 것으로 예상되는 행의 수
bytes	bigint	단계에서 산출될 것으로 예상되는 바이트 수

샘플 쿼리

다음은 단순한 SELECT 쿼리에서 EXPLAIN 명령을 사용하여 반환되는 쿼리 계획과 STL_PLAN_INFO 뷰에 대한 쿼리를 실행하여 반환되는 쿼리 계획을 비교하는 예입니다.

```
explain select * from category;
QUERY PLAN
------------------------------------------------------------
XN Seq Scan on category (cost=0.00..0.11 rows=11 width=49)
(1 row)

select * from category;
catid | catgroup | catname | catdesc
-------+----------+-----------+--------------------------------------------
1 | Sports | MLB | Major League Baseball
3 | Sports | NFL | National Football League
5 | Sports | MLS | Major League Soccer
```

```
...

select * from stl_plan_info where query=256;

query | nodeid | segment | step | locus | plannode | startupcost | totalcost
| rows | bytes
-------+--------+---------+------+-------+----------+-------------+-----------+------
+-------
256 | 1 | 0 | 1 | 0 | 104 | 0 | 0.11 | 11 | 539
256 | 1 | 0 | 0 | 0 | 104 | 0 | 0.11 | 11 | 539
(2 rows)
```

이번 예에서 PLANNODE 104는 CATEGORY 테이블에 대한 순차적 스캔을 의미합니다.

```
select distinct eventname from event order by 1;

eventname
----------------------------------------------------------------------
.38 Special
3 Doors Down
70s Soul Jam
A Bronx Tale
...

explain select distinct eventname from event order by 1;

QUERY PLAN
------------------------------------------------------------------------------------------
XN Merge (cost=1000000000136.38..1000000000137.82 rows=576 width=17)
Merge Key: eventname
-> XN Network (cost=1000000000136.38..1000000000137.82 rows=576
width=17)
Send to leader
-> XN Sort (cost=1000000000136.38..1000000000137.82 rows=576
width=17)
Sort Key: eventname
-> XN Unique (cost=0.00..109.98 rows=576 width=17)
-> XN Seq Scan on event (cost=0.00..87.98 rows=8798
width=17)
(8 rows)

select * from stl_plan_info where query=240 order by nodeid desc;
```

```
query | nodeid | segment | step | locus | plannode | startupcost |
totalcost | rows | bytes
-------+--------+---------+------+-------+----------+-----------------
+-----------------+------+--------
240 | 5 | 0 | 0 | 0 | 104 | 0                | 87.98   | 8798 | 149566
240 | 5 | 0 | 1 | 0 | 104 | 0                | 87.98   | 8798 | 149566
240 | 4 | 0 | 2 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 4 | 0 | 3 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 4 | 1 | 0 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 4 | 1 | 1 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 3 | 1 | 2 | 0 | 114 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 3 | 2 | 0 | 0 | 114 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 2 | 2 | 1 | 0 | 123 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 1 | 3 | 0 | 0 | 122 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
(10 rows)
```

STL_PROJECT

표현식을 평가하는 데 사용되는 쿼리 단계의 행이 저장됩니다.

STL_PROJECT는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_PROJECT에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.

열 명칭	데이터 유형	설명
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호
step	정수	실행된 쿼리 단계입니다.
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
tasknum	정수	단계 실행을 위해 할당된 쿼리 태스크 프로세스 수.
rows	bigint	처리된 총 행 수.
checksum	bigint	이 정보는 내부 전용입니다.

샘플 쿼리

다음은 조각 0과 세그먼트 1일 때 표현식을 평가하는 데 사용한 쿼리 단계의 행을 모두 반환하는 예입니다.

```
select query, step, starttime, endtime, tasknum, rows
from stl_project
where slice=0 and segment=1;
```

```
 query  | step |         starttime   |          endtime    | tasknum | rows
--------+------+---------------------+---------------------+---------+------
  86399 |    2 | 2013-08-29 22:01:21 | 2013-08-29 22:01:21 |      25 |   -1
  86399 |    3 | 2013-08-29 22:01:21 | 2013-08-29 22:01:21 |      25 |   -1
    719 |    1 | 2013-08-12 22:38:33 | 2013-08-12 22:38:33 |       7 |   -1
  86383 |    1 | 2013-08-29 21:58:35 | 2013-08-29 21:58:35 |       7 |   -1
```

```
   714 |    1 | 2013-08-12 22:38:17 | 2013-08-12 22:38:17 |       2 |   -1
 86375 |    1 | 2013-08-29 21:57:59 | 2013-08-29 21:57:59 |       2 |   -1
 86397 |    2 | 2013-08-29 22:01:20 | 2013-08-29 22:01:20 |      19 |   -1
   627 |    1 | 2013-08-12 22:34:13 | 2013-08-12 22:34:13 |      34 |   -1
 86326 |    2 | 2013-08-29 21:45:28 | 2013-08-29 21:45:28 |      34 |   -1
 86326 |    3 | 2013-08-29 21:45:28 | 2013-08-29 21:45:28 |      34 |   -1
 86325 |    2 | 2013-08-29 21:45:27 | 2013-08-29 21:45:27 |      28 |   -1
 86371 |    1 | 2013-08-29 21:57:42 | 2013-08-29 21:57:42 |       4 |   -1
111100 |    2 | 2013-09-03 19:04:45 | 2013-09-03 19:04:45 |      12 |   -1
   704 |    2 | 2013-08-12 22:36:34 | 2013-08-12 22:36:34 |      37 |   -1
   649 |    2 | 2013-08-12 22:34:47 | 2013-08-12 22:34:47 |      38 |   -1
   649 |    3 | 2013-08-12 22:34:47 | 2013-08-12 22:34:47 |      38 |   -1
   632 |    2 | 2013-08-12 22:34:22 | 2013-08-12 22:34:22 |      13 |   -1
   705 |    2 | 2013-08-12 22:36:48 | 2013-08-12 22:36:49 |      13 |   -1
   705 |    3 | 2013-08-12 22:36:48 | 2013-08-12 22:36:49 |      13 |   -1
     3 |    1 | 2013-08-12 20:07:40 | 2013-08-12 20:07:40 |       3 |   -1
 86373 |    1 | 2013-08-29 21:57:58 | 2013-08-29 21:57:58 |       3 |   -1
107976 |    1 | 2013-09-03 04:05:12 | 2013-09-03 04:05:12 |       3 |   -1
 86381 |    1 | 2013-08-29 21:58:35 | 2013-08-29 21:58:35 |       8 |   -1
 86396 |    1 | 2013-08-29 22:01:20 | 2013-08-29 22:01:20 |      15 |   -1
   711 |    1 | 2013-08-12 22:37:10 | 2013-08-12 22:37:10 |      20 |   -1
 86324 |    1 | 2013-08-29 21:45:27 | 2013-08-29 21:45:27 |      24 |   -1
(26 rows)
```

STL_QUERY

데이터베이스 쿼리에 대한 실행 정보를 반환합니다.

> **Note**
>
> STL_QUERY 및 STL_QUERYTEXT 뷰에는 기타 유틸리티나 DDL 명령이 아닌 쿼리 정보만 저장됩니다. Amazon Redshift에서 실행되는 모든 문에 대한 목록 및 정보는 STL_DDLTEXT 보기와 STL_UTILITYTEXT 보기에 대한 쿼리를 실행하여 확인할 수 있습니다. Amazon Redshift에서 실행되는 모든 문에 대한 전체 목록은 SVL_STATEMENTTEXT 보기에 대한 쿼리를 실행하여 확인할 수 있습니다.

STL_QUERY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
레이블	character(320)	쿼리 실행에 사용되는 파일의 이름 또는 SET QUERY_GROUP 명령을 사용하여 정의되는 레이블. 쿼리가 파일 기반이 아니거나 QUERY_GROUP 파라미터가 설정되지 않은 경우, 이 필드의 값은 `default`입니다.
xid	bigint	트랜잭션 ID.
pid	정수	프로세스 ID. 일반적으로 한 세션의 모든 쿼리는 동일 프로세스에서 실행됩니다. 따라서 동일 세션에서 일련의 쿼리를 실행하는 경우에는 이 값은 대부분 같은 값을 유지합니다. Amazon Redshift는 특정한 내부 이벤트 이후에 활성 세션을 다시 시작하고 새 PID를 할당할 수도 있습니다. 자세한 내용은 STL_RESTARTED_SESSIONS 단원을 참조하십시오.
데이터베이스	character(32)	쿼리가 실행되었을 때 사용자가 연결된 데이터베이스의 이름.
querytxt	character(4000)	실제 쿼리 텍스트
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .

열 명칭	데이터 유형	설명
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .
aborted	정수	쿼리가 시스템에 의해 중지되거나 사용자에 의해 취소되는 경우, 이 열에 **1**이 포함됩니다. 쿼리를 끝까지 실행한 경우에는(클라이언트에게 결과를 반환하는 단계까지 포함) 이 열에 **0**이 포함됩니다. 결과가 수신되기 전에 클라이언트 연결이 끊어지면 쿼리가 백엔드에서 성공적으로 완료되더라도 취소됨(**1**)으로 표시됩니다.
insert_pristine	정수	현재 쿼리를 지금/이전에 실행했을 때 쓰기 쿼리를 지금/이전에 실행할 수 있는 여부입니다. 1 = 쓰기 쿼리가 허용되지 않음. 0 = 쓰기 쿼리가 허용됨. 이 열은 디버깅 시 사용하도록 되어 있습니다.
concurrency_scaling_status	정수	쿼리가 기본 클러스터에서 실행되었는지 아니면 동시성 확장 클러스터에서 실행되었는지를 나타냅니다. 가능한 값은 다음과 같습니다. 0 - 기본 클러스터에서 실행되었습니다. 1 - 동시성 확장 클러스터에서 실행되었습니다. 1보다 큼 - 기본 클러스터에서 실행되었습니다.

샘플 쿼리

다음은 가장 최근 쿼리 5개를 나열하는 쿼리입니다.

```
select query, trim(querytxt) as sqlquery
from stl_query
order by query desc limit 5;

query |                                  sqlquery
------+---------------------------------------------------
129 | select query, trim(querytxt) from stl_query order by query;
128 | select node from stv_disk_read_speeds;
```

```
127 | select system_status from stv_gui_status
126 | select * from systable_topology order by slice
125 | load global dict registry
(5 rows)
```

다음은 2013년 2월 15일에 실행한 쿼리의 경과 시간을 내림차순으로 반환하는 쿼리입니다.

```
select query, datediff(seconds, starttime, endtime),
trim(querytxt) as sqlquery
from stl_query
where starttime >= '2013-02-15 00:00' and endtime < '2013-02-16 00:00'
order by date_diff desc;

 query | date_diff |  sqlquery
-------+-----------+-------------------------------------------
 55    |       119 | padb_fetch_sample: select count(*) from category
121    |         9 | select * from svl_query_summary;
181    |         6 | select * from svl_query_summary where query in(179,178);
172    |         5 | select * from svl_query_summary where query=148;
...
(189 rows)
```

다음 쿼리는 쿼리에 대한 대기열 시간 및 실행 시간을 보여줍니다. 동시성 확장 클러스터에서 실행된 `concurrency_scaling_status = 1`의 쿼리입니다. 기본 클러스터에서 실행된 다른 모든 쿼리입니다.

```
SELECT w.service_class AS queue
     , q.concurrency_scaling_status
     , COUNT( * ) AS queries
     , SUM( q.aborted )  AS aborted
     , SUM( ROUND( total_queue_time::NUMERIC / 1000000,2 ) ) AS queue_secs
     , SUM( ROUND( total_exec_time::NUMERIC / 1000000,2 ) )  AS exec_secs
FROM stl_query q
     JOIN stl_wlm_query w
          USING (userid,query)
WHERE q.userid > 1
  AND service_class > 5
  AND q.starttime > '2019-03-01 16:38:00'
  AND q.endtime   < '2019-03-01 17:40:00'
GROUP BY 1,2
ORDER BY 1,2;
```

STL_QUERY_METRICS

사용자 정의 쿼리 대기열(서비스 클래스)에서 실행을 마친 쿼리의 지표 정보(처리된 행의 수, CPU 사용량, 입/출력, 디스크 사용량)가 저장됩니다. 현재 실행 중인 활성 쿼리의 지표를 보려면 STV_QUERY_METRICS 시스템 뷰를 참조하세요.

쿼리 지표는 1초 간격으로 샘플링됩니다. 결과적으로 동일한 쿼리라고 해도 다르게 실행하면 약간 다른 시간을 반환할 수 있습니다. 또한 1초 이내에 실행되는 쿼리 세그먼트는 기록되지 않을 수도 있습니다.

STL_QUERY_METRICS은 지표를 쿼리, 세그먼트 및 단계 수준으로 추적 및 집계합니다. 쿼리 세그먼트 및 단계에 대한 자세한 내용은 쿼리 계획 및 실행 워크플로우 섹션을 참조하세요. 노드 조각에서 합산되는 지표도 max_rows, cpu_time 등 매우 많습니다. 노드 조각에 대한 자세한 내용은 데이터 웨어하우스 시스템 아키텍처 섹션을 참조하세요.

행이 지표를 보고하는 수준을 확인하고 싶으면 다음과 같이 segment 열과 step_type 열을 살펴보십시오.

- segment 열과 step_type 열의 값이 모두 -1이면 행이 쿼리 수준에서 지표를 보고합니다.
- segment 열이 -1이 아니고, step_type 열이 -1이면 행이 세그먼트 수준에서 지표를 보고합니다.
- segment 열과 step_type 열의 값이 모두 -1이 아니면 행이 단계 수준에서 지표를 보고합니다.

SVL_QUERY_METRICS 뷰와 SVL_QUERY_METRICS_SUMMARY 뷰는 이 뷰의 데이터를 집계하여 액세스가 가능한 형식으로 정보를 제공합니다.

STL_QUERY_METRS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_DETAIL에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 쿼리를 실행한 사용자의 ID.

열 명칭	데이터 유형	설명
service_class	정수	서비스 클래스의 ID. 쿼리 대기열은 WLM 구성에서 정의합니다. 지표는 사용자 정의 대기열에 대해서만 보고됩니다.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
segment	정수	세그먼트 번호. 하나의 쿼리는 여러 세그먼트로 구성되며, 각각의 세그먼트는 하나 이상의 단계로 구성됩니다. 쿼리 세그먼트는 병렬로 실행될 수 있습니다. 각 세그먼트는 단일 프로세스에서 실행됩니다. 세그먼트 값이 -1인 경우, 지표 세그먼트 값은 쿼리 수준으로 롤업됩니다.
step_type	정수	실행된 단계의 유형입니다. 단계 유형에 대한 자세한 내용은 단계 유형 섹션을 참조하세요.
starttime	타임스탬프	쿼리 실행이 시작된 UTC 시간으로 소수 초의 정밀도는 6자리입니다. 예: `2009-06-12 11:29:19.131358` .
슬라이스	정수	클러스터의 조각 수
max_rows	bigint	모든 조각을 집계하여 단계에서 출력된 행의 최대 수
rows	bigint	단계에서 처리된 행의 수
max_cpu_time	bigint	사용된 최대 CPU 시간(마이크로초). 세그먼트 수준일 때는 모든 조각을 통틀어 세그먼트에서 사용된 최대 CPU 시간입니다. 쿼리 수준일 때는 모든 쿼리 세그먼트에서 사용된 최대 CPU 시간입니다.
cpu_time	bigint	사용된 CPU 시간(마이크로초). 세그먼트 수준일 때는 모든 조각을 통틀어 세그먼트에서 사용된 총 CPU 시간입니다. 쿼리 수준일 때는 모든 조각과 세그먼트를 통틀어 쿼리에서 사용된 CPU 시간의 합산입니다.

열 명칭	데이터 유형	설명
max_blocks_read	bigint	모든 조각을 집계하여 세그먼트에서 읽은 1MB 블록의 최대 수. 세그먼트 수준일 때는 모든 조각을 통틀어 세그먼트에서 읽은 1MB 블록의 최대 수입니다. 쿼리 수준일 때는 모든 쿼리 세그먼트에서 읽은 1MB 블록의 최대 수입니다.
blocks_read	bigint	쿼리 또는 세그먼트에서 읽은 1MB 블록의 수
max_run_time	bigint	세그먼트의 최대 경과 시간(마이크로초). 세그먼트 수준일 때는 모든 조각에서 세그먼트의 최대 실행 시간입니다. 쿼리 수준일 때는 모든 쿼리 세그먼트의 최대 실행 시간입니다.
run_time	bigint	조각마다 합산한 총 실행 시간. 실행 시간에는 대기 시간이 포함되지 않습니다. 세그먼트 수준일 때는 모든 조각을 통틀어 합산된 세그먼트 실행 시간입니다. 쿼리 수준일 때는 모든 조각과 세그먼트를 통틀어 합산된 쿼리 실행 시간입니다. 이 값은 합산된 것이기 때문에 실행 시간과 쿼리 실행 시간은 관련이 없습니다.
max_blocks_to_disk	bigint	중간 결과를 작성하는 데 사용한 디스크 공간의 최대 크기(MB 블록). 세그먼트 수준일 때는 모든 조각을 통틀어 세그먼트에서 사용된 디스크 공간의 최대 크기입니다. 쿼리 수준일 때는 모든 쿼리 세그먼트에서 사용된 디스크 공간의 최대 크기입니다.
blocks_to_disk	bigint	쿼리 또는 세그먼트에서 중간 결과를 작성하는 데 사용한 디스크 공간 크기(MB 블록)
step	정수	실행된 쿼리 단계입니다.
max_query_scan_size	bigint	쿼리에서 스캔된 데이터의 최대 크기(MB). 세그먼트 수준일 때는 모든 조각을 통틀어 세그먼트에서 스캔된 데이터의 최대 크기입니다. 쿼리 수준일 때는 모든 쿼리 세그먼트에서 스캔된 데이터의 최대 크기입니다.
query_scan_size	bigint	쿼리에서 스캔된 데이터의 크기(MB).

열 명칭	데이터 유형	설명
query_priority	정수	쿼리의 우선순위입니다. 가능한 값은 -1, 0, 1, 2, 3 및 4입니다. 여기에서 -1은 쿼리 우선 순위가 지원되지 않음을 뜻합니다.
query_queue_time	bigint	쿼리가 대기열에 있는 시간(마이크로초)입니다.
service_class_name	character (64)	서비스 클래스의 이름입니다.

샘플 쿼리

CPU 시간이 높은(1,000초 이상) 쿼리를 찾으려면 다음과 같이 쿼리를 실행합니다.

```
Select query, cpu_time / 1000000 as cpu_seconds
from stl_query_metrics where segment = -1 and cpu_time > 1000000000
order by cpu_time;

query | cpu_seconds
------+------------
25775 |        9540
```

중첩 루프 조인을 사용하여 반환된 행의 수가 100만 개가 넘는 활성 쿼리를 찾으려면 다음과 같이 쿼리를 실행합니다.

```
select query, rows
from stl_query_metrics
where step_type = 15 and rows > 1000000
order by rows;

query | rows
------+-----------
25775 | 2621562702
```

실행 시간이 60초보다 크고, CPU 시간이 10초 미만인 활성 쿼리를 찾으려면 다음과 같이 쿼리를 실행하십시오.

```
select query, run_time/1000000 as run_time_seconds
```

```
from stl_query_metrics
where segment = -1 and run_time > 60000000 and cpu_time < 10000000;

query | run_time_seconds
------+-----------------
25775 |              114
```

STL_QUERYTEXT

SQL 명령의 쿼리 텍스트를 수집합니다.

STL_QUERYTEXT 뷰에 대한 쿼리를 실행하여 다음 문에 대해 기록된 SQL을 수집합니다.

- SELECT, SELECT INTO
- INSERT, UPDATE, DELETE
- COPY
- UNLOAD
- VACUUM 및 ANALYZE를 실행하여 생성된 쿼리
- CREATE TABLE AS (CTAS)

일정 기간 위의 문 작업에 대한 쿼리를 실행하려면 STL_QUERYTEXT 뷰와 STL_QUERY 뷰를 조인합니다.

> **Note**
>
> STL_QUERY 및 STL_QUERYTEXT 뷰에는 기타 유틸리티나 DDL 명령이 아닌 쿼리 정보만 저장됩니다. Amazon Redshift에서 실행되는 모든 문에 대한 목록 및 정보는 STL_DDLTEXT 보기와 STL_UTILITYTEXT 보기에 대한 쿼리를 실행하여 확인할 수 있습니다. Amazon Redshift에서 실행되는 모든 문에 대한 전체 목록은 SVL_STATEMENTTEXT 보기에 대한 쿼리를 실행하여 확인할 수 있습니다.

STL_DDLTEXTSTL_UTILITYTEXT 및 도 참조하십시오.SVL_STATEMENTTEXT

STL_QUERYTEXT는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_TEXT에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
xid	bigint	트랜잭션 ID.
pid	정수	프로세스 ID. 일반적으로 한 세션의 모든 쿼리는 동일 프로세스에서 실행됩니다. 따라서 동일 세션에서 일련의 쿼리를 실행하는 경우에는 이 값은 대부분 같은 값을 유지합니다. Amazon Redshift는 특정한 내부 이벤트 이후에 활성 세션을 다시 시작하고 새 PID를 할당할 수도 있습니다. 자세한 내용은 STL_RESTARTED_SESSIONS 단원을 참조하십시오. 이 열을 사용하여 STL_ERROR 뷰에 조인할 수 있습니다.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
SEQUENCE	정수	단일 문에 200자 이상이 포함된 경우, 해당 문에 대해 추가 행이 기록됩니다. 시퀀스 0이 첫 번째 행이고 1이 두 번째 행이 되는 방식입니다.
텍스트	character(200)	200자씩 증가하는 SQL 텍스트. 이 필드에는 백슬래시(\\) 및 줄 바꿈(\n) 등의 특수 문자가 포함될 수 있습니다.

샘플 쿼리

PG_BACKEND_PID() 함수를 사용하여 현재 세션에 대한 정보를 가져올 수 있습니다. 예를 들어 다음은 현재 세션에서 완료된 쿼리에 대해 쿼리 ID와 쿼리 텍스트 일부를 반환하는 쿼리입니다.

```
select query, substring(text,1,60)
from stl_querytext
where pid = pg_backend_pid()
order by query desc;
```

```
 query |                          substring
-------+-------------------------------------------------------------
 28262 | select query, substring(text,1,80) from stl_querytext where
 28252 | select query, substring(path,0,80) as path from stl_unload_l
 28248 | copy category from 's3://dw-tickit/manifest/category/1030_ma
 28247 | Count rows in target table
 28245 | unload ('select * from category') to 's3://dw-tickit/manifes
 28240 | select query, substring(text,1,40) from stl_querytext where
(6 rows)
```

저장된 SQL 재구성

STL_QUERYTEXT의 `text` 열에 저장된 SQL을 재구성하려면 SELECT 문을 실행하여 `text` 열의 1개 이상의 부분에서 SQL을 생성합니다. 재구성된 SQL을 실행하기 전에 모든 (\n) 특수 문자를 줄 바꿈으로 바꿉니다. 다음 SELECT 문의 결과는 `query_statement` 필드에서 재구성된 SQL의 행입니다.

```
SELECT query, LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END)
 WITHIN GROUP (ORDER BY sequence) as query_statement, COUNT(*) as row_count
FROM stl_querytext GROUP BY query ORDER BY query desc;
```

예를 들어 다음 쿼리는 열 3개를 선택합니다. 쿼리 자체는 200자 이상이며 STL_QUERYTEXT에 여러 부분으로 저장됩니다.

```
select
1 AS a01234567890123456789012345678901234567890123456789012345678901234567890,
2 AS b01234567890123456789012345678901234567890123456789012345678901234567890,
3 AS b01234567890123456789012345678901234567890123456789012345678901234
FROM stl_querytext;
```

이 예제에서 이 쿼리는 STL_QUERYTEXT의 `text` 열에 많은 두 부분(행)으로 저장됩니다.

```
select query, sequence, text
from stl_querytext where query=pg_last_query_id() order by query desc, sequence limit
 10;
```

```
query  | sequence |
                         text
```

```
-------+----------
+-----------------------------------------------------------------------------------------------------------------
    45 |        0 | select\n1 AS
 a01234567890123456789012345678901234567890123456789012345678901234567890,\n2 AS
 b01234567890123456789012345678901234567890123456789012345678901234567890,\n3 AS
 b01234567890123456789012345678901234567890123456
    45 |        1 | \nFROM stl_querytext;
```

STL_QUERYTEXT에 저장된 SQL을 재구성하려면 다음 SQL을 실행합니다.

```
select LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END, '')
 within group (order by sequence) AS text
from stl_querytext where query=pg_last_query_id();
```

클라이언트에서 재구성된 SQL을 사용하려면 모든 (\n) 특수 문자를 줄 바꿈으로 바꿉니다.

```


                    text


-----------------------------------------------------------------------------------------------------------------
 select\n1 AS a01234567890123456789012345678901234567890123456789012345678901234567890,
\n2 AS b01234567890123456789012345678901234567890123456789012345678901234567890,\n3 AS
 b01234567890123456789012345678901234567890123456\nFROM stl_querytext;
```

STL_REPLACEMENTS

ACCEPTINVCHARS 옵션과 함께 COPY 명령을 실행하여 잘못된 UTF-8 문자를 대체했을 때 기록되는 로그를 표시합니다. 적어도 1개 이상 대체가 필요했던 각 노드 조각에서 첫 번째 100개 행마다 로그 항목이 STL_REPLACEMENTS에 추가됩니다.

STL_REPLACEMENTS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_NESTLOOP에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에

대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_COPY_REPLACEMENTS을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	대체가 이루어진 노드 조각 번호
tbl	정수	테이블 ID.
starttime	타임스탬프	COPY 명령 시작 시간(UTC)
session	정수	COPY 명령을 실행하는 세션의 세션 ID
filename	character (256)	COPY 명령을 실행할 입력 파일의 전체 경로
line_number	bigint	입력 데이터 파일에서 잘못된 UTF-8 문자가 포함된 라인 번호 -1은 열 데이터 파일에서 복사할 때와 같이 라인 번호를 사용할 수 없음을 나타냅니다.
colname	character (127)	잘못된 UTF-8 문자가 포함된 첫 번째 필드
raw_line	character (1024)	잘못된 UTF-8 문자가 포함된 원시 로드 데이터

샘플 쿼리

다음은 가장 최근 COPY 작업에서 대체된 파일을 반환하는 예입니다.

```
select query, session, filename, line_number, colname
from stl_replacements
where query = pg_last_copy_id();

 query | session |   filename                                    | line_number | colname
 ------+---------+-----------------------------------------------+-------------+--------
    96 |    6314 | s3://DOC-EXAMPLE-BUCKET/allusers_pipe.txt   |         251 | city
    96 |    6314 | s3://DOC-EXAMPLE-BUCKET/allusers_pipe.txt   |         317 | city
    96 |    6314 | s3://DOC-EXAMPLE-BUCKET/allusers_pipe.txt   |         569 | city
    96 |    6314 | s3://DOC-EXAMPLE-BUCKET/allusers_pipe.txt   |         623 | city
    96 |    6314 | s3://DOC-EXAMPLE-BUCKET/allusers_pipe.txt   |         694 | city
...
```

STL_RESTARTED_SESSIONS

Amazon Redshift는 특정 내부 이벤트 이후 가용성을 계속해서 유지하기 위해 새로운 프로세스 ID(PID)로 활성 세션을 다시 시작할 수도 있습니다. Amazon Redshift가 세션을 다시 시작하면 STL_RESTARTED_SESSIONS가 새로운 PID와 이전 PID를 기록합니다.

자세한 내용은 이 섹션의 다음 예를 참조하세요.

STL_RESTARTED_SESSIONS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_SESSION_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
currenttime	타임스탬프	이벤트 시간
dbname	character (50)	세션과 연결된 데이터베이스 이름
newpid	정수	다시 시작된 세션의 프로세스 ID

열 명칭	데이터 유형	설명
oldpid	정수	이전 세션의 프로세스 ID
사용자 이름	character (50)	사용자 이름
remotehost	character (45)	원격 호스트의 이름 또는 IP 주소
remoteport	character (32)	원격 호스트의 포트 번호
parkedtime	타임스탬프	이 정보는 내부 전용입니다.
session_vars	character (2000)	이 정보는 내부 전용입니다.

샘플 쿼리

다음은 STL_RESTARTED_SESSIONS를 STL_SESSIONS와 조인하여 다시 시작된 세션의 사용자 이름을 나타내는 예입니다.

```
select process, stl_restarted_sessions.newpid, user_name
from stl_sessions
inner join stl_restarted_sessions on stl_sessions.process =
 stl_restarted_sessions.oldpid
order by process;

...
```

STL_RETURN

쿼리의 반환 단계 정보가 저장됩니다. 반환 단계에서는 컴퓨팅 노드에서 완료된 쿼리 결과가 리더 노드로 반환됩니다. 그런 다음 리더 노드는 데이터를 병합하여 요청하는 클라이언트에게 반환합니다. 리더 노드에서 완료된 쿼리의 경우 반환 단계에서 결과가 클라이언트에게 반환됩니다.

하나의 쿼리는 여러 세그먼트로 구성되며, 각각의 세그먼트는 하나 이상의 단계로 구성됩니다. 자세한 내용은 쿼리 처리 단원을 참조하십시오.

STL_RETURN은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_RETURN에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호
step	정수	실행된 쿼리 단계입니다.
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
tasknum	정수	단계 실행을 위해 할당된 쿼리 태스크 프로세스 수.
rows	bigint	처리된 총 행 수.

열 명칭	데이터 유형	설명
bytes	bigint	단계에서 출력되는 모든 행의 크기(바이트)
packets	정수	네트워크를 통해 전송되는 패킷 총 수
checksum	bigint	이 정보는 내부 전용입니다.

샘플 쿼리

다음은 가장 최근 쿼리에서 각 조각마다 수행된 단계를 나타내는 쿼리입니다.

```
SELECT query, slice, segment, step, endtime, rows, packets
from stl_return where query = pg_last_query_id();

 query |  slice | segment | step |          endtime           | rows | packets
-------+--------+---------+------+----------------------------+------+---------
     4 |      2 |       3 |    2 | 2013-12-27 01:43:21.469043 |    3 |       0
     4 |      3 |       3 |    2 | 2013-12-27 01:43:21.473321 |    0 |       0
     4 |      0 |       3 |    2 | 2013-12-27 01:43:21.469118 |    2 |       0
     4 |      1 |       3 |    2 | 2013-12-27 01:43:21.474196 |    0 |       0
     4 |      4 |       3 |    2 | 2013-12-27 01:43:21.47704  |    2 |       0
     4 |      5 |       3 |    2 | 2013-12-27 01:43:21.478593 |    0 |       0
     4 |   12811|       4 |    1 | 2013-12-27 01:43:21.480755 |    0 |       0
(7 rows)
```

STL_S3CLIENT

전송 시간과 기타 성능 지표를 기록합니다.

STL_S3CLIENT 테이블을 사용하여 Amazon S3에서 데이터를 전송하는 데 걸린 시간을 확인합니다.

STL_S3CLIENT는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
recordtime	타임스탬프	레코드가 기록된 시간
pid	정수	프로세스 ID. 한 세션의 모든 쿼리는 동일 프로세스에서 실행되므로 동일 세션에서 일련의 쿼리를 실행하는 경우, 이 값은 항상 같은 값을 유지합니다.
http_method	character (64)	Amazon S3 요청에 해당하는 HTTP 메서드 이름.
버킷	character (64)	S3 버킷 이름
키	character (256)	Amazon S3 객체에 해당하는 키입니다.
transfer_size	bigint	전송되는 바이트 수
data_size	bigint	데이터 바이트 수. 이 값은 비압축 데이터의 transfer_size와 동일합니다. 압축을 사용한 경우에는 이 값이 비압축 데이터의 크기를 말합니다.
start_time	bigint	전송 시작 시간(2000년 1월 1일 이후 마이크로초)
end_time	bigint	전송 종료 시간(2000년 1월 1일 이후 마이크로초)
transfer_time	bigint	전송 소요 시간(마이크로초)
compression_time	bigint	전송 시간 중 데이터 압축을 푸는 데 걸린 시간(마이크로초)

열 명칭	데이터 유형	설명
connect_time	bigint	시작부터 원격 호스트 연결이 완료될 때까지 걸린 시간(마이크로초)
app_connect_time	bigint	시작부터 원격 호스트와 SSL 연결/핸드셰이크가 완료될 때까지 걸린 시간(마이크로초)
retries	bigint	전송을 다시 시도한 횟수
request_id	char(32)	Amazon S3 HTTP 응답 헤더의 요청 ID
extended_request_id	char(128)	Amazon S3 HTTP 헤더 응답의 확장 요청 ID(x-amz-id-2).
ip_address	char(64)	서버의 IP 주소(ip V4 또는 V6)
is_partial	정수	true(1)인 경우 COPY 작업 중에 입력 파일이 범위로 분할됨을 나타내는 값입니다. 이 값이 false(0)이면 입력 파일이 분할되지 않습니다.
start_offset	bigint	COPY 작업 중에 입력 파일이 분할되는 경우 분할의 오프셋 값(바이트 단위)을 나타내는 값입니다. 파일이 분할되지 않은 경우 이 값은 0입니다.

샘플 쿼리

다음은 COPY 명령을 사용하여 파일을 로드하는 데 걸린 시간을 반환하는 쿼리입니다.

```
select slice, key, transfer_time
from stl_s3client
where query = pg_last_copy_id();
```

결과

```
 slice |   key                         | transfer_time
------+-------------------------------+---------------
     0 | listing10M0003_part_00        |    16626716
     1 | listing10M0001_part_00        |    12894494
```

```
     2 | listing10M0002_part_00      |    14320978
     3 | listing10M0000_part_00      |    11293439
  3371 | prefix=listing10M;marker=   |       99395
```

다음 쿼리는 start_time 및 end_time을 타임스탬프로 변환합니다.

```
select userid,query,slice,pid,recordtime,start_time,end_time,
'2000-01-01'::timestamp + (start_time/1000000.0)* interval '1 second' as start_ts,
'2000-01-01'::timestamp + (end_time/1000000.0)* interval '1 second' as end_ts
from stl_s3client where query> -1 limit 5;
```

```
 userid | query | slice |  pid  |         recordtime         |   start_time    |
 end_time     |          start_ts          |           end_ts
--------+-------+-------+-------+----------------------------+-----------------
+-----------------+----------------------------+----------------------------
      0 |     0 |     0 | 23449 | 2019-07-14 16:27:17.207839 | 616436837154256 |
 616436837207838 | 2019-07-14 16:27:17.154256 | 2019-07-14 16:27:17.207838
      0 |     0 |     0 | 23449 | 2019-07-14 16:27:17.252521 | 616436837208208 |
 616436837252520 | 2019-07-14 16:27:17.208208 | 2019-07-14 16:27:17.25252
      0 |     0 |     0 | 23449 | 2019-07-14 16:27:17.284376 | 616436837208460 |
 616436837284374 | 2019-07-14 16:27:17.20846  | 2019-07-14 16:27:17.284374
      0 |     0 |     0 | 23449 | 2019-07-14 16:27:17.285307 | 616436837208980 |
 616436837285306 | 2019-07-14 16:27:17.20898  | 2019-07-14 16:27:17.285306
      0 |     0 |     0 | 23449 | 2019-07-14 16:27:17.353853 | 616436837302216 |
 616436837353851 | 2019-07-14 16:27:17.302216 | 2019-07-14 16:27:17.353851
```

STL_S3CLIENT_ERROR

Amazon S3에서 파일을 로드하는 동안 조각에서 발생한 오류를 기록합니다.

STL_S3CLIENT_ERROR는 COPY 명령을 실행하면서 Amazon S3에서 데이터를 전송하는 동안 발생한 오류 정보를 확인하는 데 사용됩니다.

STL_S3CLIENT_ERROR는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다. 쿼리 ID -1은 내부용입니다.
sliceid	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
recordtime	타임스탬프	레코드가 기록된 시간
pid	정수	프로세스 ID. 한 세션의 모든 쿼리는 동일 프로세스에서 실행되므로 동일 세션에서 일련의 쿼리를 실행하는 경우, 이 값은 항상 같은 값을 유지합니다.
http_method	character (64)	Amazon S3 요청에 해당하는 HTTP 메서드 이름.
버킷	character (64)	Amazon S3 버킷 이름입니다.
키	character (256)	Amazon S3 객체에 해당하는 키입니다.
오류	character (1024)	오류 메시지.
is_partial	정수	true(1)인 경우 COPY 작업 중에 입력 파일이 범위로 분할됨을 나타내는 값입니다. 이 값이 false(0)이면 입력 파일이 분할되지 않습니다.
start_offset	bigint	COPY 작업 중에 입력 파일이 분할되는 경우 분할의 오프셋 값(바이트 단위)을 나타내는 값입니다. 파일이 분할되지 않은 경우 이 값은 0입니다.

사용 노트

"Connection timed out" 오류가 다수 발생한 경우에는 네트워킹 문제를 의심해볼 수 있습니다. Enhanced VPC Routing 기능을 사용하는 경우에는 클러스터의 VPC와 데이터 리소스 사이의 네트워크 경로가 유효한지 확인하십시오. 자세한 내용은 Amazon Redshift Enhanced VPC Routing을 참조하세요.

샘플 쿼리

다음은 현재 세션에서 COPY 명령을 완료하면서 발생한 오류를 반환하는 쿼리입니다.

```
select query, sliceid, substring(key from 1 for 20) as file,
substring(error from 1 for 35) as error
from stl_s3client_error
where pid = pg_backend_pid()
order by query desc;
```

결과

```
 query  | sliceid |     file           |              error

--------+---------+--------------------+-----------------------------------
 362228 |      12 | part.tbl.25.159.gz | transfer closed with 1947655 bytes
 362228 |      24 | part.tbl.15.577.gz | transfer closed with 1881910 bytes
 362228 |       7 | part.tbl.22.600.gz | transfer closed with 700143 bytes r
 362228 |      22 | part.tbl.3.34.gz   | transfer closed with 2334528 bytes
 362228 |      11 | part.tbl.30.274.gz | transfer closed with 699031 bytes r
 362228 |      30 | part.tbl.5.509.gz  | Unknown SSL protocol error in conne
 361999 |      10 | part.tbl.23.305.gz | transfer closed with 698959 bytes r
 361999 |      19 | part.tbl.26.582.gz | transfer closed with 1881458 bytes
 361999 |       4 | part.tbl.15.629.gz | transfer closed with 2275907 bytes
 361999 |      20 | part.tbl.6.456.gz  | transfer closed with 692162 bytes r
(10 rows)
```

STL_SAVE

쿼리의 저장 단계 정보가 저장됩니다. 저장 단계에서는 입력 스트림이 임시 테이블로 저장됩니다. 임시 테이블이란 쿼리 실행 도중 중간 결과가 일시적으로 저장되는 테이블을 말합니다.

하나의 쿼리는 여러 세그먼트로 구성되며, 각각의 세그먼트는 하나 이상의 단계로 구성됩니다. 자세한 내용은 쿼리 처리 단원을 참조하십시오.

STL_SAVE는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_SAVE에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호
step	정수	실행된 쿼리 단계입니다.
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
tasknum	정수	단계 실행을 위해 할당된 쿼리 태스크 프로세스 수.
rows	bigint	처리된 총 행 수.

열 명칭	데이터 유형	설명
bytes	bigint	단계에서 출력되는 모든 행의 크기(바이트)
tbl	정수	구체화된 임시 테이블의 ID
is_diskbased	character(1)	디스크 기반 작업으로서 이번 쿼리 단계의 수행 여부: true(t) 또는 false(f).
workmem	bigint	단계에 할당되는 유효 메모리의 바이트 수

샘플 쿼리

다음은 가장 최근 쿼리에서 각 조각마다 수행된 단계를 저장하는 쿼리입니다.

```
select query, slice, segment, step, tasknum, rows,  tbl
from stl_save where query = pg_last_query_id();

 query | slice | segment | step | tasknum | rows | tbl
-------+-------+---------+------+---------+------+-----
 52236 |     3 |       0 |    2 |      21 |    0 | 239
 52236 |     2 |       0 |    2 |      20 |    0 | 239
 52236 |     2 |       2 |    2 |      20 |    0 | 239
 52236 |     3 |       2 |    2 |      21 |    0 | 239
 52236 |     1 |       0 |    2 |      21 |    0 | 239
 52236 |     0 |       0 |    2 |      20 |    0 | 239
 52236 |     0 |       2 |    2 |      20 |    0 | 239
 52236 |     1 |       2 |    2 |      21 |    0 | 239
(8 rows)
```

STL_SCAN

쿼리의 테이블 스캔 단계를 분석합니다. 스캔은 세그먼트에서 첫 번째 단계이기 때문에 이 테이블에서 행의 단계 번호는 항상 0입니다.

STL_SCAN은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_SCAN에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호
step	정수	실행된 쿼리 단계입니다.
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
tasknum	정수	단계 실행을 위해 할당된 쿼리 태스크 프로세스 수.
rows	bigint	처리된 총 행 수.
bytes	bigint	단계에서 출력되는 모든 행의 크기(바이트)
fetches	bigint	이 정보는 내부 전용입니다.

열 명칭	데이터 유형	설명
type	정수	스캔 유형의 ID. 유효한 값 목록은 아래 표를 참조하십시오.
tbl	정수	테이블 ID.
is_rrscan	character(1)	true(t)인 경우, 단계에서 범위 제한 스캔이 사용되었음을 나타냅니다.
is_delayed_scan	character(1)	이 정보는 내부 전용입니다.
rows_pre_filter	bigint	영구 테이블 스캔의 경우, 삭제 대기 행(고스트 행)을 필터링하고 사용자 정의 쿼리 필터를 적용하기 전에 내보낸 행의 총수.
rows_pre_user_filter	bigint	영구 테이블을 스캔할 때 삭제 표시된 행(고스트 행)을 필터링한 이후부터 사용자 정의 쿼리 필터를 적용하기 전까지 처리된 행의 수
perm_table_name	character(136)	영구 테이블을 스캔할 때 스캔된 테이블의 이름
is_rlf_scan	character(1)	true(t)인 경우, 단계에서 저수준 필터링이 사용되었음을 나타냅니다.
is_rlf_scan_reason	정수	이 정보는 내부 전용입니다.
num_em_blocks	정수	이 정보는 내부 전용입니다.
checksum	bigint	이 정보는 내부 전용입니다.
runtime_filtering	character(1)	true이면 실행 시간 필터가 적용되었음을 나타냅니다.
scan_region	정수	이 정보는 내부 전용입니다.

열 명칭	데이터 유형	설명
num_sortkey_as_predicate	정수	이 정보는 내부 전용입니다.
row_fetcher_state	정수	이 정보는 내부 전용입니다.
consumed_scan_ranges	bigint	이 정보는 내부 전용입니다.
work_stealing_reason	bigint	이 정보는 내부 전용입니다.
is_vectorized_scan	character(1)	이 정보는 내부 전용입니다.
is_vectorized_scan_reason	정수	이 정보는 내부 전용입니다.
row_fetcher_reason	bigint	이 정보는 내부 전용입니다.
topology_signature	bigint	이 정보는 내부 전용입니다.
use_tpm_partition	character(1)	이 정보는 내부 전용입니다.
is_rrscan_expr	character(1)	이 정보는 내부 전용입니다.

열 명칭	데이터 유형	설명
scanned_mega_value	character(1)	이 정보는 내부 전용입니다. 이 정보는 지정된 스캔 단계에서 큰 값을 스캔했는지 여부를 보여줍니다. 큰 값은 여러 블록에 저장됩니다. 블록 크기는 기본적으로 1MB이며, 큰 값은 기본 설정에서 1MB보다 큽니다.

스캔 유형

유형 ID	설명
1	네트워크의 데이터
2	압축된 공유 메모리의 영구 사용자 테이블
3	임시 행 방향 테이블
21	Amazon S3에서 파일을 로드합니다.
22	Amazon DynamoDB에서 테이블을 로드합니다.
23	원격 SSH 연결을 통한 로드 데이터
24	원격 클러스터(정렬된 리전)의 로드 데이터. 이 유형은 크기를 조정하는 데 사용됩니다.
25	원격 클러스터(정렬되지 않은 리전)의 로드 데이터. 이 유형은 크기를 조정하는 데 사용됩니다.
28	여러 테이블에서 UNION ALL을 사용하여 시계열 뷰에서 데이터 읽기.
29	Amazon S3 외부 테이블에서 데이터 읽기.
30	Amazon S3 외부 테이블의 파티션 정보 읽기.
33	원격 Postgres 테이블에서 데이터 읽기.
36	원격 MySQL 테이블에서 데이터 읽기.
37	원격 Kinesis 스트림에서 데이터 읽기.

사용 노트

rows 값은 비교적 rows_pre_filter 값에 가까울 때가 가장 이상적입니다. rows와 rows_pre_filter 사이의 차이가 크면 실행 엔진이 나중에 무시되는 행을 스캔한다는 것을 의미하지만, 이는 비효율적입니다. rows_pre_filter 값과 rows_pre_user_filter 값의 차이는 스캔 작업 시 고스트 행의 수를 의미합니다. 이때는 VACUUM을 실행하여 삭제 표시된 행을 제거하십시오. rows 값과 rows_pre_user_filter 값의 차이는 쿼리에서 필터링되는 행의 수입니다. 사용자 필터로 인해 다수의 행이 무시되는 경우에는 sort 열 선택을 살펴보고, 그렇지 않고 정렬되지 않은 영역이 너무 커서 다수의 행이 무시되는 경우에는 VACUUM을 실행하십시오.

샘플 쿼리

다음은 테이블에 삭제만 되고 아직 정리되지 않은 행(고스트 행)이 있기 때문에 rows_pre_filter 값이 rows_pre_user_filter 값보다 큰 것을 나타내는 예입니다.

```
SELECT query, slice, segment,step,rows, rows_pre_filter, rows_pre_user_filter
from stl_scan where query = pg_last_query_id();

 query |  slice | segment | step | rows  | rows_pre_filter | rows_pre_user_filter
-------+--------+---------+------+-------+-----------------+----------------------
 42915 |      0 |       0 |    0 | 43159 |           86318 |                43159
 42915 |      0 |       1 |    0 |     1 |               0 |                    0
 42915 |      1 |       0 |    0 | 43091 |           86182 |                43091
 42915 |      1 |       1 |    0 |     1 |               0 |                    0
 42915 |      2 |       0 |    0 | 42778 |           85556 |                42778
 42915 |      2 |       1 |    0 |     1 |               0 |                    0
 42915 |      3 |       0 |    0 | 43428 |           86856 |                43428
 42915 |      3 |       1 |    0 |     1 |               0 |                    0
 42915 |  10000 |       2 |    0 |     4 |               0 |                    0
(9 rows)
```

STL_SCHEMA_QUOTA_VIOLATIONS

스키마 할당량이 초과되면 발생, 타임스탬프, XID 및 기타 유용한 정보를 기록합니다.

STL_SCHEMA_QUOTA_VIOLATIONS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_SCHEMA_QUOTA_VIOLATIONS에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는

사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
ownerid	정수	스키마 소유자의 ID입니다.
xid	bigint	문과 연결된 트랜잭션 ID
pid	정수	문과 연결된 프로세스 ID
userid	정수	항목을 생성한 사용자의 ID입니다.
schema_id	정수	네임스페이스 또는 스키마 ID입니다.
schema_name	character (128)	네임스페이스 또는 스키마 이름입니다.
할당량	정수	스키마에서 사용할 수 있는 디스크 공간(MB)입니다.
disk_usage	정수	스키마에서 현재 사용 중인 디스크 공간(MB)입니다.
disk_usage_pct	double precision	구성된 할당량 중 스키마에서 현재 사용하고 있는 디스크 공간 백분율입니다.
타임스탬프	시간대 미포함 TIMESTAMP	위반이 발생한 시간입니다.

샘플 쿼리

다음은 할당량 위반의 결과를 보여 주는 쿼리입니다.

```
SELECT userid, TRIM(SCHEMA_NAME) "schema_name", quota, disk_usage, disk_usage_pct,
 timestamp FROM
stl_schema_quota_violations WHERE SCHEMA_NAME = 'sales_schema' ORDER BY timestamp DESC;
```

이 쿼리는 지정된 스키마에 대해 다음 샘플 출력을 반환합니다.

```
userid | schema_name  | quota | disk_usage | disk_usage_pct |timestamp
-------+--------------+-------+------------+----------------
+----------------------------
104    | sales_schema | 2048  | 2798       |  136.62        | 2020-04-20
 20:09:25.494723
(1 row)
```

STL_SESSIONS

사용자 세션 이력에 대한 정보를 반환합니다.

STL_SESSIONS에는 세션 이력만 저장되는 반면 STV_SESSIONS에는 현재 활성 세션이 저장된다는 점에서 STL_SESSIONS와 STV_SESSIONS는 서로 다릅니다.

STL_SESSIONS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_SESSION_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
starttime	타임스탬프	세션이 시작된 UTC 시간
endtime	타임스탬프	세션이 종료된 UTC 시간
포함	정수	세션의 프로세스 ID.
user_name	character(50)	세션과 연결된 사용자 이름
db_name	character(50)	세션과 연결된 데이터베이스 이름

열 명칭	데이터 유형	설명
timeout_sec	int	세션이 시간 초과되기 전에 비활성 또는 유휴 상태로 유지되는 최대 시간(초)입니다. 0은 시간 제한이 설정되지 않았음을 나타냅니다.
timed_out	int	연결이 종료된 이유를 나타내는 값입니다. 다음과 같은 값을 가질 수 있습니다. • 0: 알 수 없는 오류로 인해 연결이 종료되었습니다. • 1: 연결 시간이 초과되었습니다. • 2: 클라이언트 측에서 연결을 종료했습니다. • 3: Amazon Redshift 백엔드 내부 오류로 인해 연결이 종료되었습니다.

샘플 쿼리

TICKIT 데이터베이스의 세션 이력을 보려면 다음과 같이 쿼리를 입력합니다.

```
select starttime, process, user_name, timeout_sec, timed_out
from stl_sessions
where db_name='tickit' order by starttime;
```

위 쿼리는 다음과 같은 샘플 출력을 반환합니다.

```
    starttime              | process |  user_name             | timeout_sec | timed_out
---------------------------+---------+------------------------+-------------
+-------------
2008-09-15 09:54:06.746705 |   32358 | dwuser                 | 120         | 1
2008-09-15 09:56:34.30275  |   32744 | dwuser                 | 60          | 1
2008-09-15 11:20:34.694837 |   14906 | dwuser                 | 0           | 0
2008-09-15 11:22:16.749818 |   15148 | dwuser                 | 0           | 0
2008-09-15 14:32:44.66112  |   14031 | dwuser                 | 0           | 0
2008-09-15 14:56:30.22161  |   18380 | dwuser                 | 0           | 0
2008-09-15 15:28:32.509354 |   24344 | dwuser                 | 0           | 0
2008-09-15 16:01:00.557326 |   30153 | dwuser                 | 120         | 1
2008-09-15 17:28:21.419858 |   12805 | dwuser                 | 0           | 0
```

```
2008-09-15 20:58:37.601937 |   14951 | dwuser                | 60          | 1
2008-09-16 11:12:30.960564 |   27437 | dwuser                | 60          | 1
2008-09-16 14:11:37.639092 |   23790 | dwuser                | 3600        | 1
2008-09-16 15:13:46.02195  |    1355 | dwuser                | 120         | 1
2008-09-16 15:22:36.515106 |    2878 | dwuser                | 120         | 1
2008-09-16 15:44:39.194579 |    6470 | dwuser                | 120         | 1
2008-09-16 16:50:27.02138  |   17254 | dwuser                | 120         | 1
2008-09-17 12:05:02.157208 |    8439 | dwuser                | 3600        | 0
(17 rows)
```

STL_SORT

ORDER BY 처리를 사용하는 단계와 같은 쿼리의 정렬 실행 단계를 표시합니다.

STL_SORT는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_SORT에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호

열 명칭	데이터 유형	설명
step	정수	실행된 쿼리 단계입니다.
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .
tasknum	정수	단계 실행을 위해 할당된 쿼리 태스크 프로세스 수.
rows	bigint	처리된 총 행 수.
bytes	bigint	단계에서 출력되는 모든 행의 크기(바이트)
tbl	정수	테이블 ID.
is_diskbased	character(1)	true(t)인 경우 쿼리가 디스크 기반 작업으로 수행된 것을 의미합니다. false(f)인 경우 쿼리가 메모리에서 수행된 것을 의미합니다.
workmem	bigint	단계에 할당된 유효 메모리 바이트의 총 수
checksum	bigint	이 정보는 내부 전용입니다.

샘플 쿼리

다음은 조각 0과 세그먼트 1일 때 정렬 결과를 반환하는 예입니다.

```
select query, bytes, tbl, is_diskbased, workmem
from stl_sort
where slice=0 and segment=1;
```

```
 query |  bytes  | tbl | is_diskbased |  workmem
-------+---------+-----+--------------+-----------
   567 | 3126968 | 241 | f            | 383385600
   604 |    5292 | 242 | f            | 383385600
```

```
   675 |  104776 | 251 | f           | 383385600
   525 | 3126968 | 251 | f           | 383385600
   585 |    5068 | 241 | f           | 383385600
   630 |  204808 | 266 | f           | 383385600
   704 |       0 | 242 | f           |         0
   669 | 4606416 | 241 | f           | 383385600
   696 |  104776 | 241 | f           | 383385600
   651 | 4606416 | 254 | f           | 383385600
   632 |       0 | 256 | f           |         0
   599 |     396 | 241 | f           | 383385600
 86397 |       0 | 242 | f           |         0
   621 |    5292 | 241 | f           | 383385600
 86325 |       0 | 242 | f           |         0
   572 |    5068 | 242 | f           | 383385600
   645 |  204808 | 241 | f           | 383385600
   590 |     396 | 242 | f           | 383385600
(18 rows)
```

STL_SSHCLIENT_ERROR

SSH 클라이언트에 표시되는 모든 오류를 기록합니다.

STL_SSHCLIENT_ERROR는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
recordtime	타임스탬프	오류 기록 시간
pid	정수	오류 기록 프로세스

열 명칭	데이터 유형	설명
ssh_username	character (1024)	SSH 사용자 이름
엔드포인트	character (1024)	SSH 엔드포인트
명령	character (4096)	전체 SSH 명령
오류	character (1024)	오류 메시지입니다.

STL_STREAM_SEGS

스트림과 동시 세그먼트 사이의 관계를 나열합니다.

이 맥락에서 스트림이란 Amazon Redshift 스트림입니다. 이 시스템 뷰는 구체화된 뷰로 스트리밍 모으기과 관련이 없습니다.

STL_STREAM_SEGS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_STREAM_SEGS에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
스트림	정수	쿼리의 동시 세그먼트 집합
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호

샘플 쿼리

가장 최근 쿼리에서 스트림과 동시 세그먼트 사이의 관계를 보려면 다음과 같이 쿼리를 입력합니다.

```
select *
from stl_stream_segs
where query = pg_last_query_id();

 query | stream | segment
-------+--------+---------
    10 |      1 |       2
    10 |      0 |       0
    10 |      2 |       4
    10 |      1 |       3
    10 |      0 |       1
(5 rows)
```

STL_TR_CONFLICT

데이터베이스 테이블과 트랜잭션 충돌 문제를 식별하여 해결할 수 있는 정보를 표시합니다.

트랜잭션 충돌은 두 명 이상의 사용자가 트랜잭션을 직렬화할 수 없는 테이블에서 데이터 행을 쿼리하고 수정하는 경우에 발생합니다. 직렬화 가능성을 중단하는 문을 실행하는 트랜잭션은 중지되고 롤백됩니다. 트랜잭션 충돌이 발생할 때마다 Amazon Redshift는 취소된 트랜잭션에 대한 세부 정보가 포함된 STL_TR_CONFLICT 시스템 테이블에 데이터 행을 작성합니다. 자세한 내용은 직렬화 가능 격리 단원을 참조하십시오.

STL_TR_CONFLICT는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_TRANSACTION_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
xact_id	bigint	롤백된 트랜잭션의 트랜잭션 ID
process_id	bigint	롤백된 트랜잭션과 연결된 프로세스입니다.
xact_start_ts	타임스탬프	트랜잭션이 시작된 타임스탬프(UTC)입니다.
abort_time	타임스탬프	트랜잭션이 중지된 타임스탬프(UTC)입니다.
table_id	bigint	충돌이 발생한 테이블의 테이블 ID

샘플 쿼리

특정 테이블이 관련된 충돌 정보를 반환하려면 다음과 같이 테이블 ID를 지정하여 쿼리를 실행합니다.

```
select * from stl_tr_conflict where table_id=100234
order by xact_start_ts;

xact_id|process_|      xact_start_ts       |        abort_time        |table_
       |id      |                          |                          |id
-------+--------+--------------------------+--------------------------+------
  1876 |  8551  |2010-03-30 09:19:15.852326|2010-03-30 09:20:17.582499|100234
  1928 | 15034  |2010-03-30 13:20:00.636045|2010-03-30 13:20:47.766817|100234
  1991 | 23753  |2010-04-01 13:05:01.220059|2010-04-01 13:06:06.94098 |100234
  2002 | 23679  |2010-04-01 13:17:05.173473|2010-04-01 13:18:27.898655|100234
(4 rows)
```

테이블 ID는 직렬성 위반 오류 메시지(error 1023)의 DETAIL 영역에서 가져올 수 있습니다.

