Open Geospatial Consortium Inc.

Date: 2005-11-22

Reference number of this document: OGC 05-134

Version: 1.1.0

Category: OpenGIS® Implementation Specification

Editor: Keith Ryden

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

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

License Agreement

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD.

THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable

Contents

Page

Foreword

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

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

— *Part 1: Common architecture*

— *Part 2: SQL option*

Part 3: COM/OLE option is under preparation.

Introduction

This 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 2D geometry with linear interpolation between vertices. This part of ISO 19125 is dependent on the common architectural components defined in OGC 05-126 and ISO 19125-1.

Feature collections are stored as tables with geometry valued columns in a SQL-implementation; each feature is a row in the table. The non-spatial attributes of features are mapped onto columns whose types are drawn from the set of standard SQL data types. The spatial attributes of features are mapped onto columns whose SQL data types are based on the underlying concept of additional geometric data types for SQL. A table whose rows represent these features is referred to as a feature table. Such a table contains one or more geometry valued columns. Feature-table schemas are described for two SQL-implementations: implementations based on predefined data types and SQL with Geometry Types.

In an implementation based on predefined data types, a geometry-valued column is implemented as a Foreign Key reference into a geometry table. A geometry value is stored using one or more rows in the geometry table. The geometry table may be implemented using either standard SQL numeric types or SQL binary types; schemas for both are described.

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.

Geographic information — Simple feature access —

Part 2:
SQL option

1 Scope

This part of ISO 19125 specifies an SQL schema that supports storage, retrieval, query and update of simple geospatial feature collections via the SQL Call Level Interface (SQL/CLI) (ISO/IEC 9075-3:2003).

This part of ISO 19125 establishes an architecture for the implementation of feature tables.

This part of ISO 19125 defines terms to use within the architecture.

This part of ISO 19125 defines a simple feature profile of ISO 19107.

This part of ISO 19125 describes a set of SQL Geometry Types together with SQL functions on those types. The Geometry Types and Functions described in this part of ISO 19125 represent a profile of ISO 13249-3.

This part of ISO 19125 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 part of ISO 19125 does standardize:

— names and geometric definitions of the SQL Types for Geometry;

— names, signatures and geometric definitions of the SQL Functions for Geometry.

This part of ISO 19125 describes a feature access implementation in SQL based on a profile of ISO 19107. ISO 19107 does not place any requirements on how to define the Geometry Types in the internal 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 manner not mentioned in this part of ISO 19125.

2 Conformance

In order to conform to this part of ISO 19125, an implementation shall satisfy the requirements of one of the following three conformance classes, as well as the appropriate components of ISO 19125-1:

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 part of ISO 19125.

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.

ISO/IEC 9075-1:2003, *Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework)*

ISO/IEC 9075-2:2003, *Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation)*

ISO/IEC 9075-3:2003, *Information technology — Database languages — SQL — Part 3: Call-Level Interface (SQL/CLI)*

ISO/IEC 9075-4:2003, *Information technology — Database languages — SQL — Part 4: Persistent Stored Modules (SQL/PSM)*

ISO/IEC 9075-5:1999, *Information technology — Database languages — SQL — Part 5: Host Language Bindings (SQL/Bindings)*

ISO/IEC 13249-3:2003, *Information technology — Database languages — SQL multimedia and application packages — Part 3: Spatial*

ISO 19107:2003, *Geographic information — Spatial schema*

ISO 19109:—[1)], *Geographic information — Rules for application schema*

ISO 19119:2004, *Geographic information — Services*

ISO 19125-1:2004, *Geographic information — Simple feature access — Part 1: Common architecture*

4 Terms and definitions

For the purposes of this part of ISO 19125, the following terms and definitions apply.

---

1) To be published.

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

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
SRID	Spatial Reference System Identifier
SRTEXT	Spatial Reference System Well Known Text
WKB	Well-Known Binary (representation for example, geometry)
WKTR	Well-Known Text Representation
2D	2-Dimensional
$\Re^1$	1-Dimensional space
$\Re^2$	2-Dimensional space
$\varnothing$	empty set
$\cap$	intersection
$\cup$	union
—	difference
$\in$	is a member of
$\notin$	is not a member of
$\subset$	is a proper subset of
$\subseteq$	is a subset of
$\Leftrightarrow$	if and only if
$\Rightarrow$	implies
$\forall$	for all
{ X | … }	set of X such that…
$\wedge$	and
$\vee$	or

¬	not
=	equal
≠	not equal
<	less than
>	greater than

6 Architecture

6.1 Architecture — SQL implementation of feature tables based on predefined data types

6.1.1 Overview

This part of ISO 19125 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.

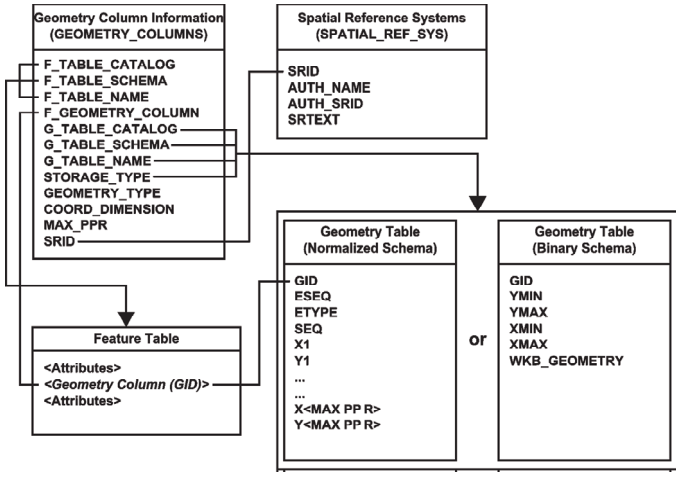


Figure 1 — Schema for feature tables using predefined data types

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 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.

6.1.4 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 and Y 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.

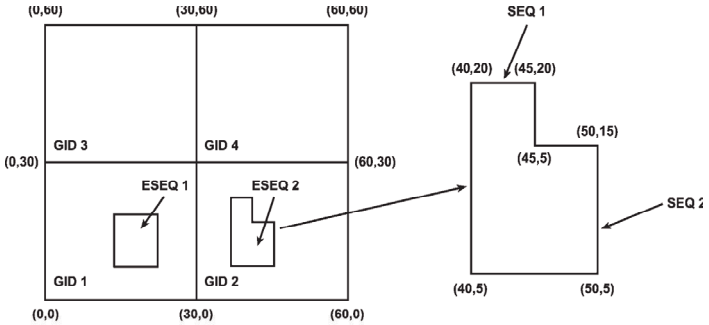
d) Polygons may contain holes, as described in the Geometry object model.

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 the above 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 >

6.1.6 Use of numeric data types

SQL-implementations usually provide several numeric data types. In this part of ISO 19125, 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.

6.1.7 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 with Geometry Types implementation of feature tables

6.2.1 Overview

This part of ISO 19125 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.

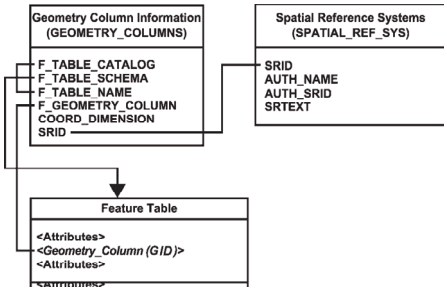


Figure 3 — Schema for feature tables using SQL with Geometry Types

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 specification 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 part of ISO 19125 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 part of ISO 19125 are preserved.

