Mixed Covers of Keys and Functional Dependencies for Maintaining the Integrity of Data under Updates

Zhuoxing Zhang
The University of Auckland
Auckland, New Zealand
zzha969@aucklanduni.ac.nz

Sebastian Link
The University of Auckland
Auckland, New Zealand
s.link@auckland.ac.nz

ABSTRACT

Covers for a set of functional dependencies (FDs) are fundamental for many areas of data management, such as integrity maintenance, query optimization, database design, and data cleaning. When declaring integrity constraints, keys enjoy native support in database systems while FDs need to be enforced by triggers or at application level. Consequently, maximizing the use of keys will provide the best support. We propose the new notion of mixed cover for a set of FDs, comprising the set of minimal keys together with a cover for the set of non-key FDs implied by the FD set. We establish sequential and parallel algorithms for computing mixed covers from a given set of FDs, and illustrate that they complement each other in terms of their performance. Even though FD covers are typically smaller in number or size than their corresponding mixed cover, the latter generate orders of magnitude lower overheads during integrity maintenance. We also quantify how mixed covers improve the performance of query, refresh and insert operations on the TPC-H benchmark under different constraint workloads.

PVLDB Reference Format:
Zhuoxing Zhang and Sebastian Link. Mixed Covers of Keys and Functional Dependencies for Maintaining the Integrity of Data under Updates. PVLDB, 17(7): 1578 - 1590, 2024.
doi:10.14778/3654621.3654626

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ZhuoxingZhang/mixed-covers.

1 INTRODUCTION

Functional dependencies (FDs) can express many requirements of applications, and RDBMSs provide means for maintaining the integrity of data by enforcing such FDs. When an FD $X \rightarrow Y$ is declared on a relation schema R, all records with values matching on all the attributes in X must have values matching on all the attributes in Y. Hence, updates on a given relation prompt the RDBMS to validate a given set F of FDs. Intuitively, both the number and size of FDs in F have a direct impact on the overhead required for integrity maintenance. Here, $|F|$ will denote the number of FDs in F, and $||F||$ the total number of attribute occurrences in F. David Maier investigated two notions of covers for a set of FDs [23]. A *cover* for F is a set G of FDs that implies the same FDs as F. G is *minimal* if there is no other cover for F with fewer FDs than G, and G is *optimal* if there is no other cover for F with fewer total attribute occurrences than G. While minimal covers can be found in polynomial time, the problem of deciding whether there is an optimal cover of a given size is *NP*-complete [23].

Covers of FDs naturally facilitate fundamental tasks in data management. Intuitively, the covers constitute smaller representations of a given set of FDs. Hence, the same benefits are realized with less effort. Firstly, smaller FD covers cause less overheads for integrity maintenance. In particular, FDs or attributes that are redundant in an FD set cause overheads during integrity maintenance that are redundant, too. This is true for normalized, narrow tables typical for transactional workloads, but also for non-normalized, wider tables typical for use in analytical tasks. In fact, modern data architectures such as active data warehouses and cloud databases are expected to process updates frequently to enable real-time decision making based on current data. Secondly, smaller FD covers offer opportunities for further query optimization [9, 10, 17, 21]. For example, query optimizers may identify redundant DISTINCT clauses or GROUP BY attributes by validating whether a key or FD is implied by the underlying FD set [17], and checking implication is faster when smaller FD covers are used. Thirdly, smaller covers result in outputs of normalization algorithms with fewer relation schemata or fewer attributes during logical schema design [6, 20, 25, 39], which facilitates more efficient data management in the lifetime of databases [24]. As a final example, smaller covers for the set of FDs that are mined from database instances are easier to comprehend for humans who need to identify FDs that are meaningful rather than those that hold only accidentally [1, 36]. Intriguingly, FDs or attributes that are mined redundantly slow down the progress of FD discovery unnecessarily [14, 26–28, 32]. In addition, even if FDs only hold accidentally on the relation they were mined from, they may still benefit query optimization [17]. These example areas of impact greatly motivate the study of covers.

A natural question arising is how we would use any set of FDs for maintaining the integrity of data? Firstly, the most important special case of FDs are keys. Given a relation schema R, an attribute subset X is called a *key* whenever the FD $X \rightarrow R$ holds. Indeed, no relation satisfying $X \rightarrow R$ can ever have different records with matching values on all the attributes in X. A main difference between keys and non-key FDs is that the former enjoy native support in RDBMSs while the latter need to be enforced by triggers or at application level. Hence, for implementation purposes only, it makes sense to rely on keys as much as possible. Secondly, every key declared on a relation schema (primary key or UNIQUE constraint) will result in a UNIQUE index, which means that integrity enforcement of keys

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 17, No. 7 ISSN 2150-8097.
doi:10.14778/3654621.3654626

becomes efficient. Hence, for performance purposes, it makes sense to rely on keys as much as possible. Thirdly, database normalization aims at the transformation of any dependencies into keys [11]. In the context of FDs only, state-of-the-art computes dependency-preserving decompositions in Third Normal Form (3NF), which are in Boyce-Codd Normal Form (BCNF) whenever possible [6, 25]. Given we can only guarantee a dependency-preserving decomposition into 3NF, not all FDs can be expressed by keys. In summary, it is therefore more natural and practical to investigate mixed notions of covers for a set F of FDs, comprising the set of minimal keys implied by F and a cover for the set of non-key FDs in F (FDs in F not implied by the set of minimal keys). Mixed covers may even be of smaller size than their FD cover, as illustrated next.

Example 1.1. Let the relation schema Traffic consist of the following attributes CAR-SERIAL#, LICENSE#, OWNER, DATE, TIME, TICKET#, and OFFENSE. The following is a minimal cover of size 14 for the FDs associated with the schema:

- CAR-SERIAL# $\longrightarrow$ LICENSE#
- LICENSE# $\longrightarrow$ CAR-SERIAL#, OWNER
- TICKET# $\longrightarrow$ CAR-SERIAL#, OWNER, DATE, TIME
- CAR-SERIAL#, DATE, TIME $\longrightarrow$ TICKET#, OFFENSE .

The following is an optimal cover of size 13:

- CAR-SERIAL# $\longrightarrow$ LICENSE#
- LICENSE# $\longrightarrow$ CAR-SERIAL#, OWNER
- TICKET# $\longrightarrow$ CAR-SERIAL#, DATE, TIME
- CAR-SERIAL#, DATE, TIME $\longrightarrow$ TICKET#, OFFENSE .

However, the following optimal mixed cover has size 12:

- three keys {TICKET#}, {CAR-SERIAL#, DATE, TIME}, {LICENSE#, DATE, TIME}
- CAR-SERIAL# $\longrightarrow$ LICENSE#
- LICENSE# $\longrightarrow$ CAR-SERIAL#, OWNER. □

Even though they are typically larger than their FD covers, mixed covers are better for integrity maintenance since keys use UNIQUE indexes, as illustrated by the next famous 3NF example [4].

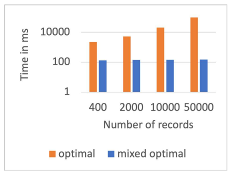
Example 1.2. Consider the 3NF schema Mail with attributes *A(DDRESS)*, *C(ITY)*, and *Z(IP)*, and FDs: $AC \rightarrow Z$ and $Z \rightarrow C$, which is its own optimal FD cover of size 5. The FD set implies two minimal keys $\{A, C\}$ and $\{A, Z\}$, plus the FD $Z \rightarrow C$, so the size of an optimal mixed cover is 6. Table 1a shows a synthetic relation over Mail, which satisfies the optimal covers, and violates all FDs not implied by the covers. Hence, the relation is a perfect sample for the FD set given. To compare how FD and mixed covers handle integrity maintenance, we have created relations of increasing sizes by taking disjoint unions of copies of the sample. Figure 1b and Figure 1c compare the times (in ms) for performing inserts for 10% (respectively, 40%) of records from the given relations, based on optimal FD and mixed covers (note the logarithmic scales). Not only does the mixed cover perform 99% faster than the FD cover, but it also shows robust scalability. This superior performance is natural due to UNIQUE indices. On perfect samples the improvement is also evident for Example 1.1, as illustrated in Figures 2a and 2b. □

The contributions of our paper are summarized as follows:

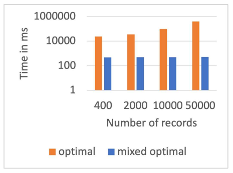
(1) For classical types of FD covers, such as non-redundant, reduced, canonical, minimal, minimal-reduced, and optimal

ADDRESS	CITY	ZIP
0	0	0
1	0	1
0	1	2
2	0	0

(a) Perfect Sample



(b) Insert 10% records



(c) Insert 40% records

Figure 1: Update time optimal FD vs mixed cover on Mail

cover [24], we analyze their computation on real and synthetic data, their reduction in numbers and size of representing FDs, and how much they reduce overheads during integrity management.

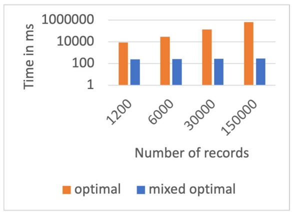
(2) For each notion of FD covers, we introduce a mixed variant, comprising the set of minimal keys implied and a corresponding cover for the set of FDs not implied by the minimal keys.

(3) Previous work has not properly justified that different notions of FD covers are actually different. In particular, for all schemata in BCNF, all notions of FD covers and mixed variants collapse into the set of minimal keys. However, we will show that the relationships between FD covers known from previous work [24] already apply to schemata on 3NF. Since this is the only case in practice where frequent integrity maintenance should happen, our result does justify the different notions of FD covers. We also show that the same relationships transcend to mixed variants.

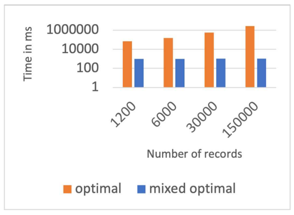
(4) We propose sequential and parallel algorithms for computing mixed covers of a target type for an FD set. Experiments illustrate that they complement one another, as the sequential algorithm performs well when the parallel algorithm does not and vice versa.

