Accessing The Columbia Clinical Repository

Stephen B. Johnson, Ph.D., George Hripcsak, M.D.,
Joan Chen, M.S., Paul Clayton, Ph.D.

Center for Medical Informatics
Columbia Presbyterian Medical Center
New York, NY 10032

The Columbia Clinical Repository is the foundation of the Clinical Information System at the Columbia Presbyterian Medical Center (CPMC). The Repository is implemented as a relational database on an IBM mainframe, using a generic design that employs a small number of tables. Client applications on remote platforms send and receive data through Database Access Modules (DAMs), which support the HL7 protocol, while applications on the mainframe manipulate data through DAMs supporting a locally defined "query template". Implementation using static (compiled) SQL is compared to dynamic (ad hoc) SQL in terms of efficiency and flexibility.

INTRODUCTION

Clinical databases are the foundation of clinical information systems. Systems such as HELP [1], TMR [2], RMRS [3], and STOR [4] have revealed many important issues about the management of clinical data and the ways in which clinical applications need to access that data. The Columbia Clinical Repository benefitted greatly from the experience of these systems, particularly the HELP system. In a manner similar to these other systems, The Repository is designed to:

- support transactions for one patient at a time, in an efficient manner;

- store controlled vocabulary (data elements) as defined in the Medical Entities Dictionary (MED) [5];

- enable Medical Logic Modules (MLMs) to monitor patient events as they occur in the CIS and notify health care personnel as necessary [6].

The Repository is interesting in that it is implemented as a relational database using a generic design that employs a very small number of tables [7]. This approach allows new data elements to be stored in the database simply by defining them in the MED, rather than adding new columns to tables. This model is similar to a relational model developed independently for the HELP system [8]; however, unlike the HELP relational design, the Repository tables are normalized. The relational design is generated from a conceptual schema (expressed in the Entity-Relation formalism) which is based on a model of clinical events [9].

Additional information about the Repository is maintained in a "metadatabase" [10], a relational database that contains the MED, the collection of Medical Logic Modules, and information about how messages (data transactions) are routed to ancillary systems. One of the most important consequences of representing the MED in a relational form is that patient data can be queried by a class of medical entity in an efficient manner. For example, one can query whether a patient has any drug order in the class "antibiotics", without having to explicitly specify each such medication.

The Repository is intended to serve as a complete electronic medical record, and has the ability to store both in-patient and out-patient data in a longitudinal manner (spanning all encounters with the hospital). The coded data that is currently available includes demographics, laboratory results, and pharmacy orders. There is also a great wealth of narrative clinical data (radiology, pathology, cardiology, operative reports, discharge summaries, etc.).

The most challenging issue confronting the developers of the Repository is providing access to clinical data both for applications running on the same platform (e.g., the Event Monitor that executes Medical Logic Modules), and for client applications making requests from other platforms on the network. The remainder of the paper describes the different techniques used to provide access to

0195-4210/94/$5.00

data, and discusses the advantages and disadvantages of the various approaches.

METHODS

The Columbia Clinical Repository is implemented on an IBM mainframe using the DB2 database management system. Clinical applications running locally on the same platform (under the CICS system) include a home-grown results review application and the Event Monitor. Applications on remote platforms include ancillary systems uploading clinical data (laboratory, pharmacy, cardiology, radiology, pathology, etc.), a primary care information system, and a resident sign-out system.

DB2 provides two forms of SQL for accessing tables: static SQL and dynamic SQL. Static SQL is used when a program containing SQL is compiled: the SQL statements are parsed, a plan of access is determined for each statement (e.g., what indexes will be used), and the SQL is replaced by appropriate procedure calls. Dynamic SQL is invoked when a program passes a SQL statement (represented as a string) to DB2 at execution time: the statement is parsed and a plan of access is determined "on the fly". This method incurs some overhead, since the DB2 catalogue (itself a DB2 database) must be queried by the DBMS to determine the best means of access.

SQL cannot be embedded directly in Medical Logic Modules and compiled into static SQL because they are executed by an interpreter (in our implementation). A similar limitation exists for the results review application which was developed using IBM's PCS/ADS system. We also do not currently possess the database and network software that would enable applications on other platforms to use static SQL that is bound to DB2 tables on the mainframe.

A satisfactory solution to these obstacles was reached through development of Data Access Modules (DAMs). These are mainframe programs, written in the PL/I language that can issue static SQL. On the mainframe, the Event Monitor interpreter and the results review application can call a DAM, pass parameters to specify a database request, and then receive data back through other parameters.

Remote applications cannot call mainframe DAMs directly, but instead make use of a utility that resides on an intermediate UNIX machine (an IBM RS 6000). This utility acts as a "clinical data mediator", carrying out the communication with the mainframe DAMs through IBM's Advanced Program to Program Communications (APPC) network protocol.

Since SQL could not be used as the standard of access, a suitable interface had to be defined. For mainframe applications, which need to query the Repository, a generic "query template" was developed as the interface between these applications and each of the DAMs [11]. When invoked, a DAM examines the parameters of the query template and chooses from a fixed set of static SQL statements to carry out the query. Data retrieved by the DAM is passed back through the template.