6.2.6 SQL Geometry Type hierarchy

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

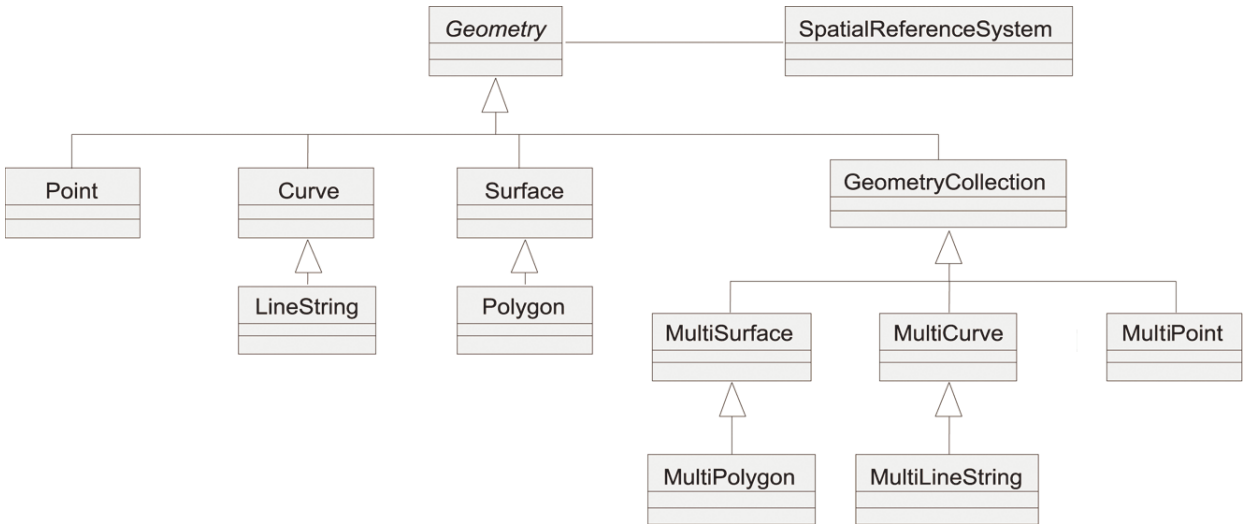


Figure 4 — SQL Geometry Type hierarchy

The root type, named Geometry, has subtypes for Point, Curve, Surface and GeometryCollection. A GeometryCollection is a Geometry that is a collection of possibly heterogeneous geometric objects. MultiPoint, MultiCurve and MultiSurface are specific subtypes of GeometryCollection 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. Capturing this association at the level of the individual geometric object allows literal Geometry values that are not yet part of a column in the database to be associated with a Spatial Reference System. Examples of such a geometric object, Geometry values is a geometric object that is used as a parameter to a spatial query or a geometric object that is part of an insert statement. Capturing this association at the level of the individual geometric object also allows functions that take two geometric objects to check for compatible Spatial Reference Systems.

A Geometry value is associated with a Spatial Reference System by storing the Spatial Reference System Identity (SRID) for the Spatial Reference System as a part of the geometric object. Each Spatial Reference System in the database is identified by a unique value of SRID.

The SRID for a geometric object is assigned to it at construction time. This allows the SQL with Geometry Types implementation to ensure that

a) the geometric object being inserted into a geometry column matches the Spatial Reference System declared for the geometry column;

b) queries that spatially join columns from different tables operate on geometry columns with consistent Spatial Reference Systems.

If either of these conditions is violated, a run-time SQL error is generated. These Spatial Reference System consistency checks are not possible in implementations based on predefined data types.

The SRID function, defined on the Geometry Type, returns the integer SRID of a geometric object.

In all operations on the Geometry Type, geometric calculations shall be done in the Spatial Reference System of the first geometric object. Returned objects shall be in the Spatial Reference System of the first geometric object unless explicitly stated otherwise.

Before a geometric object can be constructed and inserted into a table, the corresponding row for its SRID shall exist in the SPATIAL_REF_SYS table, else construction of the geometric object shall fail. When defining a table, a SQL check constraint can be used to enforce the rule that all geometric objects in a geometry column have the same SRID as that defined for the column in the GEOMETRY_COLUMNS table. The following example shows the definition of a table, named Countries, with two columns named Name and Geometry of type CHARACTER VARYING and POLYGON, respectively.

```
CREATE TABLE Countries (
       Name        CHARACTER VARYING(200) NOT NULL PRIMARY KEY,
       Geometry    Polygon NOT NULL,
       CONSTRAINT spatial_reference
       CHECK (SRID(Geometry) in (SELECT SRID from GEOMETRY_COLUMNS where F_TABLE_CATALOG = <catalog>
and F_TABLE_SCHEMA = <schema> and F_TABLE_NAME = 'Countries' and F_GEOMETRY_COLUMN = 'Geometry'))
)
```

It is expected that most implementations shall use stored procedures similar to those shown below for the purpose of adding and dropping geometry columns to and from a feature table.

The **AddGeometryColumn**(FEATURE_TABLE_CATALOG, FEATURE_TABLE_SCHEMA, FEATURE_TABLE_NAME, GEOMETRY_COLUMN_NAME, SRID) procedure shall

a) ensure that an entry for the SRID exists in the SPATIAL_REF_SYS table;

b) add an entry to the GEOMETRY_COLUMNS table that stores the SRID for the Geometry Column;

c) add the Geometry column to the feature table using a SQL ALTER TABLE statement;

d) add the Spatial Reference Check Constraint to the feature table.

The **DropGeometryColumn**(FEATURE_TABLE_CATALOG, FEATURE_TABLE_SCHEMA, FEATURE_TABLE_NAME, GEOMETRY_COLUMN_NAME) stored procedure shall

a) drop the Spatial Reference Check Constraint on the feature table;

b) drop the entry from the GEOMETRY_COLUMNS table;

c) drop the Geometry Column from the feature table.

6.2.8 SQL/CLI access to Geometry values in the SQL with Geometry Type case

Spatial data are accessed using the SQL query language extended with SQL functions on Geometry Types. The SQL pass-through capabilities of SQL/CLI allow a client to pass these or any extended SQL statements containing SQL-implementation-specific SQL extensions to a server. (Applications are free to send any SQL

statement to an SQL-implementation, even if the statement is not described within the SQL/CLI conformance levels.)

Geometry Columns are implemented using the Geometry Types described above.

GIS applications shall be able to determine the existence of a Geometry Column based on the Geometry Type or one of its subtypes using one or more of the following SQL/CLI programming techniques.

a) The SQLTypeInfo function can be used to determine both the TYPE_NAME and the underlying SQL_DATA_TYPE of an SQL Type.

b) The SQLColumns catalog function can be used to determine the TYPE_NAME and the underlying SQL_DATA_TYPE of a column in a table.

c) The SQLDescribeCol and SQLColAttributes functions can be used to determine a column's data type and description.

An SQL/CLI client application uses either one of two SQL functions:

— **GeomFromText ([in] String, [in] Integer) : Geometry,** or

— **GeomFromWKB([in] Binary, [in] Integer) : Geometry,**

or their type-specific versions (for example, PolygonFromText and PolygonFromWKB) to pass geometric objects into the database from a client application that represents them using either the Well-known Text or the Wellknown Binary representations.

The input arguments to the above functions are SQL/CLI standard character, binary and integer data types (SQL_C_CHAR, SQL_C_BINARY, SQL_C_INTEGER) and clients bind to these parameters using standard SQL/CLI binding methods.

An SQL/CLI client application uses either one of two SQL functions:

— **AsText([in]Geometry) : String**, or

— **AsBinary([in]Geometry) : Binary**

to extract geometry values from the database as either Well-known Text or Binary values.

The output arguments to the above functions are SQL/CLI standard character and binary data types (SQL_C_CHAR, SQL_C_BINARY) and clients bind to these parameters using standard SQL/CLI binding methods.

