

ISO/IEC JTC 1/SC 21 N 11158

Date: 1997-10-21

ISO/IEC FCD 13249-3:199x (E)

ISO/IEC JTC 1/SC 21/WG 3

Secretariat: Canada

Information Technology — Database Languages —

SQL Multimedia and Application Packages —

Part 3: Spatial

Document type: International Standard
Document subtype: Not applicable
Document stage: (30) Committee
Document language: E

Contents	Page
Foreword.....	viii
Introduction	ix
1 Scope.....	1
2 Normative references.....	2
3 Definitions, notations, and conventions	3
3.1 Definitions	3
3.2 Notations	4
3.3 Conventions	5
3.3.1 Subclause structure.....	5
3.3.2 Structure within a Subclause.....	5
3.3.3 Data Type, Attribute and SQL-invoked Routine Identifiers	5
3.3.4 Parameter Identifiers.....	6
3.3.5 Meta-variables	6
3.3.6 Definitional variables	6
3.3.7 Exceptions.....	6
4 Concepts	7
4.1 USAGE Privileges on User-defined Types	7
4.2 UNDER Privileges on User-defined Types	7
4.3 EXECUTE Privileges on SQL-Invoked Routines	7
4.4 Relationships of Spatial Values.....	7
4.5 Area Data Types	9
4.5.1 ST_Region Type	9
5 Spatial Data Types	10
5.1 ST_Spatial Type and Routines	10

5.1.1	ST_Spatial Type	10
5.1.2	ST_Spatial Functions	12
5.1.3	ST_SpatialReferencing Function	13
5.1.4	ST_Transform Function	14
5.1.5	ST_Buffer Function	15
5.1.6	ST_Centroid Function	16
5.1.7	ST_Envelope Function	17
5.1.8	ST_Length Function	18
5.1.9	ST_Area Function	19
5.1.10	ST_Perimeter Function	20
5.1.11	ST_Azimuth Function	21
5.1.12	ST_Distance Function	22
5.1.13	ST_Overlaps Function	23
5.1.14	ST_Meets Function	24
5.1.15	ST_Contains Function	25
5.1.16	ST_ContainedBy Function	26
5.1.17	ST_Outside Function	27
5.1.18	ST_ExtractAt Function	28
5.1.19	ST_Count Function	29
5.1.20	ST_Intersection Function	30
5.1.21	ST_Union Function	31
5.1.22	ST_Difference Function	32
5.1.23	ST_Decompose Function	33
5.1.24	ST_Dimension Function	34
6	Spatial Referencing Data Types	35
6.1	ST_SpatialReferencing Type and Routines	35
6.1.1	ST_SpatialReferencing Type	35

6.1.2 ST_SpatialReferencing Function..... 36

6.1.3 ST_Name Function..... 36

6.1.4 ST_IsEqual Function..... 37

7 Coordinates Data Types 38

7.1 ST_Coordinates Type and Routines..... 38

7.1.1 ST_Coordinates Type..... 38

7.1.2 ST_Coordinates Functions 40

7.1.3 ST_X Functions 41

7.1.4 ST_Y Functions 42

7.1.5 ST_Z Functions 43

7.1.6 ST_Is2D Function..... 44

7.1.7 ST_Is3D Function..... 44

7.1.8 ST_ExplicitCoordinates Function..... 45

8 Geometric Data Types..... 46

8.1 ST_GeometricElement Type and Routines 46

8.1.1 ST_GeometricElement Type 46

8.2 ST_GeometricGroup Type and Routines 47

8.2.1 ST_GeometricGroup Type 47

9 Point Data Types 48

9.1 ST_Point Type and Routines 48

9.1.1 ST_Point Type 48

9.1.2 ST_Coord Functions 49

10 Line Data Types..... 50

10.1 ST_Line Type and Routines..... 50

10.1.1 ST_Line Type..... 50

10.1.2 ST_Length Function..... 53

10.2 ST_Curve Type and Routines 53

10.2.1	ST_Curve Type	53
10.2.2	ST_Curve Function	56
10.2.3	ST_Polyline Function	57
10.2.4	ST_CircularArc Function	58
10.2.5	ST_EllipticalArc Function	59
10.2.6	ST_Points Functions.....	60
10.2.7	ST_CurveType Functions	61
10.2.8	ST_Length Function.....	64
10.2.9	ST_GetPointAt Function	65
10.2.10	ST_InsertPointAt Function	66
10.2.11	ST_AddPoint Function.....	68
10.2.12	ST_DeletePointAt Function	69
10.2.13	ST_UpdatePointAt Function	70
10.3	ST_Path Type and Routines	71
10.3.1	ST_Path Type	71
10.3.2	ST_Path Functions.....	73
10.3.3	ST_Length Function.....	74
10.3.4	ST_GetCurveAt Function	75
10.3.5	ST_InsertCurveAt Function	76
10.3.6	ST_AddCurve Function.....	78
10.3.7	ST_DeleteCurveAt Function	79
10.3.8	ST_UpdateCurveAt Function	80
11	Area Data Types	81
11.1	ST_Area Type and Routines	81
11.1.1	ST_Area Type	81
11.1.2	ST_Area Function.....	81
11.2	ST_Polygon Type and Routines	82

11.2.1	ST_Polygon Type	82
11.2.2	ST_Polygon Functions.....	84
11.2.3	ST_ExteriorBoundary Functions	86
11.2.4	ST_Area Function.....	87
11.2.5	ST_GetInteriorBoundaryAt Function.....	88
11.2.6	ST_InsertInteriorBoundaryAt Function	89
11.2.7	ST_AddInteriorBoundary Function	90
11.2.8	ST_DeleteInteriorBoundaryAt Function	91
11.2.9	ST_UpdateInteriorBoundaryAt Function.....	92
11.3	ST_Region Type and Routines	93
11.3.1	ST_Region Type	93
11.3.2	ST_Region Function	94
11.3.3	ST_Polygons Functions.....	95
11.3.4	ST_PolygonsClosed Function	96
11.3.5	ST_Area Function.....	97
12	Conformance	98
12.1	Introduction	98
12.2	Relationship to other International Standards.....	98
12.3	Claims of conformance.....	98
12.4	Extensions and options	98
13	Status Codes	99
	Index.....	100

Figures	Page
Figure 1 — <i>ST_SelfIntersecting</i> Visual Description	51
Figure 2 — <i>ST_Closed</i> Visual Description.....	51

Tables	Page
Table 1 — Spatial Relations in Extended Domain	8
Table 2 — <i>ST_CurveType</i> encoding.....	62
Table 3 — <i>SQLSTATE</i> class and subclass values	99

Equations	Page
Equation 1 — Enclosure and Intersection Relations	8

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75% of the national bodies casting a vote.

International Standard ISO/IEC SQL/MM was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*.

This document is based on the content of ISO Committee Draft Database Language (SQL).

This is the first edition of ISO/IEC SQL/MM—Part 3: Spatial.

Introduction

The purpose of this International Standard is to define multimedia and application specific types and their associated routines using the user-defined features in ISO/IEC 9075.

SQL/MM is structured as a multi-part standard. At present it consists of the following parts:

Part 1: Framework

Part 2: Full-Text

Part 3: Spatial

Part 4: General Purpose Facilities

Part 5: Still Image

Information Technology — Database Languages — SQL Multimedia and Application Packages — Part 3: Spatial

1 Scope

This part of ISO SQL/MM:

- a) introduces the Spatial part of this International Standard,
- b) gives the references necessary for this part of this International Standard,
- c) defines notations and conventions specific to this part this International Standard,
- d) defines concepts specific to this part of this International Standard,
- e) defines the spatial data types and their associated routines.

The spatial data types defined in this part adhere to the following:

- A spatial data type is generic to spatial data handling. It addresses the need to store, manage and retrieve information based on aspects of spatial data such as geometry, location, and topology.
- A spatial data type does not redefine the database language SQL directly or in combination with another spatial data type.

2 Normative references

The following standards contain provisions that, through reference in this text, constitute provisions to this International Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 9075-1:199x, *Information Technology — Database Languages — SQL — Part 1: Framework (SQL/Framework)*.

ISO/IEC 9075-2:199x, *Information Technology — Database Languages — SQL — Part 2: Foundation (SQL/Foundation)*.

ISO/IEC 9075-4:1996, *Information Technology — Database Languages — SQL — Part 4: Persistent Stored Modules (SQL/PSM)*.

ISO/IEC 13249-1:199x, *Information Technology — Database Languages — SQL Multimedia and Application Packages — Part 1: Framework*.

ISO/IEC 13249-2:199x, *Information Technology — Database Languages — SQL Multimedia and Application Packages — Part 2: Full-Text*.

3 Definitions, notations, and conventions

3.1 Definitions

3.1.1

area

1) A generic term for a bounded, continuous, two-dimensional object that may or may not include its boundary and may be delimited by zero or more non-nested non-intersecting inner boundaries. 2) a measure of a surface.

3.1.2

boundary

Closed one-dimensional non-intersecting element delimiting an area.

3.1.3

buffer

For some distance n , the buffer of a given spatial object is the area determined such that the distance between each point on the boundary of the buffer and the spatial object is n .

3.1.4

centroid

A point which may be considered as the center of a one or two dimensional spatial object.

3.1.5

closed

A line is considered closed if its start and end points are the same.

3.1.6

contains

A spatial object is contained by another spatial object if it is completely included within the other object.

3.1.7

contiguous

An object is contiguous if its elements are in a continuous connection.

3.1.8

coordinates

Pairs of numbers expressing horizontal distance along axes; alternatively, triplets of numbers measuring horizontal and vertical distances.

3.1.9

envelope

A rectangular box oriented along the axes which just barely encompasses the entire object.

3.1.10

geometry

The relationship of points, lines and areas in space from their defining conditions.

3.1.11

line

A generic term for a one-dimensional object.

3.1.12**meets**

Two objects meet if they are contiguous, adjoining, abutting or touching.

3.1.13**overlaps**

An object overlaps if it extends over and covers a part of another object.

3.1.14**point**

A zero dimensional object that specifies a geometric location.

3.1.15**polygon**

A closed planar area.

3.1.16**region**

An area made up of one or more non-overlapping polygons.

3.1.17**routine**

A routine is a <SQL-invoked routine> as defined in ISO/IEC 9075.

3.1.18**spatial function**

Is a function which operates on a spatial object to produce a result. Spatial functions include: buffer, area, perimeter, length, envelop, centroid and separation.

3.1.19**spatial predicate**

Is a spatial function with a boolean return type. These functions can be used as predicates in a SQL query. Spatial predicates include meets, overlaps, contains, contained by, and outside.

3.1.20**spatial referencing**

There are many different coordinate systems that the location can be encoded in. These coordinate systems must be identified so that spatial operations can be performed in the same system.

3.1.21**valid geometry**

A geometry is considered valid if it is well formed and conforms to the geometric rules. Examples of geometry that are not valid are degenerate lines and unclosed polygons.

3.2 Notations

The notation used in ISO/IEC 13249-3 is defined in ISO/IEC 9075-1.

3.3 Conventions

3.3.1 Subclause structure

Subclauses for a type definition and its associated routines will be structured as follows:

x.3.5 ST_SampleDataType Type and Routines

x.3.5.1 ST_SampleDataType Type

x.3.5.2 ST_UserDefinedFunction1 Function

x.3.5.3 ST_UserDefinedFunction2 Function

x.3.5.4 ST_UserDefinedFunction3 Function

3.3.2 Structure within a Subclause

A subclause containing a type or routine definition has the following structure:

- 1) **Purpose:** The Purpose section contains a brief description of the purpose of the type or routine.
- 2) **Definition:** This section contains the ISO/IEC 9075 syntax used to define the type or routine. In the case of routine specifications, the routine body should be defined using the facilities of ISO-9075-4 (SQL/PSM) where possible. This clause is appears in the Courier font. <key word>s, as defined in ISO 9075, are in uppercase. Parameter and variable identifiers are in lowercase. Data type, attribute and SQL-invoked routine identifiers are specified as defined in Subclause 3.3.3 Data Type, Attribute and SQL-invoked Routine Identifiers.
- 3) **Definitional Rules:** This section contains an enumerated list of rules to be applied when defining the data type and routines. If this section is empty, the section heading is be present.
- 4) **Description:** an enumerated list describing the type or routine. For a type, the first item contains a statement indicating the attributes and routines that are part of the public specification. For a routine, the first item contains the definition of the routine's parameters. If this section is empty, the section heading is not present.

3.3.3 Data Type, Attribute and SQL-invoked Routine Identifiers

Type identifiers, attribute identifiers, and routine identifiers:

- 1) shall be prefixed have a prefix. Spatial uses "ST_".
- 2) shall not use the underbar character except in the prefix (i.e. only Alphanumeric characters [a-zA-Z1-9]),
- 3) shall capitalize each word used in the identifier. For example: ST_Point, ST_GeometricGroup,
- 4) shall be in *Italic* when used in the Definitional Rules and the Description sections.

3.3.4 Parameter Identifiers

Parameter identifiers are in lowercase. Parameters are in *Italic* when used in the Definitional Rules and the Description sections. This will distinguish them from other identifiers used in the Definitional rules and Description sections.

3.3.5 Meta-variables

Meta-variables used to define Implementation-dependant or Implementation-defined constants such as *ST_MaxPointArray* follows the conventions of Subclause 3.3.3 Data Type, Attribute and SQL-invoked Routine Identifiers. In addition, this is the only text in the Definition section that is in *Italic*.

3.3.6 Definitional variables

Definitional variables used in the Definitional Rules or Description sections are in uppercase italics. This will distinguish them from other identifiers used in the Definitional Rules or Description sections

3.3.7 Exceptions

Except where otherwise specified, the phrase “an exception condition is raised:”, followed by the name of a condition, is used in the Definitional Rules and elsewhere to indicate that:

- The execution of a routine is unsuccessful.
- Application of Definitional Rules may be terminated.

The effect on any assignment target and SQL descriptor area of an SQL-statement that terminates with an exception condition, unless explicitly defined by ISO/IEC 9075, is implementation-dependent.

The phrase “a completion condition is raised:”, followed by the name of a condition, is used in Definitional Rules and elsewhere to indicate that application of Definitional Rules is not terminated and diagnostic information is to be made available; unless an exception condition is also raised, the execution of the SQL-statement is successful.

4 Concepts

****Editor's Note 3-004****

SQL/MM YOK-10 stated that material from SQL/MM YOK-08 and YOK-09 could be used in the Concepts clause. The Editor cannot do this work without a detailed change proposal.

4.1 USAGE Privileges on User-defined Types

ISO/IEC 9075 specifies that a user must have the USAGE privilege on the domain or user-defined type before they can use it for defining other objects such as SQL-invoked routines, tables, views, domains, or user-defined types. This International Standard does not include the GRANT USAGE statements for the domains and user-defined types defined in this International Standard. For each object defined by this International Standard, a GRANT statement granting USAGE privilege to an implementation-defined set of grantees shall be effectively executed when these domains and user-defined types are created, except when explicitly noted by the Definitional Rules in this International Standard. It is implementation-defined whether the GRANT statement includes WITH GRANT OPTION.

4.2 UNDER Privileges on User-defined Types

ISO/IEC 9075 specifies that a user must have the UNDER privilege on the user-defined type before they can use it for defining subtypes. This International Standard does not include the GRANT UNDER statements for the user-defined types defined in this International Standard. For each object defined by this International Standard, a GRANT statement granting UNDER privilege to an implementation-defined set of grantees shall be effectively executed when these user-defined types are created, except when explicitly noted by the Definitional Rules in this International Standard. It is implementation-defined whether the GRANT statement includes WITH GRANT OPTION.

4.3 EXECUTE Privileges on SQL-Invoked Routines

ISO/IEC 9075 specifies that users must have the EXECUTE privilege on the SQL-invoked routine before they can execute it. This International Standard does not include the GRANT EXECUTE statements for the SQL-invoked defined in this International Standard. For each such SQL-invoked routine, a GRANT statement granting EXECUTE privilege to an implementation-defined set of grantees shall be effectively executed when the SQL-invoked routines are created, except when explicitly noted by the Definitional Rules in this International Standard. It is implementation-defined whether the GRANT statement includes WITH GRANT OPTION.

4.4 Relationships of Spatial Values

Metrical extension of the domain enables so simply describe relations between spatial objects.

The relations in two-dimensional spatial objects, consisting of two major relations: *Intersection* and *Enclosure*, while *Union* is self-evident and *Difference* are able to be treated as the complement of *Intersection*. The relationships are summarized in Table 1 — Spatial Relations in Extended Domain.

