Open Geospatial Consortium Inc.

Date: 2010-08-04

Reference number of this document: OGC 06-104r4

Version: 1.2.1

Status: Corregendum

Category: OpenGIS® Implementation Standard

Editor: John R. Herring

OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option

Document type:	OpenGIS® Implementation Standard
Document subtype:	(none)
Document stage:	Approved Corrigendum
Document language:	English

License Agreement

Contents

Page

Foreword vii

Introduction viii

1 Scope 2

2 Conformance 3

3 Normative references 3

4 Terms and definitions 3

5 Symbols and abbreviated terms 4
5.1 Abbreviations 4
5.2 Symbols 4

6 Architecture 5
6.1 Architecture — SQL implementation using predefined data types 5
6.1.1 Overview 5
6.1.2 Identification of feature tables and geometry columns 6
6.1.3 Identification of Spatial Reference Systems 7
6.1.4 Feature tables 7
6.1.5 Geometry tables 7
6.1.6 Text 9
6.1.7 Use of numeric data types 13
6.1.8 Notes on SQL/CLI access to Geometry values stored in binary form 13
6.2 Architecture — SQL implementation using Geometry Types 13
6.2.1 Overview 13
6.2.2 Identification of feature tables and geometry columns 14
6.2.3 Identification of Spatial Reference Systems 15
6.2.4 Feature tables 15
6.2.5 Background information on SQL User Defined Types 15
6.2.6 SQL Geometry Type hierarchy 16
6.2.7 Geometry values and spatial reference systems 17
6.2.8 Access to Geometry values in the SQL with Geometry Type case 17
6.2.9 Text 17

7 Clause component specifications 19
7.1 Components — Implementation of feature tables based on predefined data types 19
7.1.1 Conventions 19
7.1.2 Spatial reference system information 19
7.1.3 Geometry columns information 20
7.1.4 Feature tables 24
7.1.5 Geometry tables 25
7.1.6 Operators 29
7.2 Components — SQL with Geometry Types implementation of feature tables 29
7.2.1 Conventions 29
7.2.2 SQL Geometry Types 29
7.2.3 Feature tables 29
7.2.4 SQL routines for constructing a geometry object given its Well-known Text Representation ... 30

7.2.5 SQL routines for constructing a geometric object given its Well-known Binary Representation ... 30
7.2.6 SQL routines for obtaining Well-known Text Representation of a geometric object ... 31
7.2.7 SQL routines for obtaining Well-known Binary Representations of a geometric object ... 31
7.2.8 SQL routines on type Geometry ... 31
7.2.9 SQL routines on type Point ... 37
7.2.10 SQL routines on type Curve ... 40
7.2.11 SQL routines on type LineString ... 41
7.2.12 SQL functions on type Surface ... 42
7.2.13 SQL functions on type Polygon ... 43
7.2.14 SQL functions on type Polyhedral Surface ... 45
7.2.15 SQL routines on type GeomCollection ... 47
7.2.16 SQL routines on type MultiPoint ... 48
7.2.17 SQL routines on type MultiCurve ... 48
7.2.18 SQL routines on type MultiLineString ... 49
7.2.19 SQL routines on type MultiSurface ... 50
7.2.20 SQL routines on type Text ... 51

Annex A (normative) Abstract Test Suite ... 56
A.1 Purpose of this annex ... 56
A.2 Conformance Tests ... 56
A.2.1 Feature tables ... 56
A.2.2 Geometry tables or type ... 57
A.2.3 Spatial reference systems ... 57
A.2.4 Geometric format supported ... 58
A.2.5 Geometric categories supported ... 59
A.2.6 Text ... 59
A.3 Composite Conformance Clauses ... 60
A.4 Conformance Classes ... 60
A.4.1 Types of conformance classes ... 60

Annex B (informative) Comparison of Simple feature access/SQL and SQL/MM – Spatial ... 62

Annex C (informative) Conformance tests from version 1.1 ... 64
C.1 Purpose of this annex ... 64
C.2 Test data ... 64
C.2.1 Test data semantics ... 64
C.2.2 Test data points and coordinates ... 66
C.3 Conformance tests ... 69
C.3.1 Normalized geometry schema ... 69
C.3.2 Binary geometry schema ... 79
C.3.3 Geometry types and functions ... 89

Figures

Figure 1: Schema for feature tables using predefined data types 6

Figure 2: Example of geometry table for Polygon Geometry using SQL 8

Figure 3: Schema for feature tables using SQL with Geometry Types 14

Figure 4: Figure: SQL Geometry Type hierarchy 16

Figure C 1: Test Data Concept — Blue Lake vicinity map 65

Figure C 2: Points in the Blue Lake data set 67

Tables

Table 1: Example of geometry table for Polygon Geometry........9

Table 2: Column definitions for Annotation Text metadata........ 11

Table 3: Text metadata attributes........ 18

Table 4: Geometry type codes........ 22

Table A 1 - Equivalences between V1.1 and V1.2 complinace classes........ 61

Table B 1 — Comparison of SFA-SQL and SQL/MM: Spatial........ 62

Table C 1: Coordinates associated with each point in the Blue Lake data set........ 68

Table C 2: Queries to determine that test data are accessible via the normalized geometry schema 69

Table C 3: Queries to determine that test data are accessible via the binary geometry schema 79

Table C 4: Queries that accomplish the test of geometry types and functions 89

Foreword

This standard consists of the following parts, under the general title *Geographic information — Simple feature access*:

— *Part 1: Common architecture*

— *Part 2: SQL option*

This version supersedes all previous versions of OpenGIS® Simple Features Implementation Standard for SQL, including OGC 99-049 "OpenGIS Simple Features Specification for SQL Rev 1.1," and OGC 05-134 "OpenGIS® Implementation Specification for Geographic information - Simple feature access - Part 2: SQL option."

Version 1.1 of this standard is a profile of this version in the sense that it is a proper subset of the technology included here, except for some technical corrections and clarification.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium Inc. shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation

Introduction

This second part of OpenGIS® Simple Features Access (SFA), also called ISO 19125, is to define a standard Structured Query Language (SQL) schema that supports storage, retrieval, query and update of feature collections via the SQL Call-Level Interface (SQL/CLI) (ISO/IEC 9075-3:2003). A feature has both spatial and non-spatial attributes. Spatial attributes are geometry valued, and simple features are based on two-or-fewer dimensional geometric (point, curve and surface) entities in 2 or 3 spatial dimensions with linear or planar interpolation between vertices. This standard is dependent on the common architectural components defined in Part 1 of this standard.

In a SQL-implementation, a collection of features of a single type are stored as a "feature table" usually with some geometric valued attributes (columns). Each feature is primarily represented as a row in this feature table, and described by that and other tables logically linked to this base feature table using standard SQL techniques. The non-spatial attributes of features are mapped onto columns whose types are drawn from the set of SQL data types, potentially including SQL3 user defined types (UDT). The spatial attributes of features are mapped onto columns whose types are based on the geometric data types for SQL defined in this standard and its references. Feature-table schemas are described for two sorts of SQL-implementations: implementations based a more classical SQL relational model using only the SQL predefined data types and SQL with additional types for geometry. In any case, the geometric representations have a set of SQL accessible routines to support geometric behavior and query.

In an implementation based on predefined data types, a geometry-valued column is implemented using a "geometry ID" reference into a geometry table. A geometry value is stored using one or more rows in a single geometry table all of which have the geometry ID as part of their primary key. The geometry table may be implemented using standard SQL numeric types or SQL binary types; schemas for both are described in this standard.

The term “SQL with Geometry Types” is used to refer to a SQL-implementation that has been extended with a set of “Geometry Types.” In this environment, a geometry-valued column is implemented as a column whose SQL type is drawn from this set of Geometry Types. The mechanism for extending the type system of an SQL-implementation is through the definition of user defined User Defined Types. Commercial SQL-implementations with user defined type support have been available since mid-1997 and an ISO standard is available for UDT definition. This standard does not prescribe a particular UDT mechanism, but specifies the behavior of the UDTs through a specification of interfaces that must be supported. These interfaces are describe for SQL3 UDTs in ISO*/IEC 13249-3.*

Geographic information — Simple feature access —

Part 2: SQL option

1 Scope

This standard specifies an SQL schema that supports storage, retrieval, query and update of geospatial features with simple geometry via the SQL Call Level Interface (SQL/CLI) (ISO/IEC 9075-3:2003).

This standard

a) Establishes an architectural framework for the representation of feature,

b) Establishes a set of definitions for terms used within that framework,

c) Defines a simple geometric profile of ISO 19107 for the definition of the geometric attributes used in that framework

d) Describes a set of SQL Geometry Types together with SQL functions on those types.

The Geometry Types and Functions described in this standard represent a profile of ISO 13249-3. This standard does not attempt to standardize and does not depend upon any part of the mechanism by which Types are added and maintained in the SQL environment including the following:

a) The syntax and functionality provided for defining types;

b) The syntax and functionality provided for defining SQL functions;

c) The physical storage of type instances in the database;

d) Specific terminology used to refer to User Defined Types, for example, UDT.

This standard does standardize:

a) Names and geometric definitions of the SQL Types for Geometry;

b) Names, signatures and geometric definitions of the SQL Routines for Geometry.

This standard describes a feature access implementation in SQL based on a profile of ISO 19107. ISO 19107 is a behavioral standard and does not place any requirements on how to define the internal structures of Geometry Types in the schema. ISO 19107 does not place any requirements on when or how or who defines the Geometry Types. In particular, a compliant system may be shipped to the database user with the set of Geometry Types and Functions already built into the SQL-implementation, or with the set of Geometry Types and Functions supplied to the database user as a dynamically loaded extension to the SQL-implementation or in any other implementation consistent with the behavior described in this standard, in *ISO 19107* and in *ISO/IEC CD 13249-3:2006*.

2 Conformance

In order to conform to this standard, an implementation shall satisfy the requirements of one of the following three conformance classes, as well as the appropriate components of Part 1:

a) SQL implementation of feature tables based on predefined data types:

   1) using numeric SQL types for geometry storage and SQL/CLI access,

   2) using binary SQL types for geometry storage and SQL/CLI access;

b) SQL with Geometry Types implementation of feature tables supporting both textual and binary SQL/CLI access to geometry.

Annex B provides conformance tests for each implementation of this standard.

3 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

[1] ***ISO/IEC 9075-1, Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework)***

[2] ***ISO/IEC 9075-2, Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation)***

[3] ***ISO/IEC 9075-3, Information technology — Database languages — SQL — Part 3: Call-Level Interface (SQL/CLI)***

[4] ***ISO/IEC 9075-4, Information technology — Database languages — SQL — Part 4: Persistent Stored Modules (SQL/PSM)***

[5] ***ISO/IEC 9075-5, Information technology — Database languages — SQL — Part 5: Host Language Bindings (SQL/Bindings)***

[6] ***ISO/IEC CD 13249-3:2006(E) – Text for FDIS Ballot Information technology – Database languages – SQL Multimedia and Application Packages — Part 3: Spatial, May 15, 2006.***

[7] ***ISO 19107, Geographic information — Spatial schema***

[8] ***ISO 19109, Geographic information — Rules for application schema***

[9] ***ISO 19119, Geographic information — Services***

[10] ***ISO 19125-1, Geographic information — Simple feature access — Part 1: Common architecture***

4 Terms and definitions

For the purposes of this standard, the following terms and definitions apply.

4.1
feature table
table where the columns represent feature attributes, and the rows represent features

4.2
geographic feature
representation of real world phenomenon associated with a location relative to the Earth

5 Symbols and abbreviated terms

5.1 Abbreviations

FID	Feature ID column in the implementation of feature tables based on predefined data types
GID	Geometry ID column in the implementation of feature tables based on predefined data types
MM	Multimedia
SQL	Structured query language, not an acronym, pronounced as "sequel"
SQL/MM	SQL Multimedia and Application Packages
SRID	Spatial Reference System Identifier
SRTEXT	Spatial Reference System Well Known Text
WKB	Well-Known Binary (representation for example, geometry)
WKT	Well-Known Text
WKTR	Well-Known Text Representation

5.2 Symbols

nD	n-Dimensional, where n may be any integer
$\Re^n$	n-Dimensional coordinate space, where n may be any integer
$\varnothing$	empty set, the set having no members
$\cap$	intersection, operation on two or more sets
$\cup$	union, operation on two or more sets
—	difference, operation on two sets
$\in$	is a member of, relation between an element and a set
$\notin$	is not a member of
$\subset$	is a proper subset of, i.e. a smaller set not containing all of the larger
$\subseteq$	is a subset of
$\Leftrightarrow$	if and only if, logical equivalence between statements

$\Rightarrow$	implies, logical implication where the second follows from the first statement
$\exists$	there exists
$\forall$	for all
$\ni$	such that
f: D $\rightarrow$ R	Function "f" from domain "D" to range "R"
{ X | s }	set of "X" such that the statement "s" is TRUE
$\wedge$	and, logical intersection
$\vee$	or, logical union
$\neg$	not, logical negation
$=$	equal
$\neq$	not equal
$\leq$	less than or equal to
$<$	less than
$\geq$	greater than or equal to
$>$	greater than
∂	topological boundary operator, mapping a geometric object to its boundary

6 Architecture

6.1 Architecture — SQL implementation using predefined data types

6.1.1 Overview

This standard defines a schema for the management of feature table, Geometry, and Spatial Reference System information in an SQL-implementation based on predefined data types. This part of ISO 19125 does not define SQL functions for access, maintenance, or indexing of Geometry in an SQL-implementation based on predefined data types.

Figure 1 illustrates the schema to support feature tables, Geometry, and Spatial Reference Information in an SQL-implementation based on predefined data types.

a) The `GEOMETRY_COLUMNS` table describes the available feature tables and their Geometry properties.

b) The `SPATIAL_REF_SYS` table describes the coordinate system and transformations for Geometry.

c) The `FEATURE TABLE` stores a collection of features. A feature table's columns represent feature attributes, while rows represent individual features. The Geometry of a feature is one of its feature attributes; while logically a geometric data type, a Geometry Column is implemented as a foreign key to a geometry table.

d) The `GEOMETRY TABLE` stores geometric objects, and may be implemented using either standard SQL numeric types or SQL binary types.

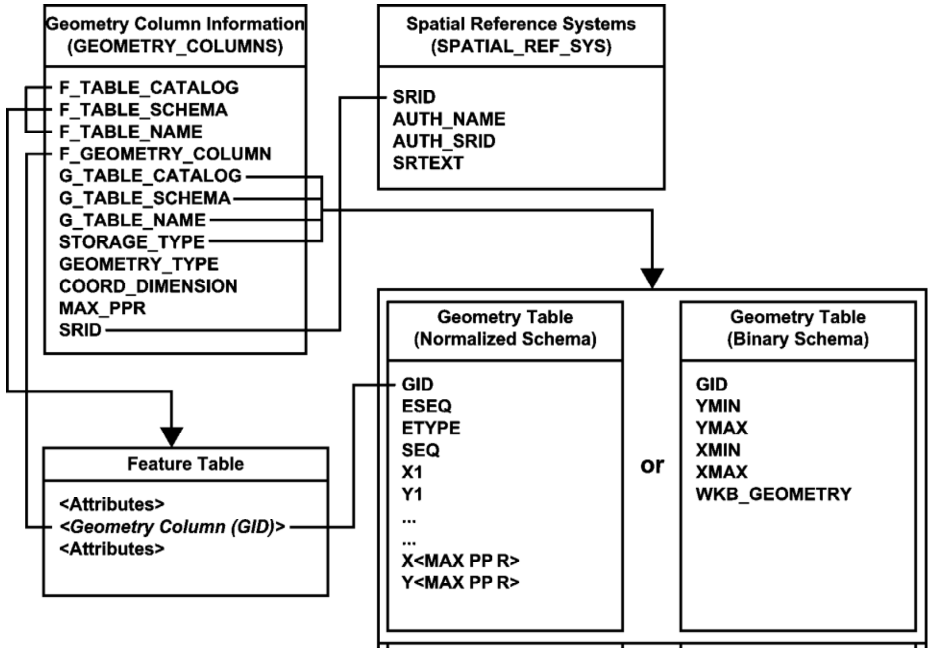


Figure 1: Schema for feature tables using predefined data types

Depending upon the storage type specified by the GEOMETRY_COLUMNS table, a geometric object is stored either as an array of coordinate values or as a single binary value. In the former case, predefined SQL numeric types are used for the coordinates and these numeric values are obtained from the geometry table until the geometric object has been fully reconstructed. In the latter case, the complete geometric object is obtained in the Well-known Binary Representation as a single value.

6.1.2 Identification of feature tables and geometry columns

Feature tables and Geometry columns are identified through the GEOMETRY_COLUMNS table. Each Geometry Column in the database has an entry in the GEOMETRY_COLUMNS table. The data stored for each geometry column consists of the following:

a) the identity of the feature table of which this Geometry Column is a member;

b) the name of the Geometry Column;

c) the spatial reference system ID (SRID) for the Geometry Column;

d) the type of Geometry for the Geometry column;

e) the coordinate dimension for the Geometry Column;

f) the identity of the geometry table that stores geometric objects for this Geometry Column;

g) the information necessary to navigate the geometry table in the case of normalized geometry storage.

6.1.3 Identification of Spatial Reference Systems

Every Geometry Column and every geometric entity is associated with exactly one Spatial Reference System. The Spatial Reference System identifies the coordinate system for all geometric objects stored in the column, and gives meaning to the numeric coordinate values for any geometric object stored in the column. Examples of commonly used Spatial Reference Systems include “Latitude Longitude” and “UTM Zone 10”.

The SPATIAL_REF_SYS table stores information on each Spatial Reference System in the database. The columns of this table are the Spatial Reference System Identifier (SRID), the Spatial Reference System Authority Name (AUTH_NAME), the Authority Specific Spatial Reference System Identifier (AUTH_SRID) and the Well-known Text description of the Spatial Reference System (SRTEXT). The Spatial Reference System Identifier (SRID) constitutes a unique integer key for a Spatial Reference System within a database.

Interoperability between clients is achieved via the SRTEXT column which stores the Well-known Text representation for a Spatial Reference System.

6.1.4 Feature tables

A feature is an abstraction of a real-world object. Feature attributes are columns in a feature table. Features are rows in a feature table. The Geometry of a feature is one of its feature attributes; while logically a geometric data type, a geometry column is implemented as a foreign key to a geometry table.

Relationships between features may be defined as foreign key references between feature tables.

6.1.5 Geometry tables

6.1.5.1 Normalized geometry schema

The normalized geometry schema stores the coordinates of geometric objects as predefined SQL numeric types. One or more coordinates (`X, Y` and optionally `Z` and `M` ordinate values) will be represented by pairs of numeric types in the geometry table, as shown in Figure 2. Each geometric object is identified by a key (GID) and consists of one or more primitive elements ordered by an element sequence (ESEQ). Each primitive element in the geometric object is distributed over one or more rows in the geometry table, identified by a primitive type (ETYPE), and ordered by a sequence number (SEQ).

The rules for geometric object representation in the normalized schema are defined as follows.

a) ETYPE designates the Geometry Type.

b) Geometric objects may have multiple elements. The ESEQ value identifies the individual elements.

c) An element may be built up from multiple parts (rows). The rows and their proper sequence are identified by the SEQ value.

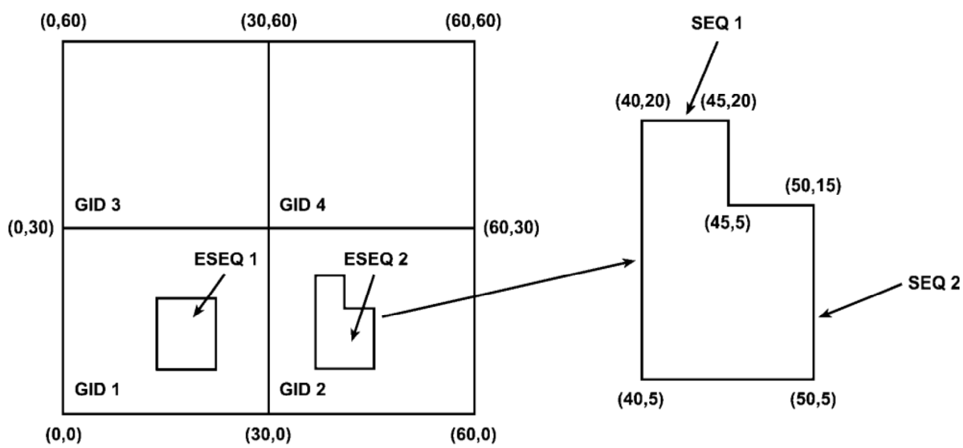
d) Polygons may contain holes, as described in the Geometry object model.

e) PolygonRings shall close when assembled from an ordered list of parts. The SEQ value designates the part order.

f) Coordinate pairs that are not used shall be set to Nil in complete sets (both X and Y). This is the only way to identify the end of the list of coordinates.

g) For geometric objects that continue onto an additional row (as defined by a constant element sequence number or ESEQ), the last Point of one row is equal to the first Point of the next.

h) There is no limit on the number of elements in the geometric object, or the number of rows in an element.



