SQL/MM Spatial: The Standard to Manage Spatial Data in Relational Database Systems

Knut Stolze

Friedrich–Schiller–University Jena
Database and Information Systems Group
Ernst–Abbe–Platz 1–4
07743 Jena, Germany

stolze@informatik.uni-jena.de

IBM Entwicklung GmbH
DB2 Extenders Development
Schönaicher Str. 220
71032 Böblingen, Germany

stolze@de.ibm.com

Abstract: Several major database systems provide extensions to support the management and analysis of spatial data in a relational database system [IBM02, Ora01, IBM01]. The functionality is also standardized in ISO/IEC 13249 SQL/MM. This paper presents part 3 of the standard and discusses it critically. The spatial data types and methods on these types are explained. The Information Schema, showing spatial columns and spatial reference systems, is an important part for the handling of spatial data. It is described in the paper as well.

1 Introduction

ISO/IEC 13249 SQL/MM is the effort to standardize extensions for multi-media and application-speci c packages in SQL. SQL, as de ned in [ISO99], is extended to manage data like texts, still images, spatial data, or to perform data mining. The standard is grouped into several parts.

Part 1 is the framework for all the subsequent parts and de nes the de nitional mechanisms and conventions used in the other parts as well the common requirements that an implementation[1] has to adhere to if it wants to support any one of the extensions de ned in the standard. Part 2 is the full-text standard, which is concerned about the mechanisms to provide extended text search capabilities, above and beyond the operators provided by SQL, e. g. the LIKE predicate. Part 5 de nes the functionality to manage still images, and part 6 is concerned with data mining. The withdrawn part 4 addressed general purpose facilities.

ISO/IEC 13249-3 SQL/MM Part 3: Spatial [ISO02c] is the international standard that de nes how to store, retrieve and process spatial data using SQL. It de nes how spatial data is to be represented as values, and which functions are available to convert, compare, and process this data in various ways.

[1] The SQL standards use the term *implementation* to refer to a program that implements the interfaces defined by the standard. Commonly, an implementation is a relational database system.

The first version of the standard was published in 1999. In the years since then, several enhancements were added to the document, and the second version is now available as Final Draft International Standard (FDIS). It is expected that it will be published as International Standard (IS) in the near future.

In this paper, geometries like points, lines, and polygons or composites thereof are also referred to as *spatial data*. Geometries can be used in many different application domains. The model as specified by the SQL/MM standard is applicable to a variety of different data spaces. The spatial reference system associated for a geometry identifies the data space used for that geometry. For example, it defines whether a geometry models a geographic feature or some other, more abstract feature.

The most common case where spatial data is used in practice are geographic information systems (GIS). There, the term *geometry* is used to denote the geometric features that cartographers have used for the past centuries to map the world. An abstract definition of the meaning of geographic geometry is a point or aggregate of points representing a feature on the ground. Thus, a geometry is usually a model of a geographic feature. The model can be expressed in terms of the feature's coordinates. The model conveys information; for example, the coordinates identify the position of the feature with respect to fixed points of reference.

In the non-geographic applications, a geometry can identify, for example, a feature in a still image where no relation to the earth can be established. Another example are locations inside a grocery store. Although a relation to the earth could be computed based on the latitudes and longitudes, a preferred representation for an application might only refer to the locations with respect to a fixed point in the store, e. g. the south-east corner.

The SQL/MM standard is divided into clauses. The clauses 5 thru 9 describe the geometry types and the methods provided for each type. The Information Schema, based on a Definition Schema is defined in clause 14. The remaining clauses, which are not described in this paper, explain the underlying spatial concepts, the angles and direction handling, and the States codes and conformance rules for products that implement the standard.

Several products exist, which implement spatial extensions for relational database systems. For example, the DB2 Spatial Extender is available for IBM's DB2 Universial Database (UDB), the IDS Spatial DataBlade and Geodetic DataBlade for IBM's Informix Dynamic Server (IDS), and Oracle offers the Oracle 9*i* Spatial product. At should be noted that the list given here only shows the more commonly known products and additional products can be found as well.

The paper explains and discusses the content of the main clauses in the standard in more detail. Section 2 shows which spatial data types exist, how they are organized, and which functionality is provided for each type. Two short examples show how to use the spatial functionality in a relational database in section 3. Like any other part of SQL/MM, the Spatial standard contains an Information Schema. The views in that Information Schema are listed in section 4. The paper is completed with a summary and outlook on the possible future developments of the standard in section 5.

2 Spatial Data Types

2.1 History

The roots of the SQL/MM Spatial standard are directly apparent in the type hierarchy. The standard was originally derived from the OpenGIS Simple Features Speci cation for SQL [OGC99], also published in the year 1999 as version 1.1 by the OpenGIS Consortium (OGC). The Simple Feature Speci cation de nes a so-called Geometry Model . The geometry model consists of a class hierarchy, which is shown in gure 1.