(5) We extend our experiments from FD covers to mixed variants. Firstly, we analyze the performance of our sequential and parallel algorithms in computing mixed covers, illustrating when one outperforms the other and therefore benefiting from both algorithms. We demonstrate on a variety of schemata and data how mixed covers provide high performance support for integrity maintenance, for both non-normalized schemata typical for analytical tasks and normalized schemata used for transactional tasks. Finally, we show that using mixed over FD covers improves query performance significantly and is essential for refresh and insert operations of the TPC-H benchmark across different constraint workloads.

In summary, mixed covers transform the classical notion of FD covers from database theory into best practice for minimizing overheads during update maintenance over both normalized and non-normalized schemata. We highlight the impact on application areas of FD covers from before. Firstly, utilizing the set K of minimal keys saves orders of magnitude during integrity maintenance, as we will show here. This actually clarifies that even larger-sized covers (namely mixed covers) can perform much better than FD covers of smaller size. Secondly, knowing K enables us to specify keys and utilize their UNIQUE indices for data management, including query optimization. This applies even when minimal keys or FDs are used that only hold accidentally [10, 17]. Thirdly, the benefit of utilizing minimal keys during logical database design has recently been illustrated by parameterizing BCNF by the number of minimal keys [39]. For data profiling [1], our ideas suggest separating FD mining into the discovery of minimal keys and non-key FDs,



(a) Insert 10% new records



(b) Insert 40% new records

Figure 2: Update time optimal FD vs mixed cover on TRAFFIC

respectively. This calls for a dedicated line of future work which is beyond the scope of this paper. In fact, our focus is the impact on integrity maintenance.

Organization. We repeat fundamental concepts about FDs in Section 2. We introduce the notion of mixed covers in Section 3, where we also establish that relationships, known to hold between different types of FD covers, also hold between the corresponding types of their mixed variants, and that these relationships already hold on 3NF schemata. Our sequential and parallel algorithms for computing mixed covers of a target type t are proposed and analyzed in Section 4. Section 5 is dedicated to experiments, before concluding and commenting on future work in Section 6. The full paper, including proofs, can be found at the URL of our Artifact.

2 FUNCTIONAL DEPENDENCIES

We will provide the necessary background on functional dependencies, including their definition within the relational model of data, solutions to their implication problem, and their notion of a cover.

Relation Schemata and Relations. We may use attribute symbols, such as A, B, C, for column names of a table. For each attribute A, $dom(A)$ denotes the set of values that may occur in column A of any table. A *relation schema* is a finite set R of attributes. Intuitively, R comprises all properties that every entity over R needs to be described by. A *tuple* or *record* over R is a function $t : R \to \bigcup_{A\in R} dom(A)$ that maps every attribute of R to a value $t(A) \in dom(A)$ from its domain. A *relation* over R is a finite set of tuples over R. Example 1.2 features relation schema MAIL and Table 1a shows a relation over MAIL with synthetic values.

SQL uniformly uses the null marker `null` to handle all interpretations of incomplete values. For convenience we include `null` in every domain, so `null` $\in dom(A)$. Every occurrence of `null` means no information is currently available [3, 15, 38].

Syntax and Semantics. A *functional dependency* (FD) over relation schema R is an expression $X \to Y$ with attribute subsets $X, Y \subseteq R$. A relation r over R is said to *satisfy* the FD $X \to Y$ over R whenever every pair of tuples $t, t' \in r$ with matching values on all the attributes in X has also matching values on all the attributes in Y, that is, if $t(X) = t'(X)$ implies $t(Y) = t'(Y)$. An FD $X \to Y$ is *trivial* if $Y \subseteq X$. As example, while the relation in Table 1a satisfies the FD $AC \to Z$, it does not satisfy the FD $CZ \to A$. We interpret every occurrence of `null` by either `null <> null` or `null = null`. As results do not differ much, we only report the former interpretation. Comprehensive treatments of nulls, see [5, 35] for examples, are not the subject of this study but interesting future work.

A *key* over relation schema R is an expression $X \subseteq R$. A relation r over R is said to *satisfy* the key X over R whenever there are no two different tuples in r with matching values on all the attributes in X, that is, if $t(X) = t'(X)$ holds for any $t, t' \in r$, then $t = t'$. It is easy to see that a relation over R satisfies the key X over R if and only if the relation satisfies the FD $X \to R$ over R. The relation in Table 1a satisfies the keys $\{A, C\}$ and $\{A, Z\}$, but not the key $\{C, Z\}$.

FDs form an important class of integrity constraints, which restrict the set of relations to those considered meaningful for the underlying application. Whenever the set F holds the integrity constraints for an application domain, then every relation ought to satisfy all constraints in F, particularly after the relation is updated. That is, we need to maintain the integrity of data under updates.

Implication Problem. Constraints interact with one another. For a set $F \cup \{X \to Y\}$ of FDs we say that F *implies* $X \to Y$, denoted by $F \models X \to Y$, whenever every relation that satisfies every FD in F also satisfies $X \to Y$. This notion is fundamental for integrity management: If a relation satisfies all FDs in F, then we do not need to check whether the relation also satisfies any other FD implied by F. However, if F does not imply $X \to Y$, then there is some relation that satisfies all FDs in F but does not satisfy $X \to Y$. Hence, if $X \to Y$ is also a meaningful FD, we still need to check if a relation satisfies $X \to Y$ even after we know it satisfies all FDs in F.

Due to this importance, the *implication problem* has been investigated deeply. For a class of constraints, the problem is to decide whether for a relation schema R, and a set $\Sigma \cup \{\varphi\}$ of constraints over R from that class, $\Sigma \models \varphi$. For a given FD set F, we denote by $F^+ = \{X \to Y \mid F \models X \to Y\}$ the set of FDs implied by F. Similarly, for an attribute subset $X \subseteq R$ and an FD set F over R, we denote by $X_F^+ = \{A \in R \mid F \models X \to A\}$ the *attribute set closure* of X given F, that is, the set of attributes functionally determined by X given F. The implication problem for FDs is *PTIME*-complete, and can be solved efficiently by using the result that $X \to Y \in F^+$ if and only if $Y \subseteq X_F^+$. In words, F implies $X \to Y$ if and only if every attribute in Y is contained in the attribute set closure of X given F. Since computing the attribute set closure can be implemented in time linear in the input (X, F), the implication problem for FDs can be decided in time linear in the input $(F, X \to Y)$ time, too [19].

For example, given the set F of FDs from Example 1.1, we can deduce that {CAR-SERIAL#, DATE} is not a key implied by F. Indeed, $\{\text{CAR-SERIAL\#, DATE}\}_F^+$ consists of CAR-SERIAL#, DATE, LICENSE#, OWNER, which does not include TICKET#, TIME, OFFENSE. Hence, TRAFFIC is not a subset of $\{\text{CAR-SERIAL\#, DATE}\}_F^+$, and thus {CAR-SERIAL#, DATE} $\to$ TRAFFIC is not implied by F.

Given an FD set F, a key X implied by F is *minimal* if and only if every proper subset $Y \subseteq X$ is not a key implied by F. For example, the key {CAR-SERIAL#, DATE, TIME} from Example 1.1 is minimal.

Covers. Efforts have been made to represent sets of FDs succinctly. For example, any FD implied by $F_0 = \{Emp \to Dep, Dep \to Mgr, Emp \to Mgr, \{Emp, Dep\} \to Mgr, Emp \to \{Dep, Mgr\}\}$ is also implied by the set $G_0 = \{Emp \to Dep, Dep \to Mgr\}$.

Representation of FD sets are known as covers. Formally, two FD sets F and G over relation schema R are *equivalent*, written $F \equiv G$, if $F^+ = G^+$, that is, when they both imply the same set of FDs. If $F \equiv G$, then we say that F is a *cover* of G. For example, the FD sets F_0 and G_0 above are covers of one another. Indeed, $F_0 \equiv G_0$ since $G_0 \subseteq F_0$ and every FD in F_0 is implied by G_0.

3 MIXED COVERS OF KEYS AND FDS

We will introduce the main concept of our work, mixed covers, before discussing mixed variants for various types of FD covers, establishing important relationships between them, and showing that these already hold on schemata in Third Normal Form (3NF).

Previous work has brought forward different types of FD covers, such as non-redundant, reduced, canonical, minimal, minimal-reduced, and optimal. Before discussing these notions below, we will now introduce the notion of a mixed cover for every type t.

Definition 3.1. For a set F of FDs, the set (K, G) is a *mixed cover* of type t if and only if K denotes the set of minimal keys implied by F, and G is an FD cover of type t for the set F' of FDs in F not implied by K, that is, $G \equiv F' = \{f \in F \mid K \not\models f\}$. □

Integrity maintenance in practice naturally leads to our definition of mixed covers. Indeed, when updates occur frequently, integrity should be enforced on normalized database schemata. State-of-the-art methods can normalize any given schema into a dependency-preserving 3NF decomposition, which will be in BCNF whenever possible [25]. If a schema is in dependency-preserving BCNF, all FDs can be enforced by minimal keys. Otherwise, integrity can be maintained by the set of minimal keys and some non-key FDs whose attributes on the right side are prime (that is, belong to some minimal key), which is the definition of 3NF [7, 8]. For that reason, it is somewhat surprising that mixed notions of covers have not been studied before. Previous work has not even investigated FD covers over normalized schemata.

Let us consider relation schemata R that are in BCNF for a given FD set F. That is, for every non-trivial FD $X \rightarrow Y \in F$ it holds that $X \rightarrow R \in F^+$. Consequently, the set of minimal keys implied by F is sufficient and necessary to maintain the integrity on a schema R that is in BCNF for F. Interestingly, all notions of an FD cover collapse into the set of minimal keys.Hence, for schemata in BCNF, there is only one choice for a cover, namely the set of minimal keys. In particular, R is in BCNF for F iff the set G in a mixed cover (K, G) for F is empty.

Given the situation for BCNF, it is natural to ask what notions of covers need to be considered when schemata are in 3NF. Here, for every non-trivial FD $X \rightarrow Y \in F$ it holds that $X \rightarrow R \in F^+$ or every attribute in Y must be prime (that is, belong to some minimal key for F). Note that a dependency-preserving decomposition into 3NF can always be achieved, where dependency-preservation means that the original set F of FDs is implied by the union of FD sets that hold on elements of the decomposition. Hence, it is sensible to investigate different notions of FD covers on schemata in 3NF.