STL_UNDONE

실행 취소된 트랜잭션에 대한 정보를 표시합니다.

STL_UNDONE은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_TRANSACTION_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
xact_id	bigint	실행 취소 트랜잭션의 ID
xact_id_undone	bigint	실행 취소된 트랜잭션 ID
undo_start_ts	타임스탬프	실행 취소 트랜잭션의 시작 시간
undo_end_ts	타임스탬프	실행 취소 트랜잭션의 종료 시간
table_id	bigint	실행 취소 트랜잭션의 영향을 받은 테이블의 ID

샘플 쿼리

모든 실행 취소 트랜잭션에 대한 간략한 로그를 보려면 다음과 같이 명령을 입력합니다.

```
select xact_id, xact_id_undone, table_id from stl_undone;
```

위의 명령은 다음과 같은 샘플 출력을 반환합니다.

```
 xact_id | xact_id_undone | table_id
```

```
---------+----------------+----------
1344 |           1344 |   100192
1326 |           1326 |   100192
1551 |           1551 |   100192
(3 rows)
```

STL_UNIQUE

SELECT 목록에서 DISTINCT 함수를 사용할 때, 혹은 UNION 또는 INTERSECT 쿼리에서 중복을 제거할 때 발생하는 실행 단계를 분석합니다.

STL_UNIQUE는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_UNIQUE에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호
step	정수	실행된 쿼리 단계입니다.

열 명칭	데이터 유형	설명
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **`2009-06-12 11:29:19.131358`** .
tasknum	정수	단계 실행을 위해 할당된 쿼리 태스크 프로세스 수.
rows	bigint	처리된 총 행 수.
`type`	character(6)	단계 유형. 유효한 값은 다음과 같습니다. • HASHED. 단계가 정렬 없이 분류된 집계를 사용한 것을 의미합니다. • PLAIN. 단계가 분류되지 않은 스칼라 집계를 사용한 것을 의미합니다. • SORTED. 단계가 정렬과 함께 분류된 집계를 사용한 것을 의미합니다.
is_diskbased	character(1)	true(t)인 경우 쿼리가 디스크 기반 작업으로 수행된 것을 의미합니다. false(f)인 경우 쿼리가 메모리에서 수행된 것을 의미합니다.
slots	정수	해시 버킷의 총 수
workmem	bigint	단계에 할당된 유효 메모리 바이트의 총 수
max_buffers_used	bigint	디스크 이동 전에 해시 테이블에서 사용한 버퍼의 최대 수
크기 조정	정수	이 정보는 내부 전용입니다.
occupied	정수	이 정보는 내부 전용입니다.
flushable	정수	이 정보는 내부 전용입니다.

열 명칭	데이터 유형	설명
used_unique_prefetching	character(1)	이 정보는 내부 전용입니다.
bytes	biginit	해당 단계에 대한 모든 출력 행의 바이트 수입니다.

샘플 쿼리

다음 쿼리를 실행한다고 가정합니다.

```
select distinct eventname
from event order by 1;
```

위 쿼리의 ID가 6313이라고 했을 때 다음은 세그먼트 0과 1의 각 조각마다 고유성 단계에서 산출되는 행의 수를 나타내는 예입니다.

```
select query, slice, segment, step, datediff(msec, starttime, endtime) as msec,
 tasknum, rows
from stl_unique where query = 6313
order by query desc, slice, segment, step;
```

```
 query | slice | segment | step | msec | tasknum | rows
-------+-------+---------+------+------+---------+------
  6313 |     0 |       0 |    2 |    0 |      22 |  550
  6313 |     0 |       1 |    1 |  256 |      20 |  145
  6313 |     1 |       0 |    2 |    1 |      23 |  540
  6313 |     1 |       1 |    1 |   42 |      21 |  127
  6313 |     2 |       0 |    2 |    1 |      22 |  540
  6313 |     2 |       1 |    1 |  255 |      20 |  158
  6313 |     3 |       0 |    2 |    1 |      23 |  542
  6313 |     3 |       1 |    1 |   38 |      21 |  146
(8 rows)
```

STL_UNLOAD_LOG

언로드 작업의 세부 정보를 기록합니다.

STL_UNLOAD_LOG는 UNLOAD 문에서 생성되는 각 파일마다 행 1개를 기록합니다. 예를 들어 UNLOAD를 실행하여 파일 12개가 생성된다면 STL_UNLOAD_LOG에 포함되는 행의 수도 12개입니다.

STL_UNLOAD_LOG는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_UNLOAD_LOG에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_UNLOAD_HISTORY 및 SYS_UNLOAD_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID입니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
pid	정수	쿼리 문과 연결된 프로세스 ID
경로	character(1280)	파일의 전체 Amazon S3 객체 경로.
start_time	타임스탬프	트랜잭션 시작 시간
end_time	타임스탬프	트랜잭션 종료 시간
line_count	bigint	파일로 언로드되는 라인(행)의 수
transfer_size	bigint	전송되는 바이트 수

열 명칭	데이터 유형	설명
file_format	character(10)	언로드된 파일의 형식입니다.

샘플 쿼리

UNLOAD 명령으로 Amazon S3에 기록된 파일 목록을 가져오려면 UNLOAD가 완료된 후 Amazon S3 목록 작업을 호출하면 됩니다. STL_UNLOAD_LOG를 쿼리할 수도 있습니다.

다음은 마지막으로 완료된 쿼리에 대해 UNLOAD로 생성된 파일의 경로 이름을 반환하는 쿼리입니다.

```
select query, substring(path,0,40) as path
from stl_unload_log
where query = pg_last_query_id()
order by path;
```

위의 명령은 다음과 같은 샘플 출력을 반환합니다.

```
 query |             path
-------+--------------------------------------
  2320 | s3://amzn-s3-demo-bucket/venue0000_part_00
  2320 | s3://amzn-s3-demo-bucket/venue0001_part_00
  2320 | s3://amzn-s3-demo-bucket/venue0002_part_00
  2320 | s3://amzn-s3-demo-bucket/venue0003_part_00
(4 rows)
```

STL_USAGE_CONTROL

STL_USAGE_CONTROL 뷰에는 사용 한도에 도달할 때 기록되는 정보가 들어 있습니다. 사용 제한에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 사용 제한 관리 섹션을 참조하세요.

STL_USAGE_CONTROL은 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
eventtime	타임스탬프	쿼리가 사용 한도를 초과한 시간(UTC)입니다.

열 명칭	데이터 유형	설명
쿼리	정수	쿼리 식별자입니다. 이 ID를 사용하여 다양한 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
xid	bigint	트랜잭션 식별자입니다.
pid	정수	쿼리와 연결된 프로세스 식별자입니다.
usage_limit_id	character(40)	Amazon Redshift에서 생성된 UUID(Universally Unique Identifier)입니다(예: `25d9297e-3e7b-41c8-9f4d-c4b6eb731c09`).
feature_type	character(30)	사용 한도를 초과한 기능입니다. 가능한 값은 `CONCURRENCY_SCALING` 및 `SPECTRUM`입니다.

샘플 쿼리

다음 SQL 예제에서는 사용 한도에 도달할 때 기록되는 정보 중 일부를 반환합니다.

```
select query, pid, eventtime, feature_type
from stl_usage_control
order by eventtime desc
limit 5;
```

STL_USERLOG

다음과 같이 데이터베이스 사용자의 변경 사항에 대한 세부 정보를 기록합니다.

- 사용자 생성
- 사용자 삭제
- 사용자 변경(이름 변경)
- 사용자 변경(속성 변경)

STL_USERLOG는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_USERLOG에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	변경 사항이 적용되는 사용자 ID
사용자 이름	character(50)	변경 사항이 적용되는 사용자의 이름
oldusername	character(50)	이름 변경 작업의 경우 변경 전 사용자 이름. 그 외 다른 작업의 경우 이 필드는 비어 있습니다.
작업	character(10)	실행 작업. 유효한 값: • Alter • 생성 • Drop • 이름 바꾸기
usecreatedb	정수	true(1)인 경우 사용자에게 데이터베이스 생성 권한이 있다는 것을 의미합니다.
usesuper	정수	true(1)인 경우 사용자가 수퍼유저임을 의미합니다.
usecatupd	정수	true(1)인 경우 사용자가 시스템 카탈로그를 업데이트할 수 있다는 것을 의미합니다.
valuntil	타임스탬프	암호 만료 날짜
pid	정수	프로세스 ID
xid	bigint	트랜잭션 ID.
recordtime	타임스탬프	쿼리 시작 시간(UTC)

샘플 쿼리

다음은 사용자 작업 4개를 실행한 후 STL_USERLOG 뷰에 대한 쿼리를 실행하는 예입니다.

```
create user userlog1 password 'Userlog1';
alter user userlog1 createdb createuser;
alter user userlog1 rename  to userlog2;
drop user userlog2;

select userid, username, oldusername, action, usecreatedb, usesuper from stl_userlog
 order by recordtime desc;
```

```
 userid |  username | oldusername |  action | usecreatedb | usesuper
--------+-----------+-------------+---------+-------------+----------
    108 | userlog2  |             | drop    |           1 |   1
    108 | userlog2  | userlog1    | rename  |           1 |   1
    108 | userlog1  |             | alter   |           1 |   1
    108 | userlog1  |             | create  |           0 |   0
 (4 rows)
```

STL_UTILITYTEXT

SELECT를 제외하고 데이터베이스에서 실행된 SQL 명령 텍스트를 수집합니다.

STL_UTILITYTEXT 뷰에 대한 쿼리를 실행하여 다음과 같이 시스템에서 실행된 SQL 문의 하위 집합을 수집합니다.

- ABORT, BEGIN, COMMIT, END, ROLLBACK
- ANALYZE
- CALL
- CANCEL
- COMMENT
- CREATE, ALTER, DROP DATABASE
- CREATE, ALTER, DROP USER

- EXPLAIN
- GRANT, REVOKE
- LOCK
- reset
- SET
- SET
- TRUNCATE

STL_DDLTEXTSTL_QUERYTEXT 및 도 참조하십시오.SVL_STATEMENTTEXT

STARTTIME 열과 ENDTIME 열은 일정 시간 동안 기록된 문을 확인하는 데 사용됩니다. 긴 SQL 텍스트 블록은 200개 문자의 길이로 구분되며, SEQUENCE 열에서 단일 문에 속하는 텍스트 조각을 식별할 수 있습니다.

STL_UTILITYTEXT는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
xid	bigint	트랜잭션 ID.
pid	정수	쿼리 문과 연결된 프로세스 ID
레이블	character(320)	쿼리 실행에 사용되는 파일의 이름 또는 SET QUERY_GROUP 명령을 사용하여 정의되는 레이블. 쿼리가 파일 기반이 아니거나 QUERY_GROUP 파라미터가 설정되지 않은 경우, 이 필드의 값은 공백입니다.

열 명칭	데이터 유형	설명
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .
SEQUENCE	정수	단일 문에 200자 이상이 포함된 경우, 해당 문에 대해 추가 행이 기록됩니다. 시퀀스 0이 첫 번째 행이고 1이 두 번째 행이 되는 방식입니다.
텍스트	character(200)	200자씩 증가하는 SQL 텍스트. 이 필드에는 백슬래시(\\) 및 줄 바꿈(\n) 등의 특수 문자가 포함될 수 있습니다.

샘플 쿼리

다음은 2012년 1월 26일에 실행된 "utility" 명령 텍스트를 반환하는 쿼리입니다. 여기에서는 SET 명령 몇 개와 SHOW ALL 명령 1개가 실행되었습니다.

```
select starttime, sequence, rtrim(text)
from stl_utilitytext
where starttime like '2012-01-26%'
order by starttime, sequence;

starttime          | sequence |              rtrim
---------------------------+-----+---------------------------------
2012-01-26 13:05:52.529235 |   0 | show all;
2012-01-26 13:20:31.660255 |   0 | SET query_group to ''
2012-01-26 13:20:54.956131 |   0 | SET query_group to 'soldunsold.sql'
...
```

저장된 SQL 재구성

STL_UTILITYTEXT의 `text` 열에 저장된 SQL을 재구성하려면 SELECT 문을 실행하여 `text` 열의 1개 이상의 부분에서 SQL을 생성합니다. 재구성된 SQL을 실행하기 전에 모든 (\n) 특수 문자를 줄 바꿈으로 바꿉니다. 다음 SELECT 문의 결과는 `query_statement` 필드에서 재구성된 SQL의 행입니다.

```
SELECT LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END) WITHIN
 GROUP (ORDER BY sequence) as query_statement
FROM stl_utilitytext GROUP BY xid order by xid;
```

예를 들어 다음 쿼리는 query_group을 제로 문자열로 설정합니다. 쿼리 자체는 200자 이상이며 STL_UTILITYTEXT에 여러 부분으로 저장됩니다.

```
set query_group to
 '0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
             000000';
```

이 예제에서 이 쿼리는 STL_UTILITYTEXT의 `text` 열에 많은 두 부분(행)으로 저장됩니다.

```
select query, sequence, text
from stl_utilitytext where query=pg_last_query_id() order by query desc, sequence limit
 10;
```

```
         starttime          | sequence |
                                                text

----------------------------+----------
+-----------------------------------------------------------------------------------------------------
 2019-07-23 22:55:34.926198 |        0 | set query_group to
 '0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000\n00000000000
 2019-07-23 22:55:34.926198 |        1 |                   000000';
```

STL_UTILITYTEXT에 저장된 SQL을 재구성하려면 다음 SQL을 실행합니다.

```
select LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END, '')
 within group (order by sequence) AS query_statement
from stl_utilitytext where query=pg_last_query_id();
```

클라이언트에서 재구성된 SQL을 사용하려면 모든 (\n) 특수 문자를 줄 바꿈으로 바꿉니다.

```
                                                query_statement
```

```
----------------------------------------------------------------------------------------------------
 set query_group to
 '0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000\n000000000000
                  000000';
```

STL_VACUUM

정리된 테이블의 행 및 블록 통계를 표시합니다.

이 뷰는 정리 작업이 시작하고 끝난 시점에서 각각 특정 정보를 나열하며, 작업 실행 시 이점에 대한 설명도 나타냅니다. 이 명령 실행에 필요한 요건에 대한 자세한 내용은 VACUUM 명령 설명을 참조하십시오.

STL_VACUUM은 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_VACUUM_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID입니다.
xid	bigint	VACUUM 문의 트랜잭션 ID. 이 테이블을 STL_QUERY 뷰로 조인하면 임의의 VACUUM 트랜잭션에서 실행된 SQL 문을 개별적으로 확인할 수 있습니다. 전체 데이터베이스를 정리하는 경우에는 각 테이블이 별도의 트랜잭션으로 정리됩니다.
table_id	정수	테이블 ID입니다.
status	character(30)	각 테이블의 VACUUM 작업 상태. 가능한 값은 다음과 같습니다. • **Started**

열 명칭	데이터 유형	설명
		• **Started Delete Only** • **Started Delete Only (Sorted >= nn%)** VACUUM FULL에서 삭제 단계만 시작되었습니다. 테이블이 이미 정렬 임계값 이상으로 정렬되어 있기 때문에 정렬 단계를 건너뛰었습니다. • **Started Sort Only** • **Started Ranged Partition** • **Started Reindex** • **Finished** 테이블에서 작업이 완료된 시간. 특정 테이블에서 정리 작업이 걸린 시간을 확인하려면 해당 트랜잭션 ID와 테이블 ID를 대상으로 Started 시간을 Finished 시간에서 빼서 계산합니다. • **Skipped** 테이블이 완전히 정렬되어 삭제 표시된 행이 없기 때문에 테이블을 건너뛰었습니다. • **Skipped (delete only)** DELETE ONLY가 지정되었고, 삭제 표시된 행이 없기 때문에 테이블을 건너뛰었습니다. • **Skipped (sort only)** SORT ONLY가 지정되었고, 테이블이 이미 완전히 정렬되었기 때문에 테이블을 건너뛰었습니다. • **Skipped (sort only, sorted>=xx%)** SORT ONLY가 지정되었고, 테이블이 이미 정렬 임계값 이상으로 정렬되었기 때문에 테이블을 건너뛰었습니다. • **Skipped (0 rows)** 테이블이 비어 있어서 건너뛰었습니다. • **VacuumBG**

열 명칭	데이터 유형	설명
		자동 vacuum 작업이 백그라운드에서 수행되었습니다. 이 상태는 자동으로 수행될 때 다른 상태 앞에 추가됩니다. 예를 들어, 자동으로 수행되는 삭제 전용 vacuum에는 `[VacuumBG] Started Delete Only` 상태가 표시된 시작 행이 있습니다. VACUUM 정렬 임계값 설정에 대한 자세한 내용은 VACUUM 섹션을 참조하세요.
rows	bigint	테이블의 실제 행 수 + 삭제되었지만 아직 디스크에 저장되어 있는(정리 대기 중인) 모든 행. 이 열에는 **`Started`** 상태인 행에 대해 정리 작업이 시작되기 전 행의 수와 **`Finished`** 상태인 행에 대해 정리 작업을 마친 후 행의 수가 표시됩니다.
sortedrows	정수	테이블에서 정렬된 행의 수. 이 열에는 Status 열이 **`Started`** 상태인 행에 대해 정리 작업이 시작되기 전 행의 수와 Status 열이 **`Finished`** 상태인 행에 대해 정리 작업을 마친 후 행의 수가 표시됩니다.
블록	정수	정리 작업 이전(**`Started`** 상태의 행)과 정리 작업 이후(**`Finished`** 열) 테이블 데이터를 저장하는 데 사용된 데이터 블록의 총 수. 각 데이터 블록이 1MB씩 사용합니다.
max_merge_partitions	정수	이 열은 성능 분석에 사용되며 병합 단계 반복당 테이블에 대해 vacuum이 처리할 수 있는 최대 파티션 수를 나타냅니다. (Vacuum은 정렬되지 않은 리전을 하나 이상의 정렬된 파티션으로 정렬합니다. 테이블의 열 수와 현재 Amazon Redshift 구성에 따라 병합 단계는 단일 병합 반복에서 최대 파티션 수를 처리할 수 있습니다. 병합 단계는 정렬된 파티션 수가 최대 병합 파티션 수를 초과해도 작동하지만 더 많은 병합 반복이 필요합니다.)
eventtime	타임스탬프	정리 작업의 시작 또는 종료 시점

열 명칭	데이터 유형	설명
reclaimable_rows	bigint	현재 cutoff_xid에 대해 회수 가능한 행 수입니다. 이 열에는 상태가 **Started**인 행에 대한 vacuum이 시작되기 전 Redshift에서 예상한 회수 가능한 행 수와, 상태가 **Finished**인 행에 대한 vacuum 이후 남은 회수 가능한 행의 실제 수가 표시됩니다.
reclaimable_space_mb	bigint	현재 cutoff_xid의 회수 가능한 공간(MB)입니다. 이 열에는 상태가 **Started**인 행에 대한 vacuum이 시작되기 전 Redshift에서 예상한 회수 가능한 공간과, 상태가 **Finished**인 행에 대한 vacuum 이후 남은 회수 가능한 공간의 실제 양이 표시됩니다.
cutoff_xid	bigint	VACUUM 작업의 컷오프 트랜잭션 ID입니다. 컷오프 이후의 모든 트랜잭션은 VACUUM 작업에 포함되지 않습니다.
is_recluster	정수	1(true)이면 VACUUUM 작업이 리클러스터 알고리즘을 실행했다는 뜻이고, 0(false)이면 실행되지 않았다는 뜻입니다.

샘플 쿼리

다음은 테이블 108313의 정리 통계를 보고하는 쿼리입니다. 이 테이블은 몇 차례 삽입 및 삭제 작업 이후에 정리되었습니다.

```
select xid, table_id, status, rows, sortedrows, blocks, eventtime,
                reclaimable_rows, reclaimable_space_mb
from stl_vacuum where table_id=108313 order by eventtime;

 xid   | table_id | status                  | rows | sortedrows | blocks |  eventtime
         | reclaimable_rows | reclaimable_space_mb
-------+----------+-------------------------+------+------------+--------
+----------------------+------------------+----------------------
 14294 |   108313 | Started                 | 1950 |        408 |     28 |  2016-05-19
 17:36:01 |              984 |                   17
 14294 |   108313 | Finished                |  966 |        966 |     11 |  2016-05-19
 18:26:13 |                0 |                    0
```

```
 15126 |   108313 | Skipped(sorted>=95%)    |  966 |        966 |     11 |  2016-05-19
 18:26:38 |                0 |                     0
```

VACUUM 작업을 시작했을 때 테이블에 포함된 행의 수는 1,950개였고, 이 행들은 1MB 블록 28개에 저장되어 있었습니다. Amazon Redshift는 vacuum 작업을 통해 984개 또는 17개 블록의 디스크 공간을 회수할 수 있다고 추정했습니다.

완료 상태 행의 경우 ROWS 열에는 966 값이 표시되고, BLOCKS 열 값은 28에서 11로 감소합니다. vacuum은 예상 디스크 공간을 재확보했으며, vacuum 작업이 완료된 후 재확보 가능한 행이나 공간이 남지 않았습니다.

정렬 단계(트랜잭션 15126)에서는 행이 정렬 키 순서에 따라 삽입되었기 때문에 정리 작업이 테이블을 건너뛸 수 있었습니다.

다음은 다수 행의 INSERT 작업 이후 SALES 테이블(이번 예의 테이블 110116)에 대한 SORT ONLY 정리 통계를 나타낸 예입니다.

```
vacuum sort only sales;

select xid, table_id, status, rows, sortedrows, blocks, eventtime
from stl_vacuum order by xid, table_id, eventtime;

xid |table_id|     status      | rows  |sortedrows|blocks|     eventtime
----+--------+-----------------+-------+----------+------+--------------------
...
2925| 110116 |Started Sort Only|1379648|   172456 |  132 | 2011-02-24 16:25:21...
2925| 110116 |Finished         |1379648|  1379648 |  132 | 2011-02-24 16:26:28...
```

STL_WINDOW

윈도 함수를 수행하는 쿼리 단계를 분석합니다.

STL_WINDOW는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STL_WINDOW에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한

설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
slice	정수	쿼리가 실행 중인 슬라이스를 식별하는 번호.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호
step	정수	실행된 쿼리 단계입니다.
starttime	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .
endtime	타임스탬프	쿼리가 완료된 시간(UTC)입니다. 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .
tasknum	정수	단계 실행을 위해 할당된 쿼리 태스크 프로세스 수.
rows	bigint	처리된 총 행 수.
is_diskbased	character(1)	true(t)인 경우 쿼리가 디스크 기반 작업으로 수행된 것을 의미합니다. false(f)인 경우 쿼리가 메모리에서 수행된 것을 의미합니다.
workmem	bigint	단계에 할당된 유효 메모리 바이트의 총 수

샘플 쿼리

다음은 조각 0과 세그먼트 3일 때 창 함수 결과를 반환하는 예입니다.

```
select query, tasknum, rows, is_diskbased, workmem
from stl_window
where slice=0 and segment=3;
```

```
 query | tasknum | rows | is_diskbased | workmem
-------+---------+------+--------------+----------
 86326 |      36 | 1857 | f            | 95256616
   705 |      15 | 1857 | f            | 95256616
 86399 |      27 | 1857 | f            | 95256616
   649 |      10 |    0 | f            | 95256616
(4 rows)
```

STL_WLM_ERROR

WLM과 관련하여 발생하는 모든 오류를 기록합니다.

STL_WLM_ERROR는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
recordtime	타임스탬프	오류 발생 시간
pid	정수	오류가 발생한 프로세스의 ID
error_string	character(256)	오류 설명

STL_WLM_RULE_ACTION

사용자 정의 대기열과 연결된 WLM 쿼리 모니터링 규칙에서 발생하는 작업 세부 정보를 기록합니다. 자세한 내용은 WLM 쿼리 모니터링 규칙 단원을 참조하십시오.

STL_WLM_RULE_ACTION은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	쿼리를 실행한 사용자
쿼리	정수	쿼리 ID.
service_class	정수	서비스 클래스의 ID. 쿼리 대기열은 WLM 구성에서 정의합니다. 5보다 큰 서비스 클래스는 사용자 정의 대기열입니다.
규칙	character(256)	쿼리 모니터링 규칙 이름
작업	character(256)	결과적 작업. 가능한 값은 다음과 같습니다. • log • 건너뛰기(다시 할당) • 건너뛰기(다시 시작) • 중단 • change_query_priority • 없음 none 값은 규칙의 조건자가 충족되었지만 다른 규칙에 따라 작업이 심각도가 더욱 높은 작업으로 대체되었음을 의미합니다.
recordtime	타임스탬프	작업이 기록된 시간
action_value	character(256)	action이 change_query_priority 인 경우 가능한 값은 highest, high, normal, low 및 lowest입니다. action이 log, hop 또는 abort인 경우 값은 비어 있습니다.
service_class_name	character(64)	서비스 클래스의 이름입니다.

샘플 쿼리

다음은 쿼리 모니터링 규칙에 따라 중지된 쿼리를 확인하는 예입니다.

```
Select query, rule
from stl_wlm_rule_action
where action = 'abort'
order by query;
```

STL_WLM_QUERY

WLM에서 처리하는 서비스 클래스의 쿼리를 실행하려고 시도할 때마다 레코드가 저장됩니다.

STL_WLM_QUERY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
xid	정수	쿼리 또는 하위 쿼리의 트랜잭션 ID
작업	정수	워크로드 관리자를 통해 쿼리를 추적하는 데 사용되는 ID. 다수의 쿼리 ID와 연결되기도 합니다. 쿼리를 다시 시작하면 새로운 작업 ID가 아닌 새로운 쿼리 ID가 할당됩니다.
쿼리	정수	쿼리 ID. 쿼리를 다시 시작하면 새로운 작업 ID가 아닌 새로운 쿼리 ID가 할당됩니다.
service_class	정수	서비스 클래스의 ID. 서비스 클래스 ID의 목록은 WLM 서비스 클래스 ID 섹션을 참조하세요.

열 명칭	데이터 유형	설명
slot_count	정수	대기열에 설정된 동시성 수준에 따라 쿼리가 사용하는 WLM 쿼리 슬롯의 수입니다. 기본값은 1. 자세한 내용은 wlm_query_slot_count를 참조하세요.
service_class_start_time	타임스탬프	쿼리가 서비스 클래스에 할당된 시간 이 시간은 UTC 표준 시간대 기준입니다.
queue_start_time	타임스탬프	쿼리가 서비스 클래스 대기열에 진입한 시간 이 시간은 UTC 표준 시간대 기준입니다.
queue_end_time	타임스탬프	쿼리가 서비스 클래스 대기열에서 나간 시간 이 시간은 UTC 표준 시간대 기준입니다.
total_queue_time	bigint	쿼리가 대기열에서 대기한 총 시간(마이크로초)
exec_start_time	타임스탬프	쿼리 실행이 서비스 클래스에서 시작된 시간 이 시간은 UTC 표준 시간대 기준입니다.
exec_end_time	타임스탬프	쿼리 실행이 서비스 클래스에서 완료된 시간 이 시간은 UTC 표준 시간대 기준입니다.
total_exec_time	bigint	쿼리를 실행하는 데 걸린 시간(마이크로초)
service_class_end_time	타임스탬프	쿼리가 서비스 클래스에서 나간 시간 이 시간은 UTC 표준 시간대 기준입니다.
final_state	character(16)	시스템에서 사용하도록 예약됩니다.
est_peak_mem	bigint	시스템에서 사용하도록 예약됩니다.
query_priority	char(20)	쿼리의 우선순위입니다. 가능한 값은 `n/a`, `lowest`, `low`, `normal`, `high` 및 `highest`입니다. 여기에서 `n/a`은 쿼리 우선 순위가 지원되지 않음을 뜻합니다.
service_class_name	character(64)	서비스 클래스 이름입니다. 서비스 클래스에 대한 자세한 내용은 WLM 시스템 테이블 및 뷰를 참조하세요.

샘플 쿼리

쿼리의 평균 대기 및 실행 시간 보기

다음 쿼리는 4보다 큰 서비스 클래스의 현재 구성을 표시합니다. 서비스 클래스 ID의 목록은 WLM 서비스 클래스 ID 섹션을 참조하세요.

다음은 각 쿼리가 쿼리 대기열에서 대기한 평균 시간과 각 서비스 클래스에서 실행하는 데 걸린 평균 시간을 마이크로초 단위로 반환하는 쿼리입니다.

```
select service_class as svc_class, count(*),
avg(datediff(microseconds, queue_start_time, queue_end_time)) as avg_queue_time,
avg(datediff(microseconds, exec_start_time, exec_end_time )) as avg_exec_time
from stl_wlm_query
where service_class > 4
group by service_class
order by service_class;
```

위 쿼리는 다음과 같은 샘플 출력을 반환합니다.

```
 svc_class | count | avg_queue_time | avg_exec_time
-----------+-------+----------------+---------------
         5 | 20103 |              0 |         80415
         5 |  3421 |          34015 |        234015
         6 |    42 |              0 |        944266
         7 |   196 |           6439 |       1364399
(4 rows)
```

쿼리의 최대 대기 및 실행 시간 보기

다음은 쿼리가 대기열에서 대기한 최대 시간과 각 서비스 클래스에서 실행하는 데 걸린 최대 시간을 마이크로초 단위로 반환하는 쿼리입니다.

```
select service_class as svc_class, count(*),
max(datediff(microseconds, queue_start_time, queue_end_time)) as max_queue_time,
max(datediff(microseconds, exec_start_time, exec_end_time )) as max_exec_time
from stl_wlm_query
where svc_class > 5
group by service_class
order by service_class;
```

```
 svc_class | count | max_queue_time | max_exec_time
```

```
-----------+-------+----------------+---------------
         6 |    42 |              0 |       3775896
         7 |   197 |          37947 |      16379473
(4 rows)
```

스냅샷 데이터를 위한 STV 테이블

STV 테이블은 현재 시스템 데이터의 스냅샷을 포함하는 가상 시스템 테이블입니다.

주제

- STV_ACTIVE_CURSORS
- STV_BLOCKLIST
- STV_CURSOR_CONFIGURATION
- STV_DB_ISOLATION_LEVEL
- STV_EXEC_STATE
- STV_INFLIGHT
- STV_LOAD_STATE
- STV_LOCKS
- STV_ML_MODEL_INFO
- STV_MV_DEPS
- STV_MV_INFO
- STV_NODE_STORAGE_CAPACITY
- STV_PARTITIONS
- STV_QUERY_METRICS
- STV_RECENTS
- STV_SESSIONS
- STV_SLICES
- STV_STARTUP_RECOVERY_STATE
- STV_TBL_PERM
- STV_TBL_TRANS
- STV_WLM_CLASSIFICATION_CONFIG
- STV_WLM_QMR_CONFIG
- STV_WLM_QUERY_QUEUE_STATE

- STV_WLM_QUERY_STATE
- STV_WLM_QUERY_TASK_STATE
- STV_WLM_SERVICE_CLASS_CONFIG
- STV_WLM_SERVICE_CLASS_STATE
- STV_XRESTORE_ALTER_QUEUE_STATE

STV_ACTIVE_CURSORS

STV_ACTIVE_CURSORS는 현재 열려있는 커서의 세부 정보를 표시합니다. 자세한 내용은 DECLARE 단원을 참조하십시오.

STV_ACTIVE_CURSORS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오. 사용자는 자신이 연 커서만 볼 수 있습니다. 수퍼유저는 모든 커서를 볼 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
name	characte (256)	커서 이름
xid	bigint	트랜잭션 컨텍스트
pid	정수	쿼리를 실행하는 리더 프로세스
starttime	타임스탬프	커서 선언 시간
row_count	bigint	커서 결과 집합의 행 수
byte_count	bigint	커서 결과 집합의 바이트 수
fetched_rows	bigint	현재 커서 결과 집합에서 가져온 행 수

STV_BLOCKLIST

STV_BLOCKLIST에는 데이터베이스에서 각 조각, 테이블 또는 열마다 사용하는 1MB 디스크 블록의 수가 저장됩니다.

아래 예에 나온 것처럼 STV_BLOCKLIST에 집계 쿼리를 사용하여 데이터베이스, 테이블, 조각 또는 열마다 할당되는 1MB 디스크 블록의 수를 확인합니다. 또한 STV_PARTITIONS를 사용하여 디스크 사용률에 대한 요약 정보를 볼 수 있습니다.

STV_BLOCKLIST는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> STV_BLOCKLIST는 프로비저닝된 클러스터 또는 서버리스 네임스페이스에서 소유한 블록만 기록합니다. 데이터베이스에 데이터 공유 생산자가 공유한 블록이 포함되어 있는 경우 해당 블록은 STV_BLOCKLIST에 포함되지 않습니다. 데이터 공유에 대한 자세한 내용을 알아보려면 Amazon Redshift에서 데이터 공유 섹션으로 이동하세요.

테이블 열

열 명칭	데이터 유형	설명
slice	정수	노드 조각
col	정수	0부터 시작되는 열 인덱스. 생성하는 모든 테이블에는 INSERT_XID, DELETE_XID, ROW_ID(OID)라는 3개의 숨겨진 열이 추가되어 있습니다. 3개의 사용자 정의 열이 있는 테이블에는 6개의 실제 열이 포함되며, 사용자 정의 열은 내부적으로 0, 1, 2로 번호 지정됩니다. 이 예에서 INSERT_XID, DELETE_XID 및 ROW_ID 열은 각각 3, 4, 5로 번호 지정됩니다.
tbl	정수	데이터베이스 테이블의 테이블 ID
blocknum	정수	데이터 블록의 ID.
num_values	정수	블록에 포함된 값의 수.

열 명칭	데이터 유형	설명
extended_limits	정수	내부용.
minvalue	bigint	블록의 최소 데이터 값. 숫자가 아닌 데이터에서 첫 8개 문자를 64비트 정수로 저장하며, 디스크 스캔에 사용됩니다.
maxvalue	bigint	블록의 최대 데이터 값. 숫자가 아닌 데이터에서 첫 8개 문자를 64비트 정수로 저장하며, 디스크 스캔에 사용됩니다.
sb_pos	정수	디스크상의 슈퍼 블록 위치를 나타내는 내부 Amazon Redshift 식별자
pinned	정수	블록이 사전 로드의 일환으로 메모리에 고정되어 있는지 여부. 0 = false, 1 = true. 기본값은 false입니다.
on_disk	정수	블록이 디스크에 자동으로 저장되는지 여부. 0 = false, 1 = true. 기본값은 false입니다.
수정됨	정수	블록이 수정되었는지 여부. 0 = false, 1 = true. 기본값은 false입니다.
hdr_modified	정수	블록 헤더가 수정되었는지 여부. 0 = false, 1 = true. 기본값은 false입니다.
unsorted	정수	블록이 정렬되지 않았는지 여부. 0 = false, 1 = true. 기본값은 true입니다.
tombstone	정수	내부용.
preferred_diskno	정수	디스크에 결함이 발생하지 않은 경우, 블록이 있어야 할 디스크 번호. 디스크가 고쳐지면 블록은 이 디스크로 다시 이동합니다.
임시	정수	블록에 임시 테이블 또는 중간 쿼리 결과 같은 임시 데이터가 포함되는지 여부. 0 = false, 1 = true. 기본값은 false입니다.
newblock	정수	블록이 새것인지(true) 또는 디스크에 한번도 커밋되지 않았는지(false) 여부. 0 = false, 1 = true.
num_readers	정수	각 블록의 참조 수

열 명칭	데이터 유형	설명
flags	정수	블록 헤더의 내부 Amazon Redshift 플래그.

샘플 쿼리

STV_BLOCKLIST는 할당된 디스크 블록당 1개의 행이 포함되므로 모든 행을 선택하는 쿼리는 매우 많은 수의 행을 반환할 수 있습니다. STV_BLOCKLIST에는 집계 쿼리만 사용하는 것이 좋습니다.

SVV_DISKUSAGE 뷰는 유사한 정보를 더욱 사용자 친화적인 형식으로 제공하지만 다음 예는 STV_BLOCKLIST 테이블에 대한 한 가지 용도를 설명한 것입니다.

VENUE 테이블의 각 열에서 사용하는 1MB 블록 수를 확인하려면 다음과 같이 쿼리를 입력합니다.

```
select col, count(*)
from stv_blocklist, stv_tbl_perm
where stv_blocklist.tbl = stv_tbl_perm.id
and stv_blocklist.slice = stv_tbl_perm.slice
and stv_tbl_perm.name = 'venue'
group by col
order by col;
```

위 쿼리는 다음 샘플 데이터와 같이 VENUE 테이블의 각 열에 할당되는 1MB 블록의 수를 반환합니다.

```
 col | count
-----+-------
   0 |  4
   1 |  4
   2 |  4
   3 |  4
   4 |  4
   5 |  4
   7 |  4
   8 |  4
(8 rows)
```

다음은 모든 조각에 대한 테이블 데이터의 실제 분산 여부를 나타내는 쿼리입니다.

```
select trim(name) as table, stv_blocklist.slice, stv_tbl_perm.rows
```

```
from stv_blocklist,stv_tbl_perm
where stv_blocklist.tbl=stv_tbl_perm.id
and stv_tbl_perm.slice=stv_blocklist.slice
and stv_blocklist.id > 10000 and name not like '%#m%'
and name not like 'systable%'
group by name, stv_blocklist.slice, stv_tbl_perm.rows
order by 3 desc;
```

위 쿼리는 다음과 같은 샘플 출력으로 가장 많은 행이 포함된 테이블의 균일한 데이터 분산을 나타냅니다.

```
table   | slice | rows
----------+-------+-------
listing  |    13 | 10527
listing  |    14 | 10526
listing  |     8 | 10526
listing  |     9 | 10526
listing  |     7 | 10525
listing  |     4 | 10525
listing  |    17 | 10525
listing  |    11 | 10525
listing  |     5 | 10525
listing  |    18 | 10525
listing  |    12 | 10525
listing  |     3 | 10525
listing  |    10 | 10525
listing  |     2 | 10524
listing  |    15 | 10524
listing  |    16 | 10524
listing  |     6 | 10524
listing  |    19 | 10524
listing  |     1 | 10523
listing  |     0 | 10521
...
(180 rows)
```

다음은 삭제 표시된 블록의 디스크 커밋 여부를 확인하는 쿼리입니다.

```
select slice, col, tbl, blocknum, newblock
from stv_blocklist
where  tombstone > 0;
```

```
slice | col |   tbl   | blocknum | newblock
------+-----+--------+----------+----------
4     |  0  | 101285 |    0     |   1
4     |  2  | 101285 |    0     |   1
4     |  4  | 101285 |    1     |   1
5     |  2  | 101285 |    0     |   1
5     |  0  | 101285 |    0     |   1
5     |  1  | 101285 |    0     |   1
5     |  4  | 101285 |    1     |   1
...
(24 rows)
```

STV_CURSOR_CONFIGURATION

STV_CURSOR_CONFIGURATION은 커서 구성 제약 조건을 표시합니다. 자세한 내용은 커서 제약 조건 단원을 참조하십시오.

STV_CURSOR_CONFIGURATION은 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
current_cursor_count	정수	현재 열려있는 커서 수
max_diskspace_usable	정수	커서에 사용할 수 있는 디스크 공간 크기(MB). 이 제약 조건은 클러스터에서 커서 결과 집합의 최대 크기를 기준으로 결정됩니다.
current_diskspace_used	정수	현재 커서에서 사용되고 있는 디스크 공간 크기(MB)

STV_DB_ISOLATION_LEVEL

STV_DB_ISOLATION_LEVEL은 데이터베이스의 현재 격리 수준을 표시합니다. 격리 수준에 대한 자세한 내용은 데이터베이스 생성 섹션을 참조하세요.

STV_DB_ISOLATION_LEVEL은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
db_name	characte (128)	데이터베이스 이름입니다.
isolation _level	characte (20)	데이터베이스의 격리 수준입니다. 가능한 값은 `Serializable` 및 `Snapshot Isolation` 입니다.

STV_EXEC_STATE

STV_EXEC_STATE 테이블은 컴퓨팅 노드에서 활성화되어 실행 중인 쿼리 및 쿼리 단계에 대한 정보를 확인하는 데 사용됩니다.

이 정보는 일반적으로 엔지니어링 문제를 해결하는 목적으로만 사용됩니다. SVV_QUERY_STATE 뷰와 SVL_QUERY_SUMMARY 뷰는 정보를 STV_EXEC_STATE에서 추출합니다.

STV_EXEC_STATE는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_DETAIL에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 다양한 다른 시스템 테이블 및 보기를 조인하는 데 사용할 수 있습니다.

열 명칭	데이터 유형	설명
slice	정수	단계가 완료된 노드 조각
segment	정수	실행된 쿼리 세그먼트입니다. 쿼리 세그먼트는 연속된 단계들의 집합입니다.
step	정수	완료된 쿼리 세그먼트의 단계입니다. 단계는 쿼리가 수행하는 가장 작은 단위입니다.
starttime	타임스탬프	단계가 실행된 시간입니다.
currenttime	타임스탬프	현재 시간
tasknum	정수	단계 완료을 위해 할당되는 쿼리 태스크 프로세스입니다.
rows	bigint	처리된 행 수
bytes	bigint	처리된 바이트 수
레이블	char(256)	쿼리 단계 이름 및 해당되는 경우, 테이블 ID와 테이블 이름으로 구성되는 단계 레이블(예: `scan tbl=100448 name =user`). 3자리 테이블 ID는 일반적으로 일시적 테이블의 스캔을 가리킵니다. `tbl=0`이 표시된다면 일반적으로 일정한 값의 스캔을 가리킵니다.
is_diskbased	char(1)	디스크 기반 작업으로서 이번 쿼리 단계의 완료 여부: true(**t**) 또는 false(**f**). 해시, 정렬, 집계 단계 같은 특정 단계만 디스크로 갈 수 있습니다. 많은 단계 형식은 항상 메모리에서 완료됩니다.
workmem	bigint	단계에 할당되는 유효 메모리의 바이트 수
num_parts	정수	해시 단계 도중 해시 테이블이 분할되는 파티션의 수. 이 열의 양수는 해시 단계가 디스크 기반 작업으로 실행되었음을 뜻하지 않습니다. 해시 단계가 디스크 기반인지 확인하려면 IS_DISKBASED 열의 값을 확인합니다.
is_rrscan	char(1)	true(**t**)인 경우, 단계에서 범위 제한 스캔이 사용되었음을 나타냅니다. 기본값은 false(**f**)입니다.

열 명칭	데이터 유형	설명
is_delayed_scan	char(1)	true(**t**)인 경우, 단계에서 지연된 스캔이 사용되었음을 나타냅니다. 기본값은 false(**f**)입니다.

샘플 쿼리

STV_EXEC_STATE에 대해 직접 쿼리를 실행하기보다는 SVL_QUERY_SUMMARY 또는 SVV_QUERY_STATE에 대해 쿼리를 실행하여 더욱 사용자 친화적인 형식으로 STV_EXEC_STATE의 정보를 가져오는 방법이 권장됩니다. 자세한 내용은 SVL_QUERY_SUMMARY 또는 SVV_QUERY_STATE 테이블 설명서를 참조하십시오.

STV_INFLIGHT

STV_INFLIGHT 테이블은 클러스터에서 현재 실행 중인 쿼리를 확인하는 데 사용됩니다. 문제를 해결하는 경우 장기 실행 쿼리의 상태를 확인하는 데 유용합니다.

단, 리더 노드 전용 쿼리를 표시하지는 않습니다. 자세한 내용은 리더 노드 전용 함수 단원을 참조하십시오. STV_INFLIGHT은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

STV_INFLIGHT를 사용한 문제 해결

쿼리 또는 쿼리 모음에 대한 성능 문제를 해결하기 위해 STV_INFLIGHT를 사용하는 경우 다음 사항에 유의하세요.

- 장기간 실행 중인 미결 트랜잭션은 일반적으로 부하를 증가시킵니다. 이러한 미결 트랜잭션으로 인해 다른 쿼리의 실행 시간이 길어질 수 있습니다.
- 장기 실행되는 복사 및 ETL 작업은 많은 컴퓨팅 리소스를 사용하는 경우 클러스터에서 실행 중인 다른 쿼리에 영향을 미칠 수 있습니다. 대부분의 경우 이러한 장기 실행 작업을 사용량이 적은 시간으로 옮기면 보고 또는 분석 워크로드의 성능이 향상됩니다.
- STV_INFLIGHT와 관련된 정보를 제공하는 뷰가 있습니다. 여기에는 SQL 명령에 대한 쿼리 텍스트를 캡처하는 STL_QUERYTEXT와 STV_INFLIGHT를 SVV_QUERY_INFLIGHT에 조인하는 STL_QUERYTEXT가 포함됩니다. 문제 해결을 위해 STV_RECENTS를 STV_INFLIGHT와 함께

사용할 수도 있습니다. 예를 들어 STV_RECENTS는 특정 쿼리가 실행 중 또는 완료 상태에 있는지 여부를 나타낼 수 있습니다. 이 정보를 STV_INFLIGHT의 결과와 결합하면 쿼리의 속성 및 컴퓨팅 리소스 영향에 대한 자세한 정보를 얻을 수 있습니다.

Amazon Redshift 콘솔을 사용하여 실행 중인 쿼리를 모니터링할 수도 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
slice	정수	쿼리가 실행 중인 조각.
쿼리	정수	쿼리 ID. 다양한 다른 시스템 테이블 및 보기를 조인하는 데 사용할 수 있습니다.
레이블	character(320)	쿼리 실행에 사용되는 파일의 이름 또는 SET QUERY_GROUP 명령을 사용하여 정의되는 레이블. 쿼리가 파일 기반이 아니거나 QUERY_GROUP 파라미터가 설정되지 않은 경우, 이 필드의 값은 공백입니다.
xid	bigint	트랜잭션 ID.
pid	정수	프로세스 ID. 한 세션의 모든 쿼리는 동일 프로세스에서 실행되므로 동일 세션에서 일련의 쿼리를 실행하는 경우, 이 값은 항상 같은 값을 유지합니다. 이 열을 사용하여 STL_ERROR 테이블에 조인할 수 있습니다.
starttime	타임스탬프	쿼리가 시작된 시간.
텍스트	character(100)	쿼리 텍스트. 문이 제한을 초과하면 100자까지 잘립니다.
suspended	정수	쿼리가 일시 중지되었는지 여부. 0 = false, 1 = true.
insert_pristine	정수	현재 쿼리를 지금/이전에 실행했을 때 쓰기 쿼리를 지금/이전에 실행할 수 있는 여부입니다. 1 = 쓰기 쿼리가 허용되지 않음. 0 = 쓰기 쿼리가 허용됨. 이 열은 디버깅 시 사용하도록 되어 있습니다.

열 명칭	데이터 유형	설명
concurrency_scaling_status	정수	쿼리가 기본 클러스터에서 실행되었는지 아니면 동시성 확장 클러스터에서 실행되었는지를 나타냅니다. 가능한 값은 다음과 같습니다. 0 - 기본 클러스터에서 실행되었습니다. 1 - 동시성 확장 클러스터에서 실행되었습니다.

샘플 쿼리

현재 데이터베이스에서 활성화되어 실행 중인 모든 쿼리를 확인하려면 다음과 같이 쿼리를 입력합니다.

```
select * from stv_inflight;
```

아래 샘플 출력은 STV_INFLIGHT 쿼리 자체와 이름이 avgwait.sql인 스크립트에서 실행된 쿼리를 포함하여 현재 2개의 쿼리가 실행 중인 것을 나타냅니다.

```
select slice, query, trim(label) querylabel, pid,
starttime, substring(text,1,20) querytext
from stv_inflight;

slice|query|querylabel | pid |        starttime         |      querytext
-----+-----+-----------+-----+--------------------------+--------------------
1011 |  21 |           | 646 |2012-01-26 13:23:15.645503|select slice, query,
1011 |  20 |avgwait.sql| 499 |2012-01-26 13:23:14.159912|select avg(datediff(
(2 rows)
```

다음 쿼리는 concurrency_scaling_status를 포함한 여러 열을 선택합니다. 이 열은 쿼리가 동시성 확장 클러스터로 전송되고 있는지 여부를 나타냅니다. 일부 결과에 대해 값이 1이면 동시성 규모 조정 컴퓨팅 리소스가 사용 중임을 나타냅니다. 자세한 내용은 동시성 확장 단원을 참조하십시오.

```
select userid,
query,
pid,
starttime,
text,
suspended,
```

```
concurrency_scaling_status
 from STV_INFLIGHT;
```

샘플 출력은 동시성 규모 조정 클러스터로 전송되는 쿼리 하나를 보여줍니다.

```
 query  | pid     |        starttime           |   text                 | suspended
 |  concurrency_scaling_status
--------+---------
+----------------------------|------------------------|---------------|-----------------------
1234567 | 123456  | 2012-01-26 13:23:15.645503 | select userid, query...  0
    1
2345678 | 234567  | 2012-01-26 13:23:14.159912 | select avg(datediff(...  0
    0
(2 rows)
```

쿼리 성능 문제 해결에 대한 자세한 팁은 쿼리 문제 해결을 참조하세요.

STV_LOAD_STATE

STV_LOAD_STATE 테이블은 현재 진행 중인 COPY 문의 상태에 대한 정보를 확인할 때 사용됩니다.

100만 개의 레코드가 로드될 때마다 COPY 명령이 이 테이블을 업데이트합니다.

STV_LOAD_STATE는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
session	정수	로드 중인 프로세스의 세션 PID
쿼리	정수	쿼리 ID. 다양한 다른 시스템 테이블 및 보기를 조인하는 데 사용할 수 있습니다.
slice	정수	노드 조각 번호

열 명칭	데이터 유형	설명
pid	정수	프로세스 ID. 한 세션의 모든 쿼리는 동일 프로세스에서 실행되므로 동일 세션에서 일련의 쿼리를 실행하는 경우, 이 값은 항상 같은 값을 유지합니다.
recordtime	타임스탬프	레코드가 기록된 시간
bytes_to_load	bigint	이 조각에서 로드되는 총 바이트 수. 로드되는 데이터가 압축되는 경우 이 값은 0입니다.
bytes_loaded	bigint	이 조각에서 로드된 바이트 수. 로드되는 데이터가 압축되는 경우에는 데이터 압축을 풀고 나서 로드되는 바이트 수가 여기에 해당합니다.
bytes_to_load_compressed	bigint	이 조각에서 로드되는 압축 데이터의 총 바이트 수. 로드되는 데이터가 압축되지 않는 경우 이 값은 0입니다.
bytes_loaded_compressed	bigint	이 조각에서 로드된 압축 데이터의 바이트 수. 로드되는 데이터가 압축되지 않는 경우 이 값은 0입니다.
lines	정수	이 조각에서 로드된 라인 수
num_files	정수	이 조각에서 로드되는 파일 수
num_files_complete	정수	이 조각에서 로드된 파일 수
current_file	character (256)	이 조각에서 로드 중인 파일 이름
pct_complete	정수	이 조각에서 완료된 데이터 로드 비율

샘플 쿼리

각 조각에서 COPY 명령 진행 상황을 보려면 다음과 같이 쿼리를 입력합니다. 다음은 PG_LAST_COPY_ID() 함수를 사용하여 마지막 COPY 명령 정보를 가져오는 예입니다.

```
select slice , bytes_loaded, bytes_to_load , pct_complete from stv_load_state where
 query = pg_last_copy_id();

 slice | bytes_loaded | bytes_to_load | pct_complete
-------+--------------+---------------+--------------
     2 |            0 |             0 |            0
     3 |     12840898 |      39104640 |           32
(2 rows)
```

STV_LOCKS

STV_LOCKS 테이블은 데이터베이스의 테이블에 대한 현재 업데이트를 모두 확인하는 데 사용됩니다.

Amazon Redshift는 사용자 2명이 동시에 똑같은 테이블을 업데이트하지 못하도록 테이블을 잠급니다. STV_LOCKS 테이블이 현재 테이블 업데이트를 모두 표시할 때 STL_TR_CONFLICT 테이블에 대한 쿼리를 실행하여 잠금 충돌 로그를 확인합니다. SVV_TRANSACTIONS 뷰를 사용하여 열린 트랜잭션과 잠금 경합 문제를 식별하십시오.

STV_LOCKS는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
table_id	bigint	잠금 처리된 테이블의 테이블 ID
last_commit	타임스탬프	테이블의 마지막 커밋 타임스탬프
last_update	타임스탬프	테이블의 마지막 업데이트 타임스탬프
lock_owner	bigint	잠금과 연결된 트랜잭션 ID
lock_owner_pid	bigint	잠금에 연결된 프로세스 ID.
lock_owner_start_ts	타임스탬프	트랜잭션 시작 시간 타임스탬프
lock_owner_end_ts	타임스탬프	트랜잭션 종료 시간 타임스탬프

열 명칭	데이터 유형	설명
lock_status	character(22)	잠금 대기 또는 잠금 유지 등 프로세스 상태

샘플 쿼리

현재 트랜잭션에서 일어나는 모든 잠금을 보려면 다음과 같이 명령을 입력합니다.

```
select table_id, last_update, lock_owner, lock_owner_pid from stv_locks;
```

위 쿼리를 실행하면 다음과 같이 현재 활성화된 잠금 3개를 표시하는 샘플 출력을 반환합니다.

```
 table_id |        last_update         | lock_owner | lock_owner_pid
----------+----------------------------+------------+----------------
100004   | 2008-12-23 10:08:48.882319 |       1043 |           5656
100003   | 2008-12-23 10:08:48.779543 |       1043 |           5656
100140   | 2008-12-23 10:08:48.021576 |       1043 |           5656
(3 rows)
```

STV_ML_MODEL_INFO

기계 학습 모델의 현재 상태에 대한 상태 정보입니다.

STV_ML_MODEL_INFO는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
schema_name	char(128)	모델의 네임스페이스입니다.
user_name	char(128)	모델의 소유자입니다.
model_name	char(128)	모델의 이름입니다.
life_cycle	char(20)	모델의 수명 주기 상태입니다.

열 명칭	데이터 유형	설명
is_refreshable	정수	훈련 쿼리의 원래 테이블과 열이 여전히 존재하고 사용자에게 여전히 권한이 있는 경우 모델을 새로 고칠 수 있는지 여부의 모델 상태입니다. 가능한 값은 1(새로 고칠 수 있음)과 0(새로 고칠 수 없음)입니다.
model_state	char(128)	모델의 현재 상태입니다.

샘플 쿼리

다음 쿼리는 기계 학습 모델의 현재 상태를 표시합니다.

```
SELECT schema_name, model_name, model_state
FROM stv_ml_model_info;

 schema_name |        model_name             |             model_state
-------------+-------------------------------+-----------------------------------
 public      | customer_churn_auto_model     | Train Model On SageMaker In Progress
 public      | customer_churn_xgboost_model | Model is Ready
(2 row)
```

STV_MV_DEPS

STV_MV_DEPS 테이블은 Amazon Redshift 내의 다른 구체화된 뷰에 대한 구체화된 뷰의 종속성을 보여줍니다.

구체화된 뷰에 대한 자세한 내용은 Amazon Redshift의 구체화된 뷰 섹션을 참조하세요.

STV_MV_DEPS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
db_name	char(128)	지정한 구체화된 뷰를 포함하는 데이터베이스입니다.
스키마	char(128)	구체화된 보기의 스키마입니다.
name	char(128)	구체화된 보기의 이름입니다.
ref_schema	char(128)	이 구체화된 뷰가 종속된 구체화된 뷰 스키마입니다.
ref_name	char(128)	이 구체화된 뷰가 종속된 구체화된 뷰의 이름입니다.
ref_database_name	char(128)	이 구체화된 뷰가 종속된 데이터베이스의 이름입니다.

샘플 쿼리

다음 쿼리는 구체화된 뷰 mv_over_foo가 정의에서 구체화된 뷰 mv_foo를 종속성으로 사용함을 나타내는 출력 행을 반환합니다.

```
CREATE SCHEMA test_ivm_setup;
CREATE TABLE test_ivm_setup.foo(a INT);
CREATE MATERIALIZED VIEW test_ivm_setup.mv_foo AS SELECT * FROM test_ivm_setup.foo;
CREATE MATERIALIZED VIEW test_ivm_setup.mv_over_foo AS SELECT * FROM
 test_ivm_setup.mv_foo;

SELECT * FROM stv_mv_deps;

 db_name | schema          | name        |   ref_schema   | ref_name |
 ref_database_name
---------+-----------------+-------------+----------------+----------
+------------------
 dev     | test_ivm_setup  | mv_over_foo | test_ivm_setup | mv_foo   | dev
```

STV_MV_INFO

STV_MV_INFO 테이블에는 모든 구체화된 보기의 행, 데이터가 기간이 지났는지 여부 및 상태 정보가 포함되어 있습니다.

구체화된 뷰에 대한 자세한 내용은 Amazon Redshift의 구체화된 뷰 섹션을 참조하세요.