GID 1	ESEQ	ETYPE	SEQ	X0	Y0	X1	Y1	X2	Y2	X3	Y3	X4	Y4
1	1	3	1	0	0	0	30	30	30	30	0	0	0
1	2	3	1	10	10	10	20	20	20	20	10	10	10
2	1	3	1	30	0	30	30	60	30	60	0	30	0
2	2	3	1	40	5	40	20	45	20	45	15	50	15
2	2	3	1	50	15	50	5	40	5	Nil	Nil	Nil	Nil
3	1	3	1	0	30	0	60	30	60	30	30	0	30
4	1	3	1	30	30	30	60	60	60	60	30	30	30

Figure 2: Example of geometry table for Polygon Geometry using SQL

6.1.5.2 Binary geometry schema

The binary Geometry schema is illustrated in Table 1, uses GID as a key and stores the geometric object using the Well-known Binary Representation for Geometry (WKBGeometry). The geometry table includes the minimum bounding rectangle for the geometric object as well as the WKBGeometry for the geometric object. This permits construction of spatial indexes without accessing the actual geometric object structure, if desired.

Table 1: Example of geometry table for Polygon Geometry Using the Well-known Binary Representation for Geometry

GID	XMIN	YMIN	XMAX	YMAX	Geometry
1	0	0	30	30	< WKBGeometry >
2	30	0	60	30	< WKBGeometry >
3	0	30	30	60	< WKBGeometry >
4	30	30	60	60	< WKBGeometry >

6.1.5.3 SQL/MM geometry schema

The geometric attributes of a feature may also be specified using an extension of SQL/MM

6.1.6 Text

6.1.6.1 ANNOTATIONS Metadata Table

Each feature table/geometry column pair that has associated annotation text entities will be represented as a row in the ANNOTATIONS metadata table, or view. The data stored for each for annotation is:

- The identity of the feature table containing the text column
- The column in the feature table that contains the text entity key for associating multiple text elements to a single text entity
- A base scale for which the text placement is designed
- Optionally, a geometry column in the feature table for associated geometry representing an envelop for the text
- The identity of the text element table containing the geometry column
- The column name in the text element table that contains the text to be placed
- The column name in the text element table that contains the location geometry of the text
- The column name in the text element table that contains the optional leader line that may be associated with the text.
- The column name in the text element table that contains text rendering data
- Default values for the text element, either by value of by using “sql-value expressions” that can be evaluated on the feature entry associated to the text.
- Default values for the text rendering data, as a collection of XML elements as a single text string.

The base scale for all size values that will be given in points[1] (1 point = 0.35146 mm). Each text object has a font size from the style. To enable annotation text, a mechanism is needed whereby text may be defined in points but (usually) based on a specific map scale. Thus, a text object would be placed using a font size of 24 point at 1:1000000 and client-rendering engines would use this information to scale the text size appropriate to changes in the map scale. This base scale would be stored once in the metadata. Any point size values in the metadata attributes column (see below) or in individual rows would be relative to this value, as would letter-spacing and word-spacing, stroke-width (for text and leader line) and both vertical and horizontal margins. Application may round to the nearest point during scaling.

6.1.6.2 Table or View Constructs for structural metadata

The following CREATE TABLE statement creates an appropriately structured table to be included in the schema, describing how text is stored in a feature table. This should be either an actual metadata table or an updateable view so that insertion of reference system information can be done directly with SQL.

Note that there is no requirement that the annotated feature have any other attributes. Unattributed annotations are in essence context-free, and may be used to place any text on the data, such as collection metadata or notes to user about unusual situations of which he may wish to be aware.

[1] There is some minor disagreement on the standard for a text point. The US-UK standard is 1/72.27 inch, Adobe Postscript use 1/72 inch. Traditional typesetters use 1/64 inch and European (based on a French standard) use approximately 1/67 inch. At the sizes of normal text at normal display scale, none of these differences are significant. These manor differences man make fine scale comparison of output difficult to make.

```
CREATE TABLE ANNOTATION_TEXT_METADATA AS
{
  F_TABLE_CATALOG                    AS CHARACTER VARYING NOT NULL,
  F_TABLE_SCHEMA                     AS CHARACTER VARYING NOT NULL,
  F_TABLE_NAME                       AS CHARACTER VARYING NOT NULL,
  F_TEXT_KEY_COLUMN                  AS CHARACTER VARYING NOT NULL,
  F_TEXT_ENVELOPE_COLUMN             AS CHARACTER VARYING NOT NULL,

  A_ELEMENT_TABLE_CATALOG            AS CHARACTER VARYING NOT NULL,
  A_ELEMENT_TABLE_SCHEMA             AS CHARACTER VARYING NOT NULL,
  A_ELEMENT_TABLE_NAME               AS CHARACTER VARYING NOT NULL,

  A_ELEMENT_TEXT_KEY_COLUMN          AS CHARACTER VARYING NOT NULL
  A_ELEMENT_TEXT_SEQ_COLUMN          AS CHARACTER VARYING NOT NULL
  A_ELEMENT_TEXT_VALUE_COLUMN        AS CHARACTER VARYING NOT NULL,
  A_ELEMENT_TEXT_LEADERLINE_COLUMN   AS CHARACTER VARYING NOT NULL,
  A_ELEMENT_TEXT_LOCATION_COLUMN     AS CHARACTER VARYING NOT NULL,
  A_ELEMENT_TEXT_ATTRIBUTES_COLUMN   AS CHARACTER VARYING NOT NULL,

  A_MAP_BASE_SCALE                   AS NUMBER NOT NULL,
  A_TEXT_DEFAULT_EXPRESSION          AS CHARACTER VARYING,
  A_TEXT_DEFAULT_ATTRIBUTES          AS CHARACTER VARYING
}
```

Note that there are no constraints on row in this table, allowing a single feature table/geometry column pair to be annotated using text from different feature table columns.

6.1.6.3 Field Description

The fields in the Annotations metadata information view are given in

Table 2: Column definitions for Annotation Text metadata

Columns	Description
F_TABLE_ CATALOG, SCHEMA, NAME	the fully qualified name of the feature table containing the geometry column to be annotated
F_TEXT_ KEY_COLUMN. ENVELOPE_CO LUMN,	The names of the column in the feature table that contain: A KEY for the text to which the text elements can use as a point of aggregation. An ENVELOPE_COLUMN that contains a geometry object that acts as an envelope for the set of text elements in this text entity. This column should also be a valid geometry column.

Columns	Description
A_ELEMENT_TABLE CATALOG, SCHEMA, NAME	the fully qualified name of the text element table containing the text elements used for the F_Text columns column defined above
A_TEXT_ELEMENT KEY_COLUMN SEQ_COLUMN VALUE_COLUM N LEADERLINE_ COLUMN LOCATUIN_CO LUMN ATTRIBUTES_ COLUMN	The names of the columns in the ELEMENT_TABLE that contain the: a) The foreign KEY for the text entity as specified in the F_TEXT_KEY_COLUMN. b) A sequence (SEQ) column which will be used to order the text elements in this text entity. Any sortable type is valid for this column in the table, although integers would be the obvious choice. c) A text string VALUE for this text element. d) The LEADERLINE for this text element — if it has one (should also be a geometry column). e) The LOCATION for this text element (should also be a geometry column). f) The local text ATTRIBUTES providing the opportunity to override the text attributes currently in force. This is an XML type, and will be a collection of XML elements each describing a text attribute of the current text element. Unspecified attributes take the value most recently defined.
A_MAP_BASE_SCALE	The base scale for all size values that will be given in points[2] (1 point = 0.35146 mm). Each text object has a font size from the style. To enable annotation text, a mechanism is needed whereby text may be defined in points but (usually) based on a specific map scale. Thus, a text object would be placed using a font size of 24 point at 1:1000000 and client-rendering engines would use this information to scale the text size appropriate to changes in the map scale. This base scale would be stored once in the metadata. Any point size values in the metadata attributes column (see below) or in individual rows would be relative to this value, as would letter-spacing and word-spacing, stroke-width (for text and leader line) and both vertical and horizontal margins. Application may round to the nearest point during scaling.

[2] There is some minor disagreement on the standard for a text point. The US-UK standard is 1/72.27 inch, Adobe Postscript use 1/72 inch. Traditional typesetters use 1/64 inch and European (based on a French standard) use approximately 1/67 inch. At the sizes of normal text at normal display scale, none of these differences are significant. These manor differences man make fine scale comparison of output difficult to make.

Columns	Description
A_TEXT_DEFAULT_ EXPRESSION ATTRIBUTES	The default values for the corresponding "A_TEXT_" columns above, for cases where these columns are NULL in the feature table. They may be values or "query" expressions in terms of other columns in the database. These defaults shall be overridden on a row by row basis when the corresponding columns in the feature table row are not NULL. Formats, which are large text strings, and interpretation for these columns are discussed in Part 1.

6.1.7 Use of numeric data types

SQL-implementations usually provide several numeric data types. In this standard, the use of a numeric data type in examples is not meant to be binding. The data type of any particular column can be determined, and casting operators between similar data types are available. Any particular implementation may use alternative data types as long as casting operations shall not lead to difficulties.

6.1.8 Notes on SQL/CLI access to Geometry values stored in binary form

SQL/CLI provides standard mechanisms to bind character, numeric and binary data values.

This subclause describes the process of retrieving geometric object values for the case where the binary storage alternative is chosen.

The WKB_GEOMETRY column in the geometry table is accessed in SQL/CLI as one of the binary SQL data types (SQL_BINARY, SQL_VARBINARY, or SQL_LONGVARBINARY).

EXAMPLE The application would use the SQL_C_BINARY value for the fCType parameter of SQLBindCol (or SQLGetData) in order to describe the application data buffer that shall receive the fetched Geometry data value. Similarly, a dynamic parameter whose value is a Geometry would be described using the SQL_C_BINARY value for the fCType parameter of SQLBindParameter.

This allows binary values to be both retrieved from and inserted into the geometry tables.

6.2 Architecture — SQL implementation using Geometry Types

6.2.1 Overview

This standard defines a schema for the management of feature table, Geometry, and Spatial Reference System information in an SQL-implementation with a Geometry Type extension.

Figure 3 illustrates the schema to support feature tables, Geometry, and Spatial Reference Information in an SQL-implementation with a Geometry Type extension.

a) The GEOMETRY_COLUMNS table describes the available feature tables and their Geometry properties.

b) The SPATIAL_REF_SYS table describes the coordinate system and transformations for Geometry.

c) The feature table stores a collection of features. A feature table's columns represent feature attributes, while rows represent individual features. The Geometry of a feature is one of the feature attributes, and is an SQL Geometry Type.

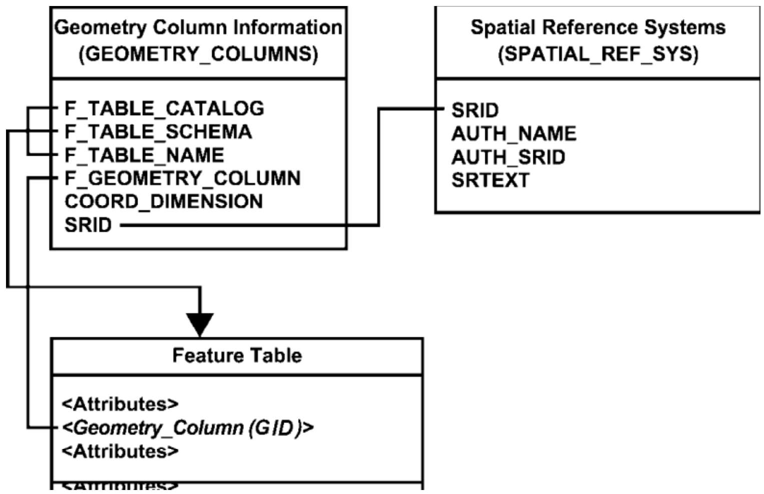


Figure 3: Schema for feature tables using SQL with Geometry Types

6.2.2 Identification of feature tables and geometry columns

Feature tables and Geometry columns are identified through the GEOMETRY_COLUMNS table. Each Geometry Column in the database has an entry in the GEOMETRY_COLUMNS table. The data stored for each geometry column consists of the following:

a) the identity of the feature table of which this Geometry Column is a member;

b) the name of the Geometry Column;

c) the spatial reference system ID for the Geometry Column;

d) the coordinate dimension for the Geometry column;

The columns in the GEOMETRY_COLUMNS table for the SQL with Geometry Types environment are a subset of the columns in the GEOMETRY_COLUMNS table defined for the SQL-implementation based on predefined data types.

An alternative method for identification of feature tables and Geometry Columns may be available for SQL-implementations with Geometry Types. In the SQL-implementation with Geometry Types, the Geometry Column may be represented as a row in the COLUMNS metadata view of the SQL INFORMATION_SCHEMA. Spatial Reference System Identity and coordinate dimension is, however, not a standard part of the

SQL INFORMATION_SCHEMA. To access this information, the GEOMETRY_COLUMNS table would still need to be referenced.

6.2.3 Identification of Spatial Reference Systems

Every Geometry Column is associated with a Spatial Reference System. The Spatial Reference System identifies the coordinate system for all geometric objects stored in the column, and gives meaning to the numeric coordinate values for any geometric object stored in the column. Examples of commonly used Spatial Reference Systems include “Latitude Longitude” and “UTM Zone 10”.

The SPATIAL_REF_SYS table stores information on each Spatial Reference System in the database. The columns of this table are the Spatial Reference System Identifier (SRID), the Spatial Reference System Authority Name (AUTH_NAME), the Authority Specific Spatial Reference System Identifier (AUTH_SRID) and the Well-known Text description of the Spatial Reference System (SRTEXT). The Spatial Reference System Identifier (SRID) constitutes a unique integer key for a Spatial Reference System within a database.

Interoperability between clients is achieved via the SRTEXT column which stores the Well-known Text representation for a Spatial Reference System.

6.2.4 Feature tables

A feature is an abstraction of a real-world object. Feature attributes are columns in a feature table. Features are rows in a feature table. The Geometry of a feature is stored in a Geometry Column whose type is drawn from a set of SQL Geometry Types.

Relationships between features may be defined as foreign key references between feature tables.

6.2.5 Background information on SQL User Defined Types

The term User Defined Type (UDT) refers to a data type that extends the SQL type system.

UDT types can be used to define the column types for tables, this allows values stored in the columns of a table to be instances of UDT.

SQL functions may be declared to take UDT values as arguments, and return UDT values as results.

An UDT may be defined as a subtype of another UDT, referred to as its supertype. This allows an instance of the subtype to be stored in any column where an instance of the supertype is expected and allows an instance of the subtype to be used as an argument or return value in any SQL function that is declared to use the supertype as an argument or return value.

The above definition of UDT is value based.

SQL implementations that support User Defined Types may also support the concept of References to User Defined Types instances that are stored as rows in a table whose type corresponds to the type of the User Defined Type. The terms RowType and Reference to RowType are also used to describe such types.

This standard allows Geometry Types to be implemented as either pure value based Types or as Types that support persistent References.

The Types for Geometry are defined in black-box terms, i.e. all access to information about a Geometry Type instance is through SQL functions. No attempt is made to distinguish functions that may access Type instance attributes (such as the dimension of a geometric object) from functions that may compute values given a Type instance (such as the centroid of a Polygon). In particular, an implementation of this standard would be free to nominate any set of functions as observer methods on attributes of a User Defined Type, as long as the signatures of the SQL functions described in this standard are preserved.

6.2.6 SQL Geometry Type hierarchy

The SQL Geometry Types are organized into a type hierarchy shown in Figure 4.

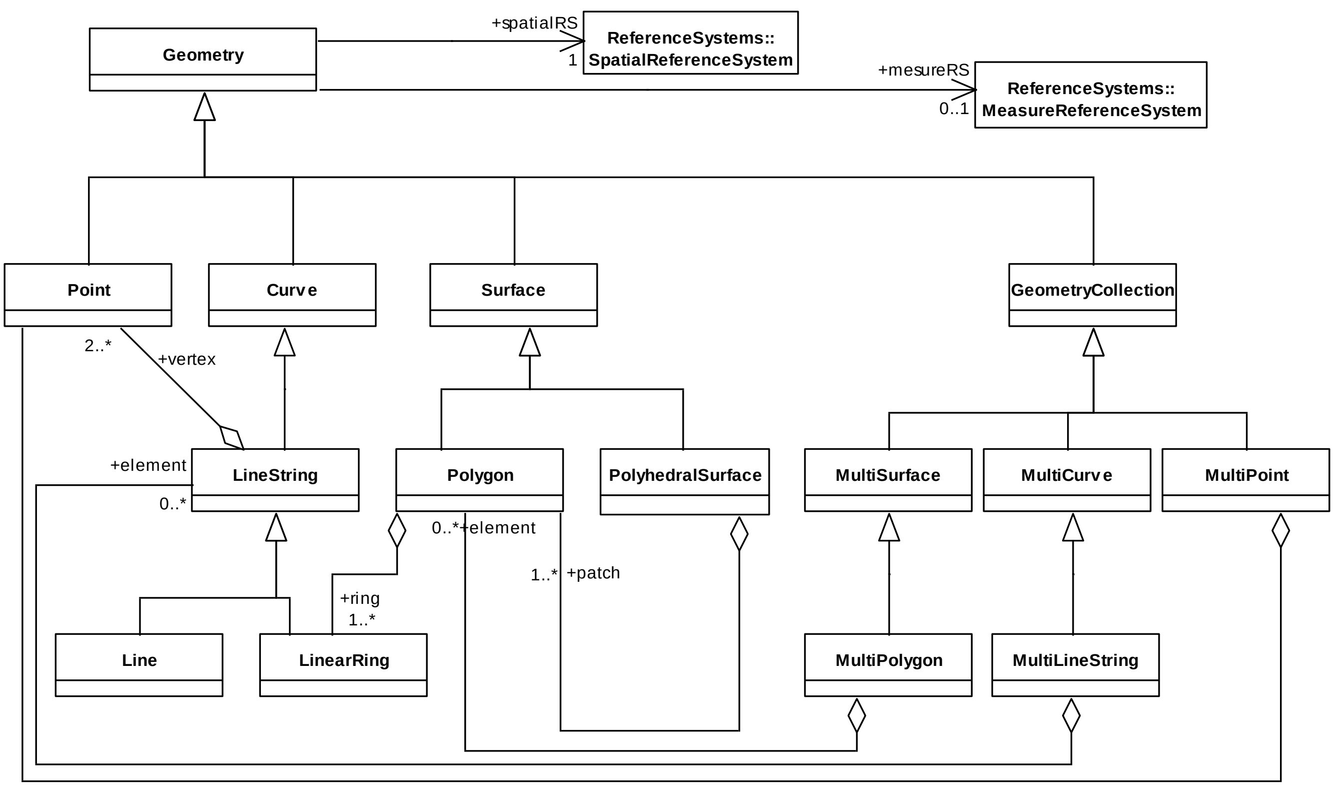


Figure 4: Figure: SQL Geometry Type hierarchy

The root type, named Geometry, has subtypes for Point, Curve, Surface and Geometry Collection. A Geometry Collection is a Geometry that is a collection of possibly heterogeneous geometric objects. MultiPoint, MultiCurve and MultiSurface are specific subtypes of Geometry Collection used to manage homogenous collections of Points, Curves and Surfaces. The 0 dimensional Geometry Types are Point and MultiPoint.

The one-dimensional Geometry Types are Curve and MultiCurve together with their subclasses. The two-dimensional Geometry Types are Surface and MultiSurface together with their subclasses.

SQL functions are defined to construct instances of the above Types given Well-known Text or Binary representations of the types. SQL functions defined on the types implement the methods described in the Geometry Object Model.

6.2.7 Geometry values and spatial reference systems

In order to model Spatial Reference System information, each geometric object in the SQL with Geometry Types implementation is associated with a Spatial Reference System as specified by SQL/MM.

In addition to the SQL/MM

6.2.8 Access to Geometry values in the SQL with Geometry Type case

Spatial data are accessed using the SQL query language extended with SQL routines to create Geometry Types as well as routines to observe or mutate their attributes, as specified by SQL/MM..

