

Extended SQL for Manipulating Clinical Warehouse Data

Stephen B. Johnson, PhD

Department of Medical Informatics
Columbia University

Damianos Chatziantoniou, PhD

Department of Computer Science
Stevens Institute of Technology

Health care institutions are beginning to collect large amounts of clinical data through patient care applications. Clinical data warehouses make these data available for complex analysis across patient records, benefiting administrative reporting, patient care and clinical research. Data gathered for patient care purposes are difficult to manipulate for analytic tasks; the schema presents conceptual difficulties for the analyst, and many queries perform poorly. An extension to SQL is presented that enables the analyst to designate groups of rows. These groups can then be manipulated and aggregated in various ways to solve a number of useful analytic problems. The extended SQL is concise and runs in linear time, while standard SQL requires multiple statements with polynomial performance. The extensions are extremely powerful for performing aggregations on large amounts of data, which is useful in clinical data mining applications.

INTRODUCTION

There has been a continual increase in the amount of computerization in health care, in clinical, administrative, and ancillary settings. Following trends in other industries, health care institutions are beginning to realize the value of pooling these data into large, integrated databases (data warehouses), and discovering knowledge within them (data mining) [1-4]. Analysis of health care data across patient records has tremendous potential to benefit patient care, administration and clinical research.

Relational databases developed for patient care have benefited from a generic design in which there are a small number of general-purpose tables [Johnson 1996]. While this design facilitates flexibility and high performance for patient care tasks, it is difficult to use in decision support applications [Johnson 1994] and in data analysis tasks [Nadkarni].

In data mining applications, the preparation of data for analysis constitutes at least 80% of the total effort [Adriaans]. This is because most data mining tools require that data be organized into a single table with specific, named columns. Typically, clinical data are distributed across several tables, and have generic columns with coded values. The transformation of raw warehouse data requires considerable intellectual effort, as well as intensive computer processing.

The principal difficulty of generic database designs is the use of coded values instead of specific column names. This results in a schema that is conceptually difficult for users to manipulate and which performs

very poorly for queries involving multiple patient records. A classic example of a generic schema is shown in Table 1, in which lab data are identified by codes NA, K, and CL.

MRN	Date	Test	Value
123	03-12-1999	NA	135
123	03-12-1999	K	3.7
123	03-12-1999	CL	109
123	06-01-1999	NA	140
123	06-01-1999	K	5.1
123	06-01-1999	CL	104
123	11-23-1999	NA	145
123	11-23-1999	K	4.2
123	11-23-1999	CL	110
700	02-17-1999	NA	139
700	02-17-1999	K	4.2
700	02-17-1999	CL	106
700	09-22-1999	NA	146
700	09-22-1999	K	4.1
700	09-22-1999	CL	108

Table 1: Lab data in generic schema

In contrast, a specific schema employs distinct column names for each attribute of interest. Table 2 shows the same data as Table 1, but arranged in a table in which the value of each lab test can be identified by its column name. This schema is far easier to use in decision support and data analysis applications.

MRN	DATE	NA	K	CL
123	03-12-1999	135	3.7	109
123	06-01-1999	140	5.1	104
123	11-23-1999	145	4.2	110
700	02-17-1999	139	4.2	106
700	09-22-1999	146	4.1	108

Table 2: Lab data in specific schema

One approach to working with generic database designs is to build a software layer that insulates users from the schema, and enables them to name the clinical attributes in which they are interested [Nadkarni]. While such tools can aid a certain class of users, some users will still need to work with clinical databases directly, e.g. database administrators and analysts proficient in SQL. In addition, the SQL generated by such tools will still suffer from tremendous performance problems, due to the limitations of standard SQL.

A different approach is to allow users to name the attributes in which they are interested while still using SQL. This can be accomplished if users have a way to name and manipulate horizontal sections of

relational tables, which are known as “groups”. Standard SQL currently has a limited facility to define groups. If it were possible to assign names to groups, they can be “picked up”, moved around and aggregated over.

This paper describes Extended Multi-Feature (EMF) SQL [9-10], and demonstrates its application to a number of difficult operations on clinical data. This approach yields queries that are simpler to understand and more efficient to execute.

METHODS

EMF SQL enables a user to define groups of rows in a table, and to assign names to these groups. The attributes of each group can then be used in other conditions in the query and in the “select” clause. The ability to refer to attributes of a group allows tables to be “pivoted”(described below), and provides a novel mechanism for complex aggregations.

