SQL Multimedia and Application Packages (SQL/MM)

Jim Melton
Oracle, Sandy, UT 84093
jim.melton@acm.org

Andrew Eisenberg
IBM, Westford, MA
andrew.eisenberg@us.ibm.com

Introduction

Regular readers of this column will have become familiar with database language SQL — indeed, most readers are already familiar with it. We have also discussed the fact that the SQL standard is being published in multiple parts and have even discussed one of those parts in some detail[1].

Another standard, based on SQL and its structured user-defined types[2], has been developed and published by the International Organization for Standardization (ISO). This standard, like SQL, is divided into multiple parts (more independent than the parts of SQL, in fact). Some parts of this other standard, known as SQL/MM, have already been published and are currently in revision, while others are still in preparation for initial publication.

In this issue, we introduce SQL/MM and review each of its parts, necessarily at a high level.

Jim Melton and Andrew Eisenberg

SQL Multimedia and Application Packages — SQL/MM

In late 1991 or early 1992, a small group of text search engine vendors, operating under the auspices of the IEEE, released a specification for a language called SFQL (Structured Full-text Query Language). The goal of SFQL was to define extensions to SQL that would be suitable for applying full-text searches to repositories of documents.

The proposal was given significant attention by the full-text community, but was immediately criticized by several other data management communities on the grounds that SFQL "hijacked" many useful keywords that were in common use by those other communities. For example, the keyword CONTAINS was proposed by SFQL to mean "the indicated unit of text *contains* the supplied word or phrase", but the spatial data community used the same keyword to mean "one spatial entity *contains* a second spatial entity". While the high-level semantics of the word may seem to be quite similar in each case, the actual code required to implement it is dramatically different.

This controversy was sufficiently generalized that the SQL standards organizations realized that many incompatible extensions to SQL would be defined by various data management communities, the end result being a situation in which no single product could possibly implement all of the extensions because of conflicts in keywords (and other related conflicts).

A summit meeting was held in Tokyo later in 1992 to seek a solution to the dilemma posed by the conflicting demands on SQL extensions. By that time, the SQL standards committees were in the process of adding object-oriented extensions to SQL and a number of SQL vendors had indicated their support for what is often called the "object-relational model". Based on suggestions from several of those vendors, the Tokyo summit developed the notion of a second standard that would define several "class libraries" of SQL object types, one for each significant category of complex data.

The structured types defined in such libraries would naturally be first-class SQL types that could be accessed through ordinary SQL:1999 facilities, including expressions that invoke SQL-invoked routines associated with such types (that is, methods).

The proposed standard was immediately known as "SQL/MM" (MM for MultiMedia). A number of candidate data domains were suggested, including full-text data, spatial data, image data (still and moving), and others. Responsibility for SQL/MM's development was given to the same ISO subcommittee as SQL (at that time, JTC1/SC21, but now JTC1/SC32), with the hope that domain experts would attend to develop the specifications for each data domain.

Like SQL, SQL/MM is a multi-part standard. Unlike SQL, the various parts of SQL/MM are quite independent from one another. However, there is one part that is common to the remainder of the word. Part 1, known as the Framework[3], provides definitions of common concepts use in the other parts and outlines the definitional approach used by those other parts. In particular, it describes the manner in which the other parts use SQL's structured user-defined types to define the types required by the subject matter of each part.

Full-Text

The term "full-text" (or, if you prefer, "full text") is normally applied to textual data that differs from ordinary character string data principally in its length, but also in database-specific operations that can be applied to it. Ordinary character strings are usually indexed by their entire values, but special types of indexes are defined for full-text data; such indexes might record information about the proximity of words and phrases to one another or about words that appear in a document and related words that do not appear in the same document. Full-text data is subject to search operations that are normally not applied to "simple" character strings. It's worth pointing out that "full-text operations" are quite different than the sort of pattern matches (such as regular expressions) with which most computer software people are intimately familiar.

