SIMPLIFIED QUERY CONSTRUCTION
(Queries Made as Easy as Possible)

Brad Arshinoff, Damon Ratcliffe, Martin Saetre and Reda Alhajj*
Department of Computer Science University of Calgary Calgary, Alberta, Canada

Tansel Özyer
TOBB Ekonomi ve Teknoloji Üniversitesi, Ankara, Turkey

Keywords: Visual Query Language, SQL, Data Extraction, Relational Query Language, Form Based Query Language, HCI, Data extraction.

Abstract: QMAEP- Queries Made as Easy as Possible, is intended to be a system that greatly simplify the process of query construction for statisticians and researchers. This document is focused on the usability of the database query language and deals with visual representations of the query process, in specific the select query. Methods of integrating simple Graphical User Interfaces (GUIs) for building queries into pre-existing database forms is explored to provide users an intuitive method for query construction. This paper explores data mining as it pertains to clinical research with emphasis on simplifying the data extraction process from complex databases so as to accommodate analysis using important statistical software such as SASS, QMath and MS Excel.

1 INTRODUCTION

Database systems are common and expandable tools used to accommodate the storage and manipulation of data. On the other hand, graphical Interfaces design techniques and the field of Human Computer Interactions (HCI) have evolved so much over the last 20 years that it is no longer uncommon for users with little general computing skills to be able to master the art of database navigation (Preece et al., 2002). Unfortunately, most GUIs in today's databases are static views of data that cannot accommodate all research. To accommodate new research, new views of data must be constructed, which bring data together in previously undefined ways. Designing and constructing new GUIs, however, is a time consuming and expensive way to reorganize how data is displayed. Adding new Graphical view of data into a database requires trained database developers, which is not a feasible way to accommodate all researchers. For this reason, the most commonly created views of data are not graphical form based views, but tabular table like views which are easier to construct. Furthermore, most researchers do not want to merely reorganize their view of data from a database, but want to perform statistical software analysis on it as well (Ezekiel, 2004). The process of composing such a query from a large and complex research database structure is a tedious and often arduous task (Ezekiel, 2004). Users composing queries may be unaware of SQL and database design specific conventions such as relationships, field naming, primary keys, constraints and types. Also, filtering criteria entry is common (Celko, 2004) and causes errors (Show et al., 1993). Most query utilities offer little support in the way of solving logic errors in filtering expressions (Siau, 1998). The goal of QMAEP is to set these researchers free from design and development specifics and allow them to concentrate on their research. This paper explores the development of the QMAEP system and the algorithms and research that have made it possible.

2 RELATED WORK

Little research has been put forward to user-database interaction research (Owei et al., 2002). This is mainly due to the low priority assigned to such research as most research put towards the designing

*Reda Alhajj is also affiliated with Department of Computer Science, Global University, Beirut, Lebanon

Arshinoff B., Ratcliffe D., Saetre M., Alhajj R. and Özyer T. (2007).
SIMPLIFIED QUERY CONSTRUCTION - (Queries Made as Easy as Possible).
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - DISI*, pages 125-130
DOI: 10.5220/0002398601250130

of new database architectures. A paper by Keng Siau (Siau, 1998) deals with Object Relationship (OR) modeling to explore the relationships within databaseVORQL (visual object relational query language). It adopts the natural view that the real world consists of objects and relationships. Shortcomings of this paper include the problem with overflow of information and screen clutter problems confusing users. It also requires users to understand QBE (query by example) table structure, which is no longer used in practice; with the QMAEP system it is hoped a new table structure which is simpler and easier to use than QBE can be offered. Similarly it also requires users to have at least some understanding of the low level query's possible. Again the intention of the QMAEP system is to improve on this idea by creating a GUI to deal with Queries at the most generic User level- working through interfaces already present in the database, rather than confusing spider web of objects already connected (as seen in Siau (Siau, 1998) and apparent in the MS ACCESS visual database model). The QMAEP Select Query Tool was developed in the GUI type relating to database forms. The implementation of forms comes from the apparent advantages to using them as seen in Doan's paper (Doan et al., 1995). GUI or 'form' based interaction offers many advantages such as the ability to provide default values easily for all elements and easy error checking on the values entered textually. Also reduction of cognitive load on users by seeing only applicable values, which is beneficial for consistency and appropriate data (Ezekiel, 2004).