The syntax of EMF SQL is shown in Figure 1. The clauses are the same as standard SQL, with the addition of one or more “grouping variables” in the group by clause, and the “such that” clause which defines what rows of the table are in each group.

```

select      <grouping columns>,
            <aggregate functions>
from        <table name>
where       <conditions>
group by    <grouping columns> ;
            <grouping variables>
such that <defining conditions>
having      <conditions>

```

Figure 1: Extended SQL Syntax

EMF SQL was used to construct a number of queries which occur frequently in analysis of clinical data, but which are known to be difficult to formulate in standard SQL. The queries were run on data in the Clinical Data Warehouse at Columbia-Presbyterian Medical Center, which contains 10 years worth of administrative and ancillary data.

The clinical queries that were studied were the following:

- **Pivoting:** transforming lab data in a generic schema (Table 1) to a specific schema (Table 2).
- **Flattening:** transforming nested mammography data (findings with modifiers that in turn have modifiers) in a generic schema into a specific schema with named columns.
- **Sequencing:** finding cardiology procedures in a generic table of longitudinal data, and returning a

specific schema in which rows have a particular temporal sequence.

- **Aggregation:** determining the median value of a series of lab values.

Each of these queries was expressed in both standard and extended SQL, and executed on data in the warehouse.

RESULTS

Pivoting Lab Values

The pivot operation converts particular coded values in tables (such as codes for laboratory tests) into specific, named columns, which are filled with the data of interest (such the numeric values of the tests). Any number of selected values can be pivoted in this manner. The pivot operation applies to any clinical database table that contains groups of coded attributes. The groups are often defined by having the same patient identifier and a common time, as in laboratory or medication data.

The data in Table 1 can be pivoted into the schema in Table 2 using the standard SQL shown in Figure 2. In this query there are three coded values of interest (NA, K, CL), representing three measures of these elements in patient specimens. There are several ways of constructing this query, but Figure 2 shows a straightforward method in which a view is constructed for each of the three lab tests. The fourth SQL statement joins these three views together in order to arrange the values of the lab tests into the same row as shown in Table 2. The conditions in the “where” clause are required to ensure that only lab tests from the same specimen occur together (they must have the same MRN and Date). In the general case, m lab tests require m views and an m -way join. If the lab table contains n rows, execution time is proportional to n^m (polynomial time).

An alternative approach to pivoting laboratory data is shown in Figure 3, using EMF SQL. In this query the “group by” clause specifies that laboratory tests that belong together have the same MRN and date. Within each group, three subgroups are named (x, y,z), corresponding to the three lab tests of interest (NA, K, CL). The defining conditions for each group are given in the “such that” clause. Once there is a way to name the three different rows, they can be pivoted in the “select” clause, which pulls out the values for sodium, potassium and chloride tests, respectively.

```

Create view Sodium as
select MRN, Date, Value,
from lab
where Test = "NA"

Create view Potassium as
select MRN, Date, Value,
from lab
where Test = "K"

Create view Chloride as
select MRN, Date, Value,
from lab
where Test = "CL"

select NA.MRN, NA.Date,
      NA.Value, K.Value, CL.Value
from Sodium NA, Potassium K,
      Chloride CL
where NA.mrn = K.mrn
      and NA.date = K.date
      and NA.mrn = CL.mrn
      and NA.date = CL.date

```

Figure 2: Standard SQL to pivot lab data

The EMF query in Figure 3 makes a single pass through the laboratory table. In the general case, m laboratory tests require m grouping variables and m defining conditions. If the table has n rows, processing time is proportional to $m \cdot n$ (linear time).

```

select mrn, any(x.value),
      any(y.value), any(z.value)
from lab
group by mrn, date ; x, y, z
such that
      x.mrn = mrn and x.test = "NA",
      y.mrn = mrn and y.test = "K",
      z.mrn = mrn and z.test = "CL"

```

Figure 3: Extended SQL to pivot lab data

Pivoting Nested Data

Some clinical information systems are beginning to capture data that is more complex in structure than simple numeric data. A typical example of complex clinical data is mammography findings. Each finding can have various modifiers that in turn have modifiers. Data with this structure are often called “nested”. One source of nested clinical data is natural language processing (NLP) [11]. For example, an excerpt from a mammogram report, “Possible mass right breast. No calcifications in breasts.” could be presented in tabular form as in Table 3:

MRN	Id	Parent	Attrib	Value
123	1	---	FND	mass
123	2	1	BOD	breast
123	3	2	LAT	right
123	4	1	CRT	poss
123	5	---	FND	calc
123	6	5	BOD	breast
123	7	5	CRT	no

Table 3: mammographic findings in generic schema

In this representation, each finding (FND) and modifier has a unique identifier. The “parent” column is used to indicate how modifiers are related to particular findings. For example, the certainty modifier (CRT) can indicate that the mass finding is possible, while calcifications are negative. Modifiers can have their own modifiers, e.g. the attribute laterality (LAT) modifies body location (BOD).

