ISO/IEC JTC 1/SC 32/WG 4: TXL-017r2
INCITS H2-2005-108r2
April 21, 2005
Authoritative file: txl017-Topo-Routine-Details.pdf
Number of Pages: 93

ISO

International Organization for Standardization

ISO/IEC JTC 1/SC 32

Data Management and Interchange

WG 4

SQL Multimedia and Application Packages

Title: Topo-Geo and Topo-Net 3: Routine Details
Author: Paul Scarponcini
Status: SQL/MM—Part 3: Spatial Change Proposal
Source: US Contribution
Abstract:

References:

[1] ISO/IEC CD 13249-3:200x(E) – Text for Continuation CD Editing Meeting **Information technology – Database languages – SQL Multimedia and Application Packages — Part 3: Spatial**, January 24, 2005.

[2] ISO TC211 19107, **Geographic Information – Spatial Schema**, 2001.

[3] ANSI INCITS H2-2003-250, **Approaching Topology**, Paul Scarponcini, April 15, 2003.

[4] ISO TC204 14825, **Intelligent Transport Systems — Geographic Data Files — Overall Data Specification**, October 10, 2002.

[5] ANSI INCITS H2-2003-336, ISO/IEC JTC 1/SC 32/WG 4:MEL-016, **Topology in Three Part Harmony**, Paul Scarponcini, August 8, 2003.

[6] ISO/IEC IS 9075-2:1999, **Information technology – Database languages – SQL – Part 2: Foundation (SQL/Foundation)**, December, 1999.

[7] ISO/IEC JTC 1/SC 32/WG 4:STX-012r1, **Consolidated Ballot Comments on SQL/MM Part 3: Spatial (3rd Edition)**, October, 2004.

[8] ISO/IEC JTC 1/SC 32/WG 4:STX-021r1, **Topo-Geo and Topo-Net 1 – The Concepts**, November, 2004.

[9] ANSI INCITS H2-2005-107r2, ISO/IEC JTC 1/SC 32/WG 4:TXL-016r2, **Topo-Geo and Topo-Net 2: Concepts, The Sequel**, April 21, 2005.

1 Introduction

1.1 Ballot Comments addressed

This paper partially addresses ballot comments [7]:

GBR-P03-001	1-Major Technical	*P03-01 Scope*	This clause includes a reference to topology, but there is no support for it in this document. Support for topology should be added. **Solution** None provided with comment.
USA-P03-001	1-Major Technical	*P03-01, Scope*	The first bullet of the second paragraph suggests that spatial data includes topology yet topology is not specified in this part. A new clause for topology needs to be added, consistent with ISO TC211 19107. **Solution** None provided with comment.

It provides the details based on the definitions and concepts presented in [8] and [9].

1.2 Changes from Previous Version

The following changes were made from the previous version of this paper:

The phrase “automatically generate the next available” has been changed to “generate a unique”.

For the ST_EDGE table definition in X.2.3, constraints changed to a check constraints with absolute values

Clause X.3.5 ST_GetFaceEdges, Description 2)c)i) switched to follow ii) and reworded for clarity and to insure consistent results.

The phrase “select count (*) from” has been changed to upper case.

X.3.18 ST_CreateTopoGeo Procedure, Description 2)j)i) has been reworded.

X.3.18 ST_CreateTopoGeo Procedure changed to NOT DETERMINISTIC, since, for example, the edge for a face with one edge can begin at any point along the face boundary.

Creation of Face 0 added to the ST_InitTopoGeo Procedure (X.3.17).

1.3 Summary of Approach

Various other documents show the evolution of the proposed topology approach [3, 5, 8]. A tabular approach consistent with the Mini-Topo model of Minimally Redundant Topology [2] has been agreed. Two models are included: Topology-Geometry (Topo-Geo), for spatial topology, includes nodes, edges, and faces whilst Topology-Network (Topo-Net), for linear topology, includes nodes and links. The approach attempts to be consistent with ISO 19107 – Spatial Schema [2] fromTC211, the OpenGIS Consortium, and ISO TC204, Intelligent Transportation Systems, the latter in support of the development of a proposed SQL implementation of their Graphic Data Files (GDF) standard, ISO 14825 [4]. Worked examples were provided in [8]. Revisions to the definitions and concepts contained in [1] have been provided in [9] to satisfy possible problems 3-303 and 3-304 [1]. This proposal is based upon the revised definitions and concepts.

1.4 Conventions

1. SMALLCAPS denote numbered editorial instructions;

New Text	New text is identified by using a red bold-underlined font except where entirely new sub-clauses are proposed;
~~Deleted Text~~	Deleted text is identified by using a blue strikethrough font;
Plain	denotes existing text to be retained;
Note to reader:...	boxed text enclose notes to the proposal reader;
Note to editor:...	boxed text enclose helpful hints to the editor;

This following abbreviations are used to reference unnumbered sections: "PS" for "Purpose", "DEFN" for "Definition", "DR" for "Definitional Rules", and "DS" for "Description".

1.5 Electronic availability

This paper is available electronically in PDF (.pdf). To access it using anonymous FTP protocols, connect to the following Internet URL:

ftp://sqlstandards.org/SC32/WG4/Meetings/TXL_2005_04_Berlin_DEU

Use the following filename to get the paper:

txl017r1-Topo-Routine-Details.pdf *PDF*

2 Proposal

2.1 Add the following new clause:

X Topology-Geometry

X.1 Topo-Geo Topology Schema

X.1.1 Introduction

The Topo-Geo Topology Schema views are defined as being in a schema named <*topology-name*> enabling these views to be accessed in the same way as any other tables in any other schema. SELECT privilege on all of these views may be granted to individual users so that they can be queried. These views are updated only by the topology functions provided.

An implementation may define objects that are associated with <*topology-name*> that are not defined in this Clause. An implementation may also add columns to tables that are defined in this Clause.

All of the topological primitives contained in the views owned by the <*topology-name*> schema constitute a single, topologically consistent, topology. Other topologies (e.g., covering a different spatial extent, existing at a different level of abstraction, or associated with a different set of features) can exist in another, independent <*topology-name*> schema.

X.1.2 ST_NODE view

Purpose

Contains the node type of topological primitives (ST_Node) contained in the <topology-name> Topology-Geometry.

Definition

```
CREATE VIEW ST_NODE AS
   SELECT NODE_ID, GEOMETRY, CONTAINING_FACE
      FROM ST_TOPO_GEO.ST_NODE
```

X.1.3 ST_EDGE view

Purpose

Contains the edge type of topological primitives (ST_Edge) contained in the <topology-name> Topology-Geometry.

Definition

```
CREATE VIEW ST_EDGE AS
   SELECT EDGE_ID, START_NODE, END_NODE,
         NEXT_LEFT_EDGE, NEXT_RIGHT_EDGE,
         LEFT_FACE, RIGHT_FACE, GEOMETRY
      FROM ST_TOPO_GEO.ST_EDGE
```

X.1.4 ST_FACE view

Purpose

Contains the face type of topological primitives (ST_Face) contained in the <topology-name> Topology-Geometry.

Definition

```
CREATE VIEW ST_FACE AS
   SELECT FACE_ID, MBR
      FROM ST_TOPO_GEO.ST_FACE
```

X.2 Topo-Geo Definition Schema

X.2.1 Introduction

The only purpose of this Topo-Geo Definition Schema is to provide a data model to support ST_Topo-Geo views and to assist understanding. The base tables of this Topo-Geo Definition Schema are defined as being in a schema named *ST_TOPO_GEO*. The table definitions are as complete as the definitional power of ISO/IEC 9075 allows. The table definitions are supplemented with assertions where appropriate. Each description comprises three parts:

1) The function of the definition is stated.
2) The SQL definition of the object is presented as a <table definition>.
3) An explanation of the object.

The specification provides only a model of the base tables that are required, and does not imply that an implementation shall provide the functionality in the manner described in this clause.

X.2.2 ST_NODE base table

Purpose

Contains the node type of topological primitives (ST_Node) contained in the Topology-Geometry.

Definition

```
CREATE TABLE ST_NODE
   (
   NODE_ID INTEGER NOT NULL,
   GEOMETRY ST_Point NOT NULL,
   CONTAINING_FACE INTEGER,

   CONSTRAINT ST_NODE_PRIMARY_KEY PRIMARY KEY(NODE_ID),
   CONSTRAINT FACE_EXISTS FOREIGN KEY(CONTAINING_FACE)
         REFERENCES ST_FACE(FACE_ID)
   )
```

Description

1) The value of *NODE_ID* is the unique identifier of the node.

2) The value of *GEOMETRY* is the spatial location of the node.

3) The value of *CONTAINING_FACE* is the unique identifier of the face containing the node if the node is an isolated node. Otherwise *CONTAINING_FACE* is the null value.

4) All geometry values in the *ST_NODE*, *ST_EDGE*, and *ST_FACE* views in the same *atopology* schema shall have the same spatial reference system identifier.

X.2.3 ST_EDGE base table

Purpose

Contains the edge type of topological primitives (ST_Edge) contained in the Topology-Geometry.

Definition

```
CREATE TABLE ST_EDGE
   (
   EDGE_ID INTEGER NOT NULL,
   START_NODE INTEGER NOT NULL,
   END_NODE INTEGER NOT NULL,
   NEXT_LEFT_EDGE INTEGER NOT NULL,
   NEXT_RIGHT_EDGE INTEGER NOT NULL,
   LEFT_FACE INTEGER NOT NULL,
   RIGHT_FACE INTEGER NOT NULL,
   GEOMETRY ST_Curve NOT NULL,

   CONSTRAINT ST_EDGE_PRIMARY_KEY PRIMARY KEY(EDGE_ID),
   CONSTRAINT START_NODE_EXISTS FOREIGN KEY(START_NODE)
         REFERENCES ST_NODE(NODE_ID),
   CONSTRAINT END_NODE_EXISTS FOREIGN KEY(END_NODE)
         REFERENCES ST_NODE(NODE_ID),
   CONSTRAINT NEXT_LEFT_EDGE_EXISTS
         CHECK (ABS(NEXT_LEFT_EDGE)) IN (SELECT EDGE_ID FROM ST_EDGE)),
   CONSTRAINT NEXT_RIGHT_EDGE_EXISTS
         FOREIGN KEY(NEXT_RIGHT_EDGE)
         CHECK (ABS(NEXT_RIGHT_EDGE)) IN (SELECT EDGE_ID FROM ST_EDGE)),
   CONSTRAINT LEFT_FACE_EXISTS FOREIGN KEY(LEFT_FACE)
         REFERENCES ST_FACE(FACE_ID),
   CONSTRAINT RIGHT_FACE_EXISTS FOREIGN KEY(RIGHT_FACE)
         REFERENCES ST_FACE(FACE_ID)
   )
```

Description

1) The value of *EDGE_ID* is the unique identifier of the edge.

2) The value of *START_NODE* is the unique identifier of the node at the start of the edge.

3) The value of *END_NODE* is the unique identifier of the node at the end of the edge.

4) The value of *NEXT_LEFT_EDGE* is the unique identifier of the next edge of the face on the left (when looking in the direction from *START_NODE* to *END_NODE),* moving counterclockwise around the face boundary.

