The Relevance of SQL to M Technology in 1993

by Keith Stendall

Abstract

With the growing number of SQL-based products becoming available to the M community, and bearing in mind the recently completed standards for embedding SQL (structured query language) statements in M programs, it is increasingly relevant for users to understand how to use SQL in relation to their current and future applications.

This article presents a personal viewpoint based upon experience of SQL-based products both inside and outside the world of M Technology. This viewpoint categorizes the use of SQL-based products into four distinct areas of application for SQL as:

- A commonly recognized interactive query interface;
- A standard method of passing queries into and out of an M environment;
- An embedded part of the M language; and
- A formal sublanguage of an RDBMS environment.

The article attempts to address the issues of why SQL-based products could be useful within these four categories but is not a formal review of specific products that are currently available.

Introduction

Awareness of SQL is reasonably high among M application developers. Some now use an SQL implementation regularly. Most think they understand what it is and what it does, although in my experience there are misconceptions with respect to its applicability to M Technology.

Most users of M applications do not understand SQL at all. Some are frightened by it. Many have no interest in it and do not see its relevance to their environment. Outside the world of M Technology there is a much greater awareness of SQL. SQL features prominently in all the major relational database management systems (RDBMS) such as ORACLE, INGRES, SYBASE, and others.

SQL is a data sublanguage invented by a group from IBM Research in 1972. SQL was originally based upon some of the early work done by E.F. Codd on the Relational Model. SQL is not the only data sublanguage that has been developed for RDBMS, but it is the only one that has been adopted as an ANSI (American National Standards Institute) and an ISO/IEC (International Organization for Standardization/International Electrotechnical Commission) international standard.

A myth surrounding SQL suggests that it is only applicable to relational databases. This is not true. In fact, the word *relational* does not appear on any of the 120 pages in the ISO/IEC standard. [1] The standard is concerned with the syntax and semantics of SQL. This is defined in terms of *tables* of data. The terms *relation* and *table* are not synonymous. [2] SQL may be applied to any database that has its data conceptually organized as logical tables. The standard for SQL in fact relates to two database languages:

- "A schema definition language (SQL-DDL) for declaring the structure and integrity constraints of an SQL database."
- "A module language and a data manipulation language (SQL-DML) for declaring the database procedures and executable statements of a specific database application program."

"The standard defines the logical data structures and basic operations for an SQL database. It provides functional capabilities for designing, accessing, maintaining, controlling and protecting the database."

The schema definition part of the language (SQL-DDL) is often not implemented directly by vendors. The preferred route is to precede SQL by some form of data dictionary or data modelling interface that is more user friendly. For this reason users often regard the data manipulation part of the language (SQL-DML) as synonymous with SQL.

It is worth picking up on the terms *database language*, as used in the standard, and *data sublanguage*, as referred to by Codd, and highlighting that both these terms are meant to identify that SQL is not a complete language such as COBOL, FORTRAN, BASIC, or M. [3] It is not possible to use SQL alone to write complete applications. SQL is concerned only with data definition and manipulation—updating and retrieving

Continued on page 34

data from a database. To handle a complete application, SQL must work in partnership with other languages, tools, and services. It is therefore a very natural step to consider how SQL could work in partnership with the M environment.

There are many ways to implement SQL in an M Technology environment. I will not attempt a complete technical discussion of any sort of model for interfacing M and SQL specifically. Work has already been done in this area and I would refer the reader to a good article in the proceedings of the 1990 MUMPS Users' Group-North America (MUG-NA) meeting in Orlando, Florida. [4]

As a preliminary step to discussing the relevance of SQL in operational terms, it is practical to address a primary issue. That is, can SQL work with an existing M database?

Can SQL Work with an Existing M Database?

An M database (held as a collection of globals) can be defined as a set of linked tables of data. Thus, SQL implementations may operate on an M database. It is not true to say, however, that all existing M databases can be defined as a set of tables (nor as the much tighter constraint of relational tables).