This tabular structure is effective for capturing the variable nature of natural language data, but is very inconvenient for analytic tasks. A more appropriate view makes each distinct finding a separate row, with specific, named columns for each possible modifier, as in Table 4:

Finding	Bodyloc	Laterality	Certainty
mass	breast	right	poss
calc	breast	---	no

Table 4: Flattened mammographic findings

Standard SQL is completely ill-equipped to deal with nested data like that shown in Table 3. Generating the flattened view in Table 4 would require five SQL statements with complex join conditions. As with the laboratory query, the query would have polynomial time requirements.

EMF SQL makes the manipulation of nested data somewhat more feasible. The query for flattening the data from Table 3 into Table 4 is shown in Figure 4. This query is similar to the query in Figure 3 used to pivot laboratory data. Here the grouping attributes are finding, certainty, body part, and laterality. As with the laboratory query, the defining conditions associate these variables with the appropriate attribute codes. The additional conditions for nesting are defined using the parent and id columns. The values of the grouping variables are “flattened” using the “select” clause.

```

Select mrn,
       any(f.value), any(c.value),
       any(b.value), any(l.value)
from mammo
group by mrn,id,attrib ; f,c,b,l
such that
  f.mrn=mrn and f.attrib="FND"
  and f.id=id,
  c.mrn=mrn and c.attrib="CRT"
  and c.parent=any(f.id),
  b.mrn=mrn and b.attrib="BOD"
  and b.parent=any(f.id),
  l.mrn=mrn and l.attrib="LAT"
  and l.parent=any(b.id)

```

Figure 4: Extended SQL for mammographic data

Analyzing Temporal Sequences

Frequently, analysts are interested in sets of clinical events that have certain temporal relationships. Often, events of different types are stored in the same table, each with a date attribute. For example, Table 5 shows the date of performance for various types procedures of particular encounters.

MRN	Encounter	Type	Date
123	03-17-1995	angiogram	03-17-1995
123	03-17-1995	angioplasty	03-18-1995
123	03-17-1995	CABG	03-25-1995
700	06-13-1995	angiogram	06-13-1995
700	06-13-1995	angioplasty	06-14-1995

Table 5: procedure data in generic schema

An analyst might be interested in encounters in which angioplasty precedes CABG. The above generic table makes this analysis very difficult. Table 5 can be “pivoted” into Table 6 in which there are specific columns for the attributes of interest. In this view it is easy to verify that patient 123 meets the criteria, while patient 700 does not.

MRN	Angioplasty	CABG
123	03-18-1995	03-25-1995
700	06-14-1995	---

Table 6: Procedure data in specific schema

Standard SQL can handle this query fairly well, but does not perform efficiently as the number of procedures and temporal conditions increases. This is another example of pivoting (like the laboratory and mammography queries above), with additional restrictions on the temporal relationships among the various events of interest.

This query is easy to express in EMF SQL, as shown in Figure 5. The grouping variables are used to identify the procedure codes of interest (angioplasty and CABG) in the “such that” clause, and to pivot the respective dates in the “select” clause.

Without the “having” clause in Figure 5, all combinations of angioplasty and CABG dates would be shown for each patient. When the temporal restriction is added, only patient 123 in Table 6 would be retrieved. The extended syntax makes it a simple matter to add additional events of interest, and additional time ordering conditions.

```

select MRN, any(x.Date), any(y.Date)
from procedure
group by mrn, type ; x,y
such that x.mrn = mrn
       and x.type = "Angioplasty",
       y.mrn = mrn and y.type = "CABG",
having any(x.date) < any(y.date)

```

Figure 5: Extended SQL for procedure data

Aggregating Lab Data

Raw clinical data are frequently inappropriate for decision making and analysis tasks. Various methods for summarizing data are required in order to draw conclusions. This is particularly true when preparing data for data mining applications.

Standard SQL provides a variety of “aggregation” functions that have a variety of common clinical applications. Examples include: counting volumes of procedures (COUNT), determining total charges for encounters (SUM), finding the minimum, maximum, and average values for laboratory tests (MIN, MAX, AVG).