The SQL/MM Full-Text standard[4] defines a number of structured user-defined types (henceforth, just "UDTs") to support the storage (presumably in an object-relational database) of textual data. One of these types is named *FullText* and it supports the construction of full-text data values, testing whether that data contains specified patterns, and conversion of that data to ordinary SQL character strings. The specification of the FullText type includes a number of methods that prepare the value associated with an instance of the type for the application of full-text searches, as well as Boolean methods that perform the searches themselves.

In addition to the FullText type, a number of additional types are defined to represent various sorts of patterns that can be used in full-text searches. Search patterns can be quite complex, including searching for text that includes specific words, words *stemmed* from (such as the past tense of a verb or the plural of a noun) specified words, words with similar definitions, and even words that sound like a given word.

Linguists among our readers will know that some languages are much more amenable to computer identification of components of text than others. For example, most Western languages use white space to separate words from one another and use special punctuation (such as a period, or full stop) to separate sentences. Other languages, such as Japanese, do not separate words from one another by spaces, depending primarily on context to distinguish words. SQL/MM Full-Text is generally acknowledged to have better support for languages for which automatic distinction of language tokens (such as words) is relatively easy.

Consider the following SQL table:

```
CREATE TABLE information (
  docno          INTEGER,
  document       FULLTEXT )
```

in which the `docno` column contains a value that captures some document identifier and the `document` column contains a full-text document.

We could retrieve from that table the identifier of documents about full-text searching that contain words closely related to "standard" in the same paragraph as words that sound like "sequel" by using a query like this:

```
SELECT docno
FROM information
WHERE document.CONTAINS
      ('STEMMED FORM OF "standard"
       IN SAME PARAGRAPH AS
       SOUNDS LIKE "sequel"') = 1
```

That query retrieves the `docno` column from the `information` table for every document for which the value returned by the CONTAINS method applied to the `document` column is 1, meaning *true*. The parameter passed to that method uses three different full-text operations: STEMMED FORM OF will find any of several words derived from "standard", such as "standards" and "standardization"; IN SAME PARAGRAPH AS requires that a second word (or phrase!) appear in the same paragraph as the stemmed word; and SOUNDS LIKE finds words that are pronounced (presumably in English, since we didn't specify a different language) like "sequel" (of which "SQL" might be a case).

Spatial

Many enterprises need the ability to store, manage, and retrieve information based on aspects of spatial data, such as geometry, location, and topology. Applications making use of spatial data include automated mapping, facilities management, geographic systems, graphics, multimedia, and even integrated circuit design. The SQL/MM Spatial standard[5] defines SQL:1999 structured user-defined types and associated methods to provide the ability to support such applications.

By its very nature, spatial data often represents 2-dimensional and 3-dimensional data. SQL/MM Spatial currently supports 0-dimensional (point), 1-dimensional (line), and 2-dimensional ("flat" shape) data; future revisions might support 3-dimensional (volumetric shapes) and possibly data of even higher dimensions.

There are an astonishingly large number of spatial reference systems in common use, the vast majority of them used to describe geographic entities and concepts on the surface of our (relatively) spherical planet. Many of those spatial reference systems deal with large structures for which the curvature of the

planet is significant; as a result, various systems have evolved to describe structures in particular regions (e.g., countries, states and provinces, etc.) for which the impacts of planet curvature vary from the impacts in other regions. (For example, lines of longitude converge towards one another as one moves close to the poles—seemingly parallel lines of longitude are in fact not parallel.)

Support for these spatial reference systems are economically critical to the design of SQL/MM Spatial, because the largest users of spatial data management systems are often governmental bodies and very large commercial enterprises that have to deal with geographic data. Such users include local governments (city planning, traffic management, accident investigation), state and provincial governments (highway planning, natural resource management), national governments (defense, border control), extractive industries (mineral and water location), and farming (plot allocation). Indeed, SQL/MM Spatial's design seems to more naturally support geospatial data than smaller-scale data such as integrated circuit design and computer graphics.

SQL/MM Spatial defines several type hierarchies. One of those hierarchies has as its most generalized type (that is, its maximal supertype) a type called ST_Geometry. That type is not instantiable (meaning that no instances of it can be created—Spatial defined less than a half-dozen such types), but it has a number of (about a dozen) subtypes that are instantiable, such as ST_Point, ST_Curve, and ST_MultiPolygon.