STV_MV_INFO는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
db_name	char(128)	구체화된 보기를 포함하는 데이터베이스입니다.
스키마	char(128)	데이터베이스의 스키마입니다.
name	char(128)	구체화된 보기 이름입니다.
updated_upto_xid	bigint	내부용으로 예약되어 있습니다.
is_stale	char(1)	t는 구체화된 보기가 오래 되었음을 나타냅니다. 오래된 구체화된 보기는 기본 테이블이 업데이트 되었지만 구체화된 보기가 새로 고쳐지지 않은 것입니다. 마지막 재시작 이후 새로 고침을 실행하지 않은 경우 정보가 정확하지 않을 수 있습니다. 구체화된 뷰가 가변 함수에 따라 달라지는 경우 is_stale 열은 항상 t로 설정됩니다. 가변 함수는 동일한 인수가 하나 이상 지정되면 다른 결과를 반환합니다. 예를 들어, 날짜 또는 타임스탬프를 반환하는 대부분의 함수는 가변 함수입니다.
owner_user_name	char(128)	구체화된 보기를 소유한 사용자입니다.
state	정수	구체화된 보기의 상태는 다음과 같습니다.

열 명칭	데이터 유형	설명
		• 0 - 새로 고칠 때 구체화된 뷰가 완전히 다시 계산됩니다. • 1 - 구체화된 뷰가 증분적입니다. • 101 - 삭제된 열로 인해 구체화된 뷰를 새로 고칠 수 없습니다. 이 제약 조건은 구체화된 보기에서 열이 사용되지 않는 경우에도 적용됩니다. • 102 - 변경된 열 형식으로 인해 구체화된 뷰를 새로 고칠 수 없습니다. 이 제약 조건은 구체화된 보기에서 열이 사용되지 않는 경우에도 적용됩니다. • 103 - 이름이 변경된 테이블로 인해 구체화된 뷰를 새로 고칠 수 없습니다. • 104 - 이름이 변경된 열로 인해 구체화된 뷰를 새로 고칠 수 없습니다. 이 제약 조건은 구체화된 보기에서 열이 사용되지 않는 경우에도 적용됩니다. • 105 - 이름이 변경된 스키마로 인해 구체화된 뷰를 새로 고칠 수 없습니다.
autorewrite	char(1)	t는 구체화된 뷰가 쿼리를 자동으로 다시 작성할 수 있음을 나타냅니다.
autorefresh	char(1)	t는 구체화된 뷰가 자동으로 새로 고쳐질 수 있음을 나타냅니다.

샘플 쿼리

모든 구체화된 보기의 상태를 보려면 다음 쿼리를 실행합니다.

```
select * from stv_mv_info;
```

위 쿼리는 다음과 같은 샘플 출력을 반환합니다.

```
 db_name |       schema       |   name   | updated_upto_xid | is_stale | owner_user_name
 | state | autorefresh | autorewrite
```

```
---------+--------------------+---------+------------------+----------
+-----------------+-------+------------+------------
 dev     | test_ivm_setup     | mv      |             1031 | f        | catch-22
 |     1 |           1 |           0
 dev     | test_ivm_setup     | old_mv  |              988 | t        | lotr
 |     1 |           0 |           1
```

STV_NODE_STORAGE_CAPACITY

STV_NODE_STORAGE_CAPACITY 테이블에서는 클러스터의 각 노드에 대한 총 스토리지 용량 및 총 사용 용량에 대한 세부 정보를 보여줍니다. 여기에는 각 노드에 대한 행이 포함됩니다.

STV_NODE_STORAGE_CAPACITY는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
노드	정수	노드 번호.
used	정수	현재 노드에서 사용 중인 1MB 디스크 블록 수. RA3 노드 유형의 경우 사용된 블록에는 로컬로 캐시된 블록과 Amazon S3에 지속되는 블록이 모두 포함됩니다.
재조정	정수	노드에 프로비저닝된 총 스토리지 용량(1MB 블록)입니다. 용량에는 내부 사용을 위해 DC2 노드 유형에서 Amazon Redshift가 예약한 공간이 포함됩니다. 용량은 사용자 데이터에 사용할 수 있는 노드 공간의 양인 공칭 노드 용량보다 큽니다. RA3 노드 유형의 경우 이 용량은 클러스터의 총 관리 스토리지 할당량과 동일합니다. 노드 유형별 용량에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 노드 유형 세부 정보 섹션을 참조하세요.

샘플 쿼리

> **Note**
>
> 다음 예의 결과는 클러스터의 노드 사양에 따라 다릅니다. SQL SELECT에 열 `capacity`를 추가하여 클러스터의 용량을 검색합니다.

다음 쿼리는 사용된 공간과 총 용량을 1MB 디스크 블록으로 반환합니다. 이 예는 2노드 dc2.8xlarge 클러스터에서 실행되었습니다.

```
select node, used from stv_node_storage_capacity order by node;
```

위 쿼리는 다음과 같은 샘플 출력을 반환합니다.

```
 node | used
------+-------
    0 | 30597
    1 | 27089
```

다음 쿼리는 사용된 공간과 총 용량을 1MB 디스크 블록으로 반환합니다. 이 예는 2노드 ra3.16xlarge 클러스터에서 실행되었습니다.

```
select node, used from stv_node_storage_capacity order by node;
```

위 쿼리는 다음과 같은 샘플 출력을 반환합니다.

```
 node | used
------+-------
    0 | 30591
    1 | 27103
```

STV_PARTITIONS

STV_PARTITIONS 테이블은 Amazon Redshift의 디스크 속도 성능과 디스크 사용률을 확인하는 데 사용됩니다.

STV_PARTITIONS에는 논리적 디스크 볼륨을 기준으로 노드 1개당 행 1개가 포함됩니다.

STV_PARTITIONS는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
owner	정수	파티션을 소유하는 디스크 노드
host	정수	파티션에 물리적으로 연결된 노드
diskno	정수	파티션이 포함된 노드
part_begin	bigint	파티션의 오프셋. 원시 디바이스는 미러 블록을 위한 열린 공간으로 논리적으로 분할됩니다.
part_end	bigint	파티션의 끝
used	정수	현재 파티션에서 사용 중인 1MB 디스크 블록 수
tossed	정수	삭제 대기 중이지만 디스크 주소를 해제하기가 안전하지 않아 아직 제거되지 않은 블록의 수. 주소가 바로 해제되면 대기 중인 트랜잭션이 동일한 디스크 위치로 쓰기를 실행할 수 있습니다. 그 결과 다음 커밋부터는 토스된 블록이 해제됩니다. 예를 들어 INSERT 작업이나 디스크 기반 쿼리 작업 도중 테이블 열이 삭제되면 디스크 블록이 토스됨(tossed)으로 표시될 수 있습니다.
재조정	정수	1MB 디스크 블록 파티션의 총 용량
reads	bigint	마지막 클러스터 재시작 이후 실행된 읽기 수
writes	bigint	마지막 클러스터 재시작 이후 실행된 쓰기 수
seek_forward	정수	이전 요청 주소를 고려했을 때 요청이 이후 주소에 적합하지 않은 횟수
seek_back	정수	이후 요청 주소를 고려했을 때 요청이 이전 주소에 적합하지 않은 횟수

열 명칭	데이터 유형	설명
is_san	정수	파티션의 SAN 종속 여부. 유효한 값은 **0**(false) 또는 **1**(true)입니다.
failed	정수	이 열은 더 이상 사용되지 않습니다.
mbps	정수	초당 디스크 속도(MB)
마운트에 의해	character (256)	디바이스 디렉터리 경로

샘플 쿼리

아래 쿼리는 1MB 디스크 블록에서 사용된 디스크 공간과 용량을 반환한 후 디스크 사용량을 원시 디스크 공간 비율로 계산합니다. 원시 디스크 공간에는 Amazon Redshift가 내부 사용 목적으로 예약하는 공간도 포함되므로 사용자가 사용할 수 있는 디스크 공간 크기인 공칭 디스크 용량보다 더 커야 합니다. Amazon Redshift 관리 콘솔의 [성능(Performance)] 탭에 있는 [사용된 디스크 공간의 백분율(Percentage of Disk Space Used)] 지표는 클러스터에서 사용하는 공칭 디스크 용량의 백분율을 보고합니다. 여기에서 사용된 디스크 공간의 비율(%) 지표를 모니터링하면서 클러스터의 공칭 디스크 용량을 넘지 않도록 사용량을 유지하는 것이 좋습니다.

> **Important**
>
> 클러스터의 공칭 디스크 용량을 초과하지 않는 것이 좋습니다. 특정 환경에서 기술적으로 가능할 수 있지만, 공칭 디스크 용량을 초과하면 클러스터의 내결함성이 감소하고 데이터 손실 위험이 증가합니다.

다음은 노드 1개당 논리적 디스크 파티션이 6개인 2노드 클러스터에서 실행한 예입니다. 각 디스크 사용률이 약 25%를 기록하면서 디스크 공간이 매우 균일하게 사용되고 있습니다.

```
select owner, host, diskno, used, capacity,
(used-tossed)/capacity::numeric *100 as pctused
from stv_partitions order by owner;

 owner | host | diskno |  used   | capacity | pctused
-------+------+--------+---------+----------+---------
   0   |  0   |    0   | 236480 |  949954  | 24.9
```

```
   0   |  0   |    1   | 236420 |  949954  | 24.9
   0   |  0   |    2   | 236440 |  949954  | 24.9
   0   |  1   |    2   | 235150 |  949954  | 24.8
   0   |  1   |    1   | 237100 |  949954  | 25.0
   0   |  1   |    0   | 237090 |  949954  | 25.0
   1   |  1   |    0   | 236310 |  949954  | 24.9
   1   |  1   |    1   | 236300 |  949954  | 24.9
   1   |  1   |    2   | 236320 |  949954  | 24.9
   1   |  0   |    2   | 237910 |  949954  | 25.0
   1   |  0   |    1   | 235640 |  949954  | 24.8
   1   |  0   |    0   | 235380 |  949954  | 24.8

(12 rows)
```

STV_QUERY_METRICS

사용자 정의 쿼리 대기열(서비스 클래스)에서 실행 중인 활성 쿼리의 지표 정보(처리된 행의 수, CPU 사용량, 입/출력, 디스크 사용량)가 저장됩니다. 완료된 쿼리의 지표를 보려면 STL_QUERY_METRICS 시스템 테이블을 참조하십시오.

쿼리 지표는 1초 간격으로 샘플링됩니다. 결과적으로 동일한 쿼리라고 해도 다르게 실행하면 약간 다른 시간을 반환할 수 있습니다. 또한 1초 이내에 실행되는 쿼리 세그먼트는 기록되지 않을 수도 있습니다.

STV_QUERY_METRICS은 지표를 쿼리, 세그먼트 및 단계 수준으로 추적 및 집계합니다. 쿼리 세그먼트 및 단계에 대한 자세한 내용은 쿼리 계획 및 실행 워크플로우 섹션을 참조하세요. 노드 조각에서 합산되는 지표도 max_rows, cpu_time 등 매우 많습니다. 노드 조각에 대한 자세한 내용은 데이터 웨어하우스 시스템 아키텍처 섹션을 참조하세요.

행이 지표를 보고하는 수준을 확인하고 싶으면 다음과 같이 segment 열과 step_type 열을 살펴보십시오.

- segment 열과 step_type 열의 값이 모두 -1이면 행이 쿼리 수준에서 지표를 보고합니다.
- segment 열이 -1이 아니고, step_type 열이 -1이면 행이 세그먼트 수준에서 지표를 보고합니다.
- segment 열과 step_type 열의 값이 모두 -1이 아니면 행이 단계 수준에서 지표를 보고합니다.

STV_QUERY_METRICS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_DETAIL에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 쿼리를 실행한 사용자의 ID.
service_class	정수	WLM 쿼리 대기열(서비스 클래스)의 ID. 쿼리 대기열은 WLM 구성에서 정의합니다. 지표는 사용자 정의 대기열에 대해서만 보고됩니다.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
starttime	타임스탬프	쿼리 실행이 시작된 UTC 시간으로 소수 초의 정밀도는 6자리입니다. 예: `2009-06-12 11:29:19.131358` .
슬라이스	정수	클러스터의 조각 수
segment	정수	세그먼트 번호. 하나의 쿼리는 여러 세그먼트로 구성되며, 각각의 세그먼트는 하나 이상의 단계로 구성됩니다. 쿼리 세그먼트는 병렬로 실행될 수 있습니다. 각 세그먼트는 단일 프로세스에서 실행됩니다. 세그먼트 값이 -1인 경우, 지표 세그먼트 값은 쿼리 수준으로 롤업됩니다.
step_type	정수	실행된 단계의 유형입니다. 단계 유형에 대한 자세한 내용은 단계 유형 섹션을 참조하세요.
rows	bigint	단계에서 처리된 행의 수
max_rows	bigint	모든 조각을 집계하여 단계에서 출력된 행의 최대 수
cpu_time	bigint	사용된 CPU 시간(마이크로초). 세그먼트 수준일 때는 모든 조각을 통틀어 세그먼트에서 사용된 총 CPU 시간입니다. 쿼리 수준일 때는 모든 조각과 세그먼트를 통틀어 쿼리에서 사용된 CPU 시간의 합산입니다.

열 명칭	데이터 유형	설명
max_cpu_time	bigint	사용된 최대 CPU 시간(마이크로초). 세그먼트 수준일 때는 모든 조각을 통틀어 세그먼트에서 사용된 최대 CPU 시간입니다. 쿼리 수준일 때는 모든 쿼리 세그먼트에서 사용된 최대 CPU 시간입니다.
blocks_read	bigint	쿼리 또는 세그먼트에서 읽은 1MB 블록의 수
max_blocks_read	bigint	모든 조각을 집계하여 세그먼트에서 읽은 1MB 블록의 최대 수. 세그먼트 수준일 때는 모든 조각을 통틀어 세그먼트에서 읽은 1MB 블록의 최대 수입니다. 쿼리 수준일 때는 모든 쿼리 세그먼트에서 읽은 1MB 블록의 최대 수입니다.
run_time	bigint	조각마다 합산한 총 실행 시간. 실행 시간에는 대기 시간이 포함되지 않습니다. 세그먼트 수준일 때는 모든 조각을 통틀어 합산된 세그먼트 실행 시간입니다. 쿼리 수준일 때는 모든 조각과 세그먼트를 통틀어 합산된 쿼리 실행 시간입니다. 이 값은 합산된 것이기 때문에 실행 시간과 쿼리 실행 시간은 관련이 없습니다.
max_run_time	bigint	세그먼트의 최대 경과 시간(마이크로초). 세그먼트 수준일 때는 모든 조각에서 세그먼트의 최대 실행 시간입니다. 쿼리 수준일 때는 모든 쿼리 세그먼트의 최대 실행 시간입니다.
max_blocks_to_disk	bigint	중간 결과를 작성하는 데 사용한 디스크 공간의 최대 크기(1MB 블록). 세그먼트 수준일 때는 모든 조각을 통틀어 세그먼트에서 사용된 디스크 공간의 최대 크기입니다. 쿼리 수준일 때는 모든 쿼리 세그먼트에서 사용된 디스크 공간의 최대 크기입니다.
blocks_to_disk	bigint	쿼리 또는 세그먼트에서 중간 결과를 작성하는 데 사용한 디스크 공간 크기(1MB 블록)
step	정수	실행된 쿼리 단계입니다.

열 명칭	데이터 유형	설명
max_query_scan_size	bigint	쿼리에서 스캔된 데이터의 최대 크기(MB). 세그먼트 수준일 때는 모든 조각을 통틀어 세그먼트에서 스캔된 데이터의 최대 크기입니다. 쿼리 수준일 때는 모든 쿼리 세그먼트에서 스캔된 데이터의 최대 크기입니다.
query_scan_size	bigint	쿼리에서 스캔된 데이터의 크기(MB).
query_priority	정수	쿼리의 우선순위입니다. 가능한 값은 -1, 0, 1, 2, 3 및 4입니다. 여기에서 -1은 쿼리 우선 순위가 지원되지 않음을 뜻합니다.
query_queue_time	bigint	쿼리가 대기열에 있는 시간(마이크로초)입니다.

단계 유형

다음 표는 데이터베이스 사용자와 관련된 단계 유형을 나열한 것입니다. 내부 전용 단계 유형은 표에서 제외되었습니다. 단계 유형이 -1이면 단계 수준에서는 지표가 보고되지 않습니다.

단계 유형	설명
1	테이블 스캔
2	행 삽입
3	행 집계
6	정렬 단계
7	병합 단계
8	분산 단계
9	브로드캐스팅 분산 단계
10	해시 조인

단계 유형	설명
11	병합 조인
12	저장 단계
14	해시
15	중첩 루프 조인
16	프로젝트 필드 및 표현식
17	반환되는 행의 수 제한
18	고유
20	행 삭제
26	반환되는 정렬 행의 수 제한
29	창 함수 계산
32	UDF
33	고유
37	행을 리더 노드에서 클라이언트로 반환
38	행을 컴퓨팅 노드에서 리더 노드로 반환
40	스펙트럼 스캔

샘플 쿼리

CPU 시간이 높은(1,000초 이상) 활성 쿼리를 찾으려면 다음과 같이 쿼리를 실행합니다.

```
select query, cpu_time / 1000000 as cpu_seconds
from stv_query_metrics where segment = -1 and cpu_time > 1000000000
order by cpu_time;

query | cpu_seconds
------+------------
```

```
25775 |        9540
```

중첩 루프 조인을 사용하여 반환된 행의 수가 100만 개가 넘는 활성 쿼리를 찾으려면 다음과 같이 쿼리를 실행합니다.

```
select query, rows
from stv_query_metrics
where step_type = 15 and rows > 1000000
order by rows;

query | rows
------+-----------
25775 | 1580225854
```

실행 시간이 60초보다 크고, CPU 시간이 10초 미만인 활성 쿼리를 찾으려면 다음과 같이 쿼리를 실행하십시오.

```
select query, run_time/1000000 as run_time_seconds
from stv_query_metrics
where segment = -1 and run_time > 60000000 and cpu_time < 10000000;

query | run_time_seconds
------+-----------------
25775 |              114
```

STV_RECENTS

STV_RECENTS 테이블은 데이터베이스에서 현재 활성 쿼리와 최근에 실행한 쿼리에 대한 정보를 확인하는 데 사용됩니다.

STV_RECENTS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

STV_RECENTS를 사용한 문제 해결

STV_RECENTS는 쿼리 또는 쿼리 모음이 현재 실행 중인지 또는 완료되었는지 확인하는 데 특히 유용합니다. 또한 쿼리가 실행된 기간도 표시됩니다. 이는 어떤 쿼리가 오래 실행되는지 파악하는 데 유용합니다.

STV_RECENTS를 STV_INFLIGHT와 같은 다른 시스템 보기에 조인하여 실행 중인 쿼리에 대한 추가 메타데이터를 수집할 수 있습니다. (샘플 쿼리 섹션에 이 작업을 수행하는 방법을 보여주는 예가 있습니다.) 또한 이 뷰에서 반환된 레코드를 Amazon Redshift 콘솔의 모니터링 기능과 함께 사용하여 실시간으로 문제 해결을 할 수도 있습니다.

STV_RECENTS를 보완하는 시스템 뷰에는 SQL 명령에 대한 쿼리 텍스트를 검색하는 SVV_QUERY_INFLIGHT와 STV_INFLIGHT를 STL_QUERYTEXT에 조인하는 STL_QUERYTEXT가 포함됩니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
status	character(20)	쿼리 상태. 유효 값은 **Running**, **Done**입니다.
starttime	타임스탬프	쿼리가 시작된 시간입니다.
기간	정수	세션 시작 후 경과 시간(마이크로초)
user_name	character(50)	프로세스를 실행한 사용자 이름
db_name	character(50)	데이터베이스 이름
쿼리	character(600)	최대 600자의 쿼리 텍스트. 그 외에 추가되는 문자는 모두 잘립니다.
pid	정수	쿼리와 연결된 세션의 프로세스 ID이며, 완료된 쿼리는 항상 -1입니다.

샘플 쿼리

현재 데이터베이스에서 실행 중인 쿼리를 확인하려면 다음과 같이 쿼리를 실행합니다.

```
select user_name, db_name, pid, query
from stv_recents
where status = 'Running';
```

아래 샘플 출력은 TICKIT 데이터베이스에서 실행 중인 쿼리가 하나인 것을 나타냅니다.

```
user_name | db_name |   pid   | query
----------+---------+---------+-------------
dwuser    | tickit  |  19996  |select venuename, venueseats from
venue where venueseats > 50000 order by venueseats desc;
```

다음은 실행 중이거나 대기열에서 실행 대기 중인 쿼리 목록(있는 경우)을 반환하는 예입니다.

```
select * from stv_recents where status<>'Done';

status |    starttime        | duration |user_name|db_name| query     | pid
-------+---------------------+----------+---------+-------+-----------+------
Running| 2010-04-21 16:11... | 281566454| dwuser  |tickit | select ...| 23347
```

다수의 쿼리를 동시에 실행하면서 일부 쿼리가 대기열에서 대기 중인 경우가 아니라면 위 쿼리에서 아무런 결과도 반환되지 않습니다.

다음은 위의 예를 확장한 예입니다. 이 경우에는 실제로 "진행 중"(대기가 아닌 실행 중)인 쿼리가 결과에서 제외됩니다.

```
select * from stv_recents where status<>'Done'
and pid not in (select pid from stv_inflight);
...
```

쿼리 성능 문제 해결에 대한 자세한 팁은 쿼리 문제 해결을 참조하세요.

STV_SESSIONS

STV_SESSIONS 테이블은 Amazon Redshift에서 활성 사용자 세션에 대한 정보를 확인하는 데 사용됩니다.

세션 이력을 보려면 STV_SESSIONS가 아닌 STL_SESSIONS 테이블을 사용합니다.

STV_SESSIONS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_SESSION_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
starttime	타임스탬프	세션이 시작된 시간
포함	정수	세션의 프로세스 ID.
user_name	character(50)	세션과 연결된 사용자
db_name	character(50)	세션과 연결된 데이터베이스 이름
timeout_sec	int	세션이 시간 초과되기 전에 비활성 또는 유휴 상태로 유지되는 최대 시간(초)입니다. 0은 시간 제한이 설정되지 않았음을 나타냅니다.

샘플 쿼리

현재 Amazon Redshift에 로그인한 사용자가 더 있는지 빠르게 확인하려면 다음과 같이 쿼리를 입력합니다.

```
select count(*)
from stv_sessions;
```

반환되는 결과가 1보다 크면 다른 사용자 1명 이상이 현재 데이터베이스에 로그인한 것을 의미합니다.

Amazon Redshift의 활성 세션을 모두 확인하려면 다음과 같이 쿼리를 입력합니다.

```
select *
from stv_sessions;
```

다음 결과에는 현재 Amazon Redshift에서 실행 중인 활성 세션 4개가 표시됩니다.

```
       starttime          | process |user_name                    | db_name
      | timeout_sec
-------------------------+---------+-----------------------------
+---------------------------+-------------
  2018-08-06 08:44:07.50 |   13779 | IAMA:aws_admin:admin_grp    | dev
      | 0
  2008-08-06 08:54:20.50 |   19829 | dwuser                      | dev
      | 120
  2008-08-06 08:56:34.50 |   20279 | dwuser                      | dev
      | 120
  2008-08-06 08:55:00.50 |   19996 | dwuser                      | tickit
      | 0
(3 rows)
```

IAMA라는 접두사가 붙은 사용자 이름은 해당 사용자가 페더레이션된 Single Sign-On을 사용하여 로그인했음을 나타냅니다. 자세한 내용은 IAM 인증을 이용한 데이터베이스 사용자 자격 증명 생성을 참조하세요.

STV_SLICES

STV_SLICES 테이블은 조각과 노드 간 현재 매핑을 확인하는 데 사용됩니다.

STV_SLICES 정보는 주로 조사 목적으로만 사용됩니다.

STV_SLICES는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
노드	정수	조각이 위치한 클러스터 노드
slice	정수	노드 조각
localslice	정수	이 정보는 내부 전용입니다.

열 명칭	데이터 유형	설명
type	character (1)	이 정보는 내부 전용입니다.

샘플 쿼리

어떤 클러스터 노드에서 어떤 조각을 관리하는지 보려면 다음과 같이 쿼리를 입력합니다.

```
select node, slice from stv_slices;
```

위 쿼리는 다음과 같은 샘플 출력을 반환합니다.

```
 node | slice
------+-------
    0 |     2
    0 |     3
    0 |     1
    0 |     0
(4 rows)
```

STV_STARTUP_RECOVERY_STATE

클러스터 재시작 시 임시로 잠기는 테이블 상태를 기록합니다. Amazon Redshift는 클러스터가 다시 시작된 후 오래된 트랜잭션을 해결하기 위해 테이블이 처리되는 동안 테이블에 임시 잠금을 설정합니다.

STV_STARTUP_RECOVERY_STATE는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
db_id	정수	데이터베이스 ID.
table_id	정수	테이블 ID.

열 명칭	데이터 유형	설명
table_name	character(137)	테이블 이름.

샘플 쿼리

임시로 잠기는 테이블을 모니터링하려면 클러스터 재시작 후 다음과 같이 쿼리를 실행합니다.

```
select * from STV_STARTUP_RECOVERY_STATE;

  db_id | tbl_id | table_name
--------+--------+------------
 100044 | 100058 | lineorder
 100044 | 100068 | part
 100044 | 100072 | customer
 100044 | 100192 | supplier
(4 rows)
```

STV_TBL_PERM

STV_TBL_PERM 테이블에는 사용자가 현재 세션에서 임시로 생성한 테이블을 포함하여 Amazon Redshift의 영구 테이블에 대한 정보가 저장됩니다. 또한 모든 데이터베이스의 전체 테이블에 대한 정보가 저장됩니다.

이 테이블은 STV_TBL_TRANS와 다릅니다. STV_TBL_TRANS 테이블에는 시스템이 쿼리 처리 중 일시적으로 생성하는 데이터베이스 테이블에 대한 정보가 저장됩니다.

STV_TBL_PERM은 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
slice	정수	테이블에 할당되는 노드 조각
id	정수	테이블 ID.

열 명칭	데이터 유형	설명
name	character (72)	테이블 이름
rows	bigint	조각에 포함된 데이터 행의 수
sorted_rows	bigint	조각에서 이미 디스크에 정렬된 행의 수. 이 수가 ROWS 수와 일치하지 않으면 테이블을 정리하여 행을 재정렬하십시오.
temp	정수	테이블의 임시 테이블 여부. 0 = false, 1 = true.
db_id	정수	테이블이 생성된 데이터베이스의 ID
insert_pristine	정수	내부용.
delete_pristine	정수	내부용.
백업	정수	클러스터 스냅샷의 테이블 포함 여부를 나타내는 값. 0 = 예, 1 = 아니요. 자세한 내용은 CREATE TABLE 명령의 BACKUP 파라미터를 참조하십시오.
dist_style	정수	조각이 속한 테이블의 배포 스타일입니다. 값에 대한 자세한 내용은 분산 스타일 보기 섹션을 참조하세요. 배포 스타일에 대한 자세한 내용은 분산 스타일 섹션을 참조하세요.
block_count	정수	조각에서 사용하는 블록 수입니다. 블록 수를 계산할 수 없는 경우 값은 -1입니다.

샘플 쿼리

다음은 서로 다른 테이블 ID와 이름 목록을 반환하는 쿼리입니다.

```
select distinct id, name
from stv_tbl_perm order by name;

   id    |          name
---------+--------------------------
```

```
 100571 | category
 100575 | date
 100580 | event
 100596 | listing
 100003 | padb_config_harvest
 100612 | sales
...
```

다른 시스템 테이블들은 테이블 ID를 사용하기 때문에 특정 테이블의 테이블 ID를 알고 있다면 매우 유용하게 사용할 수 있습니다. 위 예에서 SELECT DISTINCT는 테이블이 다수의 조각으로 분산되면서 중복된 테이블을 제거하기 위해 사용되었습니다.

VENUE 테이블의 각 열에서 사용하는 블록 수를 확인하려면 다음과 같이 쿼리를 입력합니다.

```
select col, count(*)
from stv_blocklist, stv_tbl_perm
where stv_blocklist.tbl = stv_tbl_perm.id
and stv_blocklist.slice = stv_tbl_perm.slice
and stv_tbl_perm.name = 'venue'
group by col
order by col;

 col | count
-----+-------
   0 |     8
   1 |     8
   2 |     8
   3 |     8
   4 |     8
   5 |     8
   6 |     8
   7 |     8
(8 rows)
```

사용 노트

ROWS 열에는 삭제만 되고 아직 정리되지 않은(또는 SORT ONLY 옵션과 함께 정리된) 행의 수가 포함됩니다. 따라서 STV_TBL_PERM 테이블에서 ROWS 열의 SUM 결과는 임의 테이블에 대해 직접 쿼리를 실행했을 때 COUNT(*) 결과와 일치하지 않을 수도 있습니다. 예를 들어 VENUE 테이블에서 행 2개를 삭제한 경우 COUNT(*) 결과가 200이지만 SUM(ROWS) 결과는 여전히 202입니다.

```
delete from venue
```

```
where venueid in (1,2);

select count(*) from venue;
count
-------
200
(1 row)

select trim(name) tablename, sum(rows)
from stv_tbl_perm where name='venue' group by name;

tablename | sum
-----------+-----
venue     | 202
(1 row)
```

STV_TBL_PERM의 데이터를 동기화하려면 VENUE 테이블에 대한 전체 정리를 실행하십시오.

```
vacuum venue;

select trim(name) tablename, sum(rows)
from stv_tbl_perm
where name='venue'
group by name;

tablename | sum
-----------+-----
venue     | 200
(1 row)
```

STV_TBL_TRANS

STV_TBL_TRANS 테이블은 현재 메모리에 저장 중인 임시 데이터베이스 테이블에 대한 정보를 검색하는 데 사용됩니다.

임시 테이블이란 일반적으로 쿼리 실행 시 중간 결과로 사용되는 임시 행 집합을 의미합니다. STV_TBL_TRANS는 STV_TBL_PERM과 다릅니다. STV_TBL_PERM에는 영구 데이터베이스 테이블에 대한 정보가 저장됩니다.

STV_TBL_TRANS는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
slice	정수	테이블에 할당되는 노드 조각
id	정수	테이블 ID.
rows	bigint	테이블에 포함된 데이터 행의 수
size	bigint	테이블에 할당된 바이트 수
query_id	bigint	쿼리 ID.
ref_cnt	정수	참조 수
from_suspended	정수	현재는 중단된 쿼리 실행 시 이 테이블의 생성 여부
prep_swap	정수	필요한 경우 임시 테이블의 디스크 전환 준비 여부. 전환은 메모리가 낮은 상황에서만 일어납니다.

샘플 쿼리

쿼리 ID가 90인 쿼리의 임시 테이블 정보를 보려면 다음과 같이 명령을 입력합니다.

```
select slice, id, rows, size, query_id, ref_cnt
from stv_tbl_trans
where query_id = 90;
```

위 쿼리는 다음 샘플 출력과 같이 쿼리 90의 임시 테이블 정보를 반환합니다.

```
slice | id | rows | size | query_ | ref_ | from_     | prep_
      |    |      |      | id     | cnt  | suspended | swap
------+----+------+------+--------+------+-----------+-------
 1013 | 95 |    0 |    0 |     90 |    4 |         0 |   0
    7 | 96 |    0 |    0 |     90 |    4 |         0 |   0
   10 | 96 |    0 |    0 |     90 |    4 |         0 |   0
   17 | 96 |    0 |    0 |     90 |    4 |         0 |   0
   14 | 96 |    0 |    0 |     90 |    4 |         0 |   0
```

```
  3 | 96 |    0 |    0 |     90 |    4 |          0 |   0
1013 | 99 |    0 |    0 |     90 |    4 |          0 |   0
  9 | 96 |    0 |    0 |     90 |    4 |          0 |   0
  5 | 96 |    0 |    0 |     90 |    4 |          0 |   0
 19 | 96 |    0 |    0 |     90 |    4 |          0 |   0
  2 | 96 |    0 |    0 |     90 |    4 |          0 |   0
1013 | 98 |    0 |    0 |     90 |    4 |          0 |   0
 13 | 96 |    0 |    0 |     90 |    4 |          0 |   0
  1 | 96 |    0 |    0 |     90 |    4 |          0 |   0
1013 | 96 |    0 |    0 |     90 |    4 |          0 |   0
  6 | 96 |    0 |    0 |     90 |    4 |          0 |   0
 11 | 96 |    0 |    0 |     90 |    4 |          0 |   0
 15 | 96 |    0 |    0 |     90 |    4 |          0 |   0
 18 | 96 |    0 |    0 |     90 |    4 |          0 |   0
```

위 예를 보면 쿼리 데이터에 테이블 95, 96 및 98이 포함되어 있는 것을 알 수 있습니다. 이 테이블에 0 바이트가 할당되어 있기 때문에 이러한 쿼리를 메모리에서 실행할 수 있습니다.

STV_WLM_CLASSIFICATION_CONFIG

현재 WLM 분류 규칙이 저장됩니다.

STV_WLM_CLASSIFICATION_CONFIG는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
id	정수	서비스 클래스 ID 서비스 클래스 ID의 목록은 WLM 서비스 클래스 ID 섹션을 참조하세요.
condition	character(128)	쿼리 조건
action_seq	정수	시스템에서 사용하도록 예약됩니다.
작업	character(64)	시스템에서 사용하도록 예약됩니다.
action_service_class	정수	작업이 이루어지는 서비스 클래스

샘플 쿼리

```
select * from STV_WLM_CLASSIFICATION_CONFIG;

id | condition                                    | action_seq | action |
 action_service_class
---+----------------------------------------------+------------+--------
+---------------------
 1 | (system user) and (query group: health)      |          0 | assign |
     1
 2 | (system user) and (query group: metrics)     |          0 | assign |
     2
 3 | (system user) and (query group: cmstats)     |          0 | assign |
     3
 4 | (system user)                                |          0 | assign |
     4
 5 | (super user) and (query group: superuser)    |          0 | assign |
     5
 6 | (query group: querygroup1)                   |          0 | assign |
     6
 7 | (user group: usergroup1)                     |          0 | assign |
     6
 8 | (user group: usergroup2)                     |          0 | assign |
     7
 9 | (query group: querygroup3)                   |          0 | assign |
     8
10 | (query group: querygroup4)                   |          0 | assign |
     9
11 | (user group: usergroup4)                     |          0 | assign |
     9
12 | (query group: querygroup*)                   |          0 | assign |
    10
13 | (user group: usergroup*)                     |          0 | assign |
    10
14 | (querytype: any)                             |          0 | assign |
    11
(4 rows)
```

STV_WLM_QMR_CONFIG

WLM 쿼리 모니터링 규칙(QMR)의 구성을 기록합니다. 자세한 내용은 WLM 쿼리 모니터링 규칙 단원을 참조하십시오.

STV_WLM_QMR_CONFIG는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
service_class	정수	WLM 쿼리 대기열(서비스 클래스)의 ID. 쿼리 대기열은 WLM 구성에서 정의합니다. 규칙은 사용자 정의 대기열에 한해 정의할 수 있습니다. 서비스 클래스 ID의 목록은 WLM 서비스 클래스 ID 섹션을 참조하세요.
rule_name	character(256)	쿼리 모니터링 규칙 이름
작업	character(256)	규칙 작업. 가능한 값은 log, hop, abort 및 change_query_priority 입니다.
metric_name	character(256)	지표 이름
metric_operator	character(256)	지표 연산자. 가능한 값은 >, =, <입니다.
metric_value	double	지정한 지표에서 작업이 트리거되는 임계값
action_value	character(256)	action이 change_query_priority 인 경우 가능한 값은 highest, high, normal, low 및 lowest입니다. action이 log, hop 또는 abort인 경우 값은 비어 있습니다.

샘플 쿼리

5보다 큰(사용자 정의 대기열 포함) 모든 서비스 클래스의 QMR 규칙 정의를 보려면 다음 쿼리를 실행합니다. 서비스 클래스 ID의 목록은 WLM 서비스 클래스 ID 섹션을 참조하세요.

```
Select *
from stv_wlm_qmr_config
where service_class > 5
order by service_class;
```

STV_WLM_QUERY_QUEUE_STATE

서비스 클래스에 따른 쿼리 대기열의 현재 상태를 기록합니다.

STV_WLM_QUERY_QUEUE_STATE는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
service_class	정수	서비스 클래스의 ID. 서비스 클래스 ID의 목록은 WLM 서비스 클래스 ID 섹션을 참조하세요.
position	정수	대기열에서 쿼리의 위치. **position** 값이 가장 작은 쿼리가 다음에 실행됩니다.
작업	정수	워크로드 관리자를 통해 쿼리를 추적하는 데 사용되는 ID. 다수의 쿼리 ID와 연결되기도 합니다. 쿼리를 다시 시작하면 새로운 작업 ID가 아닌 새로운 쿼리 ID가 할당됩니다.
쿼리	정수	쿼리 ID. 쿼리를 다시 시작하면 새로운 작업 ID가 아닌 새로운 쿼리 ID가 할당됩니다.
slot_count	정수	WLM 쿼리 슬롯의 수.
start_time	타임스탬프	쿼리가 대기열에 진입한 시간
queue_time	bigint	쿼리가 대기열에서 대기한 시간(마이크로초)

샘플 쿼리

다음은 서비스 클래스가 4보다 큰 대기열의 쿼리를 나타내는 쿼리입니다.

```
select * from stv_wlm_query_queue_state
where service_class > 4
order by service_class;
```

위 쿼리는 다음과 같은 샘플 출력을 반환합니다.

```
 service_class | position | task | query | slot_count |        start_time          |
 queue_time
---------------+----------+------+-------+------------+----------------------------
+------------
             5 |        0 |  455 |   476 |          5 | 2010-10-06 13:18:24.065838 |
 20937257
             6 |        1 |  456 |   478 |          5 | 2010-10-06 13:18:26.652906 |
 18350191
(2 rows)
```

STV_WLM_QUERY_STATE

WLM에서 추적 중인 쿼리의 현재 상태를 기록합니다.

STV_WLM_QUERY_STATE는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
xid	정수	쿼리 또는 하위 쿼리의 트랜잭션 ID
작업	정수	워크로드 관리자를 통해 쿼리를 추적하는 데 사용되는 ID. 다수의 쿼리 ID와 연결되기도 합니다. 쿼리를 다시 시작하면 새로운 작업 ID가 아닌 새로운 쿼리 ID가 할당됩니다.

열 명칭	데이터 유형	설명
쿼리	정수	쿼리 ID. 쿼리를 다시 시작하면 새로운 작업 ID가 아닌 새로운 쿼리 ID가 할당됩니다.
service_class	정수	서비스 클래스의 ID. 서비스 클래스 ID의 목록은 WLM 서비스 클래스 ID 섹션을 참조하세요.
slot_count	정수	WLM 쿼리 슬롯의 수.
wlm_start_time	타임스탬프	쿼리가 시스템 테이블 대기열 또는 단기 쿼리 대기열에 진입한 시간

열 명칭	데이터 유형	설명
state	character(16)	쿼리 또한 하위 쿼리의 현재 상태. 가능한 값은 다음과 같습니다. • Classified - 쿼리가 서비스 클래스에 할당되었습니다. • Completed - 쿼리 실행이 완료되었습니다. 쿼리가 성공적으로 실행되었거나 취소되었습니다. 최종 상태는 STL_QUERY의 결과를 확인하세요. • Dequeued - 내부 전용입니다. • Evicted - 다시 시작하기 위해 서비스 클래스에서 쿼리가 제거되었습니다. • Evicting - 다시 시작하기 위해 서비스 클래스에서 쿼리가 제거되었습니다. • Initialized - 내부 전용입니다. • Invalid - 내부 전용입니다. • Queued - 쿼리를 실행할 수 있는 슬롯이 없기 때문에 쿼리가 쿼리 대기열로 전송되었습니다. • QueuedWaiting - 쿼리가 쿼리 대기열에서 대기 중입니다. • Rejected - 내부 전용입니다. • Returning - 쿼리가 클라이언트에 결과를 반환하고 있습니다. • Running - 쿼리가 실행 중입니다. • TaskAssigned - 내부 전용입니다.
queue_time	bigint	쿼리가 대기열에서 대기한 시간(마이크로초)
exec_time	bigint	쿼리가 실행된 시간(마이크로초)입니다.

열 명칭	데이터 유형	설명
query_priority	char(20)	쿼리의 우선순위입니다. 가능한 값은 n/a, lowest, low, normal, high 및 highest입니다. 여기에서 n/a은 쿼리 우선 순위가 지원되지 않음을 뜻합니다.

샘플 쿼리

다음 쿼리는 4보다 큰 서비스 클래스에서 현재 실행 중인 모든 쿼리를 표시합니다. 서비스 클래스 ID의 목록은 WLM 서비스 클래스 ID 섹션을 참조하세요.

```
select xid, query, trim(state) as state, queue_time, exec_time
from stv_wlm_query_state
where service_class > 4;
```

위 쿼리는 다음과 같은 샘플 출력을 반환합니다.

```
xid    | query | state   | queue_time | exec_time
-------+-------+---------+------------+-----------
100813 | 25942 | Running |          0 |    1369029
100074 | 25775 | Running |          0 | 2221589242
```

STV_WLM_QUERY_TASK_STATE

서비스 클래스 쿼리 작업의 현재 상태를 저장합니다.

STV_WLM_QUERY_TASK_STATE는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
service_class	정수	서비스 클래스의 ID. 서비스 클래스 ID의 목록은 WLM 서비스 클래스 ID 섹션을 참조하세요.

열 명칭	데이터 유형	설명
작업	정수	워크로드 관리자를 통해 쿼리를 추적하는 데 사용되는 ID. 다수의 쿼리 ID와 연결되기도 합니다. 쿼리를 다시 시작하면 새로운 작업 ID가 아닌 새로운 쿼리 ID가 할당됩니다.
쿼리	정수	쿼리 ID. 쿼리를 다시 시작하면 새로운 작업 ID가 아닌 새로운 쿼리 ID가 할당됩니다.
slot_count	정수	WLM 쿼리 슬롯의 수.
start_time	타임스탬프	쿼리 실행이 시작된 시간
exec_time	bigint	지금까지 쿼리가 실행된 시간(마이크로초)

샘플 쿼리

다음 쿼리는 4보다 큰 서비스 클래스의 현재 쿼리 상태를 표시합니다. 서비스 클래스 ID의 목록은 WLM 서비스 클래스 ID 섹션을 참조하세요.

```
select * from stv_wlm_query_task_state
where service_class > 4;
```

위 쿼리는 다음과 같은 샘플 출력을 반환합니다.

```
service_class | task | query |         start_time         | exec_time
--------------+------+-------+----------------------------+-----------
    5         |  466 |   491 | 2010-10-06 13:29:23.063787 | 357618748
(1 row)
```

STV_WLM_SERVICE_CLASS_CONFIG

WLM의 서비스 클래스 구성을 기록합니다.

STV_WLM_SERVICE_CLASS_CONFIG는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
service_class	정수	서비스 클래스의 ID. 서비스 클래스 ID의 목록은 WLM 서비스 클래스 ID 섹션을 참조하세요.
queueing_strategy	character(32)	시스템에서 사용하도록 예약됩니다.
num_query_tasks	정수	현재 서비스 클래스의 실제 동시성 레벨. num_query_tasks 와 target_num_query_tasks 가 다르면 동적 WLM 전환이 진행 중입니다. -1 값은 Auto WLM(자동 WLM)이 구성되었음을 나타냅니다.
target_num_query_tasks	정수	가장 최근 WLM 구성 변경 시 설정된 동시성 레벨
evictable	character(8)	시스템에서 사용하도록 예약됩니다.
eviction_threshold	bigint	시스템에서 사용하도록 예약됩니다.
query_working_mem	정수	현재 서비스 클래스에 할당된 노드당 유효 메모리의 실제 크기(슬롯당 MB). query_working_mem 와 target_query_working_mem 가 다르면 동적 WLM 전환이 진행 중입니다. -1 값은 Auto WLM(자동 WLM)이 구성되었음을 나타냅니다.
target_query_working_mem	정수	가장 최근 WLM 구성 변경 시 설정된 노드당 유효 메모리의 크기(슬롯당 MB)
min_step_mem	정수	시스템에서 사용하도록 예약됩니다.
name	character(64)	서비스 클래스의 이름입니다.
max_execution_time	bigint	쿼리를 종료하기 전에 실행할 수 있는 시간(밀리초)
user_group_wild_card	불	TRUE인 경우, WLM 대기열은 WLM 구성 시 사용자 그룹 문자열에서 별표(*)를 와일드카드 문자로 처리합니다.

열 명칭	데이터 유형	설명
query_group_wild_card	불	TRUE인 경우, WLM 대기열은 WLM 구성 시 쿼리 그룹 문자열에서 별표(*)를 와일드카드 문자로 처리합니다.
concurrency_scaling	character(20)	동시성 조정이 on인지 아니면 off인지 설명합니다.
query_priority	character(20)	쿼리 우선순위 값입니다.
user_role_wild_card	불	TRUE인 경우, WLM 대기열은 WLM 구성 시 사용자 문자열에서 별표(*)를 와일드카드 문자로 처리합니다.

샘플 쿼리

첫 번째 사용자 정의 서비스 클래스는 서비스 클래스 6이고, 이름은 Service class #1로 지정됩니다. 다음 쿼리는 4보다 큰 서비스 클래스의 현재 구성을 표시합니다. 서비스 클래스 ID의 목록은 WLM 서비스 클래스 ID 섹션을 참조하세요.

```
select rtrim(name) as name,
num_query_tasks as slots,
query_working_mem as mem,
max_execution_time as max_time,
user_group_wild_card as user_wildcard,
query_group_wild_card as query_wildcard
from stv_wlm_service_class_config
where service_class > 4;

name                          | slots | mem | max_time | user_wildcard | query_wildcard
------------------------------+-------+-----+----------+---------------+---------------
Service class for super user |     1 | 535 |        0 | false         | false
Queue 1                       |     5 | 125 |        0 | false         | false
Queue 2                       |     5 | 125 |        0 | false         | false
Queue 3                       |     5 | 125 |        0 | false         | false
Queue 4                       |     5 | 627 |        0 | false         | false
Queue 5                       |     5 | 125 |        0 | true          | true
Default queue                 |     5 | 125 |        0 | false         | false
```

다음은 동적 WLM 전환 상태를 표시하는 쿼리입니다. 전환이 진행되는 동안에는 num_query_tasks와 target_query_working_mem은 목표 값과 똑같아질 때까지 업데이트됩니다. 자세한 내용은 WLM 동적 및 정적 구성 속성 단원을 참조하십시오.

```
select rtrim(name) as name,
num_query_tasks as slots,
target_num_query_tasks as target_slots,
query_working_mem as memory,
target_query_working_mem as target_memory
from stv_wlm_service_class_config
where num_query_tasks > target_num_query_tasks
or query_working_mem > target_query_working_mem
and service_class > 5;

 name             | slots | target_slots | memory | target_mem
------------------+-------+--------------+--------+------------
 Queue 3          |     5 |           15 |    125 |        375
 Queue 5          |    10 |            5 |    250 |        125
 (2 rows)
```

STV_WLM_SERVICE_CLASS_STATE

서비스 클래스의 현재 상태를 저장합니다.

STV_WLM_SERVICE_CLASS_STATE는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
service_class	정수	서비스 클래스의 ID. 서비스 클래스 ID의 목록은 WLM 서비스 클래스 ID 섹션을 참조하세요.
num_queued_queries	정수	현재 대기열에서 대기 중인 쿼리의 수
num_executing_queries	정수	현재 실행 중인 쿼리의 수

열 명칭	데이터 유형	설명
num_serviced_queries	정수	해당 서비스 클래스에 속했던 쿼리의 수
num_executed_queries	정수	Amazon Redshift 다시 시작 후 실행된 쿼리의 수.
num_evicted_queries	정수	Amazon Redshift 다시 시작 후 제거된 쿼리의 수. 제거된 쿼리에 대한 일부 이유에는 WLM 제한 시간, QMR 건너뛰기 작업 및 동시성 확장 클러스터의 쿼리 실패가 포함됩니다.
num_concurrency_scaling_queries	정수	Amazon Redshift 다시 시작 후 동시성 조정 클러스터에서 실행된 쿼리의 수.

샘플 쿼리

다음 쿼리는 5보다 큰 서비스 클래스의 상태를 표시합니다. 서비스 클래스 ID의 목록은 WLM 서비스 클래스 ID 섹션을 참조하세요.

```
select service_class, num_executing_queries,
num_executed_queries
from stv_wlm_service_class_state
where service_class > 5
order by service_class;
```

```
 service_class | num_executing_queries | num_executed_queries
---------------+-----------------------+----------------------
             6 |                     1 |                  222
             7 |                     0 |                  135
             8 |                     1 |                   39
(3 rows)
```

STV_XRESTORE_ALTER_QUEUE_STATE

클래식 크기 조정 중에 각 테이블의 마이그레이션 진행 상황을 모니터링하려면 STV_XRESTORE_ALTER_QUEUE_STATE를 사용합니다. 이는 구체적으로 대상 노드 유형이 RA3일

때 적용됩니다. RA3 노드로의 클래식 크기 조정에 대한 자세한 내용은 클래식 크기 조정에서 확인하세요.

STV_XRESTORE_ALTER_QUEUE_STATE는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_RESTORE_STATE에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	크기 조정을 시작한 사용자의 ID입니다.
db_id	정수	데이터베이스의 ID입니다.
스키마	char(128)	스키마의 이름입니다.
table_name	char(128)	테이블의 이름
tbl	정수	테이블의 ID입니다.
status	char(64)	테이블의 마이그레이션 진행 상태입니다. 가능한 값은 다음과 같습니다. • `Waiting`: 재배포 시작 대기 중 • `Applying`: 현재 재배포 중 • `Finished`: 재배포 완료됨
task_type	정수	테이블의 재배포 유형입니다. 가능한 값은 다음과 같습니다. • 1: KEY • 2: EVEN 배포 스타일에 대한 자세한 내용은 분산 스타일 섹션을 참조하세요.

샘플 쿼리

다음 쿼리는 데이터베이스에서 크기 조정 대기 중인 테이블, 현재 크기 조정 중인 테이블, 크기 조정이 완료된 테이블 수를 보여줍니다.

```
select db_id, status, count(*)
from stv_xrestore_alter_queue_state
group by 1,2 order by 3 desc

db_id  |   status    | count
-------+-------------+------
694325 | Waiting     |   323
694325 | Finished    |    60
694325 | Applying    |     1
```

기본 및 동시성 조정 클러스터에 대한 SVCS 뷰

접두사 SVCS를 포함하는 SVCS 시스템 뷰는 동시성 조정 클러스터와 기본 클러스터 모두의 쿼리에 대한 세부 정보를 제공합니다. 이 뷰는 접두사 STL을 포함하는 테이블과 유사합니다. 단, STL 테이블은 기본 클러스터에서 실행된 쿼리에 대한 정보만 제공합니다.

주제

- SVCS_ALERT_EVENT_LOG
- SVCS_COMPILE
- SVCS_CONCURRENCY_SCALING_USAGE
- SVCS_EXPLAIN
- SVCS_PLAN_INFO
- SVCS_QUERY_SUMMARY
- SVCS_S3LIST
- SVCS_S3LOG
- SVCS_S3PARTITION_SUMMARY
- SVCS_S3QUERY_SUMMARY
- SVCS_STREAM_SEGS
- SVCS_UNLOAD_LOG

SVCS_ALERT_EVENT_LOG

쿼리 옵티마이저에서 성능 문제를 야기할 수 있는 조건이 식별되면 알림 메시지가 기록됩니다. 이 뷰는 STL_ALERT_EVENT_LOG 시스템 테이블에서 파생되지만 동시성 확장 클러스터에서 실행된 쿼리에 대한 조각 수준을 표시하지 않습니다. SVCS_ALERT_EVENT_LOG 테이블은 쿼리 성능을 높일 수 있는 방법을 찾는 데 사용됩니다.

하나의 쿼리는 여러 세그먼트로 구성되며, 각각의 세그먼트는 하나 이상의 단계로 구성됩니다. 자세한 내용은 쿼리 처리 단원을 참조하십시오.

> **Note**
>
> 접두사 SVCS를 포함하는 시스템 뷰는 동시성 확장 클러스터와 기본 클러스터 모두의 쿼리에 대한 세부 정보를 제공합니다. 이 뷰는 접두사 STL을 포함하는 테이블과 유사합니다. 단, STL 테이블은 기본 클러스터에서 실행된 쿼리에 대한 정보만 제공합니다.

SVCS_ALERT_EVENT_LOG는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호
step	정수	실행된 쿼리 단계입니다.
pid	정수	쿼리 문 및 조각과 연결된 프로세스 ID. 동일한 쿼리가 다수의 조각에서 실행되는 경우에는 다수의 PID를 가질 수 있습니다.
xid	bigint	문에 연결된 트랜잭션 ID.

열 명칭	데이터 유형	설명
이벤트	character (1024)	알림 이벤트에 대한 설명
solution	character (1024)	권장 솔루션
event_time	타임스탬프	쿼리 시작 시간(UTC) 총 시간에는 대기 및 실행이 포함되며 소수점 이하 자릿수는 6자리입니다. 예: **2009-06-12 11:29:19.131358** .

사용 노트

SVCS_ALERT_EVENT_LOG는 쿼리에서 잠재적 문제를 식별한 후 쿼리 성능 튜닝의 모범 사례에 따라 데이터베이스 설계를 최적화하여 쿼리를 재작성하는 데 사용할 수 있습니다. SVCS_ALERT_EVENT_LOG는 다음과 같은 알림을 기록합니다.

- 통계 누락

  통계가 누락된 경우에는 데이터 로드 또는 중요한 업데이트에 이어 ANALYZE를 실행한 다음 COPY 작업에 STATUPDATE를 사용하십시오. 자세한 내용은 Amazon Redshift 쿼리 설계 모범 사례 단원을 참조하십시오.
- 중첩 루프

  중첩 루프는 일반적으로 데카르트 곱입니다. 이때는 쿼리를 평가하여 참여하는 테이블이 모두 효율적으로 조인되었는지 확인하십시오.
- 선택의 폭이 제한적인 필터

  반환되는 행과 스캔되는 행의 비율은 0.05미만입니다. 스캔되는 행은 `rows_pre_user_filter` 값이고, 반환되는 행은 STL_SCAN 시스템 테이블의 rows 값입니다. 이 알림은 쿼리가 결과 집합을 확인하기 위해 비정상적으로 많은 수의 행을 스캔하고 있다는 것을 의미합니다. 이러한 문제는 정렬 키가 누락되었거나 잘못되었을 때 발생할 수 있습니다. 자세한 내용은 정렬 키 단원을 참조하십시오.
- 지나치게 많은 고스트 행

  스캔 작업이 정리가 아닌 삭제된 것으로 표시되었거나, 혹은 커밋되지 않고 삽입된 비교적 다수의 행을 건너뛰었습니다. 자세한 내용은 테이블 Vacuum 단원을 참조하십시오.
- 다수의 행 분산

100만 개가 넘는 행이 해시 조인 또는 집계를 위해 재분산되었습니다. 자세한 내용은 쿼리 최적화를 위한 데이터 배포 단원을 참조하십시오.

- 다수의 행 브로드캐스팅

  100만 개가 넘는 행이 해시 조인을 위해 브로드캐스팅되었습니다. 자세한 내용은 쿼리 최적화를 위한 데이터 배포 단원을 참조하십시오.

- 직렬 실행

  쿼리 계획에서 DS_DIST_ALL_INNER 재분산 스타일이 지정되면서 내부 테이블 전체가 단일 조각으로 재분산되었기 때문에 직렬 실행을 피할 수 없습니다. 자세한 내용은 쿼리 최적화를 위한 데이터 배포 단원을 참조하십시오.

샘플 쿼리

다음은 4개 쿼리의 알림 이벤트를 나타낸 쿼리입니다.

```
SELECT query, substring(event,0,25) as event,
substring(solution,0,25) as solution,
trim(event_time) as event_time from svcs_alert_event_log order by query;

 query |             event             |          solution            |     event_time

-------+-------------------------------+------------------------------
+---------------------
  6567 | Missing query planner statist | Run the ANALYZE command      | 2014-01-03
 18:20:58
  7450 | Scanned a large number of del | Run the VACUUM command to rec| 2014-01-03
 21:19:31
  8406 | Nested Loop Join in the query | Review the join predicates to| 2014-01-04
 00:34:22
 29512 | Very selective query filter:r | Review the choice of sort key| 2014-01-06
 22:00:00

(4 rows)
```

SVCS_COMPILE

확장 클러스터에서 실행된 쿼리 또는 기본 클러스터에서 실행된 쿼리를 포함한 쿼리의 각 쿼리 세그먼트에 대한 컴파일 시간 및 위치를 기록합니다.

> **Note**
>
> 접두사 SVCS를 포함하는 시스템 뷰는 동시성 확장 클러스터와 기본 클러스터 모두의 쿼리에 대한 세부 정보를 제공합니다. 이 뷰는 접두사 SVL을 포함하는 뷰와 유사합니다. 단, SVL 뷰는 기본 클러스터에서 실행된 쿼리에 대한 정보만 제공합니다.