Data exchange standards were considered essential for applications residing on other platforms, and Health Level 7 (HL7) was chosen as the protocol. For mainframe applications, the query template was preferred over HL7 as it is somewhat simpler for application developers and MLM authors to understand.

The functions performed by data access modules can be summarized as follows:

1. Request Handling: the data request message submitted by the client process is parsed, and the various parameters in the message are checked for validity.

2. Data Conversion: data elements in the request message are translated into a standard coded form as defined by the MED.

3. Database Operations: the SQL statements required to update or query the database are executed.

4. Notification: the Event Monitor is notified about the type of database action that just

occurred. MLMs may get triggered as a result of a database update.

5. Response Generation: a response message is generated indicating whether the request has been carried out successfully or not, and any data retrieved from the database is returned as part of the response.

Adding new features to data access modules can be time consuming. To enable developers building applications on remote platforms to query the Repository more directly, a DAM was made available that supports dynamic SQL. This DAM works in a similar manner to those described above, except that the client application passes a SQL statement embedded in an HL7 message. The SQL statement is executed dynamically by DB2, and the results are returned to the client application in an HL7 format.

Some extensions to the HL7 standard were required by added "Z" segments to the HL7 message: a "ZQL" segment is passed along with standard HL7 segments for a query. In the response, a "ZMH" segment returns the number of rows retrieved, the number of database columns of which the data is composed, and the names of the columns. The data itself is returned in multiple "ZMO" segments, which each item of data (a cell of a relational table) occupying one segment.

RESULTS

Timing studies of the production system showed that review of laboratory data using a mainframe program takes 0.375 seconds per transaction (total elapsed time), on average. This is similar to the average elapsed time of 0.195 seconds previously obtained for queries executed by MLMs [12]. Upload of laboratory data requires 0.685 seconds per transaction (elapsed time), on average.

Queries submitted by remote applications were also compared. Pharmacy orders and laboratory results for a specified patients were retrieved, using the DAM that executes static SQL, and the DAM that uses dynamic SQL. The average elapsed times in seconds were as follows:

	Dynamic	Static
Order	1.58	0.98
Result	0.97	0.81

These results suggest that dynamic SQL is only slightly more expensive than static. It is interesting to note that the lab query was faster, despite involving a relational join.

DISCUSSION

HL7 is a useful standard of access for applications residing on platforms remote from the mainframe. The chief benefit of this architecture is that client applications are made independent of the implementation of the Clinical Repository. As a result, application programmers do not have to use SQL, or understand the design of the relational database.

Application developers also benefit from the other services provided by DAMs. They can use the data elements with which they are familiar, since the DAM performs the conversions. In addition, the DAM automatically notifies the Event Monitor, freeing applications from this responsibility.

Finally, the architecture enables the implementation of the Repository to be changed, (e.g., to an object-oriented DBMS), with a minimal impact on applications.

HL7 has been found to be best suited for interfaces to ancillaries that are well understood, and that do not have changing requirements, such as routine uploads and downloads. The syntax is very cumbersome for human users, e.g., developers of review programs, and authors of MLMs.

The query templates developed for mainframe applications, such as the Event Monitor, are clinically oriented, and have a marginally better syntax than HL7 for users. However, this interface is not a standard, and requires that a second set of DAMs be maintained in addition to the HL7 DAMs.

The biggest drawback in both these approaches is the software maintenance of the DAMs that interpret the client request (HL7 messages or query template) and then execute the appropriate SQL. This is consistent with the earlier finding that data access is the most costly aspect of developing Medical Logic Modules, in terms of coding, maintenance, and execution time [12].

A large part of this maintenance burden is due to the inflexibility of static SQL. In DB2 SQL, a table cannot be expressed using a variable, thus a separate SQL statement must be coded for each table in the database, and for each useful join (when information must be combined from two tables). While the number of tables is small, and the possible joins very limited, the DAMs are still rather complex.

Another limitation concerns the use of lists. For example, the client may wish to retrieve values for a given list of observations. Since DB2 does not permit the use of an array as a host variable, the list must be coded as a collection of individual host variables:

:LISTLEN = 0 OR <column> IN (:VAR1, :VAR2, ..., :VARN)

The maximum list size (N) must be determined ahead of time, and dummy values must be placed in unused variables if less than N items are requested. The test (LISTLEN = 0) must be used to insure that the condition is true when the list is empty.

While dynamic SQL is less efficient, it has the advantage of great flexibility. A skilled user can perform any desired database transaction. This method requires that the user possess complete knowledge of the design of the relational tables in the Repository. While the generic design has provided efficient access, it has proven to be difficult for users to understand.

An interesting consequence of the generic design of the Repository is that certain common queries are much simpler (and more efficient) to express in a procedural manner than using SQL alone. For example, the most frequently used form of clinical query requests the most recent N values of a given observation (e.g., the last 3 serum sodium levels). While it is possible to express this in SQL, the query is extremely complex and inefficient.