There are many aggregation functions that are very difficult to express in standard SQL. A good example is determining the median value of a laboratory test for each patient. Table 7 shows the median values for sodium tests extracted from Table 1. In standard SQL this can require 11 SQL statements using 4 temporary tables [7].

MRN	NA
123	140
700	139

Table 7. Median values of sodium tests

EMF SQL provides an extremely compact solution for complex aggregation queries. A query to find the median value of sodium tests is shown in Figure 6. The grouping variable x is fixed to each sodium test. In order to determine how many lab tests there are for each patient, the grouping variable y ranges over all tests for each patient. For each test x, the group

named z contains those rows whose values are less than the value of x. By restricting the size of z to be half the size of y, we know which x is the median value.

```

select MRN, Value
from lab
where Test = "NA"
group by mrn, value ; x, y, z
such that
    x.mrn = mrn and x.value = value,
    y.mrn = mrn,
    z.mrn = mrn and
    z.value < any(x.value),
having
    count(z.value) = count(y.mrn)/2

```

Figure 6. Extended SQL for median sodium values

DISCUSSION

The above EMF queries are far simpler in structure than their corresponding implementations in standard SQL, especially for complex aggregations like finding the median value. In addition, EMF SQL tends to be far more efficient. The execution time of EMF SQL is always a linear function of the number of grouping variables (e.g., the number of types of lab tests we want to pivot), while standard SQL tends to have a polynomial function [9-10].

EMF SQL has superior performance because it makes a single pass through a table. Groups and aggregate values such as COUNT and AVG are calculated in an incremental manner as the table is scanned. This implementation is appropriate for large warehouse tables that may contain millions of rows. In standard SQL, many of these queries would require days to execute, or be computational intractable. Performance of standard SQL can be improved through use of indexes, but these are not always available for ad-hoc queries [12].

Many users of clinical warehouse data will not want to use SQL of any kind to access the database, and will prefer tools that hide the structure of the database. Even so, the database administrator will be responsible for creating the views exploited by those users. Standard SQL makes these tasks very difficult and time consuming. If SQL is generated automatically, it may have very poor performance.

CONCLUSION

The present state of the art of data analysis and data mining requires a high degree of sophistication on the part of the analyst. Tools that allow the user to get a hold on data and manipulate them flexibly encourage creativity and discovery. The ability to name horizontal slices of tables provides new "handles" by

which data can be grasped. These methods offer considerable promise when calculating complex aggregations of large amounts of data.

References

1. Berndt DJ, Hevner AR, Studnicki J. CATCH/IT: a data warehouse to support comprehensive assessment for tracking community health. Proc Amia Symp. 1998;:250-4.
2. Prather JC, Lobach DF, Goodwin LK, Hales JW, Hage ML, Hammond WE. Medical Data mining: knowledge discovery in a clinical data warehouse. AMIA Annual Fall Symposium, 1997:101-5.
3. Nigrin DJ, Kohane IS. Data mining by clinicians. Proc Amia Symp. 1998;:957-61.
4. Bellazzi R, Magni P, Larizza C, Nicolao GD, Riva A, Stefanelli M. Mining biomedical time series by combining structural analysis and temporal abstractions. Proc Amia Symp. 1998:160-4.
5. Johnson SB. Generic Data Modeling for Clinical Repositories. JAMIA, 1996:3(5).
6. Johnson SB, Hripcsak G, Chen J, Clayton PD. Accessing the Columbia Clinical Repository. Proceedings of the Eighteenth Annual Symposium on Computer Applications in Medical Care, 1994 November 5-9; Washington (DC). New York: McGraw Hill, 1994.
7. Nadkarni PM, Brandt C. Data extraction and ad hoc query of an entity-attribute-value database. J Am Med Inform Assoc. 1998 Nov-Dec;5(6):511-27.
8. Adriaans P, Zantinge. Data Mining. New York: Addison-Wesley, 1996.
9. Chatziantoniou D, Ross K. Querying multiple features of groups in relational databases. Proceedings of the 22nd Very Large Database Conference, Bombay, India, 1996.
10. Chatziantoniou D. Ad hoc OLAP: expression and evaluation. IEEE International Conf. on Data Engineering (ICDE), Sydney, 1999.
11. Friedman C, Alderson PO, Austin JH, Cimino JJ, Johnson SB. A general natural-language text processor for clinical radiology. JAMIA 1994;1(2):161-74.
12. Chatziantoniou D. The PanQ Tool and EMF SQL for complex data management. International Conference on Knowledge Discovery and Data Mining. San Diego, CA, August 15-18. Association for Computing Machinery, 1999.