We will now discuss types of FD covers, their mixed variants, and relationships between these types. Algorithms for computing types of FD covers can be found in the literature [24]. We will demonstrate that 1) all known relationships between FD covers already apply to schemata in 3NF, and are therefore all relevant, and 2) the same is true for our notions of mixed covers. In particular, 1) offers the first actual justification for using the different notions of FD covers for maintaining integrity, and 2) extends this to the actual use of mixed covers in practice, by maximizing the use of minimal keys and their UNIQUE indexes. The strict relationships are summarized in Figure 3. A direct edge with no line on it means that every cover with the type of its origin is also a cover with the type of its destination, while the reversed edge has a line through it, indicating that the opposite direction does not hold. For instance, every optimal cover is reduced-minimal, but there are reduced-minimal covers that are not minimal.

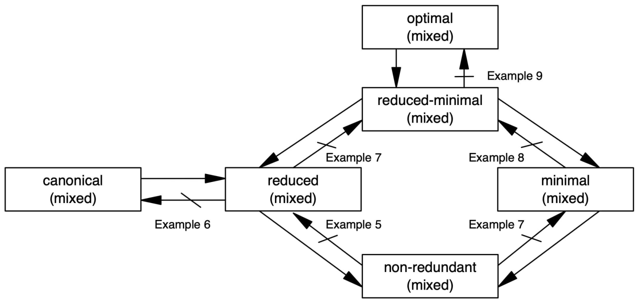


Figure 3: Strict Relationships between Types of FD (Mixed) Covers that already hold on Schemata in 3NF

Non-redundant covers eliminate FDs that are implied by others. An FD set F is *non-redundant* if there is no proper subset F' of F where $F' \equiv F$. If such an F' exists, F is *redundant*. G is a *non-redundant cover* for FD set F, if $G \equiv F$ and G is non-redundant.

An FD $X \rightarrow Y \in F$ is called *redundant in* F if $F - \{X \rightarrow Y\} \models X \rightarrow Y$. Hence, we can compute a non-redundant cover G of F by starting with $G := F$ and removing an FD from F whenever it is redundant in F. This can be done in time $O(||F||^2)$.

The first example shows that redundant (mixed) covers for schemata in 3NF exist.

Example 3.2. Consider schema $R = ABCD$ with FD set $F = \{ABC \rightarrow D, CD \rightarrow B, D \rightarrow B\}$. The set K of minimal keys is ABC and ACD, so every attribute is prime, and (R, F) is in 3NF. The FD set $G = \{ABC \rightarrow D, D \rightarrow B\}$ is a non-redundant cover of F. Similarly, for the set $G' = \{D \rightarrow B\}$, (K, G') is a mixed non-redundant cover for the mixed cover (K, F') where $F' = \{CD \rightarrow B, D \rightarrow B\}$.

Reduced covers. Since non-redundant covers of F carry no redundant FDs, removal of any further FDs would result in an FD set not equivalent to F. However, one may still reduce the size of a non-redundant cover for F by removing attributes from FDs in F.

Intuitively, we can remove *extraneous* attributes from the left or right side of an FD in F whenever this does not affect the closure of F. More precisely, let F denote an FD set over relation schema R, and $X \rightarrow Y$ an FD in F. An attribute A in R is *extraneous in* $X \rightarrow Y$ with respect to F if

(1) $X = AZ, X \neq Z$, and $(F - \{X \rightarrow Y\}) \cup \{Z \rightarrow Y\} \equiv F$, or
(2) $Y = AW, Y \neq W$, and $(F - \{X \rightarrow Y\}) \cup \{X \rightarrow W\} \equiv F$.

The FD $X \rightarrow Y \in F$ is *left-reduced* if X contains no attribute A extraneous in $X \rightarrow Y$, $X \rightarrow Y$ is *right-reduced* if Y contains no attribute A extraneous in $X \rightarrow Y$, and $X \rightarrow Y$ is *reduced* if it is left-reduced, right-reduced and $Y \neq \emptyset$. Similarly, an FD set F is left-reduced (right-reduced, reduced) if every FD in F is left-reduced (respectively, right-reduced, reduced).

Computing a reduced cover follows step (1) which computes a left-reduced cover, step (2) that returns a right-reduced cover for the output of step (1), and step (3) which eliminates any FDs $X \rightarrow \emptyset$ from the output of step (2). Steps (1) and (2) must not be switched, and step (3) is necessary. Every reduced cover is non-redundant as every attribute on the right side of any redundant FD is extraneous. Hence, a reduced cover of F is found in time $O(||F||^2)$.

The next example shows there are (mixed) covers that are non-redundant but not reduced for schemata in 3NF.

Example 3.3. Consider schema $R = ABCD$ with FD set $F = \{ABC \to D, CD \to B, D \to C\}$. The set K of minimal keys is ABC and AD, so every attribute is prime, so (R, F) is in 3NF. F is non-redundant, but not reduced. FD set $G = \{ABC \to D, D \to B, D \to C\}$ is a reduced cover of F. For $G' = \{D \to B, D \to C\}$, (K, G') is a mixed reduced cover for the mixed non-redundant cover (K, F') where $F' = \{CD \to B, D \to C\}$ is not reduced.

Canonical covers. An FD set F is *canonical* if every FD in F is of the form $X \to A$ and F is left-reduced and non-redundant. Canonical covers are reduced since every FD must be right-reduced due to having only single attributes on the right. We can compute a canonical cover of input F in time $O(||F||^2)$ by computing a reduced cover and splitting every FD with multiple attributes on the right side into separate FDs with the same left side and a single attribute on the right side. We compute reduced from canonical covers by combining FDs with equal left sides into a single FD.

The next example shows that there are (mixed) covers that are reduced but not canonical for schemata in 3NF.

Example 3.4. Consider schema $R = ABCD$ with FD set $F = \{ABC \to D, D \to BC\}$. As $K = \{ABC, AD\}$ every attribute is prime and (R, F) is in 3NF. F is reduced, but not canonical. FD set $G = \{ABC \to D, D \to B, D \to C\}$ is a canonical cover of F. For $G' = \{D \to B, D \to C\}$, (K, G') is a mixed canonical cover for mixed reduced cover (K, F') but $F' = \{D \to BC\}$ is not canonical.

Minimal covers. Reduced covers of an FD set F do not necessarily have as few FDs as any cover for F. Hence, Maier defined an FD set G as *minimal* if G has as few FDs as any equivalent set of FDs [23].

Two attribute sets X and Y are *equivalent* under an FD set F, written $X \leftrightarrow Y$, if $X_F^+ = Y_F^+$. Let $E_F(X)$ be the set of FDs in F with left sides equivalent to X. Note that $E_F(X)$ is empty when no left side of any FD in F is equivalent to X. Then the set $\overline{E}_F = \{E_F(X) \mid X \subseteq R \text{ and } E_F(X) \neq \emptyset\}$ is always a partition of F.

We can compute a minimal cover for input F in time $O(||F|| \cdot |F|)$ as follows: We start with a non-redundant cover G of input F, and then check if there is any $E_G(X)$ for which $Y \to U, Z \to V \in E_G(X)$ exist such that $G - E_G(X) \models Y \to Z$. In that case, the two FDs $Y \to U$ and $Z \to V$ can be replaced in G by the single FD $Z \to UV$, resulting in fewer FDs.

The next example shows there are (mixed) covers that are reduced but not minimal for schemata in 3NF. As every reduced cover is also non-redundant, the example shows there are (mixed) covers that are non-redundant but not minimal for schema in 3NF.

Example 3.5. Consider schema $R = ABCD$ with FD set $F = \{ABC \to D, D \to B, D \to C\}$. K contains ABC and AD, so every attribute is prime and (R, F) is in 3NF. F is reduced and non-redundant, but not minimal. FD set $G = \{ABC \to D, D \to BC\}$ is a minimal cover of F. For $G' = \{D \to BC\}$, (K, G') is a mixed minimal cover for the mixed reduced and non-redundant cover (K, F') where $F' = \{D \to B, D \to C\}$ is not minimal.

The next example shows that there are (mixed) covers that are minimal but not reduced for schemata in 3NF.

Example 3.6. Let $R = ABCDE$ with $F = \{ABC \to DE, CDE \to AC, D \to C\}$. K contains ABC, ABD, and BDE, so every attribute is prime and (R, F) is in 3NF. F is minimal, but neither left- nor right-reduced. $G = \{ABC \to DE, DE \to A, D \to C\}$ is a minimal-reduced cover of F. For $G' = \{DE \to A, D \to C\}$, (K, G') is a mixed minimal-reduced cover for the mixed minimal cover (K, F') where $F' = \{CDE \to BC, D \to C\}$ is neither left- nor right-reduced.

Minimal-reduced covers. Minimal covers may feature extraneous attributes. We can compute a minimal-reduced cover for input FD set F in time $O(||F||^2)$ by computing a reduced cover for a minimal cover of F.

Optimal covers. An FD set F is *optimal* if there is no equivalent set of FDs with fewer attribute symbols than F. Maier showed that it is *NP*-complete to decide if a given FD set is optimal [23]. It is thus unlikely a polynomial-time algorithm in the input exists.

Optimal covers can be computed as minimal covers of so-called mini covers [30]. An FD set F is *mini* if the right side of every FD in F is a single attribute, F has the fewest FDs and, within that constraint, the fewest attributes [30]. A mini cover can be computed by (1) transforming the input into a Boolean formula, (2) using a standard method to minimize the formula, and (3) transforming the minimized formula back into an FD set which will be mini.

The first Delobel-Casey transform [30] of an FD $\{A1, \dots A_m\} \to \{B_1, \dots, B_r\}$ is the Boolean expression $(A_1 \wedge \dots \wedge A_m \wedge \neg B_1) \vee \dots \vee (A_1 \wedge \dots \wedge A_m \wedge \neg B_r)$. For an FD set F, the first Delobel-Casey transform of F is a Boolean expression which is the conjunction of the first Delobel-Casey transform for each member.

The transformations between FD sets and their Boolean formulae preserve equivalence, which means the steps (1)-(3) above will indeed result in a mini cover. Optimal covers can be computed as minimal covers of mini covers for input FD set F, and run in time worst-case exponential in $||F||$ [30].