3 OVERVIEW OF THE QMAEP APPROACH

The QMAEP system has been successfully integrated into an existing clinical research database for Stroke studies belonging to researchers at a medical university in Turkey. This database is based on collecting strict, detailed information on volunteer stroke patients in an effort to enhance stroke research. In fact, in an effort to avoid costly omissions in data collection that could result in an inability to corroborate future research, an immensely large volume of data is collected on every patient. Much of the data is collected in a quasi-duplicate fashion, i.e., from the perspective of more than one physician or at multiple time points. Querying such data was previously particularly troublesome. Because there are multiple records relating to a single patient in such tables, when queried the result is a cross-product of related records meaning duplicated rows of the same patient with varying values for some fields and repeated values for others. Such a format, beyond being difficult to read for human beings, is difficult to analyze with statistical software.

Though the potential research implications of gathering a greater quantity of data then is immediately required are constructive and substantial, the immediate effect of gathering the excess information has actually been, according to some involved researchers, a hindrance on short term accomplishment. Narrowing down the pertinent information for a particular research topic has become something of a chore and physicians and statisticians alike were becoming impatient with the clumsy MS Access query editor they were using to construct queries to gather their data.

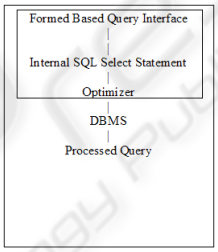


Figure 1: QMAEP position in a database framework.

3.1 Qmaep's Position in Database Framework

QMAEP can be viewed as a new layer for processing data retrieval which is referred to as "Formed Based Query Interface". It lies in a level just above the query optimizer in the database framework depicted in Figure 1. The QMAEP system itself is composed of several components designed to allow translation of a users actions on a form-based query composer into syntactically and logically correct SQL. It allows mapping from a displayed value in a form to a value data field in a database table, and finally to the creation of a new field in a query result. This requires a large degree of dynamic processing of a database schema. Under the QMAEP system of data retrieval, the users are completely abstracted from any low level representation. In terms of data mining, this tool is a rather simple method of retrieving the appropriate information.

3.2 Query Context Fields and Query Context Tables

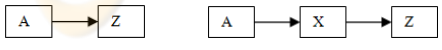
In QMAEP terminology, the group of fields to which all other fields associate is called the query context-field/s. The source table from which the query-context field/s are taken is called the query context-table. All fields in a query that are not context-fields are known as attribute fields as they can be seen as in someway describing the context fields. A table in the database schema which contains only attribute fields is known as a *attribute-table.*

For simplicity in query creation, all queries created with QMAEP must contain a context field. When a new query is created, the first field added to the query becomes the new context field. Since by the definition given, all other fields in the query are relating to and in some manner describing the context-field, a workable means of joining tables in the database is attained. All tables are joined to the context table in some manner. Because all context field values should be displayed in the result independently of whether there is a corresponding record in one or more of the attribute tables, the join mechanism used is INNER JOIN operation off the context table.

The following section will approach the task of joining tables containing attribute fields in detail and discuss the issues that must be overcome.

4 LINKING TABLES

An application database has a set of tables and a set of relations connecting the tables in a manner defined by the database schema. Any two tables from the schema that have a set of links that connect them, either directly or indirectly through other tables, are said to be *related*. Tables that do not have any set of direct or indirect links connecting them are said to be *unrelated*. Because a set of unrelated tables have no meaningful correspondence to one another, this section will concentrate on how to bring data fields from two related tables. This approach can then be generalized to bringing together *n* tables where one of the tables is defined to be the *query-context* table.



(a) (b)

Figure 2: Table A and Z have a)Direct b)Indirect Relationships.

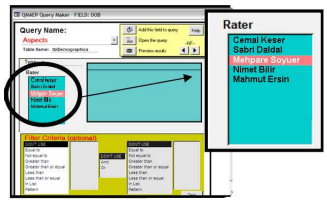


Figure 3: Dynamically creating list boxes to represent non-linking key fields.

Two tables, table A and table Z, have a *direct* relationship between them if there is exists some set of relations R in the database schema that directly connects attributes from table A to attributes of table Z without going through any intermediate table X (see Figure 2).