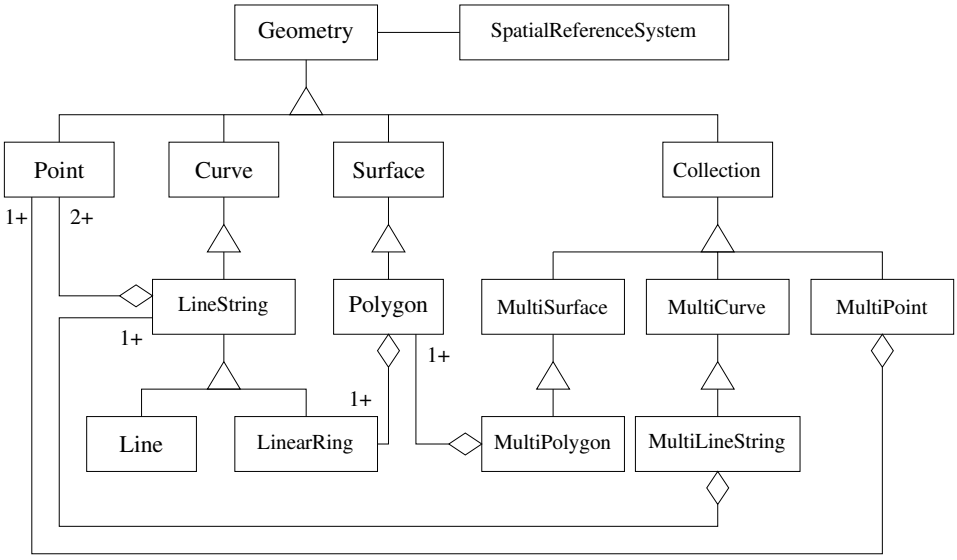


Figure 1: OpenGIS Geometry Class Hierarchy

The geometry model is an abstract model. It is used to de ne the relationship between the various classes and to establish the inheritance rules for the methods working on the instances of the classes and subclasses. For example, the method *Area* is de ned for the class *Surface* and is available for all instances of *Surface*, *Polygon*, and further subclasses, whereas the method *ExteriorRing* is only de ned on the subclass *Polygon* and, thus, cannot be used for arbitrary instances of the class *Surface*.

The SQL/MM standard uses consistently the pre x `ST_` for all tables, views, types, methods, and function names. The pre x stood originally for *Spatial* and *Temporal*. It was intended in the early stages of the standard development to de ne a combination of temporal and spatial extension. A reason for that was that spatial information is very often tied with temporal data [SWCD98, SWCD97, RA01, TJS97]. During the development of SQL/MM Spatial, it was decided that temporal has a broader scope beyond the spatial application and should be a part of the SQL standard [ISO99] as SQL/Temporal [ISO01]. The con-

tributors to SQL/MM did not want to move forward with a Spatio-temporal support until SQL/Temporal developed.[2] In the mean time, thefocus of spatial standard lied on keeping it aligned with the OGC speci cation and the standards developed by the technical commitee ISO/TC 211, for example [ISO02a, ISO02b]. The pre x `ST_` for the spatial tables, types, and methods was not changed during the organizational changes of the standards, however. Today, one might want to interpret it as *Spatial Type*.

2.2 Geometry Type Hierarchy

The OGC geometry class hierarchy is adapted for the corresponding SQL type hierarchy that is de ned in the SQL/MM standard. Figure 2 shows the standardized type hierarchy. The shaded types are the not-instantiable types.[3] All types are used to represent geometric features in the 2-dimensional space ($\mathbf{R}^2$).

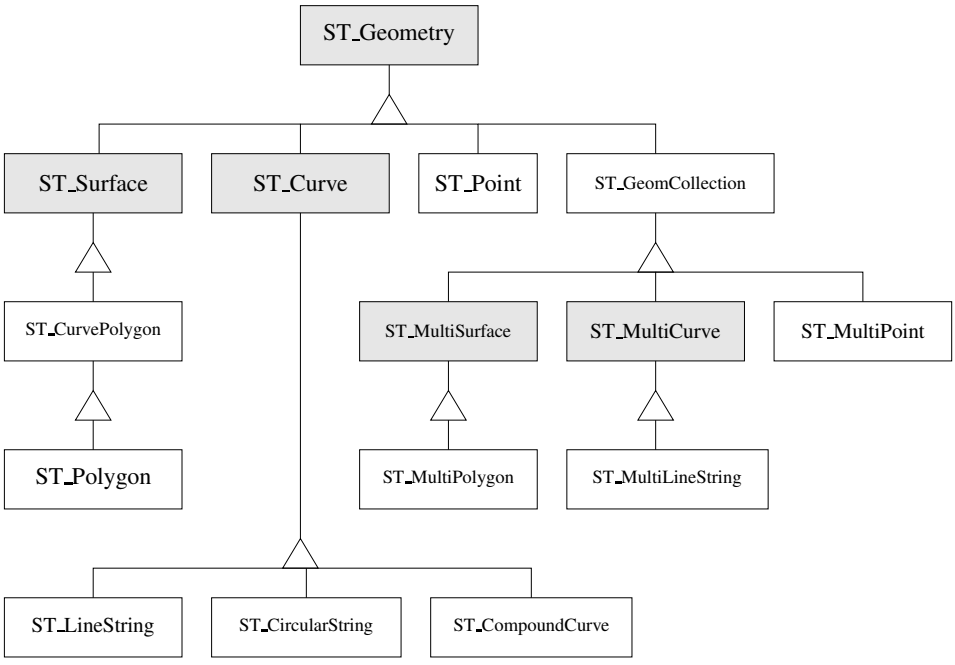


Figure 2: SQL Type Hierarchy

The major differences between the SQL type hierarchy and the OGC geometry class hierarchy are the omission of the derived types *Line* and *LinearRing*, and the addition