Table 1 — Spatial Relations in Extended Domain

Geometry Type	ST_Point	ST_Line	ST_Area
ST_Point	= (E ₁)	∈ (E ₂)	∈ (E ₃)
ST_Line		⊂, ∩ (E ₄), (I ₁)	⊂, ∩ (E ₅), (I ₂)
ST_Area			⊂, ∩ (E ₆), (I ₃)

E₁ – E₆ denote Enclosure relations, and I₁ – I₃ denote Intersection relations

$$E_1(P_i, P_j) = \begin{cases} 1 & \text{if } P_i = P_j \\ 0 & \text{if } P_i \neq P_j \end{cases}$$

$$E_2(P, L) = \begin{cases} 1 & \text{if } P \in L \\ 0 & \text{if } P \notin L \end{cases}$$

$$E_3(P, A) = \begin{cases} 1 & \text{if } P \in A \\ 0 & \text{if } P \notin A \end{cases}$$

$$E_4(L_i, L_j) = \begin{cases} 1 & \text{if } L_i \subset L_j \\ 0 & \text{otherwise} \end{cases}$$

$$E_5(L, A) = \begin{cases} 1 & \text{if } L \subset A \\ 0 & \text{otherwise} \end{cases}$$

$$E_6(A_i, A_j) = \begin{cases} 1 & \text{if } A_i \subset A_j \\ 0 & \text{otherwise} \end{cases}$$

$$I_1(L_i, L_j) = \begin{cases} L_i \cap L_j & \text{if } L_i \cap L_j \neq \emptyset \\ \text{NULL} & \text{otherwise} \end{cases}$$

$$I_2(L, A) = \begin{cases} L \cap A & \text{if } L \cap A \neq \emptyset \\ \text{NULL} & \text{otherwise} \end{cases}$$

$$I_3(A_i, A_j) = \begin{cases} A_i \cap A_j & \text{if } A_i \cap A_j \neq \emptyset \\ \text{NULL} & \text{otherwise} \end{cases}$$

Equation 1 — Enclosure and Intersection Relations

4.5 Area Data Types

4.5.1 ST_Region Type

The ST_Region type is a subtype of ST_Area. ST_Region is provided to model multiple, non-overlapping polygons. One use of the region type is to model spatial entities such as a group of islands (e.g. the Hawaiian Islands).

Another important use for a region type derives from the use of spatial operators, such as intersection or union. If a spatial operator is performed on two spatial objects, each containing a single ST_Polygon, then the result can yield a new spatial object that contains multiple, non-overlapping polygons.

5 Spatial Data Types

5.1 ST_Spatial Type and Routines

5.1.1 ST_Spatial Type

Purpose

The *ST_Spatial* type provides the functions required to create, query and manage spatially referenced information. The following definition of the *ST_Spatial* type provides the functions required to create spatial queries against information in a spatially referenced data store. *ST_Spatial* values are logical collections of *ST_GeometricElement* values in a specified spatial referencing system. The *ST_Spatial* value is indifferent to the specifics of the *ST_GeometricElement* clustering such that two representations of a *ST_Spatial* value clustered in different manners are equivalent.

Definition

```
CREATE TYPE ST_Spatial
  (ST_PrivateSR ST_SpatialReferencing,
   ST_Geometry ST_GeometricElement ARRAY[ST_MaxGeometricElementArray]
   DEFAULT EMPTY)
```

Definitional Rules

- 1) The attribute *ST_PrivateSR* is not part of the public interface. There are no GRANT statements granting EXECUTE privilege to the observer or mutator function for *ST_PrivateSR*.
- 2) If the *ST_PrivateSR* attribute is the NULL value, then the spatial object is in the Euclidean coordinate system.
- 3) *ST_MaxGeometricElementArray* is the maximum cardinality of the *ST_Elements* attribute. *ST_MaxGeometricElementArray* is an implementation-defined value.

****Editor's Note 3-093****

A NULL value for a Spatial Referencing System is defined to be the Euclidean coordinate system. A default spatial referencing system should be defined for all implementations and used instead of the NULL value.

Description

- 1) The *ST_Spatial* type provides for public use:
 - a) an attribute *ST_Geometry*,
 - b) a function *ST_Spatial(ST_GeometricElement ARRAY)*,
 - c) a function *ST_Spatial(ST_GeometricElement ARRAY, ST_SpatialReferencing)*,
 - d) a function *ST_SpatialReferencing(ST_Spatial)*,
 - e) a function *ST_Transform(ST_Spatial, ST_SpatialReferencing)*,
 - f) a function *ST_Buffer(ST_Spatial, DOUBLE PRECISION)*,

- g) a function *ST_Centroid*(*ST_Spatial*),
 - h) a function *ST_Envelope*(*ST_Spatial*),
 - i) a function *ST_Length*(*ST_Spatial*),
 - j) a function *ST_Area*(*ST_Spatial*),
 - k) a function *ST_Perimeter*(*ST_Spatial*),
 - l) a function *ST_Azimuth*(*ST_Spatial*, *ST_Spatial*),
 - m) a function *ST_Distance*(*ST_Spatial*, *ST_Spatial*),
 - n) a function *ST_Overlaps*(*ST_Spatial*, *ST_Spatial*),
 - o) a function *ST_Meets*(*ST_Spatial*, *ST_Spatial*),
 - p) a function *ST_Contains*(*ST_Spatial*, *ST_Spatial*),
 - q) a function *ST_ContainedBy*(*ST_Spatial*, *ST_Spatial*),
 - r) a function *ST_Outside*(*ST_Spatial*, *ST_Spatial*),
 - s) a function *ST_ExtractAt*(*ST_Spatial*, *INTEGER*),
 - t) a function *ST_Count*(*ST_Spatial*),
 - u) a function *ST_Intersection*(*ST_Spatial*, *ST_Spatial*),
 - v) a function *ST_Union*(*ST_Spatial*, *ST_Spatial*),
 - w) a function *ST_Difference*(*ST_Spatial*, *ST_Spatial*),
 - x) a function *ST_Decompose*(*ST_Spatial*),
 - y) a function *ST_Dimension*(*ST_Spatial*, *ST_Spatial*),
- 2) The attribute *ST_PrivateSR* specifies the spatial referencing system of the *ST_Spatial* value. The *ST_PrivateSR* attribute is set by the *ST_Spatial*(*ST_GeometricElement ARRAY*, *ST_SpatialReferencing*) function. The spatial referencing system for a *ST_Spatial* value is changed by the *ST_Transform* function.
- 3) The attribute *ST_Geometry* is an array of geometric elements.

5.1.2 ST_Spatial Functions

Purpose

Return a ST_Spatial value.

Definition

```
CREATE FUNCTION ST_Spatial
  (agea ST_GeometricElement ARRAY[ST_MaxGeometricElementArray])
  RETURNS ST_Spatial
  NULL CALL
  BEGIN
    DECLARE rtn ST_Spatial;

    SET rtn = ST_Spatial();
    SET rtn>>ST_Geometry = agea;
    SET rtn>>ST_SpatialReferencing = NULL;
    RETURN rtn;
  END

CREATE FUNCTION ST_Spatial
  (agea ST_GeometricElement ARRAY[ST_MaxGeometricElementArray],
   asr ST_SpatialReferencing)
  RETURNS ST_Spatial
  NULL CALL
  BEGIN
    DECLARE rtn ST_Spatial;

    SET rtn = ST_Spatial();
    SET rtn>>ST_Geometry = agea;
    SET rtn>>ST_PrivateSR = asr;
    RETURN rtn;
  END
```

Description

- 1) The function *ST_Spatial(ST_GeometricElement ARRAY)* takes the following input parameters:
 - a) a *ST_GeometricElement ARRAY* value *agea*.
- 2) The function *ST_Spatial(ST_GeometricElement ARRAY)* constructs and initializes a *ST_Spatial* value.
 - a) Construct a *ST_Spatial* value, *ASO*.
 - b) Set the *ST_Geometry* attribute of *ASO* to *agea*.
 - c) Set the *ST_PrivateSR* attribute of *ASO* to the NULL value.
 - d) Return *ASO*.
- 3) The function *ST_Spatial(ST_GeometricElement ARRAY, ST_SpatialReferencing)* takes the following input parameters:
 - a) a *ST_GeometricElement ARRAY* value *agea*,

- b) a *ST_SpatialReferencing* value *asr*.
- 4) The function *ST_Spatial(ST_GeometricElement ARRAY, ST_SpatialReferencing)* returns a *ST_Spatial* value.
 - a) Construct a *ST_Spatial* value, *ASO*.
 - b) Set the *ST_Geometry* attribute of *ASO* to *agea*.
 - c) Set the *ST_PrivateSR* attribute of *ASO* to *asr*.
 - d) Return *ASO*.

5.1.3 ST_SpatialReferencing Function

Purpose

Return the spatial referencing system of a *ST_Spatial* value.

Definition

```
CREATE FUNCTION ST_SpatialReferencing
  (aso ST_Spatial)
  RETURNS ST_SpatialReferencing
  NULL CALL
  BEGIN
    RETURN aso>>ST_PrivateSR;
  END
```

Description

- 1) The function *ST_SpatialReferencing(ST_Spatial)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso*.
- 2) The function *ST_SpatialReferencing(ST_Spatial)* returns the value of the attribute *ST_PrivateSR* in *aso*.

5.1.4 ST_Transform Function

Purpose

Transform a *ST_Spatial* value to a new spatial referencing system.

Definition

```
CREATE FUNCTION ST_Transform
  (aso ST_Spatial,
   asr ST_SpatialReferencing)
  RETURNS ST_Spatial
  NULL CALL
  --
  -- ! See Routine Description
  --
```

Description

- 1) The function *ST_Transform(ST_Spatial, ST_SpatialReferencing)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso*,
 - b) a *ST_SpatialReferencing* value *asr*.
- 2) The function *ST_Transform(ST_Spatial, ST_SpatialReferencing)* transforms *aso* to the spatial referencing system *asr*. The transformed *ST_Spatial* value is returned. This is an implementation-defined function.

5.1.5 ST_Buffer Function

Purpose

Return the buffer region around a ST_Spatial value.

Definition

```
CREATE FUNCTION ST_Buffer
  (aso ST_Spatial,
   awidth DOUBLE PRECISION)
  RETURNS ST_Spatial
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Buffer(ST_Spatial, DOUBLE PRECISION)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso*,
 - b) a *DOUBLE PRECISION* value *awidth*.
- 2) The function *ST_Buffer(ST_Spatial, DOUBLE PRECISION)* returns a *ST_Spatial* value in the spatial referencing system of *aso*.
- 3) The resulting value represents the expanded region around *aso*.

5.1.6 ST_Centroid Function

Purpose

Return the centroid of a *ST_Spatial* value.

Definition

```
CREATE FUNCTION ST_Centroid
  (aso ST_Spatial)
  RETURNS ST_Spatial
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Centroid(ST_Spatial)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso*.
- 2) The function *ST_Centroid(ST_Spatial)* returns a *ST_Spatial* value in the spatial referencing system of *aso*.
- 3) The returned value contains a single *ST_Point* value in the array of the *ST_Geometry* attribute. This point represents the centroid of *aso*.

5.1.7 ST_Envelope Function

Purpose

Return the minimum enclosing rectangle of a *ST_Spatial* value.

Definition

```
CREATE FUNCTION ST_Envelope
  (aso ST_Spatial)
  RETURNS ST_Envelope
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Envelope(ST_Spatial)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso*.
- 2) The function *ST_Envelope(ST_Spatial)* returns a *ST_Spatial* value in the spatial referencing system of *aso*.
- 3) The returned value contains a single *ST_Polygon* value in the array of the *ST_Geometry* attribute. This polygon represents the envelope (minimum enclosing rectangle) around *aso*.

5.1.8 ST_Length Function

Purpose

Return the length of a ST_Spatial value.

Definition

```
CREATE FUNCTION ST_Length
  (aso ST_Spatial)
  RETURNS DOUBLE PRECISION
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Length(ST_Spatial)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso*.
- 2) Let *GAC* be the *ST_Geometry* attribute of *aso*.
- 3) If any value in *GAC* is not a subtype of *ST_Line*, then an exception condition is raised: *SQL/MM Spatial exception - invalid parameter*.
- 4) The function *ST_Length(ST_Spatial)* returns the sum total of the length of each value in *GAC* as a DOUBLE PRECISION value. The units of the result are determined by the spatial referencing system of *aso*.

5.1.9 ST_Area Function

Purpose

Return the area of a ST_Spatial value.

Definition

```
CREATE FUNCTION ST_Area
  (aso ST_Spatial)
  RETURNS DOUBLE PRECISION
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Area(ST_Spatial)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso*.
- 2) Let *GAC* be the *ST_Geometry* attribute of *aso*.
- 3) If any value in *GAC* is not a subtype of *ST_Area*, then an exception condition is raised: *SQL/MM Spatial exception - invalid parameter*.
- 4) The function *ST_Area(ST_Spatial)* returns the sum total of the area of each value in *GAC* as a DOUBLE PRECISION value. The units of the result are determined by the spatial referencing system of *aso*.

5.1.10 ST_Perimeter Function

Purpose

Return the length of the perimeter of a ST_Spatial value.

Definition

```
CREATE FUNCTION ST_Perimeter
  (aso ST_Spatial)
  RETURNS DOUBLE PRECISION
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Perimeter(ST_Spatial)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso*.
- 2) Let *GAC* be the *ST_Geometry* attribute of *aso*.
- 3) If any value in *GAC* is not a subtype of *ST_Area*, then an exception condition is raised: *SQL/MM Spatial exception - invalid parameter*.
- 4) The function *ST_Perimeter(ST_Spatial)* returns the sum total of the length of each boundary in every value of *GAC* as a DOUBLE PRECISION value. The units of the result are determined by the spatial referencing system of *aso*.

5.1.11 ST_Azimuth Function

Purpose

Return the azimuth calculation.

Definition

```
CREATE FUNCTION ST_Azimuth
  (aso1 ST_Spatial
   aso2 ST_Spatial)
  RETURNS DOUBLE PRECISION
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Azimuth(ST_Spatial, ST_Spatial)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso1*,
 - b) a *ST_Spatial* value *aso2*.
- 2) If *aso2* is not in the spatial referencing system of *aso1*, then transform *aso2* into the spatial referencing system of *aso1*; let *TSO2* be the result. Otherwise, let *TSO2* be *aso2*.
- 3) Let *GAC1* be the *ST_Geometry* attribute of *aso1* and let *GAC2* be the *ST_Geometry* attribute of *TSO2*.
- 4) If the cardinality of *GAC1* is not one, then an exception condition is raised: *SQL/MM Spatial exception - invalid parameter*.
- 5) If the cardinality of *GAC2* is not one, then an exception condition is raised: *SQL/MM Spatial exception - invalid parameter*.
- 6) Let *ITM1* be the element in *GAC1* and let *ITM2* be the element in *GAC2*.
- 7) If *ITM1* is not a subtype of *ST_Point*, then an exception condition is raised: *SQL/MM Spatial exception - invalid parameter*.
- 8) If *ITM2* is not a subtype of *ST_Point*, then an exception condition is raised: *SQL/MM Spatial exception - invalid parameter*.
- 9) The function *ST_Azimuth(ST_Spatial, ST_Spatial)* returns the clockwise angle of the vector from *ITM1* to *ITM2*. The units of the result are determined by the spatial referencing system of *aso1*.

5.1.12 ST_Distance Function

Purpose

Return the distance between two ST_Spatial values.

Definition

```
CREATE FUNCTION ST_Distance
  (aso1 ST_Spatial,
   aso2 ST_Spatial)
  RETURNS DOUBLE PRECISION
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Distance(ST_Spatial, ST_Spatial)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso1*,
 - b) a *ST_Spatial* value *aso2*.
- 2) If *aso2* is not in the spatial referencing system of *aso1*, then transform *aso2* into the spatial referencing system of *aso1*; let *TSO2* be the result. Otherwise, let *TSO2* be *aso2*.
- 3) The function *ST_Distance(ST_Spatial, ST_Spatial)* returns the minimum distance between *aso1* and *TSO2* as a DOUBLE PRECISION value. The *ST_Distance* function determines the nearest two points, one from each spatial object, and calculates the distance between the points. The units of the result are determined by the spatial referencing system of *aso1*.

5.1.13 ST_Overlaps Function

Purpose

Return the boolean result of the overlaps comparison of two ST_Spatial values.

Definition

```
CREATE FUNCTION ST_Overlaps
  (aso1 ST_Spatial,
   aso2 ST_Spatial)
  RETURNS BOOLEAN
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Overlaps(ST_Spatial, ST_Spatial)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso1*,
 - b) a *ST_Spatial* value *aso2*.
- 2) If *aso2* is not in the spatial referencing system of *aso1*, then transform *aso2* into the spatial referencing system of *aso1*; let *TSO2* be the result. Otherwise, let *TSO2* be *aso2*.
- 3) Let *GAC1* be the *ST_Geometry* attribute of *aso1* and let *GAC2* be the *ST_Geometry* attribute of *TSO2*.
- 4) If any element of *GAC1* overlaps any element of *GAC2*, then the result is true. Otherwise, the result is false.

5.1.14 ST_Meets Function

Purpose

Return the boolean result of the meets comparison of two ST_Spatial values.

Definition

```
CREATE FUNCTION ST_Meets
  (aso1 ST_Spatial,
   aso2 ST_Spatial)
  RETURNS BOOLEAN
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Meets(ST_Spatial, ST_Spatial)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso1*,
 - b) a *ST_Spatial* value *aso2*.
- 2) If *aso2* is not in the spatial referencing system of *aso1*, then transform *aso2* into the spatial referencing system of *aso1*; let *TSO2* be the result. Otherwise, let *TSO2* be *aso2*.
- 3) Let *GAC1* be the *ST_Geometry* attribute of *aso1* and let *GAC2* be the *ST_Geometry* attribute of *TSO2*.
- 4) If any element of *GAC1* meets any element of *GAC2*, then the result is true. Otherwise, the result is false.

5.1.15 ST_Contains Function

Purpose

Return the boolean result of the contains comparison of two ST_Spatial values.