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.

```
CREATE TABLE SPATIAL_REF_SYS
       (
       SRID               INTEGER NOT NULL PRIMARY KEY,
       AUTH_NAME          CHARACTER VARYING(256),
       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.

7.1.3.2 Table or view constructs

```
CREATE TABLE GEOMETRY_COLUMNS (
       F_TABLE_CATALOG           CHARACTER VARYING(256) NOT NULL,
       F_TABLE_SCHEMA            CHARACTER VARYING(256) NOT NULL,
       F_TABLE_NAME              CHARACTER VARYING(256) NOT NULL,
       F_GEOMETRY_COLUMN         CHARACTER VARYING(256) NOT NULL,
       G_TABLE_CATALOG           CHARACTER VARYING(256) NOT NULL,
```

```
G_TABLE_SCHEMA            CHARACTER VARYING(256) NOT NULL,
G_TABLE_NAME              CHARACTER VARYING(256) NOT NULL,
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_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;

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;

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);

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;

```
0 = GEOMETRY1 = POINT
2 = CURVE
3 = LINESTRING
4 = SURFACE
5 = POLYGON
6 = COLLECTION
7 = MULTIPOINT
8 = MULTICURVE
9 = MULTILINESTRING
10 = MULTISURFACE
11 = MULTIPOLYGON
```

f) `COORD_DIMENSION` — the number of ordinates used in the complex, usually corresponds to the number of dimensions in 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;

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.

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 specification 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, populate and maintain this table.

```
CREATE TABLE <table name> (
       GID                    NUMERIC NOT NULL,
       ESEQ                   INTEGER NOT NULL,
       ETYPE                  INTEGER  NOT NULL,
       SEQ                    INTEGER  NOT NULL,
       X1                     <ordinate type>,
       Y1                     <ordinate type>,
       ... <repeated for each ordinate, repeated for each point>
       X<MAX_PPR>             <ordinate type>,
       Y<MAX_PPR>             <ordinate type>,
       ...,
       <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) ... — (repeated for each ordinate, for this Point);

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

i) `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;

j) `Y<MAX_PPR>` — second ordinate of last Point;

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

l) `<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>,
       XMAX                   <ordinate type>,
       YMAX                   <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) `XMAX` — the maximum *x*-coordinate of the geometric object bounding box;

e) `YMAX` — the maximum *y*-coordinate of the geometric object bounding box;

f) `WKB_GEOMETRY` — the Well-known Binary Representation of the geometric object;

g) `<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 specification.

7.2 Components — SQL with Geometry Types implementation of feature tables

7.2.1 Conventions

The components of this part of ISO 19125 for feature table implementation in a SQL with Geometry Types environment consist of the tables, SQL types and SQL functions discussed in 7.2.

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

7.2.2 Spatial reference system information

7.2.2.1 Component overview

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

This component is identical to the corresponding component described for the implementation based on predefined data types.

7.2.2.2 Table constructs

The following `CREATE TABLE` statement creates an appropriately structured `SPATIAL_REF_SYS` table.

```
CREATE TABLE SPATIAL_REF_SYS
       (
       SRID                INTEGER NOT NULL PRIMARY KEY,
       AUTH_NAME           CHARACTER VARYING(256),
       AUTH_SRID           INTEGER,
       SRTEXT              CHARACTER VARYING(2048)
       )
```

7.2.2.3 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 `AUTH_NAME`;

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

7.2.2.4 Exceptions, errors and error codes

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

7.2.3 Geometry columns information

7.2.3.1 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.

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

7.2.3.2 Table constructs

The following CREATE TABLE statement creates an appropriately structured GEOMETRY_COLUMNS table.

```
CREATE TABLE GEOMETRY_COLUMNS (
       F_TABLE_CATALOG        CHARACTER VARYING(256) NOT NULL,
       F_TABLE_SCHEMA         CHARACTER VARYING(256) NOT NULL,
       F_TABLE_NAME           CHARACTER VARYING(256) NOT NULL,
       F_GEOMETRY_COLUMN      CHARACTER VARYING(256) NOT NULL,
       COORD_DIMENSION        INTEGER,
       SRID                   INTEGER REFERENCES SPATIAL_REF_SYS,
       CONSTRAINT GC_PK PRIMARY KEY
                 (F_TABLE_CATALOG, F_TABLE_SCHEMA, F_TABLE_NAME, F_GEOMETRY_COLUMN)
       )
```

7.2.3.3 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 Geometry Column in the feature table;

c) COORD_DIMENSION — the coordinate dimension for the geometric object in this column, which shall be equal to the number of dimensions in the spatial reference system;

d) 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.

7.2.3.4 Exceptions, errors and error codes

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

7.2.4 SQL Geometry Types

7.2.4.1 Component overview

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

7.2.4.2 Language constructs

The SQL language shall support a subset of the following set of SQL Geometry Types: {Geometry, Point, Curve, LineString, Surface, Polygon, GeometryCollection, MultiCurve, MultiLineString, MultiSurface, MultiPolygon, MultiPoint}. The permissible type subsets that an implementer may choose to implement are described in Table 2.

An implementation shall preserve the subtype relationships between Geometry Types shown in Figure 4 for the types that are implemented. An implementation that implements two types A and B, where B is an immediate subtype of A in Figure 4 and is free to introduce additional types C, is outside the scope of this specification.

Geometry, Curve, Surface, MultiCurve and MultiSurface are defined to be non-instantiable types. No constructors are defined for these types.

An implementation in SQL will use the name GeomCollection instead of GeometryCollection to ensure alignment with the ISO/IEC 13249-3 specification.

The remaining seven types are defined to be instantiable. An implementation may support only a subset of these seven types as instantiable, as defined in Table 2.

Table 2 — Available and instantiable types by implementation type level

Type level	Available types	Instantiable types
1	Geometry, Point, Curve, LineString, Surface, Polygon, GeomCollection	Point, LineString, Polygon, GeomCollection
2	Geometry, Point, Curve, LineString, Surface, Polygon, GeomCollection, MultiPoint, MultiCurve, MultiLineString, MultiSurface, MultiPolygon	Point, LineString, Polygon, MultiPoint, MultiLineString,MultiPolygon
3	Geometry, Point, Curve, LineString, Surface, Polygon, GeomCollection, MultiPoint , MultiCurve, MultiLineString, MultiSurface, MultiPolygon	Point, LineString, Polygon, GeomCollection, MultiPoint, MultiLineString, MultiPolygon

7.2.5 Feature tables

7.2.5.1 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 part of ISO 19125 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.

7.2.5.2 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 a SQL Geometry Type for one 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.

7.2.5.3 Exceptions, errors and error codes

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

7.2.6 SQL functions for constructing a geometry value given its Well-known Text Representation

7.2.6.1 Component overview

The functions used to construct geometric objects from their text representations are shown in Table 3.

The `GeomFromText` function takes a geometry textual representation (a `<Geometry Tagged Text>`, as described in ISO 19125-1, 6.2), and a Spatial Reference System ID (`SRID`) and creates an instance of the appropriate Geometry Type.

The return type of the `GeomFromText` function is the `Geometry` supertype. For construction of a geometric object to be stored in columns restricted to a particular subtype, an implementation shall also provide a type-specific construction function for each instantiable subtype, as described in Table 3.

An implementation may substitute any SQL type suitable for representing text data such as `CHARACTER VARYING` for the type `String` below.

Table 3 — SQL functions for constructing a geometric object given its Well-known Text Representation

Function	Description
`GeomFromText( geometryTaggedText String, SRID Integer) : Geometry`	construct a geometric object given its Well-known text Representation
`PointFromText ( pointTaggedText String, SRID Integer): Point`	construct a Point
`LineFromText( lineStringTaggedText String, SRID Integer) : LineString`	construct a LineString
`PolyFromText( polygonTaggedText String, SRID Integer): Polygon`	construct a Polygon
`MPointFromText (multiPointTaggedText String, SRID Integer): MultiPoint`	construct a MultiPoint
`MLineFromText ( multiLineStringTaggedText String, SRID Integer): MultiLineString`	construct a MultiLineString
`MPolyFromText( multiPolygonTaggedText String, SRID Integer): MultiPolygon`	construct a MultiPolygon
`GeomCollFromText( geometryCollectionTaggedText String, SRID Integer): GeomCollection`	construct a GeometryCollection

As an optional feature, an implementation may also support "building" `Polygon` or `MultiPolygon` values given an arbitrary collection of possibly intersecting `Rings` or closed `LineString` values. Implementations that support this feature should include the functions shown in Table 4.

Table 4 — Optional SQL functions for constructing a geometric object given its Well-known Text Representation

Function	Description
`BdPolyFromText( multiLineStringTaggedText String, SRID Integer): Polygon`	construct a Polygon given an arbitrary collection of closed linestrings as a MultiLineString text representation
`BdMPolyFromText( multiLineStringTaggedText String, SRID Integer): MultiPolygon`	construct a MultiPolygon given an arbitrary collection of closed linestrings as a MultiLineString text representation

7.2.6.2 Exceptions, errors and error codes

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

7.2.6.3 Example

The following example shows the use of the `Polygon` type specific constructor:

```
INSERT INTO Countries (Name, Location)
       VALUES ('Kenya', PolyFromText('POLYGON ((x y, x y, x y, ...,  x y))', 14))
```

7.2.7 SQL functions for constructing a geometric object given its Well-known Binary Representation

7.2.7.1 Component overview

The functions used to construct geometric objects from their Well-known Binary Representations are shown in Table 5.

The `GeomFromWKB` function takes a Well-known Binary Representation of Geometry (a `<WKBGeometry>` as described in ISO 19125-1, 6.3) and a Spatial Reference System ID (`SRID`) and creates an instance of the appropriate Geometry Type.

The return type of the `GeomFromWKB` function is the `Geometry` supertype. For construction of geometric objects to be stored in columns restricted to a particular subtype, an implementation shall also provide a type specific construction function for each instantiable subtype as described in Table 5.

An implementation may substitute any SQL type used to represent binary values for the type `BINARY` in the definitions below.

Table 5 — SQL functions for constructing a geometric object given its Well-known Binary Representation

Function	Description
`GeomFromWKB (WKBGeometry Binary, SRID Integer) : Geometry`	construct a geometric object given its Well-known Binary Representation
`PointFromWKB (WKBPoint BINARY, SRID Integer): Point`	construct a Point
`LineFromWKB(WKBLineString BINARY, SRID Integer) : LineString`	construct a LineString
`PolyFromWKB(WKBPolygon BINARY, SRID Integer): Polygon`	construct a Polygon
`MPointFromWKB (WKBMultiPoint BINARY, SRID Integer): MultiPoint`	construct a MultiPoint
`MLineFromWKB (WKBMultiLineString BINARY, SRID Integer): MultiLineString`	construct a MultiLineString
`MPolyFromWKB (WKBMultiPolygon BINARY, SRID Integer): MultiPolygon`	construct a MultiPolygon
`GeomCollFromWKB (WKBGeomCollection BINARY, SRID Integer): GeomCollection`	construct GeometryCollection

As an optional feature, an implementation may also support "building" `Polygon` or `MultiPolygon` values given an arbitrary collection of possibly intersecting `Rings` or closed `LineString` values. Implementations that support this feature shall include the functions shown in Table 6.

Table 6 — Optional SQL functions for constructing a geometric object given its Well-known Binary Representation

Function	Description
`BdPolyFromWKB(WKBMultiLineString BINARY, SRID Integer): Polygon`	construct a Polygon given an arbitrary collection of closed linestrings as a MultiLineString binary representation
`BdMPolyFromWKB( WKBMultiLineString BINARY, SRID Integer): MultiPolygon`	construct a MultiPolygon given an arbitrary collection of closed linestrings as a MultiLineString binary representation

7.2.7.2 Exceptions, errors and error codes

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

7.2.7.3 Examples

The following example shows the use of the binary `Polygon` type-specific constructor in Dynamic SQL, the `:wkb` and `:srid` parameters are bound to application program variables containing the binary representation of a `Polygon` and of the `SRID`, respectively:

```
INSERT INTO Countries (Name, Location)
       VALUES ('Kenya', PolyFromWKB(:wkb, :srid))
```

7.2.8 SQL functions for obtaining Well-known Text Representation of a geometric object

7.2.8.1 Component overview

The `AsText` function, shown in Table 7, takes a single argument of type `Geometry` and returns its Well-known Text Representation. This function applies to all subtypes of `Geometry`.

Table 7 — SQL functions for obtaining the Well-known Text Representation of a geometric object

Function	Description
`AsText (g Geometry) : String`	returns the Well-known Text representation

7.2.8.2 Exceptions, errors and error codes

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

7.2.8.3 Examples

The following example shows the use of the `AsText` function to extract the name and textual representation of Geometry of all countries whose names begin with the letter K.

```
SELECT Name, AsText(Location) FROM Countries WHERE Name LIKE 'K%'
```

7.2.9 SQL functions for obtaining Well-known Binary Representations of a geometric object

7.2.9.1 Component overview

The `AsBinary` function, shown in Table 8, takes a single argument of type `Geometry` and returns its Well-known Binary Representation. This function applies to all subtypes of `Geometry`.

Table 8 — SQL functions for obtaining the Well-known Binary Representation of a geometric object

Function	Description
`AsBinary (g Geometry) : Binary`	returns the Well-known Binary Representation

7.2.9.2 Exceptions, errors and error codes

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

7.2.9.3 Example

The following example shows the use of the `AsBinary` function to extract the name and Well-known Binary representation of Geometry for all countries whose names begin with the letter K.