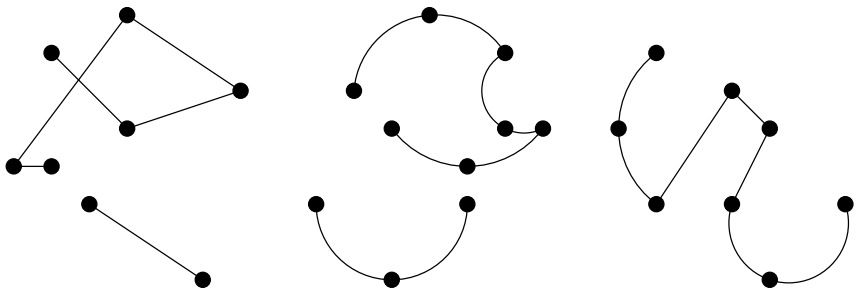
[2] SQL/Temporal was not any further developed and, like SQL/MM Part, subsequently withdrawn completely.

[3] It is implementation-defined whether *ST_MultiCurve* and *ST_MultiSurface* are instantiable or not, even though they are shown as not-instantiable in figure 2.

of a series of types. Lines and linear rings are to be represented using values of type *ST_LineString*, which covers both cases. The new types extend the OGC geometry class hierarchy with circular arcs as curves and surfaces that have circular arcs as their boundary. Furthermore, the aggregations that re ect which types are used by other types are not shown. For example, it is not obvious from the SQL type hierarchy that the type *ST_MultiPoint* consists of *ST_Point* values.

ST_Point values are 0-dimensional geometries and represent only a single location. Each point consists of an X and a Y coordinate to identify the location in the respective spatial reference system. Points can be used to model small real-world objects like lamp posts or wells. *ST_MultiPoint* values stand for a collection of single points. The points in a multi-point do not necessarily have to be distinct points. That means a multi-point supports sets as in the strict mathematical sense, but also allows for multi-set like SQL does in general.

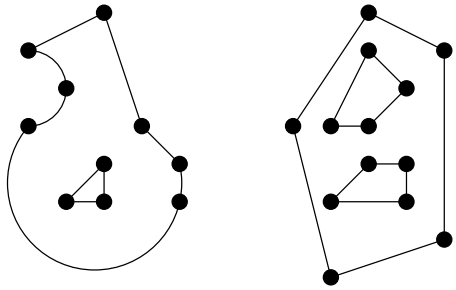
Curves are 1-dimensional geometries. The standard distingiushes between *ST_LineString*, *ST_CircularString*, and *ST_CompoundCurve*. An *ST_LineString* is de ned by a sequence of points, (X, Y) pairs, which de ne the reference points of the line string. Linear interpolation between the reference points de nes the resulting linestring. That means, two consecutive points de ne a line segment in the linear string. Circular instead of linear interpolation is used for *ST_CircularString* values. Each circular arc segment consists of three points. The rst point de nes the start point of the arc, the second is any point on the arc, other than the start or end point, and the third point is the end point of the arc. If there is more than one arc in the circular string, the end point of one arc acts as the start point of the next arc. A combination of linear and circular strings can be modeled using the *ST_CompoundCurve* type. Line segments and circular segments can be concatenated into a single curve. *ST_MultiCurve* values represent a multi-set of *ST_Curve*, and *ST_MultiLineString* a multi-set of *ST_LineString* values. Note that there are no types *ST_MultiCircularString* and *ST_MultiCompoundString*. Figure 3 illustrates some examples of the three different types of curves.



(a) Linear Strings (b) Circular Strings (c) Compounds

Figure 3: Examples of Curves

Surfaces, as 2-dimensional geometries, are de ned in the same way as curves using a sequence of points. The boundary of each surface is a curve, or a set of curves if the surface has any holes in it. The boundary of a surface consists of a set of rings, where each ring is a curve. The type *ST_CurvePolygon* stands for such a generalized surface, and the subtype *ST_Polygon* restricts the conditions for the rings of the boundary to linear strings. The types *ST_MultiSurface* and *ST_MultiPolygon* are used to model sets of curve polygons or polygons with linear boundaries.[4] A curve polygon and a polygon with linear strings as its boundary are shown in gure 4.



(a) Curve Polygon (b) Polygon

Figure 4: Examples of Polygons

The type hierarchy does not address the concept of empty geometries in form of a separate type under *ST_Geometry*. An empty geometry represent an empty set of points, and it can be the result of the intersection of two disjoint polygons. The standard allows that a value of each of the instantiable types can be an empty geometry. Thus, empty points, empty linestrings, empty polygons, and empty geometry collections etc. exist. If a method has the return an empty geometry as its result and the most speci c type is not inherent by the method, then an empty point is generated. Implicit or explicit casts can be used to convert empty geometries from one type to another.

