A Searching and Reporting System for Relational Databases Using a Graph-Based Metadata Representation

Robin Hewitt*

Hewitt Consulting, San Diego, California 92103

Alberto Gobbi* and Man-Ling Lee

Anadys Pharmaceuticals, Inc., 9050 Camino Santa Fe, San Diego, California 92121

Received February 18, 2005

Relational databases are the current standard for storing and retrieving data in the pharmaceutical and biotech industries. However, retrieving data from a relational database requires specialized knowledge of the database schema and of the SQL query language. At Anadys, we have developed an easy-to-use system for searching and reporting data in a relational database to support our drug discovery project teams. This system is fast and flexible and allows users to access all data without having to write SQL queries. This paper presents the hierarchical, graph-based metadata representation and SQL-construction methods that, together, are the basis of this system's capabilities.

1. INTRODUCTION

Today's drug discovery teams must deal effectively with increasingly complex datasets. The shift to the "fail early" paradigm in drug discovery results in an increased diversity of biological assays early in the process. In addition to having high activity against the biological target, compounds must have acceptable absorption, distribution, metabolic, excretion, and toxicological properties. A drug discovery project team can only work productively if these data are captured efficiently[1] and made readily accessible in a format that enables the team to draw conclusions and validate new ideas quickly.

Relational databases are by far the most widely used database systems today because of their ability to represent complex data using a normalized data model. Since the introduction of cartridges that enable chemical structure searching,[2] relational databases have also become the database system of choice for handling scientific data in the pharmaceutical and biotech industries. However, chemists and biologists are not trained in relational database theory. They're usually unfamiliar with the concepts of a relational data model and with the SQL query language. Nevertheless, they need the ability to search by any criteria and to create custom reports.

Drug discovery teams have a compound-centric, hierarchical view of their data. Relational databases, in contrast, have a nonhierarchical model in which all data tables are equal. Metadata are essential to translate the normalized data model to a more compound-centric view while retaining the relationships described in the relational data model. These metadata are the link between the data fields perceived by users and the data columns in relational database tables.

In the general information science literature, metadata concepts, techniques, and standards are widely published.[3] However, publications describing the specific metadata concepts and techniques needed to support pharmaceutical research are rare. We found only one paper[9] detailing a metadata design and methods for searching a database containing pharmaceutical research data. Generally, papers that describe information systems for pharmaceutical research present application architectures and user interface designs without describing their metadata format and its internal usage.[10−18] Nevertheless, it is clear that all such systems must implement methods for searching and reporting data and that these methods must be supported by some form of metadata.[19]

For example, Daylight's Thor database uses a metadata language[20] to describe database entries. The full hierarchical relationship of the data is, however, an implicit property of each record and is not fully described by the built-in metadata. MDL's ISIS Host uses text files called Hviews[21] to describe how to access data in one or more databases and tables. The information in an Hview includes table names or SELECT statements for retrieving and querying data. The Hview also includes descriptions on how to link tables to create a specific hierarchy of data fields. Hviews are used to create targeted views of a subset of data in one or more databases.

In contrast, our approach uses a generic rather than a targeted description of all data in a relational database system. Metadata in this approach are represented as a directed graph (digraph), which readily captures the users' hierarchical, compound-centric view. We implemented two engines that use these metadata. (1) The search engine is responsible for generating SQL queries that return lists of hits that fulfill user-defined constraints. (2) The report engine generates SQL queries to retrieve the data of interest for the compounds in a hit list. This two-step approach results in high overall performance, because each engine constructs queries tuned to its assigned task. The interface between the two engines

* Corresponding authors. E-mail: rhewitt@acm.org (R.H.), agobbi@anadyspharma.com (A.G.).

10.1021/ci050062e CCC: $30.25
Published on Web 05/19/2005

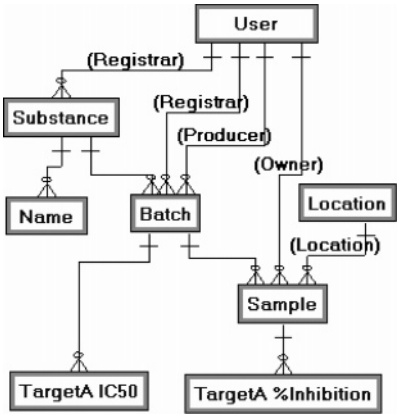

Figure 1. Example of an ERD. Some relationships are specified in parentheses.