Correctness and time complexity result from any standard technique for minimizing Boolean formulae [16], and the insight above.

The final real-world example shows there are (mixed) covers that are minimal-reduced but not optimal for schemata in 3NF.

Example 3.7. Suppose users are assigned different IDs for each server, so R consists of *Fi(rstname), L(astname), I(D), E(mail), S(erver).* $F = \{FiL \to E, E \to FiL, ES \to I, I \to FiL\}$ is minimal-reduced, and (R, F) is in 3NF since K contains $FiLS$, ES, and IS. FD set $G = \{FiL \to E, E \to FiL, ES \to I, I \to E\}$ is equivalent to F and has fewer attributes. Indeed, G is an optimal cover of F. For $G' = \{FiL \to E, E \to FiL, I \to E\}$, (K, G') is a mixed optimal cover for the mixed minimal-reduced cover (K, F') where $F' = \{FiL \to E, E \to FiL, I \to E, I \to FiL\}$ is minimal-reduced but not optimal.

4 COMPUTATION OF MIXED COVERS

We will now devise algorithms for computing mixed covers, one sequential and one parallel. We will also analyze their worst-case computational complexity, and that of closely related problems.

Algorithm 1 Mixed_Seq(F)

```
Require: Set $F$ of FDs, Type $t$ of cover
Ensure: A mixed cover $(K, G)$ for $F$ of type $t$
1: $G' \leftarrow$ cover for $F$ of type $t$ using FD cover algorithm
2: $K \leftarrow$ the set of minimal keys implied by $G'$ [22]
3: $G \leftarrow \{Y \to Z \in G' \mid \forall X \in K(X \not\subseteq Y)\}$
4: return $(K, G)$
```

Algorithm 2 Mixed_Parallel(F)

```
Require: Set $F$ of FDs, Type $t$ of cover
Ensure: A mixed cover $(K, G)$ for $F$ of type $t$
1: Compute $G'$ and $K$ in parallel:
   $G' \leftarrow$ cover of $F$ of type $t$ using FD cover algorithm
   $K \leftarrow$ the set of minimal keys implied by $F$ [22]
2: $G \leftarrow \{Y \to Z \in G' \mid \forall X \in K(X \not\subseteq Y)\}$
3: return $(K, G)$
```

4.1 Sequential Algorithm

We want to compute a mixed cover (K, G) of a given type t for a given set F of FDs. We do *not* assume input F is of target type t.

Algorithm 1 is sequential and works as follows. Firstly, we compute a type-t FD cover G' of input F. Then we compute the set K of minimal keys from G'. Finally, we obtain G by keeping those FDs of G' that are not keys. Since G' is a cover of F, K is indeed the set of minimal keys for F. The next result shows why the last step produces a type-t cover G for the set of non-key FDs for F.

Theorem 4.1. *Let G' be an FD cover of type t, K the set of minimal keys implied by G', and $G = \{\sigma \in G' \mid K \not\models \sigma\}$ the set of FDs from G' not implied by K. Then (K, G) is a mixed cover for G' of type t.* □

Correctness of Algorithm 1 follows by showing that FDs not implied by K are those whose LHS do not contain any key from K.

Lemma 4.2. *Let $Y \to Z$ denote a non-trivial FD over relation schema R, and let K denote a set of minimal keys for R. Then $K \models Y \to Z$ if and only if there is some $X \in K$ such that $X \subseteq Y$.* □

4.2 Parallel Algorithm

Algorithm 1 computes the set K of minimal keys after computing G', which is a type-t FD cover of F. It makes sense to use G' since it is typically smaller than F. However, we can also compute K from the original FD set F. In this case, we do not need to wait until G' has been computed. That is, we could compute G' and K in parallel. This results in Algorithm 2.

Note that Algorithms 1 and 2 are guaranteed to return the same result since the set K' of minimal keys implied by G' and the set K of minimal keys implied by G coincide. Indeed, G' and G are covers of one another, so they determine the same set of minimal keys.

Illustrated by Figure 4, Algorithm 2 runs faster than Algorithm 1 if and only if the computation of K from F is faster than the total of first computing G' from F and computing K from G' subsequently. As we will witness in the experiments, the two algorithms can have significantly different run times. Given sufficient resources, both algorithms may be run in parallel to ensure that maximum advantage is taken in terms of getting the result as fast as possible.

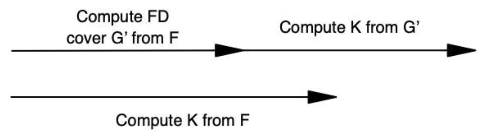


Figure 4: Main parts of Computing Mixed Covers

Algorithm 3 Mixed(G')

```
Require: Set $G'$ of FDs of type $t$
Ensure: A mixed cover $(K, G)$ for $G'$ of type $t$
1: $K \leftarrow$ the set of minimal keys implied by $G'$ [22]
2: $G \leftarrow \{Y \to Z \in G' \mid \forall X \in K(X \not\subseteq Y)\}$
3: return $(K, G)$
```

4.3 Computational Complexity

Before we discuss the worst-case complexity of Algorithms 1 and 2, it is important to recall the complexity of computing the set K of minimal keys from a set F of FDs. Indeed, the problem of deciding whether or not there is a minimal key of cardinality not greater than a given positive integer is *NP*-complete [22]. Nevertheless, Osborne and Luccesi have devised an algorithm [22] for computing K that is polynomial in R, F and K. While $|K|$ can be exponential in $|F|$, this only occurs rarely in practice, so K can often be computed efficiently. This will be confirmed by our experiments later.

Knowing K is of utmost importance to providing efficient access to data, both during update maintenance and query evaluation. Algorithms 1 and 2 compute a mixed cover of a target type t for a given FD set F, which are often efficient in practice.

Theorem 4.3. *Given FD set F, target type t over relation schema R, Algorithm 1 computes a mixed cover (K, G) for F of type t in time in $O(|G'| \cdot |K| \cdot |R| \cdot (|K| + |R|) + C_t(F))$ where $C_t(F)$ denotes the function describing the complexity of computing an FD cover G' for F of type t. Similarly, Algorithm 2 computes a mixed cover (K, G) for F of type t in time in $O(|F| \cdot |K| \cdot |R| \cdot (|K| + |R|) + C_t(F))$.* □

For example, we have $C_{\text{minimal-reduced}}(F) = |F|^2$ and $C_{\text{optimal}}(F) = 2^{|F|}$ based on the worst-case complexity of FD cover computations.

4.4 Closely Related Computations

In case the given FD set G' is already of type t, computing a mixed cover of the same type reduces essentially to computing the set of minimal keys, due to Lemma 4.2. Algorithm 3 summarizes the two simple steps for this computation. Here, the worst-case complexity remains the same as that for computing K.

Corollary 4.4. *Given an FD cover G' of type t over relation schema R, Algorithm 3 computes a mixed cover (K, G) for F of type t in time that is in $O(|G'| \cdot |K| \cdot |R| \cdot (|K| + |R|))$.* □

In case the target type is the same as the input type, and the set K of minimal keys is already given, the computation of a mixed cover is linear in G' and K.

Corollary 4.5. *Given FD cover G' of type t and the set K of minimal keys implied by G' over relation schema R, we can compute a mixed type-t cover (K, G) for G' in time $O(|G'| \cdot |K|)$.* □

5 EXPERIMENTS

We will now test the efficacy of algorithms for computing (mixed) covers. We will state our research questions, summarize the datasets, and then answer each research question before we conclude.

5.1 Research Questions

The main purpose of covers are savings on overheads required to maintain data integrity during updates. With this focus, it is interesting to ask the following research questions.

- Q1: What savings do notions of FD and mixed covers offer?
- Q2: What time does it take to compute these covers?
- Q3: What performance improvement do mixed covers achieve over FD covers on de-normalized and normalized databases?
- Q4: What impact does the use of mixed over FD covers have on industrial workloads?

The first question asks how large the savings in terms of numbers and size are that the different notions of FD and mixed covers achieve. While previous work on FD covers has established hierarchies between notions of FD covers, there has not been any experimental evaluation on that matter. We will provide such experiments and extend them to our notions of mixed covers. However, it should be clear that mixed covers will typically increase the size compared to their FD cover. The second question investigates the time it takes to compute different covers. Naturally, we expect that better covers (those of smaller cardinality or size) take longer to compute. Given an FD set and target type t, we will use Algorithms 1 and 2 to compute a mixed cover of type t, and will also compare their performance. Ultimately, however, what counts the most is how much overhead for integrity maintenance can be saved by a cover. Hence, the third research question asks how much time is spent on integrity maintenance by using different notions of covers. Here, it will turn out that mixed covers, despite their larger numbers or sizes, typically lead to dramatic savings on integrity maintenance. This justifies the one-time effort to compute them, in analytical settings such as active data warehouses where tables are denormalized, and transactional settings where tables are normalized. This will be quantified by showing which performance improvement mixed variants achieve over their FD covers on non-normalized schemata and their decompositions into 3NF. The final research question will provide insight what performance improvements mixed covers achieve over FD covers on the TPC-H benchmark. For that purpose, we will report the performance of queries, refresh and insert operations under different workloads of constraints. In particular, mixed covers also have a significant benefit on query evaluation time.

5.2 Set up and datasets

Set up. Our algorithms were implemented in Java, Version 17.0.7, and run on a 12th Gen Intel(R) Core(TM) i7-12700, 2.10GHz, with 128GB RAM, 1TB SSD, and Windows 10. We used the community edition of MySQL 8.0.29.

Data. We choose one synthetic and 16 real-world datasets that have been used as benchmarks for discovering database constraints from data, including FDs [26, 27, 36]. We use the results of discovery algorithms as inputs for computing FD covers and their mixed variants.