2.3 Methods

The majority of all spatial methods can be grouped into one of the following four categories:

- convert between geometries and external data formats,
- retrieve properties or measures from a geometry,
- compare two geometries with respect to their spatial relation,

[4] *ST_MultiSurface* constrains its values to contain only disjoint surfaces.

- generate new geometries from others

Examples and descriptions for each of the categories are given in this section. Sometimes, it is not trivial to assign a spatial method to only a single category. For instance, the method *ST_StartPoint*, which returns the first point of a linestring, retrieves a property of the linestring, i. e. the first point, but it also generates a new geometry, i. e. an *ST_Point* value.

2.3.1 Convert to and from External Data Formats

The SQL/MM standard defines three external data formats that can be used to represent geometries in an implementation-independent fashion.

- well-known text representation (WKT)
- well-known binary representation (WKB)
- geography markup language (GML)

Each type implements constructor methods that allow to generate a new geometry from the given WKT or WKB and the, optionally, provided numeric spatial reference system identifier. All instantiable types have such constructor methods. There are no constructor methods that cope with the GML representation. Functions like *ST_LineFromGML* or *ST_MPointFromGML* are used instead.

For backward compatibility, the standard also defines functions like *ST_PointFromText* or *ST_GeometryFromWKB* with exactly the same purpose as the constructor methods. Those functions were inherited from and remain for compatibility with the OpenGIS Simple Feature Specification for SQL [OGC99]. The constructor methods were introduced later in the process of the standard development to align part 3 of SQL/MM with other parts, and also to improve the overall usability of the constructors.

The three methods *ST_AsText*, *ST_AsBinary*, and *ST_AsGML* are provided for the conversion of a geometry to the respective external data format.

2.3.2 Retrieve Properties

All geometries have certain properties. A property is, for example, the dimension or if a geometry is empty. Each of the subtypes adds further, more specific properties, for example, the area of a polygon or whether a curve is simple[5]. A set of method was defined to query those properties. Due to the high number of available methods, only a small set of examples is given here.

ST_Boundary return the boundary of a geometry

[5] A *simple* curve is defined to be not self-intersecting.

ST_IsValid test whether a geometry is valid, i. e. correctly de ned; an invalid geometry could be a not-closed polygon

ST_IsEmpty test whether a geometry is empty

ST_X return the X coordinate of a point

ST_IsRing test whether a curve is a ring, i. e. the curve is closed and simple

ST_Length return the length for a linestring or multi-linestring

2.3.3 Compare Two Geometries

A very interesting part in spatial queries comes from the comparison of geometries. Questions like which buildings are in a ood zone or where are intersections of rail roads and streets can only be answered if the geometries representing the buildings, ood zones, rail roads, and streets are compared with each other.

The standard de nes the following set of methods to compare geometries in various ways.

ST_Equals test the spatial equality of two geometry

ST_Disjoint test whether two geometries do not intersect

ST_Intersects**, *****ST_Crosses*****, and *****ST_Overlaps*** test whether the interiors of the geometries intersect

ST_Touches test whether two geometries touch at their boundaries, but do not intersect in their interiors

ST_Within **and** ***ST_Contains*** test whether one geometry is fully within the other

All of the above methods return an INTEGER value, which is 1 (one) if the spatial relation does exist, and 0 (zero) otherwise.

Additionally, the method *ST_Distance* exists, which quanti es the spatial relationship of two geometries according to their distance.

2.3.4 Generate New Geometries

The last set of methods allows the user to generate new geometries from existing ones. A newly generated geometry can be the result of a set operation on the set of points represented by each geometry, or it can be calculated by some algorithm applied to a single geometry. The following methods are examples for both.

ST_Buffer generate a buffer at a speci c distance around the given geometry

ST_ConvexHull compute the convex hull for a geometry

ST_Difference**, *****ST_Intersection*****, and *****ST_Union*** construct the difference, intersection, or union between the point sets de ned by two geometries

2.4 Discussion

An observation of the type hierarchy as de ned by the SQL/MM standard raises several questions on the design decisions that were made.

2.4.1 OGC Geometry Model

The OGC geometry class hierarchy attempts to implement the *composite* design pattern [GHJV95] by adding the class *Collection*. Without the subclasses under *Collection* and an additional aggregation of *Geometry*, the pattern would t exactly. A simpli cation of the type hierarchy could have been achieved with the omission of all the subclasses of *Collection* without any loss of functionality. All the methods that are de ned on the subclasses of *Collection* have the same simple logic. The same method is called for each element (also called *part*) of the collection, and the results are combined.

Another drawback of this way of modeling the classes is that a future extension of the geometry model requires the handling of the new geometries in two different places. First, the new class has to be added under *Geometry*, and second, a corresponding class has to be added for sets of such a geometry under *Collection*.

2.4.2 SQL Type Hierarchy

A different picture shows the implementation of the OGC geometry class hierarchy as the SQL/MM type hierarchy. SQL is a set-oriented language. An important goal of a standard should be the usability of the functionality de ned. Iterating over the elements of a collection as discussed above is, although possible, not as simple in SQL. A dynamic compound statement or a recursive query has to be used for the iteration. The respective method on the element of a collection has to be invoked during the iteration, and the results are to be combined. Leaving that task up to the user will only lead to the user de ning those functions himself to prevent the repetition of that task.