Definition

```
CREATE FUNCTION ST_Contains
  (aso1 ST_Spatial,
   aso2 ST_Spatial)
  RETURNS BOOLEAN
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Contains(ST_Spatial, ST_Spatial)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso1*,
 - b) a *ST_Spatial* value *aso2*.
- 2) If *aso2* is not in the spatial referencing system of *aso1*, then transform *aso2* into the spatial referencing system of *aso1*; let *TSO2* be the result. Otherwise, let *TSO2* be *aso2*.
- 3) Let *GAC1* be the *ST_Geometry* attribute of *aso1* and let *GAC2* be the *ST_Geometry* attribute of *TSO2*.
- 4) If the elements of *GAC1* contain all the elements of *GAC2*, then the result is true. Otherwise, the result is false.

5.1.16 ST_ContainedBy Function

Purpose

Return the boolean result of the contained by comparison of two ST_Spatial values.

Definition

```
CREATE FUNCTION ST_ContainedBy
  (aso1 ST_Spatial,
   aso2 ST_Spatial)
  RETURNS BOOLEAN
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_ContainedBy(ST_Spatial, ST_Spatial)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso1*,
 - b) a *ST_Spatial* value *aso2*.
- 2) If *aso2* is not in the spatial referencing system of *aso1*, then transform *aso2* into the spatial referencing system of *aso1*; let *TSO2* be the result. Otherwise, let *TSO2* be *aso2*.
- 3) Let *GAC1* be the *ST_Geometry* attribute of *aso1* and let *GAC2* be the *ST_Geometry* attribute of *TSO2*.
- 4) If the elements of *GAC2* contain all the elements of *GAC1*, then the result is true. Otherwise, the result is false.

5.1.17 ST_Outside Function

Purpose

Return the boolean result of the disjoint comparison of two ST_Spatial values.

Definition

```
CREATE FUNCTION ST_Outside
  (aso1 ST_Spatial,
   aso2 ST_Spatial)
  RETURNS BOOLEAN
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Outside(ST_Spatial, ST_Spatial)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso1*,
 - b) a *ST_Spatial* value *aso2*.
- 2) If *aso2* is not in the spatial referencing system of *aso1*, then transform *aso2* into the spatial referencing system of *aso1*; let *TSO2* be the result. Otherwise, let *TSO2* be *aso2*.
- 3) Let *GAC1* be the *ST_Geometry* attribute of *aso1* and let *GAC2* be the *ST_Geometry* attribute of *TSO2*.
- 4) If any element of *GAC1* overlaps any element of *GAC2*, then the result is *false*. Otherwise, the result is *true*.

5.1.18 ST_ExtractAt Function

Purpose

Return a ST_Spatial value with one geometric element from a given ST_Spatial value.

Definition

```
CREATE FUNCTION ST_ExtractAt
  (aso ST_Spatial,
   aposition INTEGER)
  RETURNS ST_Spatial
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_ExtractAt(ST_Spatial, INTEGER)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso*,
 - b) an INTEGER value *aposition*.
- 2) If the *ST_Geometry* attribute of *aso* is the EMPTY value, then an exception condition is raised: *SQL/MM Spatial exception - invalid parameter*.
- 3) If *aposition* is less than 1 (one) or greater than the cardinality of the *ST_Geometry* attribute of *aso*, then an exception condition is raised: *SQL/MM Spatial exception - invalid position*.
- 4) The function *ST_ExtractAt(ST_Spatial, INTEGER)* returns a *ST_Spatial* value in the spatial referencing system of *aso*.
- 5) Let *ELX* be the element at position *aposition* in the *ST_Geometry* attribute of *aso*.
- 6) The returned value contains a single element *ELX* in the array of the *ST_Geometry* attribute.

5.1.19 ST_Count Function

Purpose

Return the number of geometric elements in a ST_Spatial value.

Definition

```
CREATE FUNCTION ST_Count
  (aso ST_Spatial)
  RETURNS INTEGER
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Count(ST_Spatial)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso*.
- 2) If the *ST_Geometry* attribute of *aso* is the EMPTY value, then 0 (zero) is returned. Otherwise, the cardinality of *ST_Geometry* attribute is returned.

5.1.20 ST_Intersection Function

Purpose

Return the spatial intersection of two ST_Spatial values.

Definition

```
CREATE FUNCTION ST_Intersection
  (aso1 ST_Spatial,
   aso2 ST_Spatial)
  RETURNS ST_Spatial
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Intersection(ST_Spatial, ST_Spatial)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso1*,
 - b) a *ST_Spatial* value *aso2*.
- 2) If *aso2* is not in the spatial referencing system of *aso1*, then transform *aso2* into the spatial referencing system of *aso1*; let *TSO2* be the result. Otherwise, let *TSO2* be *aso2*.
- 3) Let *GAC1* be the *ST_Geometry* attribute of *aso1* and let *GAC2* be the *ST_Geometry* attribute of *TSO2*.
- 4) The function *ST_Intersection(ST_Spatial, ST_Spatial)* returns a *ST_Spatial* value in the spatial referencing system of *aso1*.
- 5) The returned value contains a *ST_GeometricElement* ARRAY value resulting from the intersection of all the element of *GAC1* values with all the elements of *GAC2* : $\text{geometry}(\text{result}) = \text{geometry}(\text{aso1}) \cap \text{geometry}(\text{TSO2})$.

5.1.21 ST_Union Function

Purpose

Return the spatial union of two ST_Spatial values.

Definition

```
CREATE FUNCTION ST_Union
  (aso1 ST_Spatial,
   aso2 ST_Spatial)
  RETURNS ST_Spatial
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Union(ST_Spatial, ST_Spatial)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso1*,
 - b) a *ST_Spatial* value *aso2*.
- 2) If *aso2* is not in the spatial referencing system of *aso1*, then transform *aso2* into the spatial referencing system of *aso1*; let *TSO2* be the result. Otherwise, let *TSO2* be *aso2*.
- 3) Let *GAC1* be the *ST_Geometry* attribute of *aso1* and let *GAC2* be the *ST_Geometry* attribute of *TSO2*.
- 4) The function *ST_Union(ST_Spatial, ST_Spatial)* returns a *ST_Spatial* value in the spatial referencing system of *aso1*.
- 5) The returned value contains a *ST_GeometricElement* ARRAY value resulting from the union of all the element of *GAC1* values with all the elements of *GAC2*: $\text{geometry}(\text{result}) = \text{geometry}(\text{aso1}) \cup \text{geometry}(\text{TSO2})$.

5.1.22 ST_Difference Function

Purpose

Return the spatial difference between two ST_Spatial values.

Definition

```
CREATE FUNCTION ST_Difference
  (aso1 ST_Spatial,
   aso2 ST_Spatial)
  RETURNS ST_Spatial
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Difference(ST_Spatial, ST_Spatial)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso1*,
 - b) a *ST_Spatial* value *aso2*.
- 2) If *aso2* is not in the spatial referencing system of *aso1*, then transform *aso2* into the spatial referencing system of *aso1*; let *TSO2* be the result. Otherwise, let *TSO2* be *aso2*.
- 3) Let *GAC1* be the *ST_Geometry* attribute of *aso1* and let *GAC2* be the *ST_Geometry* attribute of *TSO2*.
- 4) Let *ITM1* be the element in *GAC1* and let *ITM2* be the element in *GAC2*.
- 5) If *ITM1* is not a subtype of *ST_Area*, then an exception condition is raised: *SQL/MM Spatial exception - invalid parameter*.
- 6) If *ITM2* is not a subtype of *ST_Area*, then an exception condition is raised: *SQL/MM Spatial exception - invalid parameter*.
- 7) The function *ST_Difference(ST_Spatial, ST_Spatial)* returns a *ST_Spatial* value in the spatial referencing system of *aso1*.
- 8) The returned value contains a *ST_GeometricElement* ARRAY value resulting from the subtraction of all the element of *GAC2* values from all the elements of *GAC1*: $\text{geometry}(\text{result}) = \text{geometry}(\text{aso1}) - \text{geometry}(\text{TSO2})$.

5.1.23 ST_Decompose Function

Purpose

Decompose the geometry elements in a ST_Spatial value.

Definition

```
CREATE FUNCTION ST_Decompose
  (aso ST_Spatial)
  RETURNS ST_Spatial
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Decompose(ST_Spatial)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso*.
- 2) The function *ST_Decompose(ST_Spatial)* returns a *ST_Spatial* value in the spatial referencing system of the *aso* parameter.
- 3) The returned value contains a *ST_GeometricElement* ARRAY value resulting from decomposing each geometry value in *aso* as follows:
 - a) A *ST_Polygon* value is decomposed to a *ST_Path* ARRAY value representing the polygon's boundaries.
 - b) A *ST_Path* value is decomposed to a *ST_Curve* ARRAY value.
 - c) A *ST_Curve* value is decomposed to a *ST_Point* ARRAY value.
 - d) A *ST_Point* value is decomposed to the EMPTY value.

****Editor's Note 3-094****

Decompose should be replaced by functions like AsExteriorPath, AsInteriorPaths, AsCurves, AsPoints.

5.1.24 ST_Dimension Function

Purpose

Return the dimension of a ST_Spatial value.

Definition

```
CREATE FUNCTION ST_Dimension
  (aso ST_Spatial)
  RETURNS INTEGER
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Dimension(ST_Spatial)* takes the following input parameters:
 - a) a *ST_Spatial* value *aso*.
- 2) Let *GAC1* be the *ST_Geometry* attribute of *aso1*.
- 3) Case:
 - a) If all values in *GAC1* are type *ST_Point*, return 0.
 - b) If all values in *GAC1* are type *ST_Line*, return 1.
 - c) If all values in *GAC1* are type *ST_Area*, return 2.
 - d) Otherwise, the NULL value is returned.

6 Spatial Referencing Data Types

6.1 ST_SpatialReferencing Type and Routines

6.1.1 ST_SpatialReferencing Type

Purpose

This opaque type encapsulates all aspects of spatial referencing.

Definition

```
CREATE TYPE ST_SpatialReferencing
    EQUALS ONLY BY RELATIVE ST_IsEqual
--
-- See Description
--
```

Definitional Rules

- 1) The spatial referencing systems are implementation-defined.

Description

- 1) The *ST_SpatialReferencing* type provides for public use:
 - a) a function *ST_SpatialReferencing*(*CHARACTER VARYING*),
 - b) a function *ST_Name*(*ST_SpatialReferencing*),
 - c) a function *ST_IsEqual*(*ST_SpatialReferencing*).
- 2) The data type is opaque. The attribute definition of this type is implementation dependent.

****Editor's Note 3-092****

ST_SpatialReferencing, as defined in MAD-049 is "an opaque type that encapsulates all aspects of spatial referencing". There is a need to include descriptive text in Cause 4: Concepts to explain how this feature may be used by an implementation and how it may relate to standards for spatial referencing (such as defined in ISO/TC 211).

6.1.2 ST_SpatialReferencing Function

Purpose

Return a value of type ST_SpatialReferencing.

Definition

```
CREATE FUNCTION ST_SpatialReferencing
  (aname CHARACTER VARYING(128))
  RETURNS ST_SpatialReferencing
  NULL CALL
  --
  -- See Routine Description
  --
```

Description

- 1) The function *ST_SpatialReferencing*(*CHARACTER VARYING*) takes the following input parameters:
 - a) a *CHARACTER VARYING* value *aname*.
- 2) The function *ST_SpatialReferencing*(*CHARACTER VARYING*) returns a value of type *ST_SpatialReferencing*.

6.1.3 ST_Name Function

Purpose

Return the name of the Spatial Referencing system.

Definition

```
CREATE FUNCTION ST_Name
  (asr ST_SpatialReferencing)
  RETURNS CHARACTER VARYING(128)
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Name*(*ST_SpatialReferencing*) takes the following input parameters:
 - a) a *ST_SpatialReferencing* value *asr*.
- 2) This function returns the implementation-defined name of the spatial referencing system.

6.1.4 ST_IsEqual Function

Purpose

Compare to ST_SpatialReferencing values.

Definition

```
CREATE FUNCTION ST_IsEqual
  (asr1 ST_SpatialReferencing,
   asr2 ST_SpatialReferencing)
  RETURNS BOOLEAN
  NULL CALL
  --
  -- See Routine Description
  --
```

Description

- 1) The function *ST_IsEqual(ST_SpatialReferencing)* takes the following input parameters:
 - a) a *ST_SpatialReferencing* value *asr1*,
 - b) a *ST_SpatialReferencing* value *asr2*.
- 2) This function returns *true* if *asr1* is equal to *asr2*; otherwise *false*.
- 3) This function is designated as the <relative function specification> in the <ordering method> of this type. This function is implementation-defined.

****Editor's Note 3-113****

SQL3 requires the <relative function specification> in the <ordering method> to be an integer function, not a boolean.

7 Coordinates Data Types

7.1 ST_Coordinates Type and Routines

7.1.1 ST_Coordinates Type

Purpose

The *ST_Coordinates* data type holds a position to objects in two-dimensional and three-dimensional space.

Definition

```
CREATE TYPE ST_Coordinates
  (ST_PrivateX ST_CoordMetaType,
   ST_PrivateY ST_CoordMetaType,
   ST_PrivateZ ST_CoordMetaType)
```

Definitional Rules

- 1) The attribute *ST_PrivateX* is not part of the public interface. There are no GRANT statements granting EXECUTE privilege to the observer or mutator function for *ST_PrivateX*.
- 2) The attribute *ST_PrivateY* is not part of the public interface. There are no GRANT statements granting EXECUTE privilege to the observer or mutator function for *ST_PrivateY*.
- 3) The attribute *ST_PrivateZ* is not part of the public interface. There are no GRANT statements granting EXECUTE privilege to the observer or mutator function for *ST_PrivateZ*.
- 4) The value of *ST_CoordMetaType* shall be INTEGER, REAL, or DOUBLE PRECISION.
- 5) If the z coordinate contains the NULL value, the *ST_Coordinates* value is considered two dimensional; otherwise the *ST_Coordinates* value is considered three dimensional.

****Editor's Note 3-091****

Under proposal MAD-048 accepted in Madrid, the type *ST_Coordinates* uses the same representation for 2-dimensional and 3 dimensional coordinates, using NULL to indicate 2-dimensional. This is highly questionable especially in value of SQL's subtyping facilities that appear to be tailor made for this purpose. Similarly, subtype could be exploited to address the problem of coordinates using different precisions (both within an implementation and between implementations).

In each case the specification is contrary to accepted principles of good design in object-oriented systems (as typified by SQL3).

Description

- 1) The *ST_SpatialReferencing* type provides for public use:
 - a) a function *ST_Coordinates(ST_CoordMetaType, ST_CoordMetaType)*,
 - b) a function *ST_Coordinates(ST_CoordMetaType, ST_CoordMetaType, ST_CoordMetaType)*,
 - c) a function *ST_X(ST_Coordinates)*,

- d) a function *ST_X*(*ST_Coordinates*, *ST_CoordMetaType*),
 - e) a function *ST_Y*(*ST_Coordinates*),
 - f) a function *ST_Y*(*ST_Coordinates*, *ST_CoordMetaType*),
 - g) a function *ST_Z*(*ST_Coordinates*),
 - h) a function *ST_Z*(*ST_Coordinates*, *ST_CoordMetaType*),
 - i) a function *ST_Is2D*(*ST_Coordinates*),
 - j) a function *ST_Is3D*(*ST_Coordinates*),
 - k) a function *ST_ExplicitCoordinates*(*ST_Coordinates*).
- 2) The attribute *ST_PrivateX* contains the coordinate value in the x axis.
 - 3) The attribute *ST_PrivateY* contains the coordinate value in the y axis.
 - 4) The attribute *ST_PrivateZ* contains the coordinate value in the z axis.

7.1.2 ST_Coordinates Functions

Purpose

Return a 2D or 3D position.

Definition

```
CREATE FUNCTION ST_Coordinates
  (c1 ST_CoordMetaType ,
   c2 ST_CoordMetaType)
  RETURNS ST_Coordinates
  NULL CALL
  BEGIN
    DECLARE rtn ST_Coordinates;

    SET rtn = ST_Coordinates();
    SET rtn>>ST_X = c1;
    SET rtn>>ST_Y = c2;
    RETURN rtn;
  END
```

```
CREATE FUNCTION ST_Coordinates
  (c1 ST_CoordMetaType ,
   c2 ST_CoordMetaType ,
   c3 ST_CoordMetaType)
  RETURNS ST_Coordinates
  NULL CALL
  BEGIN
    DECLARE rtn ST_Coordinates;

    SET rtn = ST_Coordinates();
    SET rtn>>ST_X = c1;
    SET rtn>>ST_Y = c2;
    SET rtn>>ST_Z = c3;
    RETURN rtn;
  END
```

Description

- 1) The function *ST_Coordinates(ST_CoordMetaType, ST_CoordMetaType)* takes the following input parameters:
 - a) a *ST_CoordMetaType* value *c1*,
 - b) a *ST_CoordMetaType* *c2*.
- 2) The function *ST_Coordinates(ST_CoordMetaType, ST_CoordMetaType)* returns a 2D *ST_Coordinates* value with the x coordinate set to the first parameter and the y coordinate set to the second parameter.
- 3) The function *ST_Coordinates(ST_CoordMetaType, ST_CoordMetaType)* takes the following input parameters:
 - a) a *ST_CoordMetaType* *c1*,
 - b) a *ST_CoordMetaType* *c2*,

- c) a *ST_CoordMetaType* *c3*.
- 4) The function *ST_Coordinates(ST_CoordMetaType, ST_CoordMetaType, ST_CoordMetaType)* returns a 3D *ST_Coordinates* value with the x coordinate set to the first parameter, the y coordinate set to the second parameter and the z coordinate set to the third parameter.

7.1.3 ST_X Functions

Purpose

Get and set the x coordinate of a position.

Definition

```

CREATE FUNCTION ST_X
  (acoord ST_Coordinates)
  RETURNS ST_CoordMetaType
  NULL CALL
  BEGIN
    RETURN acoord>>ST_PrivateX;
  END