5) The value of *NEXT_RIGHT_EDGE* is the unique identifier of the next edge of the face on the right (when looking in the direction from *START_NODE* to *END_NODE),* moving counterclockwise around the face boundary.

6) The value of *LEFT_FACE* is the unique identifier of the face on the left side of the edge when looking in the direction from *START_NODE* to *END_NODE*.

7) The value of *RIGHT_FACE* is the unique identifier of the face on the right side of the edge when looking in the direction from *START_NODE* to *END_NODE*.

8) The value of *GEOMETRY* is the geometry of the edge.

9) All geometry values in the *ST_NODE*, *ST_EDGE*, and *ST_FACE* views in the same *atopology* schema shall have the same spatial reference system identifier.

X.2.4 ST_FACE base table

Purpose

Contains the face type of topological primitives (ST_Face) contained in the Topology-Geometry.

Definition

```
CREATE TABLE ST_FACE
   (
   FACE_ID INTEGER NOT NULL,
   MBR ST_Polygon,

   CONSTRAINT ST_FACE_PRIMARY_KEY PRIMARY KEY(FACE_ID)
   )
```

Description

1) The value of *FACE_ID* is the unique identifier of the face.

2) The value of *MBR* is the geometry of the minimum bounding rectangle of the face.

3) All geometry values in the *ST_NODE*, *ST_EDGE*, and *ST_FACE* views in the same *atopology* schema shall have the same spatial reference system identifier.

X.3 Topo-Geo Routines

X.3.1 ST_AddIsoNode Function

Purpose

Insert a row into the <topology-name>.ST_NODE view for an isolated node.

Definition

```
CREATE FUNCTION ST_AddIsoNode
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    aface INTEGER,
    apoint ST_Point)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The function *ST_AddIsoNode(CHARACTER VARYING, INTEGER, ST_Point)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology,*
   b) an INTEGER value *aface,*
   c) an *ST_Point* value *apoint.*
2) For the function *ST_AddIsoNode(CHARACTER VARYING, INTEGER, ST_Point)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *apoint* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If any *atopology*.*ST_NODE.GEOMETRY* value is equal to *apoint*, then an exception condition is raised: *SQL/MM Spatial exception – coincident node*.
   d) Let *E1* be an INTEGER value equal to *atopology.ST_EDGE.EDGE_ID* for an edge in *atopology*. Let *G1* be an *ST_Curve* value equal to *atopology.ST_EDGE.GEOMETRY* where *EDGE_ID* is equal to *E1*. If the value returned by *G1.ST_Crosses(apoint)* is equal to 1 (one) for any value of *E1*, then an exception condition is raised: *SQL/MM Spatial exception – edge crosses node.*
   e) If *aface* is not the null value and *apoint.ST_Within(ST_GetFaceGeometry*(*atopology, aface))* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – not within face*.

f) Otherwise:

i) generate a unique node id INTEGER value *anodeid*,

ii) insert into the *atopology.ST_NODE* view values (*anodeid*, *aface*, *apoint*),

iii) return *anodeid*.

3) All geometry values in the *ST_NODE, ST_EDGE,* and *ST_FACE* views in the same *atopology* schema shall have the same spatial reference system identifier.

X.3.2 ST_MoveIsoNode Procedure

Purpose

Update the <topology-name>.ST_NODE.GEOMETRY value of an isolated node.

Definition

```
CREATE PROCEDURE ST_MoveIsoNode
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anode INTEGER,
    apoint ST_Point)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The procedure *ST_MoveIsoNode(CHARACTER VARYING, INTEGER, ST_Point)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology,*
   b) an INTEGER value *anode,*
   c) an *ST_Point* value *apoint.*
2) For the procedure *ST_MoveIsoNode(CHARACTER VARYING, INTEGER, ST_Point)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If *apoint* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   d) If any *atopology*.*ST_NODE.GEOMETRY* value is equal to *apoint*, then an exception condition is raised: *SQL/MM Spatial exception – coincident node*.
   e) If any *atopology.ST_EDGE.START_NODE* or *atopology*.*ST_EDGE.END_NODE* value is equal to *anode*, then an exception condition is raised: *SQL/MM Spatial exception – not isolated node*.
   f) Let *E1* be an INTEGER value equal to *atopology.ST_EDGE.EDGE_ID* for an edge in *atopology*. Let *G1* be an *ST_Curve* value equal to *atopology.ST_EDGE.GEOMETRY* where *EDGE_ID* is equal to *E1*. If the value returned by *G1.ST_Crosses(apoint)* is equal to 1 (one) for any value of *E1*, then an exception condition is raised: *SQL/MM Spatial exception – edge crosses node.*

g) Otherwise, update the *atopology.ST_NODE* view, set the *GEOMETRY* value equal to *apoint* where *NODE_ID* is equal to *anode*.

3) All geometry values in the *ST_NODE, ST_EDGE,* and *ST_FACE* views in the same *atopology* schema shall have the same spatial reference system identifier.

X.3.3 ST_RemoveIsoNode Procedure

Purpose

Delete a row in the <topology-name>.ST_NODE view corresponding to an isolated node.

Definition

```
CREATE PROCEDURE ST_RemoveIsoNode
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anode INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The procedure *ST_RemoveIsoNode(CHARACTER VARYING, INTEGER)* takes the following input parameters:

   a) a CHARACTER VARYING value *atopology,*

   b) an INTEGER value *anode.*

2) For the procedure *ST_RemoveIsoNode(CHARACTER VARYING, INTEGER)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If *anode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   c) If any *atopology.ST_EDGE.START_NODE* or *atopology.ST_EDGE.END_NODE* value is equal to *anode*, then an exception condition is raised: *SQL/MM Spatial exception – not isolated node*.

   d) Otherwise, delete from the *atopology.ST_NODE* view where *NODE_ID* equals *anode*.

X.3.4 ST_AddIsoEdge Function

Purpose

Insert a row for an isolated edge into the <topology-name>.ST_EDGE view connecting two existing isolated nodes.

Definition

```
CREATE FUNCTION ST_AddIsoEdge
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anode INTEGER,
    anothernode INTEGER,
    acurve ST_Curve)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The function *ST_AddIsoEdge(CHARACTER VARYING, INTEGER, INTEGER, ST_Curve)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology,*
   b) an INTEGER value *anode,*
   c) an INTEGER value *anothernode,*
   d) an *ST_Curve* value *acurve.*
2) For the function *ST_AddIsoEdge(CHARACTER VARYING, INTEGER, INTEGER, ST_Curve)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If *anothernode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   d) If *acurve* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) If the value returned by *acurve.ST_IsSimple()* is not equal to 1 (one), then an exception condition is raised: *SQL/MM Spatial exception – curve not simple*.

f) If SELECT COUNT(*) FROM *atopology.ST_NODE* WHERE *NODE_ID* is equal to *anode* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent node.*

g) If SELECT COUNT(*) FROM *atopology.ST_NODE* WHERE *NODE_ID* is equal to *anothernode* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent node.*

h) If any *atopology.ST_EDGE.START_NODE* or *atopology.ST_EDGE.END_NODE* value is equal to *anode*, then an exception condition is raised: *SQL/MM Spatial exception – not isolated node*.

i) If any *atopology.ST_EDGE.START_NODE* or *atopology.ST_EDGE.END_NODE* value is equal to *anothernode*, then an exception condition is raised: *SQL/MM Spatial exception – not isolated node*.

j) Let *F1* be the INTEGER value equal to *atopology.ST_NODE.CONTAINING_FACE* where *NODE_ID* is equal to *anode*. Let *F2* be the INTEGER value equal to *atopology.ST_NODE.CONTAINING_FACE* where *NODE_ID* is equal to *anothernode.* If *F1* is not equal to *F2*, then an exception condition is raised: *SQL/MM Spatial exception – nodes in different faces*.

k) If *acurve.ST_Within(ST_GetFaceGeometry*(*atopology, F1))* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – geometry not within face*.

l) Let *P1* be the *ST_Point* value equal to *atopology.ST_NODE.GEOMETRY* where *NODE_ID* is equal to *anode*. If *P1* is not equal to *acurve.ST_StartPoint*, then an exception condition is raised: *SQL/MM Spatial exception – start node not geometry start point*.

m) Let *P2* be the *ST_Point* value equal to *atopology.ST_NODE.GEOMETRY* where *NODE_ID* is equal to *anothernode*. If *P2* is not equal to *acurve.ST_EndPoint*, then an exception condition is raised: *SQL/MM Spatial exception – end node not geometry end point*.

n) Let *N1* be a *NODE_ID* value in *atopology.ST_NODE* such that *N1* does not exist as either an *atopology.ST_EDGE.START_NODE* or *atopology.ST_EDGE.END_NODE* value. Let *G1* be the *atopology.ST_NODE.GEOMETRY* value where *NODE_ID* is equal to *N1*. If *acurve.ST_Crosses(G1)* is not equal to 0 (zero) for all values of *N1*, then an exception condition is raised: *SQL/MM Spatial exception – geometry crosses a node*.

o) Let *G* be an *atopology.ST_EDGE.GEOMETRY* value. If *acurve.ST_Intersects(G)* is not equal to 0 (zero) for all values of *G*, then an exception condition is raised: *SQL/MM Spatial exception – geometry intersects an edge*.

p) Otherwise:

   i) generate a unique edge id INTEGER value *anedgeid*,

   ii) insert into the *atopology.ST_EDGE* view values (*anedgeid, anode, anothernode, -(anedgeid), anedgeid, F1, F1, acurve*),

   iii) return *anedgeid*.

3) All geometry values in the *ST_NODE, ST_EDGE,* and *ST_FACE* views in the same *atopology* schema shall have the same spatial reference system identifier.

X.3.5 ST_GetFaceEdges Function

Purpose

Return a table containing signed edge IDs for the edges which bound a face, in counterclockwise order.