Table 1: Statistics of Datasets Used for Experiments

dataset	#*r*	#*c*	#*fd*	*fd-size*	#*norm*	%bcnf
abalone	4,177	9	54	371	20	75%
adult	48,842	14	68	451	46	100%
fd-red	250,000	30	3,573	100,272	1341	100%
lineitem	6,001,215	16	901	8,755	562	95%
breast <>	699	11	43	207	39	95%
bridges <>	108	13	67	379	44	84%
diabetic <>	101,766	30	97,341	1,080,319	26703	91%
echo <>	132	13	91	613	72	90%
hepatitis <>	155	20	2,995	21,215	1123	73%
ncvoter <>	1,000	19	271	2,166	162	87%
pdbx <>	17,305,799	13	37	226	18	72%
uniprot <>	512,000	30	5,794	49,574	1946	79%
weather <>	262,920	18	2,955	26,763	1154	66%
claims<>	97,031	13	17	104	22	100%
dblp<>	10,000	34	708	3,688	294	82%
routes<>	67,663	9	15	67	6	83%
hospital<>	114,919	15	42	195	19	100%

Given the variety of covers we are investigating, it is an interesting question in what format we present the FD set that is input to the algorithms. Clearly, we would like to avoid presentation in the form of any cover we want to investigate, yet the format should be standardized. As a simple solution, we present the input as set of FDs $X \rightarrow Y$ that hold on the underlying dataset and where X is a minimal attribute set for all the attributes in Y, and where left sides are unique (that is, no two different FDs with the same left side occur in the input FD set). That is, if X' results from X by removing some attribute, then none of the FDs $X' \rightarrow A$ with any $A \in Y$ holds on the underlying dataset. This format permits redundant FDs (and this is the case for all our datasets we consider), which means that each of the different notions of covers have an impact. All input FD sets and their output covers are available as part of our artifact.

Table 1 shows for each dataset, its numbers of rows (#*r*), columns (#*c*), number of valid FDs (#*fd*), their size (*fd-size*), the number of schemata in a lossless, depdendency-preserving decomposition into 3NF that is in BCNF if possible (#*norm*), and the percentage of schemata in BCNF (%bcnf). When nulls occur in the dataset, we append <> to names of datasets to indicate FDs hold when we use the interpretation `null<>null`.

Integrity constraints, such as keys and FDs, encode rules that data ought to obey within the domain of application. The FDs mined from a given dataset include such rules but also FDs that only hold accidentally and for which integrity maintenance is unnecessary. Ideally, we would conduct experiments on real-world data over real-world schemata with real-world FDs. Unfortunately, these are hard to come by, but also not really necessary to gain the insight we want. The introduction has already looked at real-world schemata and FDs with synthetic data that satisfy precisely the FDs specified. Our research questions investigate overall trends for computing covers, such as the growth of runtime and output size in the input, and the update performance on non-normalized and normalized schemata of varying sizes with a variety of integrity constraints. Some of the

Table 2: Numbers and Size for Different Types of FD Covers on Datasets

cover	*input*		*non-redundant*		*reduced*		*minimal*		*minimal-reduced*		*optimal*	
dataset	*no*	*size*	*no*	*size*	*no*	*size*	*no*	*size*	*no*	*size*	*no*	*size*
abalone	54	371	40	269	41	271	40	269	40	210	40	210
adult	68	451	42	269	42	267	42	269	42	267	42	267
fd-red	3,573	100,272	1,550	46,042	1,550	6,203	1,550	46,042	1,550	6,203		
lineitem	901	8,755	677	6,284	679	4,241	720	6,284	677	4,229	677	4,211
breast	43	207	40	192	40	189	40	192	40	189	40	189
bridges	67	379	50	278	52	239	50	278	50	230	50	228
diabetic	97,341	1,080,319	62,249	656,454	62,381	653,339	62,249	656,454	62,249	652,166		
echo	91	613	71	463	71	286	71	463	71	286	71	278
hepatitis	2,995	21,215	1,439	9,689	1,450	9,404	1,439	9,689	1,439	9,349		
ncvoter	271	2,166	159	1,069	169	744	159	1,069	159	704		
pdbx	37	226	21	113	22	92	21	113	21	90	21	88
uniprot	5,794	49,574	2,140	14,855	2,184	12,523	2,140	14,855	2,140	12,320		
weather	2,955	26,763	1,558	12,874	1,575	11,882	1,558	12,874	1,558	11,750	1,558	11, 642
claims	17	104	14	85	14	83	14	85	14	83	14	83
dblp	708	3,688	310	1,615	313	1,286	310	1,615	310	1,277		
routes	15	67	6	22	6	20	6	22	6	20	6	20
hospital	42	195	17	80	18	67	17	80	17	64	17	62

Table 3: Numbers and Size for Different Types of Mixed Covers on Datasets

cover	*non-redundant*		*reduced*		*minimal*		*minimal-reduced*		*optimal*	
dataset	*no*	*size*	*no*	*size*	*no*	*size*	*no*	*size*	*no*	*size*
abalone	(29,25)	(129,136)	(29,25)	(129,130)	(29,25)	(129,136)	(29,25)	(129,130)	(29,25)	(129,130)
adult	(2,42)	(20,269)	(2,42)	(20,267)	(2,42)	(20,269)	(2,42)	(20,267)	(2,42)	(20,267)
fd-red	(3564,9)	(10692,32)	(3564,9)	(10692,21)	(3564,9)	(10692,32)	(3564,9)	(10692,21)		
lineitem	(390,432)	(2135,2718)	(390,433)	(2135,2659)	(390,432)	(2135,2718)	(390,432)	(2135,2653)	(390,432)	(2135,2650)
breast	(2,40)	(10,192)	(2,40)	(10,189)	(2,40)	(10,192)	(2,40)	(10,189)	(2,40)	(10,189)
bridges	(3,47)	(5,241)	(3,49)	(5,229)	(3,47)	(5,241)	(3,47)	(5,220)	(3,47)	(5,219)
diabetic	(6530,62234)	(64378, 656124)	(6530,62365)	(64378, 653219)	(6530,62234)	(64378, 656124)	(6530,62234)	(64378, 652059)		
echo	(39,45)	(111,191)	(39,45)	(111,180)	(39,45)	(111,191)	(39,45)	(111,180)	(39,45)	(111,180)
hepatitis	(104,1433)	(499,9626)	(104,1443)	(499,9367)	(104,1433)	(499,9626)	(104,1433)	(499,9316)		
ncvoter	(113,129)	(399,650)	(113,135)	(399,596)	(113,129)	(399,650)	(113,129)	(399,573)		
pdbx	(11,17)	(39,70)	(11,17)	(39,68)	(11,17)	(39,70)	(11,17)	(39,68)	(11,17)	(39,68)
uniprot	(850, 2093)	(3992,13627)	(850, 2135)	(3992,12291)	(850, 2093)	(3992,13627)	(850, 2093)	(3992,12098)		
weather	(523,1471)	(3814,11471)	(523,1481)	(3814,11109)	(523,1471)	(3814,11471)	(523,1471)	(3814,11036)	(523,1471)	(3814,10944)
claims	(1,13)	(1,72)	(1,13)	(1,72)	(1,13)	(1,72)	(1,13)	(1,72)	(1,13)	(1,72)
dblp	(28,310)	(72,1615)	(28,313)	(72,1286)	(28,310)	(72,1615)	(28,310)	(72,1277)		
routes	(2,5)	(7,16)	(2,5)	(7,16)	(2,5)	(7,16)	(2,5)	(7,16)	(2,5)	(7,16)
hospital	(12,16)	(35,75)	(12,17)	(35,62)	(12,16)	(35,75)	(12,16)	(35,59)	(12,16)	(35,59)

datasets, such as routes, hospital, or bridges have fewer FDs and they all appear to be sensible. We believe that our datasets and FD sets do provide a good range of real-world schemata, real-world data, and real-world-like FDs.

When we measure runtime, we always report the average over hundred independent runs. This includes the experiments where we perform integrity checking using FDs (and keys) after inserting varying numbers of previously removed records into the given dataset. For experiments concerning schemata in 3NF, we used the state-of-the-art algorithm that returns a lossless, dependency-preserving 3NF decomposition that is in BCNF whenever possible [25], and used the sub-schemata which were in 3NF but not in BCNF together with the projection of the original database on the sub-schemata, for our experiments.

Table 4: Percentage of Savings for Different Covers

dataset	*all*				*optimal terminated*			
measure	*number*		*size*		*number*		*size*	
cover	*FD*	*mix*	*FD*	*mix*	*FD*	*mix*	*FD*	*mix*
non-redundant	39.8	25.3	42.7	45.7	33.7	21.9	36.4	39.2
reduced	38.8	24.8	52.8	47.6	32.8	21.4	45.8	40.8
canonical	29.6	20.3	49.9	45.7	20.9	16.7	42.1	39.6
minimal	39.8	25.3	42.7	45.7	33.7	21.9	36.4	39.2
minimal-reduced	39.8	25.3	53.4	48.0	33.7	21.9	46.5	41.2
optimal					33.7	21.9	46.9	41.3

Table 5: Runtime (in ms) required to compute FD covers and mixed covers

cover	*non-redundant*			*reduced*			*minimal*			*minimal-reduced*			*optimal*		
dataset	*FD*	*mix-seq*	*mix-par*	*FD*	*mix-seq*	*mix-par*	*FD*	*mix-seq*	*mix-par*	*FD*	*mix-seq*	*mix-par*	*FD*	*mix-seq*	*mix-par*
abalone	0.27	1.26	1.3	1.81	2.88	2.04	0.96	2.08	1.3	2.08	3.25	2.11	5.6	6.56	5.64
adult	0.43	0.68	0.43	2.45	2.73	2.46	1.04	1.3	1.04	2	2.24	2	12,051	12,051.24	12,051
fd-red	4,426	103,343	315,090	152,891	222,541	315,926	8,033	77,205	315,319	38,037	112,009	315,450			
lineitem	89.4	973.8	2,188.5	1,625	2,579.2	2,187.4	435.7	1,617.1	2,187.8	1,315.7	2,290.8	2,191	603,110	604,208	603,141
breast	0.18	0.28	0.18	0.77	0.87	0.77	0.54	0.64	0.54	0.99	1.09	0.99	26.8	26.89	26.78
bridges	0.36	0.54	0.37	3.24	3.48	3.25	1.77	2.01	1.78	3.71	3.97	3.72	4,098.2	4,098.72	4,098.2
diabetic	2998498	8788805	10849996	26190534	32287042	26204724	11231958	17350185	11245740	20454337	26563510	20468352			
echo	0.69	2.74	2.25	5.52	7.57	5.57	2.08	4.12	2.25	5.42	7.45	5.47	2,941	2,943.03	2,941.05
hepatitis	727.2	1,101.2	1,033.2	6,427.4	7,030.2	6,434.4	2,264.8	2,857	2,271.4	3,231	3,822.4	3,236.6			
ncvoter	13.5	56.22	79.44	125.38	179.69	126.2	30.48	85.56	79.64	38.12	75.91	79.28			
pdbx	0.14	0.37	0.58	1.46	1.81	1.47	0.42	0.73	0.58	0.88	1.24	0.89	4,056	4,056.36	4,056.01
uniprot	3,902	17,000.5	28,503	63,383	78,243	63,426	4,199	12,204	28,465	6,415	14,463.5	28,462			
weather	1,355.1	5,359.4	7,237.35	6,762.85	10,436.85	7,228.6	1,743	4,970.6	7,278.2	3,058.8	6,230.4	7,273	4,365,018	4,368,772	4,365,606
claims	0.03	0.06	0.05	0.2	0.25	0.2	0.14	0.19	0.14	0.33	0.38	0.33	2,945	2,945.05	2,945
dblp	50.78	65.01	50.92	380.02	401.71	380.52	117.3	136.58	117.37	184.23	203.43	184.46			
routes	0.03	0.05	0.03	0.1	0.12	0.1	0.04	0.06	0.04	0.05	0.07	0.05	7.9	7.92	7.9
hospital	0.2	0.46	0.74	1.17	1.52	1.18	0.41	0.73	0.74	0.65	0.98	0.74	98,851	98,851.31	98,851.01