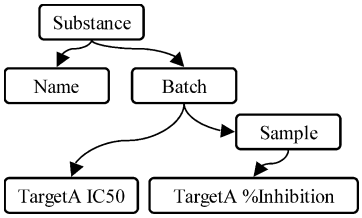

Figure 2. Example of a DRGraph. The arrows indicate the direction of the relationships between DRNodes. Information from the User table is included in the Substance, Batch, and Sample DRNodes. Information from the Location table is included in the Sample DRNode.

is the hit list. In the current implementation, only substance IDs are passed from one engine to the other. The hit list could, however, just as easily be a list of batch or sample IDs.

2. METADATA DESCRIPTION

Our metadata model is object-oriented and hierarchical. We call the metadata objects DRGraph, DRNode, and DRField. The DRGraph represents the relationships between data objects. DRNodes represent the objects themselves, and DRFields represent attributes of these objects. This approach to metadata is completely generic. It can be applied to any relational data model.

In a relational database, tables represent object definitions. Each row in a table corresponds to one (real world) object. Each object is identified by its unique primary key. "One to many" (1:N) relationships are represented by parent−foreign key pairs. The primary key value of the parent object is included as a foreign key in each row of its child table to establish this relationship. A "many to many" (N:N) relationship requires the use of an intermediate table that contains foreign keys referencing the two related tables. An entity relationship diagram (ERD) is the established technique for visualizing relational data models. The ERD is usually represented as shown in Figure 1. Tables in the ERD are connected by lines indicating their relationships.

The DRGraph (Figure 2) is similar to an ERD in that it expresses cardinality relationships, but the DRGraph uses a different convention than an ERD. The DRGraph is a digraph in which arcs travel from 1 to N in the cardinality relationship. If two nodes in the DRGraph have an N:N relationship, these nodes will have two arcs connecting them, one in each direction. For the applications described in this paper, there was no need to support N:N relationships. The DRGraph in this case is a tree, reflecting the compound-centric view required for drug discovery. The root node of the DRGraph is the Substance node and substance IDs are used to identify hits in a hit list.

The nodes in the DRGraph are DRNodes. In contrast to an ERD, in which the objects are the actual database tables, DRNodes are not literally equivalent to database tables. Each DRNode represents a set of related data fields. For example, the Sample DRNode contains fields that are in the Sample table (such as amount and barcode) and also fields from the Location and User tables (sample location and owner). Conceptually, DRNodes are more similar to an SQL view than to a table. The fields in a DRNode are DRFields.

These metadata objects (DRGraph, DRNode, and DRField) mediate between the relational database and applications that use the database. Each level provides both a physical database representation, used to produce SQL queries, and a conceptual data representation, used in user interfaces. In our implementation, the majority of the information is contained in the DRNodes. The DRGraph is a very thin object, consisting of nothing more than the DRNodes and the parent−child relationships between them.

The DRNode contains SQL fragments and additional querying information. Queries for reporting and searching require information from multiple DRNodes. These queries are constructed from multiple subqueries, one for each relevant DRNode. We begin by describing the database-specific information in a single DRNode, then we show how this information is used to construct SQL subqueries, and finally we describe how subqueries are combined to create complete SQL queries for searching and reporting. For maintenance ease, the metadata are defined in XML format. Figure 3 shows the XML definition of one DRNode, Compound Batch.

The XML element name for a DRNode is **node** (line 1). The subelements **tables** (line 7) and **fields** (line 17) contain the information needed to form one subquery. The **tables** element lists all relational tables from which the DRNode may need to retrieve data. The USER_INFORMATION table (line 10) is only included in a subquery if it is actually needed, that is, if user information fields are specifically part of the subquery. The attribute **alwaysUsed** has the value "n" to indicate that the USER_INFORMATION table may or may not be used in a particular subquery for the Batch node. When the USER_INFORMATION table is included in the subquery, the SQL in the **constraint** element (line 11) will also be included in the subquery.

The **fields** element (line 17) contains all the DRFields available inside this DRNode. Each **field** element (lines 18, 21, and 26) specifies the database column names, prefixed by table alias, for accessing a specific DRField. The Chemist field (line 21) requires the USER_INFORMATION table. This is indicated by the **additionalTable** element (line 24). The person data type defines a field based on two database columns: first_name and last_name (lines 22−23). The data type of a field is specified using the **type** attribute (line 21). Currently, 11 different data types are supported (for example, string, number, chemical structure, date, and person).