The use of procedural code permits a simple (and efficient) solution: the transaction need only retrieve N rows meeting the criteria (perform just N fetches). This is easily accomplished by embedding the SQL query within procedural code containing a loop executed N times:

```
EXEC SQL
DECLARE c CURSOR FOR
SELECT ...
FROM ...
WHERE ...

EXEC SQL OPEN c;

DO FOR I = 1 TO N;
        FETCH c INTO :structure
        [Add data from structure to
        response]
END;

EXEC SQL close c;
```

The generic design also makes certain views of clinical data very difficult to express in SQL. For example, laboratory tests (e.g., levels of sodium, potassium, and chloride) are not stored as individual columns of a table. To construct a view of lab values with test names as columns requires an SQL statement with as many joins as there are tests. However, the view can easily be constructed by a DAM, using a procedural loop like the one show above.

CONCLUSION

The database component of any Clinical Information System will need to provide access to clinical applications residing on the same platform and on remote platforms as hospital computing environments become increasingly distributed. The approach taken at Columbia has been to encapsulate important types of queries in Data Access Modules, which insulate clients from the

database structure, provide efficient access to data, and construct views of data that are too costly to define using SQL by itself.

These modules are effective for routine uploads and downloads but are complex and hard to modify to meet the needs of application developers in a timely manner.

These findings indicate an important direction for further study: data access modules supporting dynamic SQL are appropriate when a high level, flexible means of accessing data is required (e.g., in developing Medical Logic Modules). However, these DAMs must support a view of the database which users can easily understand, and provide those temporal operations that are not easily expressed in SQL (e.g., "the 3 most recent values of ...").

Acknowledgements

Support for this project was provided by the IBM Corporation.

References

[1] Pryor TA. The HELP medical record system. MD Computing, 1988;5(5):22-33.

[2] Stead WW, Hammond WE. Computer-Based Medical Records: The Centerpiece of MD Computing, 1988;5(5):48-62.

[3] McDonald CJ, Blevins L, Tierney WM, Martin DK. The Regenstrief Medical Record. MD Computing, 1988;5(5):34-47.

[4] Whiting-O'Keefe QE, Whiting A, Henke J. The STOR Clinical Information System MD Computing, 1988;5(5):34-47.

[5] Cimino JJ, Hripcsak G, Johnson SB, Clayton PD. Designing an introspective, multi-purpose controlled medical vocabulary. In: Kingsland LC, editor. Proceedings of the Thirteenth Annual Symposium on Computer Applications in Medical Care; 1989 November 5-8; Washington. Washington: IEEE Computer Society Press, 1989:513-518.

[6] George Hripcsak, James J. Cimino, Stephen B. Johnson, Paul D. Clayton. The Columbia-Presbyterian Medical Center decision-support system as a model for implementing the Arden Syntax. In: Clayton PD, editor. Proceedings of the Fifteenth Annual Symposium on Computer Applications in Medical Care; 1991 Nov 17-20; Washington, D.C. New York: IEEE Computer Society Press, 1991: ?.

[7] Friedman C, Hripcsak G, Johnson SB, Cimino JJ, Clayton PD. A generalized relational scheme for an integrated clinical patient database. In: Miller RA, editor. Proceedings of the Fourteenth Annual Symposium on Computer Applications in Medical Care; 1990 November 4-7; Washington. Washington: IEEE Computer Society Press, 1990: 335-339.

[8] Huff SM, Berthelsen CL, Pryor TA, Dudley AS. Evaluation of an SQL model of the HELP patient database. In: Clayton PD, editor. Proceedings of the Fifteenth Annual Symposium on Computer Applications in Medical Care; 1991 November 17-20; Washington. New York: McGraw Hill, 1992:386-90.

[9] Johnson SB, Friedman C, Cimino JJ, Hripcsak G, Clayton PD. Conceptual data model for a central patient database. In: Clayton PD, editor. Proceedings of the Fifteenth Annual Symposium on Computer Applications in Medical Care; 1991 November 17-20; Washington. New York: McGraw Hill, 1992: 381-385.

[10] Johnson SB, Cimino JJ, Friedman C, Hripcsak G, Clayton PD. Using metadata to integrate medical knowledge in a clinical information system. In: Miller RA, editor. Proceedings of the Fourteenth Annual Symposium on Computer Applications in Medical Care; 1990 November 4-7; Washington. Washington: IEEE Computer Society Press, 1990: 340-344.

[11] Hripcsak G, Johnson SB, Sideli RV, Clayton PD. Using Data Access Modules for Legacy Databases. Accessing the electronic medical record using HL7. In: Rindfleisch TC, editor. Proceedings of the 1994 AMIA Spring Congress; 1994 May 4-7; San Francisco. Washington, D.C.: AMIA, 1994.

[12] Hripcsak G, Johnson SB, Clayton PD. Desperately Seeking Data: Knowledge Base-Database Links. In: Safran C, ed. Proceedings of the Seventeenth Annual Symposium on Computer Applications in Medical Care, 1993 Oct 3-Nov 3; Washington (DC). New York: McGraw Hill, 1993:639-42.