The problem is that M offers almost total flexibility in the way it holds data in records within globals. Occurrences of multiple repeating fields and context-dependent fields are not uncommon in existing M databases. These constructs are at best, difficult, and at worst, impossible, to express as tables. For a good indication of how simple M global structures can be mapped as tables, I would refer the reader to an article in the proceedings of the 1991 MUG-NA meeting in New Orleans, Louisiana. [5] The only way to ensure that the totality of an M database can be expressed as tables is to constrain the flexibility offered by M when creating new data structures. Data dictionary-based 4GL M tools and database management systems have been available now for several years.

These have started the process of constraining the native flexibility of M and have imposed some control and order on the global structures created within an M database. Many of these were not designed with the aim of allowing the resulting database to be manipulated as linked tables. The U.S. Department of Veterans Affairs' FileMan is an example of a very good database management system that imposes order and data integrity upon the M database without being based upon tables (or relational theory). Even so, the resultant database largely can be mapped as linked tables and an SQL implementation can be applied. An excellent comparison between the FileMan approach and the relational approach is given in the proceedings of the 1991 MUG-NA meeting. [6]

There are at least eight (complete or partial) implementations of SQL available to the M community that can operate on an existing M database. In the interests of fairness I will not list them since there are bound to be some that I do not know about. It is certain that more SQL implementations will soon be available. I have used several of these implementations to varying degrees, as well as having had some limited exposure to SQL implementations outside the world of M Technology. Most SQL implementations for M claim to conform to the ANSI and ISO/IEC standard. [7] Some of these are indeed faithful to the standard. I believe a few are questionable in some respect. Certainly, all implementations I have seen include extensions to the standard. The result is that no two implementations are the same although all are derived from a common base.

All the SQL implementations for M that I have seen do a fairly good job of mapping existing M globals for the purpose of data retrieval. To gain the full benefits that accrue from using SQL to control M database updates, however, it is almost inevitable that some of the existing globals will have to be restructured. Some of these benefits will become apparent in the following discussions.

It would be impossible to present a comprehensive view of SQL's expressive power within the scope of this paper. For this reason, I refer the reader to an excellent text on DB2, which gives a broad view of the mechanics of SQL and a valid point of reference from outside the world of M Technology. [8]

SQL as a Commonly Recognized Interactive Query Interface

When the requirement is simply to retrieve information from an existing M database, quite a few organizations are now adding an SQL interface as the basis of an ad hoc query facility. The idea is to open up the database to users who would not normally be able to work at M code level. The targeted users include nontechnical staff and people who do not work with M Technology at all. Remember that in many organizations non-M applications built within a 4GL environment are the norm. Addition of an SQL interface to the M database often has great appeal to such organizations.

Even with an SQL interface, users still need to know how the database is structured. SQL will present the data as named tables. Each table will contain named columns of data. These tables may have been set up by using SQL-DDL or more commonly by making entries in a data dictionary. Sometimes tables are automatically created by using existing dictionary structures. Whichever approach is used, it is common to find that an M database "explodes" into a very large number of tables. This is unavoidable and is not dependent upon the

SQL implementation selected. The resulting SQL data dictionary is usually supplied with some facility for browsing through the entries. The user forms a list of table names and column names that represent the data he or she is interested in. If data are required from more than one table then the user also has to say how the tables are to be linked. With relational implementations of SQL this means specifying a *join* operation. There are many different kinds of join, the most common being the *natural join*—a join based on the values held in two columns being equal. Figure 1 gives an indication of how a simple query, based on two tables, might work.