CREATE FUNCTION ST_X
  (acoord ST_Coordinates RESULT,
   avalue ST_CoordMetaType)
  RETURNS ST_Coordinates
  BEGIN
    DECLARE rtn ST_Coordinates;
    DECLARE NullValue EXCEPTION FOR SQLSTATE 'H3F03';

    SET rtn = acoord;
    IF avalue IS NULL THEN
      SIGNAL NullValue;
    ELSE
      SET rtn>>ST_PrivateX = avalue;
    END IF;
    RETURN rtn;
  END

```

Description

- 1) The function *ST_X(ST_Coordinates)* takes the following input parameters:
 - a) a *ST_Coordinates* value *acoord*.
- 2) The function *ST_X(ST_Coordinates)* returns the value of the attribute *ST_PrivateX*.
- 3) The function *ST_X(ST_Coordinates, ST_CoordMetaType)* takes the following input parameters:
 - a) a *ST_Coordinates* value *acoord*,
 - b) a *ST_CoordMetaType* *avalue*.
- 4) The function *ST_X(ST_Coordinates, ST_CoordMetaType)* applies the NOT NULL constraint when setting *avalue* to the attribute *ST_PrivateX* in *acoord*.

7.1.4 ST_Y Functions

Purpose

Get and set the y coordinate of a position.

Definition

```

CREATE FUNCTION ST_Y
  (acoord ST_Coordinates)
  RETURNS ST_CoordMetaType
  NULL CALL
  BEGIN
    RETURN acoord>>ST_PrivateY;
  END

CREATE FUNCTION ST_Y
  (acoord ST_Coordinates RESULT,
   avalue ST_CoordMetaType)
  RETURNS ST_Coordinates
  BEGIN
    DECLARE rtn ST_Coordinates;
    DECLARE NullValue EXCEPTION FOR SQLSTATE 'H3F03';

    SET rtn = acoord;
    IF avalue IS NULL THEN
      SIGNAL NullValue;
    ELSE
      SET rtn>>ST_PrivateY = avalue;
    END IF;
    RETURN rtn;
  END

```

Description

- 1) The function *ST_Y(ST_Coordinates)* takes the following input parameters:
 - a) a *ST_Coordinates* value *acoord*.
- 2) The function *ST_Y(ST_Coordinates)* returns the value of the attribute *ST_PrivateY*.
- 3) The function *ST_Y(ST_Coordinates, ST_CoordMetaType)* takes the following input parameters:
 - a) a *ST_Coordinates* value *acoord*,
 - b) a *ST_CoordMetaType* *avalue*.
- 4) The function *ST_Y(ST_Coordinates, ST_CoordMetaType)* applies the NOT NULL constraint when setting *avalue* to the attribute *ST_PrivateY* in *acoord*.

7.1.5 ST_Z Functions

Purpose

Get and set the z coordinate of a position.

Definition

```
CREATE FUNCTION ST_Z
  (acoord ST_Coordinates)
  RETURNS ST_CoordMetaType
  NULL CALL
  BEGIN
    RETURN acoord>>ST_PrivateZ;
  END
```

```
CREATE FUNCTION ST_Z
  (acoord ST_Coordinates RESULT,
   avalue ST_CoordMetaType)
  RETURNS ST_Coordinates
  BEGIN
    DECLARE rtn ST_Coordinates;

    SET rtn = acoord;
    SET rtn>>ST_PrivateZ = avalue;
    RETURN rtn;
  END
```

Description

- 1) The function *ST_Z(ST_Coordinates)* takes the following input parameters:
 - a) a *ST_Coordinates* value *acoord*.
- 2) The function *ST_Z(ST_Coordinates)* returns the value of the attribute *ST_PrivateZ*.
- 3) The function *ST_Z(ST_Coordinates, ST_CoordMetaType)* takes the following input parameters:
 - a) a *ST_Coordinates* value *acoord*,
 - b) a *ST_CoordMetaType* *avalue*.
- 4) The function *ST_Z(ST_Coordinates, ST_CoordMetaType)* sets the value of *ST_PrivateZ*.

7.1.6 ST_Is2D Function

Purpose

Determine if a location is in 2D.

Definition

```
CREATE FUNCTION ST_Is2D
  (acoord ST_Coordinates)
  RETURNS BOOLEAN
  NULL CALL
  BEGIN
    RETURN acoord>>ST_PrivateZ IS NULL;
  END
```

Description

- 1) The function *ST_Is2D(ST_Coordinates)* takes the following input parameters:
 - a) a *ST_Coordinates* value *acoord*.
- 2) The function *ST_Is2D(ST_Coordinates)* returns true if the *ST_Coordinates* value is two-dimensional; otherwise false.

7.1.7 ST_Is3D Function

Purpose

Determine if a location is in 3D.

Definition

```
CREATE FUNCTION ST_Is3D
  (acoord ST_Coordinates)
  RETURNS BOOLEAN
  NULL CALL
  BEGIN
    RETURN acoord>>ST_PrivateZ IS NOT NULL THEN
  END
```

Description

- 1) The function *ST_Is3D(ST_Coordinates)* takes the following input parameters:
 - a) a *ST_Coordinates* value *acoord*.
- 2) The function *ST_Is3D(ST_Coordinates)* returns true if the *ST_Coordinates* value is three-dimensional; otherwise false.

7.1.8 ST_ExplicitCoordinates Function

Purpose

Return the coordinate values as an array.

Definition

```
CREATE FUNCTION ST_ExplicitCoordinates
  (acoord ST_Coordinates)
  RETURNS ST_CoordMetaType ARRAY[3]
  NULL CALL
  BEGIN
    IF c1>>ST_PrivateZ IS NULL THEN
      RETURN ARRAY[c1>>ST_PrivateX,c1>>ST_PrivateY];
    ELSE
      RETURN ARRAY[c1>>ST_PrivateX,c1>>ST_PrivateY,c1>>ST_PrivateZ];
    END IF;
  END
```

Description

- 1) The function *ST_ExplicitCoordinates(ST_Coordinates)* takes the following input parameters:
 - a) a *ST_Coordinates* value *acoord*.
- 2) If *acoord* is 2D, then the function *ST_ExplicitCoordinates(ST_Coordinates)* returns an array of type *ST_CoordMetaType* with 2 values representing the x and y values. Otherwise, an array of type *ST_CoordMetaType* with 3 values representing the x, y and z values is returned.

8 Geometric Data Types

8.1 ST_GeometricElement Type and Routines

8.1.1 ST_GeometricElement Type

Purpose

ST_GeometricElement is a supertype within which all geometric elements are spatially referenced. All objects belonging to this supertype have position specified directly or indirectly through their constituent spatial attributes.

Definition

```
CREATE TYPE ST_GeometricElement
NOT INSTANTIABLE
```

Description

1) The *ST_GeometricElement* type provides no attributes or routines for public use.

****Editor's Note 3-099****

The purpose states that "ST_GeometricElement is a type supertype within which all geometric objects are spatially referenced." Coordinates are currently spatially referenced with no requirement that all coordinates of an ST_Spatial value be of the same referencing system. ST_Spatial values are spatially referenced lists of ST_GeometricElements. ST_GeometricElements, by themselves, are not spatially referenced.

The following is a partial solution:

In the first sentence beginning "ST_GeometricElement is a type supertype...", change "spatially referenced" to "contained."

8.2 ST_GeometricGroup Type and Routines

8.2.1 ST_GeometricGroup Type

Purpose

ST_GeometricGroup is used to gather a related group of geometric primitives into one piece. The group can be composed of geometric primitives of the same or different types including other ST_GeometricGroup values. This type enables the creation of arbitrarily complex geometric figures.

Definition

```
CREATE TYPE ST_GeometricGroup
  UNDER ST_GeometricElement
  (ST_Elements ST_GeometricElement ARRAY[ST_MaxGeometricElementArray])
```

Definitional Rules

- 1) *ST_MaxGeometricElementArray* is the maximum cardinality of the *ST_Elements* attribute. *ST_MaxGeometricElementArray* is an implementation-defined value.

Description

- 1) The *ST_GeometricGroup* type provides for public use:
 - a) an attribute *ST_Elements*.
- 2) The attribute *ST_Elements* contains the collection of geometric elements.

9 Point Data Types

9.1 ST_Point Type and Routines

9.1.1 ST_Point Type

Purpose

ST_Point is used for the position of single points, including nodes, origins of grids, placement of text, etc.

Definition

```
CREATE TYPE ST_Point
  UNDER ST_GeometricElement
  (ST_PrivateCoord ST_Coordinates)
```

Definitional Rules

- 1) The attribute *ST_PrivateCoord* is not part of the public interface. There are no GRANT statements granting EXECUTE privilege to the observer or mutator function for *ST_PrivateCoord*.

Description

- 1) The *ST_Point* type provides for public use:
 - a) a function *ST_Coord(ST_Point)*,
 - b) a function *ST_Coord(ST_Point, ST_Coordinates)*.
- 2) The attribute *ST_PrivateCoord* contains the coordinates for the point.

9.1.2 ST_Coord Functions

Purpose

Get and set the coordinates.

Definition

```

CREATE FUNCTION ST_Coord
  (pnt ST_Point)
  RETURNS ST_Coordinates
  NULL CALL
  BEGIN
    RETURN pnt>>ST_PrivateCoord;
  END

CREATE FUNCTION ST_Coord
  (pnt ST_Point RESULT,
   crd ST_Coordinates)
  RETURNS ST_Point
  BEGIN
    DECLARE rtn ST_Point;
    DECLARE NullValue CONDITION FOR SQLSTATE 'H3F03';

    SET rtn = pnt;
    IF crd IS NULL THEN
      SIGNAL NullValue;
    ELSE
      SET rtn>>ST_PrivateCoord = crd;
    END IF;
    RETURN rtn;
  END

```

Description

- 1) The function *ST_Coord(ST_Point)* takes the following input parameters:
 - a) a *ST_Point* value *pnt*.
- 2) The function *ST_Coord(ST_Point)* returns the coordinates in the point *pnt*.
- 3) The function *ST_Coord(ST_Point, ST_Coordinates)* takes the following input parameters:
 - a) a *ST_Point* value *pnt*,
 - b) a *ST_Coordinates* value *crd*.
- 4) The function *ST_Coord(ST_Point, ST_Coordinates)* applies the NOT NULL constraint when setting *crd* to the *ST_PrivateCoord* attribute in *pnt*.

10 Line Data Types

10.1 ST_Line Type and Routines

10.1.1 ST_Line Type

Purpose

ST_Line is a type supertype for representing one-dimensional geometric elements in higher dimensional space.

Definition

```
CREATE TYPE ST_Line
  UNDER ST_GeometricElement
  NOT INSTANTIABLE
  (ST_SelfIntersecting BOOLEAN,
   ST_Closed BOOLEAN,
   ST_InvalidLine BOOLEAN)
```

Definitional Rules

- 1) The attribute *ST_SelfIntersecting* is read-only. There are no GRANT statements granting EXECUTE privilege to the mutator function for *ST_SelfIntersecting*.
- 2) The attribute *ST_Closed* is read-only. There are no GRANT statements granting EXECUTE privilege to the mutator function for *ST_Closed*.
- 3) The attribute *ST_InvalidLine* is read-only. There are no GRANT statements granting EXECUTE privilege to the mutator function for *ST_InvalidLine*.

Description

- 1) The *ST_Area* type provides for public use:
 - a) an attribute *ST_SelfIntersecting*,
 - b) an attribute *ST_Closed*,
 - c) an attribute *ST_InvalidLine*,
 - d) a function *ST_Length(ST_Line)*.
- 2) The value of the attribute *ST_SelfIntersecting* is determined by the contents of the line. Its value is true if the interior of the line is self-intersecting. Otherwise, its value is false.

****Editor's Note 3-102****

The attribute *ST_SelfIntersecting* is defined in terms of the undefined phrase "self-intersecting". Define the phrase "self-intersecting".

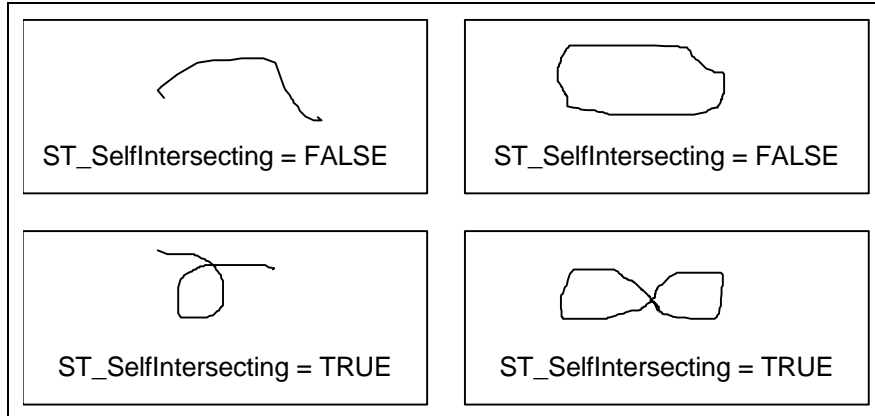


Figure 1 — ST_SelfIntersecting Visual Description

- 3) The value of *ST_Closed* is determined by the contents of the line. Its value is true if the first and last point of the line coincide. Otherwise, its value is false.

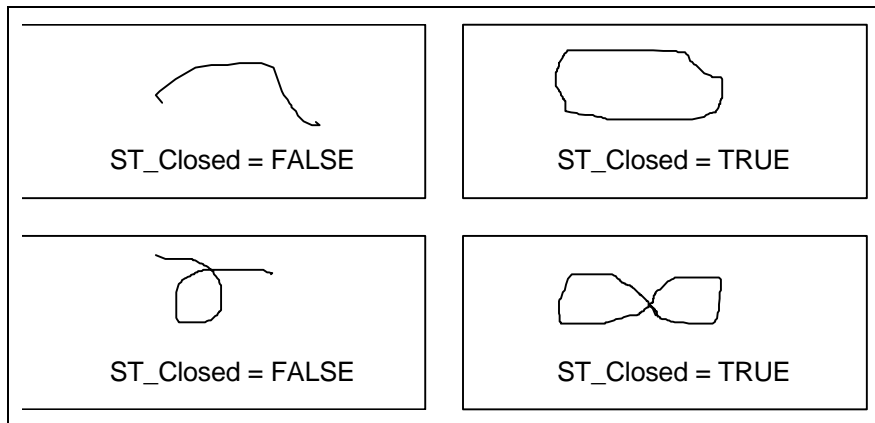


Figure 2 — ST_Closed Visual Description

- 4) The value of *ST_InvalidLine* is determined by the contents of the line. Its value is true if the line is formed properly, see the description of *ST_InvalidLine* in subtypes for specific details. Otherwise, its value is false.

****Editor's Note 3-101****

The text states that "Its [ST_InvalidLine] value is TRUE if the line is formed properly." As defined, the value is TRUE (i.e., the line is invalid) if it is improperly, not properly formed. This mistake is an example of why double negatives should be avoided.

The following is a partial solution:

Change the function to be ST_ValidLine. This proposed solution is recognized to be incomplete in that it does not give specific direction to the editor on what explicit changes to be made. It is offered merely as a suggested direction for overcoming the objection voiced in the comment.

****Editor's Note 3-100****

Spatial operations should be defined in ST_Spatial where the spatial referencing system is well known. ST_Length, which is currently defined on ST_Line and subtypes, requires spatial referencing to perform the operation.

10.1.2 ST_Length Function

Purpose

Determine the implementation-defined length of a line.

Definition

```
CREATE FUNCTION ST_Length
  (aline ST_Line)
  RETURNS DOUBLE PRECISION
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Length(ST_Line)* takes the following input parameters:
 - a) a *ST_Line* value *aline*.
- 2) The function *ST_Length(ST_Line)* determines the approximate length based on the coordinates and their spatial referencing. This will vary from implementation to implementation. The approximate length of the *ST_Line* value is returned.

****Editor's Note 3-114****

For the functions *ST_Length(ST_Line)* and *ST_Area(ST_Area)*, some description is required to explain that these function are virtual functions in the C++ sense. They are provides so the *ST_Length* function can be called on a column of type *ST_Line* (or *ST_Area* on a column of type *ST_Area*). Routine invocation will call the appropriate *ST_Length* function (or *ST_Area* function)for values of differing subtypes.

10.2 ST_Curve Type and Routines

10.2.1 ST_Curve Type

Purpose

ST_Curve is a locus of points that is continuous from the start point to the end point. The shape of the line is determined by the curve type (*ST_CurveType*) and the array of control points (*ST_Points*). The curve always starts at the first control point in the list and ends at the last control point in the array. The curve type describes the mathematical method used for generating the locus.

****Editor's Note 3-105****

6.3.2 *ST_Curve* and in 6.3.5 *ST_Path*, 6.4.2 *ST_Polygon*, 6.4.3.5 *ST_Contour*, 7 Metadata Abstract: These subclauses initially either depended upon object ADTs or would have benefited from a definition based upon object ADTs. In each case, the ADT can be comprised of a reference to one or a collection of other ADTs or enumerations. Object ADTs are no longer a part of SQL3. They have been replaced by reference types which have strict constraints on their usage. SQL3's precluding the use of reference types as a data type of an ADT attribute or as an element data type in a collection type seriously constrains the ability of SQL/MM to properly manage referenced spatial information. This can best be explained by a specific example, though the need is quite general in nature.

The detailed explanation will be included as a separate contribution from the USA. See SQL/MM:MAD-042.