5.3 Savings

For our first research question we are interested in the savings that FD covers and our mixed variants accomplish, in both numbers and sizes. This was the original motivation for FD covers [23].

For each dataset, Table 2 lists the number and size of the FD covers, while Table 3 lists the number and size of the mixed variants. We terminate the computation of optimal covers if it is not completed after 4hrs.

Our main observations are: (1) the numbers and sizes quantify the relationships we expect to hold among FD covers, and between mixed variants. In particular, minimal covers achieve the lowest cardinality possible, reduced covers remove extraneous attributes, while optimal covers guarantee the smallest possible size; (2) for mixed covers, the number and size of minimal keys can be high, which means many FDs are implied by minimal keys; (3) the total in numbers and sizes are typically larger for mixed variants than they are for their FD covers.

Averaging over all datasets, Table 4 lists the average percentage of savings across different covers over the input set, for both FD and mixed covers, respectively. In particular, the savings in numbers for mixed variants are always smaller than those of the corresponding FD covers. This is the same in terms of sizes, but with a few exceptions (non-redundant and minimal covers).

In summary, FD covers typically achieve larger savings in numbers and sizes compared to their mixed variants.

5.4 Performance

Most applications are based on the set of FDs that have been identified as business rules for the underlying domain of application. This typically means that this set is quite stable, however, business rules may change over time. Whenever that happens, corresponding covers may require re-computation. It is therefore also an important criteria at which cost the savings of different covers can be accomplished. Here, we predominantly identify cost with the time required for computing covers.

Table 5 shows the time (in ms) required to compute the different notions of FD covers (*FD*), based on our implementations of algorithms from [24], and their mixed variants using Algorithms 1 (*mix-seq*) and 2 (*mix-par*), respectively. Our main observations are:

(1) The computation times quantify the relationships between original notions of FD covers and those of our mixed variants, respectively. In particular, savings by minimizing numbers and sizes require additional times to obtain them. The cost of guaranteeing optimal sizes by optimal covers is worst-case exponential and that is visible in our results for larger inputs.

(2) The computation of mixed from their corresponding FD covers also comes at a significant cost. This, however, is expected since the problem of deciding whether a given attribute set is a minimal key for a given FD set is *NP*-complete. Emphasizing this point further, Table 6 shows the average percentage of additional time required to compute the mixed variant from the FD variant, across all datasets. Among the covers that can be computed in input-polynomial time (all but optimal covers), computing the mixed variant has a significant overhead over the time for computing the corresponding FD cover. This, however, is intuitive since the problem of computing the set of all minimal keys from a given set of FDs is likely to have no polynomial time algorithm (unless *P=NP*). Since the problem of computing an optimal FD cover is also likely exponential, the overhead for computing a mixed variant of the optimal cover is less significant. We observe that the computation of mixed covers require significant overheads compared to FD covers.

(3) Sequential and parallel algorithms can exhibit significantly different runtimes. The former works well when the input FD set F is *key-heavy*, that is, the difference in computing K from F over computing K from FD cover G' is larger than computing G' from F; while the latter works well in the other case when the input FD set is *key-light*. This shows in Figure 5, which compares the runtime of the sequential relative to the parallel algorithm, in percent. Bars below 100% mean the sequential algorithm performed faster.

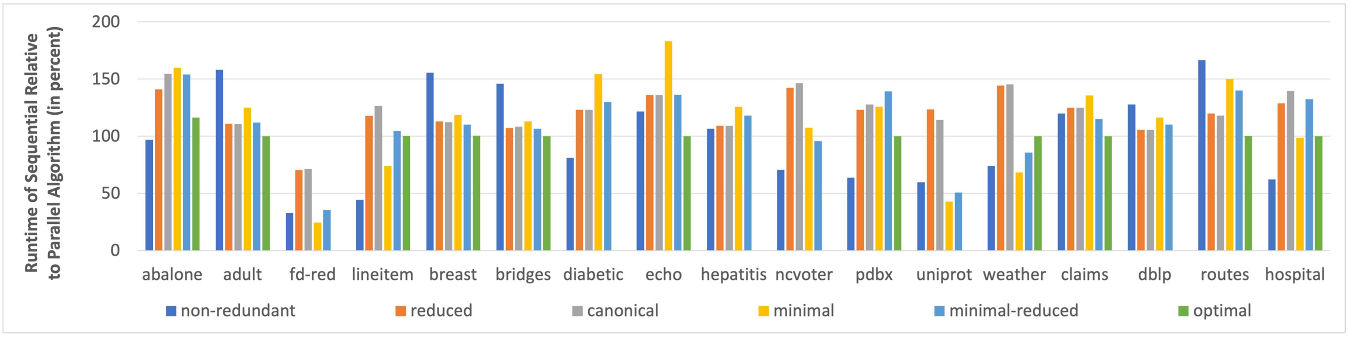


Figure 5: Runtime comparison between sequential and parallel algorithms (sequential faster when percentage below 100)

Table 6: Overhead for Computing Mixed Variant in Percent of Time to Compute FD Covers

overhead	*all*		*optimal terminated*	
dataset/cover	*seq*	*par*	*seq*	*par*
non-redundant	337.22	734.44	233.93	367.58
reduced	28.86	9.71	30.91	5.22
canonical	30.05	8.95	33.06	4.13
minimal	135.01	320.43	87.79	80.22
minimal-reduced	54.42	82.54	40.68	20.17
optimal			1.64	0.06

In summary, the computation of mixed variants requires significant overheads compared to their corresponding FD covers, and results in sizes that are typically larger. Our algorithms can reduce these overheads when the input FD sets are heavy or light on keys. Given sufficient resources, both algorithms can run in parallel to obtain the result as quickly as possible. We will see in the next sections that mixed variants have a tremendous benefit over FD covers when it comes to integrity maintenance under updates.

5.5 Maintenance over Original Schemata

We will now address our third research question and show what time savings mixed variants achieve over their corresponding FD covers when integrity constraints are maintained. In this section, we will look at the original schemata which are not normalized. This may represent scenarios for analytical tasks.

Figure 6 shows the times (in ms) for inserting records over the non-normalized schemata of our datasets. These are averaged across all FD covers, and all mixed covers, respectively. Note that the times are shown on a logarithmic scale. There were four update operations, which differ in the number of records inserted: For $i = 1, \ldots, 4$, ui inserted $i \times 10\%$ of records (after removing them first from the given dataset). In the figure, *avg-FD* refers to the average times taken across all FD covers, while *avg-mix* refers to the average times taken across all mixed covers.

The following main observations can be made. (1) For both FD covers and their mixed variants individually, times increase proportionally to the number of records inserted. (2) For each dataset and for each update operation, the time savings of the mixed variant is around one order of magnitude over their corresponding FD cover.

Table 7: Non-normalized Schemata: Average Boost of Update Performance by Mixed Over FD Covers

measure	*average FD*		*average mixed*		*avg update boost (%)*			
dataset	*no*	*size*	*no*	*size*	*u1*	*u2*	*u3*	*u4*
abalone	42.4	252.3	(29,25)	(129, 132.6)	97.5	97.8	97.9	98.0
adult	46.3	296.4	(2,42.6)	(20, 270.4)	66.7	66.3	63.8	66.4
echo	76.6	388.3	(39, 46.6)	(111, 188.1)	92.9	95.7	96.8	97.4
bridges	53.6	268.4	(3, 47.9)	(5, 232.1)	39.7	40.2	40.5	43.0
breast	40.6	192.9	(2, 40.1)	(10,190.7)	71.2	72.1	71.4	71.4
ncvoter	178.7	1033.1	(113, 138.4)	(399, 629.6)	90.6	90.4	90.5	90.6
claims	15.7	87.9	(1,13.4)	(1,74.7)	11.9	11.9	11.9	11.9
routes	7.4	27.7	(2,6.3)	(7,22.1)	66.6	66.7	66.7	66.6
hospital	21.9	89.3	(12,19.3)	(35,77.6)	73.5	73.7	73.8	73.9

Analyzing point (2) further, Table 7 details the improvement of update performance by mixed variants over FD covers, in percent. The improvements are significant but can vary by quite a margin, that is, between approximately 12% and 98%. This is expected as the schema is not normalized. Indeed, there may still be many non-key FDs whose left-hand sides are not contained in any minimal key and whose right-hand side is not prime. Such FDs may not enjoy any speed up by the UNIQUE indices introduced by minimal keys.

However, the time savings are significant, and it should be stressed that these savings occur whenever updates are made. Due to demands from organizations to make real-time decisions, using active data warehouses or cloud architectures, updates also occur more frequently in analytical settings. Hence, computing more advanced notions of covers pays off, in particular as business rules change very rarely compared to the frequency of updates.