6.2.9 Text

6.2.9.1 Text Object Implementation

6.2.9.1.1 Text Objects

The text object, and their component elements which can be used either as a feature attribute or as a free-floating object, is defined in 7.2.20.

6.2.9.2 Metadata Table (View)

The metadata at a table level allows common information to be stored at a common level and not for each record. This keep the data for each record as compact as possible. There is no specific specification for this metadata table. But the data requirements in Table 3 must be available from the metadata store. This data if created as a table would look like this:

```
CREATE TABLE ANNOTATION_TEXT_METADATA AS
   {
   F_TABLE_CATALOG            AS CHARACTER VARYING NOT NULL,
   F_TABLE_SCHEMA             AS CHARACTER VARYING NOT NULL,
   F_TABLE_NAME               AS CHARACTER VARYING NOT NULL,
   F_TEXT_COLUMN              AS CHARACTER VARYING NOT NULL,

   A_TEXT_DEFAULT_MAP_BASE_SCALE  AS CHARACTER VARUONG,
   A_TEXT_DEFAULT_EXPRESSION  AS CHARACTER VARYING,
   A_TEXT_DEFAULT_ATTRIBUTES  AS CHARACTER VARYING
   }
```

The fields in the table above are described in shall be a view of database administration tables and must contain the following fields for each text column (column of a ANNOTATION_TEXT type):

Table 3: Text metadata attributes

FIELD	DEFINITION	COMMENT
F_TABLE_CATALOG F_TABLE_SCHEMA F_TABLE_NAME	Name of the table in which the text type values are stored.	Databases have format for this based on SQL:1999.
F_TEXT_COLUMN_NAME	Name of the column in which the text type value are stored.	Databases have format for this based on SQL:1999. This column in the feature table described above must be of type ANNOTATION_TEXT.
A_TEXT_DEFAULT_MAP_BASE_SCALE	The base map scale for which the text will be displayed	
A_TEXT_DEFAULT_EXPRESSION	This column allows the actual text of a text object to come from data outside the text object VALUE field.	Any valid database column expression resulting in a string is acceptable. The expression is evaluated for the each row. If this field is null, the individual text objects may have their own embedded text or nothing shall be displayed. Any embedded text shall override this expression value. During query to support display, client applications should add this expression to their select list so that any returned records will have the information needed to evaluate this expression without round tripping back to the database. . Note that this is the one case where the data critical to the display of text is stored outside the text object or metadata. It should be obvious to anyone changing the VALUE field that they are changing the text object. It may not be obvious to someone updating a column covered by the text expression that they are affecting the text object display.
A_TEXT_DEFAULT_ATTRIBUTES	As many text attributes may be common in one table, the database may store the common ones once here and allow for individual row (record) overrides.	The Text Style, Layout and Leader Line Style described below may be stored in the metadata as well as the individual rows. Any values in the individual rows shall override the metadata values. The resulting attributes are an overlay of the metadata attributes and individual row attribute values.

7 Clause component specifications

7.1 Components — Implementation of feature tables based on predefined data types

7.1.1 Conventions

Table components are described in the context of a CREATE TABLE statement. Implementations may use base tables with different names and properties, exposing these components as updateable views, provided that the base tables defined by the implementation enforce the same constraints.

Table names and column names have been restricted to 18 characters in length to allow for the widest possible implementation.

7.1.2 Spatial reference system information

7.1.2.1 Component overview

The Spatial Reference Systems table, which is named SPATIAL_REF_SYS, stores information on each spatial reference system used in the database.

7.1.2.2 Table constructs

The following CREATE TABLE statement creates an appropriately structured SPATIAL_REF_SYS table. This table may be an updatable view of an implementation-specific table. Implementations shall either use this table format or provide stored procedures to create, to populate and to maintain this table

```
CREATE TABLE SPATIAL_REF_SYS
    (
        SRID          INTEGER NOT NULL PRIMARY
                          KEY,
        AUTH_NAME     CHARACTER VARYING,
        AUTH_SRID     INTEGER,
        SRTEXT        CHARACTER VARYING(2048)
    )
```

7.1.2.3 Field description

These fields are described as follows:

a) SRID — an integer value that uniquely identifies each Spatial Reference System within a database;

b) AUTH_NAME — the name of the standard or standards body that is being cited for this reference system. EPSG would be an example of a valid AUTH_NAME;

c) `AUTH_SRID` — the ID of the Spatial Reference System as defined by the Authority cited in `AUTH_NAME`;

d) `SRTEXT` — The Well-known Text Representation of the Spatial Reference System.

7.1.2.4 Exceptions, errors and error codes

Error handling shall be accomplished by using the standard SQL status returns.

7.1.3 Geometry columns information

7.1.3.1 Component overview

The `GEOMETRY_COLUMNS` table provides information on the feature table, spatial reference, geometry type, and coordinate dimension for each Geometry column in the database. This table may be an updatable view of an implementation-specific table. Implementations shall either use this table format or provide stored procedures to create, to populate and to maintain this table

7.1.3.2 Table or view constructs

```
CREATE TABLE GEOMETRY_COLUMNS (
 F_TABLE_CATALOG    CHARACTER VARYING   NOT NULL,
 F_TABLE_SCHEMA     CHARACTER VARYING   NOT NULL,
 F_TABLE_NAME       CHARACTER VARYING   NOT NULL,
 F_GEOMETRY_COLUMN  CHARACTER VARYING   NOT NULL,
 G_TABLE_CATALOG    CHARACTER VARYING   NOT NULL,
 G_TABLE_SCHEMA     CHARACTER VARYING   NOT NULL,
 G_TABLE_NAME       CHARACTER VARYING   NOT NULL,
 STORAGE_TYPE       INTEGER,
 GEOMETRY_TYPE      INTEGER,
 COORD_DIMENSION    INTEGER,
 MAX_PPR            INTEGER,
 SRID               INTEGER             NOT NULL
                                             REFERENCES SPATIAL_REF_SYS,
 CONSTRAINT GC_PK PRIMARY KEY
      (F_TABLE_CATALOG, F_TABLE_SCHEMA, F_TABLE_NAME, F_GEOMETRY_COLUMN)
)
```

7.1.3.3 Field description

These fields are described as follows:

a) F_TABLE_CATALOG, F_TABLE_SCHEMA, F_TABLE_NAME — the fully qualified name of the feature table containing the geometry column.

b) F_GEOMETRY_COLUMN — the name of the column in the feature table that is the Geometry Column. This column shall contain a foreign key reference into the geometry table for an implementation based on predefined data types. For a geometry types implementation, this column may contain either a foreign key to a geometry extent table or a SQL UDT.

c) G_TABLE_CATALOG, G_TABLE_SCHEMA, G_TABLE_NAME — the name of the geometry table and its schema and catalog. The geometry table implements the geometry column. In a geometry types implementation that stores the geometry in the F_GEOMETRY_COLUMN, these columns will be identical to the F_TABLE_CATALOG, F_TABLE_SCHEMA, F_TABLE_NAME column values.

d) STORAGE_TYPE — the type of storage being used for this geometry column:

0 = normalized geometry implementation,

1 = binary geometry implementation (Well-known Binary Representation for Geometry).

NULL = geometry types implementation,

e) `GEOMETRY_TYPE` — the type of geometry values stored in this column. The use of a non-leaf Geometry class name from the Geometry Object Model for a geometry column implies that domain of the column corresponds to instances of the class and all of its subclasses. The suffixes "Z", "M" and "ZM" are three distinct copies of the geometry hierarchy as presented in Figure 4. If the value is NULL, then the appropriate GEOMETRY subtype is used consistent with the `COORD_DIMENSION` and `SRID` is implied. This code list is a subset of the list presented in Part 1, Table 7.

Table 4: Geometry type codes

Code	Geometry type	Coordinates
0	GEOMETRY	\\ IN X Y
1	POINT	\\ IN X Y
2	LINESTRING	\\ IN X Y
3	POLYGON	\\ IN X Y
4	MULTIPOINT	\\ IN X Y
5	MULTILINESTRING	\\ IN X Y
6	MULTIPOLYGON	\\ IN X Y
7	GEOMCOLLECTION	\\ IN X Y
13	CURVE	\\ IN X Y
14	SURFACE	\\ IN X Y
15	POLYHEDRALSURFACE	\\ IN X Y
1000	GEOMETRYZ	\\ IN X Y Z
1001	POINTZ	\\ IN X Y Z
1002	LINESTRINGZ	\\ IN X Y Z

Code	Geometry type	Coordinates
1003	POLYGONZ	\\ IN X Y Z
1004	MULTIPOINTZ	\\ IN X Y Z
1005	MULTILINESTRINGZ	\\ IN X Y Z
1006	MULTIPOLYGONZ	\\ IN X Y Z
1007	GEOMCOLLECTIONZ	\\ IN X Y Z
1013	CURVEZ	\\ IN X Y M
1014	SURFACEZ	\\ IN X Y M
1015	POLYHEDRALSURFACEZ	\\ IN X Y Z
2000	GEOMETRY	\\ IN X Y M
2001	POINTM	\\ IN X Y M
2002	LINESTRINGM	\\ IN X Y M
2003	POLYGONM	\\ IN X Y M
2004	MULTIPOINTM	\\ IN X Y M
2005	MULTILINESTRINGM	\\ IN X Y M
2006	MULTIPOLYGONM	\\ IN X Y M
2007	GEOMCOLLECTIONM	\\ IN X Y M
2013	CURVEM	\\ IN X Y M
2014	SURFACEM	\\ IN X Y M
2015	POLYHEDRALSURFACEM	\\ IN X Y M
3000	GEOMETRYZM	\\ IN X Y Z M
3001	POINTZM	\\ IN X Y Z M
3002	LINESTRINGZM	\\ IN X Y Z M

Code	Geometry type	Coordinates
3003	POLYGONZM	\\ IN X Y Z M
3004	MULTIPOINTZM	\\ IN X Y Z M
3005	MULTILINESTRINGZM	\\ IN X Y Z M
3006	MultiPolygonZM	\\ IN X Y Z M
3007	GEOMCOLLECTIONZM	\\ IN X Y Z M
3013	CURVEZM	\\ IN X Y Z M
3014	SURFACEZM	\\ IN X Y Z M
3015	POLYHEDRALSURFACEZM	\\ IN X Y Z M

f) COORD_DIMENSION — the number of ordinates used in the complex, usually corresponds to the number of dimensions in the spatial reference system. If an "M" ordinate is included it shall be one greater than the number of dimensions of the spatial reference system.

g) MAX_PPR — (This value contains data for the normalized geometry implementation only) Points per row, the number of Points stored as ordinate columns in the geometry table. This value may be NULL only if a binary storage or SQL geometry type implementation is used.

h) SRID — the ID of the Spatial Reference System used for the coordinate geometry in this table. It is a foreign key reference to the SPATIAL_REF_SYS table and must be specified.

7.1.3.4 Exceptions, errors and error codes

Error handling shall be accomplished by using the standard SQL status returns for SQL/CLI.

7.1.4 Feature tables

The columns in a feature table are defined by feature attributes; one or more of the feature attributes will be a geometric attribute. The basic restriction in this standard for feature tables is that for each geometric attribute, they include geometry via a FOREIGN KEY to a geometry table. Features may have a feature attribute that is unique, serving as a PRIMARY KEY for the feature table. Feature-to-feature relations may similarly be defined as FOREIGN KEY references where appropriate.

The general format of a feature table shall be as follows:

```
CREATE TABLE <feature table name>    (
     <primary key column name>  <primary key column type>,
     …  (other attributes for this feature table)
     <geometry column name>   <geometry column type>,
     …  (other geometry columns for this feature table)
     PRIMARY KEY <primary key column name>,
     FOREIGN KEY <geometry column name> REFERENCES <geometry table name>,
     …  (other geometry column constraints for this feature table)
     )
```

The geometric attribute foreign key reference applies only for the case where the geometry table stores geometry in binary form. In the case where geometry is stored in normalized form, there may be multiple rows in the geometry table corresponding to a single geometry value. In this case, the geometry attribute reference may be captured by a check constraint that ensures that the Geometry Column value in the feature table corresponds to the geometry-ID value for one or more rows in the geometry table.

7.1.5 Geometry tables

7.1.5.1 Component overview

Each Geometry table stores geometric objects corresponding to a Geometry column in a feature table. Geometric objects may be stored as individual ordinate values, using SQL numeric types, or as binary objects, using the Well-known Binary Representation for Geometry. Table schemas for both implementations are provided.

7.1.5.2 Geometry stored using SQL numeric types

7.1.5.2.1 Table constructs

The following `CREATE TABLE` statement creates an appropriately structured table for Geometry stored as individual ordinate values using SQL numeric types. Implementations shall either use this table format or provide stored procedures to create, to populate and to maintain this table.

```
CREATE TABLE <table name>
      (
      GID             INTEGER NOT NULL,
      ESEQ            INTEGER NOT NULL,
      ETYPE           INTEGER NOT NULL,
      SEQ             INTEGER NOT NULL,
      X1              <ordinate type>,
      Y1              <ordinate type>,
      Z1              <ordinate type>,
                           !Optional if Z-value is included
      M1              <ordinate type>,
                           !Optional if M-value is included
      ... <repeated for each ordinate, repeated for each point>
      X<MAX_PPR>      <ordinate type>,
      Y<MAX_PPR>      <ordinate type>,
      Z1<MAX_PPR>     <ordinate type>,
                           !Optional if Z-value is included
      M1<MAX_PPR>     <ordinate type>,
                           !Optional if M-value is included
      ...,
      <attribute>     <attribute type>
      CONSTRAINT GID_PK PRIMARY KEY (GID, ESEQ, SEQ)
      )
```

7.1.5.2.2 Field descriptions

These field descriptions are follows:

a. GID — identity of this geometric object;

b. ESEQ — identifies multiple components within a geometric object;

c. ETYPE — element type of this primitive element for the geometric object. The following values are defined for ETYPE:

- 1 = Point,
- 2 = LineString,
- 3 = Polygon;

d. SEQ — identifies the sequence of rows to define a geometric object;

e. X1 — first ordinate of first Point;

f. Y1 — second ordinate of first Point;

g. Z1 — third ordinate of first Point;

h. M1 — fourth ordinate of first Point;

i. ...— (repeated for each ordinate, for this Point);

j. ... — (repeated for each coordinate, for this row);

k. X<MAX_PPR> — first ordinate of last Point. The maximum number of Points per row 'MAX_PPR' is consistent with the information in the GEOMETRY_COLUMNS table;

l. Y<MAX_PPR> — second ordinate of last Point;

m. .Z<MAX_PPR> — third ordinate of first Point;

n. M<MAX_PPR> — fourth ordinate of first Point;

o. .. — (repeated for each ordinate, for this last Point);

p. <attribute> — other attributes can be carried in the Geometry table for specific feature schema.

7.1.5.2.3 Exceptions, errors and error codes

Error handling shall use the standard SQL status returns for SQL/CLI.

7.1.5.3 Geometry stored using SQL binary types

7.1.5.3.1 Table constructs

The following CREATE TABLE statement creates an appropriately defined table for Geometry stored using the Well-known Binary Representation for Geometry. The size of the WKB_GEOMETRY column is defined by the implementation. Implementations shall either use this table format or provide stored procedures to create, populate and maintain this table.

```
CREATE TABLE <table name>
     (
     GID          NUMERIC    NOT NULL   PRIMARY KEY,
     XMIN         <ordinate type>,
     YMIN         <ordinate type>,
     ZMIN         <ordinate type>,
     MMIN         <ordinate type>,
     XMAX         <ordinate type>,
     YMAX         <ordinate type>,
     ZMAX         <ordinate type>,
     MMAX         <ordinate type>,
     WKB_GEOMETRY  BIT VARYING(implementation size limit),
     {<attribute>  <attribute type>}*
     )
```

7.1.5.3.2 Field descriptions

These fields are described as follows:

a. GID — identity of this geometric object;

b. XMIN — the minimum x-coordinate of the geometric object bounding box;

c. YMIN — the minimum y-coordinate of the geometric object bounding box;

d. ZMIN — the maximum y-coordinate of the geometric object bounding box;

e. MMIN — the maximum y-coordinate of the geometric object bounding box;

f. XMAX — the maximum x-coordinate of the geometric object bounding box;

g. YMAX — the maximum y-coordinate of the geometric object bounding box;

h. ZMAX — the maximum y-coordinate of the geometric object bounding box;

i. MMAX — the maximum y-coordinate of the geometric object bounding box;

j. WKB_GEOMETRY — the Well-known Binary Representation of the geometric object;

k. <attribute> — other attributes can be carried in the Geometry table for specific feature schema.

7.1.5.3.3 Exceptions, errors and error codes

Error handling shall use the standard SQL status returns for SQL/CLI.

7.1.6 Operators

No SQL spatial operators are defined as part of this standard.

7.2 Components — SQL with Geometry Types implementation of feature tables

7.2.1 Conventions

The components of this standard for feature table implementation in a SQL with Geometry Types environment consist of the tables, SQL types and SQL functions discussed in 7.2 with routines as specified by SQL/MM.

7.2.2 SQL Geometry Types

7.2.2.1 Component overview

The SQL Geometry Types extend the set of available predefined data types to include Geometry Types.

7.2.2.2 Language constructs

A conforming implementation shall support a subset of the following set of SQL Geometry Types: {`Geometry`, `Point`, `Curve`, `LineString`, `Surface`, `Polygon`, `PolyhedralSurface` `GeomCollection`, `MultiCurve`, `MultiLineString`, `MultiSurface`, `MultiPolygon`, and `MultiPoint`}. The permissible type subsets that an implementer may choose to implement are described in SQL/MM.

Note: Class names in SQL/MM carry a "ST_" prefix. This is optional and implementations may chose to drop this prefix as has been done in various places in this standard.

The new type listed above is `PolyhedralSurface` shall be subtyped from `Surface`, and implements the required constructors, routines and interfaces of `Surface` and `MultiSurface`. To maintain a size limit on class names, the class name in SQL for `PolyhedralSurface` will be `PolyhedSurface`.

7.2.3 Feature tables

7.2.3.1 Component overview

The columns in a feature table are defined by feature attributes; one or more of the feature attributes will be a geometric attribute. The basic restriction in this standard for feature tables is that each geometric attribute is modeled using a column whose type corresponds to a SQL Geometry Type. Features may have a feature attribute that is unique, serving as a PRIMARY KEY for the feature table. Feature-to-feature relations may be defined as FOREIGN KEY references where appropriate.

7.2.3.2 Table constructs

The general format of a feature table in the SQL with Geometry Types implementation shall be as follows:

```
CREATE TABLE <feature table name>    (
      <primary key column name>   <primary key column type>,
      …  (other attributes for this feature table)
      <geometry column name>   <geometry type>,
      …  (other geometry columns for this feature table)
      PRIMARY KEY <primary key column name>,
      CONSTRAINT SRS_1 CHECK (SRID(<geometry column name>)
         in    (
               SELECT SRID from GEOMETRY_COLUMNS
                  where F_TABLE_CATALOG = <catalog> and
                     F_TABLE_SCHEMA = <schema> and
                     F_TABLE_NAME = <feature table name> and
                     F_GEOMETRY_COLUMN = <geometry column>
               )
      … ( spatial reference constraints for other geometry columns
          in this feature table)
         )
```

The use of any SQL Geometry Type for any of the columns in the table identifies this table as a feature table. Alternatively, applications may check the GEOMETRY_COLUMNS table, where all Geometry Columns and their associated feature tables and geometry tables are listed.

7.2.3.3 Exceptions, errors and error codes

Error handling shall be accomplished by using the standard SQL status returns.

7.2.4 SQL routines for constructing a geometry object given its Well-known Text Representation

The routines ST_WKTToSQL used to construct geometric objects from their text representations are specified by SQL/MM..

7.2.5 SQL routines for constructing a geometric object given its Well-known Binary Representation

The routines ST_WKBToSQL used to construct geometric objects from their Well-known Binary Representations are specified in SQL/MM.

7.2.6 SQL routines for obtaining Well-known Text Representation of a geometric object

The SQL routines ST_AsText for obtaining the Well-known Text Representation of a geometric object are specified in SQL/MM.

7.2.7 SQL routines for obtaining Well-known Binary Representations of a geometric object

The SQL routines ST_AsBinary for obtaining the Well-known Binary Representation of a geometric object are specified in SQL/MM.

7.2.8 SQL routines on type Geometry

7.2.8.1 Supported routines

The SQL/MM `ST_Dimension, ST_GeometryType, ST_AsText, ST_AsBinary, ST_SRID, ST_IsEmpty, ST_IsSimple, ST_Boundary`, and `ST_Envelope` routines shall be supported for all Geometry Types. Also included are SQL routines for obtaining the Well-known Binary and Text Representation of a geometric object and creating values from them.