Definition

```
CREATE FUNCTION ST_GetFaceEdges
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    aface INTEGER)
   RETURNS TABLE
      (sequence INTEGER,
       edge INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The function *ST_GetFaceEdges(CHARACTER VARYING, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology,*
   b) an INTEGER value *aface.*
2) For the function *ST_GetFaceEdges(CHARACTER VARYING, INTEGER)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *aface* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) Otherwise, return a table value consisting of the following two columns:
      i) a column *edge* of type INTEGER, which contains the signed edge IDs of the bounding edges of the face having an *atopology.ST_FACE.FACE_ID* value equal to *aface*, such that edge IDs for edges having the specified face on the right side of the edge when looking in the direction of the edge from start node to end node are negated
      ii) a column *sequence* containing consecutive values of type INTEGER, which represents the ordering of the edges in column *edge* consistent with a counterclockwise traversal around the face. The row with the lowest value for *edge* shall have a *sequence* value of 1.

X.3.6 ST_ChangeEdgeGeom Procedure

Purpose

Update the <topology-name>.ST_EDGE.GEOMETRY value.

Definition

```
CREATE PROCEDURE ST_ChangeEdgeGeom
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anedge INTEGER,
    acurve ST_Curve)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The procedure *ST_ChangeEdgeGeom(CHARACTER VARYING, INTEGER, ST_Curve)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology,*
   b) an INTEGER value *anedge,*
   c) an *ST_Curve* value *acurve.*
2) For the procedure *ST_ChangeEdgeGeom(CHARACTER VARYING, INTEGER, ST_Curve)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anedge* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If *acurve* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   d) If the value returned by *acurve.ST_IsSimple()* is not equal to 1 (one), then an exception condition is raised: *SQL/MM Spatial exception – curve not simple*.
   e) Let *P1* be the *ST_Point* value equal to *atopology.ST_NODE.GEOMETRY* where *NODE_ID* equals the value of *atopology.ST_EDGE.START_NODE* where *EDGE_ID* is equal to *anedge*. If *P1* is not equal to *acurve.ST_StartPoint*, then an exception condition is raised: *SQL/MM Spatial exception – start node not geometry start point*.
   f) Let *P2* be the *ST_Point* value equal to *atopology.ST_NODE.GEOMETRY* where *NODE_ID* equals the value of *atopology.ST_EDGE.END_NODE* where *EDGE_ID* is equal to *anothernode*. If *P2* is not equal to *acurve.ST_EndPoint*, then an exception condition is raised: *SQL/MM Spatial exception – end node not geometry end point*.

g) Let *N1* be a *NODE_ID* value in *atopology.ST_NODE* such that *N1* does not exist as either an *atopology.ST_EDGE.START_NODE* or *atopology.ST_EDGE.END_NODE* value. Let *G1* be the *atopology.ST_NODE.GEOMETRY* value where NODE_ID is equal to *N1*. If *acurve.ST_Crosses(G1)* is not equal to 0 (zero) for all values of *N1*, then an exception condition is raised: *SQL/MM Spatial exception – geometry crosses a node*.

h) Let *G* be an *atopology.ST_EDGE.GEOMETRY* value. If *acurve.ST_Crosses(G)* is not equal to 0 (zero) for all values of *G*, then an exception condition is raised: *SQL/MM Spatial exception – geometry intersects an edge*.

i) Otherwise, update *atopology*.*ST_EDGE*, set the *GEOMETRY* value equal to *acurve* where *EDGE_ID* is equal to *anedge*.

3) All geometry values in the *ST_NODE, ST_EDGE,* and *ST_FACE* views in the same *atopology* schema shall have the same spatial reference system identifier.

X.3.7 ST_RemoveIsoEdge Procedure

Purpose

Delete a row in the <topology-name>.ST_EDGE view corresponding to an isolated edge.

Definition

```
CREATE PROCEDURE ST_RemoveIsoEdge
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anedge INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The procedure *ST_RemoveIsoEdge(CHARACTER VARYING, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology,*
   b) an INTEGER value *anedge.*
2) For the procedure *ST_RemoveIsoEdge(CHARACTER VARYING, INTEGER)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anedge* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If *atopology*.*ST_EDGE.LEFT_FACE* is not equal to *atopology*.*ST_EDGE.RIGHT_FACE*, then an exception condition is raised: *SQL/MM Spatial exception – not isolated edge*.
   d) Let *N1* be the *atopology.ST_EDGE.START_NODE* of the edge whose *EDGE_ID* is equal to *anedge*. If any *atopology.ST_EDGE.START_NODE* or *atopology*.*ST_EDGE.END_NODE* value is equal to *N1*, then an exception condition is raised: *SQL/MM Spatial exception – not isolated edge*.
   e) Let *N2* be the *atopology.ST_EDGE.END_NODE* of the edge whose *EDGE_ID* is equal to *anedge*. If any *atopology.ST_EDGE.START_NODE* or *atopology*.*ST_EDGE.END_NODE* value is equal to *N2*, then an exception condition is raised: *SQL/MM Spatial exception – not isolated edge*.
   f) Otherwise, delete from the *atopology.ST_EDGE* view where *EDGE_ID* equals *anedge.*

X.3.8 ST_NewEdgesSplit Function

Purpose

Split an edge by creating a new node along an existing edge, deleting the original edge and replacing it with two new edges.

Definition

```
CREATE FUNCTION ST_NewEdgesSplit
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anedge INTEGER,
    apoint ST_Point)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The function *ST_NewEdgesSplit(CHARACTER VARYING, INTEGER, ST_Point)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology,*
   b) an INTEGER value *anedge,*
   c) an *ST_Point* value *apoint.*
2) For the function *ST_NewEdgesSplit(CHARACTER VARYING, INTEGER, ST_Point)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anedge* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If *apoint* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   d) If there is no row in the *atopology*.*ST_EDGE* view with an *EDGE_ID* value equal to *anedge*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent edge*.
   e) Let *G* be equal to the *ST_Curve* value of *atopology*.*ST_EDGE.GEOMETRY* where *EDGE_ID* is equal to *anedge*. If *apoint.ST_Within(G)* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – point not on edge*.
   f) If any *atopology*.*ST_NODE.GEOMETRY* value is equal to *apoint*, then an exception condition is raised: *SQL/MM Spatial exception – coincident node*.

g) Otherwise:

i) generate a unique node id INTEGER value *newnode,*

ii) insert into the *atopology.ST_NODE* view values (*newnode, apoint,* NULL),

iii) select from *atopology.ST_EDGE START_NODE, END_NODE, NEXT_LEFT_EDGE, NEXT_RIGHT_EDGE, LEFT_FACE, RIGHT_FACE,* and *GEOMETRY* into *oldstart, oldend, oldnextleft, oldnextright, oldleft, oldright,* and *oldgeom* where *EDGE_ID* is equal to *anedge*,

iv) delete from *atopology.ST_EDGE* where *EDGE_ID* is equal to *anedge*,

v) generate two unique edge id INTEGER values *newedge1* and *newedge2,*

vi) create two new *ST_Curve* values, *newgeom1* and *newgeom2* from *oldgeom*, breaking *oldgeom* at the location defined by *apoint*,

vii) insert into *atopology.ST_EDGE* values(*newedge1, oldstart, newnode, newedge2, oldnextright, oldleft, oldright, newgeom1),*

viii) insert into *atopology.ST_EDGE* values(*newedge2, newnode, oldend, oldnextleft, -(newedge1), oldleft, oldright, newgeom2),*

ix) make any necessary updates to the next left and right face edge IDs for any edges incident on the start and end nodes of the edge being split.

x) return *newnode,*

3) Both new edges have the same direction as the edge being split.

4) To determine the two new edge ID values, select *EDGE_ID* from a*topology.ST_EDGE* where *START_NODE* or *END_NODE* is equal to *newnode*.

5) All geometry values in the *ST_NODE, ST_EDGE,* and *ST_FACE* views in the same *atopology* schema shall have the same spatial reference system identifier.

X.3.9 ST_ModEdgeSplit Function

Purpose

Split an edge by creating a new node along an existing edge, modifying the original edge and adding a new edge.

Definition

```
CREATE FUNCTION ST_ModEdgeSplit
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anedge INTEGER,
    apoint ST_Point)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The function *ST_ModEdgeSplit(CHARACTER VARYING, INTEGER, ST_Point)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology,*
   b) an INTEGER value *anedge,*
   c) an *ST_Point* value *apoint.*
2) For the function *ST_ModEdgeSplit(CHARACTER VARYING, INTEGER, ST_Point)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anedge* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If *apoint* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   d) If there is no row in the *atopology*.*ST_EDGE* view with an *EDGE_ID* value equal to *anedge*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent edge*.
   e) Let *G* be equal to the *ST_Curve* value of *atopology*.*ST_EDGE.GEOMETRY* where *EDGE_ID* is equal to *anedge*. If *apoint.ST_Within(G)* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – point not on edge*.
   f) If any *atopology*.*ST_NODE.GEOMETRY* value is equal to *apoint*, then an exception condition is raised: *SQL/MM Spatial exception – coincident node*.

g) Otherwise:

i) generate a unique node id INTEGER value *newnode,*

ii) insert into the *atopology.ST_NODE* view values (*newnode, apoint,* NULL),

iii) select from *atopology.ST_EDGE END_NODE, NEXT_LEFT_EDGE, LEFT_FACE, RIGHT_FACE,* and *GEOMETRY* into *oldend, oldnextleft, oldleft, oldright,* and *oldgeom* where *EDGE_ID* is equal to *anedge*,

iv) generate a unique edge id INTEGER value *newedge,*

v) create two new *ST_Curve* values, *newgeom1* and *newgeom2* from *oldgeom*, breaking *oldgeom* at the location defined by *apoint*,

vi) update *atopology.ST_EDGE* where *EDGE_ID* is equal to *anedge:*

1) set the *END_NODE* value equal to *newnode,*

2) set the *NEXT_LEFT_EDGE* value equal to *newedge,*

3) set the *GEOMETRY* value equal to *newgeom1,*

vii) insert into *atopology.ST_EDGE* values(*newedge, newnode, oldend, oldnextleft, -(anedge), oldleft, oldright, newgeom2),*

viii) make any necessary updates to the next left and right face edge IDs for any edges incident on the start and end nodes of the edge being split.

ix) return *newnode,*

3) The new and modified edges have the same direction as the edge being split.

4) To determine the new edge ID value, select *EDGE_ID* from a*topology.ST_EDGE* where *START_NODE* is equal to *newnode*.

5) All geometry values in the *ST_NODE, ST_EDGE,* and *ST_FACE* views in the same *atopology* schema shall have the same spatial reference system identifier.

X.3.10 ST_NewEdgeHeal Function

Purpose

Heal two edges by deleting the node connecting them, deleting both edges, and replacing them with a new edge whose direction is the same as the first edge provided.

Definition

```
CREATE FUNCTION ST_NewEdgeHeal
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anedge INTEGER,
    anotheredge INTEGER)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The function *ST_NewEdgeHeal(CHARACTER VARYING, INTEGER, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology,*
   b) an INTEGER value *anedge,*
   c) an INTEGER value *anotheredge.*
2) For the function *ST_NewEdgeHeal(CHARACTER VARYING, INTEGER, INTEGER)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anedge* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If *anotheredge* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   d) If there is no row in the *atopology*.*ST_EDGE* view with an *EDGE_ID* value equal to *anedge*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent edge*.
   e) If there is no row in the *atopology*.*ST_EDGE* view with an *EDGE_ID* value equal to *anotheredge*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent edge*.
   f) Check if edges are connected:
      i) let *COMMONNODE* be an INTEGER value equal to 0 (zero),
      ii) let *CASE* be an INTEGER value,

iii) let S1 and E1 be INTEGER values equal to the *atopology.ST_EDGE.START_NODE* and *END_NODE* values, respectively, where *EDGE_ID* is equal to *anedge*,

iv) let S2 and E2 be INTEGER values equal to the *atopology.ST_EDGE.START_NODE* and *END_NODE* values, respectively, where *EDGE_ID* is equal to *anotheredge*,

v) case:

1) if E1 is equal to S2, then set *CASE* equal to 1 (one) and *COMMONNODE* equal to E1,

2) if E1 is equal to E2, then set *CASE* equal to 2 and *COMMONNODE* equal to E1,

3) if S1 is equal to S2, then set *CASE* equal to 3 and *COMMONNODE* equal to S1,

4) if S1 is equal to E2, then set *CASE* equal to 4 and *COMMONNODE* equal to S1,