```
SELECT Name, AsBinary(Location) FROM Countries WHERE Name LIKE 'K%'
```

7.2.10 Functions on type Geometry

7.2.10.1 Component overview

The SQL functions shown in Table 9 apply to all subtypes of `Geometry`.

In all operations on the Geometry Type, geometric calculations shall be done in the Spatial Reference System of the first geometric object. Returned objects shall be in the Spatial Reference System of the first geometric object unless explicitly stated otherwise.

Table 9 — SQL functions on type Geometry

Function	Description
`Dimension(g Geometry) : Integer`	returns the dimension of the geometric object, which is less than or equal to the dimension of the coordinate space
`GeometryType(g Geometry) : String`	returns the name of the instantiable subtype of Geometry of which this geometric object is a member, as a string
`AsText(g Geometry) : String`	returns the Well-known Text Representation of this geometric object
`AsBinary(g Geometry) : Binary`	returns the Well-known Binary Representation of this geometric object
`SRID(g Geometry) : Integer`	returns the Spatial Reference System ID for this geometric object
`IsEmpty(g Geometry) : Integer`	The return type is Integer, with a return value of 1 for TRUE, 0 for FALSE, and –1 for UNKNOWN corresponding to a function invocation on NULL arguments. TRUE if this geometric object corresponds to the empty set
`IsSimple(g Geometry): Integer`	The return type is Integer, with a return value of 1 for TRUE, 0 for FALSE, and –1 for UNKNOWN corresponding to a function invocation on NULL arguments. TRUE if this geometric object is simple, as defined in the Geometry Model
`Boundary(g Geometry) : Geometry`	returns a geometric object that is the combinatorial boundary of g as defined in the Geometry Model
`Envelope(g Geometry) : Geometry`	returns the rectangle bounding g as a Polygon. The Polygon is defined by the corner points of the bounding box [(MINX, MINY),(MAXX, MINY), (MAXX, MAXY), (MINX, MAXY), (MINX, MINY)].

7.2.10.2 Exceptions, errors and error codes

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

7.2.11 SQL functions on type Point

7.2.11.1 Component overview

The SQL functions defined on `Point` are shown in Table 10.

Table 10 — SQL functions on type Point

Function	Description
`X(p Point) : Double Precision`	return the *x*-coordinate of Point p as a double precision number
`Y(p Point) : Double Precision`	return the *y*-coordinate of Point p as a double precision number

7.2.11.2 Exceptions, errors and error codes

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

7.2.12 Functions on type Curve

7.2.12.1 Component overview

The SQL functions defined on `Curve` are shown in Table 11.

Table 11 — SQL functions on type Curve

Function	Description
`StartPoint(c Curve) : Point`	return a Point containing the first Point of c
`EndPoint(c Curve) : Point`	return a Point containing the last Point of c
`IsClosed(c Curve) : Integer`	The return type is Integer, with a return value of 1 for TRUE, 0 for FALSE, and –1 for UNKNOWN corresponding to a function invocation on NULL arguments; return TRUE if c is closed, i.e., if StartPoint(c) = EndPoint(c)
`IsRing(c Curve) : Integer`	The return type is Integer, with a return value of 1 for TRUE, 0 for FALSE, and –1 for UNKNOWN corresponding to a function invocation on NULL arguments; return TRUE if c is a ring, i.e., if c is closed and simple. A simple Curve does not pass through the same Point more than once.
`Length(c Curve) : Double Precision`	return the length of c

7.2.12.2 Exceptions, errors and error codes

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

7.2.13 SQL functions on type LineString

7.2.13.1 Component overview

The SQL functions defined on `LineString` are shown in Table 12.

Table 12 — SQL functions on type LineString

Function	Description
`NumPoints(l LineString) : Integer`	return the number of Points in the LineString
`PointN(l LineString, n Integer) : Point`	return a Point containing Point n of l

7.2.13.2 Exceptions, errors and error codes

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

7.2.14 SQL functions on type Surface

7.2.14.1 Component overview

The SQL functions defined on `Surface` are shown in Table 13.

Table 13 — SQL functions on type Surface

Function	Description
`Centroid(s Surface) : Point`	return the centroid of s, which may lie outside s
`PointOnSurface(s Surface) : Point`	return a Point guaranteed to lie on the Surface
`Area(s Surface) : Double Precision`	return the area of s

7.2.14.2 Exceptions, errors and error codes

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

7.2.15 SQL functions on type Polygon

7.2.15.1 Component overview

The SQL functions defined on `Polygon` are shown in Table 14.

Table 14 — SQL functions on type Polygon

Function	Description
`ExteriorRing(p Polygon) : LineString`	return the exteriorRing of p
`NumInteriorRing(p Polygon) : Integer`	return the number of interiorRings
`InteriorRingN(p Polygon, n Integer) : LineString`	return the nth interiorRing. The order of Rings is not geometrically significant.

7.2.15.2 Exceptions, errors and error codes

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

7.2.16 SQL functions on type GeomCollection

7.2.16.1 Component overview

The SQL functions defined on `GeomCollection` are shown in Table 15.

Table 15 — SQL functions on type GeomCollection

Function	Description
`NumGeometries(g GeomCollection) : Integer`	return the number of geometric objects in the collection
`GeometryN(g GeomCollection, n Integer) : Geometry`	return the nth geometric object in the collection. The order of the elements in the collection is not geometrically significant.

7.2.16.2 Exceptions, errors and error codes

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

7.2.17 SQL functions on type MultiCurve

7.2.17.1 Component overview

The SQL functions defined on `MultiCurve` are shown in Table 16.

Table 16 — SQL functions on type MultiCurve

Function	Description
`IsClosed(mc MultiCurve) : Integer`	The return type is Integer, with a return value of 1 for TRUE, 0 for FALSE, and –1 for UNKNOWN corresponding to a function invocation on NULL arguments; return TRUE if mc is closed
`Length(mc MultiCurve) : Double Precision`	return the length of mc

7.2.17.2 Exceptions, errors and error codes

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

7.2.18 SQL functions on type MultiSurface

7.2.18.1 Component overview

The SQL functions defined on `MultiSurface` are shown in Table 17.

Table 17 — SQL functions on type MultiSurface

Function	Description
`Centroid(ms MultiSurface) : Point`	return the centroid of ms, which may lie outside ms
`PointOnSurface(ms MultiSurface) : Point`	return a Point guaranteed to lie on the MultiSurface
`Area(ms MultiSurface) : Double Precision`	return the area of ms

7.2.18.2 Exceptions, errors and error codes

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

7.2.19 SQL functions that test spatial relationships

7.2.19.1 Component overview

The functions shown in Table 18 test named spatial relationships between two geometric objects. The Relate function tests if the specified spatial relationship between two geometric objects exists, where the spatial relationship is expressed as a string encoding the acceptable values for the DE-9IM between the two geometric objects.

Table 18 — SQL functions that test spatial relationships

Function	Description
`Equals(g1 Geometry,g2 Geometry) : Integer`	The return type is Integer, with a return value of 1 for TRUE, 0 for FALSE, and –1 for UNKNOWN corresponding to a function invocation on NULL arguments. TRUE if g1 and g2 are equal
`Disjoint(g1 Geometry, g2 Geometry) : Integer`	The return type is Integer, with a return value of 1 for TRUE, 0 for FALSE, and –1 for UNKNOWN corresponding to a function invocation on NULL arguments. TRUE if the intersection of g1 and g2 is the empty set
`Touches(g1 Geometry, g2 Geometry) : Integer`	The return type is Integer, with a return value of 1 for TRUE, 0 for FALSE, and –1 for UNKNOWN corresponding to a function invocation on NULL arguments. TRUE if the only Points in common between g1 and g2 lie in the union of the boundaries of g1 and g2
`Within(g1 Geometry, g2 Geometry) : Integer`	The return type is Integer, with a return value of 1 for TRUE, 0 for FALSE, and –1 for UNKNOWN corresponding to a function invocation on NULL arguments. TRUE if g1 is completely contained in g2
`Overlaps(g1 Geometry, g2 Geometry) : Integer`	The return type is Integer, with a return value of 1 for TRUE, 0 for FALSE, and –1 for UNKNOWN corresponding to a function invocation on NULL arguments. TRUE if the intersection of g1 and g2 results in a value of the same dimension as g1 and g2 that is different from both g1 and g2

`Crosses(g1 Geometry, g2 Geometry) : Integer`	The return type is Integer, with a return value of 1 for TRUE, 0 for FALSE, and –1 for UNKNOWN corresponding to a function invocation on NULL arguments. TRUE if the intersection of g1 and g2 results in a value whose dimension is less than the maximum dimension of g1 and g2 and the intersection value includes Points interior to both g1 and g2, and the intersection value is not equal to either g1 or g2
`Intersects(g1 Geometry, g2 Geometry) : Integer`	The return type is Integer, with a return value of 1 for TRUE, 0 for FALSE, and –1 for UNKNOWN corresponding to a function invocation on NULL arguments; convenience predicate: TRUE if the intersection of g1 and g2 is not empty *Intersects(g1, g2) ⇔ Not (Disjoint(g1, g2))*

Table 18 (*continued*)

Function	Description
`Contains(g1 Geometry, g2 Geometry) : Integer`	The return type is Integer, with a return value of 1 for TRUE, 0 for FALSE, and –1 for UNKNOWN corresponding to a function invocation on NULL arguments; convenience predicate: TRUE if g2 is completely contained in g1 *Contains(g1, g2) ⇔ Within(g2, g1)*
`Relate(g1 Geometry, g2 Geometry, patternMatrix String) : Integer`	The return type is Integer, with a return value of 1 for TRUE, 0 for FALSE, and –1 for UNKNOWN corresponding to a function invocation on NULL arguments; returns TRUE if the spatial relationship specified by the patternMatrix holds

7.2.19.2 Exceptions, errors and error codes

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

7.2.19.3 Example queries

The functions and predicates in this subclause allow the expression of detailed spatial relationship queries.

Return all parcels that intersect a specified Polygon:

```
SELECT Parcel.Name, Parcel.Id FROM Parcels
       WHERE Intersects(Parcels.Location, PolyFromWKB(:wkb, : srid)) = 1
```

Return all parcels completely contained in a specified Polygon:

```
SELECT Parcel.Name, Parcel.Id FROM Parcels
       WHERE Within(Parcels. Location, PolyFromWKB(:wkb, :srid)) = 1
```

The following adjacency query may be used to select all parcels that are “adjacent” to a query parcel and share one or more boundary lines with a query parcel while excluding parcels that share only corner Points.