Consistent with the definitions of relations in Part 1, Clause 6.1.2.3, the SQL/MM `ST_Equals, ST_Disjoint, ST_Intersects, ST_Touches, ST_Crosses, ST_Within, ST_Contains, ST_Overlaps` and `ST_Relate` routines shall be supported to test named spatial relationships between two geometric objects.

The SQL/MM `ST_Distance` routines shall be supported to calculate the distance between two geometric objects.

Consistent with the set theoretic operations defined in ISO 19103, and ISO 19107, the SQL/MM `ST_Intersection, ST_Difference, ST_Union, ST_SymDifference, ST_Buffer,` and `ST_ConvexHull` routines shall be supported to implement set-theoretic and constructive operations on geometric objects. These operations are defined for all types of Geometry.

7.2.8.2 Declarations from SQL/MM (informative)

```
CREATE TYPE ST_Geometry
AS (
   ST_PrivateDimension              SMALLINT   DEFAULT    -1,
   ST_PrivateCoordinateDimension    SMALLINT   DEFAULT    2,
   ST_PrivateIs3D                   SMALLINT   DEFAULT    0,
   ST_PrivateIsMeasured             SMALLINT   DEFAULT    0
   )
NOT INSTANTIABLE
NOT FINAL
```



```
METHOD ST_Dimension()
      RETURNS SMALLINT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,


METHOD ST_GeometryType()
   RETURNS CHARACTER VARYING(ST_MaxTypeNameLength)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_AsText()
   RETURNS CHARACTER LARGE OBJECT(ST_MaxGeometryAsText)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   RETURNS NULL ON NULL INPUT,


METHOD ST_AsBinary()
   RETURNS BINARY LARGE OBJECT(ST_MaxGeometryAsBinary)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_SRID()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_SRID (ansrid INTEGER)
   RETURNS ST_Geometry
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,


METHOD ST_IsEmpty()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_IsSimple()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Boundary()
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Envelope()
   RETURNS ST_Polygon
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_WKTToSQL (awkt CHARACTER LARGE OBJECT(ST_MaxGeometryAsText))
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_WKBToSQL(awkb BINARY LARGE OBJECT(ST_MaxGeometryAsBinary))
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Equals(ageometry ST_Geometry)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Disjoint(ageometry ST_Geometry)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Intersects (ageometry ST_Geometry)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Touches(ageometry ST_Geometry)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_Crosses(ageometry ST_Geometry)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Within (ageometry ST_Geometry)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Contains(ageometry ST_Geometry)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Overlaps(ageometry ST_Geometry)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Relate(ageometry ST_Geometry, amatrix CHARACTER(9))
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```



```
METHOD ST_Distance(ageometry ST_Geometry)
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Distance(ageometry ST_Geometry,
                   aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Intersection(ageometry ST_Geometry)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Difference(ageometry ST_Geometry)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Union(ageometry ST_Geometry)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_SymDifference (ageometry ST_Geometry)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Buffer (adistance DOUBLE PRECISION)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Buffer ( adistance DOUBLE PRECISION,
                   aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_ConvexHull()
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

7.2.9 SQL routines on type Point

7.2.9.1 Supported routines

The SQL/MM `ST_X`, `ST_Y`, `ST_Z` and `ST_M` routines and all routines supported by type `Geometry` shall be supported for geometries of type Point.

7.2.9.2 Declarations from SQL/MM (informative)

```
CREATE TYPE ST_Point
UNDER ST_Geometry AS
      (
      ST_PrivateX   DOUBLE PRECISION   DEFAULT NULL,
      ST_PrivateY   DOUBLE PRECISION   DEFAULT NULL,
      ST_PrivateZ   DOUBLE PRECISION   DEFAULT NULL,
      ST_PrivateM   DOUBLE PRECISION   DEFAULT NULL
      )
INSTANTIABLE
NOT FINAL


METHOD ST_X()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_X (xcoord DOUBLE PRECISION)
   RETURNS ST_Point
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,


METHOD ST_Y()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_Y (ycoord DOUBLE PRECISION)
   RETURNS ST_Point
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,


METHOD ST_Z()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Z (zcoord DOUBLE PRECISION)
   RETURNS ST_Point
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,


METHOD ST_M()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```



```
METHOD ST_M (mcoord DOUBLE PRECISION)
   RETURNS ST_Point
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
```

7.2.10 SQL routines on type Curve

7.2.10.1 Supported routines

The SQL/MM `ST_StartPoint`, `ST_EndPoint`, `ST_IsRing` and `ST_Length` routines and all routines supported by type `Geometry` shall be supported for geometries of type Curve.

7.2.10.2 Declarations from SQL/MM (informative)

```
CREATE TYPE ST_Curve
   UNDER ST_Geometry
   NOT INSTANTIABLE
   NOT FINAL


METHOD ST_StartPoint()
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_EndPoint()
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_IsRing()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Length()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Length (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

7.2.11 SQL routines on type LineString

7.2.11.1 Supported routines

The SQL/MM `ST_NumPoints` and `ST_PointN` routines and all routines supported by type `Curve` shall be supported for geometries of type LineString.

7.2.11.2 Routing declarations from SQL/MM (informative)

```
CREATE TYPE ST_LineString
UNDER ST_Curve
AS (
   ST_PrivatePoints
         ST_Point ARRAY[ST_MaxGeometryArrayElements] DEFAULT ARRAY[]
   )
INSTANTIABLE
NOT FINAL
```

```
METHOD ST_NumPoints()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_PointN(aposition INTEGER)
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

7.2.12 SQL functions on type Surface

7.2.12.1 Supported routines

The SQL/MM `ST_Centroid`, `ST_PointOnSurface` and `ST_Area` routines and all routines supported by type `Geometry` shall be supported for geometries of type `Surface`.

7.2.12.2 Declarations from SQL/MM (informative)

```
CREATE TYPE ST_Surface
   UNDER ST_Geometry
   NOT INSTANTIABLE
   NOT FINAL


METHOD ST_Area()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_Area (aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Centroid ()
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT


METHOD ST_PointOnSurface()
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

7.2.13 SQL functions on type Polygon

7.2.13.1 Supported routines

The SQL/MM `ST_ExteriorRing`, `ST_NumInteriorRing`, and `ST_InteriorRingN` routines and all routines supported by type `Geometry` shall be supported for geometries of type `Polygon`.

7.2.13.2 Declarations from SQL/MM (informative)

```
CREATE TYPE ST_Polygon
   UNDER ST_CurvePolygon
   INSTANTIABLE
   NOT FINAL


METHOD ST_ExteriorRing()
   RETURNS ST_LineString,
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_ExteriorRing (acurve ST_LineString)
   RETURNS ST_Polygon,
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,


METHOD ST_InteriorRings()
RETURNS ST_LineString ARRAY[ST_MaxGeometryArrayElements]
LANGUAGE SQL
DETERMINISTIC
CONTAINS SQL
RETURNS NULL ON NULL INPUT,


METHOD ST_InteriorRings (acurvearray ST_LineString
      ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_Polygon
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,


METHOD ST_NumInteriorRing()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_InteriorRingN(aposition INTEGER)
   RETURNS ST_LineString
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

7.2.14 SQL functions on type Polyhedral Surface

7.2.14.1 Supported routines

The routines supported by type `Geometry`, `Surface` and `MultiPolygon` shall be supported for geometries of type Polyhedral Surface, `PolyhedSurface`. In the SQL below, the "max<thing>size" parameters are local implementation specific maximum sizes for the things so specified. Attributes of types names as "private" may be implemented in any manner as long as the semantics of the functions is consistent. When integrating this SQL with that of SQL/MM, the type-name prefix "ST_" should be used as appropriate.

7.2.14.2 Declarations proposed to be added to SQL/MM

```
CREATE TYPE PolyhedSurface
      UNDER Surface
      AS (
           PrivatePatches Surface ARRAY[MaxArraySize] DEFAULT ARRAY[]
         )
      INSTANTIABLE
      NOT FINAL


CONSTRUCTOR METHOD PolyhedSurface
         (  awktorgml CHARACTER LARGE OBJECT(MaxTextSize))
      RETURNS ST_MultiSurface
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD PolyhedSurface
        (  awktorgml CHARACTER LARGE OBJECT(MaxTextSize),
           srsid INTEGER)
     RETURNS ST_MultiSurface
     SELF AS RESULT
     LANGUAGE SQL
     DETERMINISTIC
     CONTAINS SQL
     RETURNS NULL ON NULL INPUT,


CONSTRUCTOR METHOD PolyhedSurface
        (  awkb BINARY LARGE OBJECT(MaxBinarySize))
     RETURNS ST_MultiSurface
     SELF AS RESULT
     LANGUAGE SQL
     DETERMINISTIC
     CONTAINS SQL
     RETURNS NULL ON NULL INPUT,


CONSTRUCTOR METHOD PolyhedSurface
        (  awkb BINARY LARGE OBJECT(MaxBinarySize),
           srsid INTEGER)
     RETURNS PolyhedSurface
     SELF AS RESULT
     LANGUAGE SQL
     DETERMINISTIC
     CONTAINS SQL
     RETURNS NULL ON NULL INPUT,


CONSTRUCTOR METHOD PolyhedSurface
        (  asurfacearray Surface ARRAY[MaxArraySize])
     RETURNS PolyhedSurface
     SELF AS RESULT
     LANGUAGE SQL
     DETERMINISTIC
     CONTAINS SQL
     RETURNS NULL ON NULL INPUT,
```

```
CONSTRUCTOR METHOD PolyhedSurface
        (  asurfacearray Surface ARRAY[MaxArraySize]
           srsid INTEGER)
     RETURNS PolyhedSurface
     SELF AS RESULT
     LANGUAGE SQL
     DETERMINISTIC
     CONTAINS SQL
     RETURNS NULL ON NULL INPUT,


METHOD ST_Geometries()
     RETURNS Surface ARRAY[MaxArraySize],
     LANGUAGE SQL
     DETERMINISTIC
     CONTAINS SQL
     RETURNS NULL ON NULL INPUT,


METHOD NumSurfaces()
     RETURNS INTEGER
     LANGUAGE SQL
     DETERMINISTIC
     CONTAINS SQL
     RETURNS NULL ON NULL INPUT,


METHOD SURFACE (aposition INTEGER)
     RETURNS Surface
     LANGUAGE SQL
     DETERMINISTIC
     CONTAINS SQL
     RETURNS NULL ON NULL INPUT
```

7.2.15 SQL routines on type GeomCollection

The SQL/MM `ST_NumGeometries` and `ST_GeometryN` routines shall be supported for geometries of type `GeomCollection`.

```
CREATE TYPE ST_GeomCollection
  UNDER ST_Geometry
  AS (
     ST_PrivateGeometries ST_Geometry
     ARRAY[ST_MaxGeometryArrayElements] DEFAULT ARRAY[]
     )
  INSTANTIABLE
  NOT FINAL
```

```
METHOD ST_NumGeometries()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_GeometryN (aposition INTEGER)
   RETURNS ST_Geometry
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
```

7.2.16 SQL routines on type MultiPoint

7.2.16.1 Supported routines

The SQL/MM routines supported by GeomCollection shall be supported for geometries of type MultiPoint.

7.2.16.2 Declarations from SQL/MM (informative)

```
CREATE TYPE ST_MultiPoint
   UNDER ST_GeomCollection
   INSTANTIABLE
   NOT FINAL
```

7.2.17 SQL routines on type MultiCurve

7.2.17.1 Supported routines

The SQL/MM `ST_IsClosed` and `ST_Length` routines and all routines supported by GeomCollection shall be supported for geometries of type MultiCurve.

7.2.17.2 Declarations from SQL/MM (informative)

```
CREATE TYPE ST_MultiCurve
   UNDER ST_GeomCollection
   INSTANTIABLE
   NOT FINAL


METHOD ST_IsClosed()
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Length()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Length(aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

7.2.18 SQL routines on type MultiLineString

7.2.18.1 Supported routines

The SQL/MM routines supported by `GeomCollection` shall be supported for geometries of type MultiLineString.

7.2.18.2 Declarations from SQL/MM (informative)

```
CREATE TYPE ST_MultiLineString
   UNDER ST_MultiCurve
   INSTANTIABLE
   NOT FINAL


OVERRIDING METHOD ST_Geometries()
   RETURNS ST_LineString ARRAY[ST_MaxGeometryArrayElements],


OVERRIDING METHOD ST_Geometries(ageometryarray ST_Geometry
   ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_MultiLineString
```

7.2.19 SQL routines on type MultiSurface

7.2.19.1 Supported routines

The SQL/MM `ST_Centroid`, `ST_PointOnSurface`, and `ST_Area` routines and the routines supported by `GeomCollection` shall be supported for geometries of type MultiSurface.

7.2.19.2 Declarations from SQL/MM (informative)

```
CREATE TYPE ST_MultiSurface
   UNDER ST_GeomCollection
   INSTANTIABLE
   NOT FINAL


METHOD ST_Centroid()
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,
```

```
METHOD ST_PointOnSurface()
   RETURNS ST_Point
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Area()
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


METHOD ST_Area(aunit CHARACTER VARYING(ST_MaxUnitNameLength))
   RETURNS DOUBLE PRECISION
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT,


OVERRIDING METHOD ST_Geometries()
   RETURNS ST_Surface ARRAY[ST_MaxGeometryArrayElements],
   OVERRIDING METHOD ST_Geometries
   (ageometryarray ST_Geometry
   ARRAY[ST_MaxGeometryArrayElements])
   RETURNS ST_MultiSurface
```

7.2.20 SQL routines on type Text

The `Annotation_Text`, `Annotation_Text_Element`, and `Annotation_Text_Element_Array` provide text functionality as SQL objects.

```
CREATE TYPE ANNOTATION_TEXT AS
      {
      PrivateEnvelope       AS GEOMETRY,
      PrivateElement_Array  AS ANNOTATION_TEXT_ELEMENT_ARRAY
      }
```



```
CONSTRUCTOR METHOD ANNOTATION_TEXT(anArray ANNOTATION_TEXT_ELEMENT_ARRAY)
     RETURNS ANNOTATION_TEXT
     SELF AS RESULT
     LANGUAGE SQL
     DETERMINISTIC
     CONTAINS SQL
     RETURNS NULL ON NULL INPUT,


METHOD CONCAT(b ANNOTATION_TEXT)
     RETURNS ANNOTATION_TEXT
     SELF AS RESULT
     LANGUAGE SQL
     DETERMINISTIC
     CONTAINS SQL
     RETURNS NULL ON NULL INPUT,


METHOD ENVELOPE ()
     RETURNS GEOMETRY
     LANGUAGE SQL
     DETERMINISTIC
     CONTAINS SQL
     RETURNS NULL ON NULL INPUT


METHOD ELEMENT_ARRAY ()
     RETURNS ANNOTATION_TEXT_ELEMENT_ARRAY
     LANGUAGE SQL
     DETERMINISTIC
     CONTAINS SQL
     RETURNS NULL ON NULL INPUT


CREATE TYPE ANNOTATION_TEXT_ELEMENT_ARRAY AS
     VARING ARRAY (MaxArraySize) OF ANNOTATION_TEXT_ELEMENT,
```

```
METHOD ElementN (aposition INTEGER)
      RETURNS ANNOTATION_TEXT_ELEMENT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT


METHOD ElementN (element ANNOTATION_TEXT_ELEMENT
                aposition INTEGER)
      RETURNS ANNOTATION_TEXT_ELEMENT_ARRAY
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT


CREATE TYPE ANNOTATION_TEXT_ELEMENT AS
      (
      privateValue          AS CHARACTER VARYING (MaxArraySize),
      privateLocation       AS GEOMETRY,
      privateLeaderLine     AS GEOMETRY,
      privateTextAttributes AS CHARACTER VARYING (MaxArraySize)
      )


CONSTRUCTOR METHOD AnnotationTextElement
      (  value            CHARACTER VARYING (MaxArraySize),
         location         GEOMETRY,
         leaderLine       GEOMETRY,
         textAttributes   CHARACTER VARYING (MaxArraySize))
      RETURNS ANNOTATION_TEXT_ELEMENT
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,


METHOD Value ()
      RETURNS CHARACTER VARYING (MaxArraySize)
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```



```
METHOD Value (value RETURNS ANNOTATION_TEXT_ELEMENT
      RETURNS ANNOTATION_TEXT_ELEMENT
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,


METHOD TextAttributes ()
      RETURNS CHARACTER VARYING (MaxArraySize)
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,


METHOD TextAttributes (attributes    CHARACTER VARYING (MaxArraySize))
      RETURNS ANNOTATION_TEXT_ELEMENT
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,


METHOD Location ()
      RETURNS GEOMETRY
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,


METHOD Location (location     GEOMETRY)
      RETURNS ANNOTATION_TEXT_ELEMENT
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,
```

```
METHOD LeaderLine ()
     RETURNS GEOMETRY
     LANGUAGE SQL
     DETERMINISTIC
     CONTAINS SQL
     RETURNS NULL ON NULL INPUT,