Definition

```
CREATE TYPE ST_Curve
  UNDER ST_Line
  (ST_PrivatePoints ST_Point ARRAY[ST_MaxPointArray],
   ST_PrivateCurveType CHARACTER(1))
```

Definitional Rules

- 1) *ST_MaxPointArray* is the maximum cardinality of the *ST_Points* attribute. *ST_MaxPointArray* is an implementation-defined value.
- 2) The attribute *ST_PrivatePoints* is not part of the public interface. There are no GRANT statements granting EXECUTE privilege to the observer or mutator function for *ST_PrivatePoints*.
- 3) The attribute *ST_PrivateCurveType* is not part of the public interface. There are no GRANT statements granting EXECUTE privilege to the observer or mutator function for *ST_PrivateCurveType*.

Description

- 1) The *ST_Curve* type provides for public use:
 - a) a function *ST_Curve*(*ST_Point* ARRAY, CHARACTER),
 - b) a function *ST_Polyline*(*ST_Point* ARRAY),
 - c) a function *ST_CircularArc*(*ST_Point* ARRAY),
 - d) a function *ST_EllipticalArc*(*ST_Point* ARRAY),
 - e) a function *ST_Points*(*ST_Curve*),
 - f) a function *ST_Points*(*ST_Curve*, *ST_Point* ARRAY),
 - g) a function *ST_CurveType*(*ST_Curve*),
 - h) a function *ST_CurveType*(*ST_Curve*, CHARACTER),
 - i) a function *ST_Length*(*ST_Curve*),
 - j) a function *ST_GetPointAt*(*ST_Curve*, INTEGER),

- k) a function *ST_InsertPointAt*(*ST_Curve*, *ST_Point*, *INTEGER*),
 - l) a function *ST_AddPoint*(*ST_Curve*, *ST_Point*),
 - m) a function *ST_DeletePointAt*(*ST_Curve*, *INTEGER*),
 - n) a function *ST_UpdatePointAt*(*ST_Curve*, *ST_Point*, *INTEGER*).
- 2) The attribute *ST_SelfIntersecting* (inherited from *ST_Line*) is true if the interior of the curve is self-intersecting, otherwise, false.
- 3) The attribute *ST_Closed* (inherited from *ST_Line*) is true if the first and last point of the curve coincide, otherwise, false.
- 4) The attribute *ST_InvalidLine* (inherited from *ST_Line*) is true if the curve does not have the correct number of points *NPNT* to form *nseg* complete segments, otherwise false. *NPNT* is determined by the equation in the last column of Table 2 — *ST_CurveType* encoding.
- 5) The attribute *ST_PrivatePoints* contains the collection of points for the line.
- 6) The attribute *ST_PrivateCurveType* contains the type of curve.

****Editor's Note 3-085****

There can be a proliferation of SQL codes without a facilities to pass additional information when an exception is raised. The SQL/MM Rapporteur group discussed the possibility that of adding a standard mechanism to put a message into the diagnostic area.

10.2.2 ST_Curve Function

Purpose

Return a ST_Curve value.

Definition

```
CREATE FUNCTION ST_Curve
  (lpts ST_Point ARRAY[ST_MaxPointArray],
   ctype CHARACTER(1))
  RETURNS ST_Curve
  NULL CALL
  BEGIN
    DECLARE rtn ST_Curve;

    SET rtn = ST_Curve();
    SET rtn>>ST_Points = lpts;
    SET rtn>>ST_CurveType = ctype;
    RETURN rtn;
  END
```

Description

- 1) The function *ST_Curve(ST_Point ARRAY, CHARACTER)* takes the following input parameters:
 - a) a *ST_Point ARRAY* value *lpts*,
 - b) a *CHARACTER* value *ctype*.
- 2) The function *ST_Curve(ST_Point ARRAY, CHARACTER)* returns a value of type *ST_Curve*. This function constructs a *ST_Curve* value and sets the attribute *ST_Points* to *lpts* and sets the attribute *ST_CurveType* to *ctype*.

10.2.3 ST_Polyline Function

Purpose

Return a polyline value.

Definition

```
CREATE FUNCTION ST_Polyline
  (lpts ST_Point ARRAY[ST_MaxPointArray])
  RETURNS ST_Curve
  NULL CALL
  BEGIN
    DECLARE rtn ST_Curve;

    SET rtn = ST_Curve();
    SET rtn>>ST_Points = lpts;
    SET rtn>>ST_CurveType = 'L';
    RETURN rtn;
  END
```

Description

- 1) The function *ST_Polyline(ST_Point ARRAY)* takes the following input parameters:
 - a) a *ST_Point ARRAY* value *lpts*
- 2) The function *ST_Polyline(ST_Point ARRAY)* returns a value of type *ST_Curve*. This function constructs a *ST_Curve* value and sets the attribute *ST_Points* to *lpts* and set the attribute *ST_CurveType* to 'L'.

10.2.4 ST_CircularArc Function

Purpose

Return a circular arc value.

Definition

```
CREATE FUNCTION ST_CircularArc
  (lpts ST_Point ARRAY[ST_MaxPointArray])
  RETURNS ST_Curve
  NULL CALL
  BEGIN
    DECLARE rtn ST_Curve;

    SET rtn = ST_Curve();
    SET rtn>>ST_Points = lpts;
    SET rtn>>ST_CurveType = 'C';
    RETURN rtn;
  END
```

Description

- 1) The function *ST_CircularArc(ST_Point ARRAY)* takes the following input parameters:
 - a) a *ST_Point ARRAY* value *lpts*
- 2) The function *ST_CircularArc(ST_Point ARRAY)* returns a value of type *ST_Curve*. This function constructs a *ST_Curve* value and sets the attribute *ST_Points* to *lpts* and *ST_CurveType* to 'C'.

10.2.5 ST_EllipticalArc Function

Purpose

Return an elliptical arc value.

Definition

```
CREATE FUNCTION ST_EllipticalArc
  (lpts ST_Point ARRAY[ST_MaxPointArray])
  RETURNS ST_Curve
  NULL CALL
  BEGIN
    DECLARE rtn ST_Curve;

    SET rtn = ST_Curve();
    SET rtn>>ST_Points = lpts;
    SET rtn>>ST_CurveType = 'E';
    RETURN rtn;
  END
```

Description

- 1) The function *ST_EllipticalArc(ST_Point ARRAY)* takes the following input parameters:
 - a) a *ST_Point ARRAY* value *lpts*
- 2) The function *ST_EllipticalArc(ST_Point ARRAY)* returns a value of type *ST_Curve*. This function constructs a *ST_Curve* value and sets the attribute *ST_Points* to *lpts* and *ST_CurveType* to 'E'.

10.2.6 ST_Points Functions

Purpose

Get and set the points in a curve.

Definition

```

CREATE FUNCTION ST_Points
  (acurve ST_Curve)
  RETURNS ST_Point ARRAY[ST_MaxPointArray]
  NULL CALL
  BEGIN
    RETURN acurve>>ST_PrivatePoints;
  END

CREATE FUNCTION ST_Points
  (acurve ST_Curve RESULT,
   lpts ST_Point ARRAY[ST_MaxPointArray])
  RETURNS ST_Curve
  BEGIN
    DECLARE rtn ST_Curve;
    DECLARE NullValue CONDITION FOR SQLSTATE 'H3F03';

    SET rtn = acurve;
    IF lpts IS NULL THEN
      SIGNAL NullValue;
    ELSE
      SET rtn>>ST_PrivatePoints = lpts;
    END IF;
    RETURN rtn;
  END

```

Description

- 1) The function *ST_Points(ST_Curve)* takes the following input parameters:
 - a) a *ST_Point* ARRAY value *acurve*.
- 2) The function *ST_Points(ST_Curve)* returns the *ST_PrivatePoints* attribute of *acurve*.
- 3) The function *ST_Points(ST_Curve, ST_Point ARRAY)* takes the following input parameters:
 - a) a *ST_Curve* value *acurve*,
 - b) a *ST_Point* ARRAY value *lpts*.
- 4) The function *ST_Points(ST_Curve, ST_Point ARRAY)* applies the NOT NULL constraint when setting *lpts* to the *ST_PrivatePoints* attribute of *acurve*.

10.2.7 ST_CurveType Functions

Purpose

Get and set the type of curve.

Definition

```
CREATE FUNCTION ST_CurveType
  (acurve ST_Curve)
  RETURNS CHARACTER(1)
  NULL CALL
  BEGIN
    RETURN acurve>>ST_PrivateCurveType;
  END

CREATE FUNCTION ST_CurveType
  (acurve ST_Curve RESULT,
   ctype CHARACTER(1))
  RETURNS ST_Curve
  BEGIN
    DECLARE rtn ST_Curve;
    DECLARE NullValue CONDITION FOR SQLSTATE 'H3F03';
    DECLARE InvalidParameter CONDITION FOR SQLSTATE 'H3F02';

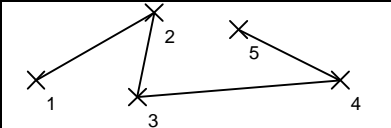
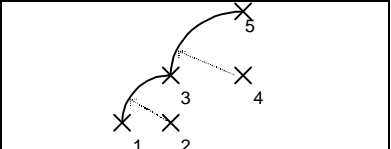
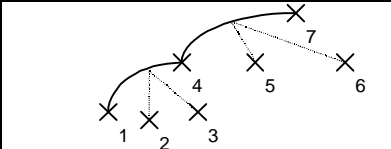
    SET rtn = acurve;
    IF ctype IS NULL THEN
      SIGNAL NullValue;
    ELSEIF ctype NOT IN ('L', 'C', 'E') THEN
      SIGNAL InvalidParameter;
    ELSE
      SET rtn>>ST_PrivateCurveType = ctype;
    END IF;
    RETURN rtn;
  END
```

Description

- 1) The function *ST_CurveType(ST_Curve)* takes the following input parameters:
 - a) a *ST_Curve* value *acurve*.
- 2) The function *ST_CurveType(ST_Curve)* returns the *ST_PrivateCurveType* attribute of *acurve*.
- 3) The function *ST_CurveType(ST_Curve, CHARACTER)* takes the following input parameters:
 - a) a *ST_Curve* value *acurve*,
 - b) a CHARACTER value *ctype*.
- 4) The function *ST_CurveType(ST_Curve, CHARACTER)* applies the NOT NULL constraint when setting *ctype* to the *ST_PrivateCurveType* in *acurve*.

The curve types are defined in Table 2 — *ST_CurveType* encoding.

Table 2 — ST_CurveType encoding

Code	Name	Description	Number of Points <i>n</i> is the number of segments (>0)
'L'	Polyline	<p>The line consists of a series of straight lines between two points connected end to end.</p>  <p>Polyline: 5 points, 4 segments</p>	1+n
'C'	Circular Arc	<p>The line consists of a series of circular arc segments connected end to end. Each segment is defined by three points. The first and third points are the end points and the middle point defines the center for the arc.</p>  <p>Circular Arc: 5 points, 2 segments</p>	1+2n
'E'	Elliptical Arc	<p>The line consists of a series of elliptical arc segments connected end to end. Each segment is defined by four points. The first and fourth points are the end points. The middle points define the foci of the ellipse.</p>  <p>Elliptical Arc: 7 points, 2 segments</p>	1+3n

****Editor's Note 3-078****

When defining a circular arc, the center point must be equidistant from the start and end points. There is a similar condition with the definition of an elliptical arc. If this condition is not valid, the *ST_InvalidLine* should be *true*.

A circular arc can also be defined by a start and end point and a third point located between the start and end point along the arc. The advantage of this definition is that it is less restrictive. It is unsure if there is a similar method of defining elliptical arcs. When it is determined, this alternate way of defining circular and elliptical arcs should be considered.

From Sam Bent:

Circular Arc:

There are three advantages to the three-point method:

- a. any three points define a circle, whereas the two-point-plus-center input has to satisfy an additional constraint (the center is equidistant from the two points). (Noted above)
- b. you can specify degenerate circles (straight lines).
- c. it's clear which direction around the circle you mean. If you can the two-point method, you'll need another parameter that says whether you mean to go clockwise around the circle, or ccw.

Ellipses:

I think it takes 5 (or 6) points to define a general conic. Circles and parabolas are special cases, and knowing it's a circle saves you two points. But ellipses and hyperbolas are the general case, you give all 5 (or 6).

From SQL/MM Rapporteur Group:

The center point method for circular arcs does not specify the rotation (clockwise or counter-clockwise) of the arc.

Ellipses must be coplanar.

****Editor's Note 3-103****

An alternative "fix" from the three-point method is to use bulge factors to define circular curves. They overcome the ambiguity of the center-point method, support intermittent straight line segments, and would provide an interface closer to a "better" implementation which would therefore eliminate the necessity for translation.

10.2.8 ST_Length Function

Purpose

Return the length of a curve.

Definition

```
CREATE FUNCTION ST_Length
  (acurve ST_Curve)
  RETURNS DOUBLE PRECISION
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Length(ST_Curve)* takes the following input parameters:
 - a) a *ST_Curve* value *acurve*.
- 2) The function *ST_Length(ST_Curve)* determines the approximate length of the curve based on the coordinates and their spatial referencing. This will vary from implementation to implementation. The approximate length of the *ST_Curve* value is returned.

10.2.9 ST_GetPointAt Function

Purpose

Return a point from a curve.

Definition

```
CREATE FUNCTION ST_GetPointAt
  (acurve ST_Curve,
   aposition INTEGER)
  RETURNS ST_Point
  NULL CALL
  BEGIN
    DECLARE a INTEGER;
    DECLARE rtn ST_Point;
    DECLARE InvalidPosition CONDITION FOR SQLSTATE 'H3F01';
    DECLARE EmptyList CONDITION FOR SQLSTATE 'H3F06';

    SET a = CARDINALITY(acurve>>ST_Points);
    IF a = 0 THEN
      SIGNAL EmptyList;
    ELSEIF aposition < 1 OR aposition > a THEN
      SIGNAL InvalidPosition;
    ELSE
      SET rtn = acurve>>ST_Points[aposition];
    END IF;
    RETURN rtn;
  END
```

Description

- 1) The function *ST_GetPointAt(ST_Curve, INTEGER)* takes the following input parameters:
 - a) a *ST_Curve* value *acurve*,
 - b) an INTEGER value *aposition*.
- 2) Let *CRD* be the cardinality of the *ST_PrivatePoints* attribute of *acurve*.
- 3) If the *aposition* is less than 1 (one) or greater than *CRD*, then an exception condition is raised: *SQL/MM Spatial exception - invalid position*.
- 4) The function *ST_GetPointAt(ST_Curve, INTEGER)* returns a point at *aposition* in the *ST_Points* attribute of *acurve*.

10.2.10 ST_InsertPointAt Function

Purpose

Insert a point into a curve.

Definition

```

CREATE FUNCTION ST_InsertPointAt
  (acurve ST_Curve RESULT,
   pt ST_Point,
   aposition INTEGER)
  RETURNS ST_Curve
  NULL CALL
  BEGIN
    DECLARE a INTEGER;
    DECLARE b INTEGER;
    DECLARE rtn ST_Curve;
    DECLARE InvalidPosition CONDITION FOR SQLSTATE 'H3F01';
    DECLARE DuplicateValue CONDITION FOR SQLSTATE 'H3F05';

    SET rtn = acurve;
    SET a = CARDINALITY(acurve>>ST_Points);
    IF aposition < 1 OR aposition > a+1 THEN
      SIGNAL InvalidPosition;
    ELSEIF a = 0 THEN
      SET rtn>>ST_Points = ARRAY[pt];
    ELSE
      IF aposition <= a THEN
        IF rtn>>ST_Points[aposition] = pt THEN
          SIGNAL DuplicateValue;
        END IF;
      ELSEIF aposition > 1 THEN
        IF rtn>>ST_Points[aposition-1] = pt THEN
          SIGNAL DuplicateValue;
        END IF;
      ELSE IF aposition = a+1 THEN
        SET rtn>>ST_Points =
          CONCATENATE(acurve>>ST_Points WITH ARRAY[pt]);
      ELSE
        SET rtn>>ST_Points =
          CAST(EMPTY as ST_Point ARRAY[ST_MaxPointArray]);
        SET b = 1;
        WHILE b < a DO
          IF b = aposition THEN
            SET rtn>>ST_Points =
              CONCATENATE(rtn>>ST_Points WITH ARRAY[pt]);
          END IF;
          SET rtn>>ST_Points =
            CONCATENATE(rtn>>ST_Points WITH
              ARRAY[acurve>>ST_Points[b]]);
          SET b = b + 1;
        END WHILE;
      END IF;
    END IF;
  END IF;
  RETURN rtn;
END

```

****Editor's Note 3-109****

Source: Peter Pistor, August 6, 1997. In *ST_InsertPointAt*, it might be that in the case where the cardinality of the list is 1 and the *aposition* is 1, the consistency check that ensures two subsequent points are not equal is not performed properly.

****Editor's Note 3-111****

Source: Peter Pistor, August 6, 1997. In the course of the discussion of SQL/MM:LGW-029r4, it became apparent that functions like *ST_InsertPointAt* are specified in a way to enforce the consistency checks (such as making sure that two consecutive points do not equal). It is felt that these consistency checks need to be documented in the Definitional Rules.

Description

- 1) The function *ST_InsertPointAt(ST_Curve, ST_Point, INTEGER)* takes the following input parameters:
 - a) a *ST_Curve* value *acurve*,
 - b) a *ST_Point* value *pt*,
 - c) an INTEGER value *aposition*.
- 2) Let *CRD* be the cardinality of the *ST_PrivatePoints* attribute of *acurve*.
- 3) If the *aposition* is less than 1 (one) or greater than *CRD*+1, then an exception condition is raised:
SQL/MM Spatial exception - invalid position.
- 4) The function *ST_InsertPointAt(ST_Curve, ST_Point, INTEGER)* inserts *pt* at *aposition* in the *ST_Points* attribute of *acurve*. To insert a point at the end of the array, specify a position value one greater than *CRD* or use the function *ST_AddPoint*.

10.2.11 ST_AddPoint Function

Purpose

Add a point to a curve.

Definition