A type (not a subtype of ST_Geometry) called ST_SpatialRefSys is used to describe spatial reference systems. Every spatial value that participates in a given query must be defined in the same spatial reference system, although a future version of the Spatial standard might relax that restriction.

In a future version of SQL/MM Spatial that is currently under development, another pair of types, ST_Angle and ST_Direction, are used to capture information about various angles and directions that are needed when storing and managing spatial information.

There are many operations that can be performed on Spatial data. Among the most common operations are: construction of a straight line from two points or from one point, a direction, and a distance; construction of a polygon from several lines, from several points, or from a point and a collection of directions and distances. Other important operations are detection of whether two lines intersect, whether two areas overlap or are adjacent to one another, whether a line is tangent to a curve, and whether two polygons share a boundary.

Most Spatial types have accessor methods that permit applications to extract fundamental information about instances of the type, such as determining the values of the X and Y coordinates of a point.

Consider the following table definition:

```
CREATE TABLE CITY (
  NAME        VARCHAR(30),
  POPULATION  INTEGER,
  CITY_PARKS  VARCHAR(30) ARRAY[10],
  LOCATION    ST_GEOMETRY )
```

We can determine the area of San Francisco by executing a query like this:

```
SELECT location.area
FROM CITY
WHERE name = 'San Francisco'
```

The expression `location.area` retrieves the `area` attribute of the ST_Geometry structured type value stored in the `location` column of the row corresponding to San Francisco. (Retrieving the value of an attribute of a structured type instance is equivalent to invoking the accessor method on that attribute.)

SQL/MM Spatial is closely related to, and fundamentally aligned with, other spatial standards being developed by another ISO Technical Committee, TC 211 (Geomatics) and by the Open GIS Consortium ("GIS" stands for "Geographic Information Systems"). Keeping standards being developed in all three forums has proved challenging, but all participants seem committed to doing so.

Still Image

One of the fastest growing applications of computers is storage and processing of visual images such as photographs. Many enterprises expend tremendous resources on the acquisition, storage, and management of collections of images, including graphics, paintings, and photographs. Such data has tremendous business value and represents large monetary outlays. One of the most challenging aspects to handling image data is that of *locating* an image already in your possession.

SQL/MM Still Image[6] represents a part of the solution to those problems. This part of the SQL/MM standard provides structured user-defined types that allow you to store new images into a database, retrieve them, modify them in various ways, and—most importantly—to locate them by applying various "visual" predicates to your collections of images.

In SQL/MM Still Image, images are represented using an SQL:1999 structured type called SI_StillImage. This type stores collections of picture elements (pixels) representing 2-dimensional images.

(Of course, images of 3-dimensional objects are very common, but the images themselves are 2-dimensional.) Images can be stored in any of several formats, depending on what the underlying implementation supports—for example, formats such as JPEG, TIFF, and GIF are commonly supported as input and output formats, as well as formats in which images are stored and manipulated. The SI_StillImage type also captures information about each image, such as its format, its dimensions (height and width in pixels), its color space, and so forth.

Methods applied to SI_StillImage instances include routines to scale an image (change its size proportionally), to crop an image (remove undesired parts), rotate an image (such as changing its orientation from horizontal to vertical), and creating a "thumbnail" image (a lower resolution image used for quick display).

Another group of data types are used to describe various features of images. The SI_AverageColor type is used to represent the "average" color of a given image; this value may be used in locating images in collections (imagine wanting to find an image that is primarily green to be used in advertising outdoor furniture). The SI_ColorHistogram type provides information about the colors in an image at a finer level of granularity than the image's average color; it indicates *how much* of each color is found in an image. The SI_PositionalColor type represents the location of specific colors in an image, supporting queries such as "since sunsets at sea have red and orange above dark blue, find me images with those color characteristic". Finally, the SI_Texture type allows the recording of information such as coarseness, contrast, and direction of granularity. An SI_FeatureList type permits recording all of the features described in this paragraph for each image.