Tables which are directly related can be joined by reading the corresponding set of relations from the database schema and adding an appropriate INNER JOIN statement to the SQL query representation. For tables that are not directly joined, a join path must be constructed by recursively traversing all intermediate tables X in the database schema. Tables that are connected indirectly can be joined by joining a single attribute table to sub-queries containing the context table and one or more other attribute tables. Thus, the problem of linking indirectly joined tables retracts to the problem of connecting those which are directly joined. Finally, recursive maze tree traversal algorithms are adopted for linking tables with respect to relationship diagram of the database.

5 SQL CONSTRUCTION

The process of creating new SQL statements within this system is limited to certain aspects of the SQL tree structure. As QMAEP is a system designed specifically for SELECT Queries for data retrieval, the basic structure of the SELECT query will contain only the query context field from the context table, and the remaining components will be filled with attribute fields. Qualifications can also be obtained from the users to be used for each field in the query, which would be entered in the Criteria box (orange) at the bottom of the form shown in Figure 3.

As discussed in section 3.2, the query context field is the first field added to any new query. This field organizes the display and retrieval of all information in the query. As the user adds to a query it still only has 1 query context field but the number of attribute fields grows. To demonstrate the use of the QMAEP Query System, an example of retrieving information from a

hospital database regarding Patients will be detailed here.

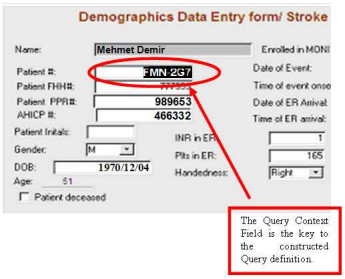


Figure 4: Partial view of an existing database form in a hospital database.

Let us begin with the user selects the patient# field from the demographics form and then loads the QMAEP editor. The form is linked in the background to one or more database tables. The linking information can be programmatically read off the form, and thus the QMAEP system can dynamically interact with any new or modified forms and/or a modified database schema. In Figure 4 a new query definition is about to be constructed and the user has chosen a PatientID field as the context field. The demographics table, which is the table that contains the PatientID field in the patient database schema, will thus become the context table.

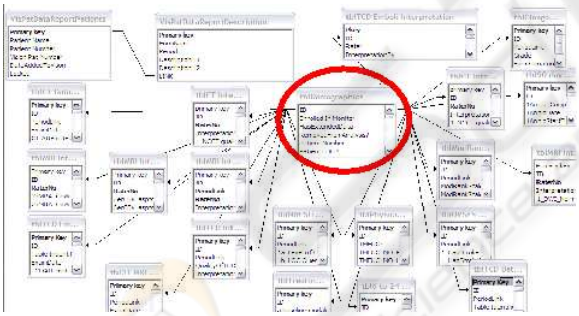


Figure 5: MS Access Object Relational View of tables. The table circled is the Query Context Table.

Because the system object model contains the information as to what table the chosen field belongs to, QMAEP now learns the Query Context Table and it adds to the domain category from Figure 6. The user having decided what will be the Query Context Table can then press a hot key to move on to the next field and add it to the query. The hot key will bring up the QMAEP query construction window. Inside the query construction window are details such as what the Query Context Table is and what the Domain values are for the chosen field. There is also an area in which users are able to define the Qualifications on the data:

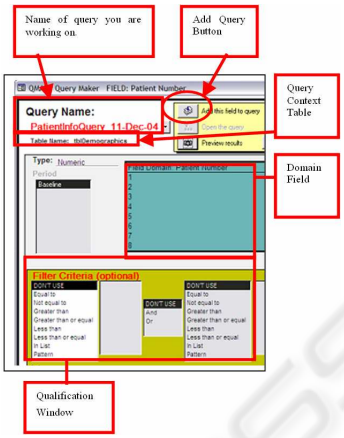


Figure 6: QMAEP Query GUI.

With the QMAEP query form filled out as in Figure 6, when the user clicks 'add field' button, QMAEP now crates the following Simple SQL statement containing only the context field. It is obvious that QMAEP is selecting the patient numbers because it is the Query Context Field.

```
SELECT      tblDemographics.[Patient Number]
FROM        tblDemographics
ORDER BY    tblDemographics.[Patient Number];
```

Figure 7: SQL statement created.

```
SELECT tblDemographics.[PatientNumber], tblDemographics.PatCommonName
FROM   tblDemographics
ORDER BY      tblDemographics.[Patient Number];
```

Figure 8: SQL statement created.