```
SELECT Parcel.Name, Parcel.Id FROM Parcels
       WHERE Touches(Parcels. Location, PolyFromWKB(:wkb, :srid)) = 1 and
           Overlaps(Boundary(Parcels. Location), Boundary(PolyFromWKB(:wkb, :srid))) = 1
```

7.2.20 SQL functions for distance relationships

7.2.20.1 Component overview

The function shown in Table 19 is used to calculate the distance between two geometric objects.

Table 19 — SQL functions for distance relationships

Function	Description
`Distance(g1 Geometry, g2 Geometry) : Double Precision`	return the distance between g1 and g2

7.2.20.2 Exceptions, errors and error codes

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

7.2.20.3 Example query

The following query returns the name of the state and the fragment(s) of the state that fall within the query Polygon for each state that intersects the query Polygon.

```
SELECT Airport.Name FROM Airports

WHERE Distance(PointFromText(:pointTaggedText, :srid), Airport.Location) < 2000
```

7.2.21 SQL functions that implement spatial operators

7.2.21.1 Component overview

The functions shown in Table 20 implement set-theoretic and constructive operations on geometric objects. These operations are defined for all types of Geometry.

Table 20 — SQL functions that implement spatial operators

Function	Description
`Intersection (g1 Geometry, g2 Geometry) : Geometry`	return a geometric object that is the intersection of geometric objects g1 and g2
`Difference (g1 Geometry, g2 Geometry) : Geometry`	return a geometric object that is the closure of the set difference of g1 and g2
`Union (g1 Geometry, g2 Geometry) : Geometry`	return a geometric object that is the set union of g1 and g2
`SymDifference(g1 Geometry, g2 Geometry) : Geometry`	return a geometric object that is the closure of the set symmetric difference of g1 and g2 (logical XOR of space)
`Buffer (g1 Geometry, d Double Precision) : Geometry`	return a geometric object defined by buffering a distance d around g1, where d is in the distance units for the Spatial Reference of g1
`ConvexHull(g1 Geometry) : Geometry`	return a geometric object that is the convex hull of g1

7.2.21.2 Exceptions, errors and error codes

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

7.2.21.3 Example query

The following query returns the name of the state and the fragment(s) of the state that fall within the query Polygon for each state that intersects the query Polygon.