METHOD LeaderLine (leaderLine    GEOMETRY)
     RETURNS ANNOTATION_TEXT_ELEMENT
     SELF AS RESULT
     LANGUAGE SQL
     DETERMINISTIC
     CONTAINS SQL
     RETURNS NULL ON NULL INPUT
```


Annex A
(normative)
Abstract Test Suite

A.1 Purpose of this annex

This annex outlines the requirements for a comprehensive test suite for each class of compliance for this standard. Each conformance clause defined in Section A.2 will address testing methods for a coherent set of requirements from the normative Clauses in this standard or other standards. Each compliance level or class, defined in Section A.4 below, will address a specified set of conformance clauses.

Some of the conformance clauses are "parameterize" in the sense that they specify use of "appropriate" test from another clause. This is done to keep the number of clauses to a minimum while allowing for a finer degree of separation between conformance classes. Each time a parameterized conformance clause is used in defining an conformance class, it parameter must be specified.

A.2 Conformance Tests

A.2.1 Feature tables

Test Purpose: To test the capability to create, access, query and modify feature tables (Section 7.1.4 or 7.2.3) and using the appropriate geometric types, as defined in the associated geometry conformance clause.

Test Method: Each test will consist of:

a) Reading a feature schema from a set of SQL statements

b) Loading feature and geometry tables from a set of text load files containing SQL statements, or file of similar content as defined for the SQL version being used.

c) Making attribute and spatial queries against the table so loaded above

d) Getting an acceptable answer as tested by an export of the query results defined above.

A.2.1.1 Features using geometry in predefined types

Use the feature implementation defined in 7.1.4.

A.2.1.2 Features using Binary or SQL geometry types

Use the feature implementation defined in 7.2.3.

A.2.2 Geometry tables or type

A.2.2.1 Normalized geometry schema

Test Purpose: To test the capability to create, access, query and modify feature spatial attributes using the appropriate geometric implementation as described in Clauses 6.1.5.1 Normalized geometry schema, 7.1.5.2 Geometry stored using SQL numeric types with metadata as in 7.1.5,Geometry columns information.

Test Method: Each test will consist of:

a) Incorporating the appropriate geometric types in the feature table test of A.2.1

A.2.2.2 Binary geometry

Test Purpose: To test the capability to create, access, query and modify feature spatial attributes using the appropriate geometric types, Section 6.1.5.2 Binary geometry schema, 7.1.5.3 Geometry stored using SQL binary types with metadata as in 7.1.3,Geometry columns information.

Test Method: Each test will consist of:

a) Incorporating the appropriate geometric types in the feature table test of A.2.1

A.2.2.3 SQL/MM geometry schema

Test Purpose: To test the capability to create, access, query and modify feature spatial attributes using the appropriate geometric types, Section 6.1.5.3 SQL/MM geometry schema, 7.2 Components — SQL with Geometry Types implementation of feature tables, with metadata as in 7.1.3,Geometry columns information.

Test Method: Each test will consist of:

a) Incorporating the appropriate geometric types in the feature table test of A.2.1

A.2.3 Spatial reference systems

A.2.3.1 2D Spatial reference systems

Test Purpose: To test the capability of creating, and using 2D coordinate systems, coordinates in X and Y.

Test Method: Each test will consist of:

a) Defining a 2D coordinate systems compatible with a test feature and geometry test as defined in A.2.1, and A.2.1.1, for geometries compatible with a 2D coordinate system

b) Execute the test as defined, and obtain appropriate query results.

A.2.3.2 3D Spatial reference systems

Test Purpose: To test the capability of creating, and using 3D coordinate systems, coordinates in X, Y and Z. This includes the capability to create both 2D and 3D coordinate systems and to use them to describe geometry values.

Test Method: Each test will consist of:

a) All tests in A.2.3.1

b) Defining a 3D coordinate systems compatible with a test feature and geometry test as defined in A.2.1, and A.2.1.1, for geometries compatible with a 3D coordinate system

c) Execute the test as defined, and obtain appropriate query results.

Note: Spatial reference systems must still be defined on a column basis, and a feature table shall not mix geometry values from different spatial reference systems within a single attribute column.

A.2.3.3 Measured Spatial reference systems

Test Purpose: To test the capability of creating, and using Measured coordinate systems coordinates having an M. This includes the ability to create geometry values both with and without measured coordinates.

Test Method: Each test will consist of:

a) Defining a measured coordinate systems compatible with a test feature and geometry test as defined in A.2.1, and A.2.1.1, for geometries compatible with a measured coordinate system

b) Execute the test as defined, and obtain appropriate query results.

Note: Spatial reference systems must still be defined on a column basis, and a feature table shall not mix geometry values from different spatial reference systems within a single attribute column.

A.2.4 Geometric format supported

Test Purpose: To test the capability of creating and using geometric values in a particular representation format from one of the following Clauses.

A.2.4.1 Geometry stored using SQL numeric types

Perform the test using Section 7.1.5.2 Geometry stored using SQL numeric types (Table)

A.2.4.2 Geometry stored using SQL binary types

Perform the test using Section 7.1.5.3 Geometry stored using SQL binary types (Binary Type)

A.2.4.3 SQL Geometry Types

Perform the test using Section 7.2.2 SQL Geometry Types (SQL Type)

A.2.5 Geometric categories supported

Test Purpose: To test the capability of creating and using geometric types as defined in the subclauses below

Test Method: Each test will consist of

a) Perform a test from Conformance Clause A.2 using appropriate geometry types.

b) Creating and using geometry types including those defined in this Section according to the types defined in the appropriate section as listed below.

A.2.5.1 Basic Geometric categories supported

Perform the test with types in Part 1 Section 6.1.3 through 6.1.15, except 6.1.12

A.2.5.2 Tins and Basic Geometric categories supported

Perform the test with types the basic test and with the addition of TINs for 6.1.12.

A.2.5.3 Full Geometric categories supported

Perform the test with types in Part 1 Section 6.1.3 through 6.1.15.

A.2.6 Text

Test purpose: To test the capability of creating and using annotations of the appropriate types from one of the following Clauses.

a) Section 6.2.9 (using predefined types – a table implementation)

b) Section 7.2.20 (using SQL UDT types)

Note: No binary implementation of annotations has been specified.

A.2.6.1 Text using predefined types supported

Perform the test with annotation text as defined in Section 6.2.9 (using predefined types – a table implementation)

A.2.6.2 Text using SQL UDT types supported

Perform the test with annotation text as defined in Section 7.2.20 (using SQL UDT types)

A.3 Composite Conformance Clauses

A.4 Conformance Classes

A.4.1 Types of conformance classes

All conformant applications (SQL data servers) must support features (one of the tests in A.2.1), but may support the other aspects of this standard dependent on a set of five choices. Conformance class choices are base on the following parameters:

a) Format of geometry supported —

gT (table using predefined types) (not valid with M, 3D, or Text S)) A.2.4.1 and A.2.1.1

gB (binary type) (tests A.2.4.2 and A.2.1.2) or

gS (SQL type) (tests A.2.4.3 and A.2.1.2)

b) Types of geometry supported —

b - Basic (no polyhedral surfaces) A.2.5.1,

t - Basic plus TINS (must be 3D) A.2.5.2 or

f - Full (must be 3D) A.2.5.3

c) Dimension of coordinate systems supported —

2D (two-dimensional) A.2.3.1 or

3D (3-dimensional) includes 2D (test A.2.3.2) (only valid with geometry choices gB or gS)

d) Measured or unmeasured Coordinate system —

M (measured) (only valid with geometry B or S) (test A.2.3.3) or

N (not measured) (no additional test)

e) Types of annotation text supported —

tT - table using predefined types) (test A.2.6.1) (valid only with geometry gB) (no additional test) or

tS - SQL type (only valid with geometry gS) (test A.2.6.2) or

tN - no text support (no additional tests), included for compatibility of SFA v1.1 (earlier) versions

This means that a conformance class may be defined by a string of 5 characters from the list above in order sbject to the restrictions listed.

For example, the maximum compliance level for SQL types is (gS, f, 3D, M, tS). The minimal compliance level for v1.1, table geometry is (gT, b, 2D, N, tN). The other equivalences between V1.1 conformance classes () and those in this version are given in Table A 1.

Table A 1 - Equivalences between V1.1 and V1.2 complinace classes

V1.1 Conformance Class	Equivalent V1.2 Conformance Class
Normalized geometry schema	(gT, b, 2D, N, tN)
Binary geometry schema	(gB, b, 2D, N, tN)
Geometry types and functions	(gS, b, 2D, N, tN)

Annex B
(informative)

Comparison of Simple feature access/SQL and SQL/MM – Spatial

This informative annex provides a comparison of SFA-SQL and SQL/MM — Spatial.

Table B 1 — Comparison of SFA-SQL and SQL/MM: Spatial

	SQL with geometry type	ISO/IEC 13249-3:2003 (SQL/MM-Spatial)	Description
Geometry Types	Point Curve Linestring Surface Polygon PolyhedralSurface GeomCollection Multipoint Multicurve Multilinestring Multisurface Multipolygon	ST_Point ST_Curve ST_Linestring ST_Circularstring ST_CompoundCurve ST_Surface ST_CurvePolygon ST_Polygon ST_PolyhedralSurface ST_Collection ST_Multipoint ST_MultiCurve ST_Multilinestring ST_Multisurface ST_Multipolygon	The type ST_PolyhedralSurface is currently not in SQL/MM but will be proposed as a result of this document.
Storage	Binary Type, Text Type, Object Type	Object Type	—
Operations	Equals Disjoint Touches Within Overlaps Crosses Intersects Contains Relate	ST_Equals ST_Disjoint ST_Touches ST_Within ST_Overlaps ST_Crosses ST_Intersects ST_Contains ST_Relate	—
Functions:	—	—	—
Point	 X() Y() Z() M() —	ST_Point() ST_X() ST_Y() ST_Z() ST_M() ST_ExplicitPoint()	Return the Point Return the *X*-coordinate of point Return the *Y*-coordinate of point Return the *Z*-coordinate of point Return the *M*-coordinate of point —

	SQL with geometry type	ISO/IEC 13249-3:2003 (SQL/MM-Spatial)	Description

Curve	`Length()`
	`StartPoint()`
	`EndPoint()`
	`IsClosed()`
	`IsRing()`
	—
		`ST_Length()`
		`ST_StartPoint()`
		`ST_EndPoint()`
		`ST_IsClosed()`
		`ST_ISRing()`
		`ST_CurveToLine`
			Return the length of curve
			Return the first Point of curve
			Return the last Point of curve
			Check whether curve is closed
			Check whether curve is closed and simple
			Transform Curve to LineString

LineString	—
	—
	`NumPoints()`
	`PointN()`
		`ST_LineString`
		`ST_Points`
		`ST_NumPoints`
		`ST_PointN`
			Return the LineString
			Return a collection of points
			Return the number of points
			Return a Point containing Point n of LineString

Annex C
(informative)

Conformance tests from version 1.1

C.1 Purpose of this annex

This conformance test is for an earlier 2D version of this standard, and has been replaced by an Abstract test suite that will be used to define a more complete set of conformance tests for the various options in this version of the standard.

In order to conform to this standard for feature collections, an implementation shall satisfy the requirements of one of the following three conformance classes:

a) SQL implementation of feature tables based on predefined data types:

   a. using numeric SQL types for geometry storage and SQL/CLI access,

   b. using binary SQL types for geometry storage and SQL/CLI access;

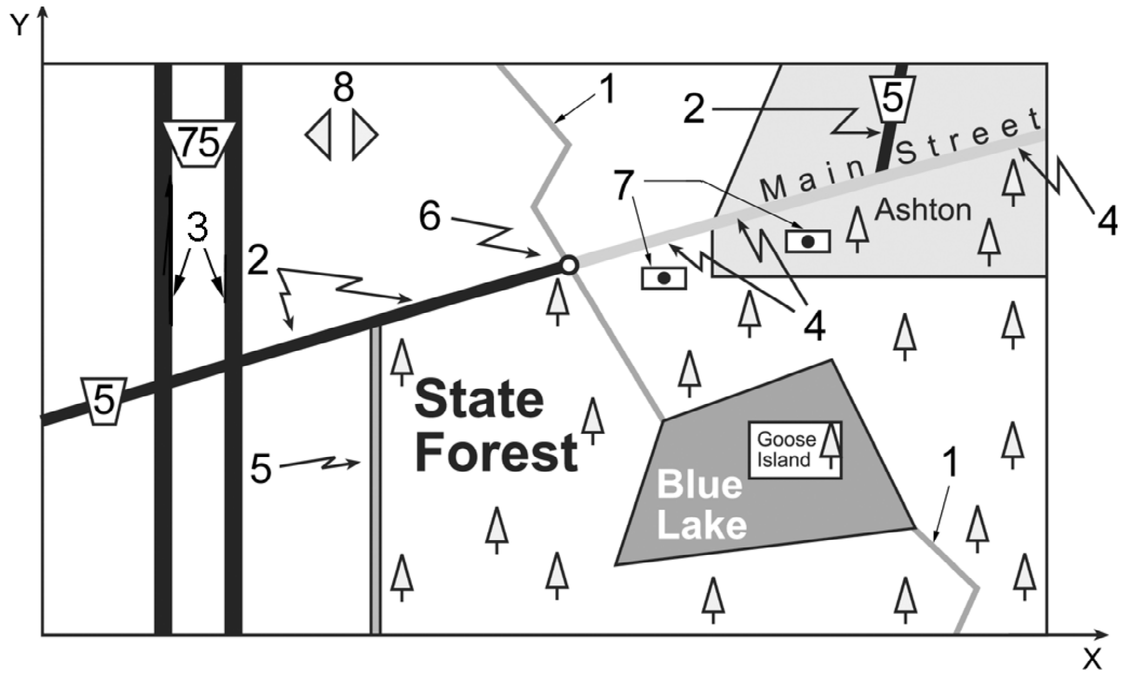
a. SQL with Geometry Types implementation of feature tables supporting both textual and binary SQL/CLI access to geometry.

This annex provides a conformance test for this standard. In general, the scope of the tests is to exercise each functional aspect of the standard at least once. The test questions and answers are defined to test that the specified functionality exists and is operable. Care has been taken to ensure that the tests are not at the level of rigor that a product quality-control process or certification test might be. However, some of the answers are further examined for reasonableness (for example, the area of a polygon is tested for correctness to two or three significant figures). The following sections further describe each test alternative.

C.2 Test data

C.2.1 Test data semantics

The data for all of the test alternatives are the same. It is a synthetic data set, developed by hand, to exercise the functionality of the standard. It is a set of features that makes up a map (see Figure B.1) of a fictional location called Blue Lake. This section describes the test data in detail.



Key: X Easting Y Northing

1	watercourse	
2	Route 5	
	▬	indicates where Route 5 is two lanes wide; indicates where Route 5 is four lanes wide
3	Route 75	
4	Main Street	
5	one-lane road	
6	○	bridge
7	▭	buildings
8	◁ ▷	fish ponds

Figure C 1: Test Data Concept — Blue Lake vicinity map

The semantics of this data set are as follows.

a) A rectangle of the Earth is shown in UTM coordinates. Horizontal coordinates take meaning from POSC Horizontal Coordinate System #32214. Note 500,000 m false Easting, and WGS 72 / UTM zone 14N. Units are metres.

b) Blue Lake (which has an island named Goose Island) is the prominent feature.

c) There is a watercourse flowing from north to south. The portion from the top neatline to the lake is called Cam Stream. The portion from the lake to the bottom neatline has no name (Name value is “Null”).

d) There is an area place named Ashton.

e) There is a State Forest whose administrative area includes the lake and a portion of Ashton. Roads form the boundary of the State Forest. The “Green Forest” is the State Forest minus the lake.

f) Route 5 extends across the map. It is two lanes wide where shown as a heavy black line. It is four lanes wide where shown as a heavy grey line.

g) There is a major divided highway, Route 75, shown as a heavy double black line, one line for each part of the divided highway. These two lines are seen as a multiline.

h) There is a bridge (Cam Bridge) where the road goes over Cam Stream, a point feature.

i) Main Street shares some pavement with Route 5, and is always four lanes wide.

j) There are two buildings along Main Street; each can be seen either as a point or as a rectangle footprint.

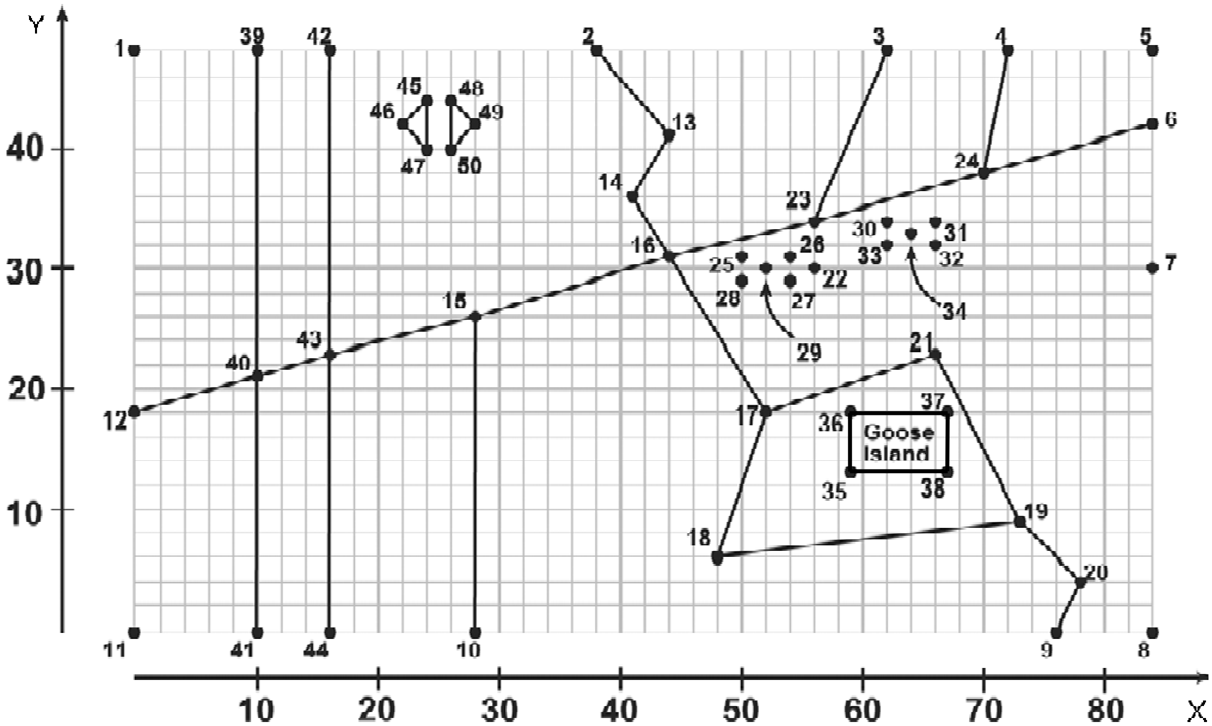
k) There is a one-lane road forming part of the boundary of the State Forest, shown as a grey line with black borders.

l) There are two fish ponds, which are seen as a collective, not as individuals; that is, they are a multi-polygon.

C.2.2 Test data points and coordinates

Figure B.2 depicts the points that are used to represent the map.

Dimensions in metres



Key

X Easting, in metres

Y Northing, in metres

Figure C 2: Points in the Blue Lake data set

Table B.1 gives these coordinates associated with each point.

Table C 1: Coordinates associated with each point in the Blue Lake data set

Point	Easting	Northing	Point	Easting	Northing
1	0	48	26	52	31
2	38	48	27	52	29
3	62	48	28	50	29
4	72	48	29	52	30
5	84	48	30	62	34
6	84	42	31	66	34
7	84	30	32	66	32
8	84	0	33	62	32
9	76	0	34	64	33
10	28	0	35	59	13
11	0	0	36	59	18
12	0	18	37	67	18
13	44	41	38	67	13
14	41	36	39	10	48
15	28	26	40	10	21
16	44	31	41	10	0
17	52	18	42	16	48
18	48	6	43	16	23
19	73	9	44	16	0
20	78	4	45	24	44
21	66	23	46	22	42
22	56	30	47	24	40
23	56	34	48	26	44
24	70	38	49	28	42
25	50	31	50	26	40

C.3 Conformance tests

C.3.1 Normalized geometry schema

C.3.1.1 Conformance test overview

The scope of this test is to determine that the test data (once inserted) are accessible via the schema defined in the standard. Table B.2 shows the queries that accomplish this test.

Table C 2: Queries to determine that test data are accessible via the normalized geometry schema

ID	Functionality Tested	Query Description	Answer
N1	GEOMETRY_COLUMNS table/view is created/updated properly	For this test, we will check to see that all of the feature tables are represented by entries in the GEOMETRY_COLUMNS table/view.	lakes, road_segments, divided_routes, buildings, buildings, forests, bridges, named_places, streams, ponds, map_neatlines
N2	GEOMETRY_COLUMNS table/view is created/updated properly	For this test, we will check to see that all of the geometry tables are represented by entries in the GEOMETRY_COLUMNS table/view.	lake_geom, road_segment_geom, divided_route_geom, forest_geom, bridge_geom, stream_geom, building_pt_geom, building_area_geom, pond_geom, named_place_geom, map_neatline_geom
N3	GEOMETRY_COLUMNS table/view is created/updated properly	For this test, we will check to see that the correct storage type for the streams table is represented in the GEOMETRY_COLUMNS table/view.	0
N4	GEOMETRY_COLUMNS table/view is created/updated properly	For this test, we will check to see that the correct geometry type for the streams table is represented in the GEOMETRY_COLUMNS table/view.	3 (corresponds to‘LINESTRING’)
N5	GEOMETRY_COLUMNS table/view is created/updated properly	For this test, we will check to see that the correct coordinate dimension for the streams table is represented in the GEOMETRY_COLUMNS table/view.	2
N6	GEOMETRY_COLUMNS table/view is created/updated properly	For this test, we will check to see that the correct value of max_ppr for the streams table is represented in the GEOMETRY_COLUMNS table/view.	3
N7	GEOMETRY_COLUMNS table/view is created/updated properly	For this test, we will check to see that the correct value of srid for the streams table is represented in the GEOMETRY_COLUMNS table/view.	101

N8	SPATIAL_REF_SYS table/view is created/updated properly	For this test, we will check to see that the correct value of srtext is represented in the SPATIAL_REF_SYS table/view.	'PROJCS["UTM_ZONE_14N", GEOGCS["World Geodetic System 72", DATUM["WGS_72", ELLIPSOID["NWL_10D", 6378135, 298.26]], PRIMEM["Greenwich", 0], UNIT["Meter", 1.0]], PROJECTION["Transverse_Mercator"], PARAMETER["False_Easting", 500000.0], PARAMETER["False_Northing", 0.0], PARAMETER["Central_Meridian", -99.0], PARAMETER["Scale_Factor", 0.9996], PARAMETER["Latitude_of_origin", 0.0], UNIT["Meter", 1.0]]'

C.3.1.2 Normalized geometry schema construction

```
-- CREATE SPATIAL_REF_SYS METADATA TABLE
CREATE TABLE spatial_ref_sys (
     srid        INTEGER NOT NULL PRIMARY KEY,
     auth_name   CHARACTER VARYING,
     auth_srid   INTEGER,
     srtext      CHARACTER VARYING(2048));
-- CREATE GEOMETRY_COLUMNS METADATA TABLE
CREATE TABLE geometry_columns (
     f_catalog_name    CHARACTER VARYING,
     f_table_schema    CHARACTER VARYING,
     f_table_name      CHARACTER VARYING,
     f_geometry_column CHARACTER VARYING,
     g_catalog_name    CHARACTER VARYING,
     g_table_schema    CHARACTER VARYING,
     g_table_name      CHARACTER VARYING,
     storage_type      INTEGER,
     geometry_type     INTEGER,
     coord_dimension   INTEGER,
     max_ppr           INTEGER,
     srid              INTEGER REFERENCES spatial_ref_sys,
     CONSTRAINT gc_pk  PRIMARY KEY (f_catalog_name, f_table_schema,
     f_table_name, f_geometry_column));
-- Create geometry tables
-- Lake Geometry
CREATE TABLE lake_geom (
     gid    INTEGER NOT NULL,
     eseq   INTEGER NOT NULL,
     etype  INTEGER NOT NULL,
     seq    INTEGER NOT NULL,
     x1     INTEGER,
     y1     INTEGER,
     x2     INTEGER,
     y2     INTEGER,
     x3     INTEGER,
     y3     INTEGER,
     x4     INTEGER,
     y4     INTEGER,
     x5     INTEGER,
     y5     INTEGER,
     CONSTRAINT l_gid_pk PRIMARY KEY (gid, eseq, seq));
-- Road Segment Geometry
```