To add additional information to the resulting query, the process is just as straightforward. User just locates another field in the database forms. Once the field is located, the hot key is toggled once again. If one were to add the 'patients name' field to the table, the resulting SQL statement would be as shown in Figure 8. Note that any filter criteria would then be entered into the WHERE clause of the SQL statement. To do this again and add other fields such as Patient FHH#, the resulting SQL statement shown in Figure 9 is created.

Because QMAEP completely abstracts the user from the physical aspects of the database schema, adding attribute fields to a query from a table other than the context-table is just as simple, as far as the user is concerned.

There are two main cases in joining the select statements from two different tables. Lets assume that we wish to combine our Query Context Table with Table Y from the database.

```
SELECT      tblDemographics.[PatientNumber], tblDemographics.PatCommonName,
            tblDemographics.[Patient FHH#]
FROM        tblDemographics
ORDER BY  tblDemographics [Patient Number];
```

Figure 9: SQL statement created.

As we have seen, the Query Context Field is decided before any attribute field is added to the query definition. The context field acts as a pseudo-key to the query definition (strictly speaking, a query definition is not required to have a key). However, because of the fact we are treating the context-field as a key to the query result there is no need to worry about the cardinality on the Query Context Table Side of any relation/s linking it to the new attribute table. In terms of the actual process that is done by QMAEP in adding attribute fields, we can generalize to the two basic cases discussed next in Sections 5.1 and 5.2. The cases are organized by the cardinality on the attribute table side of the joining relationships.

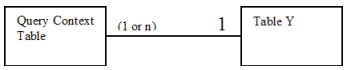


Figure 10: Schema diagram of Case A type relationships.

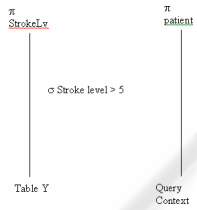


Figure 11: Example of Case A type relationship using relational Algebra tree diagrams.

5.1 Case A: (1 to 1), or (N to 1) Relationship

Case A situation (see Figure 10 and Figure 11) would occur when it is required to join an attribute table with a cardinality of 1 with respect to the context-table. Note that if the relationship between the context and attribute tables is an indirect relationship, a cardinality of 1 must be the *maximum* (hence only) cardinality on any joining relation in the join path.

One or more JOIN SQL operations will be defined in the SQL query result once the path information from one of the traversal algorithms is obtained. Alternatively, a union of the tables is sometimes possible for this type of relationship. Case A relationships are the simplest type as only a single matching attribute value from the attribute table side of the relation is possible for each value in the context field. As mentioned in Section 4, if this is the only type of relationship we are interested in when constructed queries, a directed links approach is all that is necessary for joining the tables.

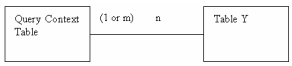


Figure 12: Schema Diagram of Case B.

5.2 Case B: (1 to M) or (N to M) Relationship

When working with this type of relationship (see Figure 12), QMAEP dynamically creates a list box for any other key fields (non-linking key fields) from the attribute table, as described in section 4. When the user choose a value from the list box, a subquery is created to extract a portion of the table such that the data extracted has a one-to-one correspondence with the context table. The subquery is named appropriately to represent both the table it represents as well its key value it was extracted with. The naming of the subquery will eventually be used to name the fields added from it, so as the user can easily identify them in the query result.

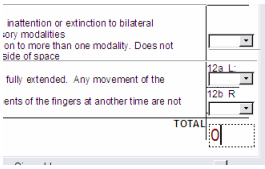


Figure 13: User chooses a field from a table with plural cardinality with respect to the context field.

Continuing with the example from Figure 8, supposed the user next chooses a field from the NIH Stroke scale (NIHSS), which has a plural cardinality with respect to the demographics because it is completed a multiple time points for each patient. When the QMAEP query window loads, it will show a list box such as the one shown in Figure 13 to allow the user to choose a period (which is the non-linking key field in tblNIH Stroke Scale) to query the field at.

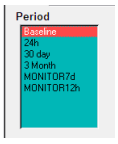


Figure 14: QMAEP's dynamically created list box for the non-linking key component of tblNIH Stroke Scale.