5.6 Maintenance after Normalization

We will now address scenarios where integrity is maintained on schemata resulting from decomposition into lossless, dependency-preserving 3NF, such as transactional workloads. For that purpose, we have computed lossless, dependency-preserving decompositions into 3NF [25], which are in BCNF whenever a lossless, dependency-preserving decomposition into BCNF exists.

Table 1 lists how many relation schemata each 3NF decomposition returned and the percentage of those in BCNF. Only 4 out of 17 datasets have a decomposition into BCNF, but only a small

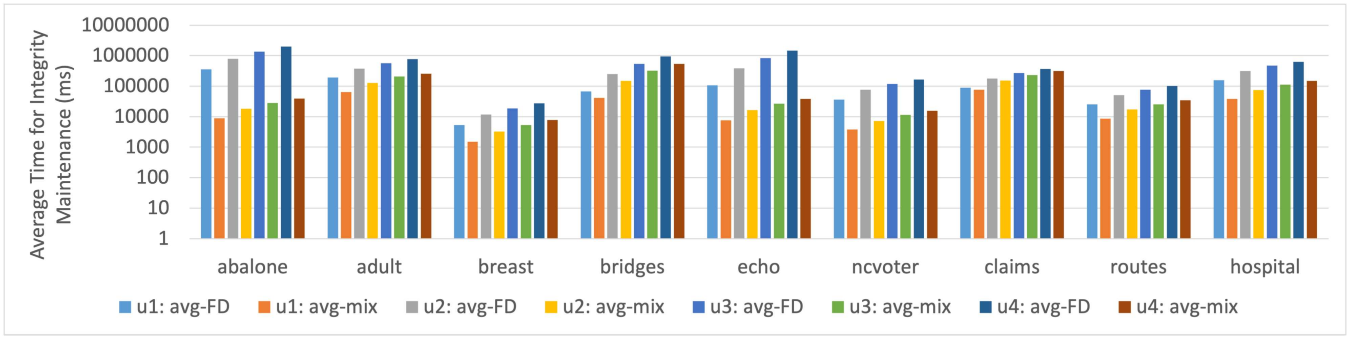


Figure 6: Non-normalized Schemata: Average time in ms for integrity maintenance across FD and across mixed covers

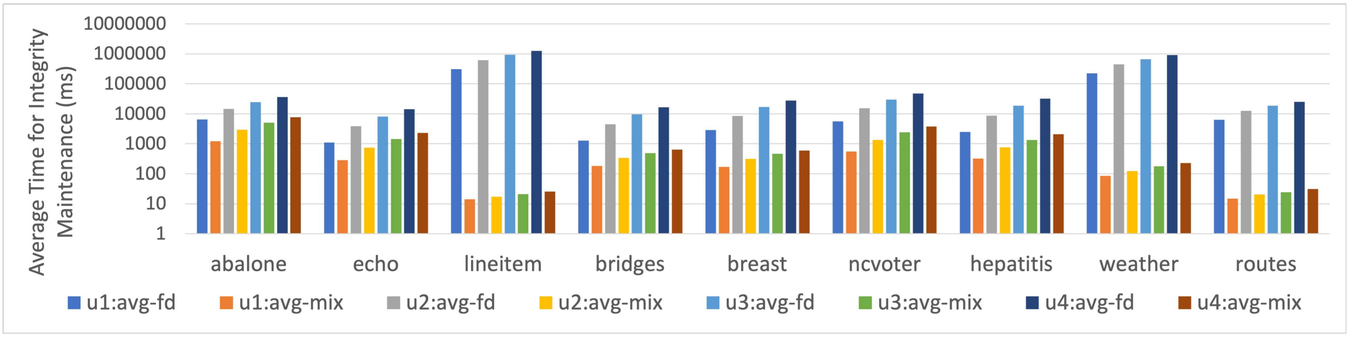


Figure 7: Normalized Schemata: Average time in ms for integrity maintenance across FD and across mixed covers

percentage of relation schemata is not in BCNF. However, these schemata constitute the bottleneck for integrity maintenance.

Figure 7 shows the average time in ms for maintaining integrity under insertions of records across FD and mixed covers on 3NF decompositions of schemata for our datasets. As some of the decompositions have many schemata, we limited our experiments to at most 10 sub-schemata, which we selected randomly. The update operations $u1, \dots, u4$ are defined as before, but are now executed on the records projected to sub-schemata of the decomposition. Times are shown on a logarithmic scale.

Our main observations are similar to the case of non-normalized schemata. However, due to the normalization effort we are able to actually process updates on some of the larger datasets in the experiments (such as weather and lineitem). The improvement of time savings ranges from 3-5 orders of magnitude in these cases.

Table 8 details the improvement of update performance by mixed variants over FD covers, in percent. For each dataset, we list the number *#sch* of schemata in 3NF (but not in BCNF) analyzed and their average number *#records* of records, average numbers and sizes for FD and mixed covers on the sub-schemata as before. It should be stressed that 3NF decompositions are purposefully shifting as much of the semantics of FDs into keys, simply because non-key FDs cause data redundancy and keys prohibit them. As a consequence, a significant proportion of the original FDs can be enforced by minimal keys, which results in tremendous improvements for update efficiency. It is evident that normalization into 3NF leads to robust and significant speed ups for integrity maintenance, ranging between 81% and 99.99% here. Note that the validation of many non-key FDs can still benefit from UNIQUE indices as the left-hand sides of these FDs may be prefixes of these indices [18].

Table 8: Normalized Schemata: Average Boost of Update Performance by Mixed Over FD Covers

measure	*average size*		*average FD*		*average mixed*		*avg upd boost (%)*			
dataset	*#sch*	*#records*	*no*	*size*	*no*	*size*	*u1*	*u2*	*u3*	*u4*
abalone	4	4176	5	26.5	(3.5,2)	(16,9.8)	96.3	96.9	97.3	97.5
echo	7	126	2	8.17	(2, 7.2)	(1, 3.5)	84.3	91.4	93.9	95.4
lineitem	2	6,001,214	3.5	21.7	(3,1)	(16.5,5.5)	99.9	99.9	99.9	99.9
bridges	7	97	2.29	11.29	(2.1, 9.3)	(1, 4.1)	81.9	90.3	93.4	95.1
breast	2	688	2	9	(2,8)	(1,4)	94.1	96.2	97.2	97.8
ncvoter	10	991	3.1	13.6	(2.3,9.1)	(1.4,5)	92.3	93.6	94.1	94.6
hepatitis	10	144	2.7	17.7	(2.1,13)	(1.6,9.8)	88.9	93.4	94.9	95.7
weather	2	262,684	4.5	33.5	(3,21)	(2.5,16.5)	99.9	99.9	99.9	99.9
routes	1	67599	2	7	(2,1)	(8,2)	99.7	99.8	99.8	99.9

In conclusion, mixed covers and normalization work well together: Mixed covers speed up integrity maintenance and this is done robustly at orders of magnitude when schemata are in 3NF. Vice versa, 3NF schemata require mixed covers to benefit from UNIQUE indices resulting from minimal keys.

5.7 Impact on TPC-H Benchmark

On each table of the TPC-H benchmark, we mined the set of FDs that hold on it. Their union F' comprises 1038 FDs of size 9291. Its mixed

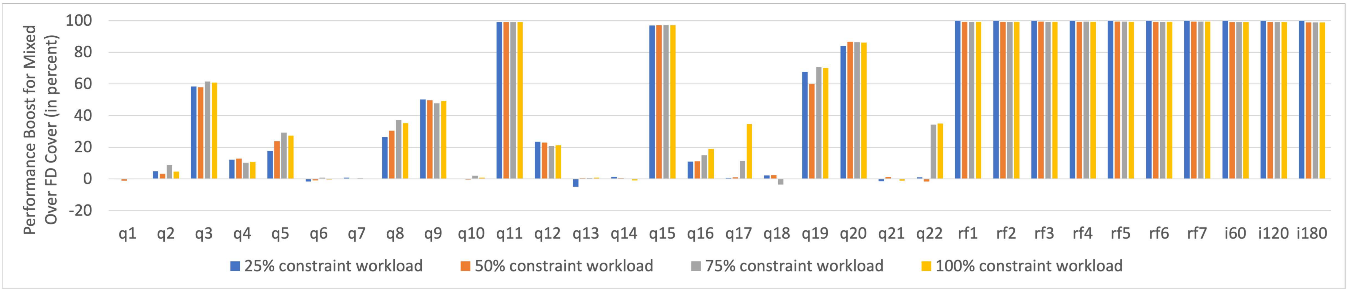


Figure 8: Performance Improvement of Optimal Mixed over Optimal FD Cover Under Different Constraint Workloads

Table 9: Numbers and Sizes of Integrity Workloads for TPC-H Experiments

cover	optimal FD F		mixed optimal (K, G)	
workload	$\|\|F\|\|$	$\|F\|$	$(\|\|K\|\|, \|\|G\|\|)$	$(\|K\|, \|G\|)$
25%	177	46	(109,53)	(43,14)
50%	205	52	(129,72)	(48,18)
75%	268	67	(165,97)	(58,25)
100%	281	70	(175,105)	(60,27)

optimal cover consists of 422 non-key FDs of size 631, and 2166 minimal keys of size 4088. We removed non-sensible FDs manually, such as FDs including column *comments* on their LHS, and then ordered the remaining FDs starting with those we perceived most sensible (those that constitute primary key dependencies) to those least sensible. The resulting FD sets represent 100% of the constraint workload. We then took 25%, 50%, and 75% of those workloads, starting from the most sensible ones. Subsequently, we computed an optimal FD cover and a mixed optimal cover for each of the sets. Numbers and sizes of the covers are summarized in Table 9.

FDs were implemented by triggers, and keys were implemented as UNIQUE constraints. For each workload, and each cover, we ran the entire TPC-H benchmark in a single thread, executing 22 query, 7 refresh, and 3 insert operations sequentially. The numbers of records were *customer*: 150k; *lineitem*: 4,423,659; *nation*: 25; *orders*: 1,500k; *part*: 200k; *partsupp*: 800k; *region*: 5; *supplier*: 10k.

Figure 8 illustrates the performance improvement for each operation resulting from the use of mixed optimal covers in place of optimal FD covers, under each workload. About half the queries benefit significantly, while the use of mixed covers is essential for efficient integrity maintenance during refresh and insert operations. This is rather consistent for all workloads considered.