By combining several features of an image, it is possible to write queries that can retrieve from a very large image base a much smaller collection of images from which you can quickly select the exact image you want. It is also possible to *screen* collections of images to find images of potential interest for various reasons. For example, you might want to determine whether a new logo you've commissioned might conflict with other logos that have already been copyrighted. An SQL statement like this one:

```
SELECT *
FROM REGISTERED_LOGOS
WHERE SI_findTexture(newLogo).
        SI_Score(Logo) > 1.2
```

would do just what you need.

Of course, not all images are "still". Additional challenges are posed by moving images, such as digitized video. That sort of data is not addressed by SQL/MM Still Image, but it is possible that some future part of SQL/MM will be oriented towards moving images.

Data Mining

The parts of SQL/MM that we've presented so far in this column are all very reasonably described as oriented towards the handling of *multimedia* data. However, as you saw in the early sections of the column, the full name of the SQL/MM standard is *SQL Multimedia and Application Packages*. In fact, work was initiated in early 2000 on a new part of SQL/MM that does not address multimedia data, but instead defines an application package.

SQL/MM Data Mining[7] defines SQL structured user-defined types—including methods on the types—to address an important aspect of modern data management: the discovery of previously unknown, but important, information buried in large quantities of data that might have been collected for other, quite distinct reasons.

Data mining is not a new concept; indeed, companies have long wanted to use data collected in the ordinary course of business as a source of information about their customers or other resources. A number of relatively small, but important, companies were founded during the 1990s to provide enterprises with data mining products, some of them based on relational database systems, but most of them dedicated applications that require importing data stored in another repository and reorganizing it into structures unique to a particular data mining approach.

SQL/MM Data Mining takes a different view of the problem: It attempts to provide a standardized interface to data mining algorithms that can be layered atop any object-relational database system and even deployed as middleware when required.

In most data management environments, applications pose questions to the data repositories that retrieve information based on specific criteria. By contrast, in a data mining environment, applications often ask the repository to find out what criteria are most important.

For example, a data mining engine can discover, informing its users of the discovery, that (to use a famous, if apocryphal, example) about half of the customers who buy both disposable diapers and beer will buy an air freshener product as well. This is not the sort of question that most users would dream up by themselves (it certainly doesn't come to *our* minds very often!), but it is precisely the kind of relationship that a data mining product will discover.

A popular question that a data mining product might be asked is "Who are my most important customers and what are the most significant attributes of

those customers and the trends in the values of those attributes?" The first part of the question may seem easy—it's usually straightforward to find out what customers have bought your products or services recently. But "most important" may have other meanings than "recent purchases"—profits are not always directly related to purchases, since growth rates, service demands, and other factors can significantly affect the meaning of importance.

Data mining tools are also used for predictive purposes, such as insurance companies mining data on existing customers to help evaluate the risks associated with new customers.

There are four different data mining techniques supported by this standard. One technique, the *rule model*, allows you to search for patterns ("rules") in the relationships between different parts of your data. A second technique, the *clustering model*, helps you group together data records that share common characteristics and identify the most important of those characteristics. The third technique, the *regression model*, helps you predict the ranking of new data based on an analysis of existing data. The final technique, the *classification model*, is very similar to the regression model, but it is oriented towards predicting which grouping or class new data will best fit based on its relationship to existing data.

For each of those techniques, as with most data mining product, there are three distinct stages through which you can mine your data. First, you have to *train* a model; this means choosing the technique most appropriate to your goals, then setting a few parameters to orient the model, and finally training the model by applying it to a reasonably-sized data set (perhaps several times for improved validity). Second, if you're using the classification or regression techniques, you can *test* the model by applying it to known data and comparing the model's predictions with that known data's classification or ranking. Finally, you *apply* the model to your business data and use the results to improve your enterprise.

The models are supported through the use of several broad categories of new structured user-defined types. For each model, a type known as DM_*Model (where the '*' is replaced by 'Clas' for a classification model, 'Rule' for a rule model, 'Clustering' for a clustering model, and 'Regression' for a regression model), is used to define the model that you want to use when mining your data. The models are parameterized using instances of the DM_*Settings ('*' is 'Clas', 'Rule', 'Clus', or 'Reg') type and the models are trained using instances of the DM_ClassificationData type. The DM_*Settings type allows various parameters of a data mining model, such as the depth of a decision tree, to be set.