Furthermore, consider the following typical user scenario: The SQL query is supposed to return all the parts of the geometries in column `spatial_column` that fall into a certain rectangle. The rectangle in question is represented using a polygon.

```
SELECT ST_Intersection(ST_Polygon(
            'polygon((10 10, 10 20, 20 20, 20 10,
                      10 10)), 1), spatial_column)
FROM   spatial_table
WHERE  ...
```

Assuming that the column contains values of type *ST_MultiPoint*, the results of the query can contain any of the following for each row:

- an empty geometry (empty point)

- a single point
- multi-points

In the above query, the results can be retrieved and visualized on the screen. Further processing the results in an SQL statement is, however, not completely trivial. Because *ST_Point* and *ST_MultiPoint* are in two independent subtrees of the type hierarchy, only the methods available on the common ancestor *ST_Geometry* can be used. A conversion of the single points to multi-points with only a single element is not supported by the standard. It only supports the conversion of empty geometries from one type to another.

A solution to address this issue might be to set the types *ST_Point* and *ST_MultiPoint* in direct relationship by using inheritance. Interpreting that in the context of the SQL type hierarchy gives: *ST_MultiPoint* is a specialized *ST_GeomCollection*, which only contains points, and *ST_Point* is a specialized *ST_MultiPoint*, consisting of only a single point. A similary approach can be implemented for the other types, which are not in the subtree under *ST_GeomCollection*. Figure 5 shows how such a type hierarchy could be de ned.

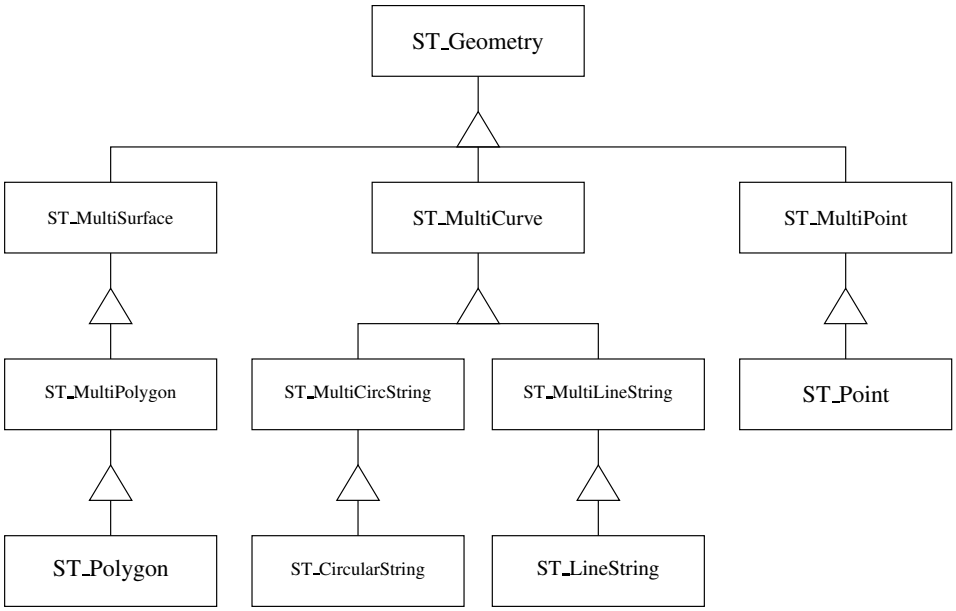


Figure 5: Modified Type Hierarchy

A similar idea was apparently the bases for the type hierarchy implemented in the SpatialWare DataBlade for IDS product [Map02]. The DB2 Spatial Extender de nes a type hierarchy exactly as in the standard, but it omits some of the optional data types [IBM02]. And yet a third approach to handle geometries stems from the Oracle 9*i* Spatial product [Ora01]. It does not attempt to model a type hierarchy to re ect more speci c properties of the different kinds of geometries but uses a single type *SDO_Geometry* instead. No

support for strong typing based on the geometries can be enforced in such an environment, but other means have to be used.

2.4.3 Methods on the Geometry Types

The SQL/MM standard provides a rich set of methods and functions. But it can be noted that some additions and changes would result in a further improvement.

The standard de nes constructor methods that can handle well-known text (WKT) and well-known binary (WKB) representations. It does not allow for a handling of the GML representation in a constructor, however. The existing constructor methods for the well-known text representation could be reused for that, given that WKT and GML are both a textual representation for geometries and can easily be distinguished be analyzing the very rst non-whitespace character. The functions like *ST_PolyFromGML* could then be removed. They are not de ned in the OGC Simple Feature speci cation, so that compatibility issues do not arise.

A set of functions allows the user to construct any geometry using one of the external data formats. Those functions act like factory functions and are named *ST_GeomFromText*, *ST_GeomFromWKB*, and *ST_GeomFromGML*. Instead of using the explicit names to denote the format handled by each function, an approach similar to the constructor methods is preferrable. An overloaded function *ST_Geometry* can be de ned with the same semantical behaviour as the existing functions. Note that constructor methods on the type *ST_Geometry* cannot be used because that type is not instantiable.