The structure of the DRNodes' XML representation readily lends itself to pivoting database tables to create views well-suited to structure−activity relationship (SAR) analysis. For

```
 1:<node id="b" parentNodeID="su" treeLabel="Batch">
 2:<joinKeys>
 3:   <primaryKey value="b.batch_id"/>
 4:   <parentPrimaryKey value="b.fk_substance_id"/>
 5:   <hitListKey value="b.fk_substance_id"/>
 6:</joinKeys>
 7:<tables>
 8:   <table alias="b" name="BATCH" alwaysUsed="y">
 9:   </table>
10:   <table alias="u" name="USER_INFORMATION" alwaysUsed="n">
11:      <constraint sql="b.fk_scientist = u.user_id" />
12:   </table>
13:
14:   ●●●
15:
16:</tables>
17:<fields>
18:   <field id="notebook" treeLabel="Notebook" type="String">
19:      <column sql='b.notebook_reference'/>
20:   </field>
21:   <field id="chemist" treeLabel="Chemist" type="Person">
22:      <column sql='u.first_name'/>
23:      <column sql='u.last_name'/>
24:      <additionalTable alias="u"/>
25:   </field>
26:   <field id="regDate" treeLabel="Registration Date" type="Date">
27:      <column sql='b.Registration_Date'/>
28:   </field>
29:
30:   ●●●
31:
32:</fields>
33:</node>
```

Figure 3. XML description of a DRNode. The first column contains line numbers. In lines 14 and 30, ●●● represents omitted definitions.

example, from a database table containing columns for inhibition and compound concentration, we can create multiple DRNodes, each with a DRField for inhibition at a specific concentration. Each view can easily be created by adding a single **constraint** element to a common DRNode definition.

To combine subqueries, additional information is needed. This information is contained in the **joinKeys** element (line 2). The hitlistKey element (line 5) specifies the database column that returns IDs for a hit list. Report queries make use of the directional hierarchy of the DRGraph, and this information is contained in the **primaryKey** and **parentPrimaryKey** elements (lines 3−4).

The DRNode definition also includes user interface information, contained in the **treeLabel** attributes. In addition, the XML representation includes some ID attributes. The combination of the field ID and the corresponding node ID uniquely identify a DRField in the DRGraph. The applications use these IDs as hash keys.

3. SQL CONSTRUCTION

Figure 4 shows the structure of a subquery created from the Compound Batch DRNode that was presented in the previous section. As shown in this figure, each engine produces slightly different subqueries. Some core SQL elements are always present in subqueries, regardless of user input. Other elements will appear when a user searches on certain fields or reports these fields. For example, the field notebook_reference (line 5) will only appear in a subquery for this DRNode if the user requests it as part of the report output. Similarly, the constraint on notebook_reference (line 19) will only appear if the user includes this field in a search. When the actual data for a DRField is in an auxiliary table, a join must be added. For example, for the chemist information to be reported (lines 6−7) or searched (line 18), the USER_INFORMATION table must be joined (lines 12 and 16). If aggregation (such as averaging) is required in a report, aggregation functions are applied to the columns in the SELECT clause (lines 2−9) and a GROUP BY clause is added after line 21.

The output from the search engine is a hit list, a list of substance IDs (line 2). The report engine accepts a hit list as input and restricts the returned rows to hits in that hit list (lines 14, 20, and 21).

The subqueries are building blocks for constructing the full search and report SQL queries. The search engine uses a consistent field alias, hitlistKey (line 2), to link all subqueries. The report engine uses the primaryKey (line 3) and parentPrimaryKey (line 4) to link subqueries. Because the linking process differs between the two engines, we present each procedure separately below.

Constructing the Full Search Query. The search engine is capable of constructing a SQL query that includes an unlimited number of user-specified constraints. To construct a full SQL query, the search engine creates subqueries for each DRNode that has at least one user constraint. It then links these subqueries using the SQL INTERSECT operator. Because each subquery is relatively simple and Oracle is efficient in executing INTERSECT operations, the overall query runs very quickly.

In principle, arbitrarily complex queries could be built with this approach by using the UNION, MINUS, and INTERSECT SQL operators. However, to keep the user interface simple we decided to allow only searches based on a list of constrained fields; hence, only the INTERSECT operator is needed. Users can store hit lists, and we provide a separate module for set-logic operations, such as OR or MINUS. Thus, users can create queries of any complexity in a simple, step-by-step fashion.

