A Database Platform for Bioinformatics

Sandeepan Banerjee

Oracle Corporation
500 Oracle Pkwy
Redwood Shores, CA
USA
sabanerj@us.oracle.com

Abstract

In recent years, new developments in genetics have generated a lot of interest in genomic and proteomic data, investing international significance (and competition) in the fledgling discipline of bioinformatics. Researchers in pharmaceutical and biotech companies have found that database products can bring a wide range of relevant technologies to bear on their problems. Benefiting from a number of new technology enhancements, Oracle has emerged as a popular platform for pharmaceutical knowledge management and bioinformatics.

We look at four powerful technologies that show promise for solving hitherto intractable problems in bioinformatics: the extensibility architecture to store gene sequence data natively and perform high-dimensional structure-searches in the database; warehousing technologies and data mining on genetic patterns; data integration technologies to enable heterogeneous queries across distributed biological sources, and internet portal technologies that allow life sciences information to be published and managed across intranets and the internet.

1. Introduction

As the mapping of the human genome draws to a close, there is increasing realization that the 'life' sciences are dependent, as never before, on computing. The atlas of the human genome promises to revolutionize medical practice and biological research for the next millennium: all human genes will eventually be found, accurate diagnostics will be developed for all heritable diseases, animal models for human disease research will be more easily developed, and cures developed for many diseases. Many of these developments will occur, not inside test-tubes in biologists' laboratories, but on high-performance computing platforms, with massive storage systems to store genomic data, databases to search through the data, identifying similarities and patterns, as well as integration software to unify the slices of knowledge developed at globally distributed institutions.

The primary goal of the public and private genomic projects is to make a series of descriptive diagrams maps of each human chromosome at increasingly finer resolutions [1]. This involves dividing the chromosomes into smaller fragments that can be isolated, and ordering these fragments to correspond to their respective locations on the chromosomes. After ordering is completed, the next step is to determine the sequence of bases A,T, C & G in each fragment. Then, various regions of the sequenced chromosomes are to be annotated with what is known of their function. Finally differences in sequences between individuals may be catalogued on a global scale. Correlating sequence information with genetic linkage data and disease gene research will reveal the molecular basis for human variation. Any two individuals differ in about one-thousandth of their genetic material, i.e. about 3 million base pairs [1]. The global population is now about 6 billion. A catalogue of all sequence differences, which will be necessary in the future to find all rare and complex diseases, would run to $18x10^{15}$ entries.

2. Database Support for Sequence Data

As the sequencing community sharply increases its activities to pile up As, Ts, Cs and Gs, it is clear that the

Proceedings of the 26th International Conference on Very Large Databases, Cairo, Egypt, 2000

goals above need industrial-strength database products as well as innovations in underlying database technologies. Databases have, so far, been used largely for managing simple business data – numbers, characters or dates. Few databases have had a native ability to deal with complex data -- whether multimedia, text, spatial data, or gene sequence data. Most databases find it hard to handle high-dimensional data, such as performing similarity queries on gene sequences, spatial queries on locations, or 'looks-like' queries on images. For the specific case of genomic data, we should be able to search for:

- Properties: What are the human sequences that are longer than 10 Kb, and have a specific annotation associated with them?
- Structural similarity: Given a particular sequence, what other sequences resembling this sequence exist in the database – for this organism and for other organisms? (The 'resemble' operation must be able to find sequences that share, say, only isolated regions of similarity, and also score the returned results.)
- Location: Given a gene or a sequence, what are the neighbouring genes/sequences?

Unless databases can treat complex data natively, specialized applications have to be used as custom middle-tiers to perform sequence searches or spatial searches. BLAST (Basic Local Alignment Search Tool) [2] is a set of similarity search programs that can apply a heuristic algorithm to detect relationships between sequences, and rank the 'hits' statistically. However, such loosely integrated specialty middle-tiers have several disadvantages: applications become too large, too complex, and far too custom-built. Even though these mid-tier products can exploit special algorithms to manipulate complex data, they run outside the database server, causing performance to degrade as interactions with the database increase.