```
CREATE FUNCTION ST_AddPoint
  (acurve ST_Curve RESULT,
   pt ST_Point)
RETURNS ST_Curve
NULL CALL
BEGIN
  RETURN ST_InsertPointAt(acurve, pt,
    CARDINALITY(acurve->>ST_Points)+1);
END
```

Description

- 1) The function *ST_AddPoint(ST_Curve, ST_Point)* takes the following input parameters:
 - a) a *ST_Curve* value *acurve*,
 - b) a *ST_Point* value *pt*.
- 2) The function *ST_AddPoint(ST_Curve, ST_Point)* appends *pt* to the end of the array in the *ST_Points* attribute of *acurve*.

10.2.12 ST_DeletePointAt Function

Purpose

Delete a point from a curve.

Definition

```

CREATE FUNCTION ST_DeletePointAt
  (acurve ST_Curve RESULT,
   aposition INTEGER)
  RETURNS ST_Curve
  NULL CALL
  BEGIN
    DECLARE a INTEGER;
    DECLARE b INTEGER;
    DECLARE c ST_Point;
    DECLARE rtn ST_Curve;
    DECLARE InvalidPosition CONDITION FOR SQLSTATE 'H3F01';

    SET rtn = acurve;
    SET rtn>>ST_Points = CAST(EMPTY as ST_Point
      ARRAY[ST_MaxPointArray]);
    SET a = CARDINALITY(acurve>>ST_Points);
    IF aposition < 1 OR aposition > a THEN
      SIGNAL InvalidPosition;
    ELSE
      SET b = 1;
      WHILE b <= a DO
        IF b <> aposition THEN
          SET c = ST_GetPointAt(acurve>>ST_Points, b);
          SET rtn = ST_AddPoint(rtn, c);
        END IF;
        SET b = b + 1;
      END WHILE;
    END IF;
    RETURN rtn;
  END

```

****Editor's Note 3-110****

Source: Peter Pistor, August 6, 1997. It seems that text is required to make known that the deletion of a point does not cause two equal points to become consecutive.

Description

- 1) The function *ST_DeletePointAt(ST_Curve, INTEGER)* takes the following input parameters:
 - a) a *ST_Curve* value *acurve*,
 - b) an *INTEGER* value *aposition*.
- 2) Let *CRD* be the cardinality of the *ST_PrivatePoints* attribute of *acurve*.
- 3) If the *aposition* is less than 1 (one) or greater than *CRD*, then an exception condition is raised: *SQL/MM Spatial exception - invalid position*.

- 4) The function *ST_DeletePointAt(ST_Curve, INTEGER)* deletes a point at *aposition* from the *ST_Points* attribute of *acurve*.

****Editor's Note 3-104****

ST_AddPoint, *ST_InsertPointAt*, and *ST_DeletePointAt*: Because many curve types require more than one point to specify a curve segment, having functions which add or delete a single point can result in curves with an inconsistent state.

The following is a partial solution:

Change these functions to add or delete curve segments instead of points. This proposed solution is recognized to be incomplete in that it does not give specific direction to the editor on what explicit changes to be made. It is offered merely as a suggested direction to overcoming the objection voiced in the comment.

10.2.13 ST_UpdatePointAt Function

Purpose

Update a point in a curve.

Definition

```
CREATE FUNCTION ST_UpdatePointAt
  (acurve ST_Curve RESULT,
   pt ST_Point,
   aposition INTEGER)
  RETURNS ST_Curve
  NULL CALL
  BEGIN
    RETURN ST_InsertPointAt(
      ST_DeletePointAt(acurve, aposition), pt, aposition);
  END
```

Description

- 1) The function *ST_UpdatePointAt(ST_Curve, ST_Point, INTEGER)* takes the following input parameters:
 - a) a *ST_Curve* value *acurve*,
 - b) a *ST_Point* value *pt*,
 - c) an *INTEGER* value *aposition*.
- 2) Let *CRD* be the cardinality of the *ST_PrivatePoints* attribute of *acurve*.
- 3) If the *aposition* is less than 1 (one) or greater than *CRD*, then an exception condition is raised: *SQL/MM Spatial exception - invalid position*.
- 4) The function *ST_UpdatePointAt(ST_Curve, ST_Point, INTEGER)* replaces the point at *aposition* in the *ST_Points* attribute of *acurve* with *pt*.

10.3 ST_Path Type and Routines

10.3.1 ST_Path Type

Purpose

The general notion of a path is a sequence of contiguous curves such that adjacent curves are joined at start points or end points only. Furthermore, the points composing all of the curves together must be able to be accessed in a one-directional, sequential order; this is accomplished by the appropriate choice for orientation for each of the individual oriented curves composing the path.

Definition

```
CREATE TYPE ST_Path
  UNDER ST_Line
  (ST_Curves ST_Curve ARRAY[ST_MaxCurveArray])
```

Definitional Rules

- 1) *ST_MaxCurveArray* is the maximum cardinality of the *ST_Curves* attribute. *ST_MaxCurveArray* is an implementation-defined value.

Description

- 1) The *ST_Path* type provides for public use:
 - a) an attribute *ST_Curves*,
 - b) a function *ST_Path(ST_Curve)*,
 - c) a function *ST_Path(ST_Curve ARRAY)*,
 - d) a function *ST_Length(ST_Path)*,
 - e) a function *ST_GetCurveAt(ST_Path, INTEGER)*,
 - f) a function *ST_AddCurve(ST_Path, ST_Curve)*,
 - g) a function *ST_InsertCurveAt(ST_Path, ST_Curve, INTEGER)*,
 - h) a function *ST_DeleteCurveAt(ST_Path, INTEGER)*,
 - i) a function *ST_UpdateCurveAt(ST_Path, ST_Curve, INTEGER)*.

- 2) The attribute *ST_SelfIntersecting* (inherited from *ST_Line*) is true if the interior of the path is self-intersecting. Otherwise, its value is false.
- 3) The attribute *ST_Closed* (inherited from *ST_Line*) is true if the first and last point of the path coincide. Otherwise, its value is false.
- 4) The attribute *ST_InvalidLine* (inherited from *ST_Line*) is true if:
 - a) The attribute *ST_Curves* is the NULL value or the EMPTY value.
 - b) If the *ST_InvalidLine* attribute is true for any curve in the *ST_Curves* attribute of the path.
Otherwise, the value is false.
- 5) The attribute *ST_Curves* contains the array of curves.

10.3.2 ST_Path Functions

Purpose

Return a ST_Path value.

Definition

```
CREATE FUNCTION ST_Path
  (cv ST_Curve)
  RETURNS ST_Path
  NULL CALL
  BEGIN
    DECLARE rtn ST_Path;

    SET rtn = ST_Path();
    SET rtn>>ST_Curves = ARRAY[cv];
    RETURN rtn;
  END

CREATE FUNCTION ST_Path
  (lcvs ST_Curve ARRAY[ST_MaxCurveArray])
  RETURNS ST_Path
  NULL CALL
  BEGIN
    DECLARE rtn ST_Path;

    SET rtn = ST_Path();
    SET rtn>>ST_Curves = lcvs;
    RETURN rtn;
  END
```

Description

- 1) The function *ST_Path(ST_Curve)* takes the following input parameters:
 - a) a *ST_Curve* value *cv*.
- 2) The function *ST_Path(ST_Curve)* returns a value of type *ST_Path*. This function constructs a *ST_Path* value and sets the attribute *ST_Curves* to *ARRAY[*cv*]*.
- 3) The function *ST_Path(ST_Curve ARRAY)* takes the following input parameters:
 - a) a *ST_Curve* ARRAY value *lcvs*.
- 4) The function *ST_Path(ST_Curve ARRAY)* returns a value of type *ST_Path*. This function constructs a *ST_Path* value and sets the attribute *ST_Curves* to *lcvs*.

10.3.3 ST_Length Function

Purpose

Return the length of a path.

Definition

```
CREATE FUNCTION ST_Length
  (apath ST_Path)
  RETURNS DOUBLE PRECISION
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Length(ST_Path)* takes the following input parameters:
 - a) a *ST_Path* value *apath*.
- 2) The function *ST_Length(ST_Path)* determines the approximate length based on the coordinates and their spatial referencing. This will vary from implementation to implementation. The approximate length of the *ST_Path* value is returned.

10.3.4 ST_GetCurveAt Function

Purpose

Return a curve from a path.

Definition

```
CREATE FUNCTION ST_GetCurveAt
  (apath ST_Path,
   aposition INTEGER)
  RETURNS ST_Curve
  NULL CALL
  BEGIN
    DECLARE a INTEGER;
    DECLARE rtn ST_Curve;
    DECLARE InvalidPosition CONDITION FOR SQLSTATE 'H3F01';
    DECLARE EmptyList CONDITION FOR SQLSTATE 'H3F06';

    SET a = CARDINALITY(apath>>ST_Curves);
    IF a = 0 THEN
      SIGNAL EmptyList;
    ELSEIF aposition < 1 OR aposition > a THEN
      SIGNAL InvalidPosition;
    ELSE
      SET rtn = apath>>ST_Curves[aposition];
    END IF;
    RETURN rtn;
  END
```

Description

- 1) The function *ST_GetCurveAt(ST_Path, INTEGER)* takes the following input parameters:
 - a) a *ST_Path* value *apath*,
 - b) an INTEGER value *aposition*.
- 2) Let *CRD* be the cardinality of the *ST_Curves* attribute of *acurve*.
- 3) If the *aposition* is less than 1 (one) or greater than *CRD*, then an exception condition is raised: *SQL/MM Spatial exception - invalid position*.
- 4) The function *ST_GetCurveAt(ST_Path, INTEGER)* returns a curve at *aposition* in the *ST_Curves* attribute of *apath*.

10.3.5 ST_InsertCurveAt Function

Purpose

Insert a curve into a path.

Definition

```

CREATE FUNCTION ST_InsertCurveAt
  (apath ST_Path RESULT,
   cv ST_Curve,
   aposition INTEGER)
RETURNS ST_Path
NULL CALL
BEGIN
  DECLARE a INTEGER;
  DECLARE b INTEGER;
  DECLARE c ST_Point;
  DECLARE d ST_Curve;
  DECLARE sp ST_Point;
  DECLARE ep ST_Point;
  DECLARE rtn ST_Path;
  DECLARE InvalidPosition CONDITION FOR SQLSTATE 'H3F01';
  DECLARE DuplicateValue CONDITION FOR SQLSTATE 'H3F05';
  DECLARE PathNotContinuous CONDITION FOR SQLSTATE 'H3F07';

  SET rtn = apath;
  SET a = CARDINALITY(apath>>ST_Curves);
  IF aposition < 1 OR aposition > a+1 THEN
    SIGNAL InvalidPosition;
  ELSEIF a = 0 THEN
    SET rtn>>ST_Curves = ARRAY[cv];
  ELSE
    IF aposition > 1 THEN
      SET d = ST_GetCurveAt(rtn, aposition-1);
      IF d = cv THEN
        SIGNAL DuplicateValue;
      ELSE
        SET sp = ST_GetPointAt(cv>>ST_Points, 1);
        SET c = ST_GetPointAt(d, CARDINALITY(d));
        IF c <> sp THEN
          SIGNAL PathNotContinuous;
        END IF;
      END IF;
    END IF;
    IF aposition <= a THEN
      SET d = ST_GetCurveAt(rtn, aposition);
      IF d = cv THEN
        SIGNAL DuplicateValue;
      ELSE
        SET ep = ST_GetPointAt(cv>>ST_Points,
          CARDINALITY(cv>>ST_Points));
        SET c = ST_GetPointAt(d, 1);
        IF c <> sp THEN
          SIGNAL PathNotContinuous;
        END IF;
      END IF;
    END IF;
  END IF;
  IF aposition = a+1 THEN
    SET rtn>>ST_Curves =

```

```

        CONCATENATE(aphath>>ST_Curves WITH ARRAY[cv]);
ELSE
    SET rtn>>ST_Curves =
        CAST(EMPTY as ST_Curve ARRAY[ST_MaxCurveArray]);
    SET b = 1;
    WHILE b < a DO
        IF b = aposition THEN
            SET rtn>>ST_Curves =
                CONCATENATE(rtn>>ST_Curves WITH ARRAY[cv]);
        END IF;
        SET rtn>>ST_Curves =
            CONCATENATE(rtn>>ST_Curves WITH
                ARRAY[aphath>>ST_Curves[b]]);
        SET b = b + 1;
    END WHILE;
END IF;
END IF;
END IF;
RETURN rtn;
END

```

Description

- 1) The function *ST_InsertCurveAt(ST_Path, ST_Curve, INTEGER)* takes the following input parameters:
 - a) a *ST_Path* value *aphath*,
 - b) a *ST_Curve* value *cv*,
 - c) an *INTEGER* value *aposition*.
- 2) Let *CRD* be the cardinality of the *ST_Curves* attribute of *acurve*.
- 3) If the *aposition* is less than 1 (one) or greater than *CRD+1*, then an exception condition is raised: *SQL/MM Spatial exception - invalid position*.
- 4) The function *ST_InsertCurveAt(ST_Path, ST_Curve, INTEGER)* inserts *cv* at *aposition* in the *ST_Curves* attribute of *aphath*. To insert a curve at the end of the array, specify a position value one greater *CRD* or use the function *ST_AddCurve*.

Editor's Note 3-107**

In Routine Descriptions: *ST_InsertCurveAt*, *ST_UpdateCurveAt*, *ST_DeleteCurveAt*:

The result of applying any of these routines individually and with a single *ST_Curve* as an argument could result in an inconsistent state.

The following is a partial solution:

For *ST_InsertCurveAt* and *ST_UpdateCurveAt*, the second argument should be *LIST(ST_Curve)* instead of *(ST_Curve)*, to enable a series of curves to be added to ensure that the net result is a contiguous path (if this becomes a constraint on paths). *ST_DeleteCurveAt* is more problematic in that, by itself, it most likely will result in a discontinuous path if the curve is intermediate to the path.

10.3.6 ST_AddCurve Function

Purpose

Add a curve to a path.

Definition

```
CREATE FUNCTION ST_AddCurve
  (apath ST_Path RESULT,
   cv ST_Curve)
RETURNS ST_Path
NULL CALL
BEGIN
  RETURN ST_InsertCurveAt(apath, cv, CARDINALITY(apath>>ST_Curves)+1);
END
```

Description

- 1) The function *ST_AddCurve(ST_Path, ST_Curve)* takes the following input parameters:
 - a) a *ST_Path* value *apath*,
 - b) a *ST_Curve* value *cv*.
- 2) The function *ST_AddCurve(ST_Path, ST_Curve)* appends *cv* to the end of the array in the *ST_Curves* attribute of *apath*.

10.3.7 ST_DeleteCurveAt Function

Purpose

Delete a curve in a path.

Definition

```

CREATE FUNCTION ST_DeleteCurveAt
  (apath ST_Path RESULT,
   aposition INTEGER)
RETURNS ST_Path
NULL CALL
BEGIN
  DECLARE a INTEGER;
  DECLARE b INTEGER;
  DECLARE c ST_Point;
  DECLARE d ST_Point;
  DECLARE sp ST_Point;
  DECLARE ep ST_Point;
  DECLARE rtn ST_Path;
  DECLARE InvalidPosition CONDITION FOR SQLSTATE 'H3F01';
  DECLARE PathNotContinuous CONDITION FOR SQLSTATE 'H3F07';

  SET rtn = apath;
  SET rtn>>ST_Curves =
    CAST(EMPTY as ST_Curve ARRAY[ST_MaxCurveArray]);
  SET a = CARDINALITY(apath>>ST_Curves);
  IF aposition < 1 OR aposition > a THEN
    SIGNAL InvalidPosition;
  ELSE
    IF aposition > 1 AND aposition < a THEN
      SET c = ST_GetCurveAt(apath>>ST_Curves, aposition-1);
      SET d = ST_GetCurveAt(apath>>ST_Curves, aposition+1);
      SET ep = ST_GetPointAt(c, CARDINALITY(c));
      SET sp = ST_GetPointAt(d, 1);
      IF sp <> ep THEN
        SIGNAL PathNotContinuous;
      END IF;
    END IF;
    SET b = 1;
    WHILE b <= a DO
      IF b <> aposition THEN
        SET c = ST_GetCurveAt(apath>>ST_Curves, b);
        SET rtn = ST_AddCurve(rtn, c);
      END IF;
      SET b = b + 1;
    END WHILE;
  END IF;
  RETURN rtn;
END

```

Description

- 1) The function *ST_DeleteCurveAt*(*ST_Path*, *INTEGER*) takes the following input parameters:
 - a) a *ST_Path* value *apath*,
 - b) an *INTEGER* value *aposition*.

- 2) Let *CRD* be the cardinality of the *ST_Curves* attribute of *acurve*.
- 3) If the *aposition* is less than 1 (one) or greater than *CRD*, then an exception condition is raised:
SQL/MM Spatial exception - invalid position.
- 4) The function *ST_DeleteCurveAt(ST_Path, INTEGER)* deletes a curve at *aposition* from the *ST_Curves* attribute of *apath*.

****Editor's Note 3-106****

Function *ST_DeleteCurveAt(ST_Path, INTEGER)* function deletes a specific curve in the specified path. In general, this will destroy the topological structure of *ST_Path*. This effect needs to be specified more explicitly. Similar comments hold for: *ST_InsertCurveAt*, and *ST_UpdateCurveAt*.

10.3.8 ST_UpdateCurveAt Function

Purpose

Update a curve in a path.