Constructing Full Queries for Reporting. The report engine is capable of constructing report queries for an unlimited number of user-specified output fields. These

	Engine	Core	SQL
1:	SR	X	**SELECT**
2:	SR	X	b.fk_substance_id hitListKey,
3:	R	X	b.batch_id primaryKey,
4:	R	X	b.fk_substance_id parentPrimaryKey,
5:	R		b.notebook_reference notebook,
6:	R		u.first_name ||' '
7:	R		|| u.last_name chemist,
8:	R		b.Registration_Date regDate,
9:	R		●●●
10:	SR	X	**FROM**
11:	SR	X	BATCH b,
12:	SR		USER_INFORMATION u,
13:	SR		●●●
14:	R	X	HITLIST_TABLE hl
15:	SR	X	**WHERE**
16:	SR		b.fk_scientist_id = u.user_id
17:	SR		**and** ●●●
18:	S		**and** u.first_name = 'Jim'
19:	S		**and** b.notebook_reference like '123-%'
20:	R	X	**and** b.fk_substance_id = hl.hit_id
21:	R	X	**and** hl.hit_number **between** 1 **and** 10

Figure 4. Example of an SQL subquery created from the DRNode defined in Figure 3. The "Engine" column indicates that a line is used for searching (S), reporting (R), or both (SR). These engines will always use the lines marked as "Core" when producing subqueries for this DRNode.

Report Column	Report Tree Node
Structure, AnadysID	Substance
Notebook Reference	Batch
	Sample
IC50	TargetA IC50
Concentration, % Inhibition	TargetA %Inhibition

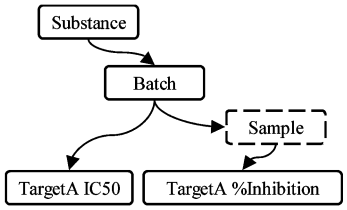

Figure 5. Example of a report and the corresponding report tree. The report contains the columns Structure, Anadys ID, Notebook Reference, IC50, Concentration, and % Inhibition. The report tree contains the DRNodes for the corresponding DRFields. In addition, the Sample node is added to complete the tree.

output fields can be any DRField at any level of the DRGraph. Because report output can come from multiple levels, merely retrieving data for a hit list is insufficient. It is also necessary to retain the hierarchical relationships between data from DRNodes at various levels. For example, if a report includes several batches with notebook references and each batch has IC_{50} values, the report must display which notebook reference corresponds to which IC_{50}. Additionally, the report engine supports aggregation. That is, users can ask for average, minimum, maximum (and so forth) values of DRFields that have a 1:N relationship to substance ID.

We illustrate the specifics of constructing full report queries using the example shown in Figure 5. In this example, the user has specified fields from four DRNodes—Substance, Batch, Target A IC50, and Target A %Inhibition. To construct full report queries, the report engine does the following:

1. List all DRFields in the report (see Figure 5).

2. Build a list of unique DRNodes from all nonaggregated fields in the report.

3. If the DRNodes from step 2 do not form a connected tree, add missing parent DRNodes until the set of DRNodes forms a connected tree. For this example, the DRNode for Sample is added. The resulting tree is a subgraph of the DRGraph, and we refer to it as the report tree.

4. Use depth-first search to find all paths from the root node (Substance) to each leaf node in the report tree (Target A IC50 and Target A %Inhibition).

5. Identify the longest path. We call this the main branch. In this example, the main branch is Substance > Batch > Sample > Target A %Inhibition.

6. For all remaining paths, find the attachment point to the main branch. Each of these remaining paths is called a side branch. For this example, there is one side branch (Substance > Batch > Target A IC50), and it attaches to the main branch at the Batch node.

7. Create a SQL statement for each branch, as described below.

The SQL statement for the main branch is shown in Figure 6. It consists of SELECT, FROM, WHERE, and ORDER BY clauses. The FROM clause includes subqueries for each aggregated and nonaggregated DRNode in the main branch as in-line subqueries. These in-line subqueries are joined by outer joins in the WHERE clause. The SELECT clause includes the hit list key, the primary keys of all nonaggregated DRNodes in the main branch, and all data columns from the in-line subqueries. Rows are ordered first by the hit list key then by primary keys in descending hierarchical order. The SQL construction for side branches is identical, except that side branches have no aggregated fields. Linking the rows of the main and side branches can be accomplished quickly because the rows from each are sorted by the same fields.