```
CREATE TABLE road_segment_geom (
     gid      INTEGER NOT NULL,
     eseq     INTEGER NOT NULL,
     etype    INTEGER NOT NULL,
     seq      INTEGER NOT NULL,
     x1       INTEGER,
     y1       INTEGER,
     x2       INTEGER,
     y2       INTEGER,
     x3       INTEGER,
     y3       INTEGER,
     CONSTRAINT rs_gid_pk PRIMARY KEY (gid, eseq, seq));
-- Divided Route Geometry
CREATE TABLE divided_route_geom (
     gid      INTEGER NOT NULL,
     eseq     INTEGER NOT NULL,
     etype    INTEGER NOT NULL,
     seq      INTEGER NOT NULL,
     x1       INTEGER,
     y1       INTEGER,
     x2       INTEGER,
     y2       INTEGER,
     x3       INTEGER,
     y3       INTEGER,
     CONSTRAINT dr_gid_pk PRIMARY KEY (gid, eseq, seq));
-- Forest Geometry
CREATE TABLE forest_geom (
     gid      INTEGER NOT NULL,
     eseq     INTEGER NOT NULL,
     etype    INTEGER NOT NULL,
     seq      INTEGER NOT NULL,
     x1       INTEGER,
     y1       INTEGER,
     x2       INTEGER,
     y2       INTEGER,
     x3       INTEGER,
     y3       INTEGER,
     x4       INTEGER,
     y4       INTEGER,
     x5       INTEGER,
     y5       INTEGER,
     CONSTRAINT f_gid_pk PRIMARY KEY (gid, eseq, seq));
-- Bridge Geometry
```

```
CREATE TABLE bridge_geom (
     gid      INTEGER NOT NULL,
     eseq     INTEGER NOT NULL,
     etype    INTEGER NOT NULL,
     seq      INTEGER NOT NULL,
     x1       INTEGER,
     y1       INTEGER,
     CONSTRAINT b_gid_pk PRIMARY KEY (gid, eseq, seq));
-- Stream Geometry
CREATE TABLE stream_geom (
     gid      INTEGER NOT NULL,
     eseq     INTEGER NOT NULL,
     etype    INTEGER NOT NULL,
     seq      INTEGER NOT NULL,
     x1       INTEGER,
     y1       INTEGER,
     x2       INTEGER,
     y2       INTEGER,
     x3       INTEGER,
     y3       INTEGER,
     CONSTRAINT s_gid_pk PRIMARY KEY (gid, eseq, seq));
-- Bulding Point Geometry
CREATE TABLE building_pt_geom (
     gid      INTEGER NOT NULL,
     eseq     INTEGER NOT NULL,
     etype    INTEGER NOT NULL,
     seq      INTEGER NOT NULL,
     x1       INTEGER,
     y1       INTEGER,
     CONSTRAINT bp_gid_pk PRIMARY KEY (gid, eseq, seq));
-- Bulding Area Geometry
CREATE TABLE building_area_geom (
     gid      INTEGER NOT NULL,
     eseq     INTEGER NOT NULL,
     etype    INTEGER NOT NULL,
     seq      INTEGER NOT NULL,
     x1       INTEGER,
     y1       INTEGER,
     x2       INTEGER,
     y2       INTEGER,
     x3       INTEGER,
     y3       INTEGER,
     x4       INTEGER,
     y4       INTEGER,
     x5       INTEGER,
     y5       INTEGER,
     CONSTRAINT ba_gid_pk PRIMARY KEY (gid, eseq, seq));
-- Pond Geometry
```



```
CREATE TABLE pond_geom (
     gid     INTEGER NOT NULL,
     eseq    INTEGER NOT NULL,
     etype   INTEGER NOT NULL,
     seq     INTEGER NOT NULL,
     x1      INTEGER,
     y1      INTEGER,
     x2      INTEGER,
     y2      INTEGER,
     x3      INTEGER,
     y3      INTEGER,
     x4      INTEGER,
     y4      INTEGER,
     CONSTRAINT p_gid_pk PRIMARY KEY (gid, eseq, seq));
-- Named Place Geometry
CREATE TABLE named_place_geom (
     gid     INTEGER NOT NULL,
     eseq    INTEGER NOT NULL,
     etype   INTEGER NOT NULL,
     seq     INTEGER NOT NULL,
     x1      INTEGER,
     y1      INTEGER,
     x2      INTEGER,
     y2      INTEGER,
     x3      INTEGER,
     y3      INTEGER,
     x4      INTEGER,
     y4      INTEGER,
     CONSTRAINT np_gid_pk PRIMARY KEY (gid, eseq, seq));
-- Map Neatline Geometry
CREATE TABLE map_neatline_geom (
     gid     INTEGER NOT NULL,
     eseq    INTEGER NOT NULL,
     etype   INTEGER NOT NULL,
     seq     INTEGER NOT NULL,
     x1      INTEGER,
     y1      INTEGER,
     x2      INTEGER,
     y2      INTEGER,
     x3      INTEGER,
     y3      INTEGER,
     x4      INTEGER,
     y4      INTEGER,
     x5      INTEGER,
     y5      INTEGER,
     CONSTRAINT mn_gid_pk PRIMARY KEY (gid, eseq, seq));
```

Copyright © 2007 Open Geospatial Consortium, Inc. All Rights Reserved.

```
-- Lakes
CREATE TABLE lakes (
     fid          INTEGER NOT NULL PRIMARY KEY,
       name      CHARACTER VARYING(64),
     shore_gid    INTEGER);
-- Road Segments
CREATE TABLE road_segments (
     fid                  INTEGER NOT NULL PRIMARY KEY,
     name                 CHARACTER VARYING(64),
     aliases              CHARACTER VARYING(64),
     num_lanes            INTEGER,
     centerline_gid       INTEGER);
-- Divided Routes
CREATE TABLE divided_routes (
     fid                  INTEGER NOT NULL PRIMARY KEY,
     name                 CHARACTER VARYING(64),
     num_lanes            INTEGER,
     centerlines_gid      INTEGER);
-- Forests
CREATE TABLE forests (
     fid                  INTEGER NOT NULL PRIMARY KEY,
     name                 CHARACTER VARYING(64),
     boundary_gid         INTEGER);
-- Bridges
CREATE TABLE bridges (
     fid               INTEGER NOT NULL PRIMARY KEY,
     name              CHARACTER VARYING(64),
     position_gid      INTEGER);
-- Streams
CREATE TABLE streams (
     fid        INTEGER NOT NULL PRIMARY KEY,
     name    CHARACTER VARYING(64),
     centerline_gid     INTEGER);
-- Buildings
CREATE TABLE buildings (
     fid                INTEGER NOT NULL PRIMARY KEY,
     address            CHARACTER VARYING(64),
     position_gid       INTEGER,
     footprint_gid      INTEGER);
-- Ponds
CREATE TABLE ponds (
     fid              INTEGER NOT NULL PRIMARY KEY,
     name             CHARACTER VARYING(64),
     type             CHARACTER VARYING(64),
     shores_gid       INTEGER);
-- Named Places
```

```
CREATE TABLE named_places (
     fid              INTEGER NOT NULL PRIMARY KEY,
     name             CHARACTER VARYING(64),
     boundary_gid     INTEGER);
-- Map Neatline
CREATE TABLE map_neatlines (
     fid              INTEGER NOT NULL PRIMARY KEY,
     neatline_gid     INTEGER);
```

C.3.1.3 Normalized geometry schema data loading

```
--Spatial Reference System
INSERT INTO spatial_ref_sys VALUES(101, 'POSC', 32214,
     'PROJCS["UTM_ZONE_14N", GEOGCS["World Geodetic System
     72",DATUM["WGS_72",  ELLIPSOID["NWL_10D", 6378135,
     298.26]],PRIMEM["Greenwich",
     0],UNIT["Meter",1.0]],PROJECTION["Transverse_Mercator"],
     PARAMETER["False_Easting", 500000.0],PARAMETER["False_Northing",
     0.0],PARAMETER["Central_Meridian", -99.0],PARAMETER["Scale_Factor",
     0.9996],PARAMETER["Latitude_of_origin", 0.0],UNIT["Meter", 1.0]]');
-- Lakes
INSERT INTO lake_geom VALUES(101, 1, 5, 1, 52,18, 66,23, 73,9, 48,6,
     52,18);
INSERT INTO lake_geom VALUES(101, 2, 5, 1, 59,18, 67,18, 67,13, 59,13,
     59,18);
INSERT INTO lakes VALUES (
     101, 'BLUE LAKE', 101);
-- Road segments
INSERT INTO road_segment_geom VALUES (
     101, 1, 3, 1, 0,18, 10,21, 16,23);
INSERT INTO road_segment_geom VALUES (
     101, 1, 3, 2, 28,26, 44,31, NULL,NULL);
INSERT INTO road_segment_geom VALUES (
     102, 1, 3, 1, 44,31, 56,34, 70,38);
INSERT INTO road_segment_geom VALUES (
     103, 1, 3, 1, 70,38, 72,48, NULL,NULL);
INSERT INTO road_segment_geom VALUES (
     104, 1, 3, 1, 70,38, 84,42, NULL,NULL);
INSERT INTO road_segment_geom VALUES (
     105, 1, 3, 1, 28,26, 28,0, NULL,NULL);
INSERT INTO road_segments VALUES(102, 'Route 5', NULL, 2, 101);
INSERT INTO road_segments VALUES(103, 'Route 5', 'Main Street', 4, 102);
INSERT INTO road_segments VALUES(104, 'Route 5', NULL, 2, 103);
INSERT INTO road_segments VALUES(105, 'Main Street', NULL, 4, 104);
```

```
INSERT INTO road_segments VALUES(106, 'Dirt Road by Green Forest', NULL,
     1, 105);
-- DividedRoutes
INSERT INTO divided_route_geom VALUES(101, 1, 9, 1, 10,48, 10,21, 10,0);
INSERT INTO divided_route_geom VALUES(101, 2, 9, 1, 16,0, 10,23, 16,48);
INSERT INTO divided_routes VALUES(119, 'Route 75', 4, 101);
-- Forests
INSERT INTO forest_geom VALUES(101, 1, 11, 1, 28,26, 28,0, 84,0, 84,42,
     28,26);
INSERT INTO forest_geom VALUES(101, 1, 11, 2, 52,18, 66,23, 73,9, 48,6,
     52,18);
INSERT INTO forest_geom VALUES(101, 2, 11, 1, 59,18, 67,18, 67,13, 59,13,
     59,18);
INSERT INTO forests VALUES(109, 'Green Forest', 101);
-- Bridges
INSERT INTO bridge_geom VALUES(101, 1, 1, 1, 44, 31);
INSERT INTO bridges VALUES(110, 'Cam Bridge', 101);
-- Streams
INSERT INTO stream_geom VALUES(101, 1, 3, 1, 38,48, 44,41, 41,36);
INSERT INTO stream_geom VALUES(101, 1, 3, 2, 44,31, 52,18, NULL,NULL);
INSERT INTO stream_geom VALUES(102, 1, 3, 1, 76,0, 78,4, 73,9 );
--
INSERT INTO streams VALUES(111, 'Cam Stream', 101);
INSERT INTO streams VALUES(112, 'Cam Stream', 102);
-- Buildings
INSERT INTO building_pt_geom VALUES(101, 1, 1, 1, 52,30);
INSERT INTO building_pt_geom VALUES(102, 1, 1, 1, 64,33);
INSERT INTO building_area_geom VALUES(101, 1, 5, 1, 50,31, 54,31,
      54,29, 50,29, 50,31);
INSERT INTO building_area_geom VALUES(102, 1, 5, 1, 66,34, 62,34, 62,32,
      66,32, 66,34);
INSERT INTO buildings VALUES(113, '123 Main Street', 101, 101);
INSERT INTO buildings VALUES(114, '215 Main Street', 102, 102);
-- Ponds
INSERT INTO pond_geom VALUES(101, 1, 11, 1, 24,44, 22,42, 24,40, 24,44 );
INSERT INTO pond_geom VALUES(101, 2, 11, 1, 26,44, 26,40, 28,42, 26,44 );
INSERT INTO ponds VALUES(120, NULL, 'Stock Pond', 101);
-- Named Places
INSERT INTO named_place_geom VALUES(101, 1, 5, 1, 62,48, 84,48, 84,30,
     56,30);
INSERT INTO named_place_geom VALUES(101, 1, 5, 2, 56,30, 56,34, 62,48,
     NULL,NULL);
INSERT INTO named_place_geom VALUES(102, 1, 5, 1, 67,13, 67,18, 59,18,
     59,13);
INSERT INTO named_place_geom VALUES(102, 1, 5, 2, 59,13, 67,13,
     NULL,NULL, NULL,NULL);
INSERT INTO named_places VALUES(117, 'Ashton', 101);
INSERT INTO named_places VALUES(118, 'Goose Island', 102);
-- Map Neatlines
```

```
INSERT INTO map_neatline_geom VALUES(101, 1, 5, 1, 0,0, 0,48, 84,48,
     84,0, 0,0);
INSERT INTO map_neatlines VALUES(115, 101);
-- Geometry Columns
INSERT INTO geometry_columns VALUES (
     'lakes', 'shore_gid',
     'lake_geom',0, 5, 2, 5, 101);
INSERT INTO geometry_columns VALUES (
     'road_segments', 'centerline_gid',
     'road_segment_geom',0, 3, 2, 3, 101);
INSERT INTO geometry_columns VALUES (
     'divided_routes', 'centerlines_gid',
     'divided_route_geom',0, 9, 2, 3, 101);
INSERT INTO geometry_columns VALUES (
     'forests', 'boundary_gid',
     'forest_geom',0, 11, 2, 5, 101);
INSERT INTO geometry_columns VALUES (
     'bridges', 'position_gid',
     'bridge_geom',0, 1, 2, 1, 101);
INSERT INTO geometry_columns VALUES (
     'streams', 'centerline_gid',
     'stream_geom',0, 3, 2, 3, 101);
INSERT INTO geometry_columns VALUES (
     'buildings', 'position_gid',
     'building_pt_geom',0, 1, 2, 1, 101);
INSERT INTO geometry_columns VALUES (
     'buildings', 'footprint_gid',
     'building_area_geom',0, 5, 2, 5, 101);
INSERT INTO geometry_columns VALUES (
     'ponds', 'shores_gid',
     'pond_geom',0, 11, 2, 4, 101);
INSERT INTO geometry_columns VALUES (
     'named_places', 'boundary_gid',
     'named_place_geom',0, 5, 2, 4, 101);
INSERT INTO geometry_columns VALUES (
     'map_neatlines', 'neatline_gid',
     'map_neatline_geom',0, 5, 2, 5, 101);
```

C.3.1.4 Normalized geometry schema test queries

```
-- Conformance Item N1
SELECT f_table_name
     FROM geometry_columns;
-- Conformance Item N2
```

```
SELECT g_table_name
      FROM geometry_columns;
-- Conformance Item N3
SELECT storage_type
FROM geometry_columns
WHERE f_table_name = 'streams';
-- Conformance Item N4
SELECT geometry_type
FROM geometry_columns
WHERE f_table_name = 'streams';
-- Conformance Item N5
SELECT coord_dimension
      FROM geometry_columns
      WHERE f_table_name = 'streams';
-- Conformance Item N6
SELECT max_ppr
      FROM geometry_columns
      WHERE f_table_name = 'streams';
-- Conformance Item N7
SELECT srid
      FROM geometry_columns
      WHERE f_table_name = 'streams';
-- Conformance Item N8
SELECT srtext
      FROM SPATIAL_REF_SYS
      WHERE SRID = 101;
```

C.3.2 Binary geometry schema

C.3.2.1 Conformance test overview

The scope of this test is to determine that the test data (once inserted) are accessible via the schema defined in the standard. Table B.3 shows the queries that accomplish this test.

Table C 3: Queries to determine that test data are accessible via the binary geometry schema

ID	Functionality Tested	Query Description	Answer
B1	Table B.1 — GEOMETRY_COLUMNS table/view is created/updated properly	For this test, we will check to see that all of the feature tables are represented by entries in the GEOMETRY_COLUMNS table/view.	lakes, road_segments, divided_routes, buildings, buildings, forests, bridges, named_places, streams, ponds, map_neatlines
B2	GEOMETRY_COLUMNS table/view is created/updated properly	For this test, we will check to see that all of the geometry tables are represented by entries in the GEOMETRY_COLUMNS table/view.	lake_geom, road_segment_geom, divided_route_geom, forest_geom, bridge_geom, stream_geom, building_pt_geom, building_area_geom, pond_geom, named_place_geom, map_neatline_geom

B3	GEOMETRY_COLUMNS table/view is created/updated properly	For this test, we will check to see that the correct storage type for the streams table is represented in the GEOMETRY_COLUMNS table/view.	1
B4	GEOMETRY_COLUMNS table/view is created/updated properly	For this test, we will check to see that the correct geometry type for the streams table is represented in the GEOMETRY_COLUMNS table/view.	3 (corresponds to 'LINESTRING')
B5	GEOMETRY_COLUMNS table/view is created/updated properly	For this test, we will check to see that the correct coordinate dimension for the streams table is represented in the GEOMETRY_COLUMNS table/view.	2
B6	GEOMETRY_COLUMNS table/view is created/updated properly	For this test, we will check to see that the correct value of srid for the streams table is represented in the GEOMETRY_COLUMNS table/view.	101
B7	SPATIAL_REF_SYS table/view is created/updated properly	For this test, we will check to see that the correct value of srtext is represented in the SPATIAL_REF_SYS table/view.	'PROJCS["UTM_ZONE_14N", GEOGCS["World Geodetic System 72", DATUM["WGS_72", ELLIPSOID["NWL_10D", 6378135, 298.26]], PRIMEM["Greenwich", 0], UNIT["Meter", 1.0]], PROJECTION["Transverse_Mercator"], PARAMETER["False_Easting", 500000.0], PARAMETER["False_Northing", 0.0], PARAMETER["Central_Meridian", -99.0], PARAMETER["Scale_Factor", 0.9996], PARAMETER["Latitude_of_origin", 0.0], UNIT["Meter", 1.0]]'

C.3.2.2 Binary geometry schema construction

```
CREATE TABLE spatial_ref_sys (
     srid       INTEGER NOT NULL PRIMARY KEY,
     auth_name  CHARACTER VARYING,
     auth_srid  INTEGER,
     srtext     CHARACTER VARYING(2048));
-- Geometry Columns
CREATE TABLE geometry_columns (
     f_table_schema    CHARACTER VARYING,
     f_table_name      CHARACTER VARYING,
     f_geometry_column CHARACTER VARYING,
      g_table_schema    CHARACTER VARYING,
     g_table_name      CHARACTER VARYING,
     storage_type      INTEGER,
     geometry_type     INTEGER,
     coord_dimension   INTEGER,
     max_ppr   INTEGER,
     srid      INTEGER REFERENCES spatial_ref_sys,
     CONSTRAINT gc_pk PRIMARY KEY (f_table_schema, f_table_name,
     f_geometry_column));
-- Lake Geometry
CREATE TABLE lake_geom (
     gid  INTEGER NOT NULL PRIMARY KEY,
     xmin INTEGER,
     ymin INTEGER,
     xmax INTEGER,
     ymax INTEGER,
     wkbgeometry  VARBINARY);
-- Road Segment Geometry
CREATE TABLE road_segment_geom (
     gid  INTEGER NOT NULL PRIMARY KEY,
     xmin INTEGER,
     ymin INTEGER,
     xmax INTEGER,
     ymax INTEGER,
     wkbgeometry  VARBINARY);
-- Divided Route Geometry
CREATE TABLE divided_route_geom (
     gid  INTEGER NOT NULL PRIMARY KEY,
     xmin INTEGER,
     ymin INTEGER,
     xmax INTEGER,
     ymax INTEGER,
     wkbgeometry  VARBINARY);
-- Forest Geometry
```