SVCS_COMPILE은 모든 사용자가 볼 수 있습니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

SCL_COMPILE에 대한 자세한 내용은 SVL_COMPILE 섹션을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID입니다.
xid	bigint	문과 연결된 트랜잭션 ID
pid	정수	문과 연결된 프로세스 ID
쿼리	정수	쿼리 ID입니다. 이 ID를 사용하여 다양한 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
segment	정수	컴파일할 쿼리 세그먼트.
locus	정수	세그먼트가 실행되는 위치이며, 컴퓨팅 노드에 있는 경우, **1**이고, 리더 노드에 있는 경우, **2**입니다.
starttime	타임스탬프	컴파일이 시작된 협정 세계시(UTC) 시간.
endtime	타임스탬프	컴파일이 끝난 UTC 시간.
컴파일	정수	컴파일이 재사용된 경우 값이 **0**이고, 세그먼트가 컴파일된 경우 **1**입니다.

샘플 쿼리

이 예에서는 쿼리 35878과 35879가 동일한 SQL 문을 실행했습니다. 쿼리 35878의 컴파일 열은 4개의 쿼리 세그먼트에 대해 1을 보여 주며, 이는 이 세그먼트들이 컴파일되었음을 나타냅니다. 쿼리 35879는 모든 세그먼트의 컴파일 열에서 0을 보여 주며, 세그먼트를 다시 컴파일할 필요가 없었음을 나타냅니다.

```
select userid, xid,  pid, query, segment, locus,
datediff(ms, starttime, endtime) as duration, compile
from svcs_compile
where query = 35878 or query = 35879
order by query, segment;

 userid |  xid   |  pid  | query | segment | locus | duration | compile
--------+--------+-------+-------+---------+-------+----------+---------
    100 | 112780 | 23028 | 35878 |       0 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       1 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       2 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       3 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       4 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       5 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       6 |     1 |     1380 |       1
    100 | 112780 | 23028 | 35878 |       7 |     1 |     1085 |       1
    100 | 112780 | 23028 | 35878 |       8 |     1 |     1197 |       1
    100 | 112780 | 23028 | 35878 |       9 |     2 |      905 |       1
    100 | 112782 | 23028 | 35879 |       0 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       1 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       2 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       3 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       4 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       5 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       6 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       7 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       8 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       9 |     2 |        0 |       0
(20 rows)
```

SVCS_CONCURRENCY_SCALING_USAGE

동시성 확장에 대한 사용 기간을 기록합니다. 각 사용 기간은 개별 동시성 확장 클러스터가 능동적으로 쿼리를 처리 중인 연속 기간입니다.

SVCS_CONCURRENCY_SCALING_USAGE 이 테이블은 슈퍼 사용자에게 표시됩니다. 데이터베이스의 슈퍼 사용자는 이 뷰를 모든 사용자에게 공개하도록 선택할 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
start_time	시간대 미포함 TIMESTAMP	사용 기간이 시작할 때
end_time	시간대 미포함 TIMESTAMP	사용 기간이 끝날 때
쿼리	bigint	이 사용 기간 동안 실행한 쿼리 수
usage_in_seconds	numeric(27,0)	이 사용 기간의 총 시간(초)

샘플 쿼리

특정 기간 동안의 사용 기간(초)을 보려면 다음 쿼리를 입력합니다.

```
select * from svcs_concurrency_scaling_usage order by start_time;

start_time | end_time | queries | usage_in_seconds
---------------------------+---------------------------+---------+-----------------
2019-02-14 18:43:53.01063 | 2019-02-14 19:16:49.781649 | 48 | 1977
```

SVCS_EXPLAIN

실행을 목적으로 제출된 쿼리의 EXPLAIN 계획을 표시합니다.

> **Note**
>
> 접두사 SVCS를 포함하는 시스템 뷰는 동시성 확장 클러스터와 기본 클러스터 모두의 쿼리에 대한 세부 정보를 제공합니다. 이 뷰는 접두사 STL을 포함하는 테이블과 유사합니다. 단, STL 테이블은 기본 클러스터에서 실행된 쿼리에 대한 정보만 제공합니다.

SVCS_EXPLAIN은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
nodeid	정수	계획 노드 식별자. 쿼리 실행 시 이 식별자를 통해 노드가 1개 이상의 단계로 매핑됩니다.
parentid	정수	상위 노드를 나타내는 계획 노드 식별자. 상위 노드 1개에는 하위 노드가 다수 종속됩니다. 예를 들어 병합 조인은 조인 테이블에서 스캔에 대한 상위 노드입니다.
plannode	character(400)	EXPLAIN 출력의 노드 텍스트. 컴퓨팅 노드에서 실행을 의미하는 계획 노드에는 EXPLAIN 출력 시 **XN** 접두사가 첨부됩니다.
정보	character(400)	계획 노드의 한정자 및 필터 정보. 예를 들어 조인 조건과 WHERE 절 제한이 이 열에 포함됩니다.

샘플 쿼리

집계 조인 쿼리의 EXPLAIN 출력이 다음과 같다고 가정하겠습니다.

```
explain select avg(datediff(day, listtime, saletime)) as avgwait
```

```
from sales, listing where sales.listid = listing.listid;
                                  QUERY PLAN

------------------------------------------------------------------------------
 XN Aggregate  (cost=6350.30..6350.31 rows=1 width=16)
  ->  XN Hash Join DS_DIST_NONE  (cost=47.08..6340.89 rows=3766 width=16)
        Hash Cond: ("outer".listid = "inner".listid)
        -> XN Seq Scan on listing  (cost=0.00..1924.97 rows=192497 width=12)
        -> XN Hash  (cost=37.66..37.66 rows=3766 width=12)
              -> XN Seq Scan on sales  (cost=0.00..37.66 rows=3766 width=12)
(6 rows)
```

이 쿼리를 실행하고, 쿼리 ID가 10이라면 SVCS_EXPLAIN 테이블을 사용하여 EXPLAIN 명령이 반환하는 것과 동일한 유형의 정보를 볼 수 있습니다.

```
select query,nodeid,parentid,substring(plannode from 1 for 30),
substring(info from 1 for 20) from svcs_explain
where query=10 order by 1,2;

query| nodeid |parentid|           substring            |    substring
-----+--------+--------+--------------------------------+-------------------
10   |      1 |      0 |XN Aggregate  (cost=6717.61..6  |
10   |      2 |      1 |  -> XN Merge Join DS_DIST_NO| Merge Cond:("outer"
10   |      3 |      2 |       -> XN Seq Scan on lis |
10   |      4 |      2 |       -> XN Seq Scan on sal |
(4 rows)
```

다음과 같은 쿼리를 가정합니다.

```
select event.eventid, sum(pricepaid)
from event, sales
where event.eventid=sales.eventid
group by event.eventid order by 2 desc;

eventid |   sum
--------+----------
    289 | 51846.00
   7895 | 51049.00
   1602 | 50301.00
    851 | 49956.00
   7315 | 49823.00
...
```

이 쿼리의 ID가 15라면 다음과 같은 시스템 테이블 쿼리에서 이전에 수행한 계획 노드가 반환됩니다. 이때 노드 순서가 반전되면서 실제 실행 순서를 표시합니다.

```
select query,nodeid,parentid,substring(plannode from 1 for 56)
from svcs_explain where query=15 order by 1, 2 desc;

query|nodeid|parentid|                              substring
-----+------+--------+-----------------------------------------------------------
15   |    8 |      7 |                                    -> XN Seq Scan on eve
15   |    7 |      5 |                              -> XN Hash(cost=87.98..87.9
15   |    6 |      5 |                              -> XN Seq Scan on sales(cos
15   |    5 |      4 |                         -> XN Hash Join DS_DIST_OUTER(cos
15   |    4 |      3 |                   -> XN HashAggregate(cost=862286577.07..
15   |    3 |      2 |           -> XN Sort(cost=1000862287175.47..10008622871
15   |    2 |      1 |  -> XN Network(cost=1000862287175.47..1000862287197.
15   |    1 |      0 |XN Merge(cost=1000862287175.47..1000862287197.46 rows=87
(8 rows)
```

다음은 창 함수가 포함된 모든 쿼리 계획의 쿼리 ID를 가져오는 쿼리입니다.

```
select query, trim(plannode) from svcs_explain
where plannode like '%Window%';

query|                                       btrim
-----+-------------------------------------------------------------------------
26   | -> XN Window(cost=1000985348268.57..1000985351256.98 rows=170 width=33)
27   | -> XN Window(cost=1000985348268.57..1000985351256.98 rows=170 width=33)
(2 rows)
```

SVCS_PLAN_INFO

SVCS_PLAN_INFO 테이블은 쿼리의 EXPLAIN 출력을 행 집합으로 표시하는 데 사용됩니다. 이 테이블은 쿼리 계획을 살펴볼 수 있는 대체 방법이기도 합니다.

> **Note**
>
> 접두사 SVCS를 포함하는 시스템 뷰는 동시성 확장 클러스터와 기본 클러스터 모두의 쿼리에 대한 세부 정보를 제공합니다. 이 뷰는 접두사 STL을 포함하는 테이블과 유사합니다. 단, STL 테이블은 기본 클러스터에서 실행된 쿼리에 대한 정보만 제공합니다.

SVCS_PLAN_INFO는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
nodeid	정수	계획 노드 식별자. 쿼리 실행 시 이 식별자를 통해 노드가 1개 이상의 단계로 매핑됩니다.
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호
step	정수	쿼리 단계를 식별할 수 있는 번호
locus	정수	단계가 실행되는 위치로 컴퓨터 노드이면 0이고, 리더 노드이면 1입니다.
plannode	정수	계획 노드의 열거 값. plannode 열의 열거형 값은 다음 테이블을 참조하십시오. (SVCS_EXPLAIN 테이블의 PLANNODE 열에는 계획 노드 텍스트가 포함됩니다)
startupcost	double precision	이 단계의 첫 번째 행을 반환하는 상대적 예상 비용
totalcost	double precision	단계를 실행하는 상대적 예상 비용
rows	bigint	단계에서 산출될 것으로 예상되는 행의 수
bytes	bigint	단계에서 산출될 것으로 예상되는 바이트 수

샘플 쿼리

다음은 단순한 SELECT 쿼리에서 EXPLAIN 명령을 사용하여 반환되는 쿼리 계획과 SVCS_PLAN_INFO 테이블에 대한 쿼리를 실행하여 반환되는 쿼리 계획을 비교하는 예입니다.

```
explain select * from category;
QUERY PLAN
--------------------------------------------------------------
XN Seq Scan on category (cost=0.00..0.11 rows=11 width=49)
(1 row)

select * from category;
catid | catgroup | catname | catdesc
-------+----------+-----------+--------------------------------------------
1 | Sports | MLB | Major League Baseball
3 | Sports | NFL | National Football League
5 | Sports | MLS | Major League Soccer
...

select * from svcs_plan_info where query=256;

query | nodeid | segment | step | locus | plannode | startupcost | totalcost
| rows | bytes
-------+--------+---------+------+-------+----------+-------------+-----------+------
+-------
256 | 1 | 0 | 1 | 0 | 104 | 0 | 0.11 | 11 | 539
256 | 1 | 0 | 0 | 0 | 104 | 0 | 0.11 | 11 | 539
(2 rows)
```

이번 예에서 PLANNODE 104는 CATEGORY 테이블에 대한 순차적 스캔을 의미합니다.

```
select distinct eventname from event order by 1;

eventname
----------------------------------------------------------------------------
.38 Special
3 Doors Down
70s Soul Jam
A Bronx Tale
...

explain select distinct eventname from event order by 1;

QUERY PLAN
------------------------------------------------------------------------------------------
XN Merge (cost=1000000000136.38..1000000000137.82 rows=576 width=17)
Merge Key: eventname
-> XN Network (cost=1000000000136.38..1000000000137.82 rows=576
```

```
width=17)
Send to leader
-> XN Sort (cost=1000000000136.38..1000000000137.82 rows=576
width=17)
Sort Key: eventname
-> XN Unique (cost=0.00..109.98 rows=576 width=17)
-> XN Seq Scan on event (cost=0.00..87.98 rows=8798
width=17)
(8 rows)

select * from svcs_plan_info where query=240 order by nodeid desc;

query | nodeid | segment | step | locus | plannode | startupcost |
totalcost | rows | bytes
-------+--------+---------+------+-------+----------+-----------------
+------------------+------+--------
240 | 5 | 0 | 0 | 0 | 104 | 0                | 87.98   | 8798 | 149566
240 | 5 | 0 | 1 | 0 | 104 | 0                | 87.98   | 8798 | 149566
240 | 4 | 0 | 2 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 4 | 0 | 3 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 4 | 1 | 0 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 4 | 1 | 1 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 3 | 1 | 2 | 0 | 114 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 3 | 2 | 0 | 0 | 114 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 2 | 2 | 1 | 0 | 123 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 1 | 3 | 0 | 0 | 122 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
(10 rows)
```

SVCS_QUERY_SUMMARY

SVCS_QUERY_SUMMARY 뷰를 사용하여 쿼리 실행에 대한 일반 정보를 찾습니다.

SVCS_QUERY_SUMMARY의 정보는 모든 노드에서 집계됩니다.

> **Note**
>
> SVCS_QUERY_SUMMARY 보기에는 Amazon Redshift가 완료하는 쿼리에 대한 정보만 포함되며 다른 유틸리티 및 DDL 명령이 실행하는 쿼리의 정보는 포함되지 않습니다. DDL 및 유틸리티 명령을 포함하여 Amazon Redshift에 의해 완료되는 모든 문의 전체 목록과 정보를 보려면 SVL_STATEMENTTEXT 보기를 쿼리할 수 있습니다.

접두사 SVCS를 포함하는 시스템 뷰는 동시성 확장 클러스터와 기본 클러스터 모두의 쿼리에 대한 세부 정보를 제공합니다. 이 뷰는 접두사 SVL을 포함하는 뷰와 유사합니다. 단, SVL 뷰는 기본 클러스터에서 실행된 쿼리에 대한 정보만 제공합니다.

SVCS_QUERY_SUMMARY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_DETAIL에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

SVL_QUERY_SUMMARY에 대한 자세한 내용은 SVL_QUERY_SUMMARY 섹션을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 다양한 다른 시스템 테이블 및 보기를 조인하는 데 사용할 수 있습니다.
stm	정수	스트림: 한 쿼리 내의 동시 세그먼트의 집합. 쿼리에는 하나 이상의 스트림이 있습니다.
seg	정수	세그먼트 번호. 하나의 쿼리는 여러 세그먼트로 구성되며, 각각의 세그먼트는 하나 이상의 단계로 구성됩니다. 쿼리 세그먼트는 병렬로 실행될 수 있습니다. 각 세그먼트는 단일 프로세스에서 실행됩니다.
step	정수	실행된 쿼리 단계입니다.
maxtime	bigint	단계가 실행되기 위한 최대 시간(마이크로초)입니다.
avgtime	bigint	단계가 실행되기 위한 평균 시간(마이크로초)입니다.
rows	bigint	쿼리 단계에 연관된 데이터 행의 수.
bytes	bigint	쿼리 단계에 연관된 데이터 바이트의 수.

열 명칭	데이터 유형	설명
rate_row	double precision	행당 쿼리 실행 속도.
rate_byte	double precision	바이트당 쿼리 실행 속도.
레이블	텍스트	쿼리 단계 이름 및 해당되는 경우, 테이블 ID와 테이블 이름으로 구성되는 단계 레이블(예: scan tbl=100448 name =user). 3자리 테이블 ID는 일반적으로 일시적 테이블의 스캔을 가리킵니다. `tbl=0`이 표시된다면 일반적으로 일정한 값의 스캔을 가리킵니다.
is_diskbased	character(1)	쿼리의 이 단계가 클러스터의 노드에서 디스크 기반 작업으로 수행되었는지 여부입니다(true(**t**) 또는 false(**f**)). 해시, 정렬, 집계 단계 같은 특정 단계만 디스크로 갈 수 있습니다. 많은 단계 형식은 항상 메모리에서 실행됩니다.
workmem	bigint	쿼리 단계에 할당된 작업 메모리의 양(바이트)입니다.
is_rrscan	character(1)	true(**t**)인 경우, 단계에서 범위 제한 스캔이 사용되었음을 나타냅니다. 기본값은 false(**f**)입니다.
is_delayed_scan	character(1)	true(**t**)인 경우, 단계에서 지연된 스캔이 사용되었음을 나타냅니다. 기본값은 false(**f**)입니다.
rows_pre_filter	bigint	영구 테이블 스캔의 경우, 삭제 대기 행(고스트 행)을 필터링하기 전에 내보낸 행의 총수.

샘플 쿼리

쿼리 단계의 처리 정보 확인

다음 쿼리는 쿼리 87의 각 단계에 대한 기본적 처리 정보를 보여 줍니다.

```
select query, stm, seg, step, rows, bytes
from svcs_query_summary
where query = 87
order by query, seg, step;
```

이 쿼리는 다음 샘플 출력에서 보듯 쿼리 87에 대한 처리 정보를 검색합니다.

```
 query | stm | seg | step |  rows  |  bytes
-------+-----+-----+------+--------+---------
87     |   0 |   0 |    0 |     90 |    1890
87     |   0 |   0 |    2 |     90 |     360
87     |   0 |   1 |    0 |     90 |     360
87     |   0 |   1 |    2 |     90 |    1440
87     |   1 |   2 |    0 | 210494 | 4209880
87     |   1 |   2 |    3 |  89500 |       0
87     |   1 |   2 |    6 |      4 |      96
87     |   2 |   3 |    0 |      4 |      96
87     |   2 |   3 |    1 |      4 |      96
87     |   2 |   4 |    0 |      4 |      96
87     |   2 |   4 |    1 |      1 |      24
87     |   3 |   5 |    0 |      1 |      24
87     |   3 |   5 |    4 |      0 |       0
(13 rows)
```

쿼리 단계가 디스크에 분산되었는지 여부 확인

다음 쿼리는 쿼리 ID 1025(쿼리의 쿼리 ID를 얻는 방법을 확인하려면 SVL_QLOG 뷰를 참조)인 쿼리의 단계가 디스크로 분산되었는지 또는 쿼리가 전적으로 메모리에서 실행되었는지 여부를 보여 줍니다.

```
select query, step, rows, workmem, label, is_diskbased
from svcs_query_summary
where query = 1025
order by workmem desc;
```

위 쿼리는 다음과 같은 샘플 출력을 반환합니다.

```
query| step|  rows  |  workmem   |  label         | is_diskbased
-----+-----+--------+------------+----------------+--------------
1025 |  0  |16000000|  141557760 |scan tbl=9      | f
1025 |  2  |16000000|  135266304 |hash tbl=142    | t
1025 |  0  |16000000|  128974848 |scan tbl=116536| f
1025 |  2  |16000000|  122683392 |dist            | f
(4 rows)
```

IS_DISKBASED의 값을 스캔하면 어떤 쿼리 단계가 디스크로 갔는지 알 수 있습니다. 쿼리 1025의 경우, 해시 단계가 디스크에서 실행되었습니다. 디스크에서 실행될 수 있는 단계에는 hash, aggr, sort

단계가 포함됩니다. 디스크 기반 쿼리 단계만 보려면 위의 예에서 **`and is_diskbased = 't'`** 절을 SQL 문에 추가합니다.

SVCS_S3LIST

세그먼트 수준에서 Amazon Redshift Spectrum 쿼리에 대한 세부 정보를 가져오려면 SVCS_S3LIST 뷰를 사용합니다. 한 개의 세그먼트로 한 개의 외부 테이블 스캔을 수행할 수 있습니다. 이 뷰는 SVL_S3LIST 시스템 뷰에서 파생되지만 동시성 확장 클러스터에서 실행된 쿼리에 대한 조각 수준을 표시하지 않습니다.

> **Note**
>
> 접두사 SVCS를 포함하는 시스템 뷰는 동시성 확장 클러스터와 기본 클러스터 모두의 쿼리에 대한 세부 정보를 제공합니다. 이 뷰는 접두사 SVL을 포함하는 뷰와 유사합니다. 단, SVL 뷰는 기본 클러스터에서 실행된 쿼리에 대한 정보만 제공합니다.

SVCS_S3LIST는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

SVL_S3LIST에 대한 자세한 내용은 SVL_S3LIST 섹션을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
쿼리	정수	쿼리 ID입니다.
segment	정수	세그먼트 번호. 쿼리는 여러 세그먼트로 구성됩니다.
노드	정수	노드 번호입니다.
eventtime	타임스탬프	이벤트가 기록된 시간(UTC)입니다.
버킷	char(256)	Amazon S3 버킷 이름입니다.
접두사	char(256)	Amazon S3 버킷 위치의 접두사입니다.
recursive	char(1)	하위 폴더에 대한 반복 스캔 여부입니다.

열 명칭	데이터 유형	설명
retrieved_files	정수	나열된 파일 수입니다.
max_file_size	bigint	나열된 파일의 최대 파일 크기입니다.
avg_file_size	double precision	나열된 파일의 평균 파일 크기입니다.
generated_splits	정수	파일 분할의 수입니다.
avg_split_length	double precision	파일 분할의 평균 길이(바이트)입니다.
duration	bigint	파일 목록의 기간(마이크로초)입니다.

샘플 쿼리

다음 예는 마지막으로 수행된 쿼리에 대해 SVCS_S3LIST를 쿼리합니다.

```
select *
from svcs_s3list
where query = pg_last_query_id()
order by query,segment;
```

SVCS_S3LOG

세그먼트 수준에서 Redshift Spectrum 쿼리에 대한 문제 해결 세부 정보를 가져오려면 SVCS_S3LOG 뷰를 사용합니다. 한 개의 세그먼트로 한 개의 외부 테이블 스캔을 수행할 수 있습니다. 이 뷰는 SVL_S3LOG 시스템 뷰에서 파생되지만 동시성 확장 클러스터에서 실행된 쿼리에 대한 조각 수준을 표시하지 않습니다.

> **Note**
>
> 접두사 SVCS를 포함하는 시스템 뷰는 동시성 확장 클러스터와 기본 클러스터 모두의 쿼리에 대한 세부 정보를 제공합니다. 이 뷰는 접두사 SVL을 포함하는 뷰와 유사합니다. 단, SVL 뷰는 기본 클러스터에서 실행된 쿼리에 대한 정보만 제공합니다.

SVCS_S3LOG는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

SVL_S3LOG에 대한 자세한 내용은 SVL_S3LOG 섹션을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
pid	정수	프로세스 ID.
쿼리	정수	쿼리 ID입니다.
segment	정수	세그먼트 번호. 하나의 쿼리는 여러 세그먼트로 구성되며, 각각의 세그먼트는 하나 이상의 단계로 구성됩니다.
step	정수	실행된 쿼리 단계입니다.
노드	정수	노드 번호입니다.
eventtime	타임스탬프	이벤트가 기록된 시간(UTC)입니다.
message	char(512)	로그 항목에 대한 메시지입니다.

샘플 쿼리

다음 예는 마지막으로 실행된 쿼리에 대해 SVCS_S3LOG를 쿼리합니다.

```
select *
from svcs_s3log
where query = pg_last_query_id()
order by query,segment;
```

SVCS_S3PARTITION_SUMMARY

세그먼트 수준에서 Redshift Spectrum 쿼리 파티션 처리에 대한 요약을 가져오려면 SVCS_S3PARTITION_SUMMARY 뷰를 사용합니다. 한 개의 세그먼트로 한 개의 외부 테이블 스캔을 수행할 수 있습니다.

> **Note**
>
> 접두사 SVCS를 포함하는 시스템 뷰는 동시성 확장 클러스터와 기본 클러스터 모두의 쿼리에 대한 세부 정보를 제공합니다. 이 뷰는 접두사 SVL을 포함하는 뷰와 유사합니다. 단, SVL 뷰는 기본 클러스터에서 실행된 쿼리에 대한 정보만 제공합니다.

SVCS_S3PARTITION_SUMMARY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

SVL_S3PARTITION에 대한 자세한 내용은 SVL_S3PARTITION 섹션을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
쿼리	정수	쿼리 ID입니다. 이 값을 사용하여 다양한 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
segment	정수	세그먼트 번호. 쿼리는 여러 세그먼트로 구성됩니다.
assignment	char(1)	노드를 통한 파티션 할당의 유형입니다.
min_starttime	타임스탬프	파티션 처리가 시작된 시간(UTC)입니다.
max_endtime	타임스탬프	파티션 처리가 완료된 시간(UTC)입니다.
min_duration	bigint	이 쿼리에 노드가 사용하는 최소 파티션 처리 시간(마이크로초)입니다.

열 명칭	데이터 유형	설명
max_duration	bigint	이 쿼리에 노드가 사용하는 최대 파티션 처리 시간(마이크로초)입니다.
avg_duration	bigint	이 쿼리에 노드가 사용하는 평균 파티션 처리 시간(마이크로초)입니다.
total_partitions	정수	외부 테이블의 총 파티션 수입니다.
qualified_partitions	정수	적격 파티션의 총 수입니다.
min_assigned_partitions	정수	한 개의 노드에 할당된 최소 파티션 수입니다.
max_assigned_partitions	정수	한 개의 노드에 할당된 최대 파티션 수입니다.
avg_assigned_partitions	bigint	한 개의 노드에 할당된 평균 파티션 수입니다.

샘플 쿼리

다음 예는 마지막으로 수행된 쿼리에 대한 파티션 스캔 세부 정보를 가져옵니다.

```
select query, segment, assignment, min_starttime, max_endtime, min_duration,
 avg_duration
from svcs_s3partition_summary
where query = pg_last_query_id()
order by query,segment;
```

SVCS_S3QUERY_SUMMARY

시스템에서 실행된 모든 Redshift Spectrum 쿼리(S3 쿼리)의 요약을 가져오려면 SVCS_S3QUERY_SUMMARY 뷰를 사용합니다. 한 개의 세그먼트로 한 개의 외부 테이블 스캔을 수행할 수 있습니다.

> **Note**
>
> 접두사 SVCS를 포함하는 시스템 뷰는 동시성 확장 클러스터와 기본 클러스터 모두의 쿼리에 대한 세부 정보를 제공합니다. 이 뷰는 접두사 SVL을 포함하는 뷰와 유사합니다. 단, SVL 뷰는 기본 클러스터에서 실행된 쿼리에 대한 정보만 제공합니다.

SVCS_S3QUERY_SUMMARY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

SVL_S3QUERY에 대한 자세한 내용은 SVL_S3QUERY 섹션을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	지정된 항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID입니다. 이 값을 사용하여 다양한 다른 시스템 테이블 및 뷰를 조인할 수 있습니다.
xid	bigint	트랜잭션 ID.
pid	정수	프로세스 ID.
segment	정수	세그먼트 번호. 하나의 쿼리는 여러 세그먼트로 구성되며, 각각의 세그먼트는 하나 이상의 단계로 구성됩니다.
step	정수	실행된 쿼리 단계입니다.
starttime	타임스탬프	이 세그먼트의 Redshift Spectrum 쿼리가 실행되기 시작한 UTC 시간입니다. 한 개의 세그먼트로 한 개의 외부 테이블 스캔을 수행할 수 있습니다.

열 명칭	데이터 유형	설명
endtime	타임스탬프	이 세그먼트의 Redshift Spectrum 쿼리가 완료된 UTC 시간입니다. 한 개의 세그먼트로 한 개의 외부 테이블 스캔을 수행할 수 있습니다.
elapsed	정수	이 세그먼트의 Redshift Spectrum 쿼리가 실행되는 데 걸린 시간(마이크로초)입니다.
aborted	정수	쿼리가 시스템에 의해 중지되거나 사용자에 의해 취소되는 경우, 이 열에 **1**이 포함됩니다. 쿼리가 실행되어 완료되면 이 열에 **0**이 포함됩니다.
external_table_name	char(136)	외부 테이블 스캔을 위한 테이블 외부 이름의 내부 형식입니다.
file_format	character(16)	외부 테이블 데이터의 파일 형식입니다.
is_partitioned	char(1)	true(**t**)인 경우, 이 열 값은 외부 테이블이 파티셔닝되어 있음을 나타냅니다.
is_rrscan	char(1)	true(**t**)인 경우, 이 열 값은 범위 제한 스캔이 적용되었음을 나타냅니다.
is_nested	varchar(1)	true(**t**)인 경우 이 열 값은 중첩 열 데이터 형식에 액세스했음을 나타냅니다.
s3_scanned_rows	bigint	Amazon S3에서 스캔되어 Redshift Spectrum 계층으로 전송된 행의 수.
s3_scanned_bytes	bigint	압축된 데이터를 기반으로 Amazon S3에서 스캔되어 Redshift Spectrum 계층으로 전송된 바이트의 수.
s3query_returned_rows	bigint	Redshift Spectrum 계층에서 클러스터로 반환된 행의 수.

열 명칭	데이터 유형	설명
s3query_returned_bytes	bigint	Redshift Spectrum 계층에서 클러스터로 반환된 바이트의 수. Amazon Redshift로 반환되는 데이터의 양이 많으면 시스템 성능이 영향을 받을 수 있습니다.
files	정수	이 Redshift Spectrum 쿼리에 대해 처리된 파일의 수. 파일 수가 적으면 병렬 처리의 이점이 제한됩니다.
files_max	정수	한 조각에서 처리된 파일의 최대 개수.
files_avg	정수	한 조각에서 처리된 파일의 평균 개수.
splits	bigint	이 세그먼트에 대해 처리되는 분할 수입니다. 이 조각에서 처리되는 분할 수입니다. 분할할 수 있는 큰 데이터 파일의 경우, 예를 들어 약 512MB보다 큰 데이터 파일의 경우 Redshift Spectrum은 병렬 처리를 위해 파일을 여러 개의 S3 요청으로 분할하려고 합니다.
splits_max	정수	이 조각에서 처리되는 최대 분할 수입니다.
splits_avg	bigint	이 조각에서 처리되는 평균 분할 수입니다.
total_split_size	bigint	처리되는 모든 분할의 총 크기입니다.
max_split_size	bigint	처리되는 최대 분할 크기(바이트)입니다.
avg_split_size	bigint	처리되는 평균 분할 크기(바이트)입니다.
total_retries	bigint	이 세그먼트에서 Redshift Spectrum 쿼리에 대한 총 재시도 횟수입니다.
max_retries	정수	처리된 개별 파일의 최대 재시도 횟수입니다.
max_request_duration	bigint	개별 파일 요청의 최대 지속 시간(마이크로초)입니다. 오랫동안 실행 중인 쿼리는 병목 현상을 나타낼 수 있습니다.

열 명칭	데이터 유형	설명
avg_request_duration	bigint	파일 요청의 평균 지속 시간(마이크로초)입니다.
max_request_parallelism	정수	이 Redshift Spectrum 쿼리에 대해 한 개의 조각에 있는 최대 병렬 요청 수입니다.
avg_request_parallelism	double precision	이 Redshift Spectrum 쿼리에 대해 한 개의 조각에 있는 평균 병렬 요청 수입니다.
total_slowdown_count	bigint	외부 테이블 스캔 중에 속도가 느려지는 오류가 발생한 총 Amazon S3 요청 수입니다.
max_slowdown_count	정수	한 조각의 외부 테이블 스캔 중에 속도가 느려지는 오류가 발생한 최대 Amazon S3 요청 수입니다.

샘플 쿼리

다음 예는 마지막으로 실행된 쿼리에 대한 스캔 단계 세부 정보를 가져옵니다.

```
select query, segment, elapsed, s3_scanned_rows, s3_scanned_bytes,
 s3query_returned_rows, s3query_returned_bytes, files
from svcs_s3query_summary
where query = pg_last_query_id()
order by query,segment;
```

```
query | segment | elapsed | s3_scanned_rows | s3_scanned_bytes | s3query_returned_rows
 | s3query_returned_bytes | files
------+---------+---------+-----------------+------------------+----------------------
+------------------------+------
 4587 |       2 |   67811 |               0 |                0 |                    0
 |                      0 |     0
 4587 |       2 |  591568 |          172462 |         11260097 |                 8513
 |                 170260 |     1
 4587 |       2 |  216849 |               0 |                0 |                    0
 |                      0 |     0
```

```
 4587 |       2 |  216671 |               0 |               0 |                     0
 |                     0 |     0
```

SVCS_STREAM_SEGS

스트림과 동시 세그먼트 사이의 관계를 나열합니다.

> **Note**
>
> 접두사 SVCS를 포함하는 시스템 뷰는 동시성 확장 클러스터와 기본 클러스터 모두의 쿼리에 대한 세부 정보를 제공합니다. 이 뷰는 접두사 STL을 포함하는 테이블과 유사합니다. 단, STL 테이블은 기본 클러스터에서 실행된 쿼리에 대한 정보만 제공합니다.

SVCS_STREAM_SEGS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
스트림	정수	쿼리의 동시 세그먼트 집합
segment	정수	쿼리 세그먼트를 식별할 수 있는 번호

샘플 쿼리

가장 최근 쿼리에서 스트림과 동시 세그먼트 사이의 관계를 보려면 다음과 같이 쿼리를 입력합니다.

```
select *
from svcs_stream_segs
where query = pg_last_query_id();

 query | stream | segment
```

```
-------+--------+---------
    10 |      1 |       2
    10 |      0 |       0
    10 |      2 |       4
    10 |      1 |       3
    10 |      0 |       1
(5 rows)
```

SVCS_UNLOAD_LOG

SVCS_UNLOAD_LOG를 사용하여 UNLOAD 작업에 대한 세부 정보를 가져옵니다.

SVCS_UNLOAD_LOG는 UNLOAD 문에서 생성되는 각 파일마다 행 1개를 기록합니다. 예를 들어, UNLOAD를 실행하여 파일 12개가 생성된다면 SVCS_UNLOAD_LOG에 포함되는 해당 행의 수도 12개입니다. 이 뷰는 STL_UNLOAD_LOG 시스템 테이블에서 파생되지만 동시성 확장 클러스터에서 실행된 쿼리에 대한 조각 수준을 표시하지 않습니다.

> **Note**
>
> 접두사 SVCS를 포함하는 시스템 뷰는 동시성 확장 클러스터와 기본 클러스터 모두의 쿼리에 대한 세부 정보를 제공합니다. 이 뷰는 접두사 STL을 포함하는 테이블과 유사합니다. 단, STL 테이블은 기본 클러스터에서 실행된 쿼리에 대한 정보만 제공합니다.

SVCS_UNLOAD_LOG는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID입니다.
쿼리	정수	쿼리 ID입니다.
pid	정수	쿼리 문과 연결된 프로세스 ID입니다.
경로	character(1280)	파일의 전체 Amazon S3 객체 경로.
start_time	타임스탬프	UNLOAD 작업의 시작 시간입니다.

열 명칭	데이터 유형	설명
end_time	타임스탬프	UNLOAD 작업의 종료 시간입니다.
line_count	bigint	파일로 언로드되는 라인(행)의 수입니다.
transfer_size	bigint	전송되는 바이트 수입니다.
file_format	character(10)	언로드된 파일의 형식입니다.

샘플 쿼리

UNLOAD 명령으로 Amazon S3에 작성된 파일 목록을 가져오려면 UNLOAD 작업을 마친 후 Amazon S3 목록 작업을 호출하면 됩니다. 하지만 Amazon S3 목록 작업은 최종 일관성(eventually consistent)을 따르기 때문에 호출을 얼마나 빠르게 실행하는가에 따라 목록이 불안정할 수 있습니다. 믿을 수 있는 완전한 목록을 바로 가져오려면 SVCS_UNLOAD_LOG에 대한 쿼리를 실행하십시오.

다음은 마지막으로 완료된 쿼리에 대해 UNLOAD로 생성된 파일의 경로 이름을 반환하는 쿼리입니다.

```
select query, substring(path,0,40) as path
from svcs_unload_log
where query = pg_last_query_id()
order by path;
```

위의 명령은 다음과 같은 샘플 출력을 반환합니다.

```
 query |             path
 ------+---------------------------------------------
  2320 | s3://amzn-s3-demo-bucket/venue0000_part_00
  2320 | s3://amzn-s3-demo-bucket/venue0001_part_00
  2320 | s3://amzn-s3-demo-bucket/venue0002_part_00
  2320 | s3://amzn-s3-demo-bucket/venue0003_part_00
(4 rows)
```

기본 클러스터의 SVL 뷰

SVL 뷰는 자세한 정보를 위해 STL 테이블 및 로그에 대한 참조를 포함하는 Amazon Redshift의 시스템 뷰입니다.

이러한 보기는 이들 테이블에서 찾을 수 있는 일반적으로 쿼리되는 데이터에 대한 보다 빠르고 쉬운 액세스를 제공합니다.

> **Note**
>
> SVL_QUERY_SUMMARY 보기에는 Amazon Redshift가 실행하는 쿼리에 대한 정보만 포함되며 다른 유틸리티 및 DDL 명령이 실행하는 쿼리의 정보는 포함되지 않습니다. DDL 및 유틸리티 명령을 포함하여 Amazon Redshift에 의해 실행되는 모든 문의 완전한 목록과 정보를 보려면 SVL_STATEMENTTEXT 뷰를 쿼리할 수 있습니다.

주제

- SVL_AUTO_WORKER_ACTION
- SVL_COMPILE
- SVL_DATASHARE_CHANGE_LOG
- SVL_DATASHARE_CROSS_REGION_USAGE
- SVL_DATASHARE_USAGE_CONSUMER
- SVL_DATASHARE_USAGE_PRODUCER
- SVL_FEDERATED_QUERY
- SVL_MULTI_STATEMENT_VIOLATIONS
- SVL_MV_REFRESH_STATUS
- SVL_QERROR
- SVL_QLOG
- SVL_QUERY_METRICS
- SVL_QUERY_METRICS_SUMMARY
- SVL_QUERY_QUEUE_INFO
- SVL_QUERY_REPORT
- SVL_QUERY_SUMMARY
- SVL_RESTORE_ALTER_TABLE_PROGRESS
- SVL_S3LIST
- SVL_S3LOG
- SVL_S3PARTITION
- SVL_S3PARTITION_SUMMARY

- SVL_S3QUERY
- SVL_S3QUERY_SUMMARY
- SVL_S3RETRIES
- SVL_SPATIAL_SIMPLIFY
- SVL_SPECTRUM_SCAN_ERROR
- SVL_STATEMENTTEXT
- SVL_STORED_PROC_CALL
- SVL_STORED_PROC_MESSAGES
- SVL_TERMINATE
- SVL_UDF_LOG
- SVL_USER_INFO
- SVL_VACUUM_PERCENTAGE

SVL_AUTO_WORKER_ACTION

자동 최적화를 위해 정의된 테이블에 Amazon Redshift에서 수행한 자동화된 작업을 기록합니다.

SVL_AUTO_WORKER_ACTION은 수퍼유저에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
table_id	정수	테이블 식별자입니다.
type	character (32)	권장 사항의 유형입니다. 가능한 값은 distkey와 sortkey입니다.
status	character (128)	권장 사항의 완료 상태입니다. 가능한 값은 Start, Complete, Skipped, Abort, Checkpoint 및 Failed입니다.
eventtime	타임스탬프	`status` 열의 타임스탬프입니다.

열 명칭	데이터 유형	설명
SEQUENCE	정수	잘린 previous_state 값의 시퀀스 번호입니다. 단일 previous_state 에 200자 이상이 포함된 경우 해당 값에 대해 추가 행이 기록됩니다. 시퀀스 0이 첫 번째 행이고 1이 두 번째 행이 되는 방식입니다.
previous_state	character (200)	권장 사항을 적용하기 전에 테이블의 이전 배포 스타일과 정렬 키입니다. 값은 200자 단위로 잘립니다.

status 열 값의 몇 가지 예는 다음과 같습니다.

- Skipped:Table not found.
- Skipped:Recommendation is empty.
- Skipped:Apply sortkey recommendation is disabled.
- Skipped:Retry exceeds the maximum limit for a table.
- Skipped:Table column has changed.
- Abort:This table is not AUTO.
- Abort:This table has been recently converted.
- Abort:This table exceeds table size threshold.
- Abort:This table is already the recommended style.
- Checkpoint: progress *21.9963*%.

샘플 쿼리

다음 예에서 결과의 행은 Amazon Redshift에서 수행한 작업을 보여줍니다.

```
select table_id, type, status, eventtime, sequence, previous_state
from SVL_AUTO_WORKER_ACTION;
```

```
 table_id |  type   |                        status                        |
 eventtime          | sequence | previous_state
----------+---------+------------------------------------------------------
+----------------------------+----------+----------------
```

```
  118082 | sortkey | Start                                                | 2020-08-22
19:42:20.727049 | 0       |
  118078 | sortkey | Start                                                | 2020-08-22
19:43:54.728819 | 0       |
  118082 | sortkey | Start                                                | 2020-08-22
19:42:52.690264 | 0       |
  118072 | sortkey | Start                                                | 2020-08-22
19:44:14.793572 | 0       |
  118082 | sortkey | Failed                                               | 2020-08-22
19:42:20.728917 | 0       |
  118078 | sortkey | Complete                                             | 2020-08-22
19:43:54.792705 | 0       | SORTKEY: None;
  118086 | sortkey | Complete                                             | 2020-08-22
19:42:00.72635  | 0       | SORTKEY: None;
  118082 | sortkey | Complete                                             | 2020-08-22
19:43:34.728144 | 0       | SORTKEY: None;
  118072 | sortkey | Skipped:Retry exceeds the maximum limit for a table. | 2020-08-22
19:44:46.706155 | 0       |
  118086 | sortkey | Start                                                | 2020-08-22
19:42:00.685255 | 0       |
  118082 | sortkey | Start                                                | 2020-08-22
19:43:34.69531  | 0       |
  118072 | sortkey | Start                                                | 2020-08-22
19:44:46.703331 | 0       |
  118082 | sortkey | Checkpoint: progress 14.755079%                      | 2020-08-22
19:42:52.692828 | 0       |
  118072 | sortkey | Failed                                               | 2020-08-22
19:44:14.796071 | 0       |
  116723 | sortkey | Abort:This table is not AUTO.                        | 2020-10-28
05:12:58.479233 | 0       |
  110203 | distkey | Abort:This table is not AUTO.                        | 2020-10-28
05:45:54.67259  | 0       |
```

SVL_COMPILE

쿼리의 각 쿼리 세그먼트에 대한 컴파일 시간 및 위치를 기록합니다.

SVL_COMPILE은 모든 사용자가 볼 수 있습니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> SVL_COMPILE에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_QUERY_HISTORY을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

SVCS_COMPILE에 대한 자세한 내용은 SVCS_COMPILE 섹션을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
xid	bigint	문에 연결된 트랜잭션 ID.
pid	정수	쿼리 문과 연결된 프로세스 ID
쿼리	정수	쿼리 ID. 다양한 다른 시스템 테이블 및 보기를 조인하는 데 사용할 수 있습니다.
segment	정수	컴파일할 쿼리 세그먼트.
locus	정수	세그먼트가 실행되는 위치이며, 컴퓨터 노드에 있는 경우, **1**이고, 리더 노드에 있는 경우, **2**입니다.
starttime	타임스탬프	컴파일이 시작된 UTC 시간.
endtime	타임스탬프	컴파일이 끝난 UTC 시간.
컴파일	정수	컴파일이 재사용된 경우, **0**이고, 세그먼트가 컴파일된 경우 **1**입니다.

샘플 쿼리

이 예에서는 쿼리 35878과 35879가 동일한 SQL 문을 실행했습니다. 쿼리 35878의 컴파일 열은 4개의 쿼리 세그먼트에 대해 1을 보여 주며, 이는 이 세그먼트들이 컴파일되었음을 나타냅니다. 쿼리 35879는 모든 세그먼트의 컴파일 열에서 0을 보여 주며, 세그먼트를 다시 컴파일할 필요가 없었음을 나타냅니다.

```
select userid, xid,  pid, query, segment, locus,
datediff(ms, starttime, endtime) as duration, compile
from svl_compile
where query = 35878 or query = 35879
order by query, segment;

 userid |  xid   |  pid  | query | segment | locus | duration | compile
--------+--------+-------+-------+---------+-------+----------+---------
    100 | 112780 | 23028 | 35878 |       0 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       1 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       2 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       3 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       4 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       5 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       6 |     1 |     1380 |       1
    100 | 112780 | 23028 | 35878 |       7 |     1 |     1085 |       1
    100 | 112780 | 23028 | 35878 |       8 |     1 |     1197 |       1
    100 | 112780 | 23028 | 35878 |       9 |     2 |      905 |       1
    100 | 112782 | 23028 | 35879 |       0 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       1 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       2 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       3 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       4 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       5 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       6 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       7 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       8 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       9 |     2 |        0 |       0
(20 rows)
```

SVL_DATASHARE_CHANGE_LOG

생산자 클러스터와 소비자 클러스터 모두에서 datashare의 변경 내용을 추적하기 위한 통합 뷰를 기록합니다.

SVL_DATASHARE_CHANGE_LOG는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_DATASHARE_CHANGE_LOG에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	작업을 수행하는 사용자의 ID입니다.
사용자 이름	varchar(128)	작업을 수행하는 사용자의 이름입니다.
pid	정수	프로세스의 ID입니다.
xid	bigint	트랜잭션의 ID입니다.
share_id	정수	영향을 받는 datashare의 ID입니다.
share_name	varchar(128)	datashare의 이름입니다.
source_database_id	정수	datashare가 속한 데이터베이스의 ID입니다.
source_database_name	varchar(128)	datashare가 속한 데이터베이스의 이름입니다.
consumer_database_id	정수	datashare에서 가져온 데이터베이스의 ID입니다.
consumer_database_name	varchar(128)	datashare에서 가져온 데이터베이스의 이름입니다.

열 명칭	데이터 유형	설명
arn	varchar(192)	가져온 데이터베이스를 뒷받침하는 리소스의 ARN입니다.
recordtime	타임스탬프	작업의 타임스탬프입니다.
작업	varchar(128)	실행 중인 작업입니다. 가능한 값은 CREATE DATASHARE, DROP DATASHARE, GRANT ALTER, REVOKE ALTER, GRANT SHARE, REVOKE SHARE, ALTER ADD, ALTER REMOVE, ALTER SET, GRANT USAGE, REVOKE USAGE, CREATE DATABASE, GRANT 또는 REVOKE USAGE(공유 데이터베이스 대상), DROP SHARED DATABASE, ALTER SHARED DATABASE입니다.
status	정수	작업의 상태입니다. 가능한 값은 SUCCESS와 ERROR-ERROR CODE입니다.
share_object_type	varchar(64)	datashare에서 추가되거나 제거된 데이터베이스 객체의 유형입니다. 가능한 값은 schema, table, column, function 및 view입니다. 생산자 클러스터에 대한 필드입니다.
share_object_id	정수	datashare에서 추가되거나 제거된 데이터베이스 객체의 ID입니다. 생산자 클러스터에 대한 필드입니다.
share_object_name	varchar(128)	datashare에서 추가되거나 제거된 데이터베이스 객체의 이름입니다. 생산자 클러스터에 대한 필드입니다.
target_user_type	varchar(16)	권한이 부여된 사용자 또는 그룹의 유형입니다. 생산자 및 소비자 클러스터 모두에 대한 필드입니다.
target_userid	정수	권한이 부여된 사용자 또는 그룹의 ID입니다. 생산자 및 소비자 클러스터 모두에 대한 필드입니다.
target_username	varchar(128)	권한이 부여된 사용자 또는 그룹의 이름입니다. 생산자 및 소비자 클러스터 모두에 대한 필드입니다.

열 명칭	데이터 유형	설명
consumer_account	varchar(16)	데이터 소비자의 계정 ID입니다. 생산자 클러스터에 대한 필드입니다.
consumer_namespace	varchar(64)	데이터 소비자 계정의 네임스페이스입니다. 생산자 클러스터에 대한 필드입니다.
producer_account	varchar(16)	datashare가 속한 생산자 계정의 계정 ID입니다. 소비자 클러스터에 대한 필드입니다.
producer_namespace	varchar(64)	datashare가 속한 제품 계정의 네임스페이스입니다. 소비자 클러스터에 대한 필드입니다.
attribute_name	varchar(64)	datashare 또는 공유 데이터베이스의 속성 이름입니다.
attribute_value	varchar(128)	datashare 또는 공유 데이터베이스의 속성 값입니다.
message	varchar(512)	작업이 실패할 경우 오류 메시지입니다.

샘플 쿼리

다음 예에서는 SVL_DATASHARE_CHANGE_LOG 뷰를 보여줍니다.

```
SELECT DISTINCT action
FROM svl_datashare_change_log
WHERE share_object_name LIKE 'tickit%';

         action
 ----------------------
  "ALTER DATASHARE ADD"
```

SVL_DATASHARE_CROSS_REGION_USAGE

SVL_DATASHARE_CROSS_REGION_USAGE 뷰를 사용하면 교차 리전 데이터 공유 쿼리로 인해 발생한 교차 리전 데이터 전송 사용량을 요약하여 확인할 수 있습니다. SVL_DATASHARE_CROSS_REGION_USAGE는 세그먼트 수준에서 세부 정보를 집계합니다.

SVL_DATASHARE_CROSS_REGION_USAGE는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_DATASHARE_CROSS_REGION_USAGE에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
쿼리	정수	쿼리의 ID입니다. 이 값을 사용하여 다른 시스템 테이블 및 뷰를 조인합니다.
segment	bigint	세그먼트의 번호입니다. 하나의 쿼리는 여러 세그먼트로 구성되며, 각각의 세그먼트는 하나 이상의 단계로 구성됩니다.
start_time	시간	데이터 전송이 시작된 UTC 시간입니다.
end_time	시간	데이터 전송이 종료된 UTC 시간입니다.
transferred_data	bigint	생산자 리전에서 소비자 리전으로 전송된 데이터의 바이트 수입니다.
source_region	char(25)	쿼리가 데이터를 전송한 생산자 리전입니다.
recordtime	타임스탬프	작업이 기록되는 시간입니다.

샘플 쿼리

다음 예에서는 SVL_DATASHARE_CROSS_REGION_USAGE 뷰를 보여줍니다.

```
SELECT query, segment, transferred_data, source_region
from svl_datashare_cross_region_usage
where query = pg_last_query_id()
order by query,segment;

  query | segment | transferred_data | source_region
--------+---------+------------------+---------------
 200048 |       2 |          4194304 |    us-west-1
 200048 |       2 |          4194304 |    us-east-2
```

SVL_DATASHARE_USAGE_CONSUMER

datashare의 활동과 사용량을 기록합니다. 이 뷰는 소비자 클러스터에만 관련이 있습니다.

SVL_DATASHARE_USAGE_CONSUMER는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰
인 SYS_DATASHARE_USAGE_CONSUMER에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	요청을 발행하는 사용자의 ID입니다.
pid	정수	쿼리를 실행하는 리더 프로세스의 ID입니다.
xid	bigint	현재 트랜잭션의 컨텍스트입니다.
request_id	varchar(50)	요청된 API 호출의 고유 ID입니다.

열 명칭	데이터 유형	설명
request_type	varchar(25)	생산자 클러스터에 대한 요청 형식입니다.
transaction_uid	varchar(50)	트랜잭션의 고유 ID입니다.
recordtime	타임스탬프	작업이 기록되는 시간입니다.
status	정수	요청된 API 호출의 상태입니다.
오류	varchar(512)	오류에 대한 메시지입니다.

샘플 쿼리

다음 예에서는 SVL_DATASHARE_USAGE_CONSUMER 뷰를 보여줍니다.

```
SELECT request_type, status, trim(error) AS error
FROM svl_datashare_usage_consumer

  request_type  | status | error
----------------+--------+--------
 "GET RELATION" |   0    |
```

SVL_DATASHARE_USAGE_PRODUCER

datashare의 활동과 사용량을 기록합니다. 이 뷰는 생산자 클러스터에만 관련이 있습니다.

SVL_DATASHARE_USAGE_PRODUCER는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_DATASHARE_USAGE_PRODUCER에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
share_id	정수	datashare의 객체 ID(OID)입니다.
share_name	varchar(128)	datashare의 이름입니다.
request_id	varchar(50)	요청된 API 호출의 고유 ID입니다.
request_type	varchar(25)	생산자 클러스터에 대한 요청 형식입니다.
object_type	varchar(64)	datashare에서 공유되는 객체의 형식입니다. 가능한 값은 schema, table, column, function 및 view입니다.
object_oid	정수	datashare에서 공유되는 객체의 ID입니다.
객체 이름	varchar(128)	datashare에서 공유되는 객체의 이름입니다.
consumer_account	varchar(16)	datashare가 공유되는 소비자 계정의 계정입니다.
consumer_namespace	varchar(64)	datashare가 공유되는 소비자 계정의 네임스페이스입니다.
consumer_transaction_uid	varchar(50)	소비자 클러스터에 있는 문의 고유 트랜잭션 ID입니다.
recordtime	타임스탬프	작업이 기록되는 시간입니다.
status	정수	datashare의 상태입니다.

열 명칭	데이터 유형	설명
오류	varchar(512)	오류에 대한 메시지입니다.
consumer_region	char(64)	소비자 클러스터가 속한 리전입니다.

샘플 쿼리

다음 예에서는 SVL_DATASHARE_USAGE_PRODUCER 뷰를 보여줍니다.

```
SELECT DISTINCT request_type
FROM svl_datashare_usage_producer
WHERE object_name LIKE 'tickit%';

   request_type
 ------------------
   "GET RELATION"
```

SVL_FEDERATED_QUERY

SVL_FEDERATED_QUERY 보기를 사용하여 연합 쿼리 호출에 대한 정보를 볼 수 있습니다.

SVL_FEDERATED_QUERY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_EXTERNAL_QUERY_DETAIL에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	쿼리를 실행하는 사용자의 ID입니다.

열 명칭	데이터 유형	설명
xid	bigint	트랜잭션 ID.
pid	정수	쿼리를 실행하는 리더 프로세스의 ID입니다.
쿼리	정수	연합 호출의 쿼리 ID입니다.
sourcetype	character(32)	연합 호출 소스 유형(예: "PG")입니다.
recordtime	타임스탬프	연합을 위해 쿼리가 전송되는 시간입니다. UTC가 사용됩니다.
querytext	character(4000)	실행을 위해 원격 PostgreSQL 엔진으로 전송되는 쿼리 문자열입니다.
num_rows	bigint	페더레이션 쿼리에서 반환되는 행의 수입니다.
num_bytes	bigint	페더레이션 쿼리에서 반환되는 바이트의 수입니다.
duration	bigint	커서 호출에서 행을 가져오는 데 소요되는 시간(마이크로초)입니다. 페더레이션 쿼리를 실행하고 결과를 가져오는 데 소요되는 시간입니다.

샘플 쿼리

연합 쿼리 호출에 대한 정보를 표시하려면 다음 쿼리를 실행합니다.

```
select query, trim(sourcetype) as type, recordtime, trim(querytext) as "PG Subquery"
 from svl_federated_query where query = 4292;
```

```
 query | type |         recordtime         |                          pg subquery
-------+------+----------------------------
+-------------------------------------------------------------
  4292 | PG   | 2020-03-27 04:29:58.485126 | SELECT "level" FROM functional.employees
 WHERE ("level" >= 6)
(1 row)
```

SVL_MULTI_STATEMENT_VIOLATIONS

트랜잭션 블록 제한을 위반하는 시스템에서 실행되는 모든 SQL 명령의 전체 레코드를 가져오려면 SVL_MULTI_STATEMENT_VIOLATIONS 뷰를 사용합니다.

Amazon Redshift가 트랜잭션 블록 또는 다중 문 요청 내에서 제한하는 다음 SQL 명령을 실행할 때 위반이 발생합니다.

- 데이터베이스 생성
- DROP DATABASE
- ALTER TABLE APPEND
- CREATE EXTERNAL TABLE
- DROP EXTERNAL TABLE
- RENAME EXTERNAL TABLE
- ALTER EXTERNAL TABLE
- CREATE TABLESPACE
- DROP TABLESPACE
- CREATE LIBRARY
- DROP LIBRARY
- REBUILDCAT
- INDEXCAT
- REINDEX DATABASE
- VACUUM
- GRANT
- COPY

> **Note**
>
> 이 뷰에 항목이 있으면 해당 애플리케이션과 SQL 스크립트를 변경합니다. 이러한 제한된 SQL 명령의 사용을 트랜잭션 블록 외부로 이동하도록 애플리케이션 코드를 변경하는 것이 좋습니다. 추가 지원이 필요한 경우 AWS Support에 문의하세요.

SVL_MULTI_STATEMENT_VIOLATIONS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	위반을 일으킨 사용자의 ID입니다.
데이터베이스	character(32)	사용자가 연결된 데이터베이스의 이름입니다.
cmdname	character(20)	트랜잭션 블록 또는 다중 문 요청 내에서 실행할 수 없는 명령의 이름입니다. 예를 들면 CREATE DATABASE, DROP DATABASE, ALTER TABLE APPEND, CREATE EXTERNAL TABLE, DROP EXTERNAL TABLE, RENAME EXTERNAL TABLE, ALTER EXTERNAL TABLE, CREATE LIBRARY, DROP LIBRARY, REBUILDCAT, INDEXCAT, REINDEX DATABASE, VACUUM, GRANT(외부 리소스 대상), CLUSTER, COPY, CREATE TABLESPACE 및 DROP TABLESPACE입니다.
xid	bigint	문과 연결된 트랜잭션 ID
pid	정수	문의 프로세스 ID입니다.