4. USER INTERFACE

QueryBuilder and ReportDesigner. Following the paradigm of separating searching from reporting, we implemented two user interfaces: QueryBuilder for users to specify searches and ReportDesigner for users to create customized reports. Users can combine any search with any report.

The user interfaces for searching and reporting have commonalities. In both the QueryBuilder and ReportDesigner

```
 1:SELECT
 2:    s1.hitListKey,
 3:    s1.primaryKey pk1,
 4:    s2.primaryKey pk2,
 5:    s3.primaryKey pk3,
 6:    s4.primaryKey pk4,
 7:    s1.smiles,
 8:    s1.compound_id,
 9:    s2.Notebook,
10:    s3.amount,
11:    s4.Inhibition,
12:    avg(s5.IC50),
13:    ●●●
14:FROM
15:    (sub-query for substance node) s1,
16:    (sub-query for batch node) s2,
17:    (sub-query for sample node) s3,
18:    (sub-query for "TargetA %Inhibition") s4,
19:    (aggregated sub-queries for any nodes
20:                  with aggregated fields) s5
21:WHERE
22:       s1.primaryKey = s2.parentPrimaryKey (+)
23:   and s2.primaryKey = s3.parentPrimaryKey (+)
24:   and s3.primaryKey = s4.parentPrimaryKey (+)
25:   and s5.fk_substance_id = s1.substance_id(+)
26:ORDER BY s1.hitListKey,
27:         s1.primaryKey, s2.primaryKey,
28:         s3.primaryKey, s4.primaryKey
```

Figure 6. SQL statement for the main branch of the report tree. Line numbers are displayed in the first column. Each subquery in lines 15−20 is formed as described at the start of Section 3.

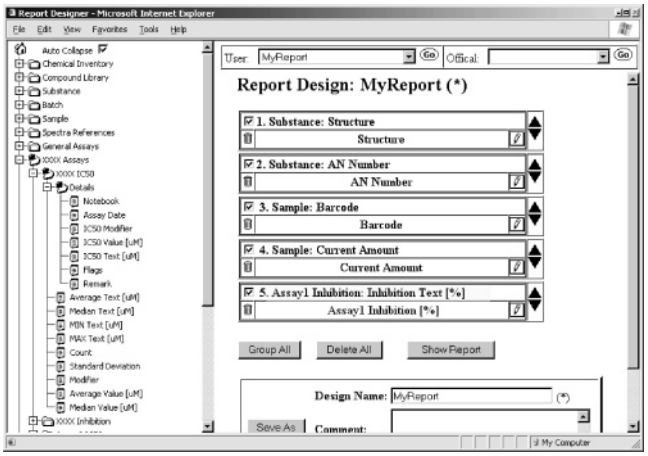

Figure 7. ReportDesigner. The left side contains a tree view of all data fields. The right side displays a list of all selected report fields. Users can remove (trashcan icon) or edit (pencil icon) their selected fields.

windows, users browse an Explorer-style tree to locate data fields (Figure 7). Each data field in the tree corresponds to one DRField, and each folder with data fields corresponds to a DRNode. By clicking on a field label, that field is added to a list of currently selected fields.

In QueryBuilder, selected fields represent search constraints (Figure 8). Each field has an associated data type, defined as part of the DRField metadata (see Figure 3, lines 18, 21, and 26). The available constraints for each field depend on that field's data type. When users click the pencil icon, the appropriate constraint editor for that data type appears. A numeric field can be constrained to be within a given range or to be equal to a value. Text fields allow for exact, wildcard, and alphabetic range constraints. For a structure field, users can specify a substructure, exact structure, or similarity search. To enter the query structure the user can use either the ChemDraw plug in[22] or the JME editor.[23]

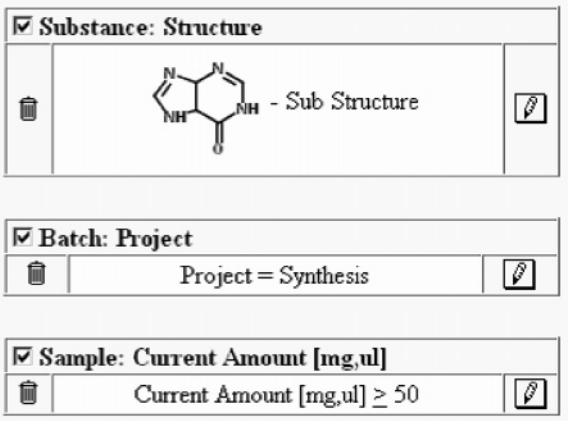

Figure 8. QueryBuilder constraint list.