```
CREATE TABLE forest_geom (
      gid  INTEGER NOT NULL PRIMARY KEY,
      xmin INTEGER,
      ymin INTEGER,
      xmax INTEGER,
      ymax INTEGER,
      wkbgeometry  VARBINARY);
-- Bridge Geometry
CREATE TABLE bridge_geom (
      gid  INTEGER NOT NULL PRIMARY KEY,
      xmin INTEGER,
      ymin INTEGER,
      xmax INTEGER,
      ymax INTEGER,
      wkbgeometry  VARBINARY);
-- Stream Geometry
CREATE TABLE stream_geom (
      gid  INTEGER NOT NULL PRIMARY KEY,
      xmin INTEGER,
      ymin INTEGER,
        xmax INTEGER,
      ymax INTEGER,
      wkbgeometry  VARBINARY);
-- Bulding Point Geometry
CREATE TABLE building_pt_geom (
      gid  INTEGER NOT NULL PRIMARY KEY,
      xmin INTEGER,
      ymin INTEGER,
      xmax INTEGER,
      ymax INTEGER,
      wkbgeometry  VARBINARY);
-- Bulding Area Geometry
CREATE TABLE building_area_geom (
      gid  INTEGER NOT NULL PRIMARY KEY,
      xmin INTEGER,
      ymin INTEGER,
      xmax INTEGER,
      ymax INTEGER,
      wkbgeometry  VARBINARY);
-- Pond Geometry
```

```
CREATE TABLE pond_geom (
      gid  INTEGER NOT NULL PRIMARY KEY,
      xmin INTEGER,
      ymin INTEGER,
      xmax INTEGER,
      ymax INTEGER,
      wkbgeometry  VARBINARY);
-- Named Place Geometry
CREATE TABLE named_place_geom (
      gid  INTEGER NOT NULL PRIMARY KEY,
      xmin INTEGER,
      ymin INTEGER,
      xmax INTEGER,
      ymax INTEGER,
      wkbgeometry  VARBINARY);
-- Map Neatline Geometry
CREATE TABLE map_neatline_geom (
      gid  INTEGER NOT NULL PRIMARY KEY,
      xmin INTEGER,
      ymin INTEGER,
      xmax INTEGER,
      ymax INTEGER,
      wkbgeometry  VARBINARY);
-- Lakes
CREATE TABLE lakes (
      fid       INTEGER NOT NULL PRIMARY KEY,
      name      CHARACTER VARYING(64),
      shore_gid INTEGER);
-- Road Segments
CREATE TABLE road_segments (
      fid       INTEGER NOT NULL PRIMARY KEY,
      name      CHARACTER VARYING(64),
      aliases   CHARACTER VARYING(64),
      num_lanes INTEGER,
      centerline_gid    INTEGER);
-- Divided Routes
CREATE TABLE divided_routes (
      fid       INTEGER NOT NULL PRIMARY KEY,
      name      CHARACTER VARYING(64),
      num_lanes INTEGER,
      centerlines_gid   INTEGER);
-- Forests
CREATE TABLE forests (
      fid    INTEGER NOT NULL PRIMARY KEY,
      name   CHARACTER VARYING(64),
      boundary_gid   INTEGER);
-- Bridges
```

```
CREATE TABLE bridges (
     fid   INTEGER NOT NULL PRIMARY KEY,
     name  CHARACTER VARYING(64),
     position_gid  INTEGER);
-- Streams
CREATE TABLE streams (
     fid     INTEGER NOT NULL PRIMARY KEY,
     name    CHARACTER VARYING(64),
     centerline_gid  INTEGER);
-- Buildings
CREATE TABLE buildings (
     fid     INTEGER NOT NULL PRIMARY KEY,
     address CHARACTER VARYING(64),
     position_gid    INTEGER,
     footprint_gid   INTEGER);
-- Ponds
CREATE TABLE ponds (
     fid     INTEGER NOT NULL PRIMARY KEY,
     name    CHARACTER VARYING(64),
     type    CHARACTER VARYING(64),
     shores_gid      INTEGER);
-- Named Places
CREATE TABLE named_places (
     fid     INTEGER NOT NULL PRIMARY KEY,
     name    CHARACTER VARYING(64),
     boundary_gid    INTEGER);
-- Map Neatline
CREATE TABLE map_neatlines (
     fid     INTEGER NOT NULL PRIMARY KEY,
     neatline_gid    INTEGER);
```

C.3.2.3 Binary geometry schema data loading

```
-- Spatial Reference Systems
INSERT INTO spatial_ref_sys VALUES
```

```
        (101, 'POSC', 32214,
        'PROJCS["UTM_ZONE_14N",
        GEOGCS["World Geodetic System 72",
        DATUM["WGS_72",ELLIPSOID["NWL_10D",6378135,298.26]],
        PRIMEM["Greenwich",0],
        UNIT["Meter",1.0]],
        PROJECTION["Transverse_Mercator"],
        PARAMETER["False_Easting", 500000.0],
        PARAMETER["False_Northing", 0.0],
        PARAMETER["Central_Meridian", -99.0],
        PARAMETER["Scale_Factor", 0.9996],
        PARAMETER["Latitude_of_origin", 0.0],
        UNIT["Meter", 1.0]]'
        );
-- Lakes
INSERT INTO lake_geom VALUES(101, 48.0, 6.0, 73.0, 23.0,
        HEXTOVARBINARY('010300000002000000050000000000000000004a400000000000
        00324000000000008050400000000000003740000000000040524000000000000022
        40000000000000484000000000000018400000000000004a40000000000000324005
        000000000000000000804d4000000000000032400000000000c05040000000000032
        400000000000c050400000000000002a400000000000804d400000000000002a4000
        00000000804d400000000000003240');
INSERT INTO lakes VALUES (
        101, 'BLUE LAKE', 101);
-- Road segments
INSERT INTO road_segment_geom VALUES (
        101, 0.0, 18.0, 44.0, 31.0,
        HEXTOVARBINARY('0102000000050000000000000000000000000000000000324000
        00000000002440000000000000354000000000000030400000000000003740000000
        0000003c400000000000003a40000000000000464000000000000003f40');
INSERT INTO road_segment_geom VALUES (
        102, 44.0, 31.0, 70.0, 38.0,
        HEXTOVARBINARY('010200000003000000000000000000464000000000000003f4000
        00000000004c400000000000004140000000000080514000000000000004340');
INSERT INTO road_segment_geom VALUES (
        103, 70.0, 38.0, 72.0, 48.0,
        HEXTOVARBINARY('0102000000020000000000000000805140000000000000434000
        00000000005240000000000000004840');
INSERT INTO road_segment_geom VALUES (
        104, 70.0, 38.0, 84.0, 42.0,
        HEXTOVARBINARY('0102000000020000000000000000805140000000000000434000
        00000000005540000000000000004540');
INSERT INTO road_segment_geom VALUES (
        105, 28.0, 0.0, 28.0, 26.0,
        HEXTOVARBINARY('0102000000020000000000000000805140000000000000434000
        00000000005540000000000000004540');
INSERT INTO road_segments VALUES(102, 'Route 5', NULL, 2, 101);
INSERT INTO road_segments VALUES(103, 'Route 5', 'Main Street', 4, 102);
INSERT INTO road_segments VALUES(104, 'Route 5', NULL, 2, 103);
```

```
INSERT INTO road_segments VALUES(105, 'Main Street', NULL, 4, 104);
INSERT INTO road_segments VALUES(106, 'Dirt Road by Green Forest', NULL,
     1, 105);
-- DividedRoutes
INSERT INTO divided_route_geom VALUES(101, 10.0, 0.0, 16.0, 48.0,
     HEXTOVARBINARY('0105000000020000000102000000030000000000000000002440
     00000000000048400000000000002440000000000000354000000000000024400000
     00000000000001020000000300000000000000000000304000000000000000000000
     0000002440000000000000374000000000000030400000000000004840');
INSERT INTO divided_routes VALUES(119, 'Route 75', 4, 101);
-- Forests
INSERT INTO forest_geom VALUES(101, 28.0, 0.0, 84.0, 42.0,
     HEXTOVARBINARY('0106000000020000000103000000020000000500000000000000
     00003c400000000000003a400000000000003c400000000000000000000000000000
     554000000000000000000000000000005540000000000000454000000000000003c40
     0000000000003a4005000000000000000000004a40000000000000324000000000000080
     5040000000000000374000000000004052400000000000002240000000000000004840
     00000000000018400000000000004a400000000000003240010300000001000000 05
     0000000000000804d400000000000003240000000000c05040000000000000032
     400000000000c050400000000000002a400000000000804d400000000000002a4000
     00000000804d400000000000003240');
INSERT INTO forests VALUES(109, 'Green Forest', 101);
-- Bridges
INSERT INTO bridge_geom VALUES(101, 44.0, 31.0, 44.0, 31.0,
     HEXTOVARBINARY('010100000000000000000046400000000000003f40');
INSERT INTO bridges VALUES(110, 'Cam Bridge', 101);
-- Streams
INSERT INTO stream_geom VALUES(101, 38.0, 18.0, 52.0, 48.0,
     HEXTOVARBINARY('0102000000050000000000000000004340000000000000484000
     000000000046400000000000008044400000000000008044400000000000004240000000
     0000004640000000000000003f400000000000004a400000000000003240');
INSERT INTO stream_geom VALUES(102, 73.0, 0.0, 78.0, 9.0,
     HEXTOVARBINARY('0102000000030000000000000000005340000000000000000000
     0000000080534000000000000010400000000000405240000000000000002240');
INSERT INTO streams VALUES(111, 'Cam Stream', 101);
INSERT INTO streams VALUES(112, 'Cam Stream', 102);
-- Buildings
INSERT INTO building_pt_geom VALUES(101, 52.0, 30.0, 52.0, 30.0,
     HEXTOVARBINARY('01010000000000000000004a400000000000003e40');
INSERT INTO building_pt_geom VALUES(102, 64.0, 33.0, 64.0, 33.0,
     HEXTOVARBINARY('010100000000000000000050400000000000804040');
INSERT INTO building_area_geom VALUES(101, 50.0, 29.0, 54.0, 31.0,
     HEXTOVARBINARY('0103000000010000000500000000000000000049400000000000
     003f400000000000004b400000000000003f400000000000004b400000000000003d
     400000000000004940000000000000003d400000000000004940000000000000003f40')
     ;
```

```
INSERT INTO building_area_geom VALUES(102, 62.0, 32.0, 66.0, 34.0,
     HEXTOVARBINARY('0103000000010000000500000000000000008050400000000000
     0041400000000000004f4000000000000041400000000000004f4000000000000040
     400000000000805040000000000000404000000000008050400000000000004140')
     ;
INSERT INTO buildings VALUES(113, '123 Main Street', 101, 101);
INSERT INTO buildings VALUES(114, '215 Main Street', 102, 102);
-- Ponds
INSERT INTO pond_geom VALUES(101, 22.0, 40.0, 28.0, 44.0,
     HEXTOVARBINARY('0106000000020000000103000000010000000400000000000000
     00003840000000000000464000000000000036400000000000004540000000000000
     38400000000000004440000000000000384000000000000046400103000000010000
     00040000000000000000003a4000000000000046400000000000003a400000000000
     0044400000000000003c4000000000000045400000000000003a4000000000000046
     40');
INSERT INTO ponds VALUES(120, NULL, 'Stock Pond', 101);
-- Named Places
INSERT INTO named_place_geom VALUES(101, 56.0, 30.0, 84.0, 48.0,
     HEXTOVARBINARY('010300000001000000060000000000000000004f400000000000
     00484000000000000055400000000000004840000000000000554000000000000003e
     400000000000004c400000000000003e400000000000004c40000000000000414000
     00000000004f400000000000004840');
INSERT INTO named_place_geom VALUES(102, 59.0, 13.0, 67.0, 18.0,
     HEXTOVARBINARY('0103000000010000000500000000000000000c050400000000000
     002a400000000000c05040000000000000324000000000000804d4000000000000032
     400000000000804d400000000000002a400000000000c050400000000000002a40')
     ;
INSERT INTO named_places VALUES(117, 'Ashton', 101);
INSERT INTO named_places VALUES(118, 'Goose Island', 102);
-- Map Neatlines
INSERT INTO map_neatline_geom VALUES(101, 0.0, 0.0, 84.0, 48.0,
     HEXTOVARBINARY('0103000000010000000500000000000000000000000000000000
     00000000000000000000000000000000004840000000000000554000000000000048
     40000000000000554000000000000000000000000000000000000000000000000000')
     ;
INSERT INTO map_neatlines VALUES(115, 101);
--Geometry Columns
INSERT INTO geometry_columns VALUES (
     'lakes', 'shore_gid',
     'lake_geom',1, 5, 2, 0);
INSERT INTO geometry_columns VALUES (
     'road_segments',
     'centerline_gid', 'road_segment_geom',1, 3, 2, 0);
INSERT INTO geometry_columns VALUES (
     'divided_routes',
     'centerlines_gid', 'divided_route_geom',1, 9, 2, 0);
INSERT INTO geometry_columns VALUES (
     'forests', 'boundary_gid',
     'forest_geom',1, 11, 2, 0);
```



```
INSERT INTO geometry_columns VALUES (
     'bridges', 'position_gid',
     'bridge_geom',1, 1, 2, 0);
INSERT INTO geometry_columns VALUES (
     'streams', 'centerline_gid',
     'stream_geom',1, 3, 2, 0);
INSERT INTO geometry_columns VALUES (
     'buildings', 'position_gid',
     'building_pt_geom',1, 1, 2, 0);
INSERT INTO geometry_columns VALUES (
     'buildings', 'footprint_gid',
     'building_area_geom',1, 5, 2, 0);
INSERT INTO geometry_columns VALUES (
     'ponds', 'shores_gid',
     'pond_geom',1, 11, 2, 0);
INSERT INTO geometry_columns VALUES (
     'named_places', 'boundary_gid',
     'named_place_geom',1, 5, 2, 0);
INSERT INTO geometry_columns VALUES (
     'map_neatlines', 'neatline_gid',
     'map_neatline_geom',1, 5, 2, 0);
```

C.3.2.4 Normalized geometry schema test queries

```
-- Conformance Item B1
SELECT f_table_name
     FROM geometry_columns;
-- Conformance Item B2
SELECT g_table_name
     FROM geometry_columns;
-- Conformance Item B3
SELECT storage_type
     FROM geometry_columns
     WHERE f_table_name = 'streams';
-- Conformance Item B4
SELECT geometry_type
     FROM geometry_columns
     WHERE f_table_name = 'streams';
-- Conformance Item B5
SELECT coord_dimension
     FROM geometry_columns
     WHERE f_table_name = 'streams';
-- Conformance Item B6
```

```
SELECT srid
      FROM geometry_columns
      WHERE f_table_name = 'streams';
-- Conformance Item B7
SELECT srtext
      FROM SPATIAL_REF_SYS
      WHERE SRID = 101;
```

C.3.3 Geometry types and functions

The scope of this test determines that

a) the database of the test (once inserted) is accessible via the schema defined in this standard;

b) that the functionality defined in this standard is implemented as described.

Table B.4 shows the queries that accomplish the first part of this test.

Table C 4: Queries that accomplish the test of geometry types and functions

ID	Functionality Tested	Query Description	Answer
T1	GEOMETRY_COLUMNS table/view is created/updated properly	For this test, we will check to see that all of the feature tables are represented by entries in the GEOMETRY_COLUMNS table/view.	lakes, road_segments, divided_routes, buildings, forests, bridges, named_places, streams, ponds, map_neatlines[a]
T2	GEOMETRY_COLUMNS table/view is created/updated properly	For this test, we will check to see that the correct geometry column for the streams table is represented in the GEOMETRY_COLUMNS table/view.	Centerline
T3	GEOMETRY_COLUMNS table/view is created/updated properly	For this test, we will check to see that the correct coordinate dimension for the streams table is represented in the GEOMETRY_COLUMNS table/view.	2
T4	GEOMETRY_COLUMNS table/view is created/updated properly	For this test, we will check to see that the correct value of srid for the streams table is represented in the GEOMETRY_COLUMNS table/view.	101[b]

ID	Functionality Tested	Query Description	Answer
T5	SPATIAL_REF_SYS table/view is created/updated properly	For this test, we will check to see that the correct value of srtext is represented in the SPATIAL_REF_SYS table/view.	'PROJCS["UTM_ZONE_14N", GEOGCS["World Geodetic System 72", DATUM["WGS_72", ELLIPSOID["NWL_10D", 6378135, 298.26]], PRIMEM["Greenwich", 0], UNIT["Meter", 1.0]], PROJECTION["Transverse_Mercator"], PARAMETER["False_Easting", 500000.0], PARAMETER["False_Northing", 0.0], PARAMETER["Central_Meridian", -99.0], PARAMETER["Scale_Factor", 0.9996], PARAMETER["Latitude_of_origin", 0.0], UNIT["Meter", 1.0]]'
T6	Dimension(g Geometry) : Integer	For this test, we will determine the dimension of Blue Lake.	2
T7	GeometryType(g Geometry) : String	For this test, we will determine the type of Route 75.	'MULTILINESTRING'
T8	AsText(g Geometry) : String	For this test, we will determine the WKT representation of Goose Island.	'POLYGON((67 13, 67 18, 59 18, 59 13, 67 13))' [c]
T9	AsBinary(g Geometry) : Blob	For this test, we will determine the WKB representation of Goose Island. We will test by applying AsText to the result of PolyFromText to the result of AsBinary.	'POLYGON((67 13, 67 18, 59 18, 59 13, 67 13))' [c]
T10	SRID(g Geometry) : Integer	For this test, we will determine the SRID of Goose Island.	101[b]
T11	IsEmpty(g Geometry) : Integer	For this test, we will determine whether the geometry of a segment of Route 5 is empty.	0 Some commercial SQL implementations with type extensibility systems support only BOOLEAN return values. Expected test results should be adjusted accordingly.
T12	IsSimple(g Geometry) : Integer	For this test, we will determine whether the geometry of a segment of Blue Lake is simple.	1 Some commercial SQL implementations with type extensibility systems support only BOOLEAN return values. Expected test results should be adjusted accordingly.
T13	Boundary(g Geometry) : Geometry	For this test, we will determine the boundary of Goose Island.	'LINESTRING(67 13, 67 18, 59 18, 59 13, 67 13)' or ‘MULTILINESTRING ((67 13, 67 18, 59 18, 59 13, 67 13))'

ID	Functionality Tested	Query Description	Answer
T14	Envelope(g Geometry) : Integer	For this test, we will determine the envelope of Goose Island.	‘POLYGON((59 13, 59 18, 67 18, 67 13, 59 13))'
T15	X(p Point) : Double Precision	For this test we will determine the X coordinate of Cam Bridge.	44,00
T16	Y(p Point) : Double Precision	For this test we will determine the Y coordinate of Cam Bridge.	31,00
T17	StartPoint(c Curve) : Point	For this test, we will determine the start point of road segment 102.	'POINT(0 18)'
T18	EndPoint(c Curve) : Point	For this test, we will determine the end point of road segment 102.	'POINT(44 31)'
T19	IsClosed(c Curve) : Integer	For this test, we will determine the boundary of Goose Island.	1 Some commercial SQL implementations with type extensibility systems support only BOOLEAN return values. Expected test results should be adjusted accordingly.
T20	IsRing(c Curve) : Integer	For this test, we will determine the boundary of Goose Island.	1 Some commercial SQL implementations with type extensibility systems support only BOOLEAN return values. Expected test results should be adjusted accordingly.
T21	Length(c Curve) : Double Precision	For this test, we will determine the length of road segment 106.	26,00 (in metres)
T22	NumPoints(l LineString) : Integer	For this test, we will determine the number of points in road segment 102.	5
T23	PointN(l LineString, n Integer) : Point	For this test, we will determine the 1st point in road segment 102.	'POINT(0 18)'
T24	Centroid(s Surface) : Point	For this test, we will determine the centroid of Goose Island.	'POINT(53 15.5)' [d]
T25	PointOnSurface(s Surface) : Point	For this test, we will determine a point on Goose Island[e].	1 Some commercial SQL implementations with type extensibility systems support only BOOLEAN return values. Expected test results should be adjusted accordingly.
T26	Area(s Surface) : Double Precision	For this test, we will determine the area of Goose Island.	40,00 (square metres)
T27	ExteriorRing(p Polygon) : LineString	For this test, we will determine the exterior ring of Blue Lake.	'LINESTRING(52 18, 66 23, 73 9, 48 6, 52 18)'
T28	NumInteriorRings(p Polygon) : Integer	For this test, we will determine the number of interior rings of Blue Lake.	1

ID	Functionality Tested	Query Description	Answer
T29	InteriorRingN(p Polygon, n Integer) : LineString	For this test, we will determine the first interior ring of Blue Lake.	'LINESTRING(59 18, 67 18, 67 13, 59 13, 59 18)'
T30	NumGeometries(g GeomCollection) : Integer	For this test, we will determine the number of geometries in Route 75.	2
T31	GeometryN(g GeomCollection, n Integer) : Geometry	For this test, we will determine the second geometry in Route 75.	'LINESTRING(16 0, 16 23, 16 48)'
T32	IsClosed(mc MultiCurve) : Integer	For this test, we will determine if the geometry of Route 75 is closed.	0 Some commercial SQL implementations with type extensibility systems support only BOOLEAN return values. Expected test results should be adjusted accordingly.
T33	Length(mc MultiCurve) : Double Precision	For this test, we will determine the length of Route 75.	96,00 (in metres)
T34	Centroid(ms MultiSurface) : Point	For this test, we will determine the centroid of the ponds.	'POINT(25 42)' [d]
T35	PointOnSurface(ms MultiSurface) : Point	For this test, we will determine a point on the ponds.[e]	1 Some commercial SQL implementations with type extensibility systems support only BOOLEAN return values. Expected test results should be adjusted accordingly.
T36	Area(ms MultiSurface) : Double Precision	For this test, we will determine the area of the ponds.	8,00 (in square metres)
T37	Equals(g1 Geometry, g2 Geometry) : Integer	For this test, we will determine if the geometry of Goose Island is equal to the same geometry as constructed from it's WKT representation.	1 Some commercial SQL implementations with type extensibility systems support only BOOLEAN return values. Expected test results should be adjusted accordingly.
T38	Disjoint(g1 Geometry, g2 Geometry) : Integer	For this test, we will determine if the geometry of Route 75 is disjoint from the geometry of Ashton.	1 Some commercial SQL implementations with type extensibility systems support only BOOLEAN return values. Expected test results should be adjusted accordingly.
T39	Touches(g1 Geometry, g2 Geometry) : Integer	For this test, we will determine if the geometry of Cam Stream touches the geometry of Blue Lake.	1 Some commercial SQL implementations with type extensibility systems support only BOOLEAN return values. Expected test results should be adjusted accordingly.

ID	Functionality Tested	Query Description	Answer
T40	Within(g1 Geometry, g2 Geometry) : Integer	For this test, we will determine if the geometry of the house at 215 Main Street is within Ashton.	1 Some commercial SQL implementations with type extensibility systems support only BOOLEAN return values. Expected test results should be adjusted accordingly.
T41	Overlaps(g1 Geometry, g2 Geometry) : Integer	For this test, we will determine if the geometry of Green Forest overlaps the geometry of Ashton.	1 Some commercial SQL implementations with type extensibility systems support only BOOLEAN return values. Expected test results should be adjusted accordingly.
T42	Crosses(g1 Geometry, g2 Geometry) : Integer	For this test, we will determine if the geometry of road segment 101 crosses the geometry of Route 75.	1 Some commercial SQL implementations with type extensibility systems support only BOOLEAN return values. Expected test results should be adjusted accordingly.
T43	Intersects(g1 Geometry, g2 Geometry) : Integer	For this test, we will determine if the geometry of road segment 101 intersects the geometry of Route 75.	1 Some commercial SQL implementations with type extensibility systems support only BOOLEAN return values. Expected test results should be adjusted accordingly.
T44	Contains(g1 Geometry, g2 Geometry) : Integer	For this test, we will determine if the geometry of Green Forest contains the geometry of Ashton.	0 Some commercial SQL implementations with type extensibility systems support only BOOLEAN return values. Expected test results should be adjusted accordingly.
T45	Relate(g1 Geometry, g2 Geometry, PatternMatrix String) : Integer	For this test, we will determine if the geometry of Green Forest relates to the geometry of Ashton using the pattern "TTTTTTTTT".	1 Some commercial SQL implementations with type extensibility systems support only BOOLEAN return values. Expected test results should be adjusted accordingly.
T46	Distance(g1 Geometry, g2 Geometry) : Double Precision	For this test, we will determine the distance between Cam Bridge and Ashton.	12 (in metres)
T47	Intersection(g1 Geometry, g2 Geometry) : Geometry	For this test, we will determine the intersection between Cam Stream and Blue Lake.	'POINT(52 18)'
T48	Difference(g1 Geometry, g2 Geometry) : Geometry	For this test, we will determine the difference between Ashton and Green Forest.	'POLYGON((56 34, 62 48, 84 48, 84 42, 56 34))' or 'MULTIPOLYGON((56 34, 62 48, 84 48, 84 42, 56 34))'[c]

ID	Functionality Tested	Query Description	Answer
T49	Union(g1 Geometry, g2 Geometry) : Integer	For this test, we will determine the union of Blue Lake and Goose Island.	'POLYGON((52 18,66 23,73 9,48 6,52 18))' or 'MULTIPOLYGON((52 18,66 23,73 9,48 6,52 18))' [c]
T50	SymDifference(g1 Geometry, g2 Geometry) : Integer	For this test, we will determine the symmetric difference of Blue Lake and Goose Island.	'POLYGON((52 18,66 23,73 9,48 6,52 18))' or 'MULTIPOLYGON((52 18,66 23,73 9,48 6,52 18))' [c]
T51	Buffer(g Geometry, d Double Precision) : Geometry	For this test, we will make a 15 mbuffer about Cam Bridge.[f]	1 Some commercial SQL implementations with type extensibility systems support only BOOLEAN return values. Expected test results should be adjusted accordingly.
T52	ConvexHull(g Geometry) : Geometry	For this test, we will determine the convex hull of Blue Lake.	'POLYGON((52 18,66 23,73 9,48 6,52 18))' or 'MULTIPOLYGON((52 18,66 23,73 9,48 6,52 18))'[c]

[a] Additional feature tables that are not part of this test will be also be returned if present.

[b] If SRID 101 already exists, or if the system assigns SRID values, appropriate adjustments should be made in the test suite.

[c] Polygon rotation is not defined by this standard; actual polygon rotation may be in a clockwise or counter-clockwise direction.

[d] No specific algorithm is specified for the Centroid function; answers may vary with implementation.

[e] For this test we will have to uses the Contains function (which we don't test until later).

[f] This test counts the number of buildings contained in the buffer that is generated. This test only works because we have a single bridge record, two building records, and we selected the buffer size such that only one of the buildings is contained in the buffer.

C.3.3.1 Geometry types and functions schema construction