5) otherwise, an exception condition is raised: *SQL/MM Spatial exception – non-connected edges,*

g) If *COMMONNODE* is equal to the *atopology.ST_EDGE.START_NODE* or *atopology.ST_EDGE.END_NODE* value for any edge other than the edge whose *EDGE_ID* is equal to either *anedge* or *anotheredge*, then an exception condition is raised: *SQL/MM Spatial exception – other edges connected*,

h) Otherwise:

i) delete from the *atopology.ST_NODE* view where *NODE_ID* is equal to *COMMONNODE*,

ii) select from *atopology.ST_EDGE START_NODE, END_NODE, NEXT_LEFT_EDGE, NEXT_RIGHT_EDGE, LEFT_FACE, RIGHT_FACE,* and *GEOMETRY* into *oldstart1, oldend1, oldnextleft1, oldnextright1, oldleft, oldright,* and *oldgeom1* where *EDGE_ID* is equal to *anedge*,

iii) select from *atopology.ST_EDGE START_NODE, END_NODE, NEXT_LEFT_EDGE, NEXT_RIGHT_EDGE,* and *GEOMETRY* into *oldstart2, oldend2, oldnextleft2, oldnextright2,* and *oldgeom2* where *EDGE_ID* is equal to *anotheredge*,

iv) delete from *atopology.ST_EDGE* where *EDGE_ID* is equal to *anedge*,

v) delete from *atopology.ST_EDGE* where *EDGE_ID* is equal to *anotheredge*,

vi) generate a unique edge id INTEGER value *newedge,*

vii) case:

1) if *CASE* is equal to 1 (one) then:

A) create a new *ST_Curve* value *newgeom* from *oldgeom1 and oldgeom2*, by connecting the end of *oldgeom1* to the start of *oldgeom2*,

B) insert into *atopology.ST_EDGE* values(*newedge, oldstart1, oldend2, oldnextleft2, oldnextright1, oldleft, oldright, newgeom*),

2) if *CASE* is equal to 2 then:

A) create a new *ST_Curve* value *newgeom* from *oldgeom1 and oldgeom2*, by reversing the direction of *oldgeom2* and connecting it to the end of *oldgeom1*,

B) insert into *atopology.ST_EDGE* values(*newedge, oldstart1, oldstart2, oldnextright2, oldnextright1, oldleft, oldright, newgeom*),

3) if *CASE* is equal to 3 then:

A) create a new *ST_Curve* value *newgeom* from *oldgeom1 and oldgeom2*, by reversing the direction of *oldgeom2* and connecting *oldgeom1* to the new end,

B) insert into *atopology.ST_EDGE* values(*newedge, oldend2, oldend1, oldnextleft1, oldnextleft2, oldleft, oldright, newgeom*),

4) if *CASE* is equal to 4 then:

A) create a new *ST_Curve* value *newgeom* from *oldgeom1 and oldgeom2*, by connecting *oldgeom1* to the end of *oldgeom2*,

B) insert into *atopology.ST_EDGE* values(*newedge, oldstart2, oldend1, oldnextleft1, oldnextright2, oldleft, oldright, newgeom*),

vii) make any necessary updates to the next left and right face edge IDs for any edges incident on the start and end nodes of the new edge,

viii) return *newedge*.

3) The direction of the new edge shall be the same as the direction of the first edge supplied.

4) All geometry values in the *ST_NODE, ST_EDGE,* and *ST_FACE* views in the same *atopology* schema shall have the same spatial reference system identifier.

X.3.11 ST_ModEdgeHeal Procedure

Purpose

Heal two edges by deleting the node connecting them, modifying the first edge provided, and deleting the second edge.

Definition

```
CREATE PROCEDURE ST_ModEdgeHeal
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anedge INTEGER,
    anotheredge INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The procedure *ST_ModEdgeHeal(CHARACTER VARYING, INTEGER, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology,*
   b) an INTEGER value *anedge,*
   c) an INTEGER value *anotheredge.*
2) For the procedure *ST_ModEdgeHeal(CHARACTER VARYING, INTEGER, INTEGER)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anedge* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If *anotheredge* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   d) If there is no row in the *atopology*.*ST_EDGE* view with an *EDGE_ID* value equal to *anedge*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent edge*.
   e) If there is no row in the *atopology*.*ST_EDGE* view with an *EDGE_ID* value equal to *anotheredge*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent edge*.
   f) Check if edges are connected:
      i) let *COMMONNODE* be an INTEGER value equal to 0 (zero),
      ii) let *CASE* be an INTEGER value,

iii) let S1 and E1 be INTEGER values equal to the *atopology.ST_EDGE.START_NODE* and *END_NODE* values, respectively, where *EDGE_ID* is equal to *anedge*,

iv) let S2 and E2 be INTEGER values equal to the *atopology.ST_EDGE.START_NODE* and *END_NODE* values, respectively, where *EDGE_ID* is equal to *anotheredge*,

v) case:

1) if E1 is equal to S2, then set *CASE* equal to 1 (one) and *COMMONNODE* equal to E1,

2) if E1 is equal to E2, then set *CASE* equal to 2 and *COMMONNODE* equal to E1,

3) if S1 is equal to S2, then set *CASE* equal to 3 and *COMMONNODE* equal to S1,

4) if S1 is equal to E2, then set *CASE* equal to 4 and *COMMONNODE* equal to S1,

5) otherwise, an exception condition is raised: *SQL/MM Spatial exception – non-connected edges,*

g) If *COMMONNODE* is equal to the *atopology.ST_EDGE.START_NODE* or *atopology.ST_EDGE.END_NODE* value for any edge other than the edge whose *EDGE_ID* is equal to either *anedge* or *anotheredge*, then an exception condition is raised: *SQL/MM Spatial exception – other edges connected*,

h) Otherwise:

i) delete from the *atopology.ST_NODE* view where *NODE_ID* is equal to *COMMONNODE*,

ii) select from *atopology.ST_EDGE GEOMETRY* into *oldgeom1* where *EDGE_ID* is equal to *anedge*,

iii) select from *atopology.ST_EDGE START_NODE, END_NODE, NEXT_LEFT_EDGE, NEXT_RIGHT_EDGE,* and *GEOMETRY* into *oldstart2, oldend2, oldnextleft2, oldnextright2,* and *oldgeom2* where *EDGE_ID* is equal to *anotheredge*,

iv) delete from *atopology.ST_EDGE* where *EDGE_ID* is equal to *anotheredge*,

v) case:

1) if *CASE* is equal to 1 (one) then:

A) create a new *ST_Curve* value *newgeom* from *oldgeom1 and oldgeom2*, by connecting the end of *oldgeom1* to the start of *oldgeom2*,

B) update *atopology.ST_EDGE* where *EDGE_ID* is equal to *anedge*:

I) set the *END_NODE* value equal to *oldend2,*

II) set the *NEXT_LEFT_EDGE* value equal to *oldnextleft2,*

III) set the *GEOMETRY* value equal to *newgeom,*

2) if *CASE* is equal to 2 then:

A) create a new *ST_Curve* value *newgeom* from *oldgeom1 and oldgeom2*, by reversing the direction of *oldgeom2* and connecting it to the end of *oldgeom1*,

B) update *atopology.ST_EDGE* where *EDGE_ID* is equal to *anedge*:

I) set the *END_NODE* value equal to *oldstart2,*

II) set the *NEXT_LEFT_EDGE* value equal to *oldnextright2,*

III) set the *GEOMETRY* value equal to *newgeom,*

3) if *CASE* is equal to 3 then:

A) create a new *ST_Curve* value *newgeom* from *oldgeom1 and oldgeom2*, by reversing the direction of *oldgeom2* and connecting *oldgeom1* to the new end,

B) update *atopology.ST_EDGE* where *EDGE_ID* is equal to *anedge*:

I) set the *START_NODE* value equal to *oldend2,*

II) set the *NEXT_RIGHT_EDGE* value equal to *oldnextleft2,*

III) set the *GEOMETRY* value equal to *newgeom,*

4) if *CASE* is equal to 4 then:

A) create a new *ST_Curve* value *newgeom* from *oldgeom1 and oldgeom2*, by connecting *oldgeom1* to the end of *oldgeom2*,

B) update *atopology.ST_EDGE* where *EDGE_ID* is equal to *anedge*:

I) set the *START_NODE* value equal to *oldstart2,*

II) set the *NEXT_RIGHT_EDGE* value equal to *oldnextright2,*

III) set the *GEOMETRY* value equal to *newgeom,*

vi) make any necessary updates to the next left and right face edge IDs for any edges incident on the start and end nodes of the new edge.

3) All geometry values in the *ST_NODE, ST_EDGE,* and *ST_FACE* views in the same *atopology* schema shall have the same spatial reference system identifier.

X.3.12 ST_AddEdgeNewFaces Function

Purpose

Add a new edge and, if in doing so it splits a face, delete the original face and replace it with two new faces.

Definition

```
CREATE FUNCTION ST_AddEdgeNewFaces
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anode INTEGER,
    anothernode INTEGER,
    acurve ST_Curve)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The function *ST_AddEdgeNewFaces(CHARACTER VARYING, INTEGER, INTEGER, ST_Curve)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology,*
   b) an INTEGER value *anode,*
   c) an INTEGER value *anothernode,*
   d) an *ST_Curve* value *acurve.*
2) For the function *ST_AddEdgeNewFaces(CHARACTER VARYING, INTEGER, INTEGER, ST_Curve)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If *anothernode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   d) If *acurve* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) If the value returned by *acurve.ST_IsSimple()* is not equal to 1 (one), then an exception condition is raised: *SQL/MM Spatial exception – curve not simple*.

f) If there is no row in the *atopology.ST_NODE* view with a *NODE_ID* value equal to *anode*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent node*.

g) If there is no row in the *atopology.ST_NODE* view with a *NODE_ID* value equal to *anothernode*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent node*.

h) Let *P1* be the *ST_Point* value equal to *atopology.ST_NODE.GEOMETRY* where *NODE_ID* is equal to *anode*. If *P1* is not equal to *acurve.ST_StartPoint*, then an exception condition is raised: *SQL/MM Spatial exception – start node not geometry start point*.

i) Let *P2* be the *ST_Point* value equal to *atopology.ST_NODE.GEOMETRY* where *NODE_ID* is equal to *anothernode*. If *P2* is not equal to *acurve.ST_EndPoint*, then an exception condition is raised: *SQL/MM Spatial exception – end node not geometry end point*.

j) Let *N1* be a *NODE_ID* value in *atopology.ST_NODE* such that *N1* does not exist as either an *atopology.ST_EDGE.START_NODE* or *atopology.ST_EDGE.END_NODE* value. Let *G1* be the *atopology.ST_NODE.GEOMETRY* value where *NODE_ID* is equal to *N1*. If *acurve.ST_Crosses(G1)* is not equal to 0 (zero) for all values of *N1*, then an exception condition is raised: *SQL/MM Spatial exception – geometry crosses a node*.

k) Let *G* be an *atopology.ST_EDGE.GEOMETRY* value. If *acurve.ST_Crosses(G)* is not equal to 0 (zero) for all values of *G*, then an exception condition is raised: *SQL/MM Spatial exception – geometry crosses an edge*.