열 명칭	데이터 유형	설명
레이블	character(320)	쿼리 실행에 사용되는 파일의 이름 또는 SET QUERY_GROUP 명령을 사용하여 정의되는 레이블. 쿼리가 파일 기반이 아니거나 QUERY_GROUP 파라미터가 설정되지 않은 경우, 이 필드의 값은 공백입니다.
starttime	타임스탬프	문 실행이 시작된 정확한 시간으로 초의 소수점 이하 자릿수는 6자리입니다(예: **2009-06-12 11:29:19.131358**).
endtime	타임스탬프	문 실행이 끝난 정확한 시간으로 초의 소수점 이하 자릿수는 6자리입니다(예: **2009-06-12 11:29:19.193640**).
SEQUENCE	정수	단일 문에 200자 이상이 포함된 경우, 해당 문에 대해 추가 행이 기록됩니다. 시퀀스 0이 첫 번째 행이고 1이 두 번째 행이 되는 방식입니다.
type	varchar(10)	SQL 문의 형식: **QUERY**, **DDL** 또는 **UTILITY**.
텍스트	character(200)	200자씩 증가하는 SQL 텍스트. 이 필드에는 백슬래시(\\) 및 줄 바꿈(\n) 등의 특수 문자가 포함될 수 있습니다.

샘플 쿼리

다음 쿼리는 위반이 있는 여러 문을 반환합니다.

```
select * from svl_multi_statement_violations order by starttime asc;

userid | database | cmdname | xid | pid | label | starttime | endtime | sequence | type
 | text
================================================================================================
1 | dev | CREATE DATABASE | 1034 | 5729 |label1 | ******** | ******* | 0 | DDL |
 create table c(b int);
1 | dev | CREATE DATABASE | 1034 | 5729 |label1 | ******** | ******* | 0 | UTILITY |
 create database b;
1 | dev | CREATE DATABASE | 1034 | 5729 |label1 | ******** | ******* | 0 | UTILITY |
 COMMIT
...
```

SVL_MV_REFRESH_STATUS

SVL_MV_REFRESH_STATUS 뷰는 구체화된 보기의 새로 고침 작업에 대한 행을 포함합니다.

구체화된 뷰에 대한 자세한 내용은 Amazon Redshift의 구체화된 뷰 섹션을 참조하세요.

SVL_MV_REFRESH_STATUS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_MV_REFRESH_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
db_name	char(128)	구체화된 보기를 포함하는 데이터베이스입니다.
userid	bigint	새로 고침을 수행한 사용자의 ID입니다.
schema_name	char(128)	구체화된 보기의 스키마입니다.
mv_name	char(128)	구체화된 보기 이름입니다.
xid	bigint	새로 고침의 트랜잭션 ID입니다.
starttime	타임스탬프	새로 고침의 시작 시간입니다.
endtime	타임스탬프	새로 고침의 종료 시간입니다.
status	텍스트	새로 고침의 상태입니다. 값 예로는 다음이 포함됩니다. • 업데이트된 MV를 증분 방식으로 새로 고칩니다. 스트리밍을 위한 구체화된 뷰인 경우 메시지에는 레코드 수와 관련된 추가 한정자가 있을 수 있습니다. 여기에는 다음이 포함됩니다.

열 명칭	데이터 유형	설명
		• Stream returned no new data(스트림에서 새 데이터를 반환하지 않았습니다.) - 검색된 레코드가 없습니다. • 스트림에서 수신한 모든 레코드를 건너뛰었습니다. - 레코드가 검색되었지만 오류 때문에 모두 건너뛰었습니다. • 일부 스트림 레코드를 건너뛰었습니다. - 레코드가 검색되었지만 오류 때문에 일부를 건너뛰었습니다. 한정자가 없는 경우 하나 이상의 레코드가 검색되고 구체화된 뷰에서 모든 레코드를 사용할 수 있습니다. 다음 한정자가 표시될 수도 있습니다. • The stream may contain more data(스트림에 더 많은 데이터가 포함될 수 있습니다.) - Amazon Redshift에서 더 이상 사용할 레코드가 없다고 판단하기 전에 새로 고침이 종료되었습니다. 스트림이 최신 상태일 수 있지만 Amazon Redshift에서 이를 확인하지는 않았습니다. • 처음부터 다시 계산된 MV를 새로 고칩니다. • 부분적으로 업데이트된 MV를 활성 트랜잭션까지 증분 방식으로 새로 고칩니다. • MV가 이미 업데이트되었습니다. • 새로 고침에 실패했습니다. 기본 테이블 열의 이름이 바뀌었습니다. • 새로 고침에 실패했습니다. 기본 테이블 열 유형이 변경되었습니다. • 새로 고침에 실패했습니다. 기본 테이블의 이름이 바뀌었습니다. • 내부 오류로 인해 새로 고침이 실패했습니다.

열 명칭	데이터 유형	설명
		• 새로 고침에 실패했습니다. 기본 테이블 열이 삭제되었습니다. • 새로 고침에 실패했습니다. MV의 스키마 이름이 변경되었습니다. • 새로 고침에 실패했습니다. MV를 찾을 수 없습니다. • 과도한 사용자 워크로드로 인해 자동 새로 고침이 중단되었습니다. • 새로 고침에 실패했습니다. 직렬화 가능한 격리 위반
refresh_type	char(32)	새로 고침 형식의 정의입니다. 예제 값에는 [수동(Manual)] 및 [자동(Auto)]이 포함됩니다.

샘플 쿼리

구체화된 보기의 새로 고침 상태를 보려면 다음 쿼리를 실행합니다.

```
select * from svl_mv_refresh_status;
```

위 쿼리는 다음과 같은 샘플 출력을 반환합니다.

```
 db_name | userid |  schema   |  name   |  xid  |         starttime          |
       endtime           |                        status                        |
 refresh_type
---------+--------+-----------+---------+-------+----------------------------
+----------------------------+------------------------------------------------------
+-----------------
 dev     |    169 | mv_schema | mv_test |  6640 | 2020-02-14 02:26:53.497935 |
 2020-02-14 02:26:53.556156 | Refresh successfully recomputed MV from scratch     |
 Manual
 dev     |    166 | mv_schema | mv_test |  6517 | 2020-02-14 02:26:39.287438 |
 2020-02-14 02:26:39.349539 | Refresh successfully updated MV incrementally       |
 Auto
```

```
dev     |    162 | mv_schema | mv_test |  6388 | 2020-02-14 02:26:27.863426 |
2020-02-14 02:26:27.918307 | Refresh successfully recomputed MV from scratch     |
Manual
dev     |    161 | mv_schema | mv_test |  6323 | 2020-02-14 02:26:20.020717 |
2020-02-14 02:26:20.080002 | Refresh successfully updated MV incrementally       |
Auto
dev     |    161 | mv_schema | mv_test |  6301 | 2020-02-14 02:26:05.796146 |
2020-02-14 02:26:07.853986 | Refresh successfully recomputed MV from scratch     |
Manual
dev     |    153 | mv_schema | mv_test |  6024 | 2020-02-14 02:25:18.762335 |
2020-02-14 02:25:20.043462 | MV was already updated                              |
Manual
dev     |    143 | mv_schema | mv_test |  5557 | 2020-02-14 02:24:23.100601 |
2020-02-14 02:24:23.100633 | MV was already updated                              |
Manual
dev     |    141 | mv_schema | mv_test |  5447 | 2020-02-14 02:23:54.102837 |
2020-02-14 02:24:00.310166 | Refresh successfully updated MV incrementally       |
Auto
dev     |      1 | mv_schema | mv_test |  5329 | 2020-02-14 02:22:26.328481 |
2020-02-14 02:22:28.369217 | Refresh successfully recomputed MV from scratch     |
Auto
dev     |    138 | mv_schema | mv_test |  5290 | 2020-02-14 02:21:56.885093 |
2020-02-14 02:21:56.885098 | Refresh failed. MV was not found                    |
Manual
```

SVL_QERROR

SVL_QERROR 보기는 사용되지 않습니다.

SVL_QLOG

SVL_QLOG 뷰에는 데이터베이스에 대해 실행된 모든 쿼리의 로그가 포함됩니다.

Amazon Redshift는 STL_QUERY 테이블에서 읽을 수 있는 정보 하위 집합으로 SVL_QLOG 뷰를 생성합니다. 최근에 실행된 쿼리의 쿼리 ID를 찾거나 쿼리 완료까지 걸린 시간을 보려면 이 테이블을 사용합니다.

SVL_QLOG는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 이 ID를 사용하여 다양한 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
xid	bigint	트랜잭션 ID.
pid	정수	쿼리에 연결된 프로세스 ID.
starttime	타임스탬프	문 실행이 시작된 정확한 시간으로 초의 소수점 이하 자릿수는 6자리입니다(예: **2009-06-12 11:29:19.131358**).
endtime	타임스탬프	문 실행이 끝난 정확한 시간으로 초의 소수점 이하 자릿수는 6자리입니다(예: **2009-06-12 11:29:19.193640**).
Elapsed	bigint	쿼리가 실행되는 데 걸린 시간 길이(마이크로초).
aborted	정수	쿼리가 시스템에 의해 중지되거나 사용자에 의해 취소되는 경우, 이 열에 **1**이 포함됩니다. 쿼리가 실행되어 완료되면 이 열에 **0**이 포함됩니다. 워크로드 관리를 위해 취소되었다가 이후에 다시 시작되는 쿼리 역시 이 열에서 **1**의 값을 갖습니다.
레이블	character(320)	쿼리 실행에 사용되는 파일의 이름 또는 SET QUERY GROUP 명령을 사용하여 정의되는 레이블. 쿼리가 파일 기반이 아니거나 QUERY_GROUP 파라미터가 설정되지 않은 경우, 이 필드의 값은 `default`입니다.
substring	character(60)	잘린 쿼리 텍스트.

열 명칭	데이터 유형	설명
source_query	정수	쿼리에 결과 캐싱을 사용한 경우 캐시된 결과의 소스인 쿼리의 쿼리 ID입니다. 결과 캐싱을 사용하지 않은 경우 이 필드 값은 NULL입니다.
concurrency_scaling_status_txt	텍스트	쿼리가 기본 클러스터에서 실행되었는지 아니면 동시성 조정 클러스터에서 실행되었는지를 설명합니다.
from_sp_call	정수	쿼리가 저장 프로시저에서 호출된 경우, 프로시저 호출의 쿼리 ID입니다. 쿼리가 저장 프로시저의 일부로 실행되지 않은 경우, 이 필드는 NULL입니다.

샘플 쿼리

다음 예는 `userid = 100`인 사용자가 실행한 가장 최근의 데이터베이스 쿼리 5개의 쿼리 ID, 실행 시간, 잘린 쿼리 텍스트를 반환합니다.

```
select query, pid, elapsed, substring from svl_qlog
where userid = 100
order by starttime desc
limit 5;

 query  |  pid  | elapsed  |           substring
--------+-------+----------+-----------------------------------------------
 187752 | 18921 | 18465685 | select query, elapsed, substring from svl_...
 204168 |  5117 |    59603 | insert into testtable values (100);
 187561 | 17046 |  1003052 | select * from pg_table_def where tablename...
 187549 | 17046 |  1108584 | select * from STV_WLM_SERVICE_CLASS_CONFIG
 187468 | 17046 |  5670661 | select * from pg_table_def where schemaname...
(5 rows)
```

다음 예는 취소된(**aborted=1**) 쿼리의 SQL 스크립트 이름(LABEL 열) 및 경과 시간을 반환합니다.

```
select query, elapsed, trim(label) querylabel
from svl_qlog where aborted=1;

 query | elapsed  |       querylabel
```

```
-------+----------+-------------------------
    16 |  6935292 | alltickittablesjoin.sql
(1 row)
```

SVL_QUERY_METRICS

SVL_QUERY_METRICS 뷰는 완료된 쿼리의 지표를 보여 줍니다. 이 보기는 STL_QUERY_METRICS 시스템 테이블에서 파생됩니다. 쿼리 모니터링 규칙을 정의하기 위한 임계값을 결정하는 데 이 보기의 값들을 사용하십시오. 자세한 내용은 WLM 쿼리 모니터링 규칙 단원을 참조하십시오.

SVL_QUERY_METRICS는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_DETAIL에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 쿼리를 실행한 사용자의 ID.
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
service_class	정수	WLM 쿼리 대기열(서비스 클래스)의 ID. 쿼리 대기열은 WLM 구성에서 정의합니다. 지표는 사용자 정의 대기열에 대해서만 보고됩니다. 서비스 클래스 ID의 목록은 WLM 서비스 클래스 ID 섹션을 참조하세요.
dimension	varchar(24)	지표가 보고되는 차원. 가능한 값은 query, segment 및 step입니다.
segment	정수	세그먼트 번호. 하나의 쿼리는 여러 세그먼트로 구성되며, 각각의 세그먼트는 하나 이상의 단계로 구성됩니다. 쿼리 세그먼트는 병렬로 실행될 수 있습니다. 각 세그먼트는 단일 프로세스에서 실행됩니다. 세그먼트 값이 0인 경우, 지표 세그먼트 값은 쿼리 수준으로 롤업됩니다.

열 명칭	데이터 유형	설명
step	정수	수행된 단계 유형의 ID입니다. 단계 유형에 대한 설명은 `step_label` 열에 나와 있습니다.
step_label	varchar(30)	수행된 단계 유형입니다.
query_cpu_time	bigint	쿼리에 사용된 CPU 시간(초)입니다. CPU 시간은 쿼리 실행 시간과 구별됩니다.
query_blocks_read	bigint	쿼리가 읽은 1MB 블록의 수.
query_execution_time	bigint	쿼리를 실행하고 경과된 시간(초)입니다. 실행 시간에는 대기열에서 대기하는 데 소모한 시간은 포함되지 않습니다. 대기열에 있는 시간은 `query_queue_time` 을 참조하십시오.
query_cpu_usage_percent	bigint	쿼리에 사용된 CPU 용량의 비율입니다.
query_temp_blocks_to_disk	bigint	쿼리가 중간 결과를 쓰는 데 사용되는 디스크 공간의 양(MB).
segment_execution_time	bigint	단일 세그먼트를 실행하고 경과된 시간(초)입니다.
cpu_skew	numeric(38,2)	임의 조각의 최대 CPU 사용량과 모든 조각의 평균 CPU 사용량을 비교한 비율입니다. 이 지표는 세그먼트 수준에서 정의됩니다.
io_skew	numeric(38,2)	임의 조각에서 읽은 최대 블록 수(I/O)와 모든 조각에서 읽은 평균 블록 수를 비교한 비율입니다.
scan_row_count	bigint	스캔 단계에 포함되는 행의 수입니다. 행 개수는 삭제 대기 행(고스트 행)을 필터링하고 사용자 정의 쿼리 필터를 적용하기 전에 내보낸 행의 총 수입니다.
join_row_count	bigint	조인 단계에서 처리한 행의 수입니다.

열 명칭	데이터 유형	설명
nested_loop_join_row_count	bigint	중첩된 루프 조인에 있는 행의 수.
return_row_count	bigint	쿼리에서 반환되는 행의 수입니다.
spectrum_scan_row_count	bigint	Amazon S3에서 Amazon Redshift Spectrum이 스캔한 행 수입니다.
spectrum_scan_size_mb	bigint	Amazon S3에서 Amazon Redshift Spectrum에 의해 스캔된 데이터 양(MB)입니다.
query_queue_time	bigint	쿼리가 대기열에 있는 시간(초)입니다.

SVL_QUERY_METRICS_SUMMARY

SVL_QUERY_METRICS_SUMMARY 뷰는 완료된 쿼리의 지표 최댓값을 보여 줍니다. 이 보기는 STL_QUERY_METRICS 시스템 테이블에서 파생됩니다. 쿼리 모니터링 규칙을 정의하기 위한 임계값을 결정하는 데 이 보기의 값들을 사용하십시오. Amazon Redshift의 쿼리 모니터링 규칙 및 지표에 대한 자세한 내용은 WLM 쿼리 모니터링 규칙 섹션을 참조하세요.

SVL_QUERY_METRICS_SUMMARY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_DETAIL에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 쿼리를 실행한 사용자의 ID.

열 명칭	데이터 유형	설명
쿼리	정수	쿼리 ID. 쿼리 열을 사용하여 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
service_class	정수	WLM 쿼리 대기열(서비스 클래스)의 ID. 쿼리 대기열은 WLM 구성에서 정의합니다. 지표는 사용자 정의 대기열에 대해서만 보고됩니다. 서비스 클래스 ID의 목록은 WLM 서비스 클래스 ID 섹션을 참조하세요.
query_cpu_time	bigint	쿼리에 사용된 CPU 시간(초)입니다. CPU 시간은 쿼리 실행 시간과 구별됩니다.
query_blocks_read	bigint	쿼리가 읽은 1MB 블록의 수.
query_execution_time	bigint	쿼리를 실행하고 경과된 시간(초)입니다. 실행 시간에는 대기열에서 대기하는 데 소모한 시간은 포함되지 않습니다.
query_cpu_usage_percent	numeric(38,2)	쿼리에 사용된 CPU 용량의 비율입니다.
query_temp_blocks_to_disk	bigint	쿼리가 중간 결과를 쓰는 데 사용되는 디스크 공간의 양(MB).
segment_execution_time	bigint	단일 세그먼트를 실행하고 경과된 시간(초)입니다.
cpu_skew	numeric(38,2)	임의 조각의 최대 CPU 사용량과 모든 조각의 평균 CPU 사용량을 비교한 비율입니다. 이 지표는 세그먼트 수준에서 정의됩니다.
io_skew	numeric(38,2)	임의 조각에서 읽은 최대 블록 수(I/O)와 모든 조각에서 읽은 평균 블록 수를 비교한 비율입니다.
scan_row_count	bigint	스캔 단계에 포함되는 행의 수입니다. 행 개수는 삭제 대기 행(고스트 행)을 필터링하고 사용자 정의 쿼리 필터를 적용하기 전에 내보낸 행의 총 수입니다.

열 명칭	데이터 유형	설명
join_row_count	bigint	조인 단계에서 처리한 행의 수입니다.
nested_loop_join_row_count	bigint	중첩된 루프 조인에 있는 행의 수.
return_row_count	bigint	쿼리에서 반환되는 행의 수입니다.
spectrum_scan_row_count	bigint	Amazon S3에서 Amazon Redshift Spectrum이 스캔한 행 수입니다.
spectrum_scan_size_mb	bigint	Amazon S3에서 Amazon Redshift Spectrum에 의해 스캔된 데이터 양(MB)입니다.
query_queue_time	bigint	쿼리가 대기열에 있는 시간(초)입니다.

SVL_QUERY_QUEUE_INFO

워크로드 관리(WLM) 쿼리 대기열 또는 커밋 대기열에서 시간을 보낸 쿼리의 세부 정보를 요약합니다.

SVL_QUERY_QUEUE_INFO 보기는 시스템이 수행한 쿼리를 필터링하고 사용자가 수행한 쿼리만을 보여줍니다.

SVL_QUERY_QUEUE_INFO 뷰는 STL_QUERY, STL_WLM_QUERY 및 STL_COMMIT_STATS 시스템 테이블의 정보를 요약합니다.

SVL_QUERY_QUEUE_INFO는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
데이터베이스	텍스트	쿼리가 실행되었을 때 사용자가 연결된 데이터베이스의 이름.

열 명칭	데이터 유형	설명
쿼리	정수	쿼리 ID.
xid	bigint	트랜잭션 ID.
userid	정수	쿼리를 생성한 사용자의 ID
querytxt	텍스트	쿼리 텍스트의 첫 100자.
queue_start_time	타임스탬프	쿼리가 WLM 대기열에 진입한 UTC 시간.
exec_start_time	타임스탬프	쿼리 실행이 시작된 UTC 시간.
service_class	정수	서비스 클래스의 ID. 서비스 클래스는 WLM 구성 파일에 정의되어 있습니다.
slots	정수	WLM 쿼리 슬롯의 수.
queue_elapsed	bigint	쿼리가 WLM 대기열에서 대기하는 데 쓴 시간(초).
exec_elapsed	bigint	쿼리를 실행하는 데 소요된 시간(초).
wlm_total_elapsed	bigint	쿼리가 WLM 대기열에서 쓴 시간(queue_elapsed)에 쿼리 실행에 쓴 시간(exec_elapsed)을 합한 값.
commit_queue_elapsed	bigint	쿼리가 커밋 대기열에서 대기하는 데 쓴 시간(초).
commit_exec_time	bigint	쿼리가 커밋 작업에서 소모한 시간(초).
service_class_name	character(64)	서비스 클래스의 이름입니다.

샘플 쿼리

다음 예는 쿼리가 WLM 대기열에서 소모한 시간을 보여 줍니다.

```
select query, service_class, queue_elapsed, exec_elapsed, wlm_total_elapsed
from svl_query_queue_info
where wlm_total_elapsed > 0;

  query  | service_class | queue_elapsed | exec_elapsed | wlm_total_elapsed
---------+---------------+---------------+--------------+-------------------
 2742669 |             6 |             2 |          916 |               918
 2742668 |             6 |             4 |          197 |               201
(2 rows)
```

SVL_QUERY_REPORT

Amazon Redshift는 여러 Amazon Redshift STL 시스템 테이블의 UNION에서 SVL_QUERY_REPORT를 생성하여 완료된 쿼리 단계에 대한 정보를 제공합니다.

이 보기는 완료된 쿼리에 대한 정보를 조각 및 단계 기준으로 분류하며, Amazon Redshift 클러스터에서 노드 및 조각 문제를 해결하는 데 도움이 될 수 있습니다.

SVL_QUERY_REPORT는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_DETAIL에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID. 다양한 다른 시스템 테이블 및 보기를 조인하는 데 사용할 수 있습니다.
slice	정수	단계가 실행된 데이터 조각입니다.

열 명칭	데이터 유형	설명
segment	정수	세그먼트 번호. 하나의 쿼리는 여러 세그먼트로 구성되며, 각각의 세그먼트는 하나 이상의 단계로 구성됩니다. 쿼리 세그먼트는 병렬로 실행될 수 있습니다. 각 세그먼트는 단일 프로세스에서 실행됩니다.
step	정수	완료된 쿼리 단계입니다.
start_time	타임스탬프	세그먼트가 실행되기 시작한 정확한 UTC 시간으로 정확한 소수부 초를 나타내는 여섯 자리가 포함됩니다. 예: **2012-12-12 11:29:19.131358**
end_time	타임스탬프	세그먼트 실행이 종료된 정확한 UTC 시간으로 정확한 소수부 초를 나타내는 여섯 자리가 포함됩니다. 예: **2012-12-12 11:29:19.131467**
경과 시간	bigint	세그먼트가 실행되는 데 걸린 시간(마이크로초)입니다.
rows	bigint	단계에 의해 만들어진 행의 수(조각당). 이 수는 단계가 수신하거나 처리한 행의 수가 아니라 단계의 실행에서 발생하는 조각의 행 수를 나타냅니다. 다시 말해 이것은 단계에서 살아남아 다음 단계로 전달되는 행의 수입니다.
bytes	bigint	단계에 의해 생성되는 바이트의 수(조각당).
레이블	char(256)	쿼리 단계 이름 및 해당되는 경우, 테이블 ID와 테이블 이름으로 구성되는 단계 레이블(예: `scan tbl=100448 name =user`). 3자리 테이블 ID는 일반적으로 일시적 테이블의 스캔을 가리킵니다. `tbl=0`이 표시된다면 일반적으로 일정한 값의 스캔을 가리킵니다.
is_diskbased	character(1)	디스크 기반 작업으로서 이번 쿼리 단계의 수행 여부: true(**t**) 또는 false(**f**). 해시, 정렬, 집계 단계 같은 특정 단계만 디스크로 갈 수 있습니다. 많은 단계 형식은 항상 메모리에서 수행됩니다.
workmem	bigint	쿼리 단계에 할당된 작업 메모리의 양(바이트)입니다. 이 값은 실제로 사용된 메모리 양이 아니라 실행 중에 사용하도록 할당된 query_working_mem 임계값입니다.

열 명칭	데이터 유형	설명
is_rrscan	character (1)	true(**t**)인 경우, 단계에서 범위 제한 스캔이 사용되었음을 나타냅니다.
is_delayed_scan	character (1)	true(**t**)인 경우, 단계에서 지연된 스캔이 사용되었음을 나타냅니다.
rows_pre_filter	bigint	영구 테이블 스캔의 경우, 삭제 대기 행(고스트 행)을 필터링하고 사용자 정의 쿼리 필터를 적용하기 전에 내보낸 행의 총수.

샘플 쿼리

다음 쿼리는 쿼리 ID가 279인 쿼리의 반환된 행의 데이터 스큐를 보여 줍니다. 이 쿼리를 사용하여 데이터베이스 데이터가 데이터 웨어하우스 클러스터의 조각들에 고르게 분산되어 있는지 파악합니다.

```
select query, segment, step, max(rows), min(rows),
case when sum(rows) > 0
then ((cast(max(rows) -min(rows) as float)*count(rows))/sum(rows))
else 0 end
from svl_query_report
where query = 279
group by query, segment, step
order by segment, step;
```

이 쿼리는 다음 샘플 출력과 비슷한 데이터를 반환해야 합니다.

```
query | segment | step |   max    |   min    |         case
------+---------+------+----------+----------+----------------------
279 |       0 |    0 | 19721687 | 19721687 |                    0
279 |       0 |    1 | 19721687 | 19721687 |                    0
279 |       1 |    0 |   986085 |   986084 | 1.01411202804304e-06
279 |       1 |    1 |   986085 |   986084 | 1.01411202804304e-06
279 |       1 |    4 |   986085 |   986084 | 1.01411202804304e-06
279 |       2 |    0 |  1775517 |   788460 |     1.00098637606408
279 |       2 |    2 |  1775517 |   788460 |     1.00098637606408
279 |       3 |    0 |  1775517 |   788460 |     1.00098637606408
279 |       3 |    2 |  1775517 |   788460 |     1.00098637606408
279 |       3 |    3 |  1775517 |   788460 |     1.00098637606408
```

```
279 |       4 |    0 |  1775517 |   788460 |     1.00098637606408
279 |       4 |    1 |  1775517 |   788460 |     1.00098637606408
279 |       4 |    2 |        1 |        1 |                    0
279 |       5 |    0 |        1 |        1 |                    0
279 |       5 |    1 |        1 |        1 |                    0
279 |       6 |    0 |       20 |       20 |                    0
279 |       6 |    1 |        1 |        1 |                    0
279 |       7 |    0 |        1 |        1 |                    0
279 |       7 |    1 |        0 |        0 |                    0
(19 rows)
```

SVL_QUERY_SUMMARY

SVL_QUERY_SUMMARY 뷰를 사용하여 쿼리 실행에 대한 일반 정보를 찾습니다.

SVV_QUERY_SUMMARY 뷰에는 SVL_QUERY_REPORT 뷰의 데이터 하위 집합이 포함됩니다. SVL_QUERY_SUMMARY의 정보는 모든 노드에서 집계됩니다.

> **Note**
>
> SVL_QUERY_SUMMARY 보기에는 Amazon Redshift가 수행하는 쿼리에 대한 정보만 포함되며 다른 유틸리티 및 DDL 명령이 실행하는 쿼리의 정보는 포함되지 않습니다. DDL 및 유틸리티 명령을 포함하여 Amazon Redshift에 의해 수행되는 모든 문의 전체 목록과 정보를 보려면 SVL_STATEMENTTEXT 보기를 쿼리할 수 있습니다.

SVL_QUERY_SUMMARY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_DETAIL에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

SVCS_QUERY_SUMMARY에 대한 자세한 내용은 SVCS_QUERY_SUMMARY 섹션을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.

열 명칭	데이터 유형	설명
쿼리	정수	쿼리 ID. 다양한 다른 시스템 테이블 및 보기를 조인하는 데 사용할 수 있습니다.
stm	정수	스트림: 한 쿼리 내의 동시 세그먼트의 집합. 쿼리에는 하나 이상의 스트림이 있습니다.
seg	정수	세그먼트 번호. 하나의 쿼리는 여러 세그먼트로 구성되며, 각각의 세그먼트는 하나 이상의 단계로 구성됩니다. 쿼리 세그먼트는 병렬로 실행될 수 있습니다. 각 세그먼트는 단일 프로세스에서 실행됩니다.
step	정수	실행된 쿼리 단계입니다.
maxtime	bigint	단계가 실행되기 위한 최대 시간(마이크로초)입니다.
avgtime	bigint	단계가 실행되기 위한 평균 시간(마이크로초)입니다.
rows	bigint	쿼리 단계에 연관된 데이터 행의 수.
bytes	bigint	쿼리 단계에 연관된 데이터 바이트의 수.
rate_row	double precision	행당 쿼리 실행 속도.
rate_byte	double precision	바이트당 쿼리 실행 속도.
레이블	텍스트	쿼리 단계 이름 및 해당되는 경우, 테이블 ID와 테이블 이름으로 구성되는 단계 레이블(예: scan tbl=100448 name =user). 3자리 테이블 ID는 일반적으로 일시적 테이블의 스캔을 가리킵니다. `tbl=0`이 표시된다면 일반적으로 일정한 값의 스캔을 가리킵니다.
is_diskbased	character(1)	쿼리의 이 단계가 클러스터의 노드에서 디스크 기반 작업으로 수행되었는지 여부입니다(true(**t**) 또는 false(**f**)). 해시, 정렬, 집계 단계 같은 특정 단계만 디스크로 갈 수 있습니다. 많은 단계 형식은 항상 메모리에서 수행됩니다.
workmem	bigint	쿼리 단계에 할당된 작업 메모리의 양(바이트)입니다.

열 명칭	데이터 유형	설명
is_rrscan	character(1)	true(**t**)인 경우, 단계에서 범위 제한 스캔이 사용되었음을 나타냅니다. 기본값은 false(**f**)입니다.
is_delayed_scan	character(1)	true(**t**)인 경우, 단계에서 지연된 스캔이 사용되었음을 나타냅니다. 기본값은 false(**f**)입니다.
rows_pre_filter	bigint	영구 테이블 스캔의 경우, 삭제 대기 행(고스트 행)을 필터링하기 전에 내보낸 행의 총수.

샘플 쿼리

쿼리 단계의 처리 정보 확인

다음 쿼리는 쿼리 87의 각 단계에 대한 기본적 처리 정보를 보여 줍니다.

```
select query, stm, seg, step, rows, bytes
from svl_query_summary
where query = 87
order by query, seg, step;
```

이 쿼리는 다음 샘플 출력에서 보듯 쿼리 87에 대한 처리 정보를 검색합니다.

```
 query | stm | seg | step |  rows  |  bytes
-------+-----+-----+------+--------+---------
87     |   0 |   0 |    0 |     90 |    1890
87     |   0 |   0 |    2 |     90 |     360
87     |   0 |   1 |    0 |     90 |     360
87     |   0 |   1 |    2 |     90 |    1440
87     |   1 |   2 |    0 | 210494 | 4209880
87     |   1 |   2 |    3 |  89500 |       0
87     |   1 |   2 |    6 |      4 |      96
87     |   2 |   3 |    0 |      4 |      96
87     |   2 |   3 |    1 |      4 |      96
87     |   2 |   4 |    0 |      4 |      96
87     |   2 |   4 |    1 |      1 |      24
87     |   3 |   5 |    0 |      1 |      24
87     |   3 |   5 |    4 |      0 |       0
(13 rows)
```

쿼리 단계가 디스크에 분산되었는지 여부 확인

다음 쿼리는 쿼리 ID 1025(쿼리의 쿼리 ID를 얻는 방법을 확인하려면 SVL_QLOG 뷰를 참조)인 쿼리의 단계가 디스크로 분산되었는지 또는 쿼리가 전적으로 메모리에서 실행되었는지 여부를 보여 줍니다.

```
select query, step, rows, workmem, label, is_diskbased
from svl_query_summary
where query = 1025
order by workmem desc;
```

위 쿼리는 다음과 같은 샘플 출력을 반환합니다.

```
query| step|  rows   |  workmem    |  label         | is_diskbased
-----+-----+--------+-----------+---------------+--------------
1025 |  0  |16000000|  141557760 |scan tbl=9     | f
1025 |  2  |16000000|  135266304 |hash tbl=142   | t
1025 |  0  |16000000|  128974848 |scan tbl=116536| f
1025 |  2  |16000000|  122683392 |dist           | f
(4 rows)
```

IS_DISKBASED의 값을 스캔하면 어떤 쿼리 단계가 디스크로 갔는지 알 수 있습니다. 쿼리 1025의 경우, 해시 단계가 디스크에서 실행되었습니다. 디스크에서 실행될 수 있는 단계에는 hash, aggr, sort 단계가 포함됩니다. 디스크 기반 쿼리 단계만 보려면 위의 예에서 **`and is_diskbased = 't'`** 절을 SQL 문에 추가합니다.

SVL_RESTORE_ALTER_TABLE_PROGRESS

RA3 노드로의 클래식 크기 조정 중에 클러스터 내 각 테이블의 마이그레이션 진행 상황을 모니터링하려면 SVL_RESTORE_ALTER_TABLE_PROGRESS를 사용합니다. 이는 크기 조정 작업 중 데이터 마이그레이션의 과거 처리량을 캡처합니다. RA3 노드로의 클래식 크기 조정에 대한 자세한 내용은 클래식 크기 조정에서 확인하세요.

SVL_RESTORE_ALTER_TABLE_PROGRESS는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_RESTORE_LOG에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

> **Note**
>
> 진행 상황이 100.00% 또는 ABORTED인 행은 7일 후에 삭제됩니다. 클래식 크기 조정 도중 또는 이후에 삭제된 테이블의 행이 여전히 SVL_RESTORE_ALTER_TABLE_PROGRESS에서 나타날 수 있습니다.

테이블 열

열 명칭	데이터 유형	설명
tbl	정수	테이블의 ID입니다.
progress	char(32)	테이블의 재배포 진행 상태입니다. 가능한 값은 0.00%~100.00%의 백분율 및 ABORTED라는 메시지입니다. ABORTED는 재배포가 완료되지 않고 중단되었음을 의미합니다. 이유는 message 열에 설명되어 있습니다.
message	char(256)	테이블의 재배포 진행 상황과 관련된 메시지입니다.

샘플 쿼리

다음 쿼리는 실행 중인 쿼리와 대기 중인 쿼리를 반환합니다.

```
select * from svl_restore_alter_table_progress;

tbl     | progress |                            message
--------+----------+-----------------------------------------------------------
105614  | ABORTED  | Abort:Table no longer contains the prior dist key column.
105610  | ABORTED  | Abort:Table no longer contains the prior dist key column.
105594  | 0.00%    | Table waiting for alter diststyle conversion.
105602  | ABORTED  | Abort:Table no longer contains the prior dist key column.
105606  | ABORTED  | Abort:Table no longer contains the prior dist key column.
105598  | 100.00%  | Restored to distkey successfully.
```

SVL_S3LIST

세그먼트 수준에서 Amazon Redshift Spectrum 쿼리에 대한 세부 정보를 가져오려면 SVL_S3LIST 뷰를 사용합니다.

SVL_S3LIST는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> SVL_S3LIST에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_EXTERNAL_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
쿼리	정수	쿼리 ID입니다.
segment	정수	세그먼트 번호. 쿼리는 여러 세그먼트로 구성됩니다.
노드	정수	노드 번호.
slice	정수	특정 세그먼트가 실행된 데이터 조각입니다.
eventtime	타임스탬프	이벤트가 기록된 시간(UTC)입니다.
버킷	텍스트	Amazon S3 버킷 이름입니다.
접두사	텍스트	Amazon S3 버킷 위치의 접두사입니다.
recursive	char(1)	하위 폴더에 대한 반복 스캔 여부입니다.
retrieved_files	정수	나열된 파일 수입니다.
max_file_size	bigint	나열된 파일의 최대 파일 크기입니다.

열 명칭	데이터 유형	설명
avg_file_size	double precision	나열된 파일의 평균 파일 크기입니다.
generated_splits	정수	파일 분할의 수입니다.
avg_split_length	double precision	파일 분할의 평균 길이(바이트)입니다.
duration	bigint	파일 목록의 기간(마이크로초)입니다.

샘플 쿼리

다음 예에서는 마지막으로 실행된 쿼리에 대해 SVL_S3LIST를 쿼리합니다.

```
select *
from svl_s3list
where query = pg_last_query_id()
order by query,segment;
```

SVL_S3LOG

세그먼트 및 노드 조각 수준에서 Amazon Redshift Spectrum 쿼리에 대한 세부 정보를 가져오려면 SVL_S3LOG 뷰를 사용합니다.

SVL_S3LOG는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> SVL_S3LOG에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_EXTERNAL_QUERY_DETAIL을 사용하

는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
pid	정수	프로세스 ID.
쿼리	정수	쿼리 ID입니다.
segment	정수	세그먼트 번호. 하나의 쿼리는 여러 세그먼트로 구성되며, 각각의 세그먼트는 하나 이상의 단계로 구성됩니다.
step	정수	실행된 쿼리 단계입니다.
노드	정수	노드 번호.
slice	정수	특정 세그먼트가 실행된 데이터 조각입니다.
eventtime	타임스탬프	단계가 실행되기 시작한 UTC 시간.
message	텍스트	로그 항목에 대한 메시지.

샘플 쿼리

다음 예에서는 마지막으로 실행된 쿼리에 대해 SVL_S3LOG를 쿼리합니다.

```
select *
from svl_s3log
where query = pg_last_query_id()
order by query,segment,slice;
```

SVL_S3PARTITION

세그먼트 및 노드 조각 수준에서 Amazon Redshift Spectrum 파티션에 대한 세부 정보를 가져오려면 SVL_S3PARTITION 뷰를 사용합니다.

SVL_S3PARTITION은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> SVL_S3PARTITION에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에 대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_EXTERNAL_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
쿼리	정수	쿼리 ID입니다.
segment	정수	세그먼트 번호. 하나의 쿼리는 여러 세그먼트로 구성되며, 각각의 세그먼트는 하나 이상의 단계로 구성됩니다.
노드	정수	노드 번호.
slice	정수	특정 세그먼트가 실행된 데이터 조각입니다.
starttime	시간대 미포함 TIMESTAMP	파티션 잘라내기 실행이 시작된 시간(UTC)
endtime	시간대 미포함 TIMESTAMP	파티션 잘라내기가 완료된 시간(UTC)
duration	bigint	경과 시간(마이크로초).
total_partitions	정수	총 파티션의 수.

열 명칭	데이터 유형	설명
qualified_partitions	정수	적격 파티션의 수.
assigned_partitions	정수	조각에 할당된 파티션의 수.
assignment	character	할당 유형.

샘플 쿼리

다음 예는 마지막으로 완료된 쿼리에 대한 파티션 세부 정보를 가져옵니다.

```
SELECT query, segment,
       MIN(starttime) AS starttime,
       MAX(endtime) AS endtime,
       datediff(ms,MIN(starttime),MAX(endtime)) AS dur_ms,
       MAX(total_partitions) AS total_partitions,
       MAX(qualified_partitions) AS qualified_partitions,
       MAX(assignment) as assignment_type
FROM svl_s3partition
WHERE query=pg_last_query_id()
GROUP BY query, segment
```

```
query | segment |           starttime           |           endtime           | dur_ms|
 total_partitions | qualified_partitions | assignment_type
------+---------+-------------------------------+-----------------------------+-------
+------------------+----------------------+----------------
99232 |       0 | 2018-04-17 22:43:50.201515    | 2018-04-17 22:43:54.674595  |  4473 |
       2526       |        334           | p
```

SVL_S3PARTITION_SUMMARY

세그먼트 수준에서 Redshift Spectrum 쿼리 파티션 처리에 대한 요약을 가져오려면 SVL_S3PARTITION_SUMMARY 뷰를 사용합니다.

SVL_S3PARTITION_SUMMARY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

SVCS_S3PARTITION에 대한 자세한 내용은 SVCS_S3PARTITION_SUMMARY 섹션을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
쿼리	정수	쿼리 ID입니다. 이 값을 사용하여 다양한 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
segment	정수	세그먼트 번호. 쿼리는 여러 세그먼트로 구성됩니다.
assignment	char(1)	노드를 통한 파티션 할당의 유형입니다.
min_starttime	타임스탬프	파티션 처리가 시작된 시간(UTC)입니다.
max_endtime	타임스탬프	파티션 처리가 완료된 시간(UTC)입니다.
min_duration	bigint	이 쿼리에 노드가 사용하는 최소 파티션 처리 시간(마이크로초)입니다.
max_duration	bigint	이 쿼리에 노드가 사용하는 최대 파티션 처리 시간(마이크로초)입니다.
avg_duration	bigint	이 쿼리에 노드가 사용하는 평균 파티션 처리 시간(마이크로초)입니다.
total_partitions	정수	외부 테이블의 총 파티션 수입니다.
qualified_partitions	정수	적격 파티션의 총 수입니다.

열 명칭	데이터 유형	설명
min_assigned_partitions	정수	한 개의 노드에 할당된 최소 파티션 수입니다.
max_assigned_partitions	정수	한 개의 노드에 할당된 최대 파티션 수입니다.
avg_assigned_partitions	bigint	한 개의 노드에 할당된 평균 파티션 수입니다.

샘플 쿼리

다음 예는 마지막으로 완료된 쿼리에 대한 파티션 스캔 세부 정보를 가져옵니다.

```
select query, segment, assignment, min_starttime, max_endtime, min_duration,
 avg_duration
from svl_s3partition_summary
where query = pg_last_query_id()
order by query,segment;
```

SVL_S3QUERY

세그먼트 및 노드 조각 수준에서 Amazon Redshift Spectrum 쿼리에 대한 세부 정보를 가져오려면 SVL_S3QUERY 뷰를 사용합니다.

SVL_S3QUERY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

> **Note**
>
> SVL_S3QUERY에는 기본 프로비저닝된 클러스터에서 실행되는 쿼리만 포함됩니다. 동시성 크기 조정 클러스터 또는 서버리스 네임스페이스에서 실행되는 쿼리는 포함되지 않습니다. 기본 클러스터, 동시성 크기 조정 클러스터, 서버리스 네임스페이스 모두에서 실행되는 쿼리에

대한 설명 계획에 액세스하려면 SYS 모니터링 뷰인 SYS_EXTERNAL_QUERY_DETAIL을 사용하는 것이 좋습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	지정된 항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID입니다.
segment	정수	세그먼트 번호. 하나의 쿼리는 여러 세그먼트로 구성되며, 각각의 세그먼트는 하나 이상의 단계로 구성됩니다.
step	정수	실행된 쿼리 단계입니다.
노드	정수	노드 번호.
slice	정수	특정 세그먼트가 실행된 데이터 조각입니다.
starttime	타임스탬프	쿼리가 실행되기 시작한 UTC 시간.
endtime	타임스탬프	쿼리 실행이 완료된 UTC 시간
elapsed	정수	경과 시간(마이크로초).
external_table_name	char(136)	s3 스캔 단계의 외부 테이블 이름의 내부 형식.
is_partitioned	char(1)	true(**t**)인 경우, 이 열 값은 외부 테이블이 파티셔닝되어 있음을 나타냅니다.
is_rrscan	char(1)	true(**t**)인 경우, 이 열 값은 범위 제한 스캔이 적용되었음을 나타냅니다.
s3_scanned_rows	bigint	Amazon S3에서 스캔되어 Redshift Spectrum 계층으로 전송된 행의 수.

열 명칭	데이터 유형	설명
s3_scanned_bytes	bigint	Amazon S3에서 스캔되어 Redshift Spectrum 계층으로 전송된 바이트의 수.
s3query_returned_rows	bigint	Redshift Spectrum 계층에서 클러스터로 반환된 행의 수.
s3query_returned_bytes	bigint	Redshift Spectrum 계층에서 클러스터로 반환된 바이트의 수.
files	정수	이 조각에서 이 S3 스캔 단계를 위해 처리된 파일 수.
splits	int	이 조각에서 처리되는 분할 수입니다. 분할할 수 있는 큰 데이터 파일의 경우, 예를 들어 약 512MB보다 큰 데이터 파일의 경우 Redshift Spectrum은 병렬 처리를 위해 파일을 여러 개의 S3 요청으로 분할하려고 합니다.
total_split_size	bigint	이 조각에서 처리되는 모든 분할의 총 크기(바이트)입니다.
max_split_size	bigint	이 조각에서 처리되는 최대 분할 크기(바이트)입니다.
total_retries	정수	처리된 파일의 총 재시도 횟수.
max_retries	정수	처리된 개별 파일의 최대 재시도 횟수.
max_request_duration	정수	개별 Redshift Spectrum 요청의 최대 지속 시간(마이크로초).
avg_request_duration	배정밀도	Redshift Spectrum 요청의 평균 지속 시간(마이크로초).

열 명칭	데이터 유형	설명
max_request_parallelism	정수	이 S3 스캔 단계를 위해 이 조각에서 대기 중인 Redshift Spectrum 요청의 최대 개수.
avg_request_parallelism	double precision	이 S3 스캔 단계를 위한 이 조각에서의 병렬 Redshift Spectrum 요청의 평균 개수.

샘플 쿼리

다음 예는 마지막으로 완료된 쿼리에 대한 스캔 단계 세부 정보를 가져옵니다.

```
select query, segment, slice, elapsed, s3_scanned_rows, s3_scanned_bytes,
 s3query_returned_rows, s3query_returned_bytes, files
from svl_s3query
where query = pg_last_query_id()
order by query,segment,slice;
```

```
query | segment | slice | elapsed | s3_scanned_rows | s3_scanned_bytes |
 s3query_returned_rows | s3query_returned_bytes | files
------+---------+-------+---------+-----------------+------------------
+----------------------+-------------------------+------
 4587 |       2 |     0 |   67811 |               0 |                0 |
      0 |                      0 |     0
 4587 |       2 |     1 |  591568 |          172462 |         11260097 |
   8513 |                  170260 |     1
 4587 |       2 |     2 |  216849 |               0 |                0 |
      0 |                      0 |     0
 4587 |       2 |     3 |  216671 |               0 |                0 |
      0 |                      0 |     0
```

SVL_S3QUERY_SUMMARY

시스템에서 실행된 모든 Amazon Redshift Spectrum 쿼리(S3 쿼리)의 요약을 가져오려면 SVL_S3QUERY_SUMMARY 뷰를 사용합니다. SVL_S3QUERY_SUMMARY는 세그먼트 수준에서 SVL_S3QUERY의 세부 정보를 집계합니다.

SVL_S3QUERY_SUMMARY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_EXTERNAL_QUERY_DETAIL에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

SVCS_S3QUERY_SUMMARY에 대한 자세한 내용은 SVCS_S3QUERY_SUMMARY 섹션을 참조하세요.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	지정된 항목을 생성한 사용자의 ID.
쿼리	정수	쿼리 ID입니다. 이 값을 사용하여 다양한 다른 시스템 테이블 및 뷰를 조인할 수 있습니다.
xid	bigint	트랜잭션 ID.
pid	정수	프로세스 ID.
segment	정수	세그먼트 번호. 하나의 쿼리는 여러 세그먼트로 구성되며, 각각의 세그먼트는 하나 이상의 단계로 구성됩니다.
step	정수	실행된 쿼리 단계입니다.
starttime	타임스탬프	쿼리가 실행되기 시작한 UTC 시간.
endtime	타임스탬프	쿼리가 완료된 UTC 시간.
elapsed	정수	쿼리가 실행되는 데 걸린 시간 길이(마이크로초).
aborted	정수	쿼리가 시스템에 의해 중지되거나 사용자에 의해 취소되는 경우, 이 열에 **1**이 포함됩니다. 쿼리가 실행되어 완료되면 이 열에 **0**이 포함됩니다.

열 명칭	데이터 유형	설명
external_table_name	char(136)	외부 테이블 스캔을 위한 테이블 외부 이름의 내부 형식입니다.
file_format	character(16)	외부 테이블 데이터의 파일 형식입니다.
is_partitioned	char(1)	true(**t**)인 경우, 이 열 값은 외부 테이블이 파티셔닝되어 있음을 나타냅니다.
is_rrscan	char(1)	true(**t**)인 경우, 이 열 값은 범위 제한 스캔이 적용되었음을 나타냅니다.
is_nested	char(1)	true(**t**)인 경우 이 열 값은 중첩 열 데이터 형식에 액세스했음을 나타냅니다.
s3_scanned_rows	bigint	Amazon S3에서 스캔되어 Redshift Spectrum 계층으로 전송된 행의 수.
s3_scanned_bytes	bigint	압축된 데이터를 기반으로 Amazon S3에서 스캔되어 Redshift Spectrum 계층으로 전송된 바이트의 수.
s3query_returned_rows	bigint	Redshift Spectrum 계층에서 클러스터로 반환된 행의 수.
s3query_returned_bytes	bigint	Redshift Spectrum 계층에서 클러스터로 반환된 바이트의 수. Amazon Redshift로 반환되는 데이터의 양이 많으면 시스템 성능이 영향을 받을 수 있습니다.
files	정수	이 Redshift Spectrum 쿼리에 대해 처리된 파일의 수. 파일 수가 적으면 병렬 처리의 이점이 제한됩니다.
files_max	정수	한 조각에서 처리된 파일의 최대 개수입니다.
files_avg	정수	한 조각에서 처리된 파일의 평균 개수입니다.

열 명칭	데이터 유형	설명
splits	int	이 세그먼트에 대해 처리되는 분할 수입니다. 이 조각에서 처리되는 분할 수입니다. 분할할 수 있는 큰 데이터 파일의 경우, 예를 들어 약 512MB보다 큰 데이터 파일의 경우 Redshift Spectrum은 병렬 처리를 위해 파일을 여러 개의 S3 요청으로 분할하려고 합니다.
splits_max	int	이 조각에서 처리되는 최대 분할 수입니다.
splits_avg	int	이 조각에서 처리되는 평균 분할 수입니다.
total_split_size	bigint	처리되는 모든 분할의 총 크기입니다.
max_split_size	bigint	처리되는 최대 분할 크기(바이트)입니다.
avg_split_size	bigint	처리되는 평균 분할 크기(바이트)입니다.
total_retries	정수	처리된 하나의 개별 파일에 대한 총 재시도 횟수.
max_retries	정수	처리된 임의의 파일에 대한 최대 재시도 횟수.
max_request_duration	정수	개별 파일 요청의 최대 지속 시간(마이크로초). 오랫동안 실행 중인 쿼리는 병목 현상을 나타낼 수 있습니다.
avg_request_duration	배정밀도	파일 요청의 평균 지속 시간(마이크로초).
max_request_parallelism	정수	이 Redshift Spectrum 쿼리에 대해 한 개의 조각에 있는 최대 병렬 요청 수입니다.

열 명칭	데이터 유형	설명
avg_request_parallelism	double precision	이 Redshift Spectrum 쿼리에 대해 한 개의 조각에 있는 평균 병렬 요청 수입니다.
total_slowdown_count	bigint	외부 테이블 스캔 중에 속도가 느려지는 오류가 발생한 총 Amazon S3 요청 수입니다.
max_slowdown_count	정수	한 조각의 외부 테이블 스캔 중에 속도가 느려지는 오류가 발생한 최대 Amazon S3 요청 수입니다.

샘플 쿼리

다음 예는 마지막으로 완료된 쿼리에 대한 스캔 단계 세부 정보를 가져옵니다.

```
select query, segment, elapsed, s3_scanned_rows, s3_scanned_bytes,
 s3query_returned_rows, s3query_returned_bytes, files
from svl_s3query_summary
where query = pg_last_query_id()
order by query,segment;
```

```
query | segment | elapsed | s3_scanned_rows | s3_scanned_bytes | s3query_returned_rows
 | s3query_returned_bytes | files
------+---------+---------+-----------------+------------------+----------------------
+------------------------+------
 4587 |       2 |   67811 |               0 |                0 |                    0
 |                      0 |     0
 4587 |       2 |  591568 |          172462 |         11260097 |                 8513
 |                 170260 |     1
 4587 |       2 |  216849 |               0 |                0 |                    0
 |                      0 |     0
 4587 |       2 |  216671 |               0 |                0 |                    0
 |                      0 |     0
```

SVL_S3RETRIES

SVL_S3RETRIES 뷰를 사용하여 Amazon S3 기반 Amazon Redshift Spectrum 쿼리가 실패한 이유에 대한 정보를 가져옵니다.

SVL_S3RETRIES는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
쿼리	정수	쿼리 ID입니다.
segment	정수	세그먼트 번호. 하나의 쿼리는 여러 세그먼트로 구성되며, 각각의 세그먼트는 하나 이상의 단계로 구성됩니다. 쿼리 세그먼트는 병렬로 실행될 수 있습니다. 각 세그먼트는 단일 프로세스에서 실행됩니다.
노드	정수	노드 번호.
slice	정수	특정 세그먼트가 실행된 데이터 조각입니다.
eventtime	시간대 미포함 TIMESTAMP	단계가 실행되기 시작한 UTC 시간.
retries	정수	쿼리에 대한 재시도 횟수입니다.
successful_fetches	정수	데이터가 반환된 횟수
file_size	bigint	이 파일의 크기(바이트)입니다.
location	텍스트	테이블의 위치

열 명칭	데이터 유형	설명		
message	텍스트	오류 메시지입니다.		

샘플 쿼리

다음 예는 실패한 S3 쿼리에 대한 데이터를 검색합니다.

```
SELECT svl_s3retries.query, svl_s3retries.segment, svl_s3retries.node,
 svl_s3retries.slice, svl_s3retries.eventtime, svl_s3retries.retries,
svl_s3retries.successful_fetches, svl_s3retries.file_size,
 btrim((svl_s3retries."location")::text) AS "location",
 btrim((svl_s3retries.message)::text)
AS message FROM svl_s3retries;
```

SVL_SPATIAL_SIMPLIFY

COPY 명령을 사용하여 단순화된 공간 지오메트리 객체에 대한 정보를 가져오기 위해 시스템 뷰 SVL_SPATIAL_SIMPLIFY를 쿼리할 수 있습니다. shapefile에서 COPY를 사용할 때 SIMPLIFY `tolerance`, SIMPLIFY AUTO 및 SIMPLIFY AUTO `max_tolerance` 수집 옵션을 지정할 수 있습니다. 단순화 결과는 SVL_SPATIAL_SIMPLIFY 시스템 뷰에 요약되어 있습니다.

SIMPLIFY AUTO `max_tolerance`가 설정되면 이 뷰에는 최대 크기를 초과한 각 지오메트리에 대한 행이 포함됩니다. SIMPLIFY `tolerance`가 설정되면 전체 COPY 작업에 대한 하나의 행이 저장됩니다. 이 행은 COPY 쿼리 ID와 지정된 단순화 허용치를 참조합니다.

SVL_SPATIAL_SIMPLIFY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_SPATIAL_SIMPLIFY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
쿼리	정수	이 행을 생성한 쿼리(COPY 명령)의 ID입니다.
line_number	정수	COPY SIMPLIFY AUTO 옵션이 지정되면 이 값은 shapefile에서 단순화된 레코드의 레코드 번호입니다.
maximum_tolerance	double	COPY 명령에 지정된 거리 허용치 값입니다. 이는 SIMPLIFY AUTO 옵션을 사용하는 최대 허용치 값이거나 SIMPLIFY 옵션을 사용하는 고정 허용치 값입니다.
initial_size	정수	단순화 전 GEOMETRY 데이터 값의 크기(바이트)입니다.
simplified	char(1)	COPY SIMPLIFY AUTO 옵션이 지정된 경우 지오메트리가 성공적으로 단순화되면 t이고, 그렇지 않으면 f입니다. 주어진 최대 허용치로 단순화한 후에도 지오메트리의 크기가 여전히 최대 지오메트리 크기보다 큰 경우 지오메트리가 성공적으로 단순화되지 않을 수 있습니다.
final_size	정수	COPY SIMPLIFY AUTO 옵션이 지정된 경우 단순화 후 지오메트리의 크기(바이트)입니다.
final_tolerance	double	

샘플 쿼리

다음 쿼리는 COPY가 단순화한 레코드 목록을 반환합니다.

```
SELECT * FROM svl_spatial_simplify WHERE query = pg_last_copy_id();
 query | line_number | maximum_tolerance | initial_size | simplified | final_size |
 final_tolerance
-------+-------------+-------------------+--------------+------------+------------
+----------------------
    20 |     1184704 |                -1 |      1513736 | t          |    1008808 |
 1.276386653895e-05
```

```
    20 |     1664115 |               -1 |      1233456 | t          |    1023584 |
6.11707814796635e-06
```

SVL_SPECTRUM_SCAN_ERROR

시스템 보기 SVL_SPECT_SCAN_ERROR을 쿼리하여 Redshift Spectrum 스캔 오류에 대한 정보를 가져올 수 있습니다.

SVL_SPECT_SCAN_ERROR은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_EXTERNAL_QUERY_ERROR에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

로그된 오류의 샘플을 표시합니다. 기본값은 쿼리당 10개 항목입니다.

열 명칭	데이터 유형	설명
userid	정수	이 행을 생성한 사용자의 ID입니다.
쿼리	정수	이 행을 생성한 쿼리의 ID입니다.
location	character(128)	쿼리되는 데이터의 위치입니다.
rowid	character(128)	파일의 오류 위치입니다. `rowid` 부분은 `:`(콜론)으로 구분되며 기타 부분은 향후에 추가할 수 있습니다. `row_offset :row_group :row_id` row_offset은 파일 내 행의 오프셋(바이트)이며 지원되지 않는 파일 형식의 경우 -1입니다. 테이블은 row_groups로 구분되며 각 그룹에는 고유한 row_ids를 가진 행이 있습니다.
colname	character(128)	쿼리에서 반환되는 열의 이름입니다.