Once a model has been created and trained, it can be tested by building instances of the DM_MiningData type that holds test data, and instances of the DM_MiningMapping type that specify the different columns in a relational table that are to be used as a data source. The result of testing a model is one or more instances of the DM_*TestResult type ('*' can only be 'Clas' or 'Reg'). When running your model against real data, you get the results in instances of the DM_*Result type ('*' can be 'Clas', 'Clus', or 'Reg'…but not 'Rule').

In most cases, you also create and use instances of DM_*Task types to control the actual testing and running of your models.

At the time this column went to press, it seemed likely that final progression of the SQL/MM Data Mining standard might be slowed just a little bit to ensure that it is fully compatible with a "sister" data mining API being developed for Java by the Java Community Process.

Summary

The SQL/MM suite of standards includes a Framework that describes the conventions used to define each of the other parts. There are other parts used to manage full-text data, spatial data, and still images, and to data mining.

Careful inspection of the references below will reveal that there is no part 4 of this multi-part standard. That's because an attempt to develop a set of classes for general mathematical operations was eventually determined to satisfy too few users at too great a cost; development of SQL/MM General Purpose Facilities was thus abandoned several years ago.

Not all parts of SQL/MM are yet commercially successful, but the seems to be growing support at least for both Full-Text and Spatial by several important players in those fields. Support for Still Image seems to be developing more slowly, and it's far too soon to say about Data Mining since that part has not yet been published. Whether additional data types (such as moving image data) are ever supported depends on many factors, including interest from the technical community depending on such data. The recent surge in consolidation within the database industry causes some to think that there is a reduction in the need for such standards, but the greater attention being paid to the Internet and the World Wide Web prove that the need for portability of data and of code continues to increase.

If you're interested in acquiring copies of the SQL/MM standard's various parts, you can do so at ANSI's electronic standards store cited below. Unfortunately, even in downloadable (PDF) form, these standards are a bit pricey. We expect that, once they

have been formally adopted as American National Standards, they will be available at the NCITS web store for very reasonable prices.

References

[1] Jim Melton, Jan-Eike Michels, Vanja Josifovski, Krishna, Kulkarni, Peter Schwarz, Kathy Zeidenstein, *SQL and Management of External Data*, SIGMOD Record, Mar., 2001.

[2] Jim Melton and Andrew Eisenberg, *SQL:1999, formerly known as SQL3*, SIGMOD Record, Feb. 1999.

[3] ISO/IEC 13249-1:2000, *Information technology — Database languages — SQL Multimedia and Application Packages — Part 1: Framework*, International Organization for Standardization, 2000.

[4] ISO/IEC 13249-2:2000, *Information technology — Database languages — SQL Multimedia and Application Packages — Part 2: Full-Text*, International Organization For Standardization, 2000.

[5] ISO/IEC 13249-3:1999, *Information technology — Database languages — SQL Multimedia and Application Packages — Part 3: Spatial*, International Organization For Standardization, 2000.

[6] ISO/IEC 13249-5:2001, *Information technology — Database languages — SQL Multimedia and Application Packages — Part 5: Still Image*, International Organization For Standardization, 2001.

[7] (ISO/IEC) FCD 13249-6, *Information technology — Database languages — SQL Multimedia and Application Packages — Part 6: Data Mining*. [FCD = Final Committee Draft for ballot]

Web References

[1] National Committee for Information Technology Standards (NCITS): http://www.ncits.org

[2] NCITS H2 – Database Committee: http://www.ncits.org/tc_home/h2.htm

[3] ISO/IEC JTC 1/SC 32: http://www.jtc1sc32.org

[4] ANSI's Electronic Standards Store: http://webstore.ansi.org

[5] NCITS' Standards Store: http://www.cssinfo.com/ncits.html

[6] ISO/IEC JTC 1/SC 32 (primarly WG 3, WG 4, and WG 5) archives: ftp://www.sqlstandards.org/SC32/