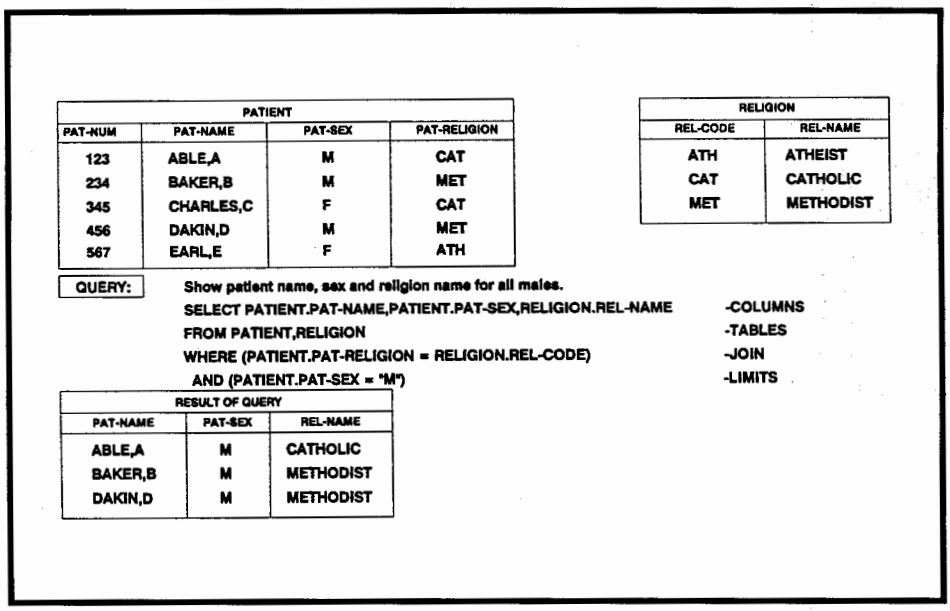

Figure 1. A simple join operation.

This example is simple and quite easy for an end user to understand. In practice, useful queries often involve joins of five, six, or more tables. This can be tedious and prone to errors. It is not unusual to see ten or twelve lines of SQL code in an average query. Although the number of joins required can be minimized by making use of prepared views of the database, they can never be eliminated entirely.

Having extracted our data using SQL, we would normally like to put it into a reasonable layout on a report. Here we hit a problem. The SQL standard does not include any facilities for specifying the position or format of data on a report. [9]

Individual vendors usually solve this problem in one of two ways: by specific extensions to the SQL syntax, or by passing the SQL query result to a purpose-built formatter. Either way, when it comes to laying out report structures the solution is always likely to be a proprietary one.

The best SQL implementations now available tend to provide a user-friendly front end to the SQL-DML scripting language. The user is guided through a process that results in the SQL code being generated automatically. Hence there is no particular need to learn the SQL syntax or (in some cases) how to join tables correctly.

If the SQL code is now going to be translated into the equivalent M code (as most current implementations do), then why should we bother with the SQL code at all? This question is largely answered in the discussion which follows, but if the sole requirement is to perform local queries and reports on an existing M database, then many good alternatives to SQL are available.

Passing Queries into and out of an M Environment

Here we examine a much broader set of requirements, whereby an M application needs to retrieve information from non-M application environments such as ORACLE, INGRES, and SYBASE.

Also, these non-M applications may need to retrieve information from an M database. It has been possible for some time now to extract data from an M database and download it into a non-M environment such as a spreadsheet or a graphics package. There are many products around that do this. They take advantage of standard protocols that have been established for the import and export of data (e.g., Data Inter-

change Format). This type of process is largely initiated from the M environment, however, and is not coordinated in any interactive way.

What if I know that an ORACLE database contains information about patient admissions and my M application needs details for those patients admitted yesterday? What if I know that an M database contains information about suppliers of motor vehicles and my INGRES application needs to know the current price and delivery details for all Ford pickup trucks?

Solutions to this type of requirement are only just starting to appear. The vast majority of these solutions are based on the idea of passing SQL script from one application environment to another. This type of process is categorized by the term *client/server*. One environment acts as a *client* and issues a request in the form of an SQL query. The other environment acts as a database *server* (potentially to many clients) and passes back the result of the query as a table of data. Figure 2 shows a simplistic view of this mechanism with an M environment acting as a server to a non-M environment.