l) If there exists a row in the *atopology.ST_EDGE* view with a *START_NODE* value equal to *anode* and an *END_NODE* value equal to *anothernode* and a *GEOMETRY* value equal to *acurve*, then an exception condition is raised: *SQL/MM Spatial exception – coincident edge*.

m) Otherwise:

   i) generate a unique edge id INTEGER value *anedgeid*,

   ii) insert into the *atopology.ST_Edge* view a new row with:

      1) an *EDGE_ID* value equal to *anedgeid*,

      2) a *START_NODE* value equal to *anode*,

      3) an *END_NODE* value equal to *anotherode*,

      4) *NEXT_LEFT_EDGE, NEXT_RIGHT_EDGE, LEFT_FACE,* and *RIGHT_FACE* values as appropriate, and

      5) a *GEOMETRY* value equal to *acurve*,

   iii) if the new edge splits a face, then:

      1) let *F* be an INTEGER value equal to the *FACE_ID* of the face being split,

      2) delete from *atopology.ST_FACE* where *FACE_ID* is equal to *F*,

      3) generate two unique face id INTEGER values *newface1* and *newface2,*

      4) update the appropriate next left and right face edge IDs and left and right face IDs for edges bounding the face being split,

      5) let *MBR1* be the ST_Polygon value equal to the value of *ST_GetFaceGeometry(atopology,newface1).ST_EnvelopeMethod()* where *FACE_ID* is equal to *newface1*,

      6) let *MBR2* be the ST_Polygon value equal to the value of *ST_GetFaceGeometry(atopology,newface2).ST_EnvelopeMethod()* where *FACE_ID* is equal to *newface2,*

      7) insert into *atopology.ST_FACE* values (*newface1, MBR1*),

      8) insert into *atopology.ST_FACE* values (*newface2, MBR2*),

iv) return anedgeid.

3) To determine the two new face ID values, select *LEFT_FACE* and *RIGHT_FACE* from a*topology.ST_EDGE* where *EDGE_ID* is equal to *anedgeid*.

4) All geometry values in the *ST_NODE, ST_EDGE,* and *ST_FACE* views in the same *atopology* schema shall have the same spatial reference system identifier.

X.3.13 ST_AddEdgeModFace Function

Purpose

Add a new edge and, if in doing so it splits a face, modify the original face and add a new face.

Definition

```
CREATE FUNCTION ST_AddEdgeModFace
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anode INTEGER,
    anothernode INTEGER,
    acurve ST_Curve)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The function *ST_AddEdgeModFace(CHARACTER VARYING, INTEGER, INTEGER, ST_Curve)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology,*
   b) an INTEGER value *anode,*
   c) an INTEGER value *anothernode,*
   d) an *ST_Curve* value *acurve.*
2) For the function *ST_AddEdgeModFace(CHARACTER VARYING, INTEGER, INTEGER, ST_Curve)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If *anothernode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   d) If *acurve* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   e) If the value returned by *acurve.ST_IsSimple()* is not equal to 1 (one), then an exception condition is raised: *SQL/MM Spatial exception – curve not simple*.
   f) If there is no row in the *atopology.ST_NODE* view with a *NODE_ID* value equal to *anode*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent node*.

g) If there is no row in the *atopology.ST_NODE* view with a *NODE_ID* value equal to *anothernode*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent node*.

h) Let *P1* be the *ST_Point* value equal to *atopology.ST_NODE.GEOMETRY* where *NODE_ID* is equal to *anode*. If *P1* is not equal to *acurve.ST_StartPoint*, then an exception condition is raised: *SQL/MM Spatial exception – start node not geometry start point*.

i) Let *P2* be the *ST_Point* value equal to *atopology.ST_NODE.GEOMETRY* where *NODE_ID* is equal to *anothernode*. If *P2* is not equal to *acurve.ST_EndPoint*, then an exception condition is raised: *SQL/MM Spatial exception – end node not geometry end point*.

j) Let *N1* be a *NODE_ID* value in *atopology.ST_NODE* such that *N1* does not exist as either an *atopology.ST_EDGE.START_NODE* or *atopology.ST_EDGE.END_NODE* value. Let *G1* be the *atopology.ST_NODE.GEOMETRY* value where *NODE_ID* is equal to *N1*. If *acurve.ST_Crosses(G1)* is not equal to 0 (zero) for all values of *N1*, then an exception condition is raised: *SQL/MM Spatial exception – geometry crosses a node*.

k) Let *G* be an *atopology.ST_EDGE.GEOMETRY* value. If *acurve.ST_Crosses(G)* is not equal to 0 (zero) for all values of *G*, then an exception condition is raised: *SQL/MM Spatial exception – geometry crosses an edge*.

l) If there exists a row in the *atopology.ST_EDGE* view with a *START_NODE* value equal to *anode* and an *END_NODE* value equal to *anothernode* and a *GEOMETRY* value equal to *acurve*, then an exception condition is raised: *SQL/MM Spatial exception – coincident edge*.

m) Otherwise:

   i) generate a unique edge id INTEGER value *anedgeid*,

   ii) insert into the *atopology.ST_Edge* view a new row with:

      1) an *EDGE_ID* value equal to *anedgeid*,

      2) a *START_NODE* value equal to *anode*,

      3) an *END_NODE* value equal to *anotherode*,

      4) *NEXT_LEFT_EDGE, NEXT_RIGHT_EDGE, LEFT_FACE,* and *RIGHT_FACE* values as appropriate, and

      5) a *GEOMETRY* value equal to *acurve*,

   iii) if the new edge splits a face, then:

      1) let *F* be an INTEGER value equal to the *FACE_ID* of the face being split,

      2) update *atopology.ST_FACE* set *MBR* equal to the value of *ST_GetFaceGeometry(atopology,F1).ST_EnvelopeMethod()* where *FACE_ID* is equal to *F1*,

      3) generate a unique face id INTEGER value *newface,*

      4) update the appropriate next left and right face edge IDs and left and right face IDs for edges bounding the face being split,

      5) let *MBR* be the ST_Polygon value equal to the value of *ST_GetFaceGeometry(atopology,newface).ST_EnvelopeMethod()* where *FACE_ID* is equal to *newface,*

      6) insert into *atopology.ST_FACE* values (*newface, MBR*),

   iv) return *anedgeid*.

3) To determine the two new face ID values, select *LEFT_FACE* and *RIGHT_FACE* from *atopology.ST_EDGE* where *EDGE_ID* is equal to *anedgeid*.

4) All geometry values in the *ST_NODE*, *ST_EDGE*, and *ST_FACE* views in the same *atopology* schema shall have the same spatial reference system identifier.

X.3.14 ST_RemEdgeNewFace Function

Purpose

Remove an edge and, if the removed edge separated two faces, delete the original faces and replace them with one new face.

Definition

```
CREATE FUNCTION ST_RemEdgeNewFace
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anedge INTEGER)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The function *ST_RemEdgeNewFace(CHARACTER VARYING, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology,*
   b) an INTEGER value *anedge.*
2) For the function *ST_RemEdgeNewFace(CHARACTER VARYING, INTEGER)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anedge* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If there is no row in the *atopology.ST_EDGE* view with an *EDGE_ID* value equal to *anedge*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent edge*.
   d) Otherwise:
      i) delete from *atopology.ST_EDGE* where *EDGE_ID* is equal to *anedge*,
      ii) if the edge removal results in the healing of two faces, then:
         1) let *F1* and *F2* be the INTEGER values equal to the *FACE_IDs* of the faces to be healed,
         2) delete from *atopology.ST_FACE* where *FACE_ID* is equal to *F1*,
         3) delete from *atopology.ST_FACE* where *FACE_ID* is equal to *F2*,
         4) generate a unique face id INTEGER value *newface,*

5) update the appropriate next left and right face edge IDs and left and right face IDs for edges bounding the faces being healed,
6) let *MBR* be the ST_Polygon value equal to the value of *ST_GetFaceGeometry(atopology,newface).ST_EnvelopeMethod()* where *FACE_ID* is equal to *newface*,
7) insert into *atopology.ST_FACE* values (*newface, MBR*),
8) return *newface*.

3) The nodes in *atopology.ST_NODE* having a *NODE_ID* value equal to *atopology.ST_EDGE.START_NODE* and *atopology.ST_EDGE.END_NODE* where *EDGE_ID* is equal to *anedge* are not deleted.
4) All geometry values in the *ST_NODE, ST_EDGE,* and *ST_FACE* views in the same *atopology* schema shall have the same spatial reference system identifier.

X.3.15 ST_RemEdgeModFace Procedure

Purpose

Remove an edge and, if the removed edge separated two faces, heal the two faces by modifying one of the faces and delete the other.

Definition

```
CREATE PROCEDURE ST_RemEdgeModFace
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    anedge INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The procedure *ST_RemEdgeModFace(CHARACTER VARYING, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology,*
   b) an INTEGER value *anedge.*
2) For the procedure *ST_RemEdgeModFace(CHARACTER VARYING, INTEGER)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anedge* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If there is no row in the *atopology.ST_EDGE* view with an *EDGE_ID* value equal to *anedge*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent edge*.
   d) Otherwise:
      i) delete from *atopology.ST_EDGE* where *EDGE_ID* is equal to *anedge*,
      ii) if the edge removal results in the healing of two faces, then:
         1) let *F1* and *F2* be the INTEGER values equal to the *FACE_IDs* of the two faces to be healed,
         2) delete from *atopology.ST_FACE* where *FACE_ID* is equal to *F2*,
         4) update the appropriate next left and right face edge IDs and left and right face IDs for edges bounding the faces being healed,

3) update *atopology.ST_FACE* set *MBR* equal to the value of *ST_GetFaceGeometry(atopology,F1).ST_EnvelopeMethod()* where *FACE_ID* is equal to *F1*.

3) The nodes in *atopology.ST_NODE* having a *NODE_ID* value equal to *atopology.ST_EDGE.START_NODE* and *atopology.ST_EDGE.END_NODE* where *EDGE_ID* is equal to *anedge* are not deleted.

4) The choice of which face to modify and which to delete is implementation-dependent.

5) All geometry values in the *ST_NODE*, *ST_EDGE*, and *ST_FACE* views in the same *atopology* schema shall have the same spatial reference system identifier.

X.3.16 ST_GetFaceGeometry Function

Purpose

Return the exact geometry of a face.

Definition

```
CREATE FUNCTION ST_GetFaceGeometry
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    aface INTEGER)
   RETURNS ST_Surface
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The function *ST_GetFaceGeometry(CHARACTER VARYING, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology,*
   b) an INTEGER value *aface.*
2) For the function *ST_GetFaceGeometry(CHARACTER VARYING, INTEGER)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *aface* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If there is no row in the *atopology.ST_FACE* view with a *FACE_ID* value equal to *aface*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent face*.
   d) Otherwise:
      i) let *T* be the table returned by *ST_GetFaceEdges(atopology, aface)*,
      ii) for each row in *T*, let *G* be the *ST_Curve* value equal to *atopology.ST_EDGE.GEOMETRY* where *EDGE_ID* is equal to the value of *edge* from that row,
      iii) let G' be the *ST_Curve* value derived from *G*, corrected for the direction of *edge;* for each value of *G*:

         Case:

         1) if *edge* is greater than 0 (zero), then G' is equal to G,
         2) otherwise, construct a value for G' by reversing the direction of G.

iv) create an *ST_Surface* geometry value *geometry,* constructed from the values of *G'*, connecting them in the order specified by the *sequence* values in *T*,

v) return *geometry*.