열 명칭	데이터 유형	설명
original_value	character(128)	쿼리되는 원래 값입니다.
modified_value	character(128)	쿼리에 지정된 데이터 처리 구성 옵션에 따라 반환되는 수정된 값입니다.
트리거	character(128)	쿼리에 지정된 데이터 처리 옵션입니다.
작업	character(128)	쿼리에 지정된 데이터 처리 옵션과 연결된 작업입니다.
action_value	character(128)	쿼리에 지정된 데이터 처리 옵션과 연결된 작업 파라미터의 값입니다.
error_code	정수	쿼리에 지정된 데이터 처리 옵션의 결과 코드입니다.

샘플 쿼리

다음 쿼리에서는 데이터 처리 작업이 수행된 행의 목록을 반환합니다.

```
SELECT * FROM svl_spectrum_scan_error;
```

쿼리는 다음과 비슷한 결과를 반환합니다.

```
  userid  query     location                                    rowid   colname
         original_value            modified_value       trigger          action
                 action_valueerror_code
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:0     league_name
         Barclays Premier League    Barclays Premier Lea UNSPECIFIED      TRUNCATE
                                    156
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:0     league_nspi
          34595                     32767                UNSPECIFIED
OVERFLOW_VALUE                                  199
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:1     league_nspi
          34151                     32767                UNSPECIFIED
OVERFLOW_VALUE                                  199
```

```
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:2     league_name
         Barclays Premier League    Barclays Premier Lea UNSPECIFIED      TRUNCATE
                                    156
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:2     league_nspi
          33223                     32767               UNSPECIFIED
OVERFLOW_VALUE                                  199
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:3     league_name
         Barclays Premier League    Barclays Premier Lea UNSPECIFIED      TRUNCATE
                                    156
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:3     league_nspi
          32808                     32767               UNSPECIFIED
OVERFLOW_VALUE                                  199
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:4     league_nspi
          32790                     32767               UNSPECIFIED
OVERFLOW_VALUE                                  199
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:5     league_name
         Spanish Primera Division   Spanish Primera Divi UNSPECIFIED      TRUNCATE
                                    156
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:6     league_name
         Spanish Primera Division   Spanish Primera Divi UNSPECIFIED      TRUNCATE
                                    156
```

SVL_STATEMENTTEXT

시스템에서 실행된 모든 SQL 명령의 완전한 레코드를 가져오려면 SVL_STATEMENTTEXT 뷰를 사용합니다.

SVL_STATEMENTTEXT 뷰에는 STL_DDLTEXT, STL_QUERYTEXT와 STL_UTILITYTEXT 테이블의 모든 행의 조합이 포함됩니다. 이 뷰에는 STL_QUERY 테이블에 대한 조인도 포함됩니다.

SVL_STATEMENTTEXT는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	항목을 생성한 사용자의 ID.
xid	bigint	문에 연결된 트랜잭션 ID.
pid	정수	문의 프로세스 ID.
레이블	character(320)	쿼리 실행에 사용되는 파일의 이름 또는 SET QUERY_GROUP 명령을 사용하여 정의되는 레이블. 쿼리가 파일 기반이 아니거나 QUERY_GROUP 파라미터가 설정되지 않은 경우, 이 필드의 값은 공백입니다.
starttime	타임스탬프	문이 실행되기 시작한 정확한 UTC 시간으로 정확한 소수부 초를 나타내는 여섯 자리가 포함됩니다. 예: **2009-06-12 11:29:19.131358**
endtime	타임스탬프	문이 실행이 종료된 정확한 UTC 시간으로 정확한 소수부 초를 나타내는 여섯 자리가 포함됩니다. 예: **2009-06-12 11:29:19.193640**
SEQUENCE	정수	단일 문에 200자 이상이 포함된 경우, 해당 문에 대해 추가 행이 기록됩니다. 시퀀스 0이 첫 번째 행이고 1이 두 번째 행이 되는 방식입니다.
type	varchar(10)	SQL 문의 유형: **QUERY**, **DDL** 또는 **UTILITY**.
텍스트	character(200)	200자씩 증가하는 SQL 텍스트. 이 필드에는 백슬래시(\\) 및 줄 바꿈(\n) 등의 특수 문자가 포함될 수 있습니다.

샘플 쿼리

다음 쿼리는 2009년 6월 16일에 실행된 DDL 문을 반환합니다.

```
select starttime, type, rtrim(text) from svl_statementtext
where starttime like '2009-06-16%' and type='DDL' order by starttime asc;

starttime                  | type |              rtrim
```

```
--------------------------|------|-------------------------------
2009-06-16 10:36:50.625097 | DDL  | create table ddltest(c1 int);
2009-06-16 15:02:16.006341 | DDL  | drop view alltickitjoin;
2009-06-16 15:02:23.65285  | DDL  | drop table sales;
2009-06-16 15:02:24.548928 | DDL  | drop table listing;
2009-06-16 15:02:25.536655 | DDL  | drop table event;
...
```

저장된 SQL 재구성

SVL_STATEMENTTEXT의 `text` 열에 저장된 SQL을 재구성하려면 SELECT 문을 실행하여 `text` 열의 1개 이상의 부분에서 SQL을 생성합니다. 재구성된 SQL을 실행하기 전에 모든 (\n) 특수 문자를 줄 바꿈으로 바꿉니다. 다음 SELECT 문의 결과는 `query_statement` 필드에서 재구성된 SQL의 행입니다.

```
select LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END, '')
 within group (order by sequence) AS query_statement
from SVL_STATEMENTTEXT where pid=pg_backend_pid();
```

예를 들어 다음 쿼리는 열 3개를 선택합니다. 쿼리 자체는 200자 이상이며 SVL_STATEMENTTEXT에 여러 부분으로 저장됩니다.

```
select
1 AS a01234567890123456789012345678901234567890123456789012345678901234567890,
2 AS b01234567890123456789012345678901234567890123456789012345678901234567890,
3 AS b012345678901234567890123456789012345678901234
FROM stl_querytext;
```

이 예제에서 이 쿼리는 SVL_STATEMENTTEXT의 `text` 열에 많은 두 부분(행)으로 저장됩니다.

```
select sequence, text from SVL_STATEMENTTEXT where pid = pg_backend_pid() order by
 starttime, sequence;
```

```
 sequence |
                text

----------
+----------------------------------------------------------------------------------------------------
        0 | select\n1 AS
 a01234567890123456789012345678901234567890123456789012345678901234567890,\n2 AS
```

```
 b01234567890123456789012345678901234567890123456789012345678901234567890,\n3 AS
 b01234567890123456789012345678901234567890123456789012345678901234
        1 | \nFROM stl_querytext;
```

STL_STATEMENTTEXT에 저장된 SQL을 재구성하려면 다음 SQL을 실행합니다.

```
select LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END, '')
 within group (order by sequence) AS text
from SVL_STATEMENTTEXT where pid=pg_backend_pid();
```

클라이언트에서 재구성된 SQL을 사용하려면 모든 (\n) 특수 문자를 줄 바꿈으로 바꿉니다.

```
                     text

-----------------------------------------------------------------------------------------------------------
 select\n1 AS a01234567890123456789012345678901234567890123456789012345678901234567890,
\n2 AS b01234567890123456789012345678901234567890123456789012345678901234567890,\n3 AS
 b01234567890123456789012345678901234567890123456789012345678901234\nFROM stl_querytext;
```

SVL_STORED_PROC_CALL

시스템 보기 SVL_STORED_PROC_CALL을 쿼리하여 시작 시간, 종료 시간, 호출이 취소되었는지 여부 등 저장 프로시저 호출에 대한 정보를 가져올 수 있습니다. 각 저장 프로시저 호출은 쿼리 ID를 수신합니다.

SVL_STORED_PROC_CALL은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_PROCEDURE_CALL에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	문을 실행하는 데 사용된 권한을 소유한 사용자의 ID입니다. 이 호출이 SECURITY DEFINER 저장 프로시저 내에서 중첩된 경우, 이는 해당 저장 프로시저 소유자의 userid입니다.
session_userid	정수	세션 생성자이자 최상위 저장 프로시저 호출의 호출자인 사용자의 ID입니다.
쿼리	정수	프로시저 호출의 쿼리 ID입니다.
레이블	character(320)	쿼리 실행에 사용되는 파일의 이름 또는 SET QUERY_GROUP 명령을 사용하여 정의되는 레이블. 쿼리가 파일 기반이 아니거나 QUERY_GROUP 파라미터가 설정되지 않은 경우, 이 필드의 값은 기본값입니다.
xid	bigint	트랜잭션 ID.
pid	정수	프로세스 ID. 일반적으로 한 세션의 모든 쿼리는 동일 프로세스에서 실행됩니다. 따라서 동일 세션에서 일련의 쿼리를 실행하는 경우에는 이 값은 대부분 같은 값을 유지합니다. Amazon Redshift는 특정한 내부 이벤트 이후에 활성 세션을 다시 시작하고 새 PID 값을 할당할 수도 있습니다. 자세한 내용은 STL_RESTARTED_SESSIONS 단원을 참조하십시오.
데이터베이스	character(32)	쿼리가 실행되었을 때 사용자가 연결된 데이터베이스의 이름.
querytxt	character(4000)	프로시저 호출 쿼리의 실제 텍스트입니다.
starttime	타임스탬프	쿼리 실행이 시작된 UTC 시간으로 소수 초의 정밀도는 6자리입니다. 예: **2009-06-12 11:29:19.131358.**
endtime	타임스탬프	쿼리 실행이 종료된 UTC 시간으로 소수 초의 정밀도는 6자리입니다. 예: **2009-06-12 11:29:19.131358.**

열 명칭	데이터 유형	설명
aborted	정수	저장 프로시저가 시스템에 의해 중지되거나 사용자에 의해 취소되는 경우, 이 열에 1이 포함됩니다. 호출이 실행되어 완료되면 이 열에 0이 포함됩니다.
from_sp_call	정수	프로시저 호출이 다른 프로시저 호출에 의해 호출된 경우, 이 열에 외부 호출의 쿼리 ID가 포함됩니다. 그렇지 않은 경우 이 필드는 NULL입니다.

샘플 쿼리

다음 쿼리는 전날의 저장 프로시저 호출의 경과 시간(내림차순)과 완료 상태를 반환합니다.

```
select query, datediff(seconds, starttime, endtime) as elapsed_time, aborted,
 trim(querytxt) as call from svl_stored_proc_call where starttime >= getdate() -
 interval '1 day' order by 2 desc;

  query | elapsed_time | aborted |                                                call
--------+--------------+---------
+-------------------------------------------------------------------------------------
   4166 |            7 |       0 | call search_batch_status(35,'succeeded');
   2433 |            3 |       0 | call test_batch (123456)
   1810 |            1 |       0 | call prod_benchmark (123456)
   1836 |            1 |       0 | call prod_testing (123456)
   1808 |            1 |       0 | call prod_portfolio ('N', 123456)
   1816 |            1 |       1 | call prod_portfolio ('Y', 123456)
```

SVL_STORED_PROC_MESSAGES

시스템 뷰 SVL_STORED_PROC_MESSAGES를 쿼리하여 저장 프로시저 메시지에 대한 정보를 가져올 수 있습니다. 저장 프로시저 호출이 취소된 경우에도 발생한 메시지가 기록됩니다. 각 저장 프로시저 호출은 쿼리 ID를 수신합니다. 기록된 메시지의 최소 수준을 설정하는 방법에 대한 자세한 내용은 stored_proc_log_min_messages를 참조하십시오.

SVL_STORED_PROC_MESSAGES는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_PROCEDURE_MESSAGES에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
userid	정수	문을 실행하는 데 사용된 권한을 소유한 사용자의 ID입니다. 이 호출이 SECURITY DEFINER 저장 프로시저 내에서 중첩된 경우, 이는 해당 저장 프로시저 소유자의 userid입니다.
session_userid	정수	세션 생성자이자 최상위 저장 프로시저 호출의 호출자인 사용자의 ID입니다.
pid	정수	프로세스 ID.
xid	bigint	프로시저 호출 쿼리의 트랜잭션 ID입니다.
쿼리	정수	프로시저 호출의 쿼리 ID입니다.
recordtime	타임스탬프	메시지가 발생한 시간(UTC)입니다.
loglevel	정수	발생한 메시지의 로그 수준의 숫자 값입니다. 가능한 값: 20 – LOG 30의 경우 – INFO 40의 경우 – NOTICE 50의 경우 – WARNING 60의 경우 – EXCEPTION의 경우
loglevel_text	character(10)	loglevel의 숫자 값에 해당하는 로그 수준입니다. 가능한 값: LOG, INFO, NOTICE, WARNING, EXCEPTION.
message	character(1024)	발생한 메시지의 텍스트입니다.
linenum	정수	발생한 문의 행 번호입니다.
querytext	문자(500)	프로시저 호출 쿼리의 실제 텍스트입니다.
레이블	character(320)	쿼리 실행에 사용되는 파일의 이름 또는 SET QUERY_GROUP 명령을 사용하여 정의되는 레이블. 쿼리가 파일 기반이 아니거나 QUERY_GROUP 파라미터가 설정되지 않은 경우, 이 필드의 값은 기본값입니다.

열 명칭	데이터 유형	설명
aborted	정수	저장 프로시저가 시스템에 의해 중지되거나 사용자에 의해 취소되는 경우, 이 열에 1이 포함됩니다. 호출이 실행되어 완료되면 이 열에 0이 포함됩니다.
message_xid	bigint	발생한 메시지의 트랜잭션 ID입니다.

샘플 쿼리

다음 SQL 문은 SVL_STORED_PROC_MESSAGES를 사용하여 발생한 메시지를 검토하는 방법을 보여 줍니다.

```
-- Create and run a stored procedure
CREATE OR REPLACE PROCEDURE test_proc1(f1 int) AS
$$
BEGIN
    RAISE INFO 'Log Level: Input f1 is %',f1;
    RAISE NOTICE 'Notice Level: Input f1 is %',f1;
    EXECUTE 'select invalid';
    RAISE NOTICE 'Should not print this';

EXCEPTION WHEN OTHERS THEN
     raise exception 'EXCEPTION level: Exception Handling';
END;
$$ LANGUAGE plpgsql;

-- Call this stored procedure
CALL test_proc1(2);

-- Show raised messages with level higher than INFO
SELECT query, recordtime, loglevel, loglevel_text, trim(message) as message, aborted
 FROM svl_stored_proc_messages
  WHERE loglevel > 30 AND query = 193 ORDER BY recordtime;

 query |         recordtime         | loglevel | loglevel_text |               message
              | aborted
-------+----------------------------+----------+---------------
+-----------------------------------+---------
```

```
   193 | 2020-03-17 23:57:18.277196 |       40 | NOTICE        | Notice Level: Input f1
 is 2         |       1
   193 | 2020-03-17 23:57:18.277987 |       60 | EXCEPTION     | EXCEPTION level:
 Exception Handling |       1
(2 rows)

-- Show raised messages at EXCEPTION level
SELECT query, recordtime, loglevel, loglevel_text, trim(message) as message, aborted
 FROM svl_stored_proc_messages
  WHERE loglevel_text = 'EXCEPTION' AND query = 193 ORDER BY recordtime;

 query |         recordtime         | loglevel | loglevel_text |               message
             | aborted
-------+----------------------------+----------+---------------
+-----------------------------------+---------
   193 | 2020-03-17 23:57:18.277987 |       60 | EXCEPTION     | EXCEPTION level:
 Exception Handling |       1
```

다음 SQL 문은 SVL_STORED_PROC_MESSAGES를 사용하여 저장 프로시저를 생성할 때 SET 옵션을 사용하여 발생한 메시지를 검토하는 방법을 보여줍니다. test_proc()의 최소 로그 수준은 NOTICE이므로 NOTICE, WARNING, EXCEPTION 수준 메시지만 SVL_STORED_PROC_MESSAGES에 기록됩니다.

```
-- Create a stored procedure with minimum log level of NOTICE
CREATE OR REPLACE PROCEDURE test_proc() AS
$$
BEGIN
    RAISE LOG 'Raise LOG messages';
    RAISE INFO 'Raise INFO messages';
    RAISE NOTICE 'Raise NOTICE messages';
    RAISE WARNING 'Raise WARNING messages';
    RAISE EXCEPTION 'Raise EXCEPTION messages';
    RAISE WARNING 'Raise WARNING messages again'; -- not reachable
END;
$$ LANGUAGE plpgsql SET stored_proc_log_min_messages = NOTICE;

-- Call this stored procedure
CALL test_proc();

-- Show the raised messages
SELECT query, recordtime, loglevel_text, trim(message) as message, aborted FROM
 svl_stored_proc_messages
  WHERE query = 149 ORDER BY recordtime;
```

```
 query |         recordtime         | loglevel_text |          message         |
 aborted
-------+----------------------------+---------------+--------------------------
+---------
   149 | 2020-03-16 21:51:54.847627 | NOTICE        | Raise NOTICE messages    |
 1
   149 | 2020-03-16 21:51:54.84766  | WARNING       | Raise WARNING messages   |
 1
   149 | 2020-03-16 21:51:54.847668 | EXCEPTION     | Raise EXCEPTION messages |
 1
(3 rows)
```

SVL_TERMINATE

사용자가 프로세스를 취소하거나 종료하는 시간을 기록합니다.

SELECT PG_TERMINATE_BACKEND(pid), SELECT PG_CANCEL_BACKEND(pid) 및 CANCEL pid는 SVL_TERMINATE에 로그 항목을 생성합니다.

SVL_TERMINATE는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_QUERY_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
pid	정수	취소되거나 종료된 프로세스의 프로세스 ID입니다.
eventtime	타임스탬프	프로세스가 취소되거나 종료된 시간입니다.
userid	정수	명령을 실행하는 사용자의 사용자 ID입니다.
type	문자열	종료 유형입니다. CANCEL 또는 TERMINATE일 수 있습니다.

다음 명령은 최근 취소된 쿼리를 보여 줍니다.

```
select * from svl_terminate order by eventtime desc limit 1;
 pid  |         eventtime          | userid |  type
------+----------------------------+--------+--------
 8324 | 2020-03-24 09:42:07.298937 |      1 | CANCEL
(1 row)
```

SVL_UDF_LOG

사용자 정의 함수(UDF)를 실행하는 중에 생성되는 시스템 정의 오류 및 경고 메시지를 기록합니다.

SVL_UDF_LOG는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_UDF_LOG에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
쿼리	bigint	쿼리 ID. 이 ID를 사용하여 다양한 다른 시스템 테이블 및 보기를 조인할 수 있습니다.
message	message	함수에 의해 생성되는 메시지.
생성 완료	타임스탬프	로그가 생성된 시간.
traceback	char(4096)	사용 가능한 경우, 이 값은 UDF에 대한 스택 트레이스백을 제공합니다. 자세한 내용을 보려면 Python Standard Library의 traceback 섹션을 참조하세요.
funcname	character(256)	실행 중인 UDF의 이름.

열 명칭	데이터 유형	설명
노드	정수	메시지가 생성된 노드.
slice	정수	메시지가 생성된 조각.
seq	정수	조각에서의 메시지 시퀀스.

샘플 쿼리

다음 예는 UDF가 시스템 정의 오류를 처리하는 방법을 보여 줍니다. 첫 번째 블록은 인수의 역을 반환하는 UDF 함수의 정의를 보여 줍니다. 두 번째 블록에서처럼 함수를 실행하고 0개의 인수를 제공하면 함수가 오류를 반환합니다. 세 번째 문은 SVL_UDF_LOG에 기록되는 오류 메시지를 읽습니다.

```
-- Create a function to find the inverse of a number

CREATE OR REPLACE FUNCTION  f_udf_inv(a int)
  RETURNS float IMMUTABLE
AS $$
   return 1/a
$$ LANGUAGE plpythonu;

-- Run the function with a 0 argument to create an error
Select f_udf_inv(0) from sales;

-- Query SVL_UDF_LOG to view the message

Select query, created, message::varchar
from svl_udf_log;

 query |          created           | message
-------+----------------------------
+-----------------------------------------------------------
  2211 | 2015-08-22 00:11:12.04819  | ZeroDivisionError: long division or modulo by
 zero\nNone
```

다음 예는 UDF에 로깅 및 경고 메시지를 추가하므로 0으로 나누기 연산은 오류 메시지와 함께 중단되는 대신 경고 메시지를 생성합니다.

```
-- Create a function to find the inverse of a number and log a warning
```

```
CREATE OR REPLACE FUNCTION f_udf_inv_log(a int)
  RETURNS float IMMUTABLE
 AS $$
  import logging
  logger = logging.getLogger() #get root logger
  if a==0:
    logger.warning('You attempted to divide by zero.\nReturning zero instead of error.
\n')
    return 0
  else:
     return 1/a
$$ LANGUAGE plpythonu;
```

다음 예제는 함수를 실행한 다음 SVL_UDF_LOG를 쿼리하여 메시지를 조회합니다.

```
-- Run the function with a 0 argument to trigger the warning
Select f_udf_inv_log(0) from sales;

-- Query SVL_UDF_LOG to view the message

Select query, created, message::varchar
from svl_udf_log;

query |          created           | message
------+----------------------------+-----------------------------------
    0 | 2015-08-22 00:11:12.04819  | You attempted to divide by zero.
                                     Returning zero instead of error.
```

SVL_USER_INFO

SVL_USER_INFO 뷰를 사용하여 Amazon Redshift 데이터베이스 사용자에 대한 데이터를 검색할 수 있습니다.

SVL_USER_INFO는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
usename	텍스트	역할에 대한 사용자 이름입니다.
usesysid	정수	사용자의 사용자 ID입니다.
usecreatedb	boolean	사용자가 데이터베이스를 생성할 수 있는 권한이 있는지 여부를 나타내는 값
usesuper	boolean	사용자가 수퍼유저인지 여부를 나타내는 값
usecatupd	boolean	사용자가 시스템 카탈로그를 업데이트할 수 있는지 여부를 나타내는 값
useconnlimit	텍스트	사용자가 열 수 있는 연결 수
syslogaccess	텍스트	사용자가 시스템 로그에 액세스할 수 있는 권한이 있는지 여부를 나타내는 값 RESTRICTED 및 UNRESTRICTED 의 두 가지 값이 가능합니다. RESTRICTED 이면 수퍼유저가 아닌 사용자가 본인의 레코드를 볼 수 있고, UNRESTRICTED 이면 수퍼유저가 아닌 사용자가 SELECT 권한이 있는 시스템 뷰 및 테이블에 있는 모든 레코드를 볼 수 있습니다.
last_ddl_ts	타임스탬프	사용자가 데이터 정의 언어(DDL) create 문을 마지막으로 실행한 타임스탬프
sessiontimeout	정수	세션이 시간 초과되기 전에 비활성 또는 유휴 상태로 유지되는 최대 시간(초)입니다. 0은 시간 제한이 설정되지 않았음을 나타냅니다. 클러스터의 유휴 또는 비활성 시간 제한 설정에 대한 자세한 내용은 Amazon Redshift 관리 가이드의 Amazon Redshift의 할당량 및 제한 섹션을 참조하세요.
external_id	텍스트	서드 파티 자격 증명 공급자 사용자의 고유 식별자입니다.

샘플 쿼리

다음 명령은 SVL_USER_INFO에서 사용자 정보를 검색합니다.

```
SELECT * FROM SVL_USER_INFO;
```

SVL_VACUUM_PERCENTAGE

SVL_VACUUM_PERCENTAGE 뷰는 vacuum을 수행한 후 테이블에 할당되는 데이터 블록의 백분율을 보고합니다. 이 백분율 숫자는 디스크 공간이 얼마나 회수되었는지 보여 줍니다. vacuum 유틸리티에 대한 자세한 내용은 VACUUM 명령을 참조하십시오.

SVL_VACUUM_PERCENTAGE는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

이 테이블의 데이터 중 일부 또는 전부는 SYS 모니터링 뷰인 SYS_VACUUM_HISTORY에서도 찾아볼 수 있습니다. SYS 모니터링 뷰의 데이터는 사용 및 이해가 더 쉽도록 형식이 지정되어 있습니다. 쿼리에 SYS 모니터링 뷰를 사용하는 것이 좋습니다.

테이블 열

열 명칭	데이터 유형	설명
xid	bigint	vacuum 문의 트랜잭션 ID.
table_id	정수	vacuum된 테이블의 테이블 ID.
percentage	bigint	vacuum 후 데이터 블록의 백분율(vacuum이 실행되기 전 테이블의 블록 수 대비).

샘플 쿼리

다음 쿼리는 테이블 100238에서의 특정 작업의 백분율을 표시합니다.

```
select * from svl_vacuum_percentage
where table_id=100238 and xid=2200;

xid  | table_id | percentage
-----+----------+------------
1337 |   100238 |         60
(1 row)
```

이 vacuum 작업 후 테이블에는 원본 블록의 60퍼센트가 포함되었습니다.

시스템 카탈로그 테이블

주제

시스템 카탈로그는 테이블과 열에 대한 정보 등 스키마 메타데이터를 저장합니다. 시스템 카탈로그 테이블에는 PG 접두사가 있습니다.

Amazon Redshift 사용자는 표준 PostgreSQL 카탈로그 테이블에 액세스할 수 있습니다. PostgreSQL 시스템 카탈로그에 대한 자세한 내용은 PostgreSQL 시스템 테이블을 참조하십시오.

PG_ATTRIBUTE_INFO

PG_ATTRIBUTE_INFO는 PostgreSQL 카탈로그 테이블 PG_ATTRIBUTE 및 내부 카탈로그 테이블 PG_ATTRIBUTE_ACL에 빌드된 Amazon Redshift 시스템 뷰입니다. PG_ATTRIBUTE_INFO에는 열 액세스 제어 목록(있는 경우)을 포함하여 테이블 또는 뷰의 열에 대한 세부 정보가 포함됩니다.

테이블 열

PG_ATTRIBUTE_INFO는 PG_ATTRIBUTE의 열 외에도 다음 열을 보여줍니다.

열 명칭	데이터 유형	설명
attacl	aclitem[]	이 열에 특별히 부여된 열 레벨 액세스 권한(있는 경우)입니다.

PG_CLASS_INFO

PG_CLASS_INFO는 PostgreSQL 카탈로그 테이블 PG_CLASS 및 PG_CLASS_EXTENDED를 기반으로 작성된 Amazon Redshift 시스템 뷰입니다. PG_CLASS_INFO에는 테이블 생성 시간 및 현재 분산 스타일에 대한 세부 정보가 포함되어 있습니다. 자세한 내용은 쿼리 최적화를 위한 데이터 배포 단원을 참조하십시오.

PG_CLASS_INFO는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

PG_CLASS_INFO는 PG_CLASS의 열 외에도 다음 열을 보여줍니다. PG_CLASS의 `oid` 열은 PG_CLASS_INFO 테이블에서 `relid`라고 합니다.

열 명칭	데이터 유형	설명
relcreationtime	타임스탬프	테이블이 생성된 시간입니다(UTC).
releffectivediststyle	정수	테이블의 배포 스타일 또는 Amazon Redshift에서 할당한 현재 배포 스타일(테이블이 자동 배포를 사용하는 경우)

PG_CLASS_INFO의 RELEFFECTIVEDISTSTYLE 열은 테이블의 현재 분산 스타일을 나타냅니다. 테이블에서 자동 분산을 사용하는 경우 RELEFFECTIVEDISTSTYLE은 10, 11 또는 12입니다. 즉, 효과적인 분산 스타일이 AUTO (ALL), AUTO (EVEN) 또는 AUTO (KEY)임을 나타냅니다. 테이블에서 자동 분산을 사용하는 경우 분산 스타일은 처음에 AUTO (ALL)로 표시된 다음, 열이 분산 키로써 유용하다 판명되면 AUTO (EVEN) 또는 AUTO (KEY)로 변경될 수 있습니다.

다음 표는 RELEFFECTIVEDISTSTYLE 열에서 각 값에 따른 분산 스타일을 나타낸 것입니다.

RELEFFECTIVEDISTSTYLE	현재 분산 스타일
0	0
1	키

RELEFFECTIVEDISTSTYLE	현재 분산 스타일
8	ALL
10	AUTO(ALL)
11	AUTO(EVEN)
12	AUTO(KEY)

예제

다음 쿼리는 카탈로그의 현재 테이블의 분산 스타일을 반환합니다.

```
select relid as tableid,trim(nspname) as schemaname,trim(relname) as
 tablename,reldiststyle,releffectivediststyle,
CASE WHEN "reldiststyle" = 0 THEN 'EVEN'::text
     WHEN "reldiststyle" = 1 THEN 'KEY'::text
     WHEN "reldiststyle" = 8 THEN 'ALL'::text
     WHEN "releffectivediststyle" = 10 THEN 'AUTO(ALL)'::text
     WHEN "releffectivediststyle" = 11 THEN 'AUTO(EVEN)'::text
     WHEN "releffectivediststyle" = 12 THEN 'AUTO(KEY)'::text ELSE '<<UNKNOWN>>'::text
 END as diststyle,relcreationtime
from pg_class_info a left join pg_namespace b on a.relnamespace=b.oid;
```

```
 tableid | schemaname | tablename | reldiststyle | releffectivediststyle | diststyle  |
      relcreationtime
---------+------------+-----------+--------------+-----------------------+------------
+----------------------------
 3638033 | public     | customer  |            0 |                     0 | EVEN       |
 2019-06-13 15:02:50.666718
 3638037 | public     | sales     |            1 |                     1 | KEY        |
 2019-06-13 15:03:29.595007
 3638035 | public     | lineitem  |            8 |                     8 | ALL        |
 2019-06-13 15:03:01.378538
 3638039 | public     | product   |            9 |                    10 | AUTO(ALL)  |
 2019-06-13 15:03:42.691611
 3638041 | public     | shipping  |            9 |                    11 | AUTO(EVEN) |
 2019-06-13 15:03:53.69192
```

```
 3638043 | public     | support   |           9 |                    12 | AUTO(KEY)  |
 2019-06-13 15:03:59.120695
(6 rows)
```

PG_DATABASE_INFO

PG_DATABASE_INFO는 PostgreSQL 카탈로그 테이블 PG_DATABASE를 확장하는 Amazon Redshift 시스템 뷰입니다.

PG_DATABASE_INFO는 모든 사용자에게 표시됩니다.

테이블 열

PG_DATABASE_INFO는 PG_DATABASE의 열 외에도 다음 열을 포함합니다. PG_DATABASE의 `oid` 열은 PG_DATABASE_INFO 테이블에서 `datid`라고 합니다. 자세한 내용은 PostgreSQL 설명서를 참조하세요.

열 명칭	데이터 유형	설명
datid	oid	시스템 테이블에서 내부적으로 사용하는 객체 식별자(OID)입니다.
datconnlimit	텍스트	이 데이터베이스에 연결할 수 있는 최대 동시 연결의 수입니다. 값 -1은 제한이 없음을 의미합니다.

PG_DEFAULT_ACL

기본 액세스 권한에 대한 정보를 저장합니다. 기본 액세스 권한에 대한 자세한 내용은 ALTER DEFAULT PRIVILEGES 섹션을 참조하세요.

PG_DEFAULT_ACL은 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
defacluser	정수	나열된 권한이 적용되는 사용자의 ID입니다.

열 명칭	데이터 유형	설명
defaclnamespace	oid	기본 권한이 적용되는 스키마의 객체 ID입니다. 스키마를 지정하지 않은 경우 기본값은 0입니다.
defaclobjtype	character	기본 권한이 적용되는 객체 유형입니다. 유효한 값은 다음과 같습니다. • r - 관계(테이블 또는 뷰) • f – 함수 • p - 저장 프로시저

열 명칭	데이터 유형	설명
defaclacl	aclitem[]	지정한 사용자 또는 사용자 그룹과 객체 유형의 기본 권한을 정의하는 문자열입니다. 기본 권한을 사용자에게 부여할 때 문자열은 다음과 같습니다. *{ username=privilegestring/grantor }* 사용자 이름 권한이 부여되는 사용자의 이름입니다. username을 지정하지 않으면 권한은 PUBLIC에게 부여됩니다. 기본 권한을 사용자 그룹에게 부여할 때 문자열은 다음과 같습니다. *{ "group groupname=privilegestring/grantor" }* privilegestring 부여할 권한을 지정하는 문자열입니다. 유효한 값은 다음과 같습니다. • a – INSERT(추가) • r – SELECT(읽기) • w – UPDATE(쓰기) • d – DELETE • R – RULE • x – 외래 키 제약 조건을 만드는 권한을 부여합니다(REFERENCES). • t – 트리거 • X – EXECUTE • U – USAGE • C – CREATE

열 명칭	데이터 유형	설명
		• T – CREATE TEMP • D – DROP • P – TRUNCATE • A – ALTER • * – 먼저 권한을 받은 사용자가 이어서 다른 사용자에게 동일한 권한을 부여할 수 있다는 것을 나타냅니다(WITH GRANT OPTION). 모든 권한 코드 문자를 포함하는 문자열은 비트마스크 위치별로 'arwdRxtXUCTDPA'와 같습니다. grantor 권한을 부여한 사용자의 이름입니다. 다음은 사용자 admin이 사용자 dbuser에게 WITH GRANT OPTION을 포함하여 모든 권한을 부여한 것을 나타내는 예입니다. `dbuser=r*a*w*d*x*X*/admin`

예제

다음은 데이터베이스에 정의되어 있는 기본 권한을 모두 반환하는 쿼리입니다.

```
select pg_get_userbyid(d.defacluser) as user,
n.nspname as schema,
case d.defaclobjtype when 'r' then 'tables' when 'f' then 'functions' end
as object_type,
array_to_string(d.defaclacl, ' + ')  as default_privileges
from pg_catalog.pg_default_acl d
left join pg_catalog.pg_namespace n on n.oid = d.defaclnamespace;

 user  | schema | object_type |              default_privileges
-------+--------+-------------+-------------------------------------------------------
```

```
 admin | tickit | tables      | user1=r/admin + "group group1=a/admin" + user2=w/admin
```

위 예의 결과를 보면 사용자 admin이 tickit 스키마에서 새롭게 생성한 모든 테이블에 대해 admin이 SELECT 권한을 user1에게, INSERT 권한을 group1에게, 그리고 UPDATE 권한을 user2에게 부여하는 것을 알 수 있습니다.

PG_EXTERNAL_SCHEMA

외부 스키마에 대한 정보를 저장합니다.

PG_EXTERNAL_SCHEMA는 모든 사용자에게 공개됩니다. 수퍼 사용자는 모든 행을 볼 수 있지만 일반 사용자는 액세스 권한을 가지고 있는 메타데이터에 한해 볼 수 있습니다. 자세한 내용은 CREATE EXTERNAL SCHEMA 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
esoid	oid	외부 스키마 ID입니다.
eskind	정수	일종의 외부 스키마입니다.
esdbname	텍스트	외부 데이터베이스 이름입니다.
esoptions	텍스트	외부 스키마 옵션입니다.

예제

다음은 외부 스키마에 대한 세부 정보를 나타내는 예입니다.

```
select esoid, nspname as schemaname, nspowner, esdbname as external_db, esoptions
from pg_namespace a,pg_external_schema b where a.oid=b.esoid;

esoid  | schemaname      | nspowner | external_db | esoptions

-------+-----------------+----------+-------------
+-------------------------------------------------------------
100134 | spectrum_schema |      100 | spectrum_db |
 {"IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
100135 | spectrum        |      100 | spectrumdb  |
 {"IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
```

```
100149 | external        |      100 | external_db |
 {"IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
```

PG_LIBRARY

사용자 정의 라이브러리에 대한 정보를 저장합니다.

PG_LIBRARY는 모든 사용자에게 표시됩니다. 수퍼유저는 모든 행을 볼 수 있지만 일반 사용자는 자체 데이터만 볼 수 있습니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
name	name	라이브러리 이름입니다.
language_oid	oid	시스템에서 사용하도록 예약됩니다.
file_store_id	정수	시스템에서 사용하도록 예약됩니다.
owner	정수	라이브러리 소유자의 사용자 ID입니다.

예제

다음은 사용자 정의 라이브러리에 대한 정보를 반환하는 예입니다.

```
select * from pg_library;

name       | language_oid | file_store_id | owner
-----------+--------------+---------------+------
f_urlparse |       108254 |          2000 |   100
```

PG_PROC_INFO

PG_PROC_INFO는 PostgreSQL 카탈로그 테이블 PG_PROC 및 내부 카탈로그 테이블 PG_PROC_EXTENDED에 빌드된 Amazon Redshift 시스템 뷰입니다. PG_PROC_INFO에는 출력 인

수와 관련된 정보를 포함하여(있는 경우) 저장 프로시저 및 함수에 대한 세부 정보가 포함되어 있습니다.

테이블 열

PG_CLASS_INFO는 PG_PROC의 열 외에도 다음 열을 보여줍니다. PG_PROC의 `oid` 열은 PG_PROC_INFO 테이블에서 `prooid`라고 합니다.

열 명칭	데이터 유형	설명
prooid	oid	함수 또는 저장 프로시저의 객체 ID입니다.
prokind	"char"	함수 또는 저장 프로시저의 유형을 나타내는 값입니다. 이 값은 정규 함수의 경우 'f', 저장 프로시저의 경우 'p', 집계 함수의 경우 'a'입니다.
proargmodes	"char"[ ]	프로시저 인수 모드의 배열입니다. IN 인수의 경우 'i', OUT 인수의 경우 'o', INOUT 인수의 경우 'b'로 인코딩되어 있습니다. 모든 인수가 IN 인수이면 이 필드는 NULL입니다. 아래 첨자는 proallargtypes 배열의 위치에 해당합니다.
proallargtypes	oid[ ]	프로시저 인수의 데이터 형식의 배열입니다. 이 배열에는 모든 유형의 인수가 포함됩니다(OUT 및 INOUT 인수 포함). 그러나 모든 인수가 IN 인수이면 이 필드는 NULL입니다. 첨자 지정은 1부터 시작합니다. 반면에 proargtypes는 첨자를 0부터 시작합니다.

PG_PROC_INFO의 필드 proargnames에는 OUT 및 INOUT을 포함하여 모든 유형의 인수 이름이 포함되어 있습니다(있는 경우).

PG_STATISTIC_INDICATOR

마지막 ANALYZE 이후 삽입 또는 삭제된 행의 수에 대한 정보를 저장합니다.
PG_STATISTIC_INDICATOR 테이블은 DML 연산 이후 업데이트가 빈번하기 때문에 통계 값이 근사치입니다.

PG_STATISTIC_INDICATOR는 슈퍼 사용자에게만 표시됩니다. 자세한 내용은 시스템 테이블 및 뷰에 있는 데이터의 가시성 단원을 참조하십시오.

테이블 열

열 명칭	데이터 유형	설명
stairelid	oid	테이블 ID입니다.
stairows	float	테이블에 포함된 행의 총 수입니다.
staiins	float	마지막 ANALYZE 이후 삽입된 행의 수입니다.
staidels	float	마지막 ANALYZE 이후 삭제 또는 업데이트된 행의 수입니다.

예제

다음은 마지막 ANALYZE 이후 테이블의 변경 정보를 반환하는 예입니다.

```
select * from pg_statistic_indicator;

stairelid | stairows | staiins | staidels
----------+----------+---------+---------
   108271 |       11 |       0 |        0
   108275 |      365 |       0 |        0
   108278 |     8798 |       0 |        0
   108280 |    91865 |       0 |   100632
   108267 |    89981 |   49990 |     9999
   108269 |      808 |     606 |      374
   108282 |   152220 |   76110 |   248566
```

PG_TABLE_DEF

테이블 열에 대한 정보를 저장합니다.

PG_TABLE_DEF는 사용자에게 공개되는 테이블에 대한 정보만 반환합니다. PG_TABLE_DEF가 예상한 결과를 반환하지 않을 때는 search_path 파라미터가 관련 스키마를 포함하여 올바로 설정되어 있는지 확인하십시오.

그 밖에 데이터 분산 스큐, 키 분산 스큐, 테이블 크기, 통계 등 테이블에 대해 더욱 포괄적인 정보를 확인하려면 SVV_TABLE_INFO를 사용하면 됩니다.

테이블 열

열 명칭	데이터 유형	설명
schemaname	name	스키마 이름
tablename	name	테이블 이름.
column	name	열 이름.
type	텍스트	열의 데이터 형식입니다.
인코딩	character(32)	열의 인코딩입니다.
distkey	boolean	이 열이 테이블의 분산 키일 경우 true입니다.
sortkey	정수	정렬 키의 열 순서입니다. 테이블이 복합 정렬 키를 사용하는 경우에는 정렬 키에 포함된 모든 열이 양의 값으로 정렬 키에서 자신의 위치를 나타냅니다. 테이블이 인터리브 정렬 키를 사용하는 경우에는 정렬 키에 포함된 각 열이 각각 양의 값이나 음의 값을 갖습니다. 이때는 절대 값이 정렬 키에서 열의 위치를 나타냅니다. 값이 0이면 정렬 키에 포함되지 않는 열입니다.
notnull	boolean	열에 NOT NULL 제약 조건이 있으면 true입니다.

예제

다음은 LINEORDER_COMPOUND 테이블에서 복합 정렬 키 열을 나타내는 예입니다.

```
select "column", type, encoding, distkey, sortkey, "notnull"
from pg_table_def
where tablename = 'lineorder_compound'
and sortkey <> 0;

column       | type    | encoding | distkey | sortkey | notnull
-------------+---------+----------+---------+---------+--------
lo_orderkey  | integer | delta32k | false   |       1 | true
lo_custkey   | integer | none     | false   |       2 | true
```

```
lo_partkey   | integer | none     | true    |       3 | true
lo_suppkey   | integer | delta32k | false   |       4 | true
lo_orderdate | integer | delta    | false   |       5 | true
(5 rows)
```

다음은 LINEORDER_INTERLEAVED 테이블에서 인터리브 정렬 키 열을 나타내는 예입니다.

```
select "column", type, encoding, distkey, sortkey, "notnull"
from pg_table_def
where tablename = 'lineorder_interleaved'
and sortkey <> 0;

column       | type    | encoding | distkey | sortkey | notnull
-------------+---------+----------+---------+---------+--------
lo_orderkey  | integer | delta32k | false   |      -1 | true
lo_custkey   | integer | none     | false   |       2 | true
lo_partkey   | integer | none     | true    |      -3 | true
lo_suppkey   | integer | delta32k | false   |       4 | true
lo_orderdate | integer | delta    | false   |      -5 | true
(5 rows)
```

PG_TABLE_DEF는 스키마에서 검색 경로에 포함된 테이블 정보만 반환합니다. 자세한 내용은 search_path 단원을 참조하십시오.

예를 들어 새로운 스키마와 테이블을 생성한 후에 PG_TABLE_DEF에 대한 쿼리를 실행한다고 가정하겠습니다.

```
create schema demo;
create table demo.demotable (one int);
select * from pg_table_def where tablename = 'demotable';

schemaname|tablename|column| type | encoding | distkey | sortkey | notnull
----------+---------+------+------+----------+---------+---------+--------
```

다음 쿼리는 새로운 테이블에 대해서 아무런 행도 반환하지 않습니다. 이때는 search_path 설정을 검사하십시오.

```
show search_path;

  search_path
---------------
 $user, public
```

```
(1 row)
```

demo 스키마를 검색 경로에 추가한 후 쿼리를 다시 실행합니다.

```
set search_path to '$user', 'public', 'demo';

select * from pg_table_def where tablename = 'demotable';

schemaname| tablename |column|  type   | encoding |distkey|sortkey| notnull
----------+-----------+------+---------+----------+-------+-------+--------
demo      | demotable | one  | integer | none     | f     |     0 | f
(1 row)
```

PG_USER_INFO

PG_USER_INFO는 사용자 ID 및 암호 만료 시간과 같은 사용자 정보를 표시하는 Amazon Redshift 시스템 보기입니다.

수퍼유저만 PG_USER_INFO를 볼 수 있습니다.

테이블 열

PG_USER_INFO에는 다음 열이 포함되어 있습니다. 자세한 내용은 PostgreSQL 설명서를 참조하세요.

열 명칭	데이터 유형	설명
usename	name	사용자 이름입니다.
usesysid	정수	사용자 ID입니다.
usecreatedb	boolean	사용자가 데이터베이스를 생성할 수 있는 경우 True입니다.
usesuper	boolean	사용자가 수퍼유저인 경우 True입니다.
usecatupd	boolean	사용자가 시스템 카탈로그를 업데이트할 수 있는 경우 True입니다.
passwd	텍스트	암호입니다.

열 명칭	데이터 유형	설명
valuntil	abstime	암호의 만료 날짜 및 시간입니다.
useconfig	text[]	런타임 변수의 세션 기본값입니다.
useconnlimit	텍스트	사용자가 열 수 있는 연결 수

카탈로그 테이블 쿼리

주제

- 카탈로그 쿼리 예제

일반적으로 카탈로그 테이블과 뷰(이름이 **PG_**로 시작하는 릴레이션)는 Amazon Redshift 테이블과 뷰로 조인할 수 있습니다.

카탈로그 테이블은 Amazon Redshift가 지원하지 않는 데이터 형식을 다수 사용합니다. 다음은 쿼리가 카탈로그 테이블을 Amazon Redshift 테이블로 조인할 때 지원되는 데이터 형식입니다.

- bool
- "char"
- "char"
- "char"
- "char"
- "char"
- name
- oid
- 텍스트
- varchar

조인 쿼리를 작성하면서 지원되지 않는 데이터 형식이 포함된 열을 명시적으로, 또는 묵시적으로 참조하는 경우에는 쿼리가 오류를 반환합니다. PG_SETTINGS 및 PG_LOCKS 테이블에서 사용하는 함수를 제외하고 일부 카탈로그 테이블에서 사용되는 SQL 함수 역시 지원되지 않습니다.

예를 들어 PG_STATS 테이블은 Amazon Redshift 테이블과 조인할 때 지원되지 않는 함수로 인해 쿼리를 실행할 수 없습니다.

다음 카탈로그 테이블과 뷰는 Amazon Redshift 테이블의 정보로 조인할 수 있는 유용한 정보를 제공합니다. 단, 일부 테이블은 데이터 형식 및 함수 제한으로 인해 부분적 액세스만 허용됩니다. 부분적 액세스만 허용되는 테이블에 대해 쿼리를 실행할 때는 주의하여 열을 선택하거나 참조하십시오.

다음 테이블은 완전 액세스가 가능하여 지원되지 않는 형식이나 함수가 전혀 없습니다.

- pg_attribute
- pg_cast
- pg_depend
- pg_description
- pg_locks
- pg_opclass

다음 테이블은 부분적 액세스가 가능하여 지원되지 않는 형식이나 함수, 또는 잘린 텍스트 열이 포함됩니다. 텍스트 열의 값은 varchar(256) 값까지 잘립니다.

- pg_class
- pg_constraint
- pg_database
- pg_group
- pg_language
- pg_namespace
- pg_operator
- pg_proc
- pg_settings
- pg_statistic
- pg_tables
- pg_type
- pg_user
- pg_views

여기서 언급하지 않는 카탈로그 테이블은 액세스가 안 되거나, 혹은 Amazon Redshift 관리자가 사용하지 못할 가능성이 높습니다. 하지만 쿼리가 Amazon Redshift 테이블 조인과 관련이 없다면 모든 카탈로그 테이블 또는 뷰에 대해 공개적으로 쿼리를 실행할 수 있습니다.

Postgres 카탈로그 테이블에서는 OID 열을 조인 열로 사용할 수 있습니다. 예를 들어 조인 조건 `pg_database.oid = stv_tbl_perm.db_id`에 따라 각 PG_DATABASE 행의 내부 데이터베이스 객체 ID는 STV_TBL_PERM 테이블에 공개되는 DB_ID 열과 일치합니다. OID 열은 테이블에서 선택 시 공개되지 않는 내부 기본 키입니다. 카탈로그 뷰에는 OID 열이 없습니다.

일부 Amazon Redshift 함수는 반드시 컴퓨팅 노드에서만 실행해야 합니다. 쿼리에서 사용자가 만든 테이블을 참조하는 경우, SQL은 컴퓨팅 노드에서 실행됩니다.

CATALOG 테이블(PG_TABLE_DEF처럼 PG 접두사가 첨부된 테이블)만 참조하거나 어떤 테이블도 참조하지 않는 쿼리는 리더 노드에서만 실행됩니다.

컴퓨팅 노드 함수를 사용하는 쿼리에서 사용자 정의 테이블이나 Amazon Redshift 시스템 테이블을 참조하지 않는 경우 다음 오류가 반환됩니다.

```
[Amazon](500310) Invalid operation: One or more of the used functions must be applied
 on at least one user created table.
```

카탈로그 쿼리 예제

아래 쿼리들은 카탈로그 테이블에 대한 쿼리를 실행하여 Amazon Redshift 데이터베이스에 대한 유용한 정보를 가져올 수 있는 몇 가지 방법을 보여줍니다.

테이블 ID, 데이터베이스, 스키마 및 테이블 이름 보기

다음은 STV_TBL_PERM 시스템 테이블과 PG_CLASS, PG_NAMESPACE 및 PG_DATABASE 시스템 카탈로그 테이블을 조인하여 테이블 ID, 데이터베이스 이름, 스키마 이름 및 테이블 이름을 반환하는 뷰 정의입니다.

```
create view tables_vw as
select distinct(stv_tbl_perm.id) table_id
,trim(pg_database.datname)   db_name
,trim(pg_namespace.nspname)   schema_name
,trim(pg_class.relname)   table_name
from stv_tbl_perm
join pg_class on pg_class.oid = stv_tbl_perm.id
join pg_namespace on pg_namespace.oid = pg_class.relnamespace
join pg_database on pg_database.oid = stv_tbl_perm.db_id;
```

다음은 테이블 ID 117855의 정보를 반환하는 예입니다.

```
select * from tables_vw where table_id = 117855;
```

```
table_id | db_name   | schema_name | table_name
---------+-----------+-------------+-----------
  117855 |       dev | public      | customer
```

Amazon Redshift 테이블당 열의 수 나열

다음 쿼리는 일부 카탈로그 테이블을 조인하여 각 Amazon Redshift 테이블에 포함된 열 수를 확인합니다. Amazon Redshift 테이블 이름은 PG_TABLES와 STV_TBL_PERM에 모두 저장됩니다. 가능한 경우 PG_TABLES를 사용하여 Amazon Redshift 테이블 이름을 반환합니다.

이번 쿼리에는 Amazon Redshift 테이블이 포함되지 않습니다.

```
select nspname, relname, max(attnum) as num_cols
from pg_attribute a, pg_namespace n, pg_class c
where n.oid = c.relnamespace and  a.attrelid = c.oid
and c.relname not like '%pkey'
and n.nspname not like 'pg%'
and n.nspname not like 'information%'
group by 1, 2
order by 1, 2;

nspname | relname  | num_cols
--------+----------+----------
public  | category |        4
public  | date     |        8
public  | event    |        6
public  | listing  |        8
public  | sales    |       10
public  | users    |       18
public  | venue    |        5
(7 rows)
```

데이터베이스의 스키마 및 테이블 나열

다음은 STV_TBL_PERM을 일부 PG 테이블에 조인하여 TICKIT 데이터베이스의 테이블 목록과 스키마 이름(NSPNAME 열)을 반환하는 쿼리입니다. 그 밖에 각 테이블의 전체 행 수도 반환합니다. 이 쿼리는 시스템에서 다수의 스키마가 똑같은 테이블 이름을 가지고 있을 때 유용합니다.

```
select datname, nspname, relname, sum(rows) as rows
from pg_class, pg_namespace, pg_database, stv_tbl_perm
where pg_namespace.oid = relnamespace
and pg_class.oid = stv_tbl_perm.id
and pg_database.oid = stv_tbl_perm.db_id
and datname ='tickit'
group by datname, nspname, relname
order by datname, nspname, relname;

datname | nspname | relname  |  rows
--------+---------+----------+--------
tickit  | public  | category |     11
tickit  | public  | date     |    365
tickit  | public  | event    |   8798
tickit  | public  | listing  | 192497
tickit  | public  | sales    | 172456
tickit  | public  | users    |  49990
tickit  | public  | venue    |    202
(7 rows)
```

테이블 ID, 데이터 형식, 열 이름 및 테이블 이름 나열

다음은 테이블 ID, 테이블 이름, 열 이름, 각 열의 데이터 형식 등 각 사용자 테이블과 테이블 열에 대한 정보를 나열하는 쿼리입니다.

```
select distinct attrelid, rtrim(name), attname, typname
from pg_attribute a, pg_type t, stv_tbl_perm p
where t.oid=a.atttypid and a.attrelid=p.id
and a.attrelid between 100100 and 110000
and typname not in('oid','xid','tid','cid')
order by a.attrelid asc, typname, attname;

attrelid |  rtrim   |    attname     |  typname
---------+----------+----------------+-----------
  100133 | users    | likebroadway   | bool
  100133 | users    | likeclassical  | bool
  100133 | users    | likeconcerts   | bool
...
  100137 | venue    | venuestate     | bpchar
  100137 | venue    | venueid        | int2
  100137 | venue    | venueseats     | int4
  100137 | venue    | venuecity      | varchar
```

```
...
```

테이블에 있는 각 열의 데이터 블록 수 계산

다음은 STV_BLOCKLIST 테이블을 PG_CLASS에 조인하여 SALES 테이블의 열에 대한 스토리지 정보를 반환하는 쿼리입니다.

```
select col, count(*)
from stv_blocklist s, pg_class p
where s.tbl=p.oid and relname='sales'
group by col
order by col;

col | count
----+-------
  0 |     4
  1 |     4
  2 |     4
  3 |     4
  4 |     4
  5 |     4
  6 |     4
  7 |     4
  8 |     4
  9 |     8
 10 |     4
 12 |     4
 13 |     8
(13 rows)
```

구성 참조

구성을 사용하여 환경을 사용자 지정할 수 있습니다. Amazon Redshift를 사용하면 다양한 파라미터와 설정을 구성하여 데이터 웨어하우징 환경을 사용자 지정하고 최적화할 수 있습니다. 구성 참조에는 사용 가능한 클러스터 속성, 데이터베이스 파라미터 및 워크로드 관리(WLM) 구성 옵션이 요약되어 있습니다. 이 참조를 통해 특정 요구 사항에 따라 성능, 보안 및 리소스 할당을 미세 조정할 수 있습니다. 다음 참조에서는 원하는 데이터 웨어하우징 설정을 충족하도록 이러한 구성을 수정하는 방법에 대한 자세한 가이드가 설명됩니다.

주제

- navigate_super_null_on_error
- parse_super_null_on_error
- pg_federation_repeatable_read
- query_group
- search_path
- spectrum_enable_pseudo_columns
- enable_spectrum_oid
- spectrum_query_maxerror
- statement_timeout
- stored_proc_log_min_messages
- timezone
- use_fips_ssl
- wlm_query_slot_count

서버 구성 수정

서버 구성은 다음과 같은 방법으로 변경할 수 있습니다.

- SET 명령을 사용하여 현재 세션에 한해 지속 시간 설정을 재정의합니다.

  예:

  ```
  set extra_float_digits to 2;
  ```

- 클러스터의 파라미터 그룹 설정을 수정합니다. 파라미터 그룹 설정에는 구성할 수 있는 파라미터가 추가로 포함됩니다. 자세한 내용은 Amazon Redshift 관리 가이드의 Amazon Redshift 파라미터 그룹 섹션을 참조하세요.
- ALTER USER 명령을 사용하여 지정한 사용자가 실행하는 모든 세션에 대해 구성 파라미터를 새로운 값으로 설정합니다.

  ```
  ALTER USER username SET parameter { TO | = } { value | DEFAULT }
  ```

SHOW 명령은 현재 파라미터 설정을 확인하는 데 사용됩니다. SHOW ALL을 사용하면 SET 명령으로 구성할 수 있는 설정이 모두 표시됩니다.

```
SHOW ALL;

name                        | setting
----------------------------+--------------
analyze_threshold_percent | 10
datestyle                   | ISO, MDY
extra_float_digits          | 2
query_group                 | default
search_path                 | $user, public
statement_timeout           | 0
timezone                    | UTC
wlm_query_slot_count        | 1
```

> **Note**
>
> 구성 파라미터는 데이터 웨어하우스에서 연결된 데이터베이스에 적용된다는 점에 유의하세요.

analyze_threshold_percent

값(기본값은 굵은 글꼴로 표시)

10 , 0 ~ 100.0

설명

테이블 분석 시 변경되는 행 비율 임계값을 설정합니다. 처리 시간을 줄이고 전반적인 시스템 성능을 향상시키기 위해 Amazon Redshift는 `analyze_threshold_percent`에 지정된 것보다 낮은 행 변경 비율을 가진 테이블에 대해 ANALYZE를 건너뜁니다. 예를 들어 테이블에 포함된 행이 100,000,000개인데 마지막 ANALYZE 이후 변경된 행이 9,000,000개라면 변경된 행의 개수가 10% 미만이기 때문에 이 테이블은 분석을 건너뜁니다. 소수의 행만 변경되었을 때 테이블을 분석하려면 `analyze_threshold_percent`를 임의의 작은 수로 설정하십시오. 예를 들어, `analyze_threshold_percent`를 0.01로 설정하면 10,000개 이상의 행이 변경된 경우 100,000,000개의 행이 있는 테이블을 건너뛰지 않습니다. 변경된 행이 없더라도 모든 테이블을 분석하려면 `analyze_threshold_percent`를 0으로 설정하십시오.