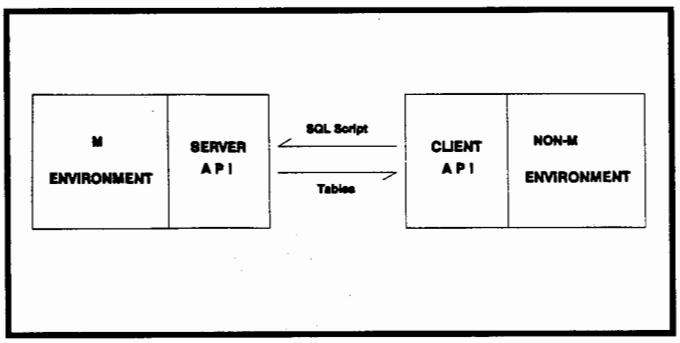

Figure 2. An example of the client/server process.

API stands for application program interface. This is the mechanism that manages the protocol needed to pass SQL script and tables between different environments using various channels of communication. To make this process work reliably in the general case, an accepted standard protocol is needed, preferably one that is controlled by some independent body such as ANSI or ISO. Unfortunately there is no such standard and one is not likely to emerge for some time. There are plenty of proprietary solutions and some of these are being adopted as pseudo-standards for the industry. As one would expect, the major RDBMS implementors are leading the way. M product vendors are very active in this area and some prototypes are available for client/server links to specific non-M environments.

It could be quite a while before a generic client/server link is available. Indeed, some of the major RDBMS vendors are reticent in their approach to cooperating fully with other environments. I do not believe that this is an attitude that will prevail in the face of commercial pressure. Client/server is clearly an issue that will affect the whole of the M Technology community.

As solutions become commercially available, users will need to evaluate them carefully. A good implementation of SQL that is faithful to the standard is a "must."[10] Any extensions made by a vendor to SQL are very unlikely to be applicable across a client/server link. Along with the SQL implementation the user will need an API for both the client and the server ends. These will most likely be sold as add-on products.

SQL as an Embedded Part of the M Language

The ISO/IEC standard includes annexes that explain the syntax and structure for embedding SQL into various host languages such as COBOL, FORTRAN, PASCAL. [11] In fact, with the recent publication of the new SQL2 standard, M may now be added to this list. (The SQL2 standard was not published at the time this paper was first presented in 1992.)

Indeed, implementations of M with embedded SQL already exist and are in use. So, what are the advantages of using SQL in this way? This article commented earlier on the almost total flexibility in the way M can be used to hold data. Historically this has been seen as one of the biggest strengths of M. More recently it has become apparent that this flexibility causes problems if it is not controlled properly.

Using SQL within M to handle all data manipulation has a number of immediate benefits. It:

- Reduces the scope and diversity of the file structures that may be maintained;
- Isolates the data handling from the functionality of an M application and separates the logical access path from the physical access path;
- Can offer a guarantee of referential integrity; and
- Can make the application easier to document, easier to understand, and thus much easier to maintain.

The guarantee of referential integrity is an important feature that warrants further explanation. Briefly, this means that a data attribute found in many files is guaranteed to be maintained in a consistent, integral way. For example, if an attribute—such as a patient identifier—is present in nine or ten separate files that are linked together, a change to a specific patient identifier in one of these files automatically would produce appropriate changes to all the other files.

Having acknowledged these benefits in relation to SQL, it is only fair to say that the same benefits can also be obtained without the use of SQL. Consider VA FileMan. Surely, all the same benefits apply. Indeed, there are other M 4GL tools available that also give some or all of these benefits.

The solution offered by such M 4GL tools is to precede the M language by a programmer interface capable of generating the M code needed for file handling. In some respects it could be said that using embedded SQL directly at the M code level is rather a retrograde step!

SQL as a Formal Sublanguage of an RDBMS Environment

This final topic involves some gazing into a crystal ball. Figure 3 illustrates how I would envisage SQL fitting into the M environment of the future.

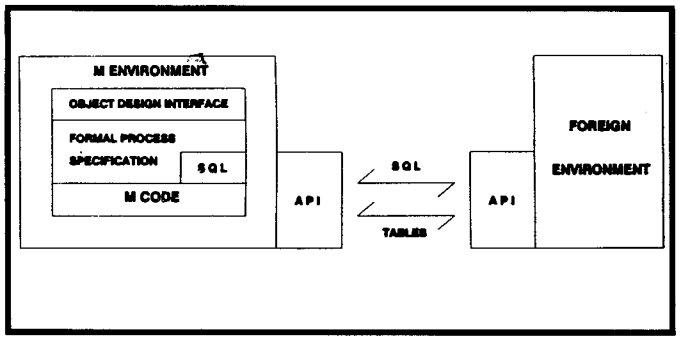

Figure 3. SQL with M in the future.