When the user then clicks the 'Add field' button a subquery definition for table [tblNIH Stroke Scale] is created as demonstrated in Figure 14.

```
SELECT [tblNIH Stroke Scale].PeriodLink,
       [tblNIH Stroke Scale].*
FROM   [tblNIH Stroke Scale]
WHERE  ((([tblNIH Stroke Scale].PeriodLink)='Baseline'));
```

Figure 15: Subquery definition for [tblNIH Stroke Scale].

```
SELECT [tblNIH Stroke Scale].PeriodLink,
       [tblNIH Stroke Scale].*
FROM   [tblNIH Stroke Scale]
WHERE  ((([tblNIH Stroke Scale].PeriodLink)='Baseline'));
```

Figure 16: SQL query for combined Queries.

This subquery is saved as a new query definition which is called [x_tbINIH Stroke Scale_Baseline]. Joining this subquery definition now reduces to the problem stated in Case A (Section 5.1). The user's query definition now will be modified to link in the subquery and pull appropriate fields (see Figure 15). As discussed in Section 4, the join type for attribute tables is always an INNER JOIN (or equivalently a LEFT JOIN off the context table).

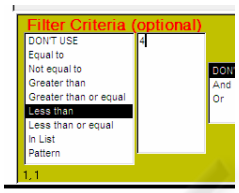


Figure 17: A possible qualification for a NIHSS value.

Lets assume also that the user had entered in a qualification for the value of baseline NIHSS, such that only patients with a NIHSS of less than 4 are to appear in the query result (see Figure 15). The query result from Figure 15 then would appear as the result shown in Figure 17. The final SQL query is shown in Figure 18.

6 CONCLUSION AND FUTURE WORK

This paper has examined a new user-friendly query system intended to assist in the process of creating queries and is intended to benefit researchers and other common database users. A case study into a particular group of medical researchers revealed that in many cases a simple to use system may be better than one which is-fully functional in terms of all existing SQL operations.

```
SELECT      tblDemographics.[PatientNumber], tblDemographics.PatCommonName,
            tblDemographics.[Patient FHH#], [x_tblNIHStrokeScale_Baseline].
            NIHStrokeScaleTotalScore
FROM    tblDemographics LEFT JOIN  [x_tblNIH Stroke Scale_Baseline] ON
            tblDemographics.ID=[x_tblNIHStroke  Scale_Baseline].ID
WHERE [x_tblNIHStrokeScale_Baseline].NIHStrokeScaleTotalScore < 4
ORDER BY  tblDemographics.[Patient Number];
```

Figure 18: SQL query for combined Queries.

The notions of a query *context table* and *query context* field were examined. A tree traversal algorithm on database relations was viewed as a way to dynamically generate necessary table linking information by reading a database schema. A method of dynamic list box creation was examined as one possible such solution.

We took a look at translation from user actions on a database system into SQL construction to generate executable queries. Relational algebra was used to illustrate an example and a step by step study of the SQL command construction was presented.

The QMAEP system is currently only integrated into one commercial use database, The system should therefore first be tested on other commercial database systems with automated integration. A new system should be provided such that data which is not intended for general queries can be treated differently, or blocked from the user. Aggregate functions, cross tab queries, update, append delete and possibly even table definition queries should be explored.

REFERENCES

Celko, J. (2004). *Joe Celko's SQL for Smarties: Trees and Hierarchies*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Doan, D. K., Paton, N. W., Kilgour, A. C., and al Qaimari, G. (1995). Multi-paradigm query interface to an object-oriented database. *Interacting with Computers*, 7(1):25–47.

Ezekiel, J. (2004). *Ethical and Regulatory Aspects of Clinical Research: Readings and Commentary*. Johns Hopkins University Press.

Owei, V., Navathe, S. B., and Rhee, H.-S. (2002). An abbreviated concept-based query language and its exploratory evaluation. *J. Syst. Softw.*, 63(1):45–67.

Preece, J., Rogers, Y., and Sharp, H. (2002). *Interaction Design*. Wiley.

Show, G. Y., Kong, H., and Lin, W. K. (1993). Intelligent user interface to sql-based database system. In *Engineering Applications of Artificial Intelligence*, volume 6, pages 307–316.

Siau, K. (1998). A visual object-relationship query language for user-database interaction. *Telemat. Inf.*, 15(1-2):103–119.