There are several groups of methods that provide (nearly) identical functionality. For example, the methods *ST_Intersects*, *ST_Crosses*, and *ST_Overlaps* all test for intersections of the interiors of the two input geometries. The only difference between the methods is that *ST_Crosses* does not allow to test if a surface intersects some other geometry, or if some other geometry intersects a point. *ST_Overlaps* requires that both geometries to be compared have the same dimension. For example, a line can overlap another line but not a polygon. *ST_Intersects* is the generalized version of the functionality to test for the overlay of geometries. It does not impose any restrictions in its input parameters. The existence of *ST_Crosses* and *ST_Overlaps* is rather questionable.

Another omission can be found in the support for external data formats. The de-facto industry standard to represent geometries is the so-called shape format [ESR97]. The shape format is not supported by the standard, but it should be considered given that existing major products already support it [IBM02, IBM01].

3 User scenarios

Two user scenarios for spatial functionality as de ned in the standard as shown in this section. The rst scenario given in section 3.1. The second scenario describes how a bank can manage its customers and make decisions on the placement of new branches can be found in section 3.2.

3.1 Insurance Company

After a recent ooding, an insurance company wants to correct the information about insured buildings that are in the ood zone, and pose an increased risk for the company. The database contains one table `rivers` that contains the rivers and their ood zones and another table `buildings` with the data for the buildings of all the policy holders.

```
rivers(name, water_amount, river_line, flood_zones)
buildings(customer_name, street, city, zip, ground_plot)
```

The column `river_line` contain the linstrings that represent all the rivers in the country. Related to that the column `flood_zones` shows the oodzones for each river. The ground plot of the customer's building is stored in the column `ground_plot`. The tables for the above shown relational model can be created with the following SQL statements.

```
CREATE TABLE rivers (
   name          VARCHAR(30)  PRIMARY KEY,
   water_amount  DOUBLE PRECISION,
   river_line    ST_LineString,
   flood_zones   ST_MultiPolygon )

CREATE TABLE buildings (
   customer_name   VARCHAR(50)  PRIMARY KEY,
   street          VARCHAR(50),
   city            VARCHAR(20),
   zip             VARCHAR(10),
   ground_plot     ST_Polygon )
```

The rst task is to update the information about the ood zones. The ood zones for the river FLOOD is to be extended by 2 kilometers in each direction. The method *ST_Buffer* is used in the following SQL statement to extend the ood zones by the speci ed radius.

```
UPDATE rivers
SET    flood_zones =
          flood_zones.ST_Buffer(2, 'KILOMETER')
WHERE  name = 'FLOOD'
```

In the next step, the company wants to nd all the customers that are now in the extended ood zone for the river. An SQL statement involving the spatial method *ST_Overlaps* can be used to to nd all those buildings.

```
SELECT customer_name, street, city, zip
FROM   buildings AS b, rivers AS r
WHERE  b.ground_plot.ST_Within(r.flood_zones) = 1
```

The so retrieved addresses can be further processed, and the customers can be informed of any changes to their policy or other information.

3.2 Banking

A bank manages its customers and branches. Each customer can have one or more accounts, and each account is managed by a branch of the bank. To improve the quality of services, the bank performs an analysis of its customers, which also involves a spatial component, the locations of the customer's homes and the branches. The tables in the bank's data base have the following de nition.

```
CREATE TABLE customers (
   customer_id  INTEGER
      PRIMARY KEY,
   name         VARCHAR(20),
   street       VARCHAR(25),
   city         VARCHAR(10),
   state        VARCHAR(2),
   zip          VARCHAR(5),
   type         VARCHAR(10),
   location     ST_Point);
```

```
CREATE TABLE branches (
   branch_id  INTEGER
      PRIMARY KEY,
   name       VARCHAR(12),
   manager    VARCHAR(20),
   street     VARCHAR(20),
   city       VARCHAR(10),
   state      VARCHAR(2),
   zip        VARCHAR(5),
   location   ST_Point,
   zone       ST_Polygon);
```

```
CREATE TABLE accounts (
   account_id   INTEGER  PRIMARY KEY,
   routing_no   INTEGER  NOT NULL,
   customer_id  INTEGER  NOT NULL,
   branch_id    INTEGER  NOT NULL,
   type         VARCHAR(10)  NOT NULL,
   balance      DECIMAL(14, 2)  NOT NULL,
   CONSTRAINT fk_customers FOREIGN KEY(customer_id)
      REFERENCES customers(customer_id),
   CONSTRAINT fk_branches FOREIGN KEY(branch_id)
      REFERENCES branches(branch_id) );
```

The rst query determines all customers with an account balance larger than $10,000.- in any of the accounts and who live more than 20 miles away from their branch.

```
SELECT DISTINCT c.customer_id, c.name
FROM   customers AS c JOIN accounts AS a ON
          ( c.customer_id = a.customer_id )
WHERE  a.balance > 10000 AND
       a.location.ST_Distance(
          ( SELECT b.location
            FROM   branches
            WHERE  b.branch_id = a.branch_id ),
          'MILES') > 20
```

The bank wants to find all the portions of the assigned sales zones of the branches that overlap. It is not intended to have more than one branch assigned to a certain area, and any duplicates are to be found and corrected. The query retrieves the identifiers for each two branches that have an overlap in the zones and also the overlapping area, encoded in the well-known text representation.

```
SELECT b1.branch_id, b2.branch_id,
       b1.zone.ST_Overlaps(b2.zone).ST_AsText()
FROM   branches AS b1 JOIN branches AS b2 ON
          ( b1.branch_id < b2.branch_id )
WHERE  b1.zone.ST_Overlaps(b2.zone).ST_IsEmpty() = 0
```