In ReportDesigner, selected fields correspond to columns in the report output. The order of fields in the ReportDesigner corresponds to the column order in the output. Users can change this order using the arrow icons shown in Figure 7. In ReportDesigner, clicking the pencil icon enables users to change formatting options such as column heading, font and background colors, and numeric precision. In addition, fields with 1:N relationships to substances may be aggregated. For numeric fields, the default aggregation is averaging, and for text fields, the default is to show the result that is alphabetically first. When aggregated, these fields have one value per substance in the report output.

Users can save their report designs and make them available to other users under a name of their choice.

Report Output. Report data are output as a table. Table reports are particularly useful for SAR analysis because users can view multiple compounds and compare the results of various assays. The hierarchical relationship of data in the report output is displayed by breaking table rows into subrows as shown in Figure 9. For reports that have a single hierarchy, an expanded view is available (Figure 10). In this view, data values from higher levels in the hierarchy are repeated for each value at the lowest level. Some reports have more than one hierarchy. These reports cannot be displayed in expanded format because they have more than one leaf node. When reports have only one hierarchy, users can switch from one output format to the other within the report output window. Both formats can be exported to Microsoft Excel for further data analysis and printing.

5. PERFORMANCE

Both QueryBuilder and ReportDesigner have been running for over a year at Anadys. During this time, we have monitored real time performance. Our database contains several hundred thousand substances and batches and several million samples. Performance statistics for SQL queries against this database are given in Table 1.

In general, performance depends on hardware configuration, database design, and the methods used to query and retrieve data. Our hardware configuration has been described previously[1]. Our database design includes one denormalization to improve query performance. Most assay result tables include an additional foreign key referencing the Substance table even when their direct parent is Batch or Sample.[24] Appropriate column indices are also important for good

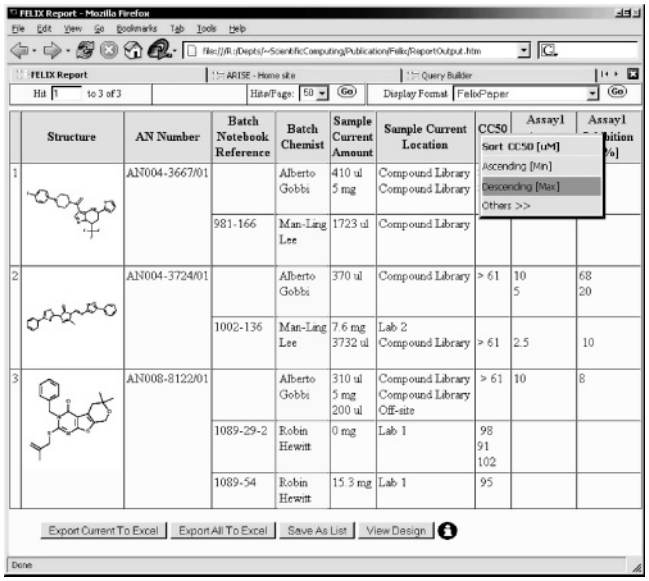

Figure 9. Example of a report output. The navigation bar on the top allows the user to navigate the report pages and to switch between report designs. Users can sort the report output by clicking on a column heading. Buttons on the bottom of the page allow users to export the report, save the hit list, or modify the report design. (Structures and data have been randomized to protect proprietary information.)

Substance Fields	Batch Fields		
Anadys ID	Chemist	Notebook Reference	CC50
AN001-1111	Joe	123-11	45
			46
	Mike	124-21	48
AN001-1112	Mike	124-33	87

Anadys ID	Chemist	Notebook Reference	CC50
AN001-1111	Joe	123-11	45
AN001-1111	Joe	123-11	46
AN001-1111	Mike	124-21	48
AN001-1112	Mike	124-33	87

Figure 10. Viewing options. Users can toggle the report output between a hierarchical format and an expanded format (described in the text).

Table 1. SQL Performance

time to execute SQL	% of search queries	% of report queries
<1 s	86	92
<2 s	92	96
<5 s	95	98
≥5 s	5	2

performance. Ongoing performance monitoring helps us identify needed indices.