X.3.17 ST_InitTopoGeo Procedure

Purpose

Create schema and views for a Topology-Geometry.

Definition

```
CREATE PROCEDURE ST_InitTopoGeo
   (atopology CHARACTER VARYING(ST_MaxTopologyName))
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The procedure *ST_InitTopoGeo(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology.*
2) For the procedure *ST_InitTopoGeo(CHARACTER VARYING)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.SCHEMATA* WHERE *SCHEMA_NAME = 'atopology'* returns a value equal to 1 (one), then an exception condition is raised: *SQL/MM Spatial exception – schema already exists*.
   c) Otherwise:
      i) create a schema with a name equal to *atopology*,
      ii) for schema *atopology*, create *ST_NODE, ST_EDGE,* and *ST_FACE* views in accordance with Clause X.1.2 ST_NODE view, X.1.3 ST_EDGE view, and X.1.4 ST_FACE view,
      iii) insert into *atopology.ST_FACE* values (0, NULL).

X.3.18 ST_CreateTopoGeo Procedure

Purpose

Create a topologically consistent Topology-Geometry from a collection of geometry values.

Definition

```
CREATE PROCEDURE ST_CreateTopoGeo
   (atopology CHARACTER VARYING(ST_MaxTopologyName),
    ageomcollection ST_GeomCollection)
   LANGUAGE SQL
   NOT DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The procedure *ST_CreateTopoGeo(CHARACTER VARYING, ST_GeomCollection)* takes the following input parameters:

   a) a CHARACTER VARYING value *atopology,*

   b) an *ST_GeomCollection* value *ageomcollection.*

2) For the procedure *ST_CreateTopoGeo(CHARACTER VARYING, ST_GeomCollection):*

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If *ageomcollection* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   c) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.SCHEMATA* WHERE *SCHEMA_NAME* = *'atopology'* returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent schema*.

   d) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.VIEWS* WHERE *TABLE_SCHEMA* = *'atopology'* AND *TABLE_NAME* = 'ST_NODE' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

   e) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.VIEWS* WHERE *TABLE_SCHEMA* = *'atopology'* AND *TABLE_NAME* = 'ST_EDGE' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

   f) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.VIEWS* WHERE *TABLE_SCHEMA* = *'atopology'* AND *TABLE_NAME* = 'ST_FACE' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

   g) If SELECT COUNT(*) FROM *atopology.ST_NODE* returns a value greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-empty view*.

h) If SELECT COUNT(*) FROM *atopology.ST_EDGE* returns a value greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-empty view*.

i) If SELECT COUNT(*) FROM *atopology.ST_FACE* returns a value greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-empty view*.

j) Otherwise:

   i) using the geometry values in *ageomcollection*, create the corresponding consistent topology,

   ii) for each face:

      1) let *faceid* be the automatically generated next available face id INTEGER value,

      2) let *mbr* be the *ST_Polygon* value which represents the minimum bounding rectangle of the face,

      3) insert into *atopology.ST_FACE* values(*faceid, mbr),*

   iii) for each node:

      1) let *nodeid* be the automatically generated next available node id INTEGER value,

      2) let *geometry* be the *ST_Point* value which specifies the node location,

      3) case:

         A) if the node is an isolated node, then let *face* be the INTEGER value equal to the face id of the containing face,

         B) otherwise, let *face* be the null value,

      4) insert into *atopology.ST_NODE* values(*nodeid, geometry, face),*

   iv) for each edge:

      1) let *edgeid* be the automatically generated next available edge id INTEGER value,

      2) let *startnode* be the INTEGER value equal to the node id of the node at the start of the edge,

      3) let *endnode* be the INTEGER value equal to the node id of the node at the end of the edge,

      4) let *nextleft* be the INTEGER value equal to the edge id of the next edge of the face on the left (when looking in the direction from its start node to its end node), moving counterclockwise around the face boundary,

      5) let *nextright* be the INTEGER value equal to the edge id of the next edge of the face on the right (when looking in the direction from the start node edge to its end node), moving counterclockwise around the face boundary,

      6) let *leftface* be the INTEGER value equal to the face id of the face on the left side of the edge (when looking in the direction from the start node of the edge to its end node),

      7) let *rightface* be the INTEGER value equal to the face id of the face on the right side of the edge (when looking in the direction from the start node of the edge to its end node),

      8) let *geometry* be the *ST_Curve* value which represents the geometry of the edge, in the same direction as the edge,

      9) insert into *atopology.ST_EDGE* values(*edgeid, startnode, endnode, nextleft, nextright, leftface, rightface, geometry).*

3) All geometry values in the *ST_NODE*, *ST_EDGE*, and *ST_FACE* views in the same *atopology* schema shall have the same spatial reference system identifier.

X.3.19 ST_ValidateTopoGeo Function

Purpose

Return a table containing topological inconsistencies for a Topology-Geometry.

Definition

```
CREATE FUNCTION ST_ValidateTopoGeo
   (atopology CHARACTER VARYING(ST_MaxTopologyName))
   RETURNS TABLE
      (error CHARACTER VARYING(30),
       primitive1 INTEGER,
       primitive2 INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The function *ST_ValidateTopoGeo(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *atopology.*
2) For the function *ST_ValidateTopoGeo(CHARACTER VARYING)*:

   Case:

   a) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELECT COUNT(*) FROM *atopology.ST_NODE* returns a value equal to 0 (zero) and SELECT COUNT(*) FROM *atopology.ST_EDGE* returns a value equal to 0 (zero) and SELECT COUNT(*) FROM *atopology.ST_FACE* returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – empty topology*.
   c) Otherwise,
      i) let T be a table value consisting of the following three columns:
         1) a column *error* of type CHARACTER VARYING, which identifies the type of inconsistency found in *atopology*,
         2) a column *primitive1* of type INTEGER, which contains the node, edge, or face id of the first offending topology primitive,
         2) a column *primitive2* of type INTEGER, which contains the node, edge, or face id of the second offending topology primitive,
      ii) for each pair of coincident nodes:
         1) let *E* be the CHARACTER VARYING value equal to “coincident nodes”,

2) let *N1* be the INTEGER value equal to *atopology.ST_NODE.NODE_ID* for a node in *atopology*,

3) let *G1* be the *ST_Point* value equal to *atopology.ST_NODE.GEOMETRY* where *NODE_ID* is equal to *N1*,

4) let *N2* be the INTEGER value equal to *atopology.ST_NODE.NODE_ID* for another node in *atopology*,

5) let *G2* be the *ST_Point* value equal to *atopology.ST_NODE.GEOMETRY* where *NODE_ID* is equal to *N2*,

6) for all values of *N1* and *N2* such that *G1* is equal to *G2*, insert into *T* values (*E, N1, N2*),

iii) for each node crossed by an edge:

1) let *E* be the CHARACTER VARYING value equal to “edge crossed node”,

2) let *N1* be the INTEGER value equal to *atopology.ST_NODE.NODE_ID* for a node in *atopology*,

3) let *G1* be the *ST_Point* value equal to *atopology.ST_NODE.GEOMETRY* where *NODE_ID* is equal to *N1*,

4) let *E2* be the INTEGER value equal to *atopology.ST_EDGE.EDGE_ID* for an edge in *atopology*,

5) let *G2* be the *ST_Curve* value equal to *atopology.ST_EDGE.GEOMETRY* where *EDGE_ID* is equal to *E2*,

6) for all values of *N1* and *E2* such that the value returned by *G2.ST_Crosses(G1)* is equal to 1 (one), insert into *T* values (*E, N1, E2*),

iv) for each non-simple edge:

1) let *E* be the CHARACTER VARYING value equal to “edge not simple”,

2) let *E1* be the INTEGER value equal to *atopology.ST_EDGE.EDGE_ID* for an edge in *atopology*,

3) let *G1* be the *ST_Curve* value equal to *atopology.ST_EDGE.GEOMETRY* where *EDGE_ID* is equal to *E1*,

4) let *E2* be equal to the null value,

6) for all values of *E1* such that the value returned by *G1.ST_IsSimple()* is equal to 0 (zero), insert into *T* values (*E, E1, E2*),

v) for each edge crossing another edge:

1) let *E* be the CHARACTER VARYING value equal to “edge crosses edge”,

2) let *E1* be the INTEGER value equal to *atopology.ST_EDGE.EDGE_ID* for an edge in *atopology*,

3) let *G1* be the *ST_Curve* value equal to *atopology.ST_EDGE.GEOMETRY* where *EDGE_ID* is equal to *E1*,

4) let *E2* be the INTEGER value equal to *atopology.ST_EDGE.EDGE_ID* for another edge in *atopology*,

5) let *G2* be the *ST_Curve* value equal to *atopology.ST_EDGE.GEOMETRY* where *EDGE_ID* is equal to *E2*,

6) for all values of *E1* and *E2* such that the value returned by *G1.ST_Crosses(G2)* is equal to 1 (one), insert into *T* values (*E, E1, E2*),

vi) for each edge having a geometry with a start point not equal to the geometry of its start node:

1) let *E* be the CHARACTER VARYING value equal to “geometry mis-match”,
2) let *E1* be the INTEGER value equal to *atopology.ST_EDGE.EDGE_ID* for an edge in *atopology*,
3) let *G1* be the *ST_Curve* value equal to *atopology.ST_EDGE.GEOMETRY* where *EDGE_ID* is equal to *E1*,
4) let *N2* be the INTEGER value equal to *atopology.ST_EDGE.START_NODE* where *EDGE_ID* is equal to *E1*,
5) let *G2* be the *ST_Point* value equal to *atopology.ST_NODE.GEOMETRY* where *NODE_ID* is equal to *N2*,
6) for all values of *E1* and *N2* such that the value returned by *G2.ST_Equals(G1.ST_StartPoint())* is not equal to 1 (one), insert into *T* values (*E, E1, N2*),

vii) for each edge having a geometry with an end point not equal to the geometry of its end node:

1) let *E* be the CHARACTER VARYING value equal to “geometry mis-match”,
2) let *E1* be the INTEGER value equal to *atopology.ST_EDGE.EDGE_ID* for an edge in *atopology*,
3) let *G1* be the *ST_Curve* value equal to *atopology.ST_EDGE.GEOMETRY* where *EDGE_ID* is equal to *E1*,
4) let *N2* be the INTEGER value equal to *atopology.ST_EDGE.END_NODE* where *EDGE_ID* is equal to *E1*,
5) let *G2* be the *ST_Point* value equal to *atopology.ST_NODE.GEOMETRY* where *NODE_ID* is equal to *N2*,
6) for all values of *E1* and *N2* such that the value returned by *G2.ST_Equals(G1.ST_EndPoint())* is not equal to 1 (one), insert into *T* values (*E, E1, N2*),

viii) for each face with no edges:

1) let *E* be the CHARACTER VARYING value equal to “face without edges”,
2) let *F1* be the INTEGER value equal to *atopology.ST_FACE.FACE_ID* for a face in *atopology*,
3) let *F2* be equal to the null value,
4) for all values of *F1* such that the value returned by SELECT COUNT(*) FROM *atopology.ST_EDGE* WHERE *LEFT_FACE* = *F1* is equal to 0 (zero) and SELECT COUNT(*) FROM *atopology.ST_EDGE* WHERE *RIGHT_FACE* = *F1* is also equal to 0 (zero), insert into *T* values (*E, F1, F2*),

ix) for each face overlapping another face:

1) let *E* be the CHARACTER VARYING value equal to “face overlaps face”,
2) let *F1* be the INTEGER value equal to *atopology.ST_FACE.FACE_ID* for a face in *atopology*,
3) let *G1* be the *ST_Surface* value returned by *ST_GetFaceGeometry(atopology,F1),*
4) let *F2* be the INTEGER value equal to *atopology.ST_FACE.FACE_ID* for another face in *atopology*,

5) let *G2* be the *ST_Surface* value returned by *ST_GetFaceGeometry(atopology,F2),*
6) for all values of *F1* and *F2* such that the value returned by *G1.ST_Overlaps(G2)* is equal to 1 (one), insert into *T* values (*E, F1, F2*),

x) for each face contained within another face:

1) let *E* be the CHARACTER VARYING value equal to “face within face”,
2) let *F1* be the INTEGER value equal to *atopology.ST_FACE.FACE_ID* for a face in *atopology*,
3) let *G1* be the *ST_Surface* value returned by *ST_GetFaceGeometry(atopology,F1),*
4) let *F2* be the INTEGER value equal to *atopology.ST_FACE.FACE_ID* for another face in *atopology*,
5) let *G2* be the *ST_Surface* value returned by *ST_GetFaceGeometry(atopology,F2),*
6) for all values of *F1* and *F2* such that the value returned by *G1.ST_Within(G2)* is equal to 1 (one), insert into *T* values (*E, F1, F2*),

xi) if all geometries do not have the same spatial reference system identifier:

1) let *E* be the CHARACTER VARYING value equal to “mixed SRIDs”,
2) let *N1* be the INTEGER value equal to *atopology.ST_NODE.NODE_ID* for a node in *atopology*,
3) let *G1* be the *ST_Point* value equal to *atopology.ST_NODE.GEOMETRY* where *NODE_ID* is equal to *N1,*
4) let *S1* be the INTEGER value returned by *G1.ST_SRID()*,
5) let *N2* be the INTEGER value equal to *atopology.ST_NODE.NODE_ID* for another node in *atopology*,
6) let *G2* be the *ST_Point* value equal to *atopology.ST_NODE.GEOMETRY* where *NODE_ID* is equal to *N2,*
7) let *S2* be the INTEGER value returned by *G2.ST_SRID()*,
8) let *E3* be the INTEGER value equal to *atopology.ST_EDGE.EDGE_ID* for an edge in *atopology*,
9) let *G3* be the *ST_Curve* value equal to *atopology.ST_EDGE.GEOMETRY* where *EDGE_ID* is equal to *E3,*
10) let *S3* be the INTEGER value returned by *G3.ST_SRID()*,
11) let *F4* be the INTEGER value equal to *atopology.ST_FACE.FACE_ID* for a face in *atopology*,
12) let *G4* be the *ST_Surface* value equal to *atopology.ST_FACE.MBR* where *FACE_ID* is equal to *F4,*
13) let *S4* be the INTEGER value returned by *G4.ST_SRID()*,
14) if any value of *N2, E3,* or *F4* has a corresponding value of *S2*, *S3*, or *S4*, respectively, such that *S2* is not equal to *S1* or *S3* is not equal to *S1* or *S4* is not equal to *S1*, insert into *T* values (*E,* NULL, NULL).

2.2 Add the following new clause:

X+1 Topology-Network

X+1.1 Topo-Net Network Schema

X+1.1.1 Introduction

The Topo-Net Network Schema views are defined as being in a schema named <*network-name*> enabling these views to be accessed in the same way as any other tables in any other schema. SELECT privilege on all of these views may be granted to individual users so that they can be queried. These views are updated only by the topology functions provided.

An implementation may define objects that are associated with <*network-name*> that are not defined in this Clause. An implementation may also add columns to tables that are defined in this Clause.

All of the topological primitives contained in the views owned by the <*network-name*> schema constitute a single, topologically consistent, topology. Other topologies (e.g., covering a different spatial extent, existing at a different level of abstraction, or associated with a different set of features) can exist in another, independent <*network-name*> schema.

X+1.1.2 ST_NETNODE view

Purpose

Contains the node type of topological primitives (ST_Node) contained in the <network-name> Topology-Network.

Definition