SET 명령을 사용하여 현재 세션에 대해서만 `analyze_threshold_percent` 파라미터를 수정할 수 있습니다. 파라미터 그룹 내에서는 파라미터를 수정할 수 없습니다.

예제

```
set analyze_threshold_percent to 15;
set analyze_threshold_percent to 0.01;
set analyze_threshold_percent to 0;
```

cast_super_null_on_error

값(기본값은 굵은 글꼴로 표시)

on, off

설명

존재하지 않는 객체 멤버 또는 배열 요소에 액세스하려고 할 때 쿼리가 기본 lax 모드에서 실행되는 경우 Amazon Redshift가 NULL 값을 반환하도록 지정합니다.

datashare_break_glass_session_var

값(기본값은 굵은 글꼴로 표시)

기본값이 없습니다. 값은 다음에 설명된 대로 권장되지 않는 작업이 발생할 때 Amazon Redshift에서 생성한 모든 문자열일 수 있습니다.

설명

일반적으로 AWS Data Exchange datashare에 권장되지 않는 특정 작업을 허용하는 권한을 적용합니다.

일반적으로 DROP DATASHARE 또는 ALTER DATASHARE SET PUBLICACCESSIBLE 문을 사용하여 AWS Data Exchange datashare를 삭제하거나 변경하지 않는 것이 좋습니다. 공개적으로 액세스할 수 있는 설정을 해제하기 위해 AWS Data Exchange datashare 삭제 또는 변경을 허용하려면 `datashare_break_glass_session_var` 변수를 일회성 값으로 설정합니다. 이 일회성 값은 Amazon Redshift에서 생성되며 해당 작업에 대한 초기 시도 후 오류 메시지로 제공됩니다.

변수를 일회성 생성 값으로 설정한 후 DROP DATASHARE 또는 ALTER DATASHARE 문을 다시 실행합니다.

자세한 내용은 ALTER DATASHARE 사용 참고 사항 또는 DROP DATASHARE 사용 참고 사항을 참조하세요.

예제

```
set datashare_break_glass_session_var to '620c871f890c49';
```

datestyle

값(기본값은 굵은 글꼴로 표시)

형식 지정 (ISO , Postgres, SQL 또는 German) 및 년 / 월 / 일 주문 (DMY, MDY , YMD)

- ISO - YYYY-MM-DD HH:MM:SS의 날짜 스타일을 사용합니다.
- Postgres - MM-DD HH:MM:SS YYYY의 날짜 스타일을 사용합니다.
- SQL - MM-DD-YYYY HH:MM:SS의 날짜 스타일을 사용합니다.
- German - DD-MM-YYYY HH:MM:SS의 날짜 스타일을 사용합니다.

설명

날짜 및 시간 값의 표시 형식을 비롯해 의미가 불명확한 날짜 입력 값에 대한 해석 규칙을 설정합니다. 문자열에는 따로 또는 함께 변경할 수 있는 파라미터가 2개 포함됩니다.

예제

```
show datestyle;
DateStyle
-----------
ISO, MDY
(1 row)

set datestyle to 'SQL,DMY';
```

default_geometry_encoding

값(기본값은 굵은 글꼴로 표시)

1, 2

설명

이 세션 중 생성된 공간 지오메트리가 경계 상자로 인코딩되는지를 지정하는 세션 구성입니다. `default_geometry_encoding`이 `1`이면 지오메트리가 경계 상자로 인코딩되지 않습니다. `default_geometry_encoding`이 `2`이면 지오메트리가 경계 상자로 인코딩됩니다. 경계 상자 지원에 대한 자세한 내용은 경계 상자 섹션을 참조하세요.

describe_field_name_in_uppercase

값(기본값은 굵은 글꼴로 표시)

off (false) , on (true)

설명

SELECT 문이 반환하는 열 이름이 대문자 또는 소문자인지 지정합니다. 이 파라미터가 on이면 열 이름이 대문자로 반환됩니다. 이 파라미터가 off이면 열 이름이 소문자로 반환됩니다. Amazon Redshift는 `describe_field_name_in_uppercase` 설정에 관계없이 열 이름을 소문자로 저장합니다.

예제

```
set describe_field_name_in_uppercase to on;

show describe_field_name_in_uppercase;

DESCRIBE_FIELD_NAME_IN_UPPERCASE
--------------------------------
on
```

downcase_delimited_identifier

값(기본값은 굵은 글꼴로 표시)

on, off

설명

이 구성은 사용 중지됩니다. 대신에 enable_case_sensitive_identifier를 사용하십시오.

슈퍼 구문 분석기에서 대문자 또는 대/소문자가 혼합된 JSON 필드를 읽을 수 있도록 합니다. 또한 데이터베이스, 스키마, 테이블 및 열의 대/소문자가 혼합된 이름으로 지원되는 PostgreSQL 데이터베이스에 대한 연합 쿼리 지원을 사용합니다. 대/소문자 구분 식별자를 사용하려면 이 파라미터를 off로 설정합니다.

사용 관련 참고 사항

- 행 수준 보안 또는 동적 데이터 마스킹 기능을 사용하는 경우 클러스터 또는 워크그룹의 파라미터 그룹에서 downcase_delimited_identifier 값을 설정하는 것이 좋습니다. 이렇게 하면 정책을 만들고 연결한 다음 정책이 적용된 관계를 쿼리하는 동안 downcase_delimited_identifier가 일정하게 유지됩니다. 행 수준 보안에 대한 자세한 내용은 행 수준 보안 섹션을 참조하세요. 동적 데이터 마스킹에 대한 자세한 내용은 동적 데이터 마스킹 섹션을 참조하세요.
- downcase_delimited_identifier를 끔으로 설정하고 테이블을 만들 때 대소문자를 구분하는 열 이름을 설정할 수 있습니다. downcase_delimited_identifier를 켬으로 설정하고 테이블을 쿼리하면 열 이름이 소문자가 됩니다. 이렇게 하면 downcase_delimited_identifier를 껐을 때 다른 쿼리 결과가 생성될 수 있습니다. 다음 예제를 검토하십시오.

```
SET downcase_delimited_identifier TO off;
--Amazon Redshift preserves case for column names and other identifiers.

--Create a table with two columns that are identical except for the case.
CREATE TABLE t ("c" int, "C" int);

INSERT INTO t VALUES (1, 2);

SELECT * FROM t;

 c | C
---+---
```

```
 1 | 2
(1 row)

SET enable_downcase_delimited_identifier TO on;
--Amazon Redshift no longer preserves case for column names and other identifiers.

SELECT * FROM t;

 c | c
---+---
 1 | 1
(1 row)
```

- 동적 데이터 마스킹 또는 행 수준 보안 정책이 첨부된 테이블을 쿼리하는 일반 사용자는 기본 downcase_delimited_identifier 설정을 사용하는 것이 좋습니다. 행 수준 보안에 대한 자세한 내용은 행 수준 보안을 참조하세요. 동적 데이터 마스킹에 대한 자세한 내용은 동적 데이터 마스킹 섹션을 참조하세요.

enable_case_sensitive_identifier

값(기본값은 굵은 글꼴로 표시)

true, false

설명

데이터베이스, 테이블 및 열의 이름 식별자가 대/소문자를 구분하는지 여부를 결정하는 구성 값입니다. 이름 식별자의 대/소문자는 큰따옴표로 묶인 경우 유지됩니다. enable_case_sensitive_identifier를 true로 설정하면 이름 식별자의 대/소문자가 유지됩니다. enable_case_sensitive_identifier를 false로 설정하면 이름 식별자의 대/소문자가 유지되지 않습니다.

큰따옴표로 묶여 있는 사용자 이름의 대/소문자는 enable_case_sensitive_identifier 구성 옵션 설정과 관계없이 항상 유지됩니다.

예시

다음 예에서는 테이블 및 열 이름에 대한 대/소문자 구분 식별자를 생성하고 사용하는 방법을 보여줍니다.

```
-- To create and use case sensitive identifiers
SET enable_case_sensitive_identifier TO true;

-- Create tables and columns with case sensitive identifiers
CREATE TABLE "MixedCasedTable" ("MixedCasedColumn" int);

CREATE TABLE MixedCasedTable (MixedCasedColumn int);

-- Now query with case sensitive identifiers
SELECT "MixedCasedColumn" FROM "MixedCasedTable";

MixedCasedColumn
------------------
(0 rows)


SELECT MixedCasedColumn FROM MixedCasedTable;

mixedcasedcolumn
------------------
(0 rows)
```

다음 예에서는 식별자의 대/소문자가 유지되지 않는 경우를 보여줍니다.

```
-- To not use case sensitive identifiers
RESET enable_case_sensitive_identifier;

-- Mixed case identifiers are lowercased
CREATE TABLE "MixedCasedTable2" ("MixedCasedColumn" int);

CREATE TABLE MixedCasedTable2 (MixedCasedColumn int);

ERROR:  Relation "mixedcasedtable2" already exists


SELECT "MixedCasedColumn" FROM "MixedCasedTable2";

 mixedcasedcolumn
------------------
(0 rows)


SELECT MixedCasedColumn FROM MixedCasedTable2;
```

```
 mixedcasedcolumn
------------------
(0 rows)
```

사용 관련 참고 사항

- 구체화된 뷰에 자동 새로 고침을 사용하는 경우 클러스터 또는 작업 그룹의 파라미터 그룹에서 enable_case_sensitive_identifier 값을 설정하는 것이 좋습니다. 이렇게 하면 구체화된 뷰가 새로 고쳐질 때 enable_case_sensitive_identifier가 일정하게 유지됩니다. 구체화된 뷰의 자동 새로 고침에 대한 자세한 내용은 구체화된 뷰 새로 고침 섹션을 참조하세요. 파라미터 그룹에서 구성 값을 설정하는 방법에 대한 자세한 내용은 Amazon Redshift 관리 안내서의 Amazon Redshift 파라미터 그룹을 참조하세요.
- 행 수준 보안 또는 동적 데이터 마스킹 기능을 사용하는 경우 클러스터 또는 워크그룹의 파라미터 그룹에서 enable_case_sensitive_identifier 값을 설정하는 것이 좋습니다. 이렇게 하면 정책을 만들고 연결한 다음 정책이 적용된 관계를 쿼리하는 동안 enable_case_sensitive_identifier가 일정하게 유지됩니다. 행 수준 보안에 대한 자세한 내용은 행 수준 보안 섹션을 참조하세요. 동적 데이터 마스킹에 대한 자세한 내용은 동적 데이터 마스킹 섹션을 참조하세요.
- enable_case_sensitive_identifier를 켬으로 설정하고 테이블을 만들 때 대소문자를 구분하는 열 이름을 설정할 수 있습니다. enable_case_sensitive_identifier를 끔으로 설정하고 테이블을 쿼리하면 열 이름이 소문자가 됩니다. 이렇게 하면 enable_case_sensitive_identifier를 켰을 때 다른 쿼리 결과가 생성될 수 있습니다. 다음 예제를 검토하십시오.

```
SET enable_case_sensitive_identifier TO on;
--Amazon Redshift preserves case for column names and other identifiers.

--Create a table with two columns that are identical except for the case.
CREATE TABLE t ("c" int, "C" int);

INSERT INTO t VALUES (1, 2);

SELECT * FROM t;

 c | C
---+---
 1 | 2
(1 row)
```

```
SET enable_case_sensitive_identifier TO off;
--Amazon Redshift no longer preserves case for column names and other identifiers.

SELECT * FROM t;

 c | c
---+---
 1 | 1
(1 row)
```

- 동적 데이터 마스킹 또는 행 수준 보안 정책이 첨부된 테이블을 쿼리하는 일반 사용자는 기본 enable_case_sensitive_identifier 설정을 사용하는 것이 좋습니다. 행 수준 보안에 대한 자세한 내용은 행 수준 보안 섹션을 참조하세요. 동적 데이터 마스킹에 대한 자세한 내용은 동적 데이터 마스킹 섹션을 참조하세요.

enable_case_sensitive_super_attribute

값(기본값은 굵은 글꼴로 표시)

true, false

설명

속성 이름이 구분되지 않은 SUPER 데이터 형식 구조를 탐색할 때 대소문자를 구분할지 여부를 결정하는 구성 값입니다. enable_case_sensitive_super_attribute를 true로 설정하면 속성 이름이 구분되지 않은 SUPER 형식 구조를 탐색할 때 대소문자를 구분합니다. 값을 false로 설정하면 속성 이름이 구분되지 않은 SUPER 형식 구조를 탐색할 때 대소문자를 구분하지 않습니다.

속성 이름을 큰따옴표로 묶은 다음 enable_case_sensitive_identifier를 true로 설정하면 enable_case_sensitive_super_attribute 구성 옵션 설정과 관계없이 대소문자를 항상 구분합니다.

enable_case_sensitive_super_attribute는 SUPER 데이터 유형이 있는 열에만 적용됩니다. 다른 모든 열의 경우 enable_case_sensitive_identifier를 대신 사용하는 것이 좋습니다.

SUPER 데이터 유형에 대한 자세한 내용은 SUPER 형식 및 Amazon Redshift의 반정형 데이터 섹션을 참조하세요.

예시

다음 예제에서는 enable_case_sensitive_super_attribute가 활성화되었을 때와 비활성화되었을 때 SUPER 값을 선택한 결과를 보여 줍니다.

```
--Create a table with a SUPER column.
CREATE TABLE tbl (col SUPER);

--Insert values.
INSERT INTO tbl VALUES (json_parse('{
 "A": "A", "a": "a"
}'));

SET enable_case_sensitive_super_attribute TO ON;

SELECT col.A FROM tbl;
  a
-----
 "A"
(1 row)

SELECT col.a FROM tbl;
  a
-----
 "a"
(1 row)

SET enable_case_sensitive_super_attribute TO OFF;

SELECT col.A FROM tbl;
  a
-----
 "a"
(1 row)

SELECT col.a FROM tbl;
  a
-----
 "a"
(1 row)
```

사용 관련 참고 사항

- 뷰와 구체화된 뷰는 생성 당시의 enable_case_sensitive_super_attribute 값을 따릅니다. 지연 바인딩 뷰, 저장 프로시저 및 사용자 정의 함수는 쿼리 시점의 enable_case_sensitive_super_attribute 값을 따릅니다.
- 구체화된 뷰에 자동 새로 고침을 사용하는 경우 클러스터 또는 작업 그룹의 파라미터 그룹에서 enable_case_sensitive_identifier value을 설정하는 것이 좋습니다. 이렇게 하면 구체화된 뷰가 새로 고쳐질 때 enable_case_sensitive_identifier가 일정하게 유지됩니다. 구체화된 뷰의 자동 새로 고침에 대한 자세한 내용은 구체화된 뷰 새로 고침 섹션을 참조하세요. 파라미터 그룹에서 구성 값을 설정하는 방법에 대한 자세한 내용은 Amazon Redshift 관리 안내서의 Amazon Redshift 파라미터 그룹을 참조하세요.
- 문 결과의 열 이름은 enable_case_sensitive_super_attribute 값에 관계없이 항상 소문자로 표시됩니다. 열 이름도 대소문자를 구분하도록 하려면 enable_case_sensitive_identifier를 활성화합니다.
- 행 수준 보안 정책이 첨부된 테이블을 쿼리하는 일반 사용자는 기본 enable_case_sensitive_identifier 설정을 사용하는 것이 좋습니다. 행 수준 보안에 대한 자세한 내용은 행 수준 보안을 참조하세요.

enable_numeric_rounding

값(기본값은 굵은 글꼴로 표시)

on(참), off(거짓)

설명

숫자 반올림 사용 여부를 지정합니다. enable_numeric_rounding이 on인 경우 Amazon Redshift는 NUMERIC 값을 INTEGER 또는 DECIMAL과 같은 다른 숫자 유형으로 변환할 때 해당 값을 반올림합니다. enable_numeric_rounding이 off인 경우 Amazon Redshift는 NUMERIC 값을 다른 숫자 유형으로 변환할 때 해당 값을 자릅니다. 숫자 유형에 대한 자세한 내용은 숫자형 섹션을 참조하세요.

예제

```
--Create a table and insert the numeric value 1.5 into it.
CREATE TABLE t (a numeric(10, 2));

INSERT INTO t VALUES (1.5);
```

```
SET enable_numeric_rounding to ON;
--Amazon Redshift now rounds NUMERIC values when casting to other numeric types.

SELECT a::int FROM t;

 a
---
 2
(1 row)


SELECT a::decimal(10, 0) FROM t;

 a
---
 2
(1 row)


 SELECT a::decimal(10, 5) FROM t;

    a
---------
 1.50000
(1 row)


SET enable_numeric_rounding to OFF;
--Amazon Redshift now truncates NUMERIC values when casting to other numeric types.

SELECT a::int FROM t;

 a
---
 1
(1 row)


SELECT a::decimal(10, 0) FROM t;

 a
---
 1
```

```
(1 row)


SELECT a::decimal(10, 5) FROM t;

    a
---------
 1.50000
(1 row)
```

enable_result_cache_for_session

값(기본값은 굵은 글꼴로 표시)

on (true) , off (false)

설명

쿼리 결과 캐싱을 사용할지 여부를 지정합니다. `enable_result_cache_for_session`이 `on`이면 Amazon Redshift는 쿼리가 제출 될 때 쿼리 결과의 유효한 캐시 된 복사본을 확인합니다. 결과 캐시에서 일치 항목이 발견되면 Amazon Redshift는 캐시된 결과를 사용하고 쿼리를 실행하지 않습니다. `enable_result_cache_for_session`이 `off`이면 Amazon Redshift는 결과 캐시를 무시하고 제출되는 모든 쿼리를 실행합니다.

예제

```
SET enable_result_cache_for_session TO off;
--Amazon Redshift now ignores the results cache
```

enable_vacuum_boost

값(기본값은 굵은 글꼴로 표시)

false, true

설명

세션에서 실행되는 모든 VACUUM 명령에 대해 VACUUM BOOST 옵션을 사용할지 여부를 지정합니다. `enable_vacuum_boost`가 `true`이면 Amazon Redshift가 세션의 모든 VACUUM 명령을 BOOST

옵션과 함께 실행합니다. enable_vacuum_boost가 false이면 Amazon Redshift가 기본적으로 BOOST 옵션과 함께 실행하지 않습니다. BOOST 옵션에 대한 자세한 내용은 VACUUM 섹션을 참조하세요.

error_on_nondeterministic_update

값(기본값은 굵은 글꼴로 표시)

false, true

설명

행당 일치 항목이 여러 개인 UPDATE 쿼리에서 오류가 발생하는지 여부를 지정합니다.

예제

```
SET error_on_nondeterministic_update TO true;

CREATE TABLE t1(x1 int, y1 int);

CREATE TABLE t2(x2 int, y2 int);

INSERT INTO t1 VALUES (1,10), (2,20), (3,30);

INSERT INTO t2 VALUES (2,40), (2,50);

UPDATE t1 SET y1=y2 FROM t2 WHERE x1=x2;

ERROR: Found multiple matches to update the same tuple.
```

extra_float_digits

값(기본값은 굵은 글꼴로 표시)

0 , -15-2

설명

float4와 float8을 포함하여 부동 소수점 값으로 표시할 자릿수를 설정합니다. 이 값은 표준 자릿수(FLT_DIG 또는 DBL_DIG)에 합산됩니다. 부분적 유효 자릿수를 포함하도록 최대 2까지 설정할 수 있

습니다. 이는 정확히 복원해야 하는 부동 소수점 데이터를 출력할 때 특히 유용합니다. 또는 음의 값으로 설정하여 불필요한 자릿수를 제거할 수도 있습니다.

예제

다음 예시에서는 extra_float_digits를 -2로 설정합니다. 먼저 현재 파라미터 설정을 표시합니다.

```
show all;
      name                      | setting
--------------------------+----------------
analyze_threshold_percent | 10
datestyle                 | ISO, MDY
extra_float_digits        | 2
query_group               | default
search_path               | $user, public
statement_timeout         | 0
timezone                  | UTC
wlm_query_slot_count      | 1
```

그런 다음 새 값을 -2로 설정합니다.

```
set extra_float_digits to -2;
```

마지막으로 업데이트된 파라미터 설정을 표시합니다.

```
show all;
      name                      | setting
--------------------------+----------------
analyze_threshold_percent | 10
datestyle                 | ISO, MDY
extra_float_digits        | -2
query_group               | default
search_path               | $user, public
statement_timeout         | 0
timezone                  | UTC
wlm_query_slot_count      | 1
```

interval_forbid_composite_literals

값(기본값은 굵은 글꼴로 표시)

false, true

설명

YEAR TO MONTH 및 DAY TO SECOND 부분을 모두 포함하는 간격의 값을 한정하는 세션 구성입니다.

`interval_forbid_composite_literals`가 `true`이면 간격에 YEAR TO MONTH와 DAY TO SECOND 부분이 모두 있는 경우 오류가 반환됩니다. 예를 들어, 다음 SQL에는 INTERVAL DAY TO SECOND와 함께 YEAR TO MONTH와 DAY TO SECOND 부분이 모두 포함되어 있습니다.

```
SELECT INTERVAL '1 year 1 day' DAY TO SECOND;
ERROR:  Interval Day To Second literal cannot contain year-month parts. Disable the GUC
 interval_forbid_composite_literals to suppress this error and silently discard the
 year-month part.
```

`interval_forbid_composite_literals`가 `false`이면 Amazon Redshift는 오류를 억제하고 INTERVAL DAY TO SECOND 값에서 YEAR TO MONTH 부분을 잘라냅니다. 예를 들어, 다음 SQL에는 INTERVAL DAY TO SECOND와 함께 YEAR TO MONTH와 DAY TO SECOND 부분이 모두 포함되어 있습니다.

```
SET interval_forbid_composite_literals to "false";
SELECT INTERVAL '1 year 1 day' DAY TO SECOND;


 intervald2s
------------------------------
1 days 0 hours 0 mins 0.0 secs
```

json_serialization_enable

값(기본값은 굵은 글꼴로 표시)

false, true

설명

ORC, JSON, Ion 및 Parquet 형식 데이터의 JSON 직렬화 동작을 수정하는 세션 구성입니다. `json_serialization_enable`이 `true`이면 모든 최상위 컬렉션이 자동으로 JSON으로 직렬화되고 VARCHAR(65535)로 반환됩니다. 복잡하지 않은 열은 영향을 받거나 직렬화되지 않습니다. 컬렉션 열은 VARCHAR(65535)로 직렬화되기 때문에 중첩 하위 필드는 더 이상 쿼리 구문의 일부로(즉, 필터 절에서) 직접 액세스할 수 없습니다. `json_serialization_enable`이 `false`이면 최상위 컬렉션이 JSON으로 직렬화되지 않습니다. 중첩 JSON 직렬화에 대한 자세한 내용은 복잡한 중첩 JSON 직렬화 섹션을 참조하세요.

json_serialization_parse_nested_strings

값(기본값은 굵은 글꼴로 표시)

false, true

설명

ORC, JSON, Ion 및 Parquet 형식 데이터의 JSON 직렬화 동작을 수정하는 세션 구성입니다. `json_serialization_parse_nested_strings`와 `json_serialization_enable`이 둘 다 `true`이면 복합 형식(예: 맵, 구조 또는 배열)에 저장된 문자열 값이 구문 분석되고 유효한 JSON인 경우 결과에 인라인으로 직접 작성됩니다. `json_serialization_parse_nested_strings`가 false이면 중첩 복합 형식 내의 문자열이 이스케이프 처리된 JSON 문자열로 직렬화됩니다. 자세한 내용은 JSON 문자열을 포함하는 복합 형식 직렬화 단원을 참조하십시오.

max_concurrency_scaling_clusters

값(기본값은 굵은 글꼴로 표시)

1 , 0 ~ 10

설명

동시성 확장이 활성화될 때 허용되는 최대 동시성 확장 클러스터 수를 설정합니다. 더 많은 동시성 확장이 필요하면 이 값을 늘립니다. 동시성 확장 클러스터 사용 및 해당 사용에 대한 요금 결제를 줄이려면 이 값을 줄입니다.

동시성 확장 클러스터의 최대 수는 조정 가능한 할당량입니다. 자세한 내용은 Amazon Redshift 관리 가이드의 Amazon Redshift 할당량 섹션을 참조하세요.

max_cursor_result_set_size

값(기본값은 굵은 글꼴로 표시)

0 (기본값은 최대 값) - 14400000MB

설명

`max_cursor_result_set_size` 파라미터는 더 이상 사용되지 않습니다. 커서 결과 집합 크기에 대한 자세한 내용은 커서 제약 조건 섹션을 참조하세요.

mv_enable_aqmv_for_session

값(기본값은 굵은 글꼴로 표시)

true, false

설명

Amazon Redshift가 세션 수준에서 구체화된 뷰의 자동 쿼리 재작성을 수행할 수 있는지 여부를 지정합니다.

navigate_super_null_on_error

값(기본값은 굵은 글꼴로 표시)

on, off

설명

존재하지 않는 객체 멤버 또는 배열 요소를 탐색하려고 할 때 쿼리가 기본 lax 모드에서 실행되는 경우 Amazon Redshift가 NULL 값을 반환하도록 지정합니다.

parse_super_null_on_error

값(기본값은 굵은 글꼴로 표시)

off, on

설명

Amazon Redshift에서 존재하지 않는 객체 멤버 또는 배열 요소를 구문 분석하려고 할 때 쿼리가 strict 모드에서 실행되는 경우 Amazon Redshift가 NULL 값을 반환하도록 지정합니다.

pg_federation_repeatable_read

값(기본값은 굵은 글꼴로 표시)

true, false

설명

PostgreSQL 데이터베이스의 결과에 대한 페더레이션 쿼리 트랜잭션 격리 수준을 지정합니다.

- pg_federation_repeatable_read가 true인 경우 페더레이션 트랜잭션은 REPEATABLE READ 격리 수준 시맨틱으로 처리됩니다. 이 값이 기본값입니다.
- pg_federation_repeatable_read가 false인 경우 페더레이션 트랜잭션은 READ COMMITTED 격리 수준 시맨틱으로 처리됩니다.

자세한 내용은 다음 자료를 참조하세요.

- Amazon Redshift를 사용하여 페더레이션 데이터에 액세스할 때의 고려 사항.
- 동시 쓰기 작업 관리.

예시

다음 명령은 한 세션 동안 pg_federation_repeatable_read를 on으로 설정합니다. show 명령은 설정된 값의 값을 표시합니다.

```
set pg_federation_repeatable_read to on;
```

```
show pg_federation_repeatable_read;

pg_federation_repeatable_read
-----------------------------
on
```

query_group

값(기본값은 굵은 글꼴로 표시)

기본값이 따로 없으며 어떤 문자열이든지 될 수 있습니다.

설명

동일한 세션에서 실행 중인 쿼리 그룹에 사용자 정의 레이블을 적용합니다. 이 레이블은 쿼리 로그에 캡처됩니다. 이를 사용하여 STL_QUERY 및 STV_INFLIGHT 테이블과 SVL_QLOG 뷰의 결과를 제한할 수 있습니다. 예를 들어 실행하는 쿼리마다 별도의 레이블을 적용하면 ID를 조회하지 않고도 쿼리를 고유하게 식별할 수 있습니다.

이 파라미터는 서버 구성 파일에 존재하지 않기 때문에 SET 명령으로 런타임에 설정해야 합니다. 긴 문자열을 레이블로 사용할 수는 있지만 STL_QUERY 테이블과 SVL_QLOG 뷰의 LABEL 열에서는 레이블의 문자 수가 30자로, 그리고 STV_INFLIGHT 테이블에서는 15자로 잘립니다.

다음 예에서 query_group은 **Monday**로 설정되고 해당 레이블로 여러 개의 쿼리가 실행됩니다.

```
set query_group to 'Monday';
SET
select * from category limit 1;
...
...
select query, pid, substring, elapsed, label
from svl_qlog where label ='Monday'
order by query;

query | pid  |        substring                    |  elapsed  | label
------+------+-------------------------------------+-----------+--------
789   | 6084 | select * from category limit 1;     | 65468     | Monday
790   | 6084 | select query, trim(label) from ... | 1260327   | Monday
791   | 6084 | select * from svl_qlog where ..     | 2293547   | Monday
792   | 6084 | select count(*) from bigsales;      | 108235617 | Monday
...
```

search_path

값(기본값은 굵은 글꼴로 표시)

'$ user', 공개, schema_names

기존 스키마 이름이 쉼표로 구분된 목록입니다. '$ user'가 있으면 SESSION_USER 과 같은 이름을 가진 스키마가 대체되고, 그렇지 않으면 무시됩니다.

설명

스키마 구성요소가 없는 단순 이름으로 객체(테이블, 함수 등)를 참조할 때 스키마의 검색 순서를 지정합니다.

- 외부 스키마 및 테이블에서는 검색 경로가 지원되지 않습니다. 외부 테이블은 명시적으로 외부 스키마로 정규화해야 합니다.
- 특정 대상 스키마 없이 생성된 객체는 검색 경로에서 첫 번째 스키마 자리에 위치합니다. 검색 경로가 비어 있으면 시스템이 오류를 반환합니다.
- 다른 스키마에 동일한 이름의 객체가 존재할 경우 검색 경로에서 먼저 발견되는 객체가 사용됩니다.
- 검색 경로에서 어떤 스키마에도 존재하지 않는 객체는 정규화(점으로 구분된) 이름으로 자신이 속하는 스키마를 지정해야만 참조할 수 있습니다.
- 시스템 카탈로그 스키마인 pg_catalog는 항상 검색됩니다. 이 스키마가 경로에 언급되면 지정된 순서에 따라 검색됩니다. 그렇지 않으면 모든 경로 항목 이전에 검색됩니다.
- 현재 세션의 임시 테이블 스키마인 pg_temp_nnn은 존재하는 경우에 한해 항상 검색됩니다. 별칭인 pg_temp를 사용하여 경로에 명시적으로 나열할 수도 있습니다. 경로에 나열되지 않으면 pg_catalog까지도 앞서서 가장 먼저 검색됩니다. 다만 임시 스키마에서는 릴레이션 이름(테이블, 뷰)만 검색됩니다. 함수 이름은 검색되지 않습니다.

예제

다음은 ENTERPRISE 스키마를 생성한 후 search_path를 새로운 스키마로 설정하는 예입니다.

```
create schema enterprise;
set search_path to enterprise;
show search_path;

 search_path
-------------
 enterprise
(1 row)
```

다음은 ENTERPRISE 스키마를 기본 search_path에 추가하는 예입니다.

```
set search_path to '$user', public, enterprise;
```

```
show search_path;

         search_path
-----------------------------
 "$user", public, enterprise
(1 row)
```

다음은 FRONTIER 테이블을 ENTERPRISE 스키마에 추가하는 예입니다.

```
create table enterprise.frontier (c1 int);
```

이때 PUBLIC.FRONTIER 테이블을 동일한 데이터베이스에 생성한 후에 사용자가 쿼리에서 스키마 이름을 지정하지 않으면 PUBLIC.FRONTIER가 ENTERPRISE.FRONTIER보다 우선합니다.

```
create table public.frontier(c1 int);
insert into enterprise.frontier values(1);
select * from frontier;

frontier
----
(0 rows)

select * from enterprise.frontier;

c1
----
1
(1 row)
```

spectrum_enable_pseudo_columns

값(기본값은 굵은 글꼴로 표시)

true, false

설명

spectrum_enable_pseudo_columns 구성 파라미터를 false로 설정하여 세션에 대해 가상 열 생성을 비활성화할 수 있습니다.

예제

다음 명령으로 세션에 대해 가상 열 생성을 비활성화할 수 있습니다.

```
set spectrum_enable_pseudo_columns to false;
```

enable_spectrum_oid

값(기본값은 굵은 글꼴로 표시)

true, false

설명

enable_spectrum_oid 구성 파라미터를 false로 설정하여 $spectrum_oid 가상 열만 비활성화할 수도 있습니다.

예제

다음 명령은 enable_spectrum_oid 구성 파라미터를 false로 설정하여 $spectrum_oid 가상 열을 비활성화합니다.

```
set enable_spectrum_oid to false;
```

spectrum_query_maxerror

값(기본값은 굵은 글꼴로 표시)

-1, 정수

설명

쿼리를 취소하기 전에 허용할 최대 오류 수를 나타내는 정수를 지정할 수 있습니다. 음수 값이면 최대 오류 데이터 처리가 해제됩니다. 결과는 SVL_SPECTRUM_SCAN_ERROR에 로그됩니다.

예제

다음 예제에서는 ORC 데이터에 잉여 문자와 유효하지 않은 문자가 포함되어 있다고 가정합니다. my_string의 열 정의에서는 3자 길이를 지정합니다. 다음은 이 예제의 샘플 데이터입니다.

```
my_string
---------------
abcdef
gh
ab
```

다음 명령은 최대 오류 수를 1로 설정하고 쿼리를 수행합니다.

```
set spectrum_query_maxerror to 1;
SELECT my_string FROM orc_data;
```

쿼리가 중지되고 결과가 SVL_SPECTRUM_SCAN_ERROR에 로그됩니다.

statement_timeout

값(기본값은 굵은 글꼴로 표시)

0 (제한 해제) , x 밀리 초

설명

지정한 시간(밀리초)을 초과하는 문을 모두 중지합니다.

`statement_timeout` 값은 Amazon Redshift에서 종료하기 전에 쿼리가 실행될 수 있는 최대 시간입니다. 이 시간에는 계획, 워크로드 관리(WLM)의 대기 및 실행 시간이 포함됩니다. 이 시간을 실행 시간만 포함하는 WLM 제한 시간(max_execution_time) 및 QMR(query_execution_time)과 비교하십시오.

WLM 구성에서 WLM 제한 시간(max_execution_time)도 지정하는 경우에는 statement_timeout과 max_execution_time 중에서 더 낮은 값이 사용됩니다. 자세한 내용은 WLM 제한 시간 단원을 참조하십시오.

예제

다음 쿼리는 1밀리초보다 오래 걸리기 때문에 시간 제한에 걸려 취소됩니다.

```
set statement_timeout = 1;

select * from listing where listid>5000;
```

```
ERROR:  Query (150) canceled on user's request
```

stored_proc_log_min_messages

값(기본값은 굵은 글꼴로 표시)

LOG, INFO, NOTICE, WARNING, EXCEPTION

설명

발생한 저장 프로시저 메시지의 최소 로깅 수준을 지정합니다. 지정된 수준 이상의 메시지가 로깅됩니다. 기본값은 LOG입니다(모든 메시지가 로깅됨). 로그 수준의 순서는 다음과 같이 가장 높은 수준에서 가장 낮은 순서까지입니다.

1. EXCEPTION
2. 경고
3. INFO
4. INFO
5. LOG

예를 들어, NOTICE의 값을 지정하면 메시지는 NOTICE, WARNING 및 EXCEPTION에 대해서만 로깅됩니다.

timezone

값(기본값은 굵은 글꼴로 표시)

UTC , 시간대

구문

```
SET timezone { TO | = } [ time_zone | DEFAULT ]

SET time zone [ time_zone | DEFAULT ]
```

설명

현재 세션의 시간대를 설정합니다. 시간대는 협정 세계시(UTC) 또는 시간대 이름의 오프셋이 될 수 있습니다.

> **Note**
>
> `timezone` 구성 파라미터는 클러스터 파라미터 그룹에서 설정할 수 없습니다. SET 명령을 사용하여 현재 세션의 시간대만 설정 가능합니다. 특정 데이터베이스 사용자가 실행하는 모든 세션의 시간대를 설정하려면 ALTER USER 명령을 사용해야 합니다. ALTER USER … SET TIMEZONE을 사용하면 현재 세션이 아닌 이후 모든 세션에 대한 시간대가 변경됩니다.

TO 또는 =과 함께 `SET timezone` (한 단어) 명령을 사용하여 시간대를 설정하면 time_zone 을 표준 시간대 이름, POSIX 스타일 형식 오프셋 또는 다음과 같이 ISO-8601 형식 오프셋을 사용합니다.

```
SET timezone { TO | = } time_zone
```

TO 또는 = 없이 SET time zone 명령어를 사용하여 시간대를 설정하면 다음과 같이 INTERVAL과 시간대 이름, POSIX 스타일 형식 오프셋 또는 ISO-8601 형식 오프셋을 사용하여 time_zone을 지정할 수 있습니다.

```
SET time zone time_zone
```

시간대 형식

Amazon Redshift는 다음 시간대 형식을 지원합니다.

- 시간대 이름
- INTERVAL
- POSIX-스타일 시간대 명세
- ISO-8601 오프셋

PST나 PDT 같은 시간대 약어는 UTC의 고정 오프셋으로 정의되어 일광 절약 시간 규칙이 없기 때문에 SET 명령에서는 시간대 약어를 지원하지 않습니다.

시간대 형식에 대한 자세한 내용은 아래를 참조하십시오.

시간대 이름 – America/New_York와 같은 전체 시간대 이름입니다. 시간대의 전체 이름에는 일광 절약 규칙도 포함될 수 있습니다.

다음은 시간대 이름의 예입니다.

- Etc/Greenwich
- America/New_York
- CST6CDT
- GB

> **Note**
>
> EST, MST, NZ, UCT 등 다수의 시간대 이름 역시 약어입니다.

유효한 시간대 이름 목록을 보려면 다음 명령을 실행하십시오.

```
select pg_timezone_names();
```

INTERVAL - UTC로부터의 오프셋입니다. 예를 들어 PST는 -8:00 또는 -8시간입니다.

다음은 INTERVAL 시간대 오프셋의 예입니다.

- -8:00
- -8시간
- 30 분

POSIX 스타일 형식 – STDoffset 또는 STDoffsetDST 형식의 시간대 지정입니다. 여기서 STD는 시간대 약어이고, offset은 UTC에서 서쪽으로 시간 단위로 나타낸 숫자 오프셋이며, DST는 선택적 일광 절약 시간대 약어입니다. 일광 절약 시간은 지정한 오프셋보다 1시간 앞서는 것으로 가정합니다.

POSIX-스타일 시간대 형식은 Greenwich에서 서쪽으로 양의 오프셋을 사용하는 반면 ISO-8601 규약은 Greenwich에서 동쪽으로 양의 오프셋을 사용합니다.

다음은 POSIX-스타일 시간대의 예입니다.

- PST8
- PST8PDT

- EST5
- EST5EDT

> **Note**
>
> Amazon Redshift는 POSIX 스타일 시간대 사양의 유효성을 검사하지 않으므로 시간대를 잘못된 값으로 설정할 수 있습니다. 예를 들어 다음 명령을 시간대를 잘못된 값으로 설정하지만 오류를 반환하지 않습니다.
>
> ```
> set timezone to ‘xxx36’;
> ```

ISO-8601 오프셋 – ±[hh]:[mm] 형식의 UTC 오프셋입니다.

다음은 ISO-8601 오프셋의 예입니다.

- -8:00
- +7:30

예시

다음은 현재 세션의 시간대를 New York으로 설정하는 예입니다.

```
set timezone = 'America/New_York';
```

다음은 현재 세션의 시간대를 UTC-8(PST)로 설정하는 예입니다.

```
set timezone to '-8:00';
```

다음은 INTERVAL을 사용하여 시간대를 PST로 설정하는 예입니다.

```
set timezone interval '-8 hours'
```

다음은 현재 세션의 시간대를 시스템 기본 시간대(UTC)로 재설정하는 예입니다.

```
set timezone to default;
```

데이터베이스 사용자의 시간대를 설정하려면 ALTER USER … SET 문을 사용하십시오. 다음은 sbuser의 시간대를 New York으로 설정하는 예입니다. 이 사용자에게는 이후 모든 세션에서도 새로운 값이 계속 적용됩니다.

```
ALTER USER dbuser SET timezone to 'America/New_York';
```

use_fips_ssl

값(기본값은 굵은 글꼴로 표시)

true, false

설명

FIPS 준수 SSL 모드 사용 여부를 지정하는 파라미터 그룹 값입니다. `use_fips_ssl`이 `true`인 경우 FIPS 준수 SSL 모드가 사용됩니다. `use_fips_ssl`이 `false`인 경우 FIPS 준수 SSL 모드가 사용되지 않습니다. 자세한 내용은 Amazon Redshift 관리 안내서의 연결에 대한 보안 옵션 구성을 참조하세요.

Amazon Redshift 프로비저닝된 클러스터의 파라미터를 구성하려면 Amazon Redshift 관리 안내서의 파라미터 그룹 정보를 참조하세요. Redshift Serverless의 파라미터를 구성하려면 Amazon Redshift 관리 안내서의 Amazon Redshift 서버리스에 대한 FIPS 준수 SSL 연결 구성과 Redshift Serverless API 참조의 CreateWorkgroup 또는 UpdateWorkgroup을 참조하세요.

wlm_query_slot_count

값(기본값은 굵은 글꼴로 표시)

1 , 1 - 50 (서비스 클래스에 대해 사용 가능한 슬롯 (동시성 수준)의 수를 초과 할 수 없음)

설명

쿼리에서 사용할 쿼리 슬롯 수를 설정합니다.

워크로드 관리(WLM)는 대기열에 대해 설정된 동시성 수준에 따라 서비스 클래스의 슬롯을 예약합니다. 예를 들어 동시성 수준이 5로 설정된 경우 서비스 클래스에는 5개의 슬롯이 있습니다. WLM은 서비스 클래스에서 사용할 수 있는 메모리를 각 슬롯마다 균일하게 할당합니다. 자세한 내용은 워크로드 관리 단원을 참조하십시오.

> **Note**
>
> wlm_query_slot_count 값이 서비스 클래스에서 사용할 수 있는 슬롯 수(동시성 레벨)보다 크면 쿼리가 중단됩니다. 따라서 오류가 발생할 경우에는 wlm_query_slot_count를 허용 값으로 줄이십시오.

정리와 같이 성능이 메모리 할당 크기의 영향을 많이 받는 작업일 때는 wlm_query_slot_count 값을 높이면 성능을 개선할 수 있습니다. 특히 속도가 느린 정리 명령에서는 SVV_VACUUM_SUMMARY 보기에서 해당 레코드를 검사하세요. SVV_VACUUM_SUMMARY 뷰에서 sort_partitions 값과 merge_increments 값이 높은 경우에는(100에 가깝거나 더욱 높을 때) 다음에 해당 테이블에 대해 Vacuum을 실행할 때 wlm_query_slot_count 값을 높이는 것이 좋습니다.

wlm_query_slot_count 값을 높이면 동시에 실행 가능한 쿼리의 수가 줄어듭니다. 예를 들어 서비스 클래스의 동시성 레벨이 5이고, wlm_query_slot_count가 3으로 설정되어 있다고 가정하겠습니다. wlm_query_slot_count가 3으로 설정되어 있는 세션에서 쿼리를 실행하면 동일한 서비스 클래스에서 동시에 2개의 쿼리를 추가로 실행할 수 있습니다. 이후 쿼리는 현재 실행 중인 쿼리가 완료되어 슬롯에 여유가 생길 때까지 대기열에서 대기합니다.

예시

다음은 SET 명령을 사용하여 현재 세션이 지속되는 동안 wlm_query_slot_count 값을 설정하는 예입니다.

```
set wlm_query_slot_count to 3;
```

문서 이력

> **Note**
>
> Amazon Redshift의 새로운 기능에 대한 설명은 새로운 소식을 참조하세요.

다음 테이블에서는 2018년 5월 이후 Amazon Redshift 데이터베이스 개발자 안내서에 대한 중요한 설명서 변경 사항에 대해 설명합니다. 이 설명서에 대한 업데이트 알림을 받으려면 RSS 피드를 구독하면 됩니다.

API 버전: 2012-12-01

Amazon Redshift 관리 가이드의 변경 사항 목록은 Amazon Redshift 관리 가이드 문서 기록 섹션을 참조하세요.

각 릴리스에서 수정된 내용과 연결된 클러스터 버전을 포함하여 새로운 기능에 대한 자세한 내용은 클러스터 버전 기록을 참조하십시오.

변경 사항	설명	날짜
공간 3D 및 4D 지오메트리와 새로운 공간 함수 지원	이제 추가 공간 함수를 사용할 수 있으며 일부 함수에 3D 및 4D 지오메트리 지원이 추가되었습니다.	2021년 8월 19일
자동 테이블 최적화를 위한 열 압축 인코딩 지원	테이블에 대해 ENCODE AUTO 옵션을 지정하여 테이블의 모든 열에 대한 압축 인코딩을 자동으로 관리할 수 있습니다.	2021년 8월 3일
Amazon Redshift Data API를 사용하여 여러 SQL 문 또는 파라미터가 있는 SQL 문 지원	이제 Amazon Redshift Data API를 사용하여 여러 SQL 문 또는 파라미터가 있는 문을 실행할 수 있습니다.	2021년 7월 28일

열 수준 재정의를 사용하여 대/소문자를 구분하지 않는 데이터 정렬 지원	이제 CREATE DATABASE 문 내에서 COLLATE 절을 사용하여 기본 데이터 정렬을 지정할 수 있습니다.	2021년 6월 24일
계정 간 데이터 공유 지원	이제 AWS 계정 간에 데이터를 공유할 수 있습니다.	2021년 4월 30일
재귀적 CTE를 통한 계층적 데이터 쿼리 지원	이제 SQL에서 재귀적 공통 테이블 표현식(CTE)을 사용할 수 있습니다.	2021년 4월 29일
데이터베이스 간 쿼리 지원	이제 클러스터의 데이터베이스에서 데이터를 쿼리할 수 있습니다.	2021년 3월 10일
COPY 및 UNLOAD 명령에 대한 세분화된 액세스 제어 지원	이제 Amazon Redshift 클러스터의 특정 사용자 및 그룹에 COPY 및 UNLOAD 명령을 실행할 수 있는 권한을 부여하여 보다 세분화된 액세스 제어 정책을 생성할 수 있습니다.	2021년 1월 12일
기본 JSON 및 비정형 데이터 지원	이제 SUPER 데이터 형식을 정의할 수 있습니다.	2020년 12월 9일
MySQL에 대한 연합 쿼리 지원	이제 지원되는 MySQL 엔진에 연합 쿼리를 작성할 수 있습니다.	2020년 12월 9일
데이터 공유 지원	이제 Amazon Redshift 클러스터 간에 데이터를 공유할 수 있습니다.	2020년 12월 9일
자동 테이블 최적화 지원	이제 자동 배포 및 정렬 키를 정의할 수 있습니다.	2020년 12월 9일
Amazon Redshift 기계 학습 지원	이제 기계 학습 모델을 생성, 훈련 및 배포할 수 있습니다.	2020년 12월 8일

구체화된 뷰의 자동 새로 고침 및 쿼리 재작성 지원	이제 자동 새로 고침을 통해 구체화된 뷰를 최신 상태로 유지할 수 있으며 자동 재작성을 통해 쿼리 성능을 개선할 수 있습니다.	2020년 11월 11일
TIME 및 TIMETZ 데이터 형식 지원	이제 TIME 및 TIMETZ 데이터 형식으로 테이블을 생성할 수 있습니다. TIME 데이터 형식은 시간대 정보 없이 시간을 저장하고 TIMETZ는 시간대 정보를 포함하여 시간을 저장합니다.	2020년 11월 11일
Lambda UDF 및 토큰화 지원	이제 Lambda UDF를 작성하여 데이터의 외부 토큰화를 사용할 수 있습니다.	2020년 10월 26일
테이블 열 인코딩 변경 지원	이제 테이블 열 인코딩을 변경할 수 있습니다.	2020년 10월 20일
데이터베이스 간 쿼리 지원	이제 Amazon Redshift에서 클러스터의 여러 데이터베이스를 쿼리할 수 있습니다.	2020년 10월 15일
HyperLogLog 스케치 지원	이제 Amazon Redshift에서 HyperLogLogSketches를 저장하고 처리할 수 있습니다.	2020년 10월 2일
Apache Hudi 및 Delta Lake 지원	Redshift Spectrum용 외부 테이블 생성 기능이 향상되었습니다.	2020년 9월 24일
공간 데이터 쿼리 개선 사항 지원	개선 사항에는 shapefile 및 여러 가지 새로운 공간 SQL 함수 로드가 포함됩니다.	2020년 9월 15일

구체화된 뷰 지원 외부 테이블	Amazon Redshift에서 외부 데이터 원본을 참조하는 구체화된 뷰를 생성할 수 있습니다.	2020년 6월 19일
외부 테이블에 쓰기 지원	CREATE EXTERNAL TABLE AS SELECT를 실행하여 새 외부 테이블에 작성하거나 INSERT INTO를 실행하여 기존 외부 테이블에 데이터를 삽입하여 외부 테이블에 작성할 수 있습니다.	2020년 6월 8일
스키마의 스토리지 제어 지원	스키마의 스토리지 제어를 관리하는 명령 및 뷰에 대한 업데이트입니다.	2020년 6월 2일
연합 쿼리 일반 가용성에 대한 지원	연합 쿼리를 사용하여 데이터를 쿼리하는 방법에 대한 정보를 업데이트했습니다.	2020년 4월 16일
추가 공간 기능에 대한 지원	추가 공간 기능에 대한 설명을 추가했습니다.	2020년 4월 2일
구체화된 보기 정식 출시 지원	구체화된 보기는 클러스터 버전 1.0.13059부터 공식 출시되었습니다.	2020년 2월 19일
열 수준 권한 지원	열 수준 권한은 클러스터 버전 1.0.13059부터 사용할 수 있습니다.	2020년 2월 19일
ALTER TABLE	ALTER TABLE 명령을 ALTER DISTSTYLE ALL 절과 함께 사용하여 테이블의 배포 스타일을 변경할 수 있습니다.	2020년 2월 11일

연동 쿼리 지원	업데이트된 CREATE EXTERNAL SCHEMA를 사용하여 연동 쿼리를 설명하도록 가이드를 업데이트했습니다.	2019년 12월 3일
데이터 레이크 내보내기 지원	UNLOAD 명령의 새 파라미터를 설명하도록 가이드를 업데이트했습니다.	2019년 12월 3일
공간 데이터 지원	공간 데이터 지원을 설명하도록 가이드를 업데이트했습니다.	2019년 11월 21일
새로운 콘솔 지원	새 Amazon Redshift 콘솔을 설명하도록 안내서를 업데이트했습니다.	2019년 11월 11일
자동 테이블 정렬 지원	Amazon Redshift에서 테이블 데이터를 자동으로 정렬할 수 있습니다.	2019년 11월 7일
VACUUM BOOST 옵션 지원	테이블에 대해 VACUUM 명령을 수행할 때 BOOST 옵션을 사용할 수 있습니다.	2019년 11월 7일
기본 IDENTITY 열 지원	기본 IDENTITY 열이 있는 테이블을 만들 수 있습니다.	2019년 9월 19일
AZ64 압축 인코딩 지원	AZ64 압축 인코딩으로 일부 열을 인코딩할 수 있습니다.	2019년 9월 19일
쿼리 우선 순위 지원	자동 WLM 대기열의 쿼리 우선 순위를 설정할 수 있습니다.	2019년 8월 22일
AWS Lake Formation 지원	Amazon Redshift Spectrum과 함께 Lake Formation Data Catalog를 사용할 수 있습니다.	2019년 8월 8일

COMPUPDATE PRESET	COPY 명령을 COMPUPDATE PRESET과 함께 사용하여 Amazon Redshift에서 압축 인코딩을 선택하도록 지정할 수 있습니다.	2019년 6월 13일
ALTER COLUMN	ALTER COLUMN과 함께 ALTER TABLE 명령을 사용하여 VARCHAR 열 크기를 늘릴 수 있습니다.	2019년 5월 22일
저장 프로시저 지원	Amazon Redshift에서 PL/pgSQL 저장 프로시저를 정의할 수 있습니다.	2019년 4월 24일
자동 워크로드 관리(WLM) 구성 지원	Amazon Redshift가 자동 WLM과 함께 실행되도록 할 수 있습니다.	2019년 4월 24일
Zstandard로 언로드	UNLOAD 명령을 사용하여 Amazon S3에 언로드된 텍스트 및 쉼표로 구분된 값(CSV) 파일에 Zstandard 압축을 적용할 수 있습니다.	2019년 4월 3일
동시성 확장	동시성 조정이 사용되면 동시 읽기 쿼리의 증가를 처리하는 데 필요한 추가 클러스터 용량을 Amazon Redshift에서 자동으로 추가합니다.	2019년 3월 21일
CSV로 언로드	UNLOAD 명령을 사용하여 CSV 텍스트 형식으로 지정된 파일로 언로드할 수 있습니다.	2019년 3월 13일

AUTO 분산 스타일	자동 배포를 활성화하려면 CREATE TABLE 문을 사용하여 AUTO 배포 스타일을 지정하면 됩니다. 자동 분산을 사용하면 Amazon Redshift에서는 테이블 데이터를 기반으로 최적의 배포 스타일을 할당합니다. 배포 변경은 배경에서 몇 초 동안 발생합니다.	2019년 1월 23일
Parquet에서 COPY 실행 시 SMALLINT 지원	이제 COPY 명령이 Parquet 형식 파일에서 SMALLINT 데이터 형식을 사용하는 열로의 로드를 지원합니다. 자세한 내용은 열 기반 데이터 형식에서의 COPY 명령 섹션을 참조하세요.	2019년 1월 2일
DROP EXTERNAL DATABASE	DROP SCHEMA 명령과 함께 DROP EXTERNAL DATABASE 절을 포함해 외부 데이터베이스를 삭제할 수 있습니다.	2018년 12월 3일
교차 리전 UNLOAD	REGION 파라미터를 지정하여 다른 AWS리전에서 Amazon S3 버킷에 대해 UNLOAD를 실행할 수 있습니다.	2018년 10월 31일

자동 vacuum 삭제	Amazon Redshift는 백그라운드에서 VACUUM DELETE 작업을 자동으로 수행하기 때문에 DELETE ONLY vacuum을 실행해야 하는 경우는 거의 없습니다. Amazon Redshift는 로드가 감소한 기간 동안 VACUUM DELETE가 실행되도록 예약하고 부하가 많은 기간 동안 작업을 일시 중지합니다.	2018년 10월 31일
자동 분산	CREATE TABLE 문으로 배포 스타일을 지정하지 않은 경우 Amazon Redshift는 테이블 데이터를 기반으로 최적의 배포 스타일을 할당합니다. 배포 변경은 배경에서 몇 초 동안 발생합니다.	2018년 10월 31일
AWS Glue Data Catalog에 대한 세분화된 액세스 제어	이제 AWS Glue Data Catalog에 저장된 데이터에 대한 액세스 수준을 지정할 수 있습니다.	2018년 10월 15일
데이터 형식으로 UNLOAD	UNLOAD 명령과 함께 MANIFEST VERBOSE 옵션을 지정해 열의 이름 및 데이터 유형, 파일 크기 및 행 개수를 포함한 메타데이터를 매니페스트 파일에 추가할 수 있습니다.	2018년 10월 10일

단일 ALTER TABLE 문을 사용하여 여러 파티션 추가	Redshift Spectrum 외부 테이블의 경우 단일 ALTER TABLE ADD 문에서 PARTITION 절 여러 개를 결합할 수 있습니다. 자세한 내용은 외부 테이블 수정에 대한 예를 참조하세요.	2018년 10월 10일
UNLOAD 명령과 헤더	UNLOAD 명령에 HEADER 옵션을 사용하여 열 이름을 포함하는 헤더 라인을 각 출력 파일 맨 위에 추가할 수 있습니다.	2018년 9월 19일
새 시스템 테이블과 뷰	SVL_S3Retries, SVL_USER_INFO, STL_DISK_FULL_DIAG 설명서가 추가되었습니다.	2018년 8월 31일
Amazon Redshift Spectrum의 중첩 데이터 지원	이제 Amazon Redshift Spectrum 테이블에 저장된 중첩 데이터에 대해 쿼리를 실행할 수 있습니다. 자세한 내용은 자습서: Amazon Redshift Spectrum을 사용한 중첩 데이터에 대한 쿼리를 참조하세요.	2018년 8월 8일
SQA 기본 활성화	이제는 모든 새로운 클러스터에서 SQA(단기 쿼리 가속화)가 기본적으로 활성화됩니다. SQA는 기계 학습을 사용하여 쿼리 실행 시간의 성능이 높아지고 결과가 빨라지며 예측이 향상됩니다. 자세한 내용은 단기 쿼리 가속화 섹션을 참조하세요.	2018년 8월 8일