Efficiently accessing hierarchical data from the flat representation imposed by relational databases is inherently difficult.[25,26] By using a hit list to represent search results, we avoid having to deal with hierarchy in the query engine. Because each DRNode subquery returns substance IDs, these can be combined using set-logic operators, which are efficiently implemented in Oracle.

In the report engine, hierarchy cannot be avoided. Our approach to SQL construction for report queries, described in Section 3, is designed to maximize performance by producing only the minimal number of queries required to retrieve all data while preserving hierarchical relationships.[27]

To maximize performance when users produce a report from a search, search queries execute in a separate thread. As soon as the first hit is returned from Oracle, a hit list object is created and passed to the report engine. Reporting begins immediately while the remaining hits are being added to the hit list.

6. CONCLUSION

In this paper, we have described the implementation of a search and retrieval application that enables users who are not database experts to access a complex database. We employ a metadata description to translate the normalized relational data model, optimized for data storage, into a compound-centric, hierarchical representation (the DRGraph). We have implemented two user-intuitive graphical user interfaces, QueryBuilder and ReportDesigner. QueryBuilder allows users to construct complex queries with constraints on any combination of fields in the database. ReportDesigner allows users to create customized reports that may include any data column or derived field. SQL for searching and reporting is fast, allowing users to quickly test new hypotheses. Hence, there is no need to set up a data warehouse.

The metadata description currently defines 109 DRNodes with a total of 880 DRFields. Subqueries for some of these DRNodes contain as many as 12 tables. Thus, queries for both search and reporting sometimes become large, involving many joins. Even so, performance to date has been very satisfactory. If, however, the SQL for a DRNode subquery ever became so complex as to create a performance problem, the metadata description in the DRNode could then be used to create a denormalized database table to maintain high performance. This step, if ever needed, can be automated.

The searching and reporting applications are flexible enough to use with any data model. Users do not need to ask information specialists for database views or reports, because they create their own queries and report layouts using QueryBuilder and ReportDesigner. Maintenance of this application is minimal. Hence, both scientists and information specialists can devote their time and skills to solving other problems.

ACKNOWLEDGMENT

We thank the people involved with the many Open Source projects who have made available the software modules used in our system.[28−35] It would not have been possible to implement so much so quickly without this tremendous resource.

REFERENCES AND NOTES

(1) Gobbi, A.; Funeriu, S.; Ioannou, J.; Wang, J.; Lee, M.; Palmer, C.; Bamford, B.; Hewitt, R. Process-Driven Information Management System at a Biotech Company: Concept and Implementation. *J. Chem. Inf. Comput. Sci.* **2004**, *44*, 964−975.

(2) Companies providing structure search cartridges include Daylight Inc., Mission Viejo, CA, U. S. A.; Elsevier MDL, San Leandro, CA, U. S. A.; Accelrys, San Diego, CA, U. S. A.; IDBS, Guildford, UK.; ChemAxon, Budapest, Hungary; CambridgeSoft, Cambridge, MA, U. S. A.

(3) See, for example, refs 4−8.

(4) Object Management Group; *Common Warehouse Metamodel (CWM) Specification*, version 1.0; 2001. http://www.omg.org/cwm.