Definition

```
CREATE FUNCTION ST_UpdateCurveAt
  (apath ST_Path RESULT,
   cv ST_Curve,
   aposition INTEGER)
  RETURNS ST_Path
  NULL CALL
  BEGIN
    RETURN ST_InsertCurveAt(
      ST_DeleteCurveAt(apath, aposition), cv, aposition);
  END
```

Description

- 1) The function *ST_UpdateCurveAt(ST_Path, ST_Curve, INTEGER)* takes the following input parameters:
 - a) a *ST_Path* value *apath*,
 - b) a *ST_Curve* value *cv*,
 - c) an *INTEGER* value *aposition*.
- 2) Let *CRD* be the cardinality of the *ST_Curves* attribute of *acurve*.
- 3) If the *aposition* is less than 1 (one) or greater than *CRD*, then an exception condition is raised:
SQL/MM Spatial exception - invalid position.
- 4) The function *ST_UpdateCurveAt(ST_Path, ST_Curve, INTEGER)* replaces the curve at *aposition* in the *ST_Curves* attribute of *apath* with *cv*.

11 Area Data Types

11.1 ST_Area Type and Routines

11.1.1 ST_Area Type

Purpose

ST_Area is a type supertype for description of two dimensional geometric elements in vector format. The dimensionality of the space in which area objects exists, 2 or greater, is defined through the choice of coordinates.

Definition

```
CREATE TYPE ST_Area
  UNDER ST_GeometricElement
  NOT INSTANTIABLE
```

Description

- 1) The *ST_Area* type provides for public use:
 - a) a function *ST_Area(ST_Area)*.

11.1.2 ST_Area Function

Purpose

This function determines the implementation-defined area of an area.

Definition

```
CREATE FUNCTION ST_Area
  (anarea ST_Area)
  RETURNS DOUBLE PRECISION
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Area(ST_Area)* takes the following input parameters:
 - a) a *ST_Area* value *anarea*.
- 2) The function *ST_Area(ST_Area)* determines the approximate area of the area. This will vary from implementation to implementation. The approximate area of the *ST_Area* type is returned.

11.2 ST_Polygon Type and Routines

11.2.1 ST_Polygon Type

Purpose

The *ST_Polygon* data type is provided to define an enclosed area. An enclosed area is defined by one exterior boundary and zero or more interior boundaries and all the points which are both inside the exterior boundary and outside of the interior boundaries. Interior boundaries are commonly called holes or islands.

A properly constructed polygon is considered closed. During the construction of a polygon, some systems will create an incomplete polygon. The polygon may be in one of many states as specified in the attribute description of *ST_Closed*.

****Editor's Note 3-108****

Like *ST_Line*, an attribute like *ST_InvalidPolygon* (or *ST_ValidPolygon*) should be defined for *ST_Area* and its subtypes. Perhaps a common read-only attribute, *ST_Invalid* (or *ST_Valid*) should be defined on *ST_GeometricElement* and all of its subtypes. This attribute could indicate if the geometry is well defined. This would resolve also resolve the problem for any *ST_GeometricElement* definition.

Definition

```
CREATE TYPE ST_Polygon
  UNDER ST_Area
  (ST_PrivateExteriorBoundary ST_Path,
   ST_InteriorBoundaries ST_Path ARRAY[ST_MaxPathArray] DEFAULT EMPTY,
   ST_InsidePoint ST_Point,
   ST_Closed BOOLEAN)
```

Definitional Rules

- 1) *ST_MaxPathArray* is the maximum cardinality of the *ST_InteriorBoundaries* attribute. *ST_MaxPathArray* is an implementation-defined value.
- 2) The attribute *ST_PrivateExteriorBoundary* is not part of the public interface. There are no GRANT statements granting EXECUTE privilege to the observer or mutator function for *ST_PrivateExteriorBoundary*.
- 2) The attribute *ST_Closed* is read-only. There are no GRANT statements granting EXECUTE privilege to the mutator function for *ST_Closed*.

Description

- 1) The *ST_Polygon* type provides for public use:
 - a) an attribute *ST_InteriorBoundaries*,
 - b) an attribute *ST_InsidePoint*,
 - c) an attribute *ST_Closed*,
 - d) a function *ST_Polygon(ST_Path)* returns a value of type *ST_Polygon*,
 - e) a function *ST_Polygon(ST_Path, ST_Path ARRAY)*,

- f) a function *ST_Polygon*(*ST_Path*, *ST_Path* ARRAY, *ST_Point*),
 - g) a function *ST_ExteriorBoundary*(*ST_Polygon*),
 - h) a function *ST_ExteriorBoundary*(*ST_Polygon*, *ST_Path*),
 - i) a function *ST_Area*(*ST_Polygon*),
 - j) a function *ST_GetInteriorBoundaryAt*(*ST_Polygon*, *INTEGER*),
 - k) a function *ST_InsertInteriorBoundaryAt*(*ST_Polygon*, *ST_Path*, *INTEGER*),
 - l) a function *ST_AddInteriorBoundary*(*ST_Polygon*, *ST_Path*),
 - m) a function *ST_DeleteInteriorBoundaryAt*(*ST_Polygon*, *INTEGER*),
 - n) a function *ST_UpdateInteriorBoundaryAt*(*ST_Polygon*, *ST_Path*, *INTEGER*).
- 2) The attribute *ST_PrivateExteriorBoundary* contains the exterior boundary of the polygon.
- 3) The attribute *ST_InteriorBoundaries* contains the collection of interior boundaries. If the polygon does not have any holes, then this attribute should be set to the EMPTY value.
- 4) The attribute *ST_InsidePoint* is an arbitrary point that falls in the interior of the polygon. This point shall not intersect the exterior boundary or any interior boundaries. *ST_InsidePoint* is optional or may be generated automatically by the implementation.
- 5) The attribute *ST_Closed* is true if:
- a) all the paths in *ST_ExteriorBoundary* and *ST_InteriorBoundaries* are closed,
 - b) all paths in *ST_InteriorBoundaries* do not cross the exterior boundary or each other but the interior paths may touch the exterior path or other interior paths at a finite number of points,
 - c) all the paths in *ST_InteriorBoundaries* are contained within the exterior boundary and are not contained within any other interior path,
 - d) the *ST_InsidePoint* is in the interior of the polygon.
- Otherwise, false.

11.2.2 ST_Polygon Functions

Purpose

Return a ST_Polygon value.

Definition

```

CREATE FUNCTION ST_Polygon
  (eb ST_Path)
  RETURNS ST_Polygon
  NULL CALL
  BEGIN
    DECLARE rtn ST_Polygon;

    SET rtn = ST_Polygon();
    SET rtn>>ST_ExteriorBoundary = eb;
    SET rtn>>ST_InteriorBoundaries =
      CAST (EMPTY AS ST_Path ARRAY[ST_MaxPathArray]);
    SET rtn>>ST_InsidePoint = NULL;
    RETURN rtn;
  END

CREATE FUNCTION ST_Polygon
  (eb ST_Path,
   ibl ST_Path ARRAY[ST_MaxPathArray])
  RETURNS ST_Polygon
  NULL CALL
  BEGIN
    DECLARE rtn ST_Polygon;

    SET rtn = ST_Polygon();
    SET rtn>>ST_ExteriorBoundary = eb;
    SET rtn>>ST_InteriorBoundaries = ibl;
    SET rtn>>ST_InsidePoint = NULL;
    RETURN rtn;
  END

CREATE FUNCTION ST_Polygon
  (eb ST_Path,
   ibl ST_Path ARRAY[ST_MaxPathArray],
   ip ST_Point)
  RETURNS ST_Polygon
  NULL CALL
  BEGIN
    DECLARE rtn ST_Polygon;

    SET rtn = ST_Polygon();
    SET rtn>>ST_ExteriorBoundary = eb;
    SET rtn>>ST_InteriorBoundaries = ibl;
    SET rtn>>ST_InsidePoint = ip;
    RETURN rtn;
  END

```

Description

- 1) The function *ST_Polygon(ST_Path)* returns a value of type *ST_Polygon* takes the following input parameters:

- a) a *ST_Path* value *eb*.
- 2) The function *ST_Polygon(ST_Path)* returns a value of type *ST_Polygon*. This function constructs a *ST_Polygon* value and sets the attribute *ST_ExteriorBoundary* to *eb*, sets the attribute *ST_InteriorBoundaries* to the EMPTY value, and sets the attribute *ST_InsidePoint* to the NULL value.
- 3) The function *ST_Polygon(ST_Path, ST_Path ARRAY)* takes the following input parameters:
 - a) a *ST_Path* value *eb*,
 - b) a *ST_Path* ARRAY value *ibl*.
- 4) The function *ST_Polygon(ST_Path, ST_Path ARRAY)* returns a value of type *ST_Polygon*. This function constructs a *ST_Polygon* value and sets the attribute *ST_ExteriorBoundary* to *eb*, sets the attribute *ST_InteriorBoundaries* to *ibl*, and the attribute *ST_InsidePoint* to the NULL value.
- 5) The function *ST_Polygon(ST_Path, ST_Path ARRAY, ST_Point)* takes the following input parameters:
 - a) a *ST_Path* value *eb*,
 - b) a *ST_Path* ARRAY value *ibl*,
 - c) a *ST_Point* value *ip*.
- 6) The function *ST_Polygon(ST_Path, ST_Path ARRAY, ST_Point)* returns a value of type *ST_Polygon*. This function constructs a *ST_Polygon* value and sets the attribute *ST_ExteriorBoundary* to *eb*, sets the attribute *ST_InteriorBoundaries* to *ibl*, and sets the attribute *ST_InsidePoint* to *ip*.

11.2.3 ST_ExteriorBoundary Functions

Purpose

Get and set the exterior boundary of a polygon.

Definition

```
CREATE FUNCTION ST_ExteriorBoundary
  (apoly ST_Polygon)
  RETURNS ST_Path
  NULL CALL
  BEGIN
    RETURN apoly>>ST_PrivateExteriorBoundary;
  END

CREATE FUNCTION ST_ExteriorBoundary
  (apoly ST_Polygon RESULT,
   apath ST_Path)
  RETURNS ST_Polygon
  BEGIN
    DECLARE rtn ST_Polygon;
    DECLARE NullValue CONDITION FOR SQLSTATE 'H3F03';

    SET rtn = apoly;
    IF apath IS NULL THEN
      SIGNAL NullValue;
    ELSE
      SET rtn>>ST_PrivateExteriorBoundary = apath;
    END IF;
    RETURN rtn;
  END
```

Description

- 1) The function *ST_ExteriorBoundary(ST_Polygon)* takes the following input parameters:
 - a) a *ST_Polygon* value *apoly*.
- 2) The function *ST_ExteriorBoundary(ST_Polygon)* returns the *ST_PrivateExteriorBoundary* attribute of *apoly*.
- 3) The function *ST_ExteriorBoundary(ST_Polygon, ST_Path)* takes the following input parameters:
 - a) a *ST_Polygon* value *apoly*,
 - b) a *ST_Path* value *apath*.
- 4) The function *ST_ExteriorBoundary(ST_Polygon, ST_Path)* applies the NOT NULL constraint when setting *apath* to the *ST_PrivateExteriorBoundary* attribute in *apoly*.

11.2.4 ST_Area Function

Purpose

Determines the implementation-defined area of a polygon.

Definition

```
CREATE FUNCTION ST_Area
  (apoly ST_Polygon)
  RETURNS DOUBLE PRECISION
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Area(ST_Polygon)* takes the following input parameters:
 - a) a *ST_Polygon* value *apoly*.
- 2) The function *ST_Area(ST_Polygon)* determines the approximate area of the polygon. This will vary from implementation to implementation. The approximate area of the *ST_Area* value is returned.

11.2.5 ST_GetInteriorBoundaryAt Function

Purpose

Return an interior boundary from a polygon.

Definition

```
CREATE FUNCTION ST_GetInteriorBoundaryAt
  (apoly ST_Polygon,
   aposition INTEGER)
  RETURNS ST_Path
  NULL CALL
  BEGIN
    DECLARE a INTEGER;
    DECLARE rtn ST_Path;
    DECLARE InvalidPosition CONDITION FOR SQLSTATE 'H3F01';
    DECLARE EmptyList CONDITION FOR SQLSTATE 'H3F06';

    SET a = CARDINALITY(apoly>>ST_InteriorBoundaries);
    IF a = 0 THEN
      SIGNAL EmptyList;
    ELSEIF aposition < 1 OR aposition > a THEN
      SIGNAL InvalidPosition;
    ELSE
      SET rtn = apoly>>ST_InteriorBoundaries[aposition];
    END IF;
    RETURN rtn;
  END
```

Description

- 1) The function *ST_GetInteriorBoundaryAt(ST_Polygon, INTEGER)* takes the following input parameters:
 - a) a *ST_Polygon* value *apoly*,
 - b) an INTEGER value *aposition*.
- 2) Let *CRD* be the cardinality of the *ST_InteriorBoundaries* attribute of *apoly*.
- 3) If the *aposition* is less than 1 (one) or greater than *CRD*, then an exception condition is raised: *SQL/MM Spatial exception - invalid position*.
- 4) The function *ST_GetInteriorBoundaryAt(ST_Polygon, INTEGER)* returns the interior boundary at *aposition* in the *ST_InteriorBoundaries* attribute of *apoly*.

11.2.6 ST_InsertInteriorBoundaryAt Function

Purpose

Insert an interior boundary into a polygon.

Definition

```

CREATE FUNCTION ST_InsertInteriorBoundaryAt
  (apoly ST_Polygon RESULT,
   ib ST_Path,
   aposition INTEGER)
RETURNS ST_Polygon
LANGUAGE SQL
NULL CALL
BEGIN
  DECLARE a INTEGER;
  DECLARE b INTEGER;
  DECLARE rtn ST_Polygon;
  DECLARE InvalidPosition CONDITION FOR SQLSTATE 'H3F01';
  DECLARE DuplicateValue CONDITION FOR SQLSTATE 'H3F05';

  SET rtn = apoly;
  SET a = CARDINALITY(apoly>>ST_InteriorBoundaries);
  IF aposition < 1 OR aposition > a+1 THEN
    SIGNAL InvalidPosition;
  ELSEIF a = 0 THEN
    SET rtn>>ST_InteriorBoundaries = ARRAY[ib];
  ELSE
    IF aposition <= a THEN
      IF rtn>>ST_InteriorBoundaries[aposition] = ib THEN
        SIGNAL DuplicateValue;
      END IF;
    ELSEIF aposition > 1 THEN
      IF rtn>>ST_InteriorBoundaries[aposition-1] = ib THEN
        SIGNAL DuplicateValue;
      END IF;
    ELSE IF aposition = a+1 THEN
      SET rtn>>ST_InteriorBoundaries =
        CONCATENATE(apoly>>ST_InteriorBoundaries WITH ARRAY[ib]);
    ELSE
      SET rtn>>ST_InteriorBoundaries =
        CAST(EMPTY as ST_Path ARRAY[ST_MaxCurveArray]);
      SET b = 1;
      WHILE b < a DO
        IF b = aposition THEN
          SET rtn>>ST_InteriorBoundaries =
            CONCATENATE(rtn>>ST_InteriorBoundaries WITH
              ARRAY[ib]);
        END IF;
        SET rtn>>ST_InteriorBoundaries =
          CONCATENATE(rtn>>ST_InteriorBoundaries WITH
            ARRAY[apoly>>ST_InteriorBoundaries[b]]);
        SET b = b + 1;
      END WHILE;
    END IF;
  END IF;
  END IF;
  RETURN rtn;
END

```

Description

- 1) The function *ST_InsertInteriorBoundaryAt(ST_Polygon, ST_Path, INTEGER)* takes the following input parameters:
 - a) a *ST_Polygon* value *apoly*,
 - b) a *ST_Path* value *ib*,
 - c) an INTEGER value *aposition*.
- 2) Let *CRD* be the cardinality of the *ST_InteriorBoundaries* attribute of *apoly*.
- 3) If the *aposition* is less than 1 (one) or greater than *CRD+1*, then an exception condition is raised:
SQL/MM Spatial exception - invalid position.
- 4) The function *ST_InsertInteriorBoundaryAt(ST_Polygon, ST_Path, INTEGER)* inserts *ib* at *aposition* in the *ST_InteriorBoundaries* attribute of *apoly*.

11.2.7 ST_AddInteriorBoundary Function**Purpose**

Add an interior boundary to a polygon.

Definition

```
CREATE FUNCTION ST_AddInteriorBoundary
  (apoly ST_Polygon RESULT,
   ib ST_Path)
  RETURNS ST_Polygon
  NULL CALL
  BEGIN
    RETURN ST_InsertInteriorBoundaryAt(apoly, ib,
      CARDINALITY(apoly>>ST_InteriorBoundaries)+1);
  END
```

Description

- 1) The function *ST_AddInteriorBoundary(ST_Polygon, ST_Path)* takes the following input parameters:
 - a) a *ST_Polygon* value *apoly*,
 - b) a *ST_Path* value *ib*.
- 2) The function *ST_AddInteriorBoundary(ST_Polygon, ST_Path)* appends *ib* to the end of the array in the *ST_InteriorBoundaries* attribute of *apoly*.

11.2.8 ST_DeleteInteriorBoundaryAt Function

Purpose

Delete an interior boundary from a polygon.