Amazon Redshift Advisor	이제 Amazon Redshift Advisor에서 클러스터 성능을 개선하고 운영 비용을 절감하는 방법에 대한 맞춤형 권장 사항을 확인할 수 있습니다. 자세한 내용은 Amazon Redshift 권장 사항를 참조하세요.	2018년 7월 26일
즉각적인 별칭 참조	이제 별칭 처리된 표현식을 정의한 후 즉시 참조할 수 있습니다. 자세한 내용은 목록 SELECT를 참조하세요.	2018년 7월 18일
외부 테이블을 생성할 때 압축 유형 지정	이제 Amazon Redshift Spectrum에서 외부 테이블을 생성할 때 압축 유형을 지정할 수 있습니다. 자세한 내용은 외부 테이블 생성 섹션을 참조하세요.	2018년 27월 6일
PG_LAST_UNLOAD_ID	새 시스템 정보 함수인 PG_LAST_UNLOAD_ID에 대한 설명서가 추가되었습니다. 자세한 내용은 PG_LAST_UNLOAD_ID를 참조하십시오.	2018년 27월 6일
ALTER TABLE RENAME COLUMN	이제 ALTER TABLE이 외부 테이블의 열 이름 바꾸기를 지원합니다. 자세한 내용은 외부 테이블 수정에 대한 예를 참조하세요.	2018년 6월 7일

이전 업데이트

다음 표에서는 2018년 6월 이전 Amazon Redshift 데이터베이스 개발자 안내서의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

변경 사항	설명	변경 날짜
Parquet의 COPY 명령에 SMALLINT 포함	이제 COPY 명령이 Parquet 형식 파일에서 SMALLINT 데이터 형식을 사용하는 열로의 로드를 지원합니다. 자세한 내용은 열 기반 데이터 형식에서 COPY 명령 섹션을 참조하세요.	2019년 1월 2일
열 기반 형식에서의 COPY 명령	이제 COPY 명령이 Parquet 및 ORC 열 기반 데이터 형식을 사용하는 Amazon S3의 파일에서 로드를 지원합니다. 자세한 내용은 열 기반 데이터 형식에서 COPY 명령 섹션을 참조하세요.	2018년 5월 17일
SQA에 대한 동적 최대 실행 시간	이제 WLM(Workload Management)이 기본적으로, 클러스터의 워크로드 분석에 따라 SQA(Short Query Acceleration) 최대 실행 시간 값을 동적으로 할당합니다. 자세한 내용은 단기 쿼리의 최대 실행 시간 단원을 참조하십시오.	2018년 5월 17일
STL_LOAD_COMMITS의 새 열	STL_LOAD_COMMITS 시스템 테이블에 `file_format` 이라는 새 열이 추가되었습니다.	2018년 5월 10일
STL_HASHJOIN 및 다른 시스템 로그 테이블의 새 열	STL_HASHJOIN 시스템 테이블에 `hash_segment` , `hash_step` , `checksum`이라는 3개의 열이 새로 추가되었습니다. 또한 `checksum`이 STL_MERGEJOIN, STL_NESTLOOP, STL_HASH, STL_SCAN, STL_SORT, STL_LIMIT, STL_PROJECT에 추가되었습니다.	2018년 5월 17일
STL_AGGR에 새로운 열 추가	STL_AGGR 시스템 테이블에 `resizes`와 `flushable` , 2개 열이 새롭게 추가되었습니다.	2018년 4월 19일
REGEX 함수의 새로운 옵션	REGEXP_INSTR 및 REGEXP_SUBSTR 함수의 경우, 이제 어떤 일치 항목을 사용할 것인지 결정하고 대소문자를 구분하여 비교할 것인지 여부를 지정할 수 있습니다. 또한 REGEXP_INSTR에서는 일치하는 항목의 첫 번째 문자 위치를 반환할지 아니면 일치하는 항목이 끝난 뒤의 첫 번째 문자 위치를 반환할지도 지정할 수 있습니다.	2018년 3월 22일

변경 사항	설명	변경 날짜
시스템 테이블에 새로 추가된 열	STL_COMMIT_STATS 시스템 테이블에 tombstone dblocks, tossedblocks, batched_by가 추가되었습니다. STV_SLICES 시스템 보기에는 localslice 열이 추가되었습니다.	2018년 3월 22일
외부 테이블 열 추가 및 삭제	이제 ALTER TABLE에서 Amazon Redshift Spectrum 외부 테이블에 ADD COLUMN 및 DROP COLUMN을 사용할 수 있습니다.	2018년 3월 22일
Redshift Spectrum 신규 AWS 리전	이제 뭄바이와 상파울루 리전에서 Redshift Spectrum을 사용할 수 있습니다. 지원되는 리전 목록은 Amazon Redshift Spectrum 리전 섹션을 참조하세요.	2018년 3월 22일
테이블 한도 20,000으로 증가	8xlarge 클러스터 노드 유형의 최대 테이블 개수가 20,000개가 되었습니다. 대형(large) 및 배수(xlarge) 노드 유형의 한도는 9,900개입니다. 자세한 내용은 제한 및 할당량 단원을 참조하십시오.	2018년 3월 13일
Redshift Spectrum의 JSON 및 Ion 지원	Redshift Spectrum을 사용하여 JSON 또는 Ion 데이터 형식의 스칼라 데이터가 포함된 파일을 참조할 수 있습니다. 자세한 내용은 CREATE EXTERNAL TABLE 단원을 참조하십시오.	2018년 2월 26일
Redshift Spectrum에서 IAM 역할 함께 묶기	AWS Identity and Access Management(IAM) 역할을 함께 묶어 클러스터가 다른 AWS 계정에 속한 역할을 포함하여 클러스터에 연결되지 않은 다른 역할을 수임할 수 있습니다. 자세한 내용은 Amazon Redshift Spectrum에서 IAM 역할 연결 단원을 참조하십시오.	2018년 2월 1일
ADD PARTITION이 IF NOT EXISTS를 지원	ALTER TABLE용 ADD PARTITION 절이 이제 IF NOT EXISTS 옵션을 지원합니다. 자세한 내용은 ALTER TABLE 단원을 참조하십시오.	2018년 1월 11일
외부 테이블용 DATE 데이터	Redshift Spectrum 외부 테이블이 이제 DATE 데이터 형식을 지원합니다. 자세한 내용은 CREATE EXTERNAL TABLE 단원을 참조하십시오.	2018년 1월 11일

변경 사항	설명	변경 날짜
Redshift Spectrum 신규 AWS 리전	이제 싱가포르, 시드니, 서울 및 프랑크푸르트 리전에서 Redshift Spectrum을 사용할 수 있습니다. 지원되는 AWS 리전 목록은 Amazon Redshift Spectrum 리전 섹션을 참조하세요.	2017년 11월 16일
Amazon Redshift 워크로드 관리(WLM)의 단기 쿼리 가속화	단기 쿼리 가속화(SQA)는 선택한 단기 실행 쿼리를 장기 실행 쿼리보다 우선적으로 적용합니다. SQA 쿼리가 대기열에서 장기 쿼리 뒤에서 대기해야 하지 않도록 SQA는 전용 공간에서 단기 실행 쿼리를 실행합니다. SQA가 있으면 단기 실행 쿼리가 더 빠르게 실행하기 시작하며 사용자가 더 빨리 결과를 확인합니다. 자세한 내용은 단기 쿼리 가속화 단원을 참조하십시오.	2017년 11월 16일
WLM에서 건너뛴 쿼리 다시 할당	건너뛴 쿼리를 취소하고 다시 시작하는 대신, Amazon Redshift 워크로드 관리(WLM)는 이제 사용 가능한 쿼리를 새 대기열에 다시 할당합니다. WLM은 쿼리를 다시 할당할 때 쿼리를 새 대기열로 이동하고 실행을 계속하여 시간과 시스템 리소스를 절약합니다. 다시 할당할 수 없는 건너뛴 쿼리는 다시 시작되거나 취소됩니다. 자세한 내용은 WLM 쿼리 대기열 건너뛰기 단원을 참조하십시오.	2017년 11월 16일
사용자에 대한 시스템 로그 액세스	사용자가 볼 수 있는 대부분의 시스템 로그 테이블에서 일반적으로 일반 사용자는 다른 사용자가 생성한 행을 볼 수 없습니다. 일반 사용자가 다른 사용자가 생성한 행을 포함하여 사용자 가시성 테이블에 있는 모든 행을 볼 수 있도록 허용하려면 ALTER USER 또는 사용자 생성를 실행하고 SYSLOG ACCESS 파라미터를 UNRESTRICTED로 설정합니다.	2017년 11월 16일

변경 사항	설명	변경 날짜
결과 캐싱	결과 캐싱 을 사용하면 쿼리를 실행할 때 Amazon Redshift가 결과를 캐시합니다. 쿼리를 다시 실행하면 Amazon Redshift는 쿼리 결과의 유효한 캐시된 복사본을 확인합니다. 결과 캐시에서 일치 항목이 발견되면 Amazon Redshift는 캐시된 결과를 사용하고 쿼리를 실행하지 않습니다. 결과 캐싱은 기본적으로 설정되어 있습니다. 결과 캐싱을 해체하려면 enable_result_cache_for_session 구성 파라미터를 off로 설정합니다.	2017년 11월 16일
열 메타데이터 함수	PG_GET_COLS 및 PG_GET_LATE_BINDING_VIEW_COLS는 Amazon Redshift 테이블, 뷰 및 Late Binding 뷰에 대한 열 메타데이터를 반환합니다.	2017년 11월 16일
CTAS의 WLM 대기열 건너뛰기	Amazon Redshift 워크로드 관리(WLM)는 이제 SELECT 문과 같은 읽기 전용 쿼리뿐 아니라 CREATE TABLE AS(CTAS) 문의 쿼리 대기열 건너뛰기를 지원합니다. 자세한 내용은 WLM 쿼리 대기열 건너뛰기 단원을 참조하십시오.	2017년 10월 19일
Amazon Redshift Spectrum 매니페스트 파일	Redshift Spectrum 외부 테이블을 생성할 때 Amazon S3의 데이터 파일 위치를 나열하는 매니페스트 파일을 지정할 수 있습니다. 자세한 내용은 CREATE EXTERNAL TABLE 단원을 참조하십시오.	2017년 10월 19일
Amazon Redshift Spectrum 신규 AWS 리전	이제 EU(아일랜드) 및 아시아 태평양(도쿄) 리전에서 Redshift Spectrum을 사용할 수 있습니다. 지원되는 AWS 리전 목록은 Amazon Redshift Spectrum 제한 사항 섹션을 참조하세요.	2017년 10월 19일
Amazon Redshift Spectrum 추가 파일 형식	이제 Regex, OpenCSV 및 Avro 데이터 파일 형식을 기반으로 Redshift Spectrum 외부 테이블을 생성할 수 있습니다. 자세한 내용은 CREATE EXTERNAL TABLE 단원을 참조하십시오.	2017년 10월 5일

변경 사항	설명	변경 날짜
Amazon Redshift Spectrum 외부 테이블용 가상 열	Redshift Spectrum 외부 테이블에서 `$path` 및 `$size` 가상 열을 선택하여 Amazon S3에 있는 참조된 데이터 파일의 위치와 크기를 볼 수 있습니다. 자세한 내용은 가상 열 단원을 참조하십시오.	2017년 10월 5일
JSON 유효성 검사 함수	IS_VALID_JSON 및 IS_VALID_JSON_ARRAY 함수를 사용하여 유효한 JSON 형식을 검사할 수 있습니다. 다른 JSON 함수에는 이제 선택 사항인 `null_if_invalid` 인수가 포함됩니다.	2017년 10월 5일
LISTAGG DISTINCT	LISTAGG 집계 함수 및 LISTAGG 창 함수와 함께 DISTINCT 절을 사용하여 연결 전에 지정된 표현식에서 중복 값을 제거할 수 있습니다.	2017년 10월 5일
열 이름 대문자로 보기	SELECT 결과에서 열 이름을 대문자로 보기 위해 describe_field_name_in_uppercase 구성 파라미터를 `true`로 설정할 수 있습니다.	2017년 10월 5일
외부 테이블에서 헤더 행 건너뛰기	`skip.header.line.count` 명령에서 CREATE EXTERNAL TABLE 속성을 설정하여 Redshift Spectrum 데이터 파일의 시작 부분에서 헤더 행을 건너뛸 수 있습니다.	2017년 10월 5일
스캔하는 행의 수	WLM 쿼리 모니터 규칙은 scan_row_count metric을 사용하여 스캔 단계에서 행 개수를 반환합니다. 행 개수는 삭제 대기 행(고스트 행)을 필터링하고 사용자 정의 쿼리 필터를 적용하기 전에 내보낸 행의 총 수입니다. 자세한 내용은 프로비저닝된 Amazon Redshift에 대한 쿼리 모니터링 지표 단원을 참조하십시오.	2017년 9월 21일
SQL 사용자 정의 함수	스칼라 SQL 사용자 정의 함수(UDF)에는 함수 호출 시 실행되어 단일 값을 반환하는 SQL SELECT 절이 포함됩니다. 자세한 내용은 Scalar SQL UDF 단원을 참조하십시오.	2017년 8월 31일

변경 사항	설명	변경 날짜
Late Binding 보기	Late Binding 보기는 테이블, 사용자 정의 함수 같은 기본 데이터베이스 객체에 바인딩되지 않습니다. 결과적으로 보기와 보기가 참조하는 객체 사이에 종속성이 없습니다. 참조 객체가 존재하지 않는 경우에도 보기를 새로 만들 수 있습니다. 종속성이 없기 때문에 뷰에 영향을 주지 않고서 참조 객체를 중단하거나 변경할 수 있습니다. Amazon Redshift는 뷰가 쿼리될 때까지 종속성을 확인하지 않습니다. Late Binding 보기를 새로 만들려면 CREATE VIEW 문으로 WITH NO SCHEMA BINDING 절을 지정하십시오. 자세한 내용은 CREATE VIEW 단원을 참조하십시오.	2017년 8월 31일
OCTET_LENGTH 함수	OCTET_LENGTH는 지정된 문자열의 길이를 바이트 수대로 반환합니다.	2017년 8월 18일
ORC 및 Grok 파일 형식 지원	Amazon Redshift Spectrum은 Redshift Spectrum 데이터 파일에 대하여 ORC 및 Grok 데이터 형식을 지원합니다. 자세한 내용은 Amazon Redshift Spectrum의 쿼리용 데이터 파일 단원을 참조하십시오.	2017년 8월 18일
RegexSerDe 지원	Amazon Redshift Spectrum이 이제 RegexSerDe 데이터 형식을 지원합니다. 자세한 내용은 Amazon Redshift Spectrum의 쿼리용 데이터 파일 단원을 참조하십시오.	2017년 7월 19일
SVV_TABLES 및 SVV_COLUMNS에 새로운 열 추가	`domain_name` 열과 `remarks` 열이 SVV_COLUMNS에 추가되었습니다. 그리고 SVV_TABLES에는 remarks 열이 추가되었습니다.	2017년 7월 19일
SVV_TABLES 및 SVV_COLUMNS 시스템 뷰	SVV_TABLES 및 SVV_COLUMNS 시스템 뷰가 열에 대한 정보를 비롯해 로컬 및 외부 테이블과 뷰에 대한 세부 정보까지 제공합니다.	2017년 7월 7일

변경 사항	설명	변경 날짜
Amazon Redshift Spectrum에서 Amazon EMR Hive 메타스토어를 사용하는 데 VPC 불필요	Redshift Spectrum에서 Amazon EMR Hive 메타스토어를 사용할 때 Amazon Redshift 클러스터와 Amazon EMR 클러스터가 동일한 VPC와 동일한 서브넷에 속해야 한다는 요건이 제거되었습니다. 자세한 내용은 Amazon Redshift Spectrum의 외부 카탈로그 작업 단원을 참조하십시오.	2017년 7월 7일
더 작은 파일 크기로 언로드	기본적으로 UNLOAD는 최대 크기가 6.2GB인 파일을 Amazon S3에 다수 생성합니다. 이때 더 작은 크기의 파일을 생성하려면 UNLOAD 명령에 MAXFILESIZE를 지정하십시오. 최대 파일 크기는 5MB에서 6.2GB까지 지정할 수 있습니다. 자세한 내용은 UNLOAD 단원을 참조하십시오.	2017년 7월 7일
TABLE PROPERTIES	CREATE EXTERNAL TABLE 또는 ALTER TABLE에서 TABLE PROPERTIES numRows 파라미터를 설정하여 테이블의 행 수를 반영하도록 테이블 통계를 업데이트합니다.	2017년 6월 6일
ANALYZE PREDICATE COLUMNS	시간과 클러스터 리소스를 절약할 목적으로 조건자로 사용 가능한 열만 분석할 수 있습니다. PREDICATE COLUMNS 절에서 ANALYZE를 실행할 때는 join, filter condition 또는 group by 절에서 사용되었거나, 혹은 정렬 키 또는 분산 키로 사용되는 열만 분석 작업에 포함됩니다. 자세한 내용은 테이블 분석 단원을 참조하십시오.	2017년 5월 25일
Amazon Redshift Spectrum에 대한 IAM 정책	Redshift Spectrum만 사용하여 Amazon S3 버킷에 대한 액세스 권한을 부여하려면 사용자 에이전트 `AWS Redshift/Spectrum` 에 대한 액세스를 허용하는 조건을 포함합니다. 자세한 내용은 Amazon Redshift Spectrum에 대한 IAM 정책 단원을 참조하십시오.	2017년 5월 25일

변경 사항	설명	변경 날짜
Amazon Redshift Spectrum 재귀 스캔	Redshift Spectrum이 이제는 하위 폴더의 파일과 Amazon S3에서 지정하는 폴더까지 스캔합니다. 자세한 내용은 Redshift Spectrum용 외부 테이블 단원을 참조하십시오.	2017년 5월 25일
쿼리 모니터링 규칙	WLM 쿼리 모니터링 규칙을 사용하여 WLM 대기열에 대한 지표 기반 성능 경계를 정의하고 쿼리가 이러한 경계(로그, 건너뛰기 또는 중단)를 벗어날 때 수행할 작업을 지정할 수 있습니다. 쿼리 모니터링 규칙은 워크로드 관리(WLM) 구성 시 정의합니다. 자세한 내용은 WLM 쿼리 모니터링 규칙 단원을 참조하십시오.	2017년 21월 4일
Amazon Redshift Spectrum	Redshift Spectrum을 사용하면 쿼리를 효율적으로 실행하여 데이터를 테이블에 로드하지 않고도 Amazon S3의 파일에서 데이터를 가져올 수 있습니다. Redshift Spectrum은 Amazon S3에서 직접 데이터 파일을 스캔하기 때문에 대용량 데이터 집합에서도 쿼리 실행 속도가 매우 빠릅니다. 대부분의 처리가 Amazon Redshift Spectrum 계층에서 이루어지며, 데이터가 대부분 Amazon S3에 그대로 남습니다. 또한 다수의 클러스터가 Amazon S3의 동일한 데이터 집합에 대해 동시에 쿼리를 실행할 수 있기 때문에 각 클러스터의 데이터를 일일이 복사할 필요가 없습니다. 자세한 내용은 Amazon Redshift Spectrum 섹션을 참조하세요.	2017년 4월 19일

변경 사항	설명	변경 날짜
Redshift Spectrum을 지원하는 새로운 시스템 테이블	Redshift Spectrum 지원을 위해 다음과 같이 새로운 시스템 뷰가 추가되었습니다. • SVL_S3QUERY • SVL_S3QUERY_SUMMARY • SVV_EXTERNAL_COLUMNS • SVV_EXTERNAL_DATABASES • SVV_EXTERNAL_PARTITIONS • SVV_EXTERNAL_TABLES • PG_EXTERNAL_SCHEMA	2017년 4월 19일
APPROXIMATE PERCENTILE_DISC 집계 함수	이제 APPROXIMATE PERCENTILE_DISC 집계 함수를 사용할 수 있습니다.	2017년 4월 4일
2017년 4월 4일	이제 AWS Key Management Service 키(SSE-KMS)를 사용한 서버 측 암호화로 데이터를 Amazon S3로 언로드할 수 있습니다. 그 밖에도 COPY가 KMS로 암호화된 데이터 파일을 Amazon S3에서 투명하게 로드합니다. 자세한 내용은 UNLOAD 단원을 참조하십시오.	2017년 2월 9일

변경 사항	설명	변경 날짜
새로운 권한 부여 구문	이제는 IAM_ROLE, MASTER_SYMMETRIC_KEY, ACCESS_KEY_ID, SECRET_ACCESS_KEY, SESSION_TOKEN 파라미터를 사용하여 COPY, UNLOAD 및 CREATE LIBRARY 명령에 필요한 권한 부여 및 액세스 정보를 제공할 수 있습니다. 새로운 권한 부여 구문은 단일 문자열 인수를 CREDENTIALS 파라미터에 입력하는 방법에 비해 더욱 유연한 대안이 될 것입니다. 자세한 내용은 권한 부여 파라미터 단원을 참조하십시오.	2017년 2월 9일
스키마 할당량 추가	이제 클러스터 1개당 최대 9,900개까지 스키마를 생성할 수 있습니다. 자세한 내용은 CREATE SCHEMA 단원을 참조하십시오.	2017년 2월 9일
기본 테이블 인코딩	이제 CREATE TABLE 및 ALTER TABLE이 대부분 새로운 열에 LZO 압축 인코딩을 할당합니다. 기본적으로 정렬 키로 정의된 열, BOOLEAN, REAL 또는 DOUBLE PRECISION 데이터 형식으로 정의된 열, 그리고 임시 테이블에 RAW 인코딩이 할당됩니다. 자세한 내용은 ENCODE 단원을 참조하십시오.	2017년 2월 6일
ZSTD 압축 인코딩	이제 Amazon Redshift는 ZSTD 열 압축 인코딩을 지원합니다.	2017년 1월 19일
PERCENTILE_CONT 및 MEDIAN 집계 함수	PERCENTILE_CONT 및 MEDIAN는 이제 창 함수 외에 집계 함수로도 사용할 수 있습니다.	2017년 1월 19일
사용자 정의 함수(UDF) 사용자 로깅	UDF에서 Python 로깅 모듈을 사용하여 사용자 정의 오류 및 경고 메시지를 생성할 수 있습니다. 쿼리를 실행한 후에는 SVL_UDF_LOG 시스템 뷰에 대한 쿼리를 실행하여 로그 메시지를 가져올 수도 있습니다. 사용자 정의 메시지에 대한 자세한 내용은 Python UDF에서 오류 및 경고 로깅 섹션을 참조하세요.	2016년 12월 8일

변경 사항	설명	변경 날짜
ANALYZE COMPRESSION 감소 추정	ANALYZE COMPRESSION 명령은 이제 각 열의 디스크 공간에서 추정되는 감소 비율(%)을 보고합니다. 자세한 내용은 ANALYZE COMPRESSION 단원을 참조하십시오.	2016년 11월 10일
연결 제한	이제 사용자 1명이 동시에 열 수 있는 데이터베이스 연결 수의 제한을 설정할 수 있습니다. 그 밖에 데이터베이스 1개에 대한 동시 연결 수도 제한할 수 있습니다. 자세한 내용은 사용자 생성 및 데이터베이스 생성 섹션을 참조하세요.	2016년 11월 10일
COPY 정렬 순서 강화	이제 정렬 키 순서대로 데이터를 로드할 때 COPY 명령을 실행하면 새로운 행이 정렬된 테이블 영역에 자동으로 추가됩니다. 이러한 순서 강화를 위한 특정 요건은 정렬 키 순서로 데이터 로드 섹션을 참조하세요.	2016년 11월 10일
압축을 지원하는 CTAS	CREATE TABLE AS(CTAS)가 이제 열의 데이터 형식에 따라 압축 인코딩을 새로운 테이블에 자동 할당합니다. 자세한 내용은 열 및 테이블 속성의 상속 단원을 참조하십시오.	2016년 10월 28일
시간대 데이터 형식을 포함한 타임스탬프	Amazon Redshift가 이제 시간대(TIMESTAMPTZ) 데이터 형식을 포함하여 타임스탬프를 지원합니다. 그 밖에도 새로운 데이터 형식을 지원하기 위해 몇 가지 새로운 함수가 추가되었습니다. 자세한 내용은 날짜 및 시간 함수 단원을 참조하십시오.	2016년 9월 29일
분석 임계값	Amazon Redshift는 마지막 ANALYZE 명령 이후 변경된 행의 비율이 ANALYZE 파라미터에서 지정하는 분석 임계값보다 낮으면 처리 시간을 줄이고 analyze_threshold_percent 작업의 전반적인 시스템 성능을 높일 목적으로 테이블 분석을 건너뜁니다. 기본적으로 `analyze_threshold_percent` 는 10입니다.	2016년 8월 9일

변경 사항	설명	변경 날짜
새로운 STL_RESTARTED_SESSIONS 시스템 테이블	Amazon Redshift가 세션을 다시 시작하면 STL_RESTARTED_SESSIONS가 새로운 프로세스 ID(PID)와 이전 PID를 기록합니다.	2016년 8월 9일
Date 및 Time 함수 설명서 업데이트	함수 요약에 날짜 및 시간 함수 링크가 추가되었고, 일관성 유지를 위해 함수 참조 설명서가 업데이트되었습니다.	2016년 6월 24일
STL_CONNECTION_LOG에 새로운 열 추가	STL_CONNECTION_LOG 시스템 테이블에 SSL 연결을 추적할 수 있는 열 2개가 새롭게 추가되었습니다. 따라서 평상시대로 감사 로그를 Amazon Redshift 테이블에 로드할 때도 sslcompression 열과 sslexpansion 열을 대상 테이블에 추가해야 합니다.	2016년 5월 5일
MD5-해시 암호	암호 및 사용자 이름을 MD5-해시 문자열로 입력하여 사용자 생성 또는 ALTER USER 명령의 암호를 지정할 수 있습니다.	2016년 4월 21일
STV_TBL_PERM에 새로운 열 추가	backup 시스템 뷰에 새롭게 추가된 STV_TBL_PERM 열에서 테이블이 클러스터 스냅샷에 포함되어있는지 알 수 있습니다. 자세한 내용은 BACKUP 단원을 참조하십시오.	2016년 4월 21일
테이블 백업 없음	스테이징 테이블 같이 중요한 데이터가 없는 테이블일 경우에는 CREATE TABLE 또는 CREATE TABLE AS 문에서 BACKUP NO를 지정하여 Amazon Redshift가 테이블을 자동 또는 수동 스냅샷에 저장하지 않도록 방지할 수 있습니다. 테이블 백업 없음을 사용하면 스냅샷을 생성하거나 스냅샷에서 복원할 때 처리 시간이 줄어들 뿐만 아니라 Amazon S3의 스토리지 공간이 절약됩니다.	2016년 4월 7일

변경 사항	설명	변경 날짜
VACUUM 삭제 임계값	이제 VACUUM 명령이 나머지 행의 95% 이상이 삭제 표시되지 않도록 스토리지 공간을 회수합니다. 결과적으로 VACUUM은 삭제된 행을 100% 회수할 때와 비교하여 삭제 단계에 필요한 시간을 크게 줄일 수 있습니다. 단일 테이블의 기본 임계값은 VACUUM 명령을 실행할 때 TO threshold PERCENT 파라미터를 추가하여 변경할 수 있습니다.	2016년 4월 7일
SVV_TRANSACTIONS 시스템 테이블	SVV_TRANSACTIONS 시스템 뷰가 현재 데이터베이스의 테이블을 잠그고 있는 트랜잭션에 대한 정보를 기록합니다.	2016년 4월 7일
IAM 역할을 사용하여 다른 AWS 리소스에 액세스	Amazon S3, DynamoDB, Amazon EMR, Amazon EC2 등의 다른 AWS 리소스와 클러스터 사이에 데이터를 이동시키려면 클러스터에 리소스에 대한 액세스 권한을 비롯해 필요한 작업 권한이 있어야 합니다. 하지만 이제는 COPY, UNLOAD, CREATE LIBRARY 명령 등으로 액세스 키 페어를 입력하는 방법에 비해 더욱 안전한 대안으로서 클러스터가 인증 및 권한 부여에 사용할 IAM 역할을 직접 지정할 수 있습니다. 자세한 내용은 역할 기반 액세스 제어 단원을 참조하십시오.	2016년 3월 29일
VACUUM 정렬 임계값	이제 VACUUM 명령이 테이블 행의 95% 이상이 이미 정렬된 테이블에 대해서는 정렬 단계를 건너뜁니다. 단일 테이블의 기본 임계값은 VACUUM 명령을 실행할 때 TO threshold PERCENT 파라미터를 추가하여 변경할 수 있습니다.	2016년 3월 17일
STL_CONNECTION_LOG에 새로운 열 추가	STL_CONNECTION_LOG 시스템 테이블에 열 3개가 새롭게 추가되었습니다. 따라서 평상시대로 감사 로그를 Amazon Redshift 테이블에 로드할 때도 sslversion, sslcipher 및 mtu 열을 대상 테이블에 새로 추가해야 합니다.	2016년 3월 17일
bzip2 압축을 지원하는 UNLOAD	이제 bzip2 압축을 사용하여 UNLOAD 명령을 실행할 수 있는 옵션이 생겼습니다.	2016년 2월 8일

변경 사항	설명	변경 날짜
ALTER TABLE APPEND	ALTER TABLE APPEND가 기존 원본 테이블에서 데이터를 이동시켜 대상 테이블에 행을 추가합니다. ALTER TABLE APPEND는 일반적으로 데이터가 복제되지 않고 이동되므로 유사한 CREATE TABLE AS 또는 INSERT INTO 작업보다 훨씬 빠릅니다.	2016년 2월 8일
WLM 쿼리 대기열 건너뛰기	워크로드 관리(WLM)가 WLM 제한 시간 등으로 인해 SELECT 문 같은 읽기 전용 쿼리를 취소하면 WLM이 해당 쿼리를 건너뛰어 다음 일치하는 대기열로 우회합니다. 자세한 내용은 WLM 쿼리 대기열 건너뛰기 섹션을 참조하세요.	2016년 1월 7일
ALTER DEFAULT PRIVILEGES	ALTER DEFAULT PRIVILEGES 명령을 사용하여 앞으로 지정된 사용자가 생성할 객체에 기본적으로 적용되는 액세스 권한 집합을 정의할 수 있습니다.	2015년 12월 10일
bzip2 파일 압축	COPY 명령이 bzip2로 압축된 파일에서 데이터를 로드할 수 있도록 지원합니다.	2015년 12월 10일
NULLS FIRST 및 NULLS LAST	ORDER BY 절에서 NULLS의 순위를 결과 집합 앞에 놓을지, 혹은 마지막에 놓을지 지정할 수 있습니다. 자세한 내용은 ORDER BY 절 및 창 함수 구문 요약 섹션을 참조하세요.	2015년 11월 19일
CREATE LIBRARY의 REGION 키워드	UDF 라이브러리 파일이 저장된 Amazon S3 버킷이 Amazon Redshift 클러스터와 동일한 AWS 리전에 속하지 않는 경우에는 REGION 옵션을 사용하여 데이터가 저장되는 리전을 지정할 수 있습니다. 자세한 내용은 CREATE LIBRARY 단원을 참조하십시오.	2015년 11월 19일
사용자 정의 스칼라 함수(UDF)	이제 사용자 정의 스칼라 함수를 생성하여 Python 2.7 Standard Library의 Amazon Redshift 지원 모듈에서, 혹은 Python 프로그래밍 언어 기반 사용자 정의 UDF에서 지원되는 SQL 외 처리 기능을 구현할 수 있습니다. 자세한 내용은 Amazon Redshift의 사용자 정의 함수 단원을 참조하십시오.	2015년 9월 11일

변경 사항	설명	변경 날짜
WLM 구성에 따른 동적 속성	WLM 구성 파라미터가 이제 일부 속성을 동적으로 적용할 수 있도록 지원합니다. 그 밖의 속성은 정적 변경을 그대로 유지하며, 따라서 구성 변경 사항을 적용하려면 연결된 클러스터를 재부팅해야 합니다. 자세한 내용은 WLM 동적 및 정적 구성 속성 및 워크로드 관리 섹션을 참조하세요.	2015년 8월 3일
LISTAGG 함수	LISTAGG 함수 및 LISTAGG 창 함수 함수는 열 값 집합을 연결하여 하나의 문자열을 생성 및 반환합니다.	2015년 7월 30일
사용 중단되는 파라미터	max_cursor_result_set_size 구성 파라미터는 이제 사용되지 않습니다. 커서 결과 집합의 크기는 클러스터의 노드 유형에 따라 제한됩니다. 자세한 내용은 커서 제약 조건 단원을 참조하십시오.	2015년 7월 24일
COPY 명령 설명서 개정	COPY 명령 참조 설명서가 더욱 자료 친화적으로, 그리고 더욱 액세스하기 쉽게 대폭적으로 바뀌었습니다.	2015년 7월 15일
Avro 형식의 COPY	COPY 명령이 Amazon S3 및 Amazon EMR의 데이터 파일에서 그리고 원격 호스트에서 SSH 연결을 사용하여 데이터를 Avro 형식으로 로드할 수 있도록 지원합니다. 자세한 내용은 AVRO 및 Avro에서 복사 예제 섹션을 참조하세요.	2015년 7월 8일
STV_STARTUP_RECOVERY_STATE	클러스터 재시작 시 STV_STARTUP_RECOVERY_STATE 시스템 테이블이 임시로 잠기는 테이블 상태를 기록합니다. Amazon Redshift는 클러스터가 다시 시작된 후 오래된 트랜잭션을 해결하기 위해 테이블이 처리되는 동안 테이블에 임시 잠금을 설정합니다.	2015년 5월 25일
순위 함수의 ORDER BY 옵션	이제 ORDER BY 절이 몇 가지 창 순위 함수에서는 옵션으로 사용됩니다. 자세한 내용은 CUME_DIST 창 함수, DENSE_RANK 창 함수, RANK 창 함수, NTILE 창 함수, PERCENT_RANK 창 함수 및 ROW_NUMBER 창 함수 섹션을 참조하세요.	2015년 5월 25일

변경 사항	설명	변경 날짜
인터리브 정렬 키	인터리브 정렬 키는 정렬 키의 각 열마다 동일한 가중치를 적용합니다. 기본적인 복합 키가 아닌 인터리브 정렬 키를 사용하면 특히 대용량 테이블에서 보조 정렬 열에 제한적 조건자를 사용하여 쿼리 성능이 크게 향상됩니다. 그 밖에도 다수의 쿼리가 동일한 테이블의 다른 열을 필터링할 때 전반적인 성능이 높아집니다. 자세한 내용은 정렬 키 및 CREATE TABLE 섹션을 참조하세요.	2015년 5월 11일
쿼리 성능 튜닝 주제 개정	쿼리 성능 튜닝의 확장으로 쿼리 성능을 분석할 수 있는 새로운 쿼리와 예가 추가되었습니다. 또한 해당 주제가 더욱 이해하기 쉽게, 그리고 완성도 높게 개정되었습니다. 성능을 높일 수 있는 쿼리 작성법에 대한 자세한 내용은 Amazon Redshift 쿼리 설계 모범 사례 섹션을 참조하세요.	2015년 3월 23일
SVL_QUERY_QUEUE_INFO	SVL_QUERY_QUEUE_INFO 뷰는 WLM 쿼리 대기열 또는 커밋 대기열에서 대기하는 쿼리의 세부 정보를 요약합니다.	2015년 2월 19일
SVV_TABLE_INFO	SVV_TABLE_INFO 뷰를 사용하여 압축 인코딩, 분산 키, 정렬 스타일, 데이터 분산 스큐, 테이블 크기, 통계 등 쿼리 성능에 영향을 미칠 수 있는 테이블 설계 문제를 진단하고 해결할 수 있습니다.	2015년 2월 19일
서버 측 파일 암호화를 지원하는 UNLOAD	이제 UNLOAD 명령이 Amazon S3 서버 측 암호화(SSE)를 자동으로 사용하여 언로드 데이터 파일을 모두 암호화합니다. 서버 측 암호화로 성능 변화는 거의 없이 데이터 보안 계층이 한층 추가되는 셈입니다.	2014년 10월 31일
CUME_DIST 창 함수	CUME_DIST 창 함수는 창 또는 파티션에 속하는 값의 누적 분포를 계산합니다.	2014년 10월 31일
MONTHS_BETWEEN 함수	MONTHS_BETWEEN 함수는 두 날짜 사이의 월 수를 계산합니다.	2014년 10월 31일

변경 사항	설명	변경 날짜
NEXT_DAY 함수	NEXT_DAY 함수는 지정한 날짜 이후 지정한 요일이 처음 도래하는 날짜를 반환합니다.	2014년 10월 31일
PERCENT_RANK 창 함수	PERCENT_RANK 창 함수는 임의의 행의 백분율 순위를 계산합니다.	2014년 10월 31일
RATIO_TO_REPORT 창 함수	RATIO_TO_REPORT 창 함수는 창 또는 파티션에서 값의 합에 대한 임의의 값 비율을 계산합니다.	2014년 10월 31일
TRANSLATE 함수	TRANSLATE 함수는 임의의 표현식에서 발견되는 모든 지정 문자를 지정한 대체 문자로 변경합니다.	2014년 10월 31일
NVL2 함수	NVL2 함수은 지정하는 표현식의 평가 결과가 NULL 또는 NOT NULL인지 여부에 따라 두 값 중 하나를 반환합니다.	2014년 10월 16일
MEDIAN 창 함수	MEDIAN 창 함수는 창 또는 파티션에서 값의 범위에 대한 중간 값을 계산합니다.	2014년 10월 16일
GRANT 및 REVOKE 명령을 위한 ON ALL TABLES IN SCHEMA schema_name 절	GRANT 및 REVOKE 명령이 업데이트되면서 ON ALL TABLES IN SCHEMA schema_name 절이 새롭게 추가되었습니다. 이로써 단일 명령으로 스키마의 모든 테이블에 대한 권한을 변경할 수 있게 되었습니다.	2014년 10월 16일
DROP SCHEMA, DROP TABLE, DROP USER, DROP VIEW 명령을 위한 IF EXISTS 절	DROP SCHEMA, DROP TABLE, DROP USER 및 DROP VIEW 명령이 업데이트되면서 IF EXISTS 절이 새롭게 추가되었습니다. 이로써 명령 실행 시 지정한 객체가 존재하지 않더라도 오류 메시지와 함께 종료되지 않고 아무런 변경 없이 메시지만 반환됩니다.	2014년 10월 16일

변경 사항	설명	변경 날짜
CREATE SCHEMA 및 CREATE TABLE 명령을 위한 IF NOT EXISTS 절	CREATE SCHEMA 및 CREATE TABLE 명령이 업데이트되면서 IF NOT EXISTS 절이 새롭게 추가되었습니다. 이로써 명령 실행 시 지정한 객체가 이미 존재하더라도 오류 메시지와 함께 종료되지 않고 아무런 변경 없이 메시지만 반환됩니다.	2014년 10월 16일
COPY 명령의 UTF-16 인코딩 지원	COPY 명령이 이제는 UTF-8 인코딩은 물론이고 UTF-16 인코딩까지 사용하는 데이터 파일에서 로드할 수 있도록 지원합니다. 자세한 내용은 ENCODING 단원을 참조하십시오.	2014년 9월 29일
새로운 워크로드 관리 자습서	자습서: 수동 워크로드 관리(WLM) 대기열 구성은 워크로드 관리(WLM) 대기열을 구성하여 쿼리 처리 및 리소스 할당을 개선할 수 있는 프로세스에 대해서 설명합니다.	2014년 9월 25일
AES 128비트 암호화	이제 COPY 명령이 Amazon S3 클라이언트 측 암호화를 사용하여 암호화된 데이터 파일에서 로드할 경우 AES 256비트 암호화는 물론이고 AES 128비트 암호화까지 지원합니다. 자세한 내용은 Amazon S3에서 암호화된 데이터 파일 로드 단원을 참조하십시오.	2014년 9월 29일
PG_LAST_UNLOAD_COUNT 함수	PG_LAST_UNLOAD_COUNT 함수는 가장 최근 UNLOAD 작업에서 처리된 행의 수를 반환합니다. 자세한 내용은 PG_LAST_UNLOAD_COUNT 단원을 참조하십시오.	2014년 9월 15일
새로운 쿼리 문제 해결 단원	쿼리 문제 해결은 Amazon Redshift 쿼리를 실행할 때 만날 가능성이 있는 가장 공통적이고 심각한 문제 몇 가지를 식별하여 해결할 수 있도록 빠른 참조를 제공합니다.	2014년 7월 7일
새로운 데이터 로딩 자습서	튜토리얼: Amazon S3에서 데이터 로드는 Amazon S3 버킷에 저장된 데이터 파일에서 Amazon Redshift 데이터베이스 테이블로 데이터를 로드하는 프로세스에 대해서 처음부터 끝까지 설명합니다.	2014년 7월 1일

변경 사항	설명	변경 날짜
PERCENTILE_CONT 창 함수	PERCENTILE_CONT 창 함수는 연속 분포 모델을 가정하는 역분포 함수로서 백분위 값과 정렬 명세를 가지며, 정렬 명세와 관련하여 지정된 백분위 값에 해당하는 보간 값을 반환합니다.	2014년 6월 30일
PERCENTILE_DISC 창 함수	PERCENTILE_DISC 창 함수는 이산 분포 모델을 가정하는 역분포 함수로서 백분위 값과 정렬 명세를 가지며, 지정된 집합에서 요소를 반환합니다.	2014년 6월 30일
GREATEST 및 LEAST 함수	GREATEST 및 LEAST 함수 함수는 표현식 목록에서 가장 크거나 가장 작은 값을 반환합니다.	2014년 6월 30일
2014년 6월 30일	COPY 명령이 Amazon Redshift 클러스터와 다른 리전에 속한 Amazon S3 버킷 또는 Amazon DynamoDB 테이블에서 데이터를 로드할 수 있도록 지원합니다. 자세한 내용은 COPY 명령 참조의 REGION 섹션을 참조하세요.	2014년 6월 30일
모범 사례 확장	Amazon Redshift 모범 사례가 확장 및 재구성을 통해 탐색 계층의 최상위까지 이동하여 원하는 모범 사례를 더욱 쉽게 찾을 수 있게 되었습니다.	2014년 5월 28일
단일 파일에 대한 UNLOAD	UNLOAD 명령을 실행할 때 PARALLEL OFF 옵션을 추가하면 테이블 데이터를 Amazon S3의 단일 파일에 직렬 방식으로 언로드할 수 있습니다. 데이터 크기가 최대 파일 크기인 6.2GB보다 크면 UNLOAD가 파일을 추가로 생성합니다.	2014년 5월 6일
REGEXP 함수	REGEXP_COUNT, REGEXP_INSTR 및 REGEXP_REPLACE 함수는 정규 표현식 패턴의 일치 여부에 따라 문자열을 조작합니다.	2014년 5월 6일
Amazon EMR에서 COPY	COPY 명령이 Amazon EMR 클러스터에서 직접 데이터를 로드할 수 있도록 지원합니다. 자세한 내용은 Amazon EMR에서 데이터 로드 단원을 참조하십시오.	2014년 4월 18일

변경 사항	설명	변경 날짜
WLM 동시성 레벨 제한의 상향 조정	이제 사용자 정의 쿼리 대기열에서 최대 50개까지 쿼리를 동시에 실행할 수 있도록 워크로드 관리(WLM)를 구성할 수 있습니다. 이번 상향 조정으로 사용자는 WLM 구성을 변경하여 시스템 성능을 관리할 수 있는 유연성이 향상되었습니다. 자세한 내용은 수동 WLM 구현 섹션을 참조하세요.	2014년 4월 18일
커서 크기를 관리하기 위한 새로운 구성 파라미터	`max_cursor_result_set_size` 구성 파라미터는 대용량 쿼리의 커서 결과 집합마다 반환될 수 있는 최대 데이터 크기(MB)를 정의합니다. 이 파라미터 값은 클러스터의 동시 커서 수에도 영향을 미치기 때문에 값을 구성하여 클러스터의 커서 수를 늘리거나 줄일 수도 있습니다. 자세한 내용은 이 가이드의 DECLARE 및 Amazon Redshift 관리 가이드의 커서 결과 집합의 최대 크기 구성 섹션을 참조하세요.	2014년 3월 28일
JSON 형식의 COPY 지원	COPY 명령이 Amazon S3의 데이터 파일에서, 그리고 원격 호스트에서 SSH 연결을 사용하여 데이터를 JSON 형식으로 로드할 수 있도록 지원합니다. 자세한 내용은 JSON 형식의 COPY 지원의 사용 시 주의 사항를 참조하십시오.	2014년 3월 25일
새로운 시스템 테이블 STL_PLAN_INFO	STL_PLAN_INFO 테이블은 쿼리 계획을 다른 방식으로 살펴볼 수 있도록 EXPLAIN 명령을 보완합니다.	2014년 3월 25일
새로운 함수 REGEXP_SUBSTR	REGEXP_SUBSTR 함수는 문자열에서 정규 표현식 패턴을 검색하여 추출되는 문자를 반환합니다.	2014년 3월 25일
STL_COMMIT_STATS에 새로운 열 추가	STL_COMMIT_STATS 테이블에 `numxids`와 `oldestxid` , 2개 열이 새롭게 추가되었습니다.	2014년 3월 6일

변경 사항	설명	변경 날짜
SSH 연결 시 COPY에서 gzip 및 lzop 지원	COPY 명령이 SSH 연결을 통해 데이터를 로드할 때 gzip 및 lzop 압축을 지원합니다.	2014년 2월 13일
새로운 함수	ROW_NUMBER 창 함수는 현재 행의 번호를 반환합니다. STRTOL 함수는 지정한 기수의 문자열 표현식을 등가의 정수 값으로 변환합니다. PG_CANCEL_BACKEND 및 PG_TERMINATE_BACKEND를 사용하면 사용자가 쿼리 및 세션 연결을 취소할 수 있습니다. LAST_DAY 함수는 Oracle 호환성을 위해 추가되었습니다.	2014년 2월 13일
새로운 시스템 테이블	STL_COMMIT_STATS 시스템 테이블은 다양한 커밋 단계의 타이밍과 커밋된 블록 수를 포함하여 커밋 성능에 대한 지표를 제공합니다.	2014년 2월 13일
단일 노드 클러스터를 통한 FETCH	단일 노드 클러스터에서 커서를 사용할 때 FETCH 명령을 사용하여 가져올 수 있는 최대 행의 수는 1,000개입니다. 단일 노드 클러스터에서는 FETCH FORWARD ALL이 지원되지 않습니다.	2014년 2월 13일
DS_DIST_ALL_INNER 재분산 전략	Explain 계획 출력 시 DS_DIST_ALL_INNER는 외부 테이블이 DISTSTYLE ALL을 사용하기 때문에 내부 테이블 전체가 단일 조각으로 재분산되었다는 것을 나타냅니다. 자세한 내용은 조인 유형 예 및 쿼리 계획 평가 섹션을 참조하세요.	2014년 1월 13일
쿼리를 위한 새로운 시스템 테이블	Amazon Redshift는 고객이 튜닝 및 문제 해결을 위해 쿼리 실행을 평가하는 데 사용할 수 있도록 새로운 시스템 테이블을 추가했습니다. 자세한 내용은 SVL_COMPILE 섹션을 참조하세요.STL_SCANSTL_RETURNSTL_SAVESTL_ALERT_EVENT_LOG	2014년 1월 13일

변경 사항	설명	변경 날짜
단일 노드 커서	이제 단일 노드 클러스터에서도 커서가 지원됩니다. 단일 노드 클러스터는 한 번에 2개의 커서를 가질 수 있으며, 결과 집합의 최대 크기는 32GB입니다. 단일 노드 클러스터에서는 ODBC Cache Size 파라미터를 1,000으로 설정하는 것이 바람직합니다. 자세한 내용은 DECLARE 단원을 참조하십시오.	2013년 12월 13일
ALL 분산 스타일	ALL 분산은 몇 가지 쿼리 유형일 때 실행 시간을 극적으로 단축할 수 있습니다. ALL 분산 스타일을 사용하는 테이블은 복사본이 모든 노드로 분산됩니다. 이 테이블은 다른 모든 테이블과 효과적으로 공동 배치되기 때문에 쿼리 실행 중 재분산이 필요하지 않습니다. ALL 분산을 사용하면 스토리지 요건과 로드 시간이 증가하기 때문에 모든 테이블에 적합하다고는 할 수 없습니다. 자세한 내용은 쿼리 최적화를 위한 데이터 배포 단원을 참조하십시오.	2013년 11월 11일
원격 호스트에서 COPY 지원	Amazon S3의 데이터 파일과 Amazon DynamoDB 테이블에서 테이블을 로드하는 것 외에도 COPY 명령은 SSH 연결을 사용하여 Amazon EMR 클러스터, Amazon EC2 인스턴스 및 기타 원격 호스트에서 텍스트 데이터를 로드할 수 있습니다. Amazon Redshift는 여러 동시 SSH 연결을 사용하여 데이터를 병렬로 읽고 로드합니다. 자세한 내용은 원격 호스트에서 데이터 로드 단원을 참조하십시오.	2013년 11월 11일
사용된 WLM 메모리 비율(%)	워크로드 관리(WLM) 구성 시 각 대기열마다 특정 메모리 비율을 지정하여 워크로드 밸런스를 유지할 수 있습니다. 자세한 내용은 수동 WLM 구현 단원을 참조하십시오.	2013년 11월 11일

변경 사항	설명	변경 날짜
2013년 11월 11일	APPROXIMATE COUNT(DISTINCT)를 사용하는 쿼리는 실행 속도가 더욱 빠를 뿐만 아니라 상대 오차는 약 2%입니다. APPROXIMATE COUNT(DISTINCT) 함수는 HyperLogLog 알고리즘을 사용합니다. 자세한 내용은 COUNT 함수을 참조하세요.	2013년 11월 11일
최근 쿼리 정보를 가져올 수 있는 새로운 SQL 함수	새로운 SQL 함수 4개가 최근 쿼리와 COPY 명령에 대한 세부 정보를 가져옵니다. 이 새로운 함수의 추가로 시스템 로그 테이블에 대한 쿼리가 더욱 쉬워졌을 뿐만 아니라 대부분 경우 시스템 테이블에 액세스하지 않고도 필요한 세부 정보를 얻을 수 있습니다. 자세한 내용은 PG_BACKEND_PID 섹션을 참조하세요.PG_LAST_COPY_IDPG_LAST_COPY_COUNT PG_LAST_QUERY_ID	2013년 11월 1일
UNLOAD를 위한 MANIFEST 옵션	UNLOAD 명령의 MANIFEST 옵션은 COPY 명령의 MANIFEST 옵션을 보완하는 역할을 합니다. UNLOAD에서 MANIFEST 옵션을 사용하면 매니페스트 파일이 자동으로 생성됩니다. 이 파일에는 Amazon S3에서 UNLOAD 작업으로 생성된 데이터 파일이 명시적으로 나열됩니다. 그런 다음 COPY 명령에서 동일한 매니페스트 파일을 사용하여 데이터를 로드할 수 있습니다. 자세한 내용은 Amazon S3로 데이터 언로드 및 UNLOAD 예 섹션을 참조하세요.	2013년 11월 1일
COPY를 위한 MANIFEST 옵션	COPY 명령에 MANIFEST 옵션을 사용하여 Amazon S3에서 로드되는 데이터 파일을 명시적으로 나열할 수 있습니다.	2013년 10월 18일

변경 사항	설명	변경 날짜
쿼리 문제 해결을 위한 시스템 테이블	쿼리 문제 해결에 사용되는 시스템 테이블에 대한 설명서가 추가되었습니다. 현재 로깅을 위한 STL 뷰 단원에는 다음과 같은 시스템 테이블에 대한 설명서가 포함되어 있습니다. STL_AGGR, STL_BCAST, STL_DIST, STL_DELETE, STL_HASH, STL_HASHJOIN, STL_INSERT, STL_LIMIT, STL_MERGE, STL_MERGEJOIN, STL_NESTLOOP, STL_PARSE, STL_PROJECT, STL_SCAN, STL_SORT, STL_UNIQUE, STL_WINDOW.	2013년 10월 3일
CONVERT_TIMEZONE 함수	CONVERT_TIMEZONE 함수는 일광 절약 시간에 따라 자동으로 조정할 수 있는 옵션을 통해 서로 다른 시간대 사이에 타임스탬프를 변환합니다.	2013년 10월 3일
SPLIT_PART 함수	SPLIT_PART 함수는 지정 구분자를 기준으로 문자열을 분할한 후 지정된 위치에 해당하는 부분을 반환합니다.	2013년 10월 3일
STL_USERLOG 시스템 테이블	STL_USERLOG는 데이터베이스 사용자가 생성, 변경 또는 삭제될 때 발생하는 변경 사항 관련 세부 정보를 기록합니다.	2013년 10월 3일
LZO 열 인코딩 및 LZOP 파일 압축	LZO 열 압축 인코딩에는 매우 높은 압축비와 뛰어난 성능이 결합되어 있습니다. Amazon S3에서 COPY는 LZOP 압축으로 압축된 파일에서 로드를 지원합니다.	2013년 9월 19일
JSON, 정규 표현식 및 커서	JSON 문자열의 구문 분석, 정규 표현식을 사용한 패턴 일치, 그리고 ODBC 연결을 통해 대용량 데이터 세트를 가져오기 위한 커서 사용 등에 대한 지원이 추가되었습니다. 자세한 내용은 JSON 함수, 패턴 일치 조건, DECLARE 섹션을 참조하세요.	2013년 9월 10일
COPY를 위한 ACCEPTINVCHAR 옵션	잘못된 UTF-8 문자가 포함된 데이터라고 해도 COPY 명령에서 ACCEPTINVCHAR 옵션을 지정하면 성공적으로 로드할 수 있습니다.	2013년 8월 29일

변경 사항	설명	변경 날짜
COPY를 위한 CSV 옵션	이제 COPY 명령이 CSV 형식의 입력 파일에서 데이터를 로드하도록 지원합니다.	2013년 8월 9일
2013년 8월 9일	CRC32 함수가 중복 검사를 주기적으로 실행합니다.	2013년 8월 9일
WLM 와일드카드	워크로드 관리(WLM)가 사용자 그룹과 쿼리 그룹을 대기열에 추가하는 데 필요한 와일드 카드를 지원합니다. 자세한 내용은 와일드카드 단원을 참조하십시오.	2013년 8월 1일
WLM 제한 시간	임의의 WLM 대기열에서 쿼리가 사용할 수 있는 시간 크기를 제한할 때는 각 대기열마다 WLM 제한 시간 값을 설정할 수 있습니다. 자세한 내용은 WLM 제한 시간 단원을 참조하십시오.	2013년 8월 1일
새로운 COPY 옵션, 'auto' 및 'epochsecs'	COPY 명령은 날짜 및 시간 형식을 자동으로 인식합니다. 여기에 더하여 새로운 시간 형식인 'epochsecs' 및 'epochmillisecs'가 지원되면서 COPY 명령이 epoch 형식으로도 데이터를 로드할 수 있습니다.	2013년 7월 25일
CONVERT_TIMEZONE 함수	CONVERT_TIMEZONE 함수는 서로 다른 시간대끼리 타임스탬프를 변환합니다.	2013년 7월 25일
FUNC_SHA1 함수	FUNC_SHA1 함수는 SHA1 알고리즘을 사용하여 문자열을 변환합니다.	2013년 7월 15일
2013년 7월 15일	쿼리가 사용할 수 있는 시간 크기를 제한할 때는 WLM을 구성하면서 max_execution_time 파라미터를 설정할 수 있습니다. 자세한 내용은 WLM 구성 수정 단원을 참조하십시오.	2013년 7월 22일
4바이트 UTF-8 문자	이제 VARCHAR 데이터 형식이 4바이트 UTF-8 문자를 지원합니다. 5바이트 이상의 UTF-8 문자는 지원되지 않습니다. 자세한 내용은 스토리지 및 범위 단원을 참조하십시오.	2013년 7월 18일
SVL_QERROR	SVL_QERROR 시스템 뷰는 더 이상 사용되지 않습니다.	2013년 7월 12일

변경 사항	설명	변경 날짜
문서 이력 수정	이제 문서 이력 페이지에 설명서의 업데이트 날짜가 표시됩니다.	2013년 7월 12일
STL_UNLOAD_LOG	STL_UNLOAD_LOG가 언로드 작업의 세부 정보를 기록합니다.	2013년 7월 5일
JDBC fetch size 파라미터	JDBC를 사용하여 대용량 데이터 세트를 가져올 때 클라이언트 측 메모리 부족 오류를 방지하려면 클라이언트에서 JDBC fetch size 파라미터를 설정하여 데이터를 일괄적으로 가져올 수 있습니다. 자세한 내용은 JDBC Fetch Size 파라미터 설정 단원을 참조하십시오.	2013년 6월 27일
UNLOAD 암호화 파일	이제 UNLOAD가 테이블 데이터를 Amazon S3의 암호화 파일로 언로드하도록 지원합니다.	2013년 5월 22일
임시 자격 증명	이제 COPY 및 UNLOAD가 임시 자격 증명을 사용하도록 지원합니다.	2013년 11월 4일
명료화 추가	테이블 설계 및 데이터 로딩에 대한 설명이 이해하기 쉽게 확장되었습니다.	2013년 14월 2일
모범 사례 추가	Amazon Redshift 테이블 설계 모범 사례 및 데이터 로드에 대한 Amazon Redshift 모범 사례가 추가되었습니다.	2013년 14월 2일
암호 제약 조건의 명료화	CREATE USER 및 ALTER USER에 필요한 암호 제약 조건이 이해하기 쉽게 명료화되었으며, 그 밖에 여러 가지 사소한 부분이 수정되었습니다.	2013년 14월 2일
새 안내서	이 설명서는 Amazon Redshift 개발자 안내서의 최초 릴리스입니다.	2013년 14월 2일