Table 10 quantifies these observations in more detail. Table 10a shows the average performance improvement in percent over all 22 queries, over all seven refresh operations, and over all insert operations, under each of the four workloads considered, when optimal mixed covers are used instead of optimal FD covers. In addition, Table 10b breaks down the absolute times saved in ms.

Across all constraint workloads, we saved at least 25% of query time, and 99% of refresh and insertion time by using mixed instead of FD covers. Averaged over all workloads, we save more than 53 seconds for queries, 31 minutes for refresh operations, and more than 2 hours and 58 minutes for inserts.

Table 10: Average Performance Improvement of Query, Refresh and Insert Opertations for Optimal Mixed over Optimal FD Covers under Different Constraint Workloads F

(a) Relative in percent

F	query	refresh	insert
25%	25.01%	99.96%	99.95%
50%	25.31%	99.32%	99.03%
75%	28.66%	99.32%	99.04%
100%	29.52%	99.32%	99.04%

(b) Absolute in ms

F	query	refresh	insert
25%	52,981	226,586	643,554
50%	54,238	2,414,183	13,902,559
75%	54,074	2,412,048	14,078,463
100%	53,655	2,416,338	14,184,594

6 CONCLUSION AND FUTURE WORK

Starting from a simple question how FD covers can be used for integrity maintenance, we proposed our notion of mixed variants. Surprisingly, this simple extension bridges classical work on FD covers with that in database normalization and integrity maintenance. Our results show that mixed variants provide the right notion for maintaining data integrity on 3NF schemata and elsewhere. We showed that their relationships known from previous work already hold on schemata in 3NF, and the same relationships apply to their mixed variants. We further illustrated that - while the numbers and sizes typically exceed that of their FD covers - mixed variants achieve orders of magnitude better update performance, on both normalized schemata in transactional settings and non-normalized schemata common in analytical settings. Our sequential and parallel algorithms offer valuable alternatives in reducing the time to compute mixed covers. Finally, we quantified the significant reduction of query evaluation time, and the necessity of using mixed covers for refresh and insert operations on the TPC-H benchmark. The paper shows how simple changes of classical database notions can transform concepts from database theory into best practice.

Future work should look at appropriate notions of covers for more expressive constraints, such as variants of FDs [12, 31], denial constraints [29], order [34] or join dependencies [2]. While FDs have been extended from relational to most other data models, such as Web [37] or graph models [13, 33], covers have not been studied yet. Naturally, our results lend themselves for extensions to more expressive data models, too. Practical work would introduce native support for specifying and maintaining FDs by relational database systems. This is required for schemata that are in 3NF but not BCNF.

REFERENCES

[1] Ziawasch Abedjan, Lukasz Golab, Felix Naumann, and Thorsten Papenbrock. 2018. *Data Profiling.* Morgan & Claypool Publishers.

[2] Alfred V. Aho, Catriel Beeri, and Jeffrey D. Ullman. 1979. The Theory of Joins in Relational Databases. *ACM Trans. Database Syst.* 4, 3 (1979), 297–314.

[3] Paolo Atzeni and Nicola M. Morfuni. 1986. Functional Dependencies and Constraints on Null Values in Database Relations. *Inf. Control.* 70, 1 (1986), 1–31.

[4] Catriel Beeri and Philip A. Bernstein. 1979. Computational Problems Related to the Design of Normal Form Relational Schemas. *ACM Trans. Database Syst.* 4, 1 (1979), 30–59.

[5] Laure Berti-Équille, Hazar Harmouch, Felix Naumann, Noël Novelli, and Saravanan Thirumuruganathan. 2018. Discovery of Genuine Functional Dependencies from Relational Data with Missing Values. *Proc. VLDB Endow.* 11, 8 (2018), 880–892.

[6] Joachim Biskup, Umeshwar Dayal, and Philip A. Bernstein. 1979. Synthesizing Independent Database Schemas. In *SIGMOD.* 143–151.

[7] E. F. Codd. 1971. Further Normalization of the Data Base Relational Model. *Research Report / RJ / IBM / San Jose, California* RJ909 (1971).

[8] E. F. Codd. 1974. Recent Investigations in Relational Data Base Systems. In *Information Processing, Proceedings of the 6th IFIP Congress 1974, Stockholm, Sweden, August 5-10, 1974.* 1017–1021.

[9] Marius Eich, Pit Fender, and Guido Moerkotte. 2016. Faster Plan Generation through Consideration of Functional Dependencies and Keys. *Proc. VLDB Endow.* 9, 10 (2016), 756–767.

[10] Marius Eich, Pit Fender, and Guido Moerkotte. 2018. Efficient generation of query plans containing group-by, join, and groupjoin. *VLDB J.* 27, 5 (2018), 617–641.

[11] Ronald Fagin. 1981. A Normal Form for Relational Databases That Is Based on Domains and Keys. *ACM Trans. Database Syst.* 6, 3 (1981), 387–415. https://doi.org/10.1145/319587.319592

[12] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.* 33, 2 (2008), 6:1–6:48.

[13] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Functional Dependencies for Graphs. In *SIGMOD.* 1843–1857.

[14] Chang Ge, Ihab F. Ilyas, and Florian Kerschbaum. 2019. Secure Multi-Party Functional Dependency Discovery. *Proc. VLDB Endow.* 13, 2 (2019), 184–196.

[15] Sven Hartmann and Sebastian Link. 2012. The implication problem of data dependencies over SQL table definitions: Axiomatic, algorithmic and logical characterizations. *ACM Trans. Database Syst.* 37, 2 (2012), 13:1–13:40.

[16] Tarun Kumar Jain, Dharmender Singh Kushwaha, and Arun Kumar Misra. 2008. Optimization of the Quine-McCluskey method for the minimization of the boolean expressions. In *Fourth International Conference on Autonomic and Autonomous Systems (ICAS'08).* IEEE, 165–168.

[17] Jan Kossmann, Thorsten Papenbrock, and Felix Naumann. 2022. Data dependencies for query optimization: a survey. *VLDB J.* 31, 1 (2022), 1–22.

[18] Tapio Lahdenmäki and Michael Leach. 2005. *Relational Database Index Design and the Optimizers: DB2, Oracle, SQL Server, et al.* Wiley. 1–310 pages.

[19] Mark Levene and George Loizou. 2012. *A guided tour of relational databases and beyond.* Springer Science & Business Media.

[20] Sebastian Link and Ziheng Wei. 2021. Logical Schema Design that Quantifies Update Inefficiency and Join Efficiency. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021.* 1169–1181.

[21] Xiaoxuan Liu, Shuxian Wang, Mengzhu Sun, Sicheng Pan, Ge Li, Siddharth Jha, Cong Yan, Junwen Yang, Shan Lu, and Alvin Cheung. 2023. Leveraging Application Data Constraints to Optimize Database-Backed Web Applications. *Proc. VLDB Endow.* 16, 6 (2023), 1208–1221.

[22] Claudio L. Lucchesi and Sylvia L. Osborn. 1978. Candidate Keys for Relations. *J. Comput. Syst. Sci.* 17, 2 (1978), 270–279.

[23] David Maier. 1980. Minimum Covers in Relational Database Model. *J. ACM* 27, 4 (1980), 664–674.

[24] David Maier. 1983. *The Theory of Relational Databases.* Computer Science Press.

[25] Sylvia L. Osborn. 1979. Testing for Existence of a Covering Boyce-Codd normal Form. *Inf. Process. Lett.* 8, 1 (1979), 11–14.

[26] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. 2015. Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms. *Proc. VLDB Endow.* 8, 10 (2015), 1082–1093.

[27] Thorsten Papenbrock and Felix Naumann. 2016. A Hybrid Approach to Functional Dependency Discovery. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016.* 821–833.

[28] Eduardo H. M. Pena, Eduardo C. de Almeida, and Felix Naumann. 2019. Discovery of Approximate (and Exact) Denial Constraints. *Proc. VLDB Endow.* 13, 3 (2019), 266–278.

[29] Eduardo H. M. Pena, Fábio Porto, and Felix Naumann. 2022. Fast Algorithms for Denial Constraint Discovery. *Proc. VLDB Endow.* 16, 4 (2022), 684–696.

[30] Xiaoning Peng and Zhijun Xiao. 2016. Optimal covers in the relational database model. *Acta Informatica* 53, 5 (2016), 459–468.

[31] Abdulhakim Ali Qahtan, Nan Tang, Mourad Ouzzani, Yang Cao, and Michael Stonebraker. 2020. Pattern Functional Dependencies for Data Cleaning. *Proc. VLDB Endow.* 13, 5 (2020), 684–697.

[32] Hemant Saxena, Lukasz Golab, and Ihab F. Ilyas. 2019. Distributed Implementations of Dependency Discovery Algorithms. *Proc. VLDB Endow.* 12, 11 (2019), 1624–1636.

[33] Philipp Skavantzos and Sebastian Link. 2023. Normalizing Property Graphs. *Proc. VLDB Endow.* 16, 11 (2023), 3031–3043.

[34] Jaroslaw Szlichta, Parke Godfrey, Lukasz Golab, Mehdi Kargar, and Divesh Srivastava. 2017. Effective and Complete Discovery of Order Dependencies via Set-based Axiomatization. *Proc. VLDB Endow.* 10, 7 (2017), 721–732.

[35] Ziheng Wei and Sebastian Link. 2021. Embedded Functional Dependencies and Data-completeness Tailored Database Design. *ACM Trans. Database Syst.* 46, 2 (2021), 7:1–7:46.

[36] Ziheng Wei and Sebastian Link. 2023. Towards the efficient discovery of meaningful functional dependencies. *Inf. Syst.* 116 (2023), 102224.

[37] Cong Yu and H. V. Jagadish. 2006. Efficient Discovery of XML Data Redundancies. In *VLDB.* 103–114.

[38] Carlo Zaniolo. 1984. Database Relations with Null Values. *J. Comput. Syst. Sci.* 28, 1 (1984), 142–166.

[39] Zhuoxing Zhang, Wu Chen, and Sebastian Link. 2023. Composite Object Normal Forms: Parameterizing Boyce-Codd Normal Form by the Number of Minimal Keys. *Proc. ACM Manag. Data* 1, 1 (2023), 13:1–13:25.