```
CREATE VIEW ST_NETNODE AS
   SELECT NODE_ID, GEOMETRY
      FROM ST_TOPO_NET.ST_NETNODE
```

X+1.1.3 ST_NETLINK view

Purpose

Contains the link type of topological primitives (ST_Link) contained in the <network-name> Topology-Network.

Definition

```
CREATE VIEW ST_NETLINK AS
   SELECT LINK_ID, START_NODE, END_NODE, GEOMETRY
      FROM ST_TOPO_NET.ST_NETLINK
```

X+1.2 Topo-Net Definition Schema

X+1.2.1 Introduction

The only purpose of this Topo-Net Definition Schema is to provide a data model to support ST_Topo-Net views and to assist understanding. The base tables of this Topo-Net Definition Schema are defined as being in a schema named *ST_TOPO_NET*. The table definitions are as complete as the definitional power of ISO/IEC 9075 allows. The table definitions are supplemented with assertions where appropriate. Each description comprises three parts:

1) The function of the definition is stated.
2) The SQL definition of the object is presented as a <table definition>.
3) An explanation of the object.

The specification provides only a model of the base tables that are required, and does not imply that an implementation shall provide the functionality in the manner described in this clause.

X+1.2.2 ST_NETNODE base table

Purpose

Contains the node type of topological primitives (ST_Node) contained in the Topology-Network.

Definition

```
CREATE TABLE ST_NETNODE
   (
   NODE_ID INTEGER NOT NULL,
   GEOMETRY ST_Point,

   CONSTRAINT ST_NETNODE_PRIMARY_KEY PRIMARY KEY(NODE_ID)
   )
```

Description

1) The value of *NODE_ID* is the unique identifier of the node.
2) The value of *GEOMETRY* is the spatial location of the node. For logical networks, *GEOMETRY* is the null value.
3) All non-NULL geometry values in the *ST_NETNODE* and *ST_NETLINK* views in the same *anetwork* schema shall have the same spatial reference system identifier.

X+1.2.3 ST_NETLINK base table

Purpose

Contains the link type of topological primitives (ST_Link) contained in the Topology-Network.

Definition

```
CREATE TABLE ST_NETLINK
   (
   LINK_ID INTEGER NOT NULL,
   START_NODE INTEGER NOT NULL,
   END_NODE INTEGER NOT NULL,
   GEOMETRY ST_Curve,

   CONSTRAINT ST_NETLINK_PRIMARY_KEY PRIMARY KEY(LINK_ID),
   CONSTRAINT START_NODE_EXISTS FOREIGN KEY(START_NODE)
         REFERENCES ST_NETNODE(NODE_ID),
   CONSTRAINT END_NODE_EXISTS FOREIGN KEY(END_NODE)
         REFERENCES ST_NETNODE(NODE_ID)
   )
```

Description

1) The value of *LINK_ID* is the unique identifier of the link.
2) The value of *START_NODE* is the unique identifier of the node at the start of the link.
3) The value of *END_NODE* is the unique identifier of the node at the end of the link.
4) The value of *GEOMETRY* is the geometry of the link. For logical networks, *GEOMETRY* is the null value.
3) All non-NULL geometry values in the *ST_NETNODE* and *ST_NETLINK* views in the same *anetwork* schema shall have the same spatial reference system identifier.

X+1.3 Topo-Net Routines

X+1.3.1 ST_AddIsoNode Function

Purpose

Insert a row into the <network-name>.ST_NETNODE view for an isolated node.

Definition

```
CREATE FUNCTION ST_AddIsoNode
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    apoint ST_Point)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum cardinality of the CHARACTER VARYING network name.

Description

1) The function *ST_AddIsoNode(CHARACTER VARYING, ST_Point)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork,*
   b) an *ST_Point* value *apoint.*
2) For the function *ST_AddIsoNode(CHARACTER VARYING, ST_Point)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) Otherwise:
      i) generate a unique node id INTEGER value *anodeid*,
      ii) insert into the *anetwork.ST_NETNODE* view values (*anodeid*, *apoint*),
      iii) return *anodeid*.
3) All non-NULL geometry values in the *ST_NETNODE* and *ST_NETLINK* views in the same *anetwork* schema shall have the same spatial reference system identifier.

X+1.3.2 ST_MoveIsoNode Procedure

Purpose

Update the <network-name>.ST_NETNODE.GEOMETRY value of an isolated node.

Definition

```
CREATE PROCEDURE ST_MoveIsoNode
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    anode INTEGER,
    apoint ST_Point)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum cardinality of the CHARACTER VARYING network name.

Description

1) The procedure *ST_MoveIsoNode(CHARACTER VARYING, INTEGER, ST_Point)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork,*
   b) an INTEGER value *anode,*
   c) an *ST_Point* value *apoint.*
2) For the procedure *ST_MoveIsoNode(CHARACTER VARYING, INTEGER, ST_Point)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If any *anetwork.ST_NETLINK.START_NODE* or *anetwork.ST_NETLINK.END_NODE* value is equal to *anode*, then an exception condition is raised: *SQL/MM Spatial exception – not isolated node*.
   d) Otherwise, update the *anetwork.ST_NETNODE* view, set the *GEOMETRY* value equal to *apoint* where *NODE_ID* is equal to *anode*.
3) All non-NULL geometry values in the *ST_NETNODE* and *ST_NETLINK* views in the same *anetwork* schema shall have the same spatial reference system identifier.

X+1.3.3 ST_RemoveIsoNode Procedure

Purpose

Delete a row in the <network-name>.ST_NETNODE view corresponding to an isolated node.

Definition

```
CREATE PROCEDURE ST_RemoveIsoNode
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    anode INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum cardinality of the CHARACTER VARYING network name.

Description

1) The procedure *ST_RemoveIsoNode(CHARACTER VARYING, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork,*
   b) an INTEGER value *anode.*
2) For the procedure *ST_RemoveIsoNode(CHARACTER VARYING, INTEGER)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If any *anetwork.ST_NETLINK.START_NODE* or *anetwork.ST_NETLINK.END_NODE* value is equal to *anode*, then an exception condition is raised: *SQL/MM Spatial exception – not isolated node*.
   d) Otherwise, delete from the *anetwork.ST_NETNODE* view where *NODE_ID* equals *anode*.

X+1.3.4 ST_AddLink Function

Purpose

Insert a row for a link into the <network-name>.ST_NETLINK view connecting two existing nodes.