To reduce competition amoungst the branches for its own customers, the bank wants to find all the customers that live within a 10 mile radius of a branch, which does not manage their accounts. The accounts are to be transferred to a closer branch if the customer agrees.

```
SELECT c.name, c.phone, b.branch_id
FROM   branches AS b, customers AS c
WHERE  b.location.ST_Buffer(10, 'MILES').
          ST_Contains(c.location) = 1 AND
       NOT EXISTS (
          SELECT 1
          FROM   accounts AS a
          WHERE  a.customer_id = c.customer_id AND
                 a.branch_id = b.branch_id )
```

4 Information Schema

SQL/MM Part 3: Spatial defines an Information Schema to provide an application that uses the spatial extension with a mechanism to determine the supported and available features. The Information Schema consists of four views, which are explained here, after a short introduction on the history.

4.1 History

The SQL/MM Information Schema was also inherited from the OGC Simple Feature Specification for SQL [OGC99]. The OGC specification used ad still uses the views `GEOMETRY_COLUMNS` and `SPATIAL_REF_SYS` with a different set of columns and different semantics of the data shown in the view.

The OGC specification describes two completely different concepts, called environments, that can be used for the implementation of a spatial extension for a database system, based on whether structured types, so-called ADTs, are supported or not. The Information

Schema for the environment without structured types is more complex because additional information has to be maintained.

For the second environment that exploits the structured type support, which is in alignment with the SQL/MM standard, the OGC specification refers in the description for the `GEOMETRY_COLUMNS` view only back to the other environment and states: the columns in the GEOMETRY_COLUMNS metadata view for the SQL92 with Geometry Types envirconment are a subset of the columns in the GEOMETRY_COLUMNS view defined for the SQL92 environment. It is not exactly clear what the subset is supposed to be.

The original Information Schema defined in the SQL/MM standard was based on those information. For example, the view `GEOMETRY_COLUMNS` included a column named `COORD_DIMENSION`, which has no well-defined meaning when storing arbitrary geometries on a spatial column. The original views were replaced by a new definition of the Information Schema in the second edition of the SQL/MM standard. The following sections refer to the newly introduced views.

4.2 SQL/MM Spatial Information Schema

The Spatial Information Schema consists of 4 views that list the spatial columns, the supported spatial reference systems, the units of measure, and the implementation-defined meta-variables. The entity-relation-ship diagram in figure 6 shows those views and also their relationship to the views defined in the Information Schema in [ISO99].

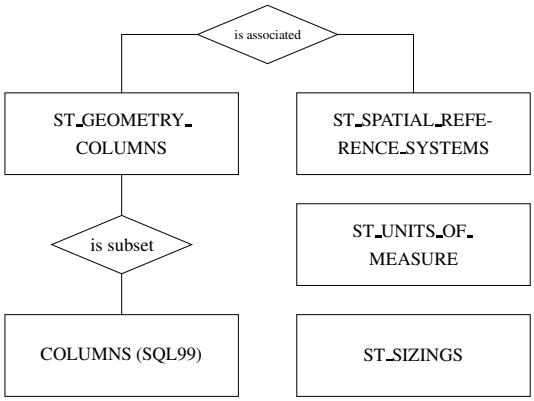


Figure 6: Spatial Information Schema

ST_GEOMETRY_COLUMNS The view lists all columns in all tables that have a declared type of *ST_Geometry* or one of its subtypes. It is not necessary to associate a specific spatial reference system with a column, but an application can do so. The view shows the

column identifier, consisting of catalog, schema, table, and column name, and the identifying name and the numeric identifier of the spatial reference system associated with the column.

The view is written in such a way to query the view `COLUMNS` from the SQL Information Schema to retrieve the infermation about all existing spatial columns and then merges the SRS information for each of the columns that has an associated spatial reference system using an outer join.

ST_SPATIAL_REFERENCE_SYSTEMS A spatial reference system has two unique identifiers, a name and a numeric identifier. The name is used in the same way as for all other SQL objects like schemata, functions, or columns. The numeric identifier is used in the methods that require the SRS information as input, for instance to construct a geometry in a specific SRS or to transform a geometry to another SRS.

Along with the identifiers, the view represents the organization that defined this spatial reference system together with the identifier assigned by that organization and the actual definition of the SRS.

ST_UNITS_OF_MEASURE Different units can be used to calculate distances between geometries, the length of curves, or the area of surfaces. The view lists those units that are supported. An identifying name, the type of the unit (angular or linear), and the conversion factor to the base unit within each type is stored. The conversion factor for the base units in each type are always 1 (one).

ST_SIZINGS Similar to the SQL99 Information Schema view `SIZINGS`, the SQL/MM standard requires that an implementation creates a view `ST_SIZINGS`. This view contains the spatial-specific meta-variables and their values. An example of a meta-variable is the maximum possible length that can be used for a well-known text representation of a geometry. This meta-variable is called *ST_MaxGeometryAsText*.