Definition

```

CREATE FUNCTION ST_DeleteInteriorBoundaryAt
  (apoly ST_Polygon RESULT,
   aposition INTEGER)
RETURNS ST_Polygon
NULL CALL
BEGIN
  DECLARE a INTEGER;
  DECLARE b INTEGER;
  DECLARE c ST_Point;
  DECLARE rtn ST_Polygon;
  DECLARE InvalidPosition CONDITION FOR SQLSTATE 'H3F01';

  SET rtn = apoly;
  SET rtn>>ST_InteriorBoundaries =
    CAST(EMPTY as ST_Curve ARRAY[ST_MaxCurveArray]);
  SET a = CARDINALITY(apoly>>ST_InteriorBoundaries);
  IF aposition < 1 OR aposition > a THEN
    SIGNAL InvalidPosition;
  ELSE
    SET b = 1;
    WHILE b <= a DO
      IF b <> aposition THEN
        SET c =
          ST_GetInteriorBoundaryAt(apoly>>ST_InteriorBoundaries,
            b);
        SET rtn = ST_AddInteriorBoundary(rtn, c);
      END IF;
      SET b = b + 1;
    END WHILE;
  END IF;
  RETURN rtn;
END

```

Description

- 1) The function *ST_DeleteInteriorBoundaryAt(ST_Polygon, INTEGER)* takes the following input parameters:
 - a) a *ST_Polygon* value *apoly*,
 - b) an INTEGER value *aposition*.
- 2) Let *CRD* be the cardinality of the *ST_InteriorBoundaries* attribute of *apoly*.
- 3) If the *aposition* is less than 1 (one) or greater than *CRD*, then an exception condition is raised: *SQL/MM Spatial exception - invalid position*.
- 4) The function *ST_DeleteInteriorBoundaryAt(ST_Polygon, INTEGER)* deletes an exterior boundary at *aposition* from the *ST_InteriorBoundaries* attribute of *apoly*.

11.2.9 ST_UpdateInteriorBoundaryAt Function

Purpose

Update an interior boundary in a polygon.

Definition

```
CREATE FUNCTION ST_UpdateInteriorBoundaryAt
  (apoly ST_Polygon RESULT,
   ib ST_Path,
   aposition INTEGER)
  RETURNS ST_Polygon
  NULL CALL
  BEGIN
    RETURN ST_InsertInteriorBoundaryAt(
      ST_DeleteInteriorBoundaryAt(apoly, aposition), ib, aposition);
  END
```

Description

- 1) The function *ST_UpdateInteriorBoundaryAt(ST_Polygon, ST_Path, INTEGER)* takes the following input parameters:
 - a) a *ST_Polygon* value *apoly*,
 - b) a *ST_Path* value *ib*,
 - c) an INTEGER value *aposition*.
- 2) Let *CRD* be the cardinality of the *ST_InteriorBoundaries* attribute of *apoly*.
- 3) If the *aposition* is less than 1 (one) or greater than *CRD*, then an exception condition is raised: *SQL/MM Spatial exception - invalid position*.
- 4) The function *ST_UpdateInteriorBoundaryAt(ST_Polygon, ST_Path, INTEGER)* replaces the interior boundary at *aposition* in the *ST_InteriorBoundaries* attribute of *apoly* with *ib*.

11.3 ST_Region Type and Routines

11.3.1 ST_Region Type

Purpose

The *ST_Region* data type is provided to representing multiple, non-overlapping polygons.

Definition

```
CREATE TYPE ST_Region
  UNDER ST_Area
  (ST_PrivatePolygons ST_Polygon ARRAY[ST_MaxPolygonArray] DEFAULT EMPTY)
```

Definitional Rules

- 1) *ST_MaxPolygonArray* is the maximum cardinality of the *ST_PrivatePolygons* attribute. *ST_MaxPolygonArray* is an implementation-defined value.
- 2) The attribute *ST_PrivatePolygons* is not part of the public interface. There are no GRANT statements granting EXECUTE privilege to the observer or mutator function for *ST_PrivatePolygons*.
- 3) Let *PS* be the *ST_PrivatePolygons* attribute in a *ST_Region* value. If *PE* is an element of *PS*, then *PE* shall not overlap any other element of *PS*.

Description

- 1) The *ST_Region* type provides for public use:
 - a) a function *ST_Region*(*ST_Polygon* ARRAY),
 - b) a function *ST_Polygons*(*ST_Region*),
 - c) a function *ST_Polygons*(*ST_Region*, *ST_Polygon* ARRAY),
 - d) a function *ST_PolygonsClosed*(*ST_Region*),
 - e) a function *ST_Area*(*ST_Region*).
- 2) The attribute *ST_PrivatePolygons* contains the collection of polygons in the region.

11.3.2 ST_Region Function

Purpose

Return a ST_Region value.

Definition

```
CREATE FUNCTION ST_Region
  (lpoly ST_Polygon ARRAY[ST_MaxPolygonArray])
  RETURNS ST_Region
  NULL CALL
  BEGIN
    DECLARE rtn ST_Region;

    SET rtn = ST_Region();
    SET rtn>>ST_PrivatePolygons = lpoly;
    RETURN rtn;
  END
```

Description

- 1) The function *ST_Region(ST_Polygon ARRAY)* takes the following input parameters:
 - a) a *ST_Polygon ARRAY* value *lpoly*.
- 2) The function *ST_Region(ST_Polygon ARRAY)* returns a value of type *ST_Region*. This function constructs a *ST_Region* value and sets the attribute *ST_PrivatePolygons* to *lpoly*.

11.3.3 ST_Polygons Functions

Purpose

Get and set the polygons in a region.

Definition

```
CREATE FUNCTION ST_Polygons
  (aregion ST_Region)
  RETURNS ST_Polygon ARRAY[ST_MaxPolygonArray]
  NULL CALL
  BEGIN
    RETURN aregion>>ST_PrivatePolygons;
  END

CREATE FUNCTION ST_Polygons
  (aregion ST_Region RESULT,
   lpoly ST_Polygon ARRAY[ST_MaxPolygonArray])
  RETURNS ST_Region
  BEGIN
    DECLARE rtn ST_Region;
    DECLARE NullValue CONDITION FOR SQLSTATE 'H3F03';

    SET rtn = aregion;
    IF lpoly IS NULL THEN
      SIGNAL NullValue;
    ELSE
      rtn>>ST_PrivatePolygons = lpoly;
    END IF;
    RETURN rtn;
  END
```

Description

- 1) The function *ST_Polygons(ST_Region)* takes the following input parameters:
 - a) a *ST_Region* value *aregion*.
- 2) The function *ST_Polygons(ST_Region)* returns the *ST_PrivatePolygons* attribute in *aregion*.
- 3) The function *ST_Polygons(ST_Region, ST_Polygon ARRAY)* takes the following input parameters:
 - a) a *ST_Region* value *aregion*,
 - a) a *ST_Polygon ARRAY* value *lpoly*.
- 4) The function *ST_Polygons(ST_Region, ST_Polygon ARRAY)* applies the NOT NULL constraint when setting *lpoly* to the *ST_PrivatePolygons* attribute in *aregion*.

11.3.4 ST_PolygonsClosed Function

Purpose

Return true if all polygons in the region are closed.

Definition

```
CREATE FUNCTION ST_PolygonsClosed
  (aregion ST_Region)
  RETURNS BOOLEAN
  NULL CALL
  BEGIN
    DECLARE a INTEGER;
    DECLARE b INTEGER;
    DECLARE rtn BOOLEAN;

    SET rtn = TRUE;
    SET a = CARDINALITY(aregion>>ST_PrivatePolygons);
    SET b = 1;
    WHILE rtn = TRUE AND b <= a DO
      SET rtn = ST_Closed(aregion>>ST_PrivatePolygons[b]);
      SET b = b + 1;
    END WHILE;
    RETURN rtn;
  END
```

Description

- 1) The function *ST_PolygonsClosed* takes the following input parameters:
 - a) a *ST_Region* value *aregion*.
- 2) The function *ST_PolygonsClosed* is true if all the polygons in the *ST_PrivatePolygons* attribute of *aregion* are closed, otherwise false.

11.3.5 ST_Area Function

Purpose

Determines the implementation-defined area of a region.

Definition

```
CREATE FUNCTION ST_Area
  (aregion ST_Region)
  RETURNS DOUBLE PRECISION
  NULL CALL
  --
  -- See Description
  --
```

Description

- 1) The function *ST_Area(ST_Region)* takes the following input parameters:
 - a) a *ST_Region* value *aregion*.
- 2) The function *ST_Area(ST_Region)* determines the approximate area of the polygon. This will vary from implementation to implementation. The approximate area of the *ST_Area* value is returned.

12 Conformance

12.1 Introduction

This part of ISO/IEC 13249 specifies conforming SQL/MM Spatial implementations.

A conforming SQL/MM Spatial implementation shall provide the public Spatial data types and functions according to the associated Definitions and Description Rules specified in this part of ISO/IEC 13249.

A conforming SQL/MM Spatial implementation shall provide <SQL-invoked function>s each of whose <routine body> is either a <SQL routine body> or an <external body reference> that specifies PARAMETER STYLE SQL as defined in Subclause 11.42, “<SQL-invoked routine>“ in part 2 of ISO/IEC 9075.

A conforming SQL/MM Spatial implementation is not required to perform the exact sequences of actions defined in the Description Rules and in the <SQL routine body>s contained this International Standard, but shall achieve the same effect as those sequences.

12.2 Relationship to other International Standards

***** Editor’s Note 3-097 *****

Relationships to other International Standards to be supplied. This section needs to explain the dependency on ISO/IEC 9075.

12.3 Claims of conformance

Claims of conformance to this part of ISO/IEC 13249 shall state:

***** Editor’s Note 3-098 *****

Claims of conformance are to be supplied.

- 1) The definitions for all elements and actions that this part of ISO/IEC 13249 specifies as implementation-defined.

12.4 Extensions and options

A conforming implementation may provide support for additional implementation-defined routines defined using the Spatial data types.

An implementation remains conforming even if it provides user options to process Spatial routines in a non-conforming manner.

13 Status Codes

The character string value returned in an SQLSTATE parameter comprises a 2-character class value followed by a 3-character subclass value. The class value for each condition and the subclass value or values for each class value are specified in Table 3 — SQLSTATE class and subclass values.

The "Category" column has the following meanings: "S" means that the class value given corresponds to successful completion and is a completion condition; "W" means that the class value given corresponds to a successful completion but with a warning and is a completion condition; "N" means that the class value corresponds to a no-data situation and is a completion condition; "X" means that the class value given corresponds to an exception condition.

Table 3 — SQLSTATE class and subclass values

Category	Condition	Class	Subcondition	Subclass
X	SQL/MM Spatial exception	H3	invalid position	F01
X	SQL/MM Spatial exception	H3	invalid parameter	F02
X	SQL/MM Spatial exception	H3	null parameter	F03
X	SQL/MM Spatial exception	H3	invalid number of points	F04
X	SQL/MM Spatial exception	H3	duplicate value	F05
X	SQL/MM Spatial exception	H3	empty array	F06
X	SQL/MM Spatial exception	H3	path not continuous	F07

Index

Index entries appearing in **boldface** indicated a page where the word, phrase, attribute, routine, or type was described; index entries appearing in *italics* indicates a page where the attribute, routine, or type was declared; and index entries appearing in roman type indicate a page where the word, phrase, attribute, routine, or type was used.

—E—

Editor's Note

3-004, **7**
 3-078, **63**
 3-085, **55**
 3-091, **38**
 3-092, **35**
 3-093, **10**
 3-094, **33**
 3-097, **98**
 3-098, **98**
 3-099, **46**
 3-100, **52**
 3-101, **51**
 3-102, **50**
 3-103, **63**
 3-104, **70**
 3-105, **53**
 3-106, **80**
 3-107, **77**
 3-108, **82**
 3-109, **67**
 3-110, **69**
 3-111, **67**
 3-113, **37**
 3-114, **53**

—S—

SQLSTATE, 99

H3F01, 28, 65, 66, 67, 69, 70, 75, 76, 77, 79, 80,
 88, 89, 90, 91, 92
 H3F02, 18, 19, 20, 21, 28, 32
 H3F03, 41, 42, 49, 60, 61, 86, 95
 H3F05, 66, 76, 89
 H3F06, 65, 75, 88
 H3F07, 76, 79

ST_AddCurve, 71, 77, **78**, 79
 ST_AddInteriorBoundary, 83, **90**, 91
 ST_AddPoint, 55, 67, **68**, 69
 ST_Area, 9, 11, **19**, 20, 32, 34, **81**, 83, **87**, 93, **97**
 ST_Azimuth, 11, **21**
 ST_Buffer, 10, **15**
 ST_Centroid, 11, **16**
 ST_CircularArc, 54, **58**
 ST_Closed, 50, 51, 55, 72, 82, 83
 ST_ContainedBy, 11, **26**
 ST_Contains, 11, **25**
 ST_Coord, 48, **49**

ST_Coordinates, **38**, 39, **40**, 41, 42, 43, 44, 45, 48, 49
 ST_CoordMetaType, 38, 39, 40, 41, 42, 43, 45
 ST_Count, 11, **29**
 ST_Curve, 33, **53**, 54, 55, **56**, 57, 58, 59, 60, 61, 64,
 65, 66, 67, 68, 69, 70, 71, 73, 75, 76, 77, 78, 80, 91
 ST_Curves, 71, 72, 73, 75, 76, 77, 78, 79, 80
 ST_CurveType, 53, 54, 56, 57, 58, 59, **61**, 62
 ST_Decompose, 11, **33**
 ST_DeleteCurveAt, 71, **79**, 80
 ST_DeleteInteriorBoundaryAt, 83, **91**, 92
 ST_DeletePointAt, 55, **69**, 70
 ST_Difference, 11, **32**
 ST_Dimension, 11, **34**
 ST_Distance, 11, **22**
 ST_Elements, 10, 47
 ST_EllipticalArc, 54, **59**
 ST_Envelope, 11, **17**
 ST_ExplicitCoordinates, 39, **45**
 ST_ExteriorBoundary, 82, 83, 84, 85, **86**
 ST_ExtractAt, 11, **28**
 ST_GeometricElement, 10, 11, 12, 13, 30, 31, 32, 33,
46, 47, 48, 50, 81
 ST_GeometricGroup, **47**
 ST_Geometry, 10, 11, 12, 13, 16, 17, 18, 19, 20, 21,
 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 34
 ST_GetCurveAt, 71, **75**, 76, 79
 ST_GetInteriorBoundaryAt, 83, **88**, 91
 ST_GetPointAt, 54, **65**, 69, 76, 79
 ST_InsertCurveAt, 71, **76**, 77, 78, 80
 ST_InsertInteriorBoundaryAt, 83, **89**, 90, 92
 ST_InsertPointAt, 55, **66**, 67, 68, 70
 ST_InsidePoint, 82, 83, 84, 85
 ST_InteriorBoundaries, 82, 83, 84, 85, 88, 89, 90, 91,
 92
 ST_Intersection, 11, **30**
 ST_InvalidLine, 50, 51, 55, 63, 72
 ST_Is2D, 39, **44**
 ST_Is3D, **44**
 ST_IsEqual, 35, **37**
 ST_Length, 11, **18**, 50, **53**, 54, **64**, 71, **74**
 ST_Line, 18, 34, **50**, 53, 54, 55, 71, 72
 ST_MaxCurveArray, 71, 73, 77, 79, 89, 91
 ST_MaxGeometricElementArray, 10, 12, 47
 ST_MaxPathArray, 82, 84
 ST_MaxPointArray, 54, 56, 57, 58, 59, 60, 66, 69
 ST_MaxPolygonArray, 93, 94, 95
 ST_Meets, 11, **24**
 ST_Name, 35, **36**
 ST_Outside, 11, **27**
 ST_Overlaps, 11, **23**

ST_Path, 33, **71**, **73**, 74, 75, 76, 77, 78, 79, 80, 82,
 83, 84, 85, 86, 88, 89, 90, 92
 ST_Perimeter, 11, **20**
 ST_Point, 16, 21, 33, 34, **48**, 49, 54, 55, 56, 57, 58,
 59, 60, 65, 66, 67, 68, 69, 70, 76, 79, 82, 83, 84,
 85, 91
 ST_Points, 53, 54, 56, 57, 58, 59, **60**, 65, 66, 67, 68,
 69, 70
 ST_Polygon, 9, 17, 33, **82**, 83, **84**, 85, 86, 87, 88, 89,
 90, 91, 92, 93, 94, 95
 ST_Polygons, 93, **95**
 ST_PolygonsClosed, 93, **96**
 ST_Polyline, 54, **57**
 ST_PrivateCoord, 48, 49
 ST_PrivateCurveType, 54, 55, 61
 ST_PrivateExteriorBoundary, 82, 83, 86
 ST_PrivatePoints, 54, 55, 60, 65, 67, 69, 70, 75, 77,
 80
 ST_PrivatePolygons, 93, 94, 95, 96
 ST_PrivateSR, 10, 11, 12, 13
 ST_PrivateX, 38, 39, 41, 45
 ST_PrivateY, 38, 39, 42, 45
 ST_PrivateZ, 38, 39, 43, 44, 45
 ST_Region, 9, **93**, **94**, 95, 96, 97
 ST_SelfIntersecting, 50, 51, 55, 72
 ST_Spatial, **10**, 11, **12**, 13, 14, 15, 16, 17, 18, 19, 20,
 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34
 ST_SpatialReferencing, 10, 11, 12, 13, 14, **35**, **36**, 37,
 38
 ST_Transform, 10, 11, **14**
 ST_Union, 11, **31**
 ST_UpdateCurveAt, 71, **80**
 ST_UpdateInteriorBoundaryAt, 83, **92**
 ST_UpdatePointAt, 55, **70**
 ST_X, 38, 39, 40, **41**
 ST_Y, 39, 40, **42**
 ST_Z, 39, 40, **43**