```
CREATE TABLE spatial_ref_sys (
     srid       INTEGER NOT NULL PRIMARY KEY,
     auth_name  CHARACTER VARYING,
     auth_srid  INTEGER,
     srtext     CHARACTER VARYING(2048));
-- Lakes
CREATE TABLE lakes (
     fid       INTEGER NOT NULL PRIMARY KEY,
     name      CHARACTER VARYING(64),
     shore     POLYGON);
-- Road Segments
CREATE TABLE road_segments (
     fid       INTEGER NOT NULL PRIMARY KEY,
     name      CHARACTER VARYING(64),
     aliases   CHARACTER VARYING(64),
     num_lanes INTEGER,
     centerlineLINESTRING);
-- Divided Routes
CREATE TABLE divided_routes (
     fid       INTEGER NOT NULL PRIMARY KEY,
     name      CHARACTER VARYING(64),
     num_lanes INTEGER,
     centerlines       MULTILINESTRING);
-- Forests
CREATE TABLE forests (
     fid    INTEGER NOT NULL PRIMARY KEY,
     name   CHARACTER VARYING(64),
     boundary       MULTIPOLYGON);
-- Bridges
CREATE TABLE bridges (
     fid   INTEGER NOT NULL PRIMARY KEY,
     name  CHARACTER VARYING(64),
     position      POINT);
-- Streams
CREATE TABLE streams (
     fid     INTEGER NOT NULL PRIMARY KEY,
     name    CHARACTER VARYING(64),
     centerline      LINESTRING);
-- Buildings
CREATE TABLE buildings (
     fid     INTEGER NOT NULL PRIMARY KEY,
     address CHARACTER VARYING(64),
     positionPOINT,
     footprint       POLYGON);
-- Ponds
```

```
CREATE TABLE ponds (
      fid     INTEGER NOT NULL PRIMARY KEY,
      name    CHARACTER VARYING(64),
      type    CHARACTER VARYING(64),
      shores  MULTIPOYLGON);
-- Named Places
CREATE TABLE named_places (
      fid     INTEGER NOT NULL PRIMARY KEY,
      name    CHARACTER VARYING(64),
      boundaryPOLYGON);
-- Map Neatline
CREATE TABLE map_neatlines (
      fid     INTEGER NOT NULL PRIMARY KEY,
      neatlinePOLYGON);
```

C.3.3.2 Geometry types and functions schema data loading

```
-- Spatial Reference System
INSERT INTO spatial_ref_sys VALUES
      (101, 'POSC', 32214, 'PROJCS["UTM_ZONE_14N",
      GEOGCS["World Geodetic System 72",
      DATUM["WGS_72",
      ELLIPSOID["NWL_10D", 6378135, 298.26]],
      PRIMEM["Greenwich", 0],
      UNIT["Meter", 1.0]],
      PROJECTION["Transverse_Mercator"],
      PARAMETER["False_Easting", 500000.0],
      PARAMETER["False_Northing", 0.0],
      PARAMETER["Central_Meridian", -99.0],
      PARAMETER["Scale_Factor", 0.9996],
      PARAMETER["Latitude_of_origin", 0.0],
      UNIT["Meter", 1.0]]');
-- Lakes
INSERT INTO lakes VALUES (
      101, 'BLUE LAKE',
      PolyFromText(
      'POLYGON(
              (52 18,66 23,73 9,48 6,52 18),
              (59 18,67 18,67 13,59 13,59 18)
           )',
           101));
-- Road segments
INSERT INTO road_segments VALUES(102, 'Route 5', NULL, 2,
      LineFromText(
      'LINESTRING( 0 18, 10 21, 16 23, 28 26, 44 31 )' ,101));
```

```
INSERT INTO road_segments VALUES(103, 'Route 5', 'Main Street', 4,
      LineFromText(
      'LINESTRING( 44 31, 56 34, 70 38 )' ,101));
INSERT INTO road_segments VALUES(104, 'Route 5', NULL, 2,
      LineFromText(
      'LINESTRING( 70 38, 72 48 )' ,101));
INSERT INTO road_segments VALUES(105, 'Main Street', NULL, 4,
      LineFromText(
      'LINESTRING( 70 38, 84 42 )' ,101));
INSERT INTO road_segments VALUES(106, 'Dirt Road by Green Forest', NULL,
      1,
      LineFromText(
      'LINESTRING( 28 26, 28 0 )',101));
-- DividedRoutes
INSERT INTO divided_routes VALUES(119, 'Route 75', 4,
      MLineFromText(
      'MULTILINESTRING((10 48,10 21,10 0),
      (16 0,16 23,16 48))', 101));
-- Forests
INSERT INTO forests VALUES(109, 'Green Forest',
      MPolyFromText(
      'MULTIPOLYGON(((28 26,28 0,84 0,84 42,28 26),
      (52 18,66 23,73 9,48 6,52 18)),((59 18,67 18,67 13,59 13,59 18)))',
      101));
-- Bridges
INSERT INTO bridges VALUES(110, 'Cam Bridge', PointFromText(
      'POINT( 44 31 )', 101));
-- Streams
INSERT INTO streams VALUES(111, 'Cam Stream',
      LineFromText(
      'LINESTRING( 38 48, 44 41, 41 36, 44 31, 52 18 )', 101));
INSERT INTO streams VALUES(112, NULL,
      LineFromText(
      'LINESTRING( 76 0, 78 4, 73 9 )', 101));
-- Buildings
INSERT INTO buildings VALUES(113, '123 Main Street',
      PointFromText(
      'POINT( 52 30 )', 101),
      PolyFromText(
      'POLYGON( ( 50 31, 54 31, 54 29, 50 29, 50 31) )', 101));
INSERT INTO buildings VALUES(114, '215 Main Street',
      PointFromText(
      'POINT( 64 33 )', 101),
      PolyFromText(
      'POLYGON( ( 66 34, 62 34, 62 32, 66 32, 66 34) )', 101));
-- Ponds
```

```
INSERT INTO ponds VALUES(120, NULL, 'Stock Pond',
     MPolyFromText(
     'MULTIPOLYGON( ( ( 24 44, 22 42, 24 40, 24 44) ),
     ( ( 26 44, 26 40, 28 42, 26 44) ) )', 101));
-- Named Places
INSERT INTO named_places VALUES(117, 'Ashton',
     PolyFromText(
     'POLYGON( ( 62 48, 84 48, 84 30, 56 30, 56 34, 62 48) )', 101));
INSERT INTO named_places VALUES(118, 'Goose Island',
     PolyFromText(
     'POLYGON( ( 67 13, 67 18, 59 18, 59 13, 67 13) )', 101));
-- Map Neatlines
INSERT INTO map_neatlines VALUES(115,
     PolyFromText(
     'POLYGON( ( 0 0, 0 48, 84 48, 84 0, 0 0 ) )', 101));
```

C.3.3.3 Geometry types and functions schema test queries

```
-- Conformance Item T1
SELECT f_table_name
     FROM geometry_columns;
-- Conformance Item T2
SELECT f_geometry_column
     FROM geometry_columns
     WHERE f_table_name = 'streams';
-- Conformance Item T3
SELECT coord_dimension
     FROM geometry_columns
     WHERE f_table_name = 'streams';
-- Conformance Item T4
SELECT srid
     FROM geometry_columns
     WHERE f_table_name = 'streams';
-- Conformance Item T5
SELECT srtext
     FROM SPATIAL_REF_SYS
     WHERE SRID = 101;
-- Conformance Item T6
SELECT Dimension(shore)
     FROM lakes
     WHERE name = 'Blue Lake';
-- Conformance Item T7
SELECT GeometryType(centerlines)
     FROM lakes
     WHERE name = 'Route 75';
```

```
-- Conformance Item T8
SELECT AsText(boundary)
      FROM named_places
      WHERE name = 'Goose Island';
-- Conformance Item T9
SELECT AsText(PolyFromWKB(AsBinary(boundary),101))
      FROM named_places
      WHERE name = 'Goose Island';
-- Conformance Item T10
SELECT SRID(boundary)
      FROM named_places
      WHERE name = 'Goose Island';
-- Conformance Item T11
SELECT IsEmpty(centerline)
      FROM road_segments
      WHERE name = 'Route 5'
            AND aliases = 'Main Street';
-- Conformance Item T12
SELECT IsSimple(shore)
      FROM lakes
      WHERE name = 'Blue Lake';
-- Conformance Item T13
SELECT AsText(Boundary((boundary),101)
      FROM named_places
      WHERE name = 'Goose Island';
-- Conformance Item T14
SELECT AsText(Envelope((boundary),101)
      FROM named_places
      WHERE name = 'Goose Island';
-- Conformance Item T15
SELECT X(position)
      FROM bridges
      WHERE name = ‘Cam Bridge’;
-- Conformance Item T16
SELECT Y(position)
      FROM bridges
      WHERE name = 'Cam Bridge';
-- Conformance Item T17
SELECT AsText(StartPoint(centerline))
      FROM road_segments
      WHERE fid = 102;
-- Conformance Item T18
SELECT AsText(EndPoint(centerline))
      FROM road_segments
      WHERE fid = 102;
-- Conformance Item T19
SELECT IsClosed(LineFromWKB(AsBinary(Boundary(boundary)),SRID(boundary)))
      FROM named_places
      WHERE name = 'Goose Island';
```



```
-- Conformance Item T20
SELECT IsRing(LineFromWKB(AsBinary(Boundary(boundary)),SRID(boundary)))
      FROM named_places
      WHERE name = 'Goose Island';
-- Conformance Item T21
SELECT Length(centerline)
      FROM road_segments
      WHERE fid = 106;
-- Conformance Item T22
SELECT NumPoints(centerline)
      FROM road_segments
      WHERE fid = 102;
-- Conformance Item T23
SELECT AsText(PointN(centerline, 1))
      FROM road_segments
      WHERE fid = 102;
-- Conformance Item T24
SELECT AsText(Centroid(boundary))
      FROM named_places
      WHERE name = 'Goose Island';
-- Conformance Item T25
SELECT Contains(boundary, PointOnSurface(boundary))
      FROM named_places
      WHERE name = 'Goose Island';
-- Conformance Item T26
SELECT Area(boundary)
      FROM named_places
      WHERE name = 'Goose Island';
-- Conformance Item T27
SELECT AsText(ExteriorRing(shore))
      FROM lakes
      WHERE name = 'Blue Lake';
-- Conformance Item T28
SELECT NumInteriorRing(shore)
      FROM lakes
      WHERE name = 'Blue Lake';
-- Conformance Item T29
SELECT AsText(InteriorRingN(shore, 1))
      FROM lakes
      WHERE name = 'Blue Lake';
-- Conformance Item T30
SELECT NumGeometries(centerlines)
      FROM divided_routes
      WHERE name = 'Route 75';
-- Conformance Item T31
```

```
SELECT AsText(GeometryN(centerlines, 2))
      FROM divided_routes
      WHERE name = 'Route 75';
-- Conformance Item T32
SELECT IsClosed(centerlines)
      FROM divided_routes
      WHERE name = 'Route 75';
-- Conformance Item T33
SELECT Length(centerlines)
      FROM divided_routes
      WHERE name = 'Route 75';
-- Conformance Item T34
SELECT AsText(Centroid(shores))
      FROM ponds
      WHERE fid = 120;
-- Conformance Item T35
SELECT Contains(shores, PointOnSurface(shores))
      FROM ponds
      WHERE fid = 120;
-- Conformance Item T36
SELECT Area(shores)
      FROM ponds
      WHERE fid = 120;
-- Conformance Item T37
SELECT Equals(boundary,
      PolyFromText('POLYGON( ( 67 13, 67 18, 59 18, 59 13, 67 13) )',1))
      FROM named_places
      WHERE name = 'Goose Island';
-- Conformance Item T38
SELECT Disjoint(centerlines, boundary)
      FROM divided_routes, named_places
      WHERE divided_routes.name = 'Route 75'
            AND named_places.name = 'Ashton';
-- Conformance Item T39
SELECT Touches(centerline, shore)
      FROM streams, lakes
      WHERE streams.name = 'Cam Stream'
            AND lakes.name = 'Blue Lake';
-- Conformance Item T40
SELECT Within(boundary, footprint)
      FROM named_places, buildings
      WHERE named_places.name = 'Ashton'
            AND buildings.address = '215 Main Street';
-- Conformance Item T41
SELECT Overlaps(forests.boundary, named_places.boundary)
      FROM forests, named_places
      WHERE forests.name = 'Green Forest'
            AND named_places.name = 'Ashton';
-- Conformance Item T42
```

```
SELECT Crosses(road_segments.centerline, divided_routes.centerlines)
     FROM road_segments, divided_routes
     WHERE road_segment.fid = 102
          AND divided_routes.name = 'Route 75';
-- Conformance Item T43
SELECT Intersects(road_segments.centerline, divided_routes.centerlines)
     FROM road_segments, divided_routes
     WHERE road_segments.fid = 102
          AND divided_routes.name = 'Route 75';
-- Conformance Item T44
SELECT Contains(forests.boundary, named_places.boundary)
     FROM forests, named_places
     WHERE forests.name = 'Green Forest'
          AND named_places.name = 'Ashton';
-- Conformance Item T45
SELECT Relate(forests.boundary, named_places.boundary, 'TTTTTTTTT')
     FROM forests, named_places
     WHERE forests.name = 'Green Forest'
          AND named_places.name = 'Ashton';
-- Conformance Item T46
SELECT Distance(position, boundary)
     FROM bridges, named_places
     WHERE bridges.name = 'Cam Bridge'
          AND named_places.name = 'Ashton';
-- Conformance Item T47
SELECT AsText(Intersection(centerline, shore))
     FROM streams, lakes
     WHERE streams.name = 'Cam Stream'
          AND lakes.name = 'Blue Lake';
-- Conformance Item T48
SELECT AsText(Difference(named_places.boundary, forests.boundary))
     FROM named_places, forests
     WHERE named_places.name = 'Ashton'
          AND forests.name = 'Green Forest';
-- Conformance Item T49
SELECT AsText(Union(shore, boundary))
     FROM lakes, named_places
     WHERE lakes.name = 'Blue Lake'
          AND named_places.name = ‘Goose Island’;
-- Conformance Item T50
SELECT AsText(SymDifference(shore, boundary))
     FROM lakes, named_places
     WHERE lakes.name = 'Blue Lake'
          AND named_places.name = 'Ashton';
-- Conformance Item T51
```

```
SELECT count(*)
      FROM buildings, bridges
      WHERE Contains(Buffer(bridges.position, 15.0), buildings.footprint)
      = 1;
-- Conformance Item T52
SELECT AsText(ConvexHull(shore))
      FROM lakes
      WHERE lakes.name = 'Blue Lake';
```