Definition

```
CREATE FUNCTION ST_AddLink
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    anode INTEGER,
    anothernode INTEGER,
    acurve ST_Curve)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum cardinality of the CHARACTER VARYING network name.

Description

1) The function *ST_AddLink(CHARACTER VARYING, INTEGER, INTEGER, ST_Curve)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork,*
   b) an INTEGER value *anode,*
   c) an INTEGER value *anothernode,*
   d) an *ST_Curve* value *acurve.*
2) For the function *ST_AddLink(CHARACTER VARYING, INTEGER, INTEGER, ST_Curve)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *anode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If *anothernode* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   d) If SELECT COUNT(*) FROM *anetwork.ST_NETNODE* WHERE *NODE_ID* is equal to *anode* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent node,*
   e) If SELECT COUNT(*) FROM *anetwork.ST_NETNODE* WHERE *NODE_ID* is equal to *anothernode* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent node,*

f) Let *P1* be the *ST_Point* value equal to *anetwork.ST_NETNODE.GEOMETRY* where *NODE_ID* is equal to *anode*. If *P1* is not equal to the null value and *P1* is not equal to *acurve.ST_StartPoint*, then an exception condition is raised: *SQL/MM Spatial exception – start node not geometry start point*.

g) Let *P2* be the *ST_Point* value equal to *anetwork.ST_NETNODE.GEOMETRY* where *NODE_ID* is equal to *anothernode*. If *P2* is not equal to the null value and *P2* is not equal to *acurve.ST_EndPoint*, then an exception condition is raised: *SQL/MM Spatial exception – end node not geometry end point*.

h) Otherwise:

   i) generate a unique link id INTEGER value *alinkid*,

   ii) insert into the *anetwork.ST_NETLINK* view values (*alinkid, anode, anothernode, acurve*),

   iii) return *alinkid*.

3) All non-NULL geometry values in the *ST_NETNODE* and *ST_NETLINK* views in the same *anetwork* schema shall have the same spatial reference system identifier.

X+1.3.5 ST_ChangeLinkGeom Procedure

Purpose

Update the <network-name>.ST_NETLINK.GEOMETRY value.

Definition

```
CREATE PROCEDURE ST_ChangeLinkGeom
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    alink INTEGER,
    acurve ST_Curve)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum cardinality of the CHARACTER VARYING network name.

Description

1) The procedure *ST_ChangeLinkGeom(CHARACTER VARYING, INTEGER, ST_Curve)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork,*
   b) an INTEGER value *alink,*
   c) an *ST_Curve* value *acurve.*
2) For the procedure *ST_ChangeLinkGeom(CHARACTER VARYING, INTEGER, ST_Curve)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *alink* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If *acurve* is not the null value, then:
      i) let *P1* be the *ST_Point* value equal to *anetwork.ST_NETNODE.GEOMETRY* where *NODE_ID* equals the value of *anetwork.ST_NETLINK.START_NODE* where *LINK_ID* is equal to *alink*. If *P1* is not equal to the null value and *P1* is not equal to *acurve.ST_StartPoint*, then an exception condition is raised: *SQL/MM Spatial exception – start node not geometry start point,*
      ii) let *P2* be the *ST_Point* value equal to *anetwork.ST_NETNODE.GEOMETRY* where *NODE_ID* equals the value of *anetwork.ST_NETLINK.END_NODE* where *LINK_ID* is equal to *anothernode*. If *P2* is not equal to the null value and *P2* is not equal to *acurve.ST_EndPoint*, then an exception condition is raised: *SQL/MM Spatial exception – end node not geometry end point.*

d) Otherwise, update *anetwork.ST_NETLINK*, set the *GEOMETRY* value equal to *acurve* where *LINK_ID* is equal to *alink*.

3) All non-NULL geometry values in the *ST_NETNODE* and *ST_NETLINK* views in the same *anetwork* schema shall have the same spatial reference system identifier.

X+1.3.6 ST_RemoveLink Procedure

Purpose

Delete a row in the <network-name>.ST_NETLINK view corresponding to a link.

Definition

```
CREATE PROCEDURE ST_RemoveLink
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    alink INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum cardinality of the CHARACTER VARYING network name.

Description

1) The procedure *ST_RemoveLink(CHARACTER VARYING, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork,*
   b) an INTEGER value *alink.*
2) For the procedure *ST_RemoveLink(CHARACTER VARYING, INTEGER)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *alink* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) Otherwise, delete from the *anetwork.ST_NETLINK* view where *LINK_ID* equals *alink.*

X+1.3.7 ST_InitTopoNet Procedure

Purpose

Create schema and views for a Topology-Network.

Definition

```
CREATE PROCEDURE ST_InitTopoNet
   (anetwork CHARACTER VARYING(ST_MaxNetworkName))
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum cardinality of the CHARACTER VARYING network name.

Description

1) The procedure *ST_InitTopoNet(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork.*
2) For the procedure *ST_InitTopoNet(CHARACTER VARYING)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.SCHEMATA* WHERE *SCHEMA_NAME = 'anetwork'* returns a value equal to 1 (one), then an exception condition is raised: *SQL/MM Spatial exception – schema already exists*.
   c) Otherwise:
      i) create a schema with a name equal to *anetwork*,
      ii) for schema *anetwork*, create *ST_NETNODE* and *ST_NETLINK* views in accordance with Clause X+1.1.2 ST_NETNODE view and X+1.1.3 ST_NETLINK view.

X+1.3.8 ST_NewLogLinkSplit Function

Purpose

Split a link in a logical network by creating a new node along an existing link, deleting the original link and replacing it with two new links.

Definition

```
CREATE FUNCTION ST_NewLogLinkSplit
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    alink INTEGER)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum cardinality of the CHARACTER VARYING network name.

Description

1) The function *ST_NewLogLinkSplit(CHARACTER VARYING, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork,*
   b) an INTEGER value *alink.*
2) For the function *ST_NewLogLinkSplit(CHARACTER VARYING, INTEGER)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *alink* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If there is no row in the *anetwork*.*ST_NETLINK* view with an *LINK_ID* value equal to *alink*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent link*.
   d) If the value of *anetwork*.*ST_NETLINK.GEOMETRY* where *LINK_ID* is equal to *alink* is not equal to the NULL value, then an exception condition is raised: *SQL/MM Spatial exception – not a logical link*.
   e) Otherwise:
      i) generate a unique node id INTEGER value *newnode,*
      ii) insert into the *anetwork.ST_NETNODE* view values (*newnode,* NULL),
      iii) select from *anetwork.ST_NETLINK START_NODE* and *END_NODE* into *oldstart* and *oldend* where *LINK_ID* is equal to *alink*,
      iv) delete from *anetwork.ST_NETLINK* where *LINK_ID* is equal to *alink*,

v) generate two unique link id INTEGER values *newlink1* and *newlink2,*

vi) insert into *anetwork.ST_NETLINK* values(*newlink1, oldstart, newnode,* NULL*),*

vii) insert into *anetwork.ST_NETLINK* values(*newlink2, newnode, oldend,* NULL*),*

viii) return *newnode,*

3) Both new links have the same direction as the link being split.

4) To determine the two new link ID values, select LINK_ID from *anetwork.ST_NETLINK* where *START_NODE* or *END_NODE* is equal to *newnode*.

X+1.3.9 ST_ModLogLinkSplit Function

Purpose

Split a link in a logical network by creating a new node along an existing link, modifying the original link and adding a new link.

Definition

```
CREATE FUNCTION ST_ModLogLinkSplit
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    alink INTEGER)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum cardinality of the CHARACTER VARYING network name.

Description

1) The function *ST_ModLogLinkSplit(CHARACTER VARYING, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork,*
   b) an INTEGER value *alink.*
2) For the function *ST_ModLogLinkSplit(CHARACTER VARYING, INTEGER)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *alink* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If there is no row in the *anetwork*.*ST_NETLINK* view with an *LINK_ID* value equal to *alink*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent link*.
   d) If the value of *anetwork*.*ST_NETLINK.GEOMETRY* where *LINK_ID* is equal to *alink* is not equal to the null value, then an exception condition is raised: *SQL/MM Spatial exception – not a logical link*.
   e) Otherwise:
      i) generate a unique node id INTEGER value *newnode,*
      ii) insert into the *anetwork.ST_NETNODE* view values (*newnode,* NULL),
      iii) select from *atopology.ST_NETLINK END_NODE* into *oldend* where *LINK_ID* is equal to *alink*,
      iv) generate a unique link id INTEGER value *newlink,*

v) update *anetwork.ST_NETLINK* where *LINK_ID* is equal to *alink* set the *END_NODE* value equal to *newnode,*

vi) insert into *anetwork.ST_NETLINK* values(*newlink, newnode, oldend,* NULL*),*

vii) return *newnode.*

3) The new and modified links have the same direction as the link being split.

4) To determine the new link ID value, select *LINK_ID* from *anetwork.ST_NETLINK* where *START_NODE* is equal to *newnode*.

X+1.3.10 ST_NewGeoLinkSplit Function

Purpose

Split a link in a network with geometry by creating a new node along an existing link, deleting the original link and replacing it with two new links.

Definition

```
CREATE FUNCTION ST_NewGeoLinkSplit
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    alink INTEGER,
    apoint ST_Point)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum cardinality of the CHARACTER VARYING network name.

Description

1) The function *ST_NewGeoLinkSplit(CHARACTER VARYING, INTEGER, ST_Point)* takes the following input parameters:

   a) a CHARACTER VARYING value *anetwork,*

   b) an INTEGER value *alink,*

   c) an *ST_Point* value *apoint.*

2) For the function *ST_NewGeoLinkSplit(CHARACTER VARYING, INTEGER, ST_Point)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If *alink* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   c) If *apoint* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   d) If there is no row in the *anetwork*.*ST_NETLINK* view with an *LINK_ID* value equal to *alink*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent link*.

   e) Let *G* be equal to the *ST_Curve* value of *anetwork*.*ST_NETLINK.GEOMETRY* where *LINK_ID* is equal to *alink*. If *G* is equal to the null value, then an exception condition is raised: *SQL/MM Spatial exception – link has null geometry*.

   f) If *apoint.ST_Within(G)* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – point not on link*.

g) Otherwise:

i) generate a unique node id INTEGER value *newnode,*

ii) insert into the *anetwork.ST_NETNODE* view values (*newnode, apoint*),

iii) select from *anetwork.ST_NETLINK START_NODE, END_NODE,* and *GEOMETRY* into *oldstart, oldend,* and *oldgeom* where *LINK_ID* is equal to *alink*,

iv) delete from *anetwork.ST_NETLINK* where *LINK_ID* is equal to *alink*,

v) generate two unique link id INTEGER values *newlink1* and *newlink2,*

vi) create two new *ST_Curve* values, *newgeom1* and *newgeom2* from *oldgeom*, breaking *oldgeom* at the location defined by *apoint*,

vii) insert into *anetwork.ST_NETLINK* values(*newlink1, oldstart, newnode, newgeom1),*

viii) insert into *anetwork.ST_NETLINK* values(*newlink2, newnode, oldend, newgeom2),*

ix) return *newnode.*

3) Both new links have the same direction as the link being split.

4) To determine the two new link ID values, select *LINK_ID* from *anetwork.ST_NETLINK* where *START_NODE* or *END_NODE* is equal to *newnode*.

5) All non-NULL geometry values in the *ST_NETNODE* and *ST_NETLINK* views in the same *anetwork* schema shall have the same spatial reference system identifier.

X+1.3.11 ST_ModGeoLinkSplit Function

Purpose