4.3 Discussion

Two views in the Information Schema should be reconsidered because their current definition is not adequate. First, the view `ST_SPATIAL_REFERENCE_SYSTEMS` uses a very simplified way to manage spatial reference systems. The European Petrol Survey Group (EPSG) worked on a more expressive schema for spatial reference systems, which is also described in another ISO standard [ISO02b], although in a different context.

The view `ST_SIZINGS` has the same intention as the view `SIZINGS` defined in SQL99 [ISO99], only specialized for the facilities in SQL/MM Spatial. If SQL99 would provide a mechanism for implementations of other standards, including SQL/MM to add new entries to its view, then `ST_SIZINGS` becomes obsolete and could be removed from the SQL/MM

standard. Unfortunately, SQL99 does not describe the requested mechanisms, so both views are still necessary.

5 Summary and Outlook

SQL/MM Part 3: Spatial standardizes the storing, retrieving and processing of spatial data as part of a relational database system. It de nes a set of types and methods for the representation of 0, 1, or 2-dimensional geographic features.

The SQL/MM standard is expected to be published in its second version in the near future. The second version shows improvements for the Information Schema and adds the support for the Geography Markup Language (GML) and angles and directions. It also de nes many functions in a more precise manner.

This paper described the facilities de ned in the standard, and discussed them critically. The implemented type hierarchy is suitable for its purpose, but together with the strong typing and the set-oriented data management imposed be SQL, it inconveniences the spatial data processing. The methods provided for each type cover a wide range of spatial functionality. The multitude of methods lead to the situation where functionality is already duplicated (with very minor differences).

Some future directions of the SQL/MM Spatial standard already show in the new working draft that was started recently and will eventually become the third version of the standard. The Japanese standards committee supplied a change proposal that adds a function *ST_ShortestPath*, which calculates the shortest path in a network (or graph) of linestrings between two given points.

Additional functionality that implements more spatial oriented logic could be included as well in the future. For example, a function *ST_Nearest* to nd for a given geometry the nearest one from another set of geometries is desirable. The user cousd exploit it to get the answers to questions like nd me the closest restaurant to my current location .

The support for modi cation of geometries directly in the database system using spatial methods can be extended. There are no simple functions to change a point in a linestring, or to generalize geometries if it is too detailed, i. e. too many points are used to de ne it. Existing products already support methods like *SE_ChangeVertex* or *SE_Generalize* [IBM01].

Open questions also remain with respect to other SQL standards. For example, the Information Schema de ned in the SQL/MM Spatial standard contains information about the spatial reference systems applicable for geometries stored in the same database as the Information Schema. The access to external data stores using SQL/MED [ISO00] is not considered today.

The support for raster data, for example huge images taken for whole countries, imposes special requirements on a spatial database. Geographical Information Systems (GIS) rely on raster data to provide additional information for the user. The current SQL/MM standard does not consider raster data at all, and the needed infrastructure should be de ned.

Literaturverzeichnis

[ESR97] Environmental Systems Research Institute, Inc. *ESRI Shapefile Technical Description*, 1997.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissidis. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[IBM01] International Business Machines, Corp. *Informix Spatial DataBlade, Version 8.11*, 2001.

[IBM02] International Business Machines, Corp. *DB2 Spatial Extender – User's Guide and Reference, Version 8.1*, 2002.

[ISO99] ISO/IEC 9075-2:1999. *Information Technology – Database Languages – SQL – Part 2: Foundation (SQL/Foundation)*, 1999.

[ISO00] ISO/IEC 9075-9:2000. *Information Technology – Database Languages – SQL – Part 9: SQL/MED*, 2000.

[ISO01] ISO/IEC 9075-2:2001 WD. *Information Technology – Database Languages – SQL – Part 7: Temporal (SQL/Foundation)*, 2001.

[ISO02a] ISO/DIS 19107:2002. *Geographic Information - Spatial Schema*, 2002.

[ISO02b] ISO/DIS 19111:2002. *Geographic Information - Spatial Referencing by Coordinates*, 2002.

[ISO02c] ISO/IEC 13249-3:2002 FDIS. *Information technology – Database languages – SQL Multimedia and Application Packages – Part 3: Spatial*, 2nd edition, 2002.

[Map02] MapInfo, Corp. *MapInfo SpatialWare – User's Guide, Version 4.5*, 2002.

[OGC99] OpenGIS Consortium. *OpenGIS Simple Features Specification for SQL, Revision 1.1*, 1999.

[Ora01] Oracle, Corp. *Oracle Spatial User's Guide and Reference, Release 9.0.1*, 2001.

[RA01] K. H. Ryu and Y. A. Ahn. Application of Moving Objects and Spatiotemporal Reasoning. Technical report, TimeCenter, 2001.

[SWCD97] P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and Querying Moving Objects. In *Proceedings of the Thirteenth International Conference on Data Engineering (ICDE13)*, Birmingham, UK, 1997.

[SWCD98] P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. *Temporal Databases: Research and Practice*, chapter Querying the Uncertain Position of Moving Objects. Springer-Verlage, 1998.

[TJS97] V. J. Tsotras, C. S. Jensen, and R. T. Snodgrass. A Notation for Spatiotemporal Queries. Technical report, TimeCenter, 1997.