Applications would be designed using some form of object design interface. This is effectively a set of front-end tools, which themselves are modelled in a data dictionary and may be modified to cater to individual organizational requirements.

The object design interface will be capable of defining a specific process in terms of a formal specification combined with operations on the database expressed in terms of SQL.

The SQL can be translated into M code along with the formal process definition to produce a run-time M code module that can either operate exclusively on the local M database or pass SQL script via a client/server mechanism to a separate environment where the query is run. The remote query will yield either a database update within that environment or the retrieval of a table of data that will be passed back to the M environment.

This scheme would enable realization of the full benefits of both M and SQL. The layering of SQL on top of M is important to allow full flexibility of access to both M and other database environments without disturbance to the formal specification.

Continued on page 38

Both the formal process definition and the data manipulation, expressed as SQL script, benefit from the portability and power of the facilities available at the M code level.

The object design interface is the essential mechanism that allows process and data definition to be brought back together into classes of reusable object components. In this "total solution," SQL is used primarily for its two key properties:

- A formal definition of how to handle a database without loss of generality or integrity; and
- A formal standardized syntax for use in multiple heterogeneous database environments.

In defining what I see as the future position of SQL in relation to M, it is apparent that I am also describing a view of M as the underlying support for higher layers of interface. I am putting forward my belief that applications will not be built directly at the M code level. In fact, M ends up almost as part of the operating system environment. The higher-level tools are written in M code and are used to produce applications that are also written in M. There is no distinction between tools and applications here. A tool is just a specialized form of application for use by application builders. The software environment is essentially seamless. The smallest and largest applications and the most complex of tools are all part of a continuum that is infinitely extensible. Furthermore, as standards evolve for the exchange of designs (as expressed by formal specifications), applications can be transferred totally between environments to be regenerated in some other source language.

This may present a somewhat different view of the ultimate use of SQL and M than you had expected. M as a language disappears from view, but so do FORTRAN, COBOL, etc.

This is not to say that the role of M will be diminished in any way. In fact, the reverse is true. A lot of important software technology will be built upon the foundation stone of M and will be entirely dependent upon it. Isaac Newton is said to have remarked that insofar as he had achieved anything worthwhile, he had done so by standing on the shoulders of giants. I believe that M has very broad shoulders!

The views set forth here may be science fiction to many readers. In fact, most of the mechanisms described here are currently available or under development today. No single product as yet brings all the facilities together as a complete environment, but that point is rapidly approaching. M

Keith Stendall is managing director at Coltec Systems Limited in Lichfield, England. This article is based on a paper presented by the author at the 1992 MUG-Europe meeting in Vienna, Austria. The original presentation was published in the proceedings of that meeting.

Endnotes

1. "Information Processing Systems Database Language SQL with Integrity Enhancement," *International Standard ISO/IEC 9075* and ANSI Standard X3.135.
2. E.F. Codd, *The Relational Model for Database Management, Version 2* (Reading, Massachusetts: Addison-Wesley Press, 1990), 17-20.
3. "Information Processing Systems," ISO.
4. E.J. McIntosh, "A Model for Interfacing MUMPS and SQL," *MUG Quarterly* 20:3 (June 1990): 14-20.
5. D.W. Middleton, "Query Optimisation in MUMPS," *MUG Quarterly* 21:3 (June 1991): 32-39.
6. T.K. Winn and M.L. Hoye, "Relational Features of VA FileMan," *MUG Quarterly* 21:3 (June 1991): 46-51.
7. "Information Processing Systems," ISO.
8. C.J. Date and C.J. White, *A Guide to DB2, Third Edition* (Reading, Massachusetts: Addison-Wesley Press, 1990).
9. "Information Processing Systems," ISO.
10. Ibid.
11. Ibid.