Further, optimizations across data sources cannot be performed efficiently. Since BLAST-like servers know nothing about textual annotations, one cannot search for similarity AND annotation efficiently. For example, given a (pseudo) query *'Find the names of all sequences where GappedSearch('IKDLLDTTLVLVNAI++LSS D') returns a score less than 2, AND any annotation associated with the sequence contains the keyword 'Swiss Protein'',* we do not know which of the two clauses in the predicate is more restrictive, and therefore important to evaluate first during query execution.

Finally, each specialty server comes with its own utilities and practices for administering data, making the overall system hard to manage. Since processing for complex data is beset with problems when done outside the database, we have to ask what the best way is to support specific types of complex data inside databases. As it is not clear what constitutes a full set of such types, it seems inefficient to provide, on an ad hoc basis, support for each new type that comes along. In other words, unless all possible complex types can be accommodated in some comprehensive architecture, they will continue to be devilled by issues in re-engineering, cross-type query optimization, uniform programmatic access and so on.

3. Extending Databases

We approached the complex data problem from the standpoint of creating such an architecture. Databases must be made inherently *extensible* to be able to efficiently handle various rich, application-domain-specific complex data types. Extensibility is the ability to provide support for any user-defined datatype (structured or unstructured) efficiently without having to re-architect the DBMS. Such types – which can be plugged into the database to extend its capabilities for specific domains – are also called *data cartridges* [3].

An extensible database system needs support for:

- user-defined types -- the ability to define new datatypes corresponding to domain entities like sequence,
- user-defined operators -- like `Resembles()` or `Distance()` to add domain-specific operators that can be called from SQL,
- domain-specific indexing - support for indexes specific to genomic data , spatial data etc., which can be used to speed the query, and
- optimizer extensibility - intelligent ordering of query predicates involving user-defined types, especially for multi-domain queries.

3.1 User-defined Types

The Oracle Type System (OTS) [4] provides a high-level SQL-based interface for defining types. The behaviour for these types can be implemented in Java, C/C++ or PL/SQL. The DBMS automatically provides the low-level infrastructure services needed for input-output, heterogeneous client-side access for new data types, and optimisations for data transfers between the application and the database and on. Two central constructs in OTS are *object types*, whose structure is fully known to the database, and *opaque types* whose structure is not.

An *object type*, distinct from native SQL data types such as NUMBER, VARCHAR or DATE, is user-defined. It specifies both the underlying persistent data (called 'attributes' of the object type) and the related behaviour ('methods' of the object type). Object types are used to extend the server's modelling capabilities. You can use object types to make better models of complex entities in the real world by binding data attributes to semantic behaviour. There can be one or more attributes in an object type. The attributes of an object type can be the native data types, other object types, 'large objects' or LOBs, or reference types. We also provide collections of native types, objects types, LOBs or references. Object types can have methods to access and manipulate their attributes, and these methods can be run within the execution environment of the database server. In addition, methods can be dispatched to run outside the database. With OTS, it is possible to (i) create database abstractions for sequence, gene, annotation etc., (ii) program behaviour for these abstractions – say `Size()` for a sequence, (iii) create collections of sequence to yield aggregations like chromosome and so on.

The *opaque type* mechanism provides a way to create new fundamental types in the database whose internal structure is not known to the DBMS. The internal structure is modelled in some 3GL language (such as C). The database provides storage for the type instances. Type methods or functions that access the internal structure are external methods or external procedures in the same 3GL language used to model the structure.

The benefit of opaque types arises in cases where there is an external data model and behaviour available to store or manipulate sequences – say as a C library. For instance, object models for genomic use have been devised as part of the Life Sciences Research Domain Special Interest Group (LSR-SIG) under the Object Management Group (OMG) umbrella [5]. Implementing these objects as opaque types enables them to store genomic data persistently in the database, but at the same time call on behaviour implemented external to the database for purposes of insert, updates, deletes or queries on the data.

3.2 User-defined operators

Typically, databases provide a set of pre-defined operators to operate on built-in data types. Operators can be related to arithmetic `(+, -, *, /)`, comparison `(=, >, <)`, Boolean logic (`NOT, AND, OR`), string comparison (`LIKE`) and so on. We have also found it useful to add to Oracle the capability to define domain-specific operators. For example, it is possible to define a `Resembles()` operator for comparing sequences. The actual implementation of the operator is left to the user, and he can choose to bind them to functions, type methods, packages, external library routines and so on. User-defined operators can be invoked anywhere built-in operators can be used — i.e., wherever expressions can occur. User-defined operators can be used in the select list of a `SELECT` command, the condition of a `WHERE` clause, the `ORDER BY` clause, and the `GROUP BY` clause. After a user has defined a new operator, it can be used in SQL statements like any other built-in operator. For example, if the user defines a new operator `Contains()` which takes as input a decoded DNA fragment and a particular sequence, returning `TRUE` if the fragment contains the specified sequence, then we can write a SQL query as

```
SELECT ID FROM DNATABLE WHERE
Contains(fragment,
'GCCATAGACTACA');
```

This ability to increase the semantics of the query language by adding domain-specific operators is akin to extending the query service of the database.

When an operator is invoked, the evaluation of the operator is transformed to the execution of one of the functions bound to it. Just as databases use indexes to efficiently evaluate some built-in operators (a B+Tree index is typically used to evaluate comparison operators), in Oracle user-defined domain indexes (see below) can be used to efficiently evaluate user-defined operators.

3.3 Extensible Indexing

Typically, databases have supported a few standard access methods (B+Trees, Hash Indexes) on the set of built-in data types. As we add the ability to store complex domain data, there arises a need for indexing such data using domain-specific indexing techniques. For simple data types such as integers and small strings, all aspects of indexing can be easily handled by the base database. For gene sequences, however, we would need special indexes to efficiently perform 3-D structural comparison, similarity or substructure search, 'distance' evaluation and so on.

The framework to develop new index types is based on the concept of cooperative indexing where a user-supplied implementations and the Oracle server cooperate to build and maintain indexes for complex types such as genetic, text or spatial data. The user is responsible for defining the index structure, maintaining the index content during load and update operations, and searching the index during query processing. The index structure itself can either be stored in the Oracle database, or externally (e.g. in operating system files), though most implementers find it desirable to have the physical storage of domain indexes within the database for reasons of concurrency control and recovery.

To this end, Oracle introduces the concept of an Indextype. The purpose of an Indextype is to enable efficient search and retrieval functions for complex domains such as text, spatial, image, and genomics. An Indextype is analogous to the sorted or bit-mapped index types that can be found built into the Oracle server, with the exception that the former depends on user implementation.

With such 'extensible' indexing, the user:

- Defines the structure of the domain index as a new Indextype
- Stores the index data either inside the Oracle database (in the form of tables) or outside the Oracle database
- Manages, retrieves, and uses the index data to evaluate user queries.

In the absence of such user-defined domain index capabilities, many applications -- such as the aforementioned BLAST -- maintain separate memory- or file-based indexes for complex data. A considerable amount of code and effort is required to:

- maintain consistency between external indexes and the related database data
- support compound or multi-domain queries (involving tabular values, or data from other domains)
- manage the system (backup, recovery, allocate storage, etc.) with multiple forms of persistent storage (files and databases)

By supporting extensible indexes, the Oracle server significantly reduces the level of effort needed to develop solutions involving high-performance access to complex data types.

3.4 Extensible Optimizer

A typical optimizer generates an *execution plan* for a SQL statement. Consider a `SELECT` statement. The execution plans for such a statement includes (i) an access method for each table in the `FROM` clause, and (ii) an ordering (called the join order) of the various tables in the `FROM` clause. System-defined access methods include indexes, hash clusters, and table scans. The optimizer chooses a plan by generating a set of join orders or permutations, computing the cost of each, and selecting the one with the lowest cost.

For each table in the join order, the optimizer estimates the cost of each possible access method using built-in algorithms. Databases collect and maintain statistics about the data in tables – such as the number of distinct values, the minimum and the maximum, histograms of distribution and so on, to help the optimizer in its estimations.

As discussed earlier, extensible indexing functionality enables users to define new operators, index types, and domain indexes. For such user-defined operators and domain indexes, the extensible optimizer gives developers control over the three main inputs used by the optimizer: statistics, selectivity, and cost. The extensibility of the optimiser lies in the user's ability to collect domain-specific statistics, and, based on such statistics, predict the selectivity and cost of each domain-specific operation. The user's inputs are 'rolled up' with the rest of the optimizer's heuristics to generate the optimal execution plan.

Whenever a domain index is to be 'analysed', a call is made to a user-specified statistics collection function. The representation and meaning of these user-collected statistics is not known to the database, but are to be used later by the user in estimating the cost or selectivity of a domain operation. In addition to domain indexes, user-defined statistics collection functions are also supported for individual columns of a table and data types (whether built-in or user-defined).

The *selectivity* of a predicate or a clause is the fraction of rows in a table that will be chosen by the clause or predicate; it is used to determine the optimal join order. By default, the optimizer uses a built-in algorithm to estimate the selectivity of selection and join predicates. However, since algorithm has no intelligence about functions, type methods, or user-defined operators, the presence of these may result in a poor choice of join order – i.e. a very expensive execution plan. So, if we were to build a domain index for sequences and implement a Contains() operator based on this index, we would also specify the selectivity of the operator. This could be based on the arguments it receives (a very long sequence is likely quite selective, whereas a short sequence like '*GCT*' is not selective at all), or on the actual distribution of sequence data based on analysed statistics. Thereafter, if a user executed a query of the form

```
    SELECT * FROM DNATABLE WHERE
    Contains(fragment,
    'GCCATAGACTACA') AND id > 100;
```

then the selectivity of the first clause of predicate could computed by invoking the user supplied implementation, and an execution plan generated to determine whether the Contains operator should be applied before the `>` operator or vice-versa.

A similar consideration applies to *cost*. The optimizer also estimates the cost of various access paths while choosing an optimal plan. For example, it may compute the cost of using an index as well as a full table scan in order to be able choose between the two. However, for user-defined domain indexes with user-specified internal structre, cost cannot be estimated easily. For proper optimization, the cost model in Oracle has been extended to enable users to define costs for domain indexes, user-defined functions, type methods etc. The user-defined costs can be in the form of default costs that the optimizer simply looks up, or can be full-blown cost functions based on user-collected statistics, which the optimizer calls at run time.

4. Mining Sequence Data

The current approach for finding genes has a large experimental component. Any small increase in the accuracy of computer classification of genes can result in substantial time and cost savings. Oracle has developed a suite of software tools that analyse large collections of data to discover new patterns and forecast relationships [6]. This process of sifting through enormous databases to extract hidden information is called 'data mining'. Mining sequence data can help discover relationships between genes, discover gene expression, discover drugs based on functional information and so on. Oracle's data-mining tool -- Darwin -- has been utilized for bioinformatics. Darwin was built to address the terabyte databases found in genomics databases. In fact, various parallelism technologies built into Darwin ensure that there is no limit to the size of data it can mine. Darwin currently provides classification and regression Trees (C&RT), neural networks, and k-nearest neighbours algorithms, k-means Naïve-Bayes and enhanced clustering (self-organizing maps or SOM) algorithms.

A simple case of mining genetic data could be to classify cancers based solely on gene expression. Classifiers are first trained on the genes in a training set, and then applied to the remaining genes to assign them to specific clusters. Thence, Darwin's algorithms can be used to help identify new clusters. This suggests a general stratagem for predicting cancer classes for other types of cancer, creating new biological knowledge [7].

Another use of mining relates to predicting which sections of a piece of DNA are 'active' and which are not. Chromosomes have coding sequences (exons), interspersed with non-coding sequences (introns.) It has recently been discovered through mining that a non-linear correlation statistic for DNA sequences, called the Average Mutual Information (AMI) [8], is very effective at distinguishing exons from introns. The AMI is a non-linear function based on a vector of 12 frequencies each dependent on the positions of the bases A, C, T & G. The inductive process of mining helps us arrive at such complex insights, which deductive analyses have little hope of unearthing.

When terabytes of data are involved, traditional data mining relies on analysts trying to guess which small subset of the information in a database is relevant. Because of their limited capacity for data, traditional methods often operate on only 1-2% of the data available in every record. Yet discarded variables often contain key information: correlations that aren't obvious, patterns one wouldn't expect, or significant fluctuations that are normally overshadowed by larger trends. Darwin, on the other hand, can afford to look at every bit of data in each record because of its parallel architecture. This architecture is shown in Fig 1.

Darwin's architecture is based on a distributed-memory SPMD (single program multiple data) paradigm. This shared-nothing approach facilitates scalability in performance by reducing inter-processor communications and making optimal use of local memory and disk resources. The processors work on their local section of the dataset and all inter-processor communication is achieved by the use of a message-passing library called MPI, which provides the basic programming and process model. The main architectural component of the server is a unified data access and manipulation library: StarData, which provides most of the data access and transformation infrastructure that supports the machine learning modules StarTree, StarNet, StarMatch etc. A

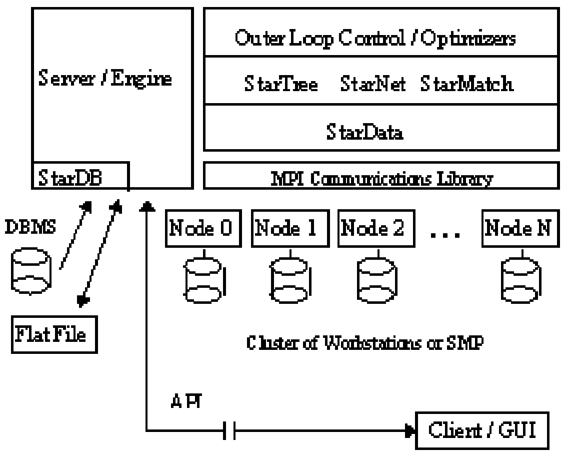
Server / Engine
Outer Loop Control / Optimizers
StarTree StarNet StarMatch
StarData
StarDB
MPI Communications Library
DBMS
Node 0
Node 1
Node 2
...
Node N
Flat File
Cluster of Workstations or SMP
API
Client / GUI

Figure 1: Darwin Data Mining Architecture

toolset provides a client/API to support general or application specific graphical user interfaces (GUIs). The API also allows the toolset to be integrated with, or embedded into, other products. This API can also be called from the member functions of object- or opaque-types, making it possible to integrating the data mining functionality with the data modelling aspects.

Mining of sequence data is still in its infancy because the methodologies are much more involved, and because a large number of tools have to be integrated before progress can be made. However, this area has a potential of yielding rich dividends in the years to come.

5. Integrating Heterogeneous Data

Not all bioinformatics data will exist in the same database. Sequence data for the human genome is likely to end up spread across a handful of public or private databases. Sequence data for other organisms will also be distributed, across hundreds of institutions. Annotations to this data will make it change and grow all the time.

Pharmaceutical companies will have their own private data. Researchers everywhere will like to integrate all sorts of heterogeneous data sources.

Oracle's 'gateway' technologies make it possible for informatics applications to access and manipulate non-Oracle system data. Researchers can query any number of non-Oracle systems from an Oracle database in a heterogeneously distributed environment. Generic connectivity enables connectivity using industry standards such as ODBC and OLEDB. Gateways extend distributed capabilities to a heterogeneous environment – so distributed transactions as well as distributed queries, joins, inserts, deletes can be performed easily. Gateways make the data's location, SQL dialect, network and operating system transparent to the end user, making it easy to implement in a heterogeneous environment.

6. Portal Technologies

While it is important to query on sequences, mine sequence data and so on, it is also important for database platforms to support the dispersion of information over the Internet and intranets. Oracle has emerged as a crucial 'back-end' for commercial web sites – because of new features that enable records to be published directly to browsers as dynamic HTML or XML, server-based Java execution, support for web-based secure transactions, connection pooling etc., coupled with traditional high-availability, scalability and reliability features. While portals related to genomics and bioinformatics need many of the features that commercial horizontal or vertical industry portals do, there are some additional requirements in this domain that are worth discussing.

6.1 'Soft Goods' Sales

Bioinformatics marketplaces buy and sell information rather than 'hard goods'. Portals in this area must be able to measure the usage of soft goods (e.g. the number and complexity of queries against a sequence database, or amount of data downloaded by a subscriber.) Oracle provides a wide array of server-based features as well as application packages to enable soft goods transactions over the Internet.

6.2 Visualization

It is not only important for a bioinformatics portal to serve sequence, aggregate or annotation data, but it is also important for the user to be able to visualize such data. Oracle enables the publishing of graphical data in formats such as the Vector Markup Language (VML) that can be used to display sequences. It is possible to generate XML from the database, and transform this to VML using XSL transformation capabilities. It is also possible to display aggregated or processed data – say scatter plots resulting from mining – as charts or plots using a number of popular charting packages.

6.3 Security & Access Control

Organizations searching against sequence stores want to protect not only the results of their queries, but also the nature of queries themselves. To this end, Oracle provides comprehensive PKI-based security to protect information on data as well as user-sessions.

7. Acknowledgements

Over the years a number of individuals at Oracle have contributed to the gradual building of this platform. Anil Nori, Vishu Krishnamurthy, Jayanta Banerjee, S. Muralidhar, Jags Srinivasan, Ravi Murthy, Nipun Agarwal, Seema Sundara and others have contributed to database extensibility. Pablo Tamayo and others have set the direction of mining sequences. Benny Souder and others have created the 'gateway' and data integration technologies.

8. References

[1] Human Genome Program, *Primer on Molecular Genetics*, Washington D.C, U.S. Department of Energy, 1992.
See http://www.ornl.gov/hgmis/publicat/primer/intro.html

[2] Altschul, S.F., Gish, W., Miller, W., Myers, E.W. & Lipman, D.J, Basic local alignment search tool, *J. Mol. Biol.* 215:403-410, 1990.
See http://www.ncbi.nlm.nih.gov/BLAST/

[3] Oracle Corp., *Oracle8i Data Cartridge Developer's Guide: Release 8.1.5 (Part No. A68002-01)* Redwood Shores, Oracle Corp., 1999.

[4] Oracle Corp., *Oracle8i Concepts: Release 8.1.5 (Part No. A67781-01)* Redwood Shores, Oracle Corp., 1999.

[5] Life Sciences Research Group, Genomic Maps RFP, Philadelphia, Object Management Group, 1999.
See http://lsr.lbl.gov/

[6] Oracle Corp., *Darwin: Release 3.6.1 (Part No. A83710-01)* Redwood Shores, Oracle Corp., 2000.

[7] T.R. Golub, D.K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J.P. Mesirov, H. Coller, M. Loh, J.R. Downing, Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring, *Science, Oct 15 1999:531-537*, 1999. M.A. Caligiuri, C.D. Bloomfield, and E.S. Lander.

[8] I. Grosse, K. Marx, S. Buldyrev, G. Grinstein, H. Herzel, P. Hoffman, A. Li, C. Meneses, and H.E. Stanley. Data Mining of Large Gene Datasets Using the Mutual Information Function. to appear in *Journal of Biomolecular Structure and Dynamics.*