```
SELECT States.Name, Intersection(PolyFromWKB(:wkb, :srid), States.Location)
FROM States
WHERE Intersects(PolyFromWKB(:wkb, :srid), States.Location)
```

7.2.22 SQL function usage and references to Geometry

The SQL Functions that operate on Geometry Types have been defined above to take geometric objects as arguments. This conforms to the model for value based UDTs in SQL.

SQL Type may also support the concept of persistent references to instances of the Type. To support the latter type of implementation, a reference to a Geometry Type instance, `REF(Geometry)`, may be used in place of a Geometry value in the SQL functions defined in this subclause.

Annex A
(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 A.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 GeomCollection Multipoint Multicurve Multilinestring Multisurface Multipolygon	ST_Point ST_Curve ST_Linestring ST_Circularstring ST_CompoundCurve ST_Surface ST_CurvePolygon ST_Polygon ST_Collection ST_Multipoint ST_MultiCurve ST_Multilinestring ST_Multisurface ST_Multipolygon	—
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() —	ST_Point() ST_X() ST_Y() ST_ExplicitPoint()	Return the Point Return the X-coordinate of point Return the Y-coordinate of point —
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 B
(normative)

Conformance tests

B.1 Purpose of this annex

In order to conform to this part of ISO 19125 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:

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

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

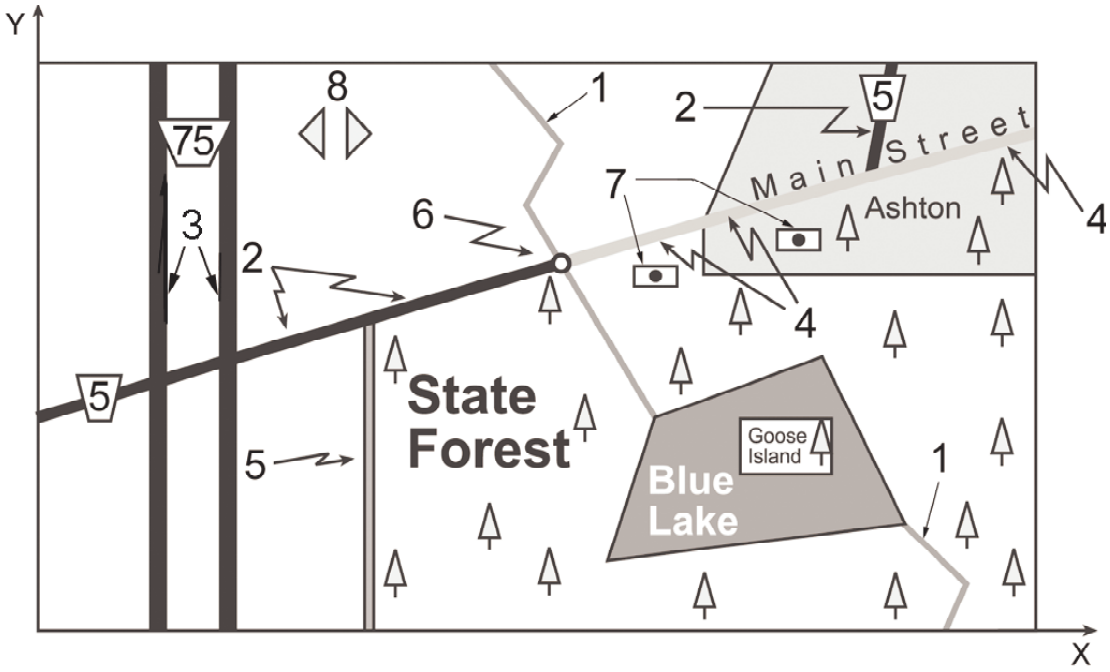
b) SQL with Geometry Types implementation of feature tables supporting both textual and binary SQL/CLI access to geometry.

This annex provides a conformance test for this part of ISO 19125. In general, the scope of the tests is to exercise each functional aspect of the specification 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.

B.2 Test data

B.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 specification. 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 B.1 — Test Data Concept — Joe's 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.

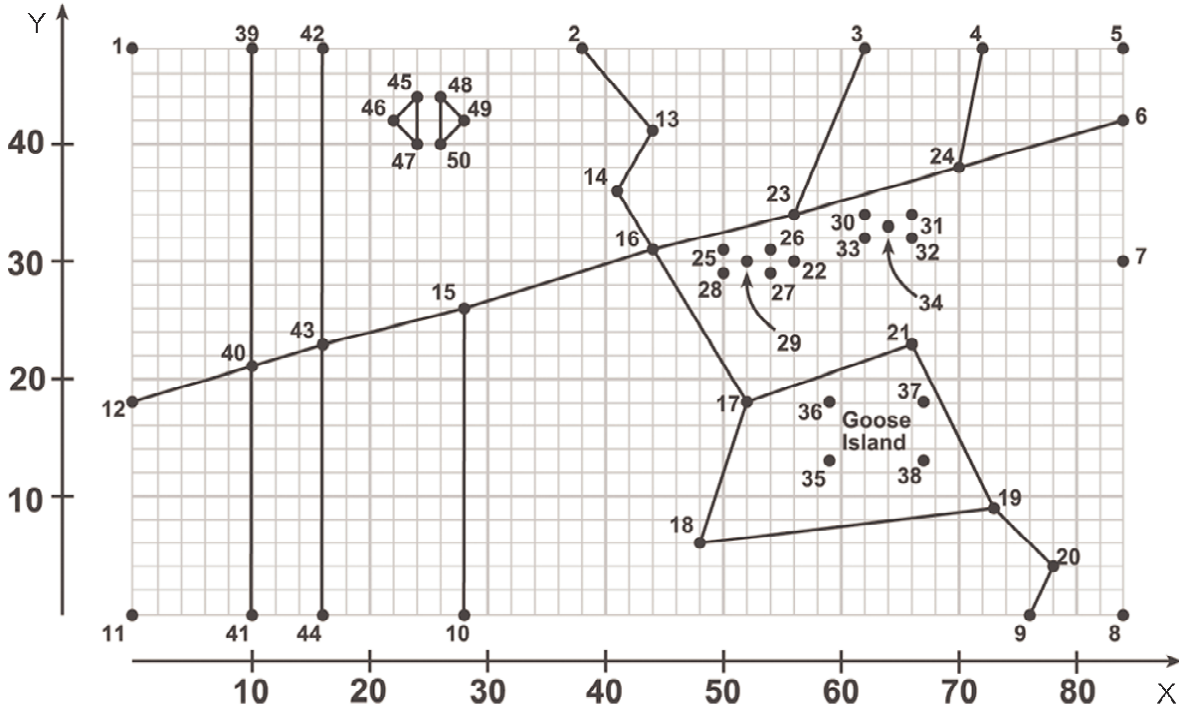
k) There is a one-lane road forming part of the boundary of the State Forest, shown as a grey line with black borders.

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

B.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 B.2 — Points in the Blue Lake data set

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

Table B.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

B.3 Conformance tests

B.3.1 Normalized geometry schema

B.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 specification. Table B.2 shows the queries that accomplish this test.

Table B.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]]'

B.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  VARCHAR(256),
       auth_srid  INTEGER,
       srtext     VARCHAR(2048));

-- CREATE GEOMETRY_COLUMNS METADATA TABLE

CREATE TABLE geometry_columns (
       f_catalog_name    VARCHAR(256),
       f_table_schema    VARCHAR(256),
       f_table_name      VARCHAR(256),
       f_geometry_column VARCHAR(256),
       g_catalog_name    VARCHAR(256),
       g_table_schema    VARCHAR(256),

       g_table_name      VARCHAR(256),
       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));

-- Lakes

CREATE TABLE lakes (
        fid               INTEGER NOT NULL PRIMARY KEY,

        name              VARCHAR(64),
        shore_gid         INTEGER);

-- Road Segments

CREATE TABLE road_segments (
        fid               INTEGER NOT NULL PRIMARY KEY,
        name              VARCHAR(64),
        aliases           VARCHAR(64),
        num_lanes         INTEGER,
        centerline_gid    INTEGER);

-- Divided Routes

CREATE TABLE divided_routes (
        fid               INTEGER NOT NULL PRIMARY KEY,
        name              VARCHAR(64),
        num_lanes         INTEGER,
        centerlines_gid   INTEGER);

-- Forests

CREATE TABLE forests (
        fid            INTEGER NOT NULL PRIMARY KEY,
        name           VARCHAR(64),
        boundary_gid   INTEGER);

-- Bridges

CREATE TABLE bridges (
        fid           INTEGER NOT NULL PRIMARY KEY,
        name          VARCHAR(64),
        position_gid  INTEGER);

-- Streams

CREATE TABLE streams (
        fid             INTEGER NOT NULL PRIMARY KEY,
        name            VARCHAR(64),
        centerline_gid  INTEGER);

-- Buildings

CREATE TABLE buildings (
        fid             INTEGER NOT NULL PRIMARY KEY,
        address         VARCHAR(64),
        position_gid    INTEGER,
        footprint_gid   INTEGER);

-- Ponds

CREATE TABLE ponds (
        fid             INTEGER NOT NULL PRIMARY KEY,
        name            VARCHAR(64),
```

```
       type            VARCHAR(64),
       shores_gid      INTEGER);

-- Named Places

CREATE TABLE named_places (
       fid             INTEGER NOT NULL PRIMARY KEY,
       name            VARCHAR(64),
       boundary_gid    INTEGER);

-- Map Neatline

CREATE TABLE map_neatlines (
       fid             INTEGER NOT NULL PRIMARY KEY,
       neatline_gid    INTEGER);
```

B.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);
```

B.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;
```

B.3.2 Binary geometry schema

B.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 specification. Table B.3 shows the queries that accomplish this test.

Table B.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]]'

B.3.2.2 Binary geometry schema construction

```
CREATE TABLE spatial_ref_sys (
       srid       INTEGER NOT NULL PRIMARY KEY,
       auth_name  VARCHAR(256),
       auth_srid  INTEGER,
       srtext     VARCHAR(2048));

-- Geometry Columns
```

```
CREATE TABLE geometry_columns (
       f_table_schema    VARCHAR(256),
       f_table_name      VARCHAR(256),
       f_geometry_column VARCHAR(256),

       g_table_schema    VARCHAR(256),
       g_table_name      VARCHAR(256),
       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              VARCHAR(64),
        shore_gid         INTEGER);

-- Road Segments

CREATE TABLE road_segments (
        fid               INTEGER NOT NULL PRIMARY KEY,
        name              VARCHAR(64),

        aliases           VARCHAR(64),
        num_lanes         INTEGER,
        centerline_gid    INTEGER);

-- Divided Routes
```

```
CREATE TABLE divided_routes (
       fid               INTEGER NOT NULL PRIMARY KEY,
       name              VARCHAR(64),
       num_lanes         INTEGER,
       centerlines_gid   INTEGER);

-- Forests

CREATE TABLE forests (
       fid            INTEGER NOT NULL PRIMARY KEY,
       name           VARCHAR(64),
       boundary_gid   INTEGER);

-- Bridges

CREATE TABLE bridges (
       fid           INTEGER NOT NULL PRIMARY KEY,
       name          VARCHAR(64),
       position_gid  INTEGER);

-- Streams

CREATE TABLE streams (
       fid             INTEGER NOT NULL PRIMARY KEY,
       name            VARCHAR(64),
       centerline_gid  INTEGER);

-- Buildings

CREATE TABLE buildings (
       fid             INTEGER NOT NULL PRIMARY KEY,
       address         VARCHAR(64),
       position_gid    INTEGER,
       footprint_gid   INTEGER);

-- Ponds

CREATE TABLE ponds (
       fid             INTEGER NOT NULL PRIMARY KEY,
       name            VARCHAR(64),
       type            VARCHAR(64),
       shores_gid      INTEGER);

-- Named Places

CREATE TABLE named_places (
       fid             INTEGER NOT NULL PRIMARY KEY,
       name            VARCHAR(64),
       boundary_gid    INTEGER);

-- Map Neatline

CREATE TABLE map_neatlines (
       fid             INTEGER NOT NULL PRIMARY KEY,
       neatline_gid    INTEGER);
```