(5) Poole, J.; Chang, D.; Tolbert, D.; Mellor, D. *Common Warehouse Metamodel: An Introduction to the Standard for Data Warehouse Integration*; Wiley: New York, 2001.
(6) Tannenbaum, A. *Metadata Solutions: Using Metamodels, Repositories, XML, and Enterprise Portals to Generate Information on Demand*; Addison-Wesley Professional: Reading, MA, 2001.
(7) *Metadata in Practice*; Hillmann, D. I., Westbrooks, E. L., Eds.; American Library Association: Chicago, IL, 2004.
(8) Marco, D.; Jennings, M. *Universal Meta Data Models*; Wiley: New York, 2004.
(9) Nadkarni, P. M.; Brandt, C. Data Extraction and Ad Hoc Query of an Entity-Attribute-Value Database. *J. Am. Med. Inf. Assoc.* **1998**, *5*, 511−527.
(10) Hara, K.; Miyazaki, T.; Nishi, T.; Iwamoto, K.; Ideshita, T. Relational database system RIQS (relational information query system). *Basic Software Dev. Div., Jpn.* **1985**, *77*, 55−62.
(11) Hripcsak, G. IAIMS Architecture. *J. Am. Inf. Assoc.* **1997**, *4*, Suppl. S20−20.
(12) Ihnenfeldt, W.-D.; Vogt, J. H.; Bienfait, F.; Oellien, F.; Nicklaus, M. C. Enhanced CACTVS Browser of the Open NCI Database. *J. Chem. Inf. Comput. Sci.* **2002**, *42*, 46−57.
(13) Walker, M. J.; Hull, R. D.; Singh, S. B. CKB—The Compound Knowledge Base: A Text Based Chemical Search System. *J. Chem. Inf. Comput. Sci.* **2002**, *42*, 1293−1295.
(14) Cohen, J. Informatics challenges in HTS data retrieval and mining. *PharmaChem* **2002**, *1*, 22−24.
(15) Smith, P. M. Federated databases: The next level. *Abstracts of Papers*, 224th ACS National Meeting, Boston, MA, 2002; CINF-086.
(16) Fisk, J. M.; Mutalik, P.; Levin, F. W.; Erdos, J.; Taylor, C.; Nadkarni, P. Integrating Query of Relational and Textual Data in Clinical Databases: A Case Study. *J. Am. Med. Inf. Assoc.* **2003**, *10*, 21−38.
(17) Kasprzyk, A.; Keefe, D.; Smedley, D.; London, D.; Spooner, W.; Melsopp, C.; Hammond, M.; Rocca-Sierra, P.; Cox, T.; Birney, E. EnsMart: A generic system for fast and flexible access to biological data. *Genome Res.* **2004**, *14*, 160−169.
(18) Atipat, R.; Rotstein, S. H. AQUIRE: ARQULE's United Repository and Information Exchange System. *Abstracts of Papers*, 228th ACS National Meeting, Philadelphia, PA, 2004; CINF-041.
(19) Rojnuckarin, A.; Gschwend, D. A.; Rotstein, S. H.; Hartsough, D. S. ArQiologist: An Integrated Decision Support Tool for Lead Optimization. *J. Chem. Inf. Model.* **2005**, *45*, 2−9.
(20) The data type definitions are stored in the Thor data type database. James, C. A.; Weininger, D.; Delany, J. *Daylight Installation and Administration Guide*; Daylight CIS Inc.: Mission Viejo, CA, 2004. http://www.daylight.com/dayhtml/doc/admin/admin.thorcon.html#RTFToC101 (accessed: November 20, 2004).
(21) *ISIS/Host, Hview Developer's Guide*; Elsevier MDL: San Leandro, CA.
(22) CambridgeSoft, Cambridge, MA.
(23) Ertl, P. JME Molecular Editor. http://www.molinspiration.com/jme/.
(24) Denormalization can result in inconsistencies in the database. However, an API used for any changes on the Substance and Batch tables prevents this from happening.
(25) Bohannon, P.; Freire, J.; Roy, P, Siméon, J. From XML Schema to Relations: A Cost-Based Approach to XML Storage. *Proceedings of the 18th International Conference on Data Engineering*, San Jose, CA, February 26−March 1, 2002, 64.
(26) Krishnamurthy, R.; Kaushik, R.; Naughton, J. XML-to-SQL Query Translation Literature: The State of the Art and Open Problems. *XML Symposium (XSym)* **2003**, 1−18.
(27) It is not possible to include subqueries of two DRNodes with the same parent in one SQL statement. If the two child nodes were joined by an outer join to the parent node, the result would be a duplication of rows due to the Cartesian product of the child rows. If the two child nodes are joined by an equijoin, some rows are missed if either of the child subqueries does not return any rows. Therefore, the number of SQL statements cannot be further reduced.
(28) The Apache Jakarta Project, Apache Tomcat. http://jakarta.apache.org/tomcat/index.html.
(29) The Apache Jakarta Project. http://jakarta.apache.org.
(30) Heterogeneous Modeling and Design, UC Berkley, EECS, Ptolemy Project, Ptplot. http://ptolemy.eecs.berkeley.edu/java/ptplot/.
(31) The Apache XML Project, Batik SVG Toolkit. http://xml.apache.org/batik/.
(32) The FreeHep Java Library. http://java.freehep.org/index.html.
(33) The Apache XML Project. http://xml.apache.org/.
(34) The Apache XML Project, Xalan-Java. http://xml.apache.org/xalan-j/index.html.
(35) JDOM. http://www.jdom.org/.

CI050062E