DiscoveryLink: A system for integrated access to life sciences data sources

by L. M. Haas
P. M. Schwarz
P. Kodali
E. Kotlar
J. E. Rice
W. C. Swope

Vast amounts of life sciences data reside today in specialized data sources, with specialized query processing capabilities. Data from one source often must be combined with data from other sources to give users the information they desire. There are database middleware systems that extract data from multiple sources in response to a single query. IBM's DiscoveryLink is one such system, targeted to applications from the life sciences industry. DiscoveryLink provides users with a virtual database to which they can pose arbitrarily complex queries, even though the actual data needed to answer the query may originate from several different sources, and none of those sources, by itself, is capable of answering the query. We describe the DiscoveryLink offering, focusing on two key elements, the wrapper architecture and the query optimizer, and illustrate how it can be used to integrate the access to life sciences data from heterogeneous data sources.

The human genome has been sequenced, but an even greater challenge remains: to use the information created through this and other processes to prevent and cure disease. Knowledge about genes will help us understand how genetics influence disease development, aid researchers looking for genes associated with particular diseases, and contribute to the discovery of new treatments. To progress in this quest, we must start to answer questions such as: What proteins are encoded by the 35 000 human genes? (It is estimated that there may be as many as one million proteins present in the body.) Under what conditions (which cells/when) are they manufactured? What biological pathways do they participate in? Which of these proteins are appropriate targets against which to develop new therapeutics? Finally, what molecules can be identified and optimized to act as therapeutics against these targets? As we start to answer these questions, we may be able to find effective drugs more quickly, to design drugs that are more selective and have fewer side effects, and even to produce drugs that may be tailored to a particular individual's genes (pharmacogenomics). As one indication of the possibilities, some analysts predict[1] that the market for personalized medicine could become as large as $800 million by 2005.

A myriad of different data sources in differing formats have been set up to support different aspects of genomics, proteomics, and the drug design process. Some of these data sources are huge—and growing rapidly. Celera Genomics estimates that it has already generated 50 terabytes of genomic data. With the automated high throughput experimental technologies that have been developed in recent years, it is possible to sequence 20 million DNA (deoxyribonucleic acid) base pairs a day. Technologies for testing chemical compounds have improved as well, making it possible to run high throughput assays at the rate of thousands a day, leading to an explosion in the size of test databases. With the promise of high throughput techniques for protein identification, the volume of data that must be an-

alyzed to find good candidates for drugs is only going to increase.

Not only are there vast quantities of data, but much of the data reside in specialized data sources, with specialized query processing capabilities. Sequence data are often stored in flat files or in databases and then converted to specialized formats (e.g., FASTA[2]) to run particular homology search algorithms (e.g., BLAST[3]). Proprietary chemical structure data sources used for drug design support substructure and similarity search. Reference data are often found in online databases such as MedLine.[4] Assay data are frequently stored in relational format (e.g., ActivityBase[5]), and different companies get information on patents or reports from a variety of text retrieval systems supporting content search of differing degrees of sophistication. These various technologies provide efficient means of finding particular pieces of data of a specific type.

But extracting the data from these specialized stores solves only part of the problem. To obtain real value from these data, they must be combined with data from other sources to give researchers the *information* they desire. Only by integrating the data from many sources will scientists be able to identify correlations across the spectrum from genomics to proteomics to drug design. The variety of different formats and search algorithms, while making it possible to optimize the access to a particular kind of data, unfortunately makes it difficult to integrate data of different types, or even to integrate data from different providers of information.

Many different approaches to integrating access to these data sources are possible. Often, integration is provided by applications that can talk to one of several data sources, depending on the user's request. In these systems, the data sources are typically "hard-wired"; replacing one data source with another means rewriting a portion of the application. In addition, data from different sources cannot be compared in response to a single request unless the comparison is likewise wired into the application. Moving all relevant data to a warehouse allows greater flexibility in retrieving and comparing data, but at the cost of reimplementing or losing the specialized functions of the original source, as well as the cost of maintenance. A third approach is to create a homogeneous object layer to encapsulate diverse sources. This encapsulation makes applications easier to write, and more extensible, but does not solve the problem of comparing data from multiple sources.

Database middleware systems offer users the ability to combine data from multiple sources in a single query, without creating a physical warehouse. By "wrapping" the actual sources, they provide extensibility and encapsulation as well. Several research projects[6–9] have focused on middleware to bridge sources of "nonstandard data types" (that is, types other than the simple strings and numbers stored by most relational database management systems). DiscoveryLink[10,11] is an IBM offering that uses database middleware technology to provide integrated access to data sources used in the life sciences industry. DiscoveryLink provides users with a virtual database to which they can pose arbitrarily complex queries in the high-level, nonprocedural query language SQL (Structured Query Language). DiscoveryLink efficiently answers these queries, even though the necessary data may be scattered across several different sources, and none of those sources, by itself, is capable of answering the query. In other words, DiscoveryLink can optimize queries and compensate for SQL function that may be lacking in a data source. Additionally, queries can exploit the specialized functions of a data source, so that no functionality is lost in accessing the source through DiscoveryLink.

In this paper, we present an overview of DiscoveryLink and show how it can be used to integrate the access to life sciences data from heterogeneous data sources. As motivation, the next section sketches several common research scenarios that substantiate the need for cross-source queries and query optimization, and which we will use to illuminate our discussion. In the section on a wrapper architecture, we describe the DiscoveryLink offering as it exists today. Then in the section on query processing, we walk through the optimization and execution of a few queries, pointing out the benefits of the database middleware approach and highlighting areas for improvement. The next version of DiscoveryLink will be enhanced by changes to query optimization following the Garlic[9] approach. We describe these changes in the section on future enhancements and illustrate their effect on the processing of one of our earlier queries. The section on field experience recounts our experiences with DiscoveryLink to date, describing briefly some ongoing functional and performance studies. In the section on discussion, we reflect on DiscoveryLink's overall role in the process of data integration. In the next section we discuss related work, and then conclude with a report on current and future work.

Motivation

A key feature of DiscoveryLink is that it enables users to ask queries that span multiple sources of information. Such queries arise frequently for researchers in the life sciences. Today, a query that spans multiple sources must be mapped to a series of requests to two or more separate sources. Either the end user must figure out the best sequence of requests, submit them, and then manually intersect the results, or, if the particular type of request is fairly common, an application might be written to hide the sequence of requests. This, however, could require a long and complex program, while providing only limited flexibility.

In this section, we describe several scenarios in which scientists must use multiple data sources in order to get the information they need. We show how the information could be obtained using DiscoveryLink, and contrast that with the way it would be obtained today without the benefit of DiscoveryLink. We refer again to these scenarios (particularly the last one) in future sections. We start with a description of three data sources.

Three data sources. To understand the biological mechanisms of disease and to discover new therapies, researchers need to have access to data from heterogeneous databases. These databases may include DNA databases such as GenBank,[12] protein databases such as SWISS-PROT,[13] proprietary databases for storing structural information about compounds, databases for storing physical properties and activities of chemical entities, and reference databases such as MedLine.[4] A researcher might wish to access and integrate information from some or all of the above-mentioned databases. For our scenarios, we assume that the researcher has available a protein sequence database, a chemical structure database, and a relational database holding assay results.

Each entry in the protein sequence database is indexed by a sequence identifier (*protein_id*) and contains sequence data, citation information, and taxonomic data as well as annotation data that describe the function of the protein, information about protein structure, similarities to other proteins, associated diseases, sequence conflicts, and cross-references to other data. We assume that the data also contain information on the family to which the protein belongs. All of these data are in text format.

The chemical structure database maintains collections of molecules with information about their 2-D chemical structure as well as their physical properties, including molecular weight and logP values (*logP*, the log of the partition coefficient, is an indication of how well the body can use the compound). This database is indexed on a molecule identifier (*compound_id*). This data source can also handle a similarity query. Given a sample molecule (represented in a standard format such as a MOLFILE[14] or as a SMILES[15] string, similarity queries compute a score in the range of [0, 1] for every molecule meeting certain criteria, measuring how similar each is to the sample molecule; the query returns all relevant molecules of a collection ordered by this score. This kind of query results in ranking molecules of a collection in the same way as is done for Web pages in a World Wide Web (WWW) search engine or for images in an image processing system.

For our scenarios, we assume that the researcher has available a protein sequence database, a chemical structure database, and a relational database holding assay results.

The third database, a relational database such as Oracle**, contains information about the assay results. An example of such a database schema would be ActivityBase. The main information in this data source is stored in the "results" table, which details the molecules that have been tested against a given receptor (a type of protein) and lists their IC50 values, which are a measure of the binding affinity of the molecule to the receptor site. Auxiliary tables then list further details of the experimental conditions. Each entry in the results table is indexed by a compound key comprised of the molecule and receptor identifiers.

Scenario 1: A new protein. In this first simple scenario, a biologist at a pharmaceutical company has a new protein sequence. The biologist wants to find out if this sequence is already known and, if not known, find any sequences that are homologous (i.e., similar) to the new sequence. The pharmaceutical company has its own curated copy of the protein database in-house. However, our biologist wants to check the publicly available version to see if there

are any additional data that have not yet made their way into the in-house version.

To accomplish this, our biologist would run a BLASTP search using the new sequence against the in-house version of the database and then do the same with the public version. After obtaining the result sets from both versions, the biologist would have to combine the results, eliminating the sequences present in both (using the protein_id numbers) so as to get a unique list. However, because the application for accessing the in-house version might be different from the Web interface used to access the public version, our biologist needs to combine the results by cutting and pasting, or write an application or script to perform that task. With DiscoveryLink, the entire task can be carried out as a simple query, relying on the SQL "union" operator to spawn the two BLASTP searches and to eliminate duplicates in the result. In this simple scenario, the difference between writing, say, a Perl script and writing an SQL query may not seem too great. However, as more sites are involved, with more interfaces and more choices in how to actually retrieve the result, the difference between hand-coding (and hand-optimizing) the script and writing a nonprocedural statement that is automatically optimized will become increasingly pronounced.

Of course, these results will no doubt lead to other queries, across other databases, as the biologist checks what assays have been done against these homologous proteins, what compounds were tested, and so on.

Scenario 2: A merger. After the merger of two pharmaceutical companies, the discovery informatics group is tasked to provide easy access to both companies' chemical structure and assay databases, located at two different locations. The databases contain information about chemical compounds that have been tested, over the years, against various targets. Merging the databases, which results in a greater number of active compounds, increases the likelihood of developing new leads for a particular therapeutic area. Suppose the researchers are interested in compounds similar to fluoxetine, also known as Prozac**. Though the *compound_id*s in the two databases are likely to be different, the similarity search function can be used to query the databases and extract the required information. For simplicity, we assume that the structures are stored in the same database format, e.g., either SMILES[15] strings or MOLFILEs.[14] If not, a function to convert between representations would be needed.

Today this problem can be addressed by writing an application that accesses both chemical structure databases individually (using a similarity search on each of the two databases for fluoxetine's structure) and puts the two result sets into a common representation. Then a second pair of queries would be done against the assay databases to find what targets these compounds have been screened against. Perhaps the compounds from company A have been tested against serotonin receptors, while those from company B were tested against dopamine receptors. Depending on the activities of the set of compounds, various scenarios emerge: if the activities of compounds from company A are high (low IC50 values) and activities for the compounds from company B are low (high IC50 values), it means that this group of compounds might be selective against the serotonin class of receptors. If the activities of compounds from both databases are high, it means that this group of compounds is not selective toward one of these types of receptors.

In the case of DiscoveryLink, a single query can retrieve the similar structures and their matching assays from both companies' databases. Views could be defined to create a canonical representation of the data. Furthermore, the query will be optimized and executed efficiently. DiscoveryLink gives the end user the perspective of a single data source, saving effort and frustration.

Again, the story is not likely to end here. Before proposing the synthesis and testing of a newly found compound, the researcher needs to know the toxicity profile of the compound and related compounds and also the pathways in which the compound or related compounds might be involved. This would require gathering information from a (proprietary) toxicity database and a database with information on metabolic pathways, such as KEGG,[16] using the structures and names of the compounds to look up the data—a potentially difficult set of queries without the benefit of an engine such as DiscoveryLink.

Scenario 3: Serotonin research. In the brain stem, the most primitive part of the brain, lie clusters of serotonin neurons. The nerve fiber terminals of the serotonin neurons extend throughout the central nervous system from the cerebral cortex to the spinal cord. This neurotransmitter is responsible for numerous fundamental physiological aspects of the body,

including control of appetite, sleep, memory and learning, temperature regulation, mood, behavior (including sexual and hallucinogenic behavior), cardiovascular function, muscle contraction, endocrine regulation, and depression. Serotonin (5-HT, or 5-Hydroxytryptamine) is implicated in a broad range of disorders like depression, schizophrenia, and Parkinson's disease. Major depression results from a deficiency of available serotonin, or inefficient serotonin receptors. Agents that modulate the processing of 5-HT by, for example, inhibiting or stimulating its release, can be useful for treating such diseases. Prozac, for example, is an agent that inhibits the uptake of 5-HT back into the nerve terminal. Analysts project a greater than $10 billion market for serotonin-related drugs in the next decade.

Suppose our scientist, a chemist by background, wants to see what compounds are active against the family of serotonin receptors. To do so, the scientist could ask DiscoveryLink to display the structures of compounds that scored low in an assay in which the receptor screened was a member of the serotonin family. This simple query would in fact require a three-way "join" of information from all three data sources. Without DiscoveryLink, the scientist would need to make (at least) three separate requests: to the assay database to find the assays with low IC50s; to the protein family/sequence database to eliminate those assays where the receptor was not a member of the family of serotonin receptors; and to the structure database to retrieve the structures of the compounds tested in the remaining assays. Note that the second and third steps might, in fact, require multiple requests, one for each assay returned, unless the protein and chemical structure sources can both accept a list of elements to check. In any case, making the individual requests and assembling the results would be a tedious process for the scientist.

Furthermore, there are many possible ways to process this query. Instead of starting with the assay database, our scientist might start by finding out what proteins are in the family of serotonin receptors, and then determine for which of these there were assays with the right activity. If there are only a few serotonin receptors, and many assays, this would probably be the best way to go, because it would be quicker to look up each of the receptors in the family to find its assays than to look up, for each assay, whether its receptor was in the correct family. However, if not aware of these considerations, the scientist could easily make a mistake, increasing the tediousness of the task dramatically. By contrast, since DiscoveryLink processes the entire request at once, it can *optimize* the query, ensuring that the query is executed efficiently.

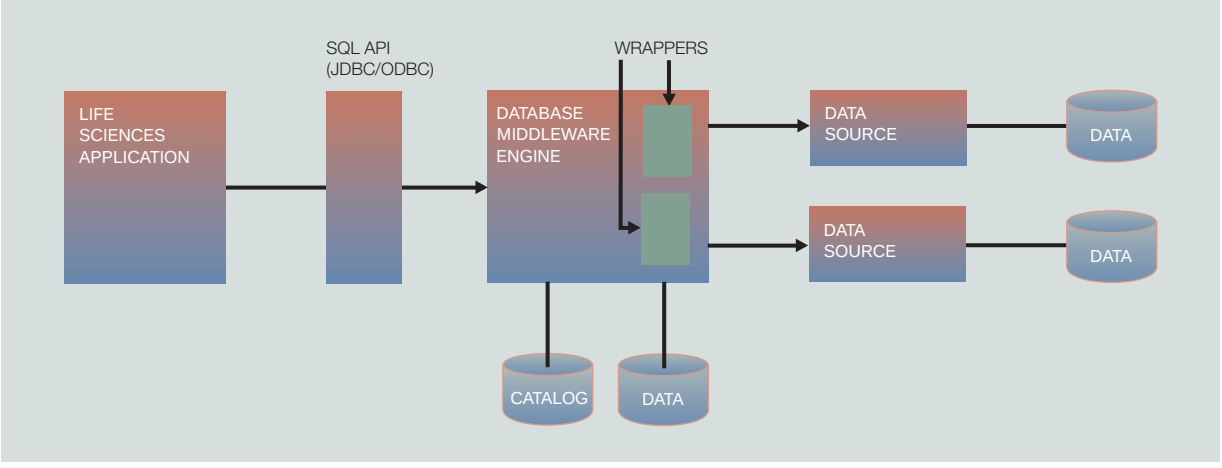
Browsing through the results of this query, our scientist recognizes ketanserin, a compound that is highly selective against the HTR2A class of serotonin receptors. Our chemist would likely investigate compounds similar to ketanserin to find out whether they are selective against one particular class of receptor, in which case they might be good drug candidates, or whether they are active against all classes of the family of serotonin receptors, in which case they would need to be modified in order to be more selective. The scientist might ask a query such as: "Show me compounds with structures similar to ketanserin that are active against any members of the family of serotonin receptors and that have other drug-like characteristics." This query again requires information from all three data sources, and this time exploits the ability of the chemical structure store to search by similarity. It would be even harder for the scientist to determine the best way to perform this query: whether to look for compounds like ketanserin first, or for assays against the family of serotonin receptors, or for the compounds with drug-like characteristics (appropriate molecular weight, logP etc.).

In the sections on query processing and future enhancements, we return to this scenario and describe how these two queries would be processed by DiscoveryLink.

A wrapper architecture

DiscoveryLink is a fusion of Garlic,[9] a federated database management system prototype developed by IBM Research to integrate heterogeneous data, and DataJoiner*, an IBM federated database management product for relational data sources based on DATABASE 2* Universal Database (DB2 UDB*).[17] From the DataJoiner side, DiscoveryLink inherits proven technology for federating relational data sources, as well as DB2's powerful query optimizer and complete query execution engine. From the Garlic side, DiscoveryLink inherits a modular architecture that facilitates integration of new data sources, especially data sources that store nontraditional datatypes and embody specialized search algorithms. In the next two sections, we discuss how this heritage is embodied in the current version of DiscoveryLink. This section is devoted to the DiscoveryLink architecture, and in particular to *wrappers*, software mod-

Figure 1 DiscoveryLink architecture



From L. M. Haas, P. Kodali, J. E. Rice, P. M. Schwarz, and W. C. Swope, "Integrating Life Sciences Data—with a Little Garlic," *Proceedings of the IEEE International Symposium on Bio-Informatics and Biomedical Engineering*, IEEE, New York (2000); © 2000 IEEE, used by permission.

ules that act as intermediaries between data sources and the DiscoveryLink server. The next section describes how the DiscoveryLink server uses information supplied by wrappers to develop execution plans for application queries. For illustration we make use of the three data sources and the query scenario described in the section on motivation (Scenario 3).

The overall architecture of DiscoveryLink, shown in Figure 1, is common to many heterogeneous database systems, including TSIMMIS,[8] DISCO,[18] Pegasus,[6] DIOM,[7] HERMES,[19] and Garlic.[9] Applications connect to the DiscoveryLink server using any of a variety of standard database client interfaces, such as Open Database Connectivity (ODBC) or Java Database Connectivity (JDBC**), and submit queries to DiscoveryLink in standard SQL.[20] (The current offering does not support the INSERT, UPDATE, or DELETE SQL statements.) The information required to answer the query comes from one or more data sources, which have been identified to DiscoveryLink through a process called *registration*. Data sources of interest to the life sciences range from simple data files to complex domain-specific systems that not only store data but also incorporate specialized algorithms for searching or manipulating data. The ability to use these specialized capabilities must not be lost when the data are accessed through DiscoveryLink.

When an application submits a query to the DiscoveryLink server, the server identifies the relevant data sources and develops a query execution plan for obtaining the requested data. The plan typically breaks the original query into fragments that represent work to be delegated to individual data sources, plus additional processing to be performed by the DiscoveryLink server to further filter, aggregate, or merge the data. The ability of the DiscoveryLink server to further process data received from sources allows applications to take advantage of the full power of the SQL language, even if some of the information they request comes from data sources with little or no native query processing capability, such as files.

The DiscoveryLink server communicates with a data source by means of a wrapper,[21] a software module tailored to a particular family of data sources. The wrapper for a data source is responsible for four tasks:

1. Mapping the information stored by the data source into DiscoveryLink's relational data model
2. Informing DiscoveryLink about the data sources' query processing capabilities
3. Mapping the query fragments submitted to the wrapper into requests that can be processed using the native query language or programming interface of the data source
4. Issuing such requests and, following their execution, returning results

Because wrappers are the key to extensibility in DiscoveryLink, one of our primary goals for the wrapper architecture was to enable the implementation of wrappers for the widest possible variety of data sources with a minimum of effort. Our experience with the Garlic prototype has shown that this is feasible. To make the range of data sources that can be accessed using DiscoveryLink as broad as possible, we require only that a data (or application) source have some form of programmatic interface that can respond to queries and, at a minimum, be able to return unfiltered data modeled as rows of a table. The author of a wrapper need not implement a standard query interface that may be too high-level or too low-level for the underlying data source. Instead, a wrapper provides information about a data source's query processing capabilities and specialized search facilities to the DiscoveryLink server, which dynamically determines how much of a given query the data source is capable of handling. This approach allows wrappers for simple data sources to be built quickly, while retaining the ability to exploit the unique query processing capabilities of nontraditional data sources such as search engines for chemical structures or images. Using the Garlic prototype, we validated this design by wrapping a diverse set of data sources including flat files, relational databases, Web sites, and specialized search engines for images and text.

To make wrapper authoring as simple as possible, we require only a small set of key services from a wrapper, and ensure that a wrapper can be written with very little knowledge of DiscoveryLink's internal structure. As a result, the cost of writing a basic wrapper is small. In our experience, a wrapper that just makes the data at a new source available to DiscoveryLink, without attempting to exploit much of the data source's native query processing capability, can be written in a matter of days. Because the DiscoveryLink server can compensate for missing functionality at the data sources, even this sort of simple wrapper allows applications to apply the full power of SQL to retrieve the new data and integrate the data with information from other sources, albeit with perhaps less than optimal performance. Once a basic wrapper is written, it can be incrementally improved to exploit more of the data source's query processing capability, leading to better performance and increased functionality as specialized search algorithms or other novel query processing facilities of the data source are exposed.

A DiscoveryLink wrapper is a C++ program, packaged as a shared library that can be loaded dynamically by the DiscoveryLink server when needed. Typically, a single wrapper is capable of accessing several data sources, as long as they share a common or similar application programming interface (API). This is because the wrapper does not encode information on the schema used in the data source. Thus, schemas can evolve without requiring any change in the wrapper, as long as the source's API remains unchanged. For example, the Oracle wrapper provided with DiscoveryLink can be used to access any number of Oracle databases, each having a different schema. In fact, the same wrapper supports several Oracle release levels as well.

The process of using a wrapper to access a data source begins with registration, the means by which a wrapper is defined to DiscoveryLink and configured to provide access to selected collections of data. Registration consists of several steps, each taking the form of an SQL Data Definition Language (DDL) statement. Several new DDL statements have been defined for DiscoveryLink, and some existing DDL statements have been extended. Each registration statement stores configuration meta-data in system catalogs maintained by the DiscoveryLink server.

The first step in registration is to define the wrapper itself and identify the shared library that must be loaded before the wrapper can be used. A new CREATE WRAPPER statement has been defined for this purpose. The wrapper for chemical structures databases such as the one described in the section on three data sources might be registered as follows:

```
CREATE WRAPPER ChemWrapper LIBRARY
   'libchemdb.a'
```

Similar statements would define the wrappers for the other two data sources.

Note that we have not yet identified particular data sources, only the software required to access any data source of these three kinds. The next step of the registration process is to define specific data sources, using the CREATE SERVER statement. If several sources of the same type are to be used, only one CREATE WRAPPER statement is needed, but a separate CREATE SERVER would be needed for each source. For the chemical structures database in our examples, the statement might be as follows:

```
CREATE SERVER Chem-HTS WRAPPER ChemWrapper
        OPTIONS(NODE 'hts1.bigpharma.com',
                PORT '2003', VERSION '3.2b')
```

Figure 2 Wrapper schemas

Protein_Sequence Wrapper Schema

```
CREATE NICKNAME PROTEINS {
    PROTEIN_ID VARCHAR(30)  NOT NULL,
    NAME VARCHAR(60),
    FAMILY VARCHAR(256),
    DISEASES VARCHAR(256)
} SERVER PROTEINDB
```

Assay Wrapper Schema

```
CREATE NICKNAME ASSAYS {
    COMPOUND_ID VARCHAR(10)  NOT NULL,
    SCREEN_NAME VARCHAR(30)  NOT NULL,
    IC50 DECIMAL(12,10)
} SERVER ORACLE12
```

Molecule Wrapper Schema

```
CREATE NICKNAME COMPOUNDS {
    COMPOUND_ID VARCHAR(10)  NOT NULL,
    STRUCTURE LONG VARCHAR,
    MOL_WT DECIMAL(10,6),
    LOGP DECIMAL(10,2)
} SERVER CHEM-HTS
CREATE FUNCTION MAPPING FOR
    SIMILARITY(LONG VARCHAR, LONG VARCHAR)
    RETURNS FLOAT
    SERVER CHEM-HTS
```

From L. M. Haas, P. Kodali, J. E. Rice, P. M. Schwarz, and W. C. Swope, "Integrating Life Sciences Data—with a Little Garlic," *Proceedings of the IEEE International Symposium on Bio-Informatics and Biomedical Engineering,* IEEE, New York (2000); © 2000 IEEE, used by permission.

This statement registers a data source that will be known to DiscoveryLink as "Chem-HTS," and indicates that it is to be accessed using the previously registered wrapper "ChemWrapper." The additional information specified in the OPTIONS clause is a set of (option name, option value) pairs that are stored in the DiscoveryLink catalogs but meaningful only to the relevant wrapper. In this case, they indicate to the wrapper that the "Chem-HTS" data source can be contacted via a particular IP address and port number, and that it is using version 3.2b of the chemical database software. In general, the set of valid option names and option values will vary from wrapper to wrapper, since different data sources require different configuration information. Options can be specified on each of the registration DDL statements, and provide a simple but powerful form of extensible meta-data. Because options are understood only by wrappers, only the appropriate wrapper can validate that the option names and values specified on a registration statement are meaningful and mutually compatible. As a result, wrappers participate in each step of the registration process, and may reject, alter, or augment the option information provided in the registration DDL statement.

The third registration step is to identify, for each data source, particular collections of data that will be exposed to DiscoveryLink applications as tables. This is done using the CREATE NICKNAME statement. Collectively, these statements define the schema of each data source and form the basis of the integrated schema seen by applications.

In our example, we need three sets of CREATE NICKNAME statements, one set for each of the three previously defined data sources. Based on our previous description of these sources, Figure 2 shows representative CREATE NICKNAME statements that define partial schemas for each source. (The syntax shown is simplified for purposes of illustration.) The Protein_Sequence source exports a single relation, Proteins, with columns representing the unique identifier for a protein, the common (print) name, the protein family, and a list of diseases with which the protein has been associated. In real life, a DBA (database administrator) would likely declare a fuller set of columns, representing more of the information contained in the source; we simplify the schema in the interest of space only. Similarly, the DBA makes visible a single table, Assays, from the Oracle source, for which we show only three columns: the id of the compound being tested, the screen name identifying the protein (receptor) involved, and an IC50 value for the test. The IC50 value represents the concentration of compound required to produce a 50 percent inhibition of enzyme (protein) activity. Finally, the chemical structures database exports a table of compounds along with several important fields, including the structure, molecular weight, and logP. Note that the nickname definitions give the types of attributes in terms of standard SQL datatypes. This represents a commitment on the part of the wrapper to translate types used by the data source to these types as necessary.

Any specialized search capabilities of a data source are modeled as user-defined functions, and identifying these functions by means of CREATE FUNCTION MAPPING statements is the fourth step in registration. Thus the definition of the chemical structures data source in Figure 2 also includes a

CREATE FUNCTION MAPPING statement, registering that source's function *similarity*(A, B). The mapping identifies this function to the query processor and declares its signature and return value (in this case, the similarity score) in terms of standard SQL datatypes. As with nicknames, the wrapper must convert values of these types to and from the corresponding types used by the data source.

Once registration is completed, the newly defined nicknames and functions can be used in queries. When an application issues a query, the DiscoveryLink server uses the meta-data in the catalogs to determine which data sources hold the requested information. To break the query into fragments and develop an optimized execution plan, the DiscoveryLink server must take into account the query processing power of each data source. This information is obtained by requesting a *server attributes table* (SAT) from the data source's wrapper. The SAT contains a long list of parameters that are set to appropriate values by the wrapper. For example, if the parameter PUSHDOWN is set to "N," DiscoveryLink will not request that the data source perform query fragments more complex than:

```
SELECT <column_list> FROM <nickname>
```

If PUSHDOWN is set to "Y," more complex requests may be generated, depending on the nature of the query and the values of other SAT parameters. For example, if the wrapper sets the BASIC_PRED parameter to "Y," requests may include predicates like:

```
. . . WHERE logP > 4
```

The parameter MAX_TABS is used to indicate a data source's ability to perform joins. If it is set to "1," no joins are supported. Otherwise MAX_TABS indicates the maximum number of nicknames that can appear in the FROM clause of the query fragment to be sent to the data source.

Information about the cost of query processing by a data source is supplied to the DiscoveryLink optimizer in a similar way, using a fixed set of parameters such as CPU_RATIO, the relative speed of the data source's processor relative to the one hosting the DiscoveryLink server. Additional parameters like average number of instructions per invocation and average number of I/O operations per invocation can be provided for data source functions defined to DiscoveryLink with function mappings, as can statistics about tables defined as nicknames. Once defined, these parameters and statistics can be easily updated whenever necessary.

This approach is easy for wrapper writers, and has proven satisfactory for describing the query processing capabilities and costs of simple data sources, and of the relational database engines supported by the DataJoiner product. However, it is difficult to extend this approach to more idiosyncratic data sources. Web servers, for example, may be able to supply many pieces of information about some entity, but frequently will only allow certain attributes to be used as search criteria. This sort of restriction is difficult to express using a fixed set of parameters. Similarly, the cost of executing a query fragment at a data source may not be easily expressed in terms of fixed parameters if, for example, the cost depends on the value of an argument to a function. In the section on future enhancements, we describe a more flexible approach, pioneered by Garlic, that will be included in the next release of DiscoveryLink.

Once the optimizer has chosen a plan for a query, query fragments are distributed to the data sources for execution. Each wrapper maps the query fragment it receives into a sequence of operations that make use of its data source's native programming interface and/or query language. Once the plan has been translated, it can be executed immediately or saved for later execution. The DiscoveryLink server's execution engine is pipelined and employs a fixed set of functions (Open/Fetch/Close) that each wrapper must implement to control the execution of a query fragment. When accepting parameters from the server or returning results, the wrapper is responsible for converting values from the data source type system to DiscoveryLink's SQL-based type system.

Query processing

In this section, we show how the DiscoveryLink server creates an optimized execution plan for a query, drawing on information obtained from wrappers about the query processing capabilities of data sources and the location and schema of the data themselves. DiscoveryLink follows a traditional, dynamic programming approach to optimization.[22] Plans are tree structures with Plan Operators, or POPs, as nodes. Each POP is characterized by a fixed set of plan *properties*. These properties include *Cost*, *Tables*, *Columns*, and *Predicates*, where the latter three keep track of the relations and attributes accessed and the predicates applied by the plan, respectively. Each POP works on one or more inputs,

and produces some output (usually a stream of tuples). The input to a POP may include one or more streams of tuples produced by other POPs. DiscoveryLink's POPs include operators for join, sort, filter (to apply predicates), temp (to make a temporary collection), and scan (to retrieve locally stored data). DiscoveryLink also provides a generic POP, called *Remote Query*, which encapsulates work to be done at a data source.

A *plan enumerator* is a component of the optimizer that builds plans for the query bottom-up in three phases, applying pruning to eliminate inefficient plans at every step. In the first phase, it creates plans to access individual relations used in the query. In the second phase, it iteratively combines these single-relation plans to create join plans. Finally, the enumerator adds any POPs necessary to get complete query plans. The winning plan is chosen on the basis of cost. The overall cost is computed by the optimizer using parameter values and statistics supplied by the wrappers during registration, taking into account local processing costs, communication costs, and the costs to initiate a subquery to a data source, as well as the costs of any expensive functions or predicates.[23,24]

Consider the following example, based on Scenario 3 of the section on motivation. Recall that the first step in our chemist's investigation was to look for compounds that were active against the family of serotonin receptors, to find out whether they were selective against one particular receptor or class of receptors (in which case they might be good drug candidates) or whether they were active against all members of the family of serotonin receptors (in which case they would need to be modified so as to be more selective). Seeing the results of the following query in a structure-activity relationship (SAR) table would aid in this analysis:

Show me all the compounds that have been tested against members of the family of serotonin receptors and have IC50 values in the nanomolar/ml range.

Assuming the scientist wishes to see the structures of the compounds as well as their identifiers, this query involves information from all three data sources described above. Using DiscoveryLink, a single query can access these multiple databases and combine the resulting information. In SQL, using the wrapper schemas of Figure 2, the above query can be written as:

```
SELECT a.compound_id, a.IC50, p.name,
   c.structure
FROM Assays a, Proteins p, Compounds c
WHERE a.screen_name = p.protein_id
   AND a.compound_id = c.compound_id
   AND p.family LIKE '%serotonin%'
   AND a.IC50 < 1E-8
```

Of course, our scientist is unlikely to write such a query! Instead, the scientist will probably just fill in some values for the predicates (maybe by selecting them from a list of possible values) in a nice GUI (graphical user interface). Under the covers, the application would generate this query and pass it to DiscoveryLink, which can then parse, optimize, and execute it.

Optimizing the query. As mentioned above, the optimizer examines the query bottom-up, first finding plans for accessing each of the individual tables, then finding plans for joining pairs of tables, and, finally, finding plans for the three-way join. The optimizer uses information from the wrappers about the speed of the various sources, their network connections, and the size and distribution of their data to predict the costs of the various plans. Using information about their query capabilities, it ensures that it does not ask the sources to do anything they cannot do, and adds any operators it needs to compensate for function missing in the sources. It may also be able to rewrite the query in ways that will make query processing more efficient.

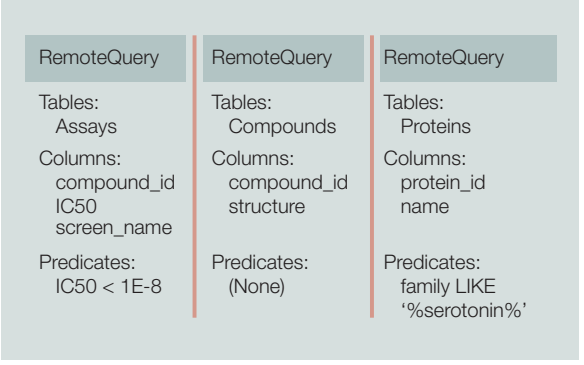
Figure 3 shows the plans created in the first phase of optimization for each of the tables. Each plan consists of a single operator, *RemoteQuery*, but each has a different set of properties. For example, the first plan accesses the assay table, applying the predicate on IC50, and returning the columns needed for the select list (*compound_id*, *IC50*) and to join the Assay table to the protein table (*screen_name*). The second plan accesses the compound table, returning the structure for the select list as well as *compound_id*, to join the compound table with the assay table. The third plan accesses Proteins, applying the LIKE predicate at the data source and returning *protein_id* to join this table to the assay table. In each case, the plan chosen reflects information about the data source's query capabilities that was supplied to the optimizer by the source's wrapper. By setting parameter values in the server attribute table appropriately, the wrapper for the Assay database indicated that the underlying data source could apply basic predicates. As a result, the optimizer could safely del-

egate evaluation of the predicate "IC50 < 1E-8" to the data source. Similarly, the wrapper for the text data source indicated to the optimizer that the source could apply LIKE predicates, allowing the optimizer to include the predicate "p.family LIKE '%serotonin%' " in the access plan for this source.

In the second phase, the optimizer will look at all pairs of tables and construct multiple plans for joining each pair.[25] There will be a plan for each feasible join method (way of executing the join) and for each possible join order (order in which the tables are accessed). For simplicity, we assume there are only two join methods. In the first method, the data resulting from the plan for the inner table of the join (the second table accessed) is brought to DiscoveryLink and stored temporarily, so that the join predicate is evaluated in DiscoveryLink. Alternatively, each join value from the outer table can be sent to the data source, and both the join and the local predicates can be evaluated at the source, once for each outer table value. (This latter join method has been called a *bind join*.[21]) Under these assumptions this phase would produce eight plans, two for joining Assays and Compounds in that order, two for joining them in reverse order, and two for joining Assays and Proteins in that order, two in the reverse. The DiscoveryLink optimizer actually has several more join methods to choose from, and some, such as *hash join*,[26] might well lead to better plans than the ones described here.

Once the two-way joins are built, the optimizer looks at alternative ways of joining these with the single table plans for the remaining table. This query requested no additional work (sorting, for example), so to complete the plans all that is needed is a final *Return* operator that eliminates any extra columns, returning only those needed. Figure 4 shows three of the many plans the optimizer would create for this query. In general, the number of plans examined is exponential in the number of tables being joined. The first plan starts by finding the structure of every compound, then sees which of them received a low IC50 score in an assay, and, finally, looks up the proteins they bound to in those assays to see if they are in the serotonin receptor family. The second plan finds the assays with low IC50 scores, then finds the structure of the compound tested in each of those assays, and finally determines whether the proteins that these compounds bound to are members of the serotonin receptor family. The third plan starts by finding the proteins that are members of the serotonin receptor family, finds assays in which some compound bound tightly to them, and finally retrieves the structures of just those compounds. The first two plans make a temporary table of the results of the remote query on Proteins so that they only access that table once. The first plan probes the Assays table in Oracle once for each compound. Likewise, the third plan asks the chemical structures source to return the appropriate compound structure once for each *compound_id* that generated a low IC50 when screened with a protein in the serotonin receptor family.

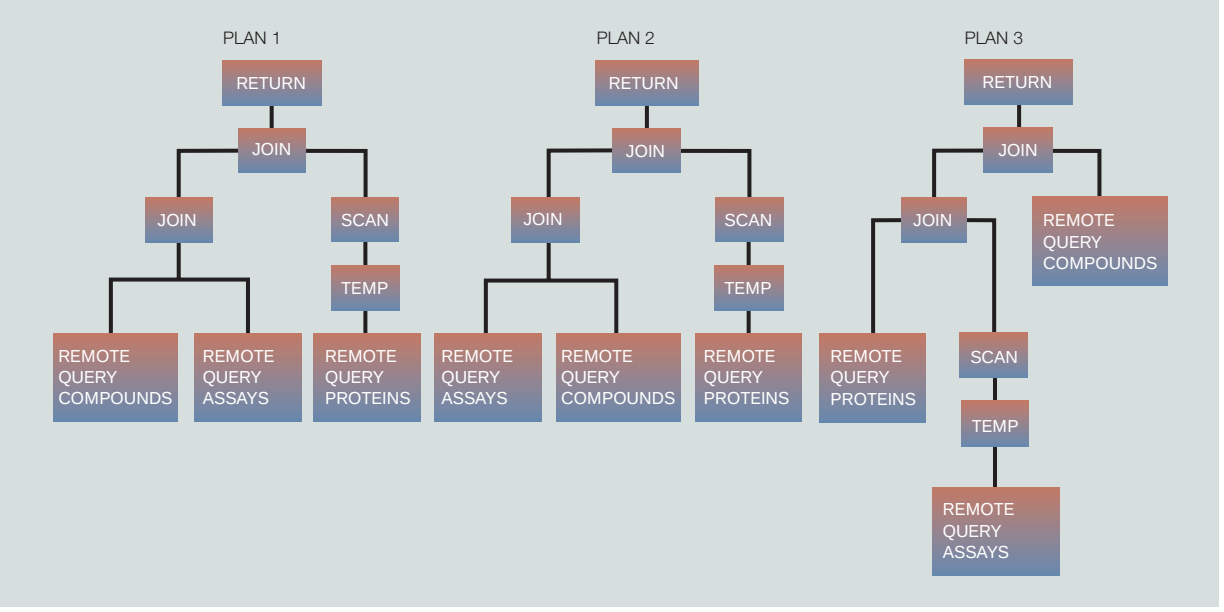
Figure 3 Single table access plans, first phase of optimization



From L. M. Haas, P. Kodali, J. E. Rice, P. M. Schwarz, and W. C. Swope, "Integrating Life Sciences Data—with a Little Garlic," *Proceedings of the IEEE International Symposium on Bio-Informatics and Biomedical Engineering*, IEEE, New York (2000); © 2000 IEEE, used by permission.

Which of these plans is best depends on many factors. Since the first plan begins by retrieving the structure for every compound in the chemical structures database, it is unlikely to be good unless there are very few compounds. The second plan only fetches structures for those compounds that turn up with a low IC50 score in one or more assays, which should be an improvement in most circumstances. Since it accesses the Protein data source only once, creating a temporary table at the server, this plan may perform well if relatively few proteins are in the serotonin receptor family, the DiscoveryLink server is fast, and accessing the text data source is slow. The third plan defers access to the Compounds table until the end, which ensures that only the structures of compounds that qualify for the final result will be retrieved, i.e., those that had low IC50 scores in assays against relevant proteins. In other respects, this plan is similar to the second plan and similar arguments apply.

Figure 4 Three plans for the full query



From L. M. Haas, P. Kodali, J. E. Rice, P. M. Schwarz, and W. C. Swope, "Integrating Life Sciences Data—with a Little Garlic," *Proceedings of the IEEE International Symposium on Bio-Informatics and Biomedical Engineering*, IEEE, New York (2000); © 2000 IEEE, used by permission.

As this example shows, there are many different plans possible for even relatively simple queries. Depending on the data, the selectivity of the predicates, the complexity of the operations, and the machine and network speeds, plan costs may vary by orders of magnitude. A cost-based optimizer is essential to be able to execute cross-source queries with reasonable performance.

Executing the query. DiscoveryLink coordinates execution of the chosen plan, requesting data from the wrappers as the plan dictates. To illustrate, we assume that the optimizer has chosen the second plan of Figure 4 as the best way to execute the query. Plan 2 starts by accessing the Assays table exported by our Oracle database, applying the predicate on IC50. To start this process, DiscoveryLink tells the Oracle wrapper to begin retrieving data for the RemoteQuery operator. The wrapper creates a connection to the Oracle server, and requests the data it needs. Since it is talking to a relational engine, this request is expressed as an SQL query, namely:

```
SELECT a.compound_id, a.IC50, a.screen_name
FROM   ASSAYS a
WHERE  a.IC50 < 0.00000001
```

Those assays that survive the IC50 test are returned to DiscoveryLink. When DiscoveryLink receives the first result row, it asks the wrapper for the chemical structures database to retrieve the structure of the compound tested. In turn, the wrapper makes a request to the chemical structures database itself. This request will likely consist of a call to one of the interface routines supplied by the source, passing in the compound identifier obtained from the assay data. The structure data returned by this call is passed back to DiscoveryLink, which attaches it to the assay data. This process is repeated for each qualifying assay, completing the join between Assays and Compounds. Note that assays for which the tested compound's structure is not available in the chemical database will be dropped from the result. If this is not desired, an outer join could be used to preserve the presence of these assays in the result set.

As soon as the first assay-structure pair is produced by the first join, DiscoveryLink requests that the Protein_Sequence wrapper execute its piece of the plan. As above, the Protein_Sequence wrapper in turn requests data from its source. If the scientist is using Protein_Sequence over the Web, this request looks like a query URL (uniform resource locator), and re-

turns an HTML (HyperText Markup Language) page (or pages) with the result. The wrapper then parses each HTML page to retrieve the next set of results. These results are stored by DiscoveryLink in a local table, and processing of the final join begins. For each combined assay-structure record, DiscoveryLink might scan the local table of protein results, looking for any whose *protein_id* matches the *screen_name* from the assay. Any matches found meet all the criteria of the query and hence are returned to the user.

In this section, we saw how the optimization capabilities of DiscoveryLink work. In fact, for relational sources, and many simpler sources, the Server Attribute Table plus cost parameters approach provides excellent results. For other sources, however, which cannot be neatly characterized by the parameters in the Server Attribute Table, this approach can lead to suboptimal results. For example, a source that could answer some, but not all basic predicates, might be forced to declare that it could not handle basic predicates—leading to inefficient plans if all data must be shipped back to DiscoveryLink before predicates are applied. In the next section, we consider the second half of Scenario 3, in which our chemist focuses on compounds structurally similar to ketanserin. We show how optimizing this query can exploit the more advanced query planning technology that will be included in future versions of DiscoveryLink.

Future enhancements

DiscoveryLink does not yet fully exploit the technology pioneered by its forebears, DataJoiner and Garlic. The process begun with the current version of DiscoveryLink will be completed with the next version, due to be generally available in 2002. Features to be incorporated from DataJoiner include support for "long" datatypes (BLOB, CLOB, etc.), the ability to update information at data sources via SQL statements submitted to DiscoveryLink (including full transaction management for those data sources that support external coordination of transactions), the ability to invoke stored procedures that run at data sources, and the ability to use DiscoveryLink DDL statements to create new data collections at data sources. Other forthcoming features stem from advanced technology that is being added to the database engine at the heart of DiscoveryLink. This engine is more sophisticated than that of either DataJoiner or Garlic. Improvements to the database engine will allow certain queries to be answered using prematerialized automatic summary tables stored by DiscoveryLink, with little or no access to the data sources themselves. Another new feature will allow DiscoveryLink servers with multiple processors to access several data sources in parallel within a single unit of work.

The improvements listed above are important, but the subject of this section is a more fundamental change in the way DiscoveryLink develops optimized execution plans for queries. To demonstrate the need for this change, and how query planning will work in future versions of DiscoveryLink, we return to Scenario 3. After browsing the results of the first query, the chemist decides to investigate the drug potential of compounds similar to ketanserin. The chemist would like to see an SAR table containing the following information:

Show me all the compounds that have been tested against members of the serotonin family of receptors, have IC50 values in the nanomolar/ml range, a molecular weight between 375 and 425, and a logP between 4 and 5. Order the results by how similar the compound tested is to ketanserin.

Like the chemist's earlier query, this request can be expressed as a single SQL statement that combines data from all three data sources:[27]

```
SELECT a.compound_id., a.IC50, p.name,
       c.mol_wt, c.logP, c.structure,
       similarity(c.structure,
                  :KETANSERIN_MOL) AS rank
FROM Assays a, Proteins p, Compounds c
WHERE a.screen_name = p.protein_id
   AND a.compound_id = c.compound_id
   AND p.family LIKE '%serotonin%'
   AND a.IC50 < 1E-8
   AND c.mol_wt BETWEEN 375 AND 425
   AND c.logP BETWEEN 4 AND 5
ORDER BY rank
```

However, accurately determining the cost of the various possible plans for this query is more difficult. In the earlier query, assuming the parameters are correctly set and the statistics characterizing the size and distribution of the data are up-to-date, estimating plan costs and result cardinalities is relatively straightforward. This query introduces two new problems. The first is estimating the cost of evaluating the similarity function. The costing parameters maintained in the current version of DiscoveryLink for a function implemented by a data source include a cost for the initial invocation and a "per-row" cost

for each additional invocation. However, the only way to take the value of a function argument into account is through a cost adjustment based on the size of the argument value, in bytes. This is unlikely to give very accurate results. For example, if different similarity calculation algorithms can be used for different classes of pattern molecules, the cost parameters must be set to reflect some amalgamation of all the algorithms. As another example, a BLAST function asked to do a blastp comparison against a moderate amount of data will return in seconds, whereas if asked to do a tblastn comparison against a large data set it may need hours. A simple case statement, easily written by the wrapper provider, could model the differences and allow more sensible choices of plans. While the costs of such powerful functions can in other cases be hard to predict, many vendors do, in fact, know quite a bit about the costs of their functions, because they often model costs themselves to improve their systems' performance.

The second problem is estimating the cost of ordering the compounds returned by similarity. To DiscoveryLink, the evaluation of the similarity function and ordering the result set by the rank value returned are separate operations. The optimizer first estimates the cost of executing the similarity function the required number of times (itself an estimate based on the selectivity of the other predicates in the query) and then adds on the estimated cost of a SORT operator (for both the case where the SORT is performed by DiscoveryLink and the case where it is performed by the data source). In reality, it is quite possible that the chemical structures data source can order the result by *compound_id* "for free" as a by-product of evaluating the similarity function. However, the cost for ordering by another attribute, e.g., molecular weight, might be quite different, or, the data source might not be able to order results by that attribute at all.

The solution to these and many similar problems is not to define a richer set of parameters for more precisely modeling data sources' query processing capabilities and their costs. Experience with DataJoiner has shown that even for a modest set of data sources, all sharing a common relational data model and query language, the number of parameters required to capture their idiosyncracies soon becomes untenable. The situation will only be exacerbated by the greater number and kind of data sources anticipated for DiscoveryLink.

Instead, the solution, validated in the Garlic prototype, is to involve the wrappers directly in planning of individual queries. Instead of attempting to model the behavior of a data source using a fixed set of parameters with statically determined values, the DiscoveryLink server will request information from the wrapper about a data source's ability to process a specific query fragment. In return, the server will receive one or more *wrapper plans*, each describing a specific portion of the fragment that can be processed, along with an estimate for the cost of computing the result and its estimated size.

Consider the query introduced above. During the first phase of optimization, when single-table access plans are being considered, the chemical structures database will receive the following fragment for consideration:[28]

```
SELECT c.mol_wt, c.logP, c.structure,
       similarity(c.structure,
                  :KETANSERIN_MOL) AS rank
FROM Compounds c
WHERE c.mol_wt BETWEEN 375 AND 425
   AND c.logP BETWEEN 4 AND 5
ORDER BY rank
```

Let us assume that, in a single operation, the chemical structures database can either apply the predicates on molecular weight and logP, or compute the similarity and order the results by rank, but not both. The wrapper might return two wrapper plans for this fragment. The first would indicate that the data source could perform the following portion of the fragment:

```
SELECT c.mol_wt, c.logP, c.structure,
FROM Compounds c
WHERE c.mol_wt BETWEEN 375 AND 425
   AND c.logP BETWEEN 4 AND 5
```

with an estimated execution cost of 3.2 seconds and an estimated result size of 500 compounds. To estimate the total cost of the query fragment using this wrapper plan, the DiscoveryLink optimizer would add to the cost for the wrapper plan the cost of invoking the similarity function on each of the 500 compounds returned and sorting the resulting records by rank.

The second wrapper plan would indicate that the data source could perform the following portion of the fragment:

```
SELECT c.mol_wt, c.logP, c.structure,
       similarity(c.structure,
                  :KETANSERIN_MOL) AS rank
FROM Compounds c
ORDER BY rank
```

with an estimated execution cost of 6.4 seconds and an estimated result size of 300 000 compounds (i.e., all the compounds in the database, sorted by similarity to ketanserin). To compute the total cost in this case, the optimizer would augment the cost for the wrapper plan with the cost of using the DiscoveryLink engine to apply the predicates on molecular weight and logP to each of the 300 000 compounds returned from the data source. Note that when asked to produce this plan, the wrapper has the pattern structure (`:KETANSERIN_MOL`) available, and can take its properties into account to obtain the best possible estimate of how expensive the similarity computation will be. Furthermore, if the result from the data source is naturally ordered by rank, the wrapper's estimate need not include any additional cost for sorting.

Wrappers participate in query planning in the same way during the join enumeration portion of optimization. In our example, the wrapper might be asked to consider the following "bind join" query fragment:

```
SELECT c.mol_wt, c.logP, c.structure,
       similarity(c.structure,
                  :KETANSERIN_MOL) AS rank
FROM Compounds c
WHERE c.mol_wt BETWEEN 375 AND 425
   AND c.logP BETWEEN 4 AND 5
   AND c.compound_id = :H0
ORDER BY rank
```

This is similar to the single-table access, but in this case the chemical structures database is being asked to supply the inner stream for a bind join. For each *compound_id* produced by the rest of the query (and represented above by the host variable `:H0`), the chemical structures database is asked to find the chemical properties of the corresponding compound and its similarity with respect to ketanserin, and return them if the properties satisfy the predicates on molecular weight and logP. If the data source cannot do lookups by *compound_id*, the wrapper would return no wrapper plans at all for this request. If such lookups are supported, the wrapper would return one or more plans, as above, and indicate in each one whether the similarity computation or any of the additional predicates would also be evaluated.

Since a wrapper may be asked to consider many query fragments during the planning of a single query, it is important that communication with the wrapper be efficient. This is achieved easily in DiscoveryLink, since the shared library that contains a wrapper's query planning code is loaded on demand into the address space of the DiscoveryLink server process handling the query. The overhead for communicating with a wrapper is therefore merely the cost of a local procedure call.

The improved approach to query planning described in this section will have many advantages over DiscoveryLink's current methodology. It is both simple and extremely flexible. Instead of using an ever-expanding set of parameters to invest the DiscoveryLink server with detailed knowledge of each data source's capabilities, we let this knowledge reside where it falls more naturally, in the wrapper for the source in question, and ask only that the wrapper respond to specific requests in the context of a specific query. As the examples above have shown, sources that only support searches on the values of certain fields or combinations of fields are easily accommodated, as are sources that can only sort results under certain circumstances or can only perform certain computations in combination with others. Since a wrapper need only respond to a request with a single plan, or in some cases no plans at all, the new approach does not sacrifice the current system's ability to start with a simple wrapper that evolves to reflect more of the underlying data source's query processing power.

This approach to query planning need not place too much of a burden on the wrapper writer, either. In Reference 29, we showed that it is possible to provide a simple default cost model and costing functions, along with a utility to gather and update all necessary cost parameters. The default model proved to do an excellent job of modeling simple data sources, and did a good job of predicting costs even for sources that could apply quite complex predicates. Reference 29 further showed that even an approximate cost model dramatically improved the choice of plans over no information or fixed default values. We therefore believe that this method of query planning is not only viable, but necessary. With this advanced system for optimization, DiscoveryLink will have the ease of extension, flexibility, and performance required to meet the needs of life sciences applications.

Field experience

DiscoveryLink is a new offering, and as a result, we are only beginning to understand how it will be used in practice. Today, two customer pilots are underway. The first focuses on linking chemical information with biological information by bringing together data about the structure of compounds with information on assays that have been done using these compounds. The second pilot is linking chemical, biological, and bioinformatic data, stored in a combination of (different) relational databases and flat files. (These pilots and an earlier study with Garlic were the inspiration for our examples.) In both pilots, the information is geographically distributed, spanning in one case, the United States, and in the other, both shores of the Atlantic Ocean. The schemas used to represent the information in both cases are quite complex, involving 30 or more nicknames, and requiring complex joins and unions both within and across sources to assemble information required for the respective applications. Hence the query processing capability of DiscoveryLink is being well tested by these projects.

Additionally, several vendors of life sciences data sources are considering offerings which would couple DiscoveryLink with their sources and with an application or an object framework to build a platform for data integration. These vendors see that the ease of linking their data to data from other sources will help to distinguish their offerings from those of others in the field. Further, an object layer on top of DiscoveryLink would make it more attractive to a broader, not necessarily SQL-savvy audience.

Performance is a key issue for any data management and retrieval system, and a number of questions arise for a middleware system such as DiscoveryLink. One relatively simple question is, what effect will going through the DiscoveryLink middleware have on the performance of queries against a single data source? In other words, if a user were to issue the same query both through DiscoveryLink and directly to the data source, what would be the difference in the execution times?

We have done an initial study on this issue with one customer, Aventis. In this experiment, we ran a set of their existing queries against both their production database (PrDB) and against a DiscoveryLink installation configured to access (via the relational wrapper) the same database. Queries were submitted via their existing Web-based query application, which was modified to submit queries against either DiscoveryLink wrapping PrDB, or directly against PrDB. The application, Web server, PrDB, and DiscoveryLink all ran on separate machines: the application on a Compaq running Windows NT** 4.0, the Web server on a second Compaq running Windows NT 4.0, IIS** (Internet Information Server) 4.0, and IE (Internet Explorer**) 5.0, PrDB on an Alpha 2100 running Windows NT 4.0, and DiscoveryLink on an RS/6000* H70 running AIX*. Pushdown was enabled, so DiscoveryLink could choose to use as much or as little of PrDB's processing power as it saw fit. Two experiments were done, a functional test and a load test.

In the *functional test*, virtual users (simulated via Web-based testing software) ran scripts consisting of a sequence of steps. In each script, the virtual user would log on to a Web-based application, and run a sequence of two to four queries, then log off of the application. Each script was run 20 times before proceeding to the next, and all tests against PrDB were completed before testing against DiscoveryLink began. (Hence both systems had the opportunity to benefit from any buffering possible.) Tests were run during quiet hours, but the network was not isolated during testing. Total transaction time was measured for each run of each script, and averaged over the 20 runs. In addition, the query results were tested to verify that correct answers were being returned. No errors were found. In all, nine different scripts were run. Queries ranged from selections against a single table to four-way joins, usually including a mixture of inner and outer joins. Many had subqueries, some of which were unions of simpler queries. Both the number of fields selected and the number of predicates varied greatly in number, and often involved complex functions. The amount of data returned also varied from query to query, though none retrieved huge numbers of results. Perhaps most important, each script was representative of the way scientists at Aventis typically use the system to search for studies, protocols, compounds, and/or libraries.

Results of the functional test are shown in Table 1. All times are in seconds. In general, transactions against DiscoveryLink performed comparably to transactions directly against PrDB. In some cases, DiscoveryLink was, on average, a few seconds slower, in others a few seconds faster. In one case, for script number four, the transactions through DiscoveryLink were substantially faster than those directly against PrDB.[30] In all, we concluded that at least for Aventis's standard sorts of transactions there was no

performance penalty for using the DiscoveryLink middleware.

The *load test* evaluated the robustness of DiscoveryLink as the number of simultaneous users was increased. Scalability is essential for any database system, but especially so for database middleware, because requests that might originally have been submitted to multiple independent systems may all be routed through the middleware instead. (For example, an application that previously had to submit separate requests to the ADME [absorption, distribution, metabolism, and excretion] and high throughput screening databases can now send both to DiscoveryLink, so DiscoveryLink will see a higher number of requests than any one underlying data source.) The load tests used scripts similar to those used to run the functional test, and were driven by the same testing software. Several different scenarios were run. In one, all virtual users ran the same script, while in another, half of the virtual users ran one script and half another. In the final scenario, the virtual users were divided into five groups, each of which ran a different script. Each scenario was run for 20 minutes starting with one virtual user and quickly building to 20. Experiments with greater maximum loads (40 and 60 virtual users) were also run, but high standard deviations and large numbers of errors from other components of the system rendered the measurements less reliable.

The results of the load test can be found in Table 2. Again, we measured the total transaction times from start to end of script, and took the average over all executions for all virtual users. Times are again shown in seconds. In general, results were not significantly different between the two application configurations (direct against PrDB and direct against DiscoveryLink). The DiscoveryLink configuration performed better on both scripts in the two-script scenario, and worse for the five-script scenario, though the variability of the results for this latter case makes conclusions hard to draw. What is clear is that at 20 users, there was no significant difference between the configurations (again, DiscoveryLink is not adding overhead), and response times for both configurations are comparable to those when only a single user is running (i.e., both configurations scaled well).

So far, we have only discussed queries against a single data source. What about the cross-source queries for which DiscoveryLink is intended? We are working with Aventis to develop a benchmark for these as well, in which we will compare the performance of cross-source queries against DiscoveryLink with that of an application asking multiple queries of distinct sources and then assembling the results. In the meantime, we rely on studies with Garlic and DB2 DataJoiner, the two key components of DiscoveryLink. In Reference 31, Daimler-Benz compared the performance of three state-of-the-art middleware systems to determine the best platform for a new application that needed to combine data from multiple database systems. Their benchmark covered a broad range of workloads, including single-user and multiuser tests with queries ranging from simple selections and projections to complex joins and aggregates. DataJoiner performed well in virtually all tests. The authors draw particular attention to the join tests, in which DataJoiner's performance was up to 60 000 percent better than the competition's, concluding that "since the integration of heterogeneous schemas is mainly done by means of join operations, a well-designed query optimizer plays a kernel role in the solution to the heterogeneity problem because it greatly influences the performance."

Experiments using the Garlic research prototype indicate that query optimization is important for cross-source queries even when the sources are nonrela-

Table 1 Results of the functional tests on DiscoveryLink

Script	PrDB		DiscoveryLink	
	Avg. RT	Std. Dev.	Avg. RT	Std. Dev.
1	39.95	0.96	40.27	0.85
2	66.98	3.12	64.48	2.57
3	33.08	2.17	31.59	2.03
4	53.10	4.00	43.44	2.78
5	32.57	2.07	31.45	1.46
6	33.72	1.97	33.78	1.72
7	33.67	0.64	34.22	2.69
8	36.92	4.84	42.47	7.04
9	32.84	2.78	32.46	2.25

Table 2 Results of the load tests on DiscoveryLink

Scenario	20 Users			
	PrDB		DiscoveryLink	
	Avg.	StdD	Avg.	StdD
Single script	39.4	1.6	40.2	1.6
Script 1 of 2	44.7	3.5	37.1	2.9
Script 2 of 2	49.0	3.4	41.0	3.4
Five scripts	46.4	12.2	53.3	8.6

tional and highly heterogeneous. The Garlic-style optimizer provides the flexibility needed to choose good quality plans under these circumstances.[32] A follow-up study[29] showed that an accurate cost model is essential, hence the need to adopt the new query planning interface outlined in the section on query processing.

To summarize, today we can state with a fair amount of confidence that the use of DiscoveryLink will not introduce significant overhead for queries accessing a single data source, and that DiscoveryLink will perform well even under significant loads. Further, we have reason to believe, from both the DataJoiner and the Garlic studies, that performance on cross-source queries will be good: as long as good plans exist, DiscoveryLink should find them. We expect to have further confirmation of this from our current pilot projects, which are using DiscoveryLink in a variety of interesting ways as infrastructure for scientific research in the life sciences.

Discussion

From the preceding pages, we hope it is clear that DiscoveryLink can play a useful role in integrating access to life science data. Yet DiscoveryLink is not magic; a completely integrated information space requires significant additional work. In particular, DiscoveryLink does not solve the problems of semantic data integration. In many, if not most, research labs, similar or related information is often modeled differently in different data sources. The discrepancies may range from simple formatting differences (one data source uses uppercase, another lower), to differences in vocabulary (one source refers to Tylenol**, another to Acetaminophen). Common keys may not exist between sources because objects were identified differently by different data providers.

While DiscoveryLink does not eliminate the problems caused by semantic conflicts, it does offer some facilities that can be used to hide conflicts or translate between representations. By writing queries, for example, that explicitly call translation functions, or that join in a translation table or data dictionary, many conflicts can be resolved. In the examples above, an uppercase function could be used to allow the formatting difference to be bridged, and a join to a lexicon would eliminate the terminology problem. A DBA might have to build a translation table to map between different keys in different sources; DiscoveryLink offers a place to store the table and the ability to use it in queries across these sources. Such approaches "solve" semantic problems at the expense of query processing time, but do not require converting and rebuilding entire databases. The task of reconciling the differences by writing appropriate queries and translation tables or functions is, however, left to the DBA or application programmers. DiscoveryLink merely provides the capability.

Another characteristic of life sciences data and research environments is frequent change. Data are being constantly accumulated, with volumes increasing rapidly. As more data of a particular type are acquired, and better understood, schemas change to reflect the new knowledge. Further, new sources of information are always appearing as new technologies and informatics companies evolve. In such an environment, flexibility is essential.

DiscoveryLink has been designed with that goal in mind. The powerful query processor and nonprocedural SQL interface protect applications (to the extent possible) from changes in the underlying data source, due to the principle of logical data independence. Often a new source of information can be added simply by registering it and adjusting a view definition to include it. Changes in interfaces can often be hidden from the application by modifying the translation portion of the wrapper, or installing a new wrapper with the new version of the source. The query processing technology is built to handle complex queries, and to scale to terabytes of data. Hence the database middleware concept itself contributes to dealing well with change.

Further, the wrapper architecture has been designed for extensibility. Only a small number of functions need to be written to create a working wrapper. Simple sources can be wrapped quickly, in a week or two; more complex sources may require from a few weeks to a few months to completely model, but even for these a working wrapper with perhaps limited functionality can be completed quickly. Templates are provided for each function today, and default cost modeling code will be provided for the next version. Wrappers are built so as to enable as much sharing of code as possible, so that one wrapper can be written to handle multiple versions of a data source, and so that wrappers for similar sources can build on existing wrappers. The ability to separate schema information from wrapper code means that changes in the schema of a data source need not require code changes in the wrappers. The addition of a new data source requires no change to any existing wrappers.

Thus the wrappers also help the system adapt to the many changes possible in the environment.

While not a complete solution to all heterogeneous data source woes, DiscoveryLink is well-suited to the life sciences environment. It serves as a platform for data integration, allowing complex cross-source queries and optimizing them for high performance. In addition, several of its features can help in the resolution of semantic discrepancies, providing mechanisms DBAs can use to bridge the gaps between data representations. Finally, the high-level SQL interface and the flexibility and careful design of the wrapper architecture make it easy to accommodate the many types of change prevalent in this environment.

Related work

Most data retrieval systems in the life science industry today are point solutions, "solving" the problem of searching or managing one particular type of data. Each domain in the life science industry has its own complicated data types and database formats. For example, in the cheminformatics domain, there are approximately 30 different formats for storing structural information for molecules. The problem is made even more complex by the diversity of database schemas and sources for chemical inventory, compound registry, compound properties, assay protocols, and synthesis protocols. Point solutions in the cheminformatics domain include algorithms for searching the databases for structures (e.g., MDL[33] and Daylight[34]), solutions for calculating compound properties,[35,36] and applications to study interactions of small molecules with macromolecules such as proteins.[37,38] Similarly in bioinformatics/genomics, the number of data types and data sources is very broad.[39] While these solutions enable many applications that would otherwise not be possible, they also create islands of data that the end user is forced to address. By allowing integration of these heterogeneous solutions, DiscoveryLink provides a means of bridging the data islands they create.

Other vendors are trying to integrate data from a specific domain—a huge problem in and of itself. Many of these vendors have well-established products in a particular domain. For example, MSI[40] and Oxford Molecular[41] provide products that integrate several related data sources. Genomica Corporation's[42] tools combine clinical, epidemiology, genetic, molecular biology, and biochemistry applications into a single software environment that spans a number of domains, enabling scientists to accelerate genetic discoveries and pharmacogenomics. The Genomica Reference Database (RDB) centralizes public domain mapping data from worldwide genome centers. All of these systems integrate specific data sources rather than providing a general framework for data integration as DiscoveryLink does.

DiscoveryLink serves as a platform for data integration, allowing complex cross-source queries and optimizing these for high performance.

More general work on integrating heterogeneous data sources for the life sciences domain includes Kleisli,[43] OPM,[44] TAMBIS,[45] and SRS.[46] Kleisli's CPL language allows the expression of complicated transformations across heterogeneous data sources, but its procedural nature makes optimization difficult. CPL is geared toward biomedical sources, while SQL (used by DiscoveryLink) is more general purpose. OPM has a more flexible object model than DiscoveryLink, but its multidatabase query processor has a less powerful optimization capability. TAMBIS has concentrated more on the benefits of providing a source-independent ontology of bioinformatics concepts and less on the details of efficient cross-source query processing.

SRS[47] (Sequence Retrieval System) is an indexed flat-file system, built on the model of a document retrieval system. The data files contain structured text, labeled with identifiable field names, e.g., author, keyword, organism, etc. Fields are parsed and an index is built for each field. The user can query the data set using the parsed terms (keywords, author name, etc.) in Boolean combination. There are in excess of 500–600 independent sequence-related data sets available in the public domain, each in a slightly different format that research scientists would like to access. SRS has created a parser that, with a modest amount of work, can be configured to parse a new data set and develop queriable indexes to it, and has systematically indexed a large number of these resources. Furthermore, SRS combines the indexes in a system that allows cross-database queries, simply executing the same query against all of the indexed data sets, sequentially, and reporting all of the results. This simple model is reasonably effective be-

cause there is a strong overlap in the field names and content of the various data sets. However, this system does not extend readily to data types other than sequences, and, even for sequence data, does not provide the rich query capability of SQL nor the optimization capability of DiscoveryLink. DiscoveryLink could be used by SRS as a richer means of integrating the various sources, or DiscoveryLink could wrap SRS as a single source of sequence data.

Solutions such as SYNERGY** [48] and Tripos** [49] provide useful access to diverse life sciences data sources and analysis applications through a domain neutral object framework. SYNERGY has been constructed as a network of object-based components built on Java** and CORBA** (Common Object Request Broker Architecture**) technologies, while Tripos relies on CORBA for its distributed framework, or MetaLayer. As with DiscoveryLink, both SYNERGY and Tripos can integrate heterogeneous data sources and programs, and have no built-in data types or analyses. Instead, the kinds of data upon which the framework can operate and the analyses available for these data types are discovered by the program at run time. However, these systems' focus is on building applications from objects rather than on queries and query optimization. As a result, this type of object layer is complementary to the DiscoveryLink technology, and when used in conjunction with it can provide a powerful solution.

Other solutions including SeqStore**, [50] Gene Expression Datamart [48] and those provided by Incyte, [51] have taken a data warehousing or data mart approach to provide fast access to preintegrated data (a data mart is a "small" warehouse designed to support a specific activity). From a performance perspective, we believe the optimization technology for federated data sources described here makes the replication of data and consequent maintenance unnecessary for most applications. Nevertheless, there are situations in which, because of semantic incompatibilities or slow networks, it is preferable to warehouse some of the data and then join this warehouse with other sources using a federated system such as DiscoveryLink.

Compared to other database middleware systems such as TSIMMIS, [8] DISCO, [18] Pegasus, [6] DIOM, [7] and HERMES, [19] DiscoveryLink is unique in supporting the full SQL3 language across diverse sources. Because these systems are all research prototypes, they have not yet focused on the features needed to make a system industrial strength. Nimble Technology's Nimble Integration Suite [52] is an XML (Extensible Markup Language)-based integration product that uses XML-QL [53] as the integration language. Although also based on advanced database research (from the University of Washington), this technology is relatively new and unproven compared to relational query processing. Other commercial database middleware systems provide query across multiple relational sources (for example, DataJoiner [54] from IBM and similar products from Oracle, [55] and Sybase [56]). DiscoveryLink is unique among these systems in its support for writing new wrappers, its capability to create wrappers for nonrelational sources, its capability to add new sources dynamically, and, with the exception of DataJoiner, in its optimization capabilities.

Status and future work

In this paper we have described IBM's DiscoveryLink offering. DiscoveryLink allows users to query data that may be physically stored in many disparate, specialized data stores as if all those data were colocated in a single virtual database. Queries against these data may exploit all of the power of SQL, regardless of how much or how little SQL function the various data sources provide. In addition, queries may employ any additional functionality provided by individual data stores, allowing users the best of both the SQL and the specialized data source worlds. A sophisticated query optimization facility ensures that the query is executed as efficiently as possible. This optimizer will become even more discerning in the next version of DiscoveryLink. We have offered evidence that often DiscoveryLink does not add significant overhead to single-source queries, and we have summarized work showing that the optimizer technologies of both the current and the future versions are necessary and are capable of choosing good query execution plans.

DiscoveryLink is a new offering, but it is based on a fusion of well-tested technologies such as DB2 UDB, DB2 DataJoiner, and the Garlic research project. Both DB2 UDB (originally DB2 C/S) and DB2 DataJoiner have been available as products since the early 1990s, and have been used by thousands of customers over the past decade. The Garlic project began in 1994, and much of its technology was developed as the result of joint studies with customers, including an early study with Merck Pharmaceuticals. DiscoveryLink's extensible wrapper architecture and the forthcoming version of the optimizer derive from Garlic. As part of Garlic, we successfully built and

queried wrappers for a diverse set of data sources, including two relational database systems (DB2 and Oracle), a patent server stored in Lotus Notes**, searchable sites on the World Wide Web (including a database of business listings and a hotel guide), and specialized search engines for collections of images, chemical structures, and text.

Currently, we are working on building up a portfolio of wrappers specific to the life sciences industry. In addition to key relational data sources such as Oracle and Microsoft's SQL Server**,[57] we are writing wrappers for common genomic sources such as SWISS-PROT[13] and GenBank,[12] chemical structure sources such as Daylight,[34] and general sources of interest to the industry such as Lotus Notes, Microsoft Excel**, flat files, and text management systems. We are also working with key industry vendors to wrap the data sources they supply. While we will continue to create wrappers as quickly as possible, we anticipate that most installations will require one or more new wrappers to be created, due to the sheer number of data sources that exist, and the fact that many potential users have their own proprietary sources as well. Hence we are training a staff of wrapper writers who will be able to build new wrappers as part of the DiscoveryLink software and services offering.

Of course, there are plenty of areas in which further research is needed. For the query engine, key topics are the exploitation of parallelism to enhance performance, and richer support for modeling of object features in foreign data sources. There is also a need for additional tools and facilities that enhance the basic DiscoveryLink offering. We have done some preliminary work on a system for data annotation that provides a rich model of annotations, while exploiting the DiscoveryLink engine to allow querying of both annotations and data separately and in together. We are also building a tool to help users create mappings between source data and a target, integrated schema[58,59] to ease the burden of view definition and reconciliation of schemas and data that plagues today's system administrators. We hope that as DiscoveryLink matures it will serve as a basis for more advanced solutions that will use its ability to integrate access to data from multiple sources to pull real information out of the oceans of data in which life sciences researchers are currently drowning.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Oracle Corporation, Eli Lilly and Co., Sun Microsystems, Inc., Microsoft Corporation, McNeil Consumer Healthcare, Netgenics, Inc., Tripos Associates, Inc., Object Management Group, Genetics Computer Group, Inc., or Lotus Development Corporation.

Cited references and notes

1. K. Howard, "The Bioinformatics Gold Rush," *Scientific American*, July 2000.
2. See http://www.ncbi.nlm.nih.gov/BLAST/fasta.html.
3. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic Local Alignment Search Tool," *Journal of Molecular Biology* **215**, No. 3, 403–410 (1990).
4. See http://www.nlm.nih.gov/medlineplus/medline.html.
5. See http://www.idbs.co.uk/.
6. M.-C. Shan, R. Ahmed, J. Davis, W. Du, and W. Kent, "Pegasus: A Heterogeneous Information Management System," W. Kim, Editor, *Modern Database Systems*, Chapter 32, ACM Press (Addison-Wesley Publishing Co.), Reading, MA (1994).
7. L. Liu and C. Pu, "The Distributed Interoperable Object Model and Its Application to Large-Scale Interoperable Database Systems," *Proceedings of the Fourth International Conference on Information and Knowledge Management*, ACM, New York (1995).
8. Y. Papakonstantinou, H. Garcia-Molina, and J. Widom, "Object Exchange Across Heterogeneous Information Sources," *Proceedings of the IEEE Conference on Data Engineering*, Taipei, Taiwan, IEEE, New York (1995), pp. 251–260.
9. M. Carey et al., "Towards Heterogeneous Multimedia Information Systems," *Proceedings of the Fifth International Workshop on Research Issues in Data Engineering*, Taipei, Taiwan, March 1995, IEEE, New York (1995).
10. L. M. Haas, P. Kodali, J. E. Rice, P. M. Schwarz, and W. C. Swope, "Integrating Life Sciences Data—with a Little Garlic," *Proceedings of the IEEE International Symposium on Bio-Informatics and Biomedical Engineering*, IEEE, New York (2000).
11. T. Studt, "Next Generation Database Management Tools," *R&D Magazine, Drug Discovery & Development*, January 2000, http://www.dddmag.com/feats/0001net.htm.
12. See chapter 2 of Reference 39.
13. A. Bairoch and R. Apweiler, "The SWISS-PROT Protein Sequence Database and Its Supplement TrEMBL in 2000," *Nucleic Acids Research* **28**, No. 1, 45–48 (2000).
14. A. Dalby, J. Nourse, W. D. Hounshell, A. Gushurst, D. Grier, B. Leland, and J. Laufer, "Description of Several Chemical Structure File Formats Used by Computer Programs Developed at Molecular Design Limited," *Journal of Chemical Information and Computer Sciences* **32**, No. 3, 244–255 (1992).
15. D. Weininger, "SMILES," *Journal of Chemical Information and Computer Sciences* **28**, No. 1, 31–36 (1988).
16. See http://www.genome.ad.jp/kegg/.
17. D. Chamberlin, *A Complete Guide to DB2 Universal Database,* Morgan Kaufmann Publishers, San Francisco, CA (1998).
18. A. Tomasic, L. Raschid, and P. Valduriez, "Scaling Heterogeneous Databases and the Design of DISCO," *Proceedings of the 16th International Conference on Distributed Computer Systems*, Hong Kong, 1996, IEEE, New York (1996).
19. S. Adali, K. Candan, Y. Papakonstantinou, and V. S. Subrahmanian, "Query Caching and Optimization in Distributed Mediator Systems," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, June 1996, ACM, New York (1996), pp. 137–148.

20. K. Kulkarni, "Object-Oriented Extensions in SQL3: A Status Report," *Proceedings of the ACM SIGMOD Conference on Management of Data*, Minneapolis, May 1994, ACM, New York (1994).
21. M. Tork Roth and P. Schwarz, "Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources," *Proceedings of the Conference on Very Large Data Bases (VLDB)*, Athens, Greece, August 1997, ACM, New York (1997).
22. P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, "Access Path Selection in a Relational Database Management System," *Proceedings of the ACM SIGMOD Conference on Management of Data*, Boston, MA, May 1979, ACM, New York (1979), pp. 23–34.
23. J. Hellerstein and M. Stonebraker, "Predicate Migration: Optimizing Queries with Expensive Predicates," *Proceedings of the ACM SIGMOD Conference on Management of Data*, Washington, DC, May 1993, ACM, New York (1993), pp. 267–276.
24. S. Chaudhuri and L. Gravano, "Optimizing Queries over Multimedia Repositories," *Proceedings of the ACM SIGMOD Conference on Management of Data*, Montreal, Canada, June 1996, ACM, New York (1996), pp. 91–102.
25. Actually, the optimizer normally ignores pairs when there is no predicate connecting them (e.g., Compounds and Proteins in this query), because typically these "cross-products" do not make good plans.
26. L. Shapiro, "Join Processing in Database Systems with Large Main Memories," *ACM Transactions on Database Systems* **11**, No. 3, 239–264 (1986).
27. The host variable :KETANSERIN_MOL is presumed to contain an appropriate representation of the ketanserin structure, perhaps as generated by a sketching tool.
28. In this paper, we represent query fragments in SQL; the actual wrapper interface will use an equivalent data structure that does not require parsing by the wrapper.
29. M. Tork Roth, F. Ozcan, and L. Haas, "Cost Models DO Matter: Providing Cost Information for Diverse Data Sources in a Federated System," *Proceedings of the Conference on Very Large Data Bases (VLDB)*, Edinburgh, Scotland, September 1999, ACM, New York (1999).
30. We did not analyze why this occurred, because it was not unexpected: our experience with DataJoiner over the years is that it usually introduces little overhead, and occasionally can run a transaction faster than the native data source. There are several possible reasons why this script might run faster through DiscoveryLink, among them, DiscoveryLink's superior optimizer and the fact that it ran on a separate machine, hence could apply more hardware to the problem. In this instance, the result is probably due to the DiscoveryLink engine exploiting the resources of its separate machine, because the four queries in script four are fairly simple, and with one exception leave little room for optimization.
31. F. Rezende and K. Hergula, "The Heterogeneity Problem and Middleware Technology: Experiences with and Performance of Database Gateways," *Proceedings of the Conference on Very Large Data Bases (VLDB)*, New York, August 1998, ACM, New York (1998).
32. L. Haas, D. Kossmann, E. Wimmers, and J. Yang, "Optimizing Queries Across Diverse Data Sources," *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB)*, Athens, Greece, August 1997, Morgan Kaufmann Publishers, San Francisco, CA (1997).
33. See http://www.mdli.com.
34. See http://www.daylight.com.
35. Y. Martin, "Comparison of Programs That Calculate Octanol-Water logp Using Starlist," *Proceedings of the 12th Annual Daylight User Group Meeting*, Daylight Chemical Information Systems (1997).
36. G. Klopman and H. S. Rosenkranz, "Toxicity Estimation by Chemical Substructure Analysis: The Tox ii Program," *Toxicology Letters* **79**, 145–155 (1995).
37. R. C. Glen and A. W. R. Payne, "A Genetic Algorithm for the Automated Generation of Molecules Within Constraints," *Journal of Computer-Aided Molecular Design* **9**, No. 2, 181–202 (1995).
38. I. D. Kuntz, "Structure-Based Strategies for Drug Design and Discovery," *Science* **257**, 1078–1082 (1992).
39. *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins*, A. D. Baxevanis and B. F. F. Ouellette, Editors, Wiley-Liss, New York (1998).
40. See http://www.msi.com.
41. See http://www.oxfordmolecular.com.
42. See http://www.genomica.com.
43. S. Davidson, C. Overton, V. Tannen, and L. Wong, "BioKleisli: A Digital Library for Biomedical Researchers," *International Journal of Digital Libraries* **1**, No. 1, 36–53 (1997).
44. I-M. A. Chen, A. S. Kosky, V. M. Markowitz, and E. Szeto, "Constructing and Maintaining Scientific Database Views in the Framework of the Object-Protocol Model," *Proceedings of the Ninth International Conference on Scientific and Statistical Database Management*, IEEE, New York (1997), pp. 237–248.
45. N. W. Paton, R. Stevens, P. Baker, C. A. Goble, S. Bechhofer, and A. Brass, "Query Processing in the TAMBIS Bioinformatics Source Integration System," *Proceedings of the 11th International Conference on Scientific and Statistical Database Management*, IEEE, New York (1999), pp. 138–147.
46. P. Carter, T. Coupaye, D. Kreil, and T. Etzold, "SRS: Analyzing and Using Data from Heterogeneous Textual Databanks," S. Letovsky, Editor, *Bioinformatics: Databases and Systems*, Chapter 18, Kluwer Academic Press (1998).
47. T. Etzold and P. Argos, "SRS: An Indexing and Retrieval Tool for Flat File Data Libraries," *Computer Applications in the Biosciences* **9**, 49–57 (1993).
48. See http://www.netgenics.com/.
49. See http://www.tripos.com.
50. See http://www.gcg.com/.
51. See http://www.incyte.com/.
52. See http://www.nimble.com/.
53. See http://www.w3.org/TR/NOTE-xml-ql.
54. See http://www.software.ibm.com/data/datajoiner/.
55. See http://www.oracle.com/.
56. See http://www.sybase.com/.
57. See http://www.microsoft.com.
58. L. M. Haas, R. J. Miller, B. Niswonger, M. Tork Roth, P. M. Schwarz, and E. L. Wimmers, "Transforming Heterogeneous Data with Database Middleware: Beyond Integration," *IEEE Data Engineering Bulletin* **22**, No. 1, 31–36 (1999).
59. R. J. Miller, L. M. Haas, and M. A. Hernandez, "Schema Mapping as Query Discovery," *Proceedings of the Conference on Very Large Data Bases (VLDB)*, Cairo, Egypt, September 2000, ACM, New York (2000).

Accepted for publication November 17, 2000.

Laura M. Haas *IBM Software Group, Silicon Valley Laboratory, 555 Bailey Road, San Jose, California 95141 (electronic mail: lmhaas@us.ibm.com).* Dr. Haas is manager of DB2 Query Compiler and Life Sciences Development for IBM. She was formerly the manager of Data Integration Research at IBM's Almaden

Research Center. She received her Ph.D. degree in 1981 from the University of Texas at Austin. Since joining IBM, she has worked on distributed relational database (R*), extensible query processing (Starburst), and the integration of heterogeneous data (Garlic and Clio). Technology from these projects forms the basis of the DB2 UDB query processor and enables access to heterogeneous data sources in the latest releases of DB2. Dr. Haas was vice-chair of ACM SIGMOD from 1989 to 1997. She has served as an associate editor of the ACM journal *Transactions on Database Systems*, as program chair of the 1998 ACM SIGMOD conference, and was recently elected to the VLDB Board of Trustees. She has received IBM awards for Outstanding Technical Achievement and Outstanding Contributions, and a YWCA Tribute to Women in Industry (TWIN) award. Her research interests include schema mapping, data integration, and query processing.

Peter M. Schwarz *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (electronic mail: schwarz@almaden.ibm.com).* Dr. Schwarz is a research staff member in the Middleware Systems and Technology Department of IBM's Almaden Research Center. He received his Ph.D. degree from Carnegie-Mellon University in 1984, working with Alfred Spector on concurrency control and recovery for typed objects. At IBM, Dr. Schwarz has worked on algorithms for log-based recovery in database systems and middleware for integrating heterogeneous data sources. His interests also include object-oriented programming languages and type systems.

Prasad Kodali *3rd Millennium Inc., 125 Cambridge Park Drive, Cambridge, Massachusetts 02140 (electronic mail: pkodali@3rdmill.com).* Dr. Kodali is Informatics Project Lead at 3rd Millennium, where he is involved in developing advanced informatics solutions for pharmaceutical and biotechnology companies. He was previously the product manager of data integration products at NetGenics, Inc. He received his Ph.D. degree in computational chemistry from Pennsylvania State University. His research interests include data integration in drug discovery, computational algorithms, computer-assisted drug design, and life science informatics.

Elon Kotlar *Aventis Pharmaceuticals, Bridgewater, New Jersey 08807 (electronic mail: elon.kotlar@aventis.com).* Mr. Kotlar is a global project leader in the Drug Innovation and Approval Information Solutions organization at Aventis Pharmaceuticals. He received his B.A. in the biological basis of behavior from the University of Pennsylvania in 1996 and then worked in diagnostic radiology research at the Hospital of the University of Pennsylvania. At Aventis Pharmaceuticals he has worked to provide scientists with solutions to integrate data across the drug discovery process.

Julia E. Rice *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (electronic mail: julia@almaden.com.* Dr. Rice is a research staff member and manager in IBM Research at the Almaden Research Center. She joined the computational chemistry team at IBM in 1988 and worked on understanding and predicting the nonlinear optical properties of organic molecules. Following that, she led the teams that developed the quantum chemistry and architecture components of the computational chemistry software package, Mulliken. More recently, her interests have expanded to include database and in particular informatics issues in life sciences. Research in Dr. Rice's group currently includes 3-D molecular similarity matching of databases of flexible molecules, as well as the use of annotations in life sciences. Her group has played a key role in bridging the gap between scientists and the use of database technology in the DiscoveryLink project. Dr. Rice received her Ph.D. in theoretical chemistry from the University of Cambridge, England. She spent a postdoctoral year at the University of California, Berkeley and then held a research fellowship at Newnham College, Cambridge before joining IBM. Dr. Rice was named as one of the 750 most highly cited chemists worldwide for the period 1981–1997 (ISI survey). She is currently a member of the Executive Committee of the Physical Chemistry section of the American Chemical Society. Dr. Rice was awarded the YWCA Tribute to Women in Industry Award (TWIN) in 1999.

William C. Swope *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (electronic mail: swope@almaden.ibm.com).* Dr. Swope is a research staff member currently helping with the Blue Gene Protein Science project. He started his career in IBM at IBM Instruments, Inc., an IBM subsidiary that developed scientific instrumentation, where he worked in an advanced processor design group. He also worked for six years at the IBM Scientific Center in Palo Alto, California, where he helped IBM customers develop software for numerically intensive scientific applications. In 1992 Dr. Swope joined the IBM Research Division at Almaden, where he has been involved in software development for computational chemistry applications and in technical data management for petroleum and life sciences applications. He obtained his undergraduate degree in chemistry and physics from Harvard University and his Ph.D. degree in quantum chemistry from the University of California at Berkeley. He then performed postdoctoral research on the statistical mechanics of condensed phases in the chemistry department at Stanford University. He maintains a number of scientific relationships and collaborations with academic and commercial scientists involved in the life sciences and, in particular, drug development.