B.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]],PRO
JECTION["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('0103000000020000000500000000000000000004a400000000000003240000000000008050400000000000
00374000000000004052400000000000000224000000000000048400000000000001840000000000004a4000000000000032
```

```
400500000000000000000804d4000000000000032400000000000c0504000000000000032400000000000c0504000000000000
002a400000000000804d400000000000002a400000000000804d400000000000003240');

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('0102000000050000000000000000000000000000000000032400000000000002440000000000000354000
0000000000304000000000000037400000000000003c400000000000003a4000000000000046400000000000003f40');

INSERT INTO road_segment_geom VALUES (102, 44.0, 31.0, 70.0, 38.0,
HEXTOVARBINARY('0102000000030000000000000000004640000000000003f400000000000004c4000000000000041400000
0000000805140000000000004340');

INSERT INTO road_segment_geom VALUES (103, 70.0, 38.0, 72.0, 48.0,
HEXTOVARBINARY('0102000000020000000000000008051400000000000043400000000000005240000000000004840')
;

INSERT INTO road_segment_geom VALUES (104, 70.0, 38.0, 84.0, 42.0,
HEXTOVARBINARY('0102000000020000000000000008051400000000000043400000000000005540000000000004540')
;

INSERT INTO road_segment_geom VALUES (105, 28.0, 0.0, 28.0, 26.0,
HEXTOVARBINARY('0102000000020000000000000008051400000000000043400000000000005540000000000004540')
;

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('0105000000020000000102000000030000000000000000002440000000000000484000000000000024400
0000000000003540000000000000244000000000000000000010200000003000000000000000003040000000000000000000
0000000002440000000000003740000000000003040000000000004840');

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('0106000000020000000103000000020000000500000000000000000003c40000000000003a4000000000
00003c400000000000000000000000000000055400000000000000000000000000000055400000000000004540000000000000
3c400000000000003a4005000000000000000004a400000000000003240000000000008050400000000000003740000000000
0040524000000000000022400000000000004840000000000000184000000000000004a400000000000003240010300000001
0000000500000000000000804d4000000000000032400000000000c0504000000000000032400000000000c05040000000
0000002a400000000000804d400000000000002a400000000000804d400000000000003240');

INSERT INTO forests VALUES(109, 'Green Forest', 101);

-- Bridges

INSERT INTO bridge_geom VALUES(101, 44.0, 31.0, 44.0, 31.0,
HEXTOVARBINARY('0101000000000000000004640000000000003f40');

INSERT INTO bridges VALUES(110, 'Cam Bridge', 101);

-- Streams

INSERT INTO stream_geom VALUES(101, 38.0, 18.0, 52.0, 48.0,
HEXTOVARBINARY('0102000000050000000000000000004340000000000004840000000000000464000000000000804440000
0000000804440000000000000424000000000000046400000000000003f400000000000004a400000000000003240');

INSERT INTO stream_geom VALUES(102, 73.0, 0.0, 78.0, 9.0,
HEXTOVARBINARY('0102000000030000000000000000053400000000000000000000000000008053400000000000010400000
000000040524000000000000002240');

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('0101000000000000000000004a400000000000003e40');

INSERT INTO building_pt_geom VALUES(102, 64.0, 33.0, 64.0, 33.0,
HEXTOVARBINARY('0101000000000000000000504000000000000804040');

INSERT INTO building_area_geom VALUES(101, 50.0, 29.0, 54.0, 31.0,
HEXTOVARBINARY('0103000000100000005000000000000000000494000000000000003f400000000000004b400000000000
003f400000000000004b400000000000003d4000000000000049400000000000003d4000000000000049400000000000003f
40');

INSERT INTO building_area_geom VALUES(102, 62.0, 32.0, 66.0, 34.0,
HEXTOVARBINARY('0103000000100000005000000000000000080504000000000000041400000000000004f400000000000
004140000000000004f4000000000000040400000000000805040000000000000404000000000008050400000000000004140
40');

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('0106000000020000000103000000010000000400000000000000000038400000000000004640000000000
0003640000000000000454000000000000038400000000000004440000000000000384000000000000046400103000000001
000000040000000000000000003a400000000000004640000000000003a400000000000004440000000000003c40000000
000000454000000000000003a400000000000004640');

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('0103000000010000000600000000000000000004f400000000000004840000000000000554000000000000
004840000000000000554000000000000003e400000000000004c400000000000003e400000000000004c4000000000000041
400000000000004f400000000000004840');

INSERT INTO named_place_geom VALUES(102, 59.0, 13.0, 67.0, 18.0,
HEXTOVARBINARY('0103000000010000000500000000000000000c050400000000000002a400000000000c050400000000000
003240000000000804d4000000000000032400000000000804d400000000000002a400000000000c050400000000000002a
40');

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('0103000000010000000500000000000000000000000000000000000000000000000000000000000000000000
004840000000000005540000000000000484000000000000055400000000000000000000000000000000000000000000000000
00');

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);
```

B.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;
```

B.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 the specification;

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

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

Table B.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]
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]

Table B.4 (*continued*)

ID	Functionality Tested	Query Description	Answer
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))'
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]

Table B.4 (*continued*)

ID	Functionality Tested	Query Description	Answer
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
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 GeometryCollection) : Integer	For this test, we will determine the number of geometries in Route 75.	2
T31	GeometryN(g GeometryCollection, 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.

Table B.4 (*continued*)

ID	Functionality Tested	Query Description	Answer
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.
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]

Table B.4 (*continued*)

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 specification; 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.

B.3.3.1 Geometry types and functions schema construction

```
CREATE TABLE spatial_ref_sys (
       srid       INTEGER NOT NULL PRIMARY KEY,
       auth_name  VARCHAR(256),
       auth_srid  INTEGER,
       srtext     VARCHAR(2048));

-- Lakes

CREATE TABLE lakes (
       fid               INTEGER NOT NULL PRIMARY KEY,
       name              VARCHAR(64),
       shore             POLYGON);

-- Road Segments

CREATE TABLE road_segments (
       fid               INTEGER NOT NULL PRIMARY KEY,
       name              VARCHAR(64),
```

```
        aliases           VARCHAR(64),
        num_lanes         INTEGER,
        centerline        LINESTRING);

-- Divided Routes

CREATE TABLE divided_routes (
        fid               INTEGER NOT NULL PRIMARY KEY,
        name              VARCHAR(64),
        num_lanes         INTEGER,
        centerlines       MULTILINESTRING);

-- Forests

CREATE TABLE forests (
        fid            INTEGER NOT NULL PRIMARY KEY,
        name           VARCHAR(64),
        boundary       MULTIPOLYGON);

-- Bridges

CREATE TABLE bridges (
        fid           INTEGER NOT NULL PRIMARY KEY,
        name          VARCHAR(64),
        position      POINT);

-- Streams

CREATE TABLE streams (
        fid             INTEGER NOT NULL PRIMARY KEY,
        name            VARCHAR(64),
        centerline      LINESTRING);

-- Buildings

CREATE TABLE buildings (
        fid             INTEGER NOT NULL PRIMARY KEY,
        address         VARCHAR(64),
        position        POINT,
        footprint       POLYGON);

-- Ponds

CREATE TABLE ponds (
        fid             INTEGER NOT NULL PRIMARY KEY,
        name            VARCHAR(64),
        type            VARCHAR(64),
        shores          MULTIPOYLGON);

-- Named Places

CREATE TABLE named_places (
        fid             INTEGER NOT NULL PRIMARY KEY,
        name            VARCHAR(64),
        boundary        POLYGON);

-- Map Neatline

CREATE TABLE map_neatlines (
        fid             INTEGER NOT NULL PRIMARY KEY,
        neatline        POLYGON);
```

B.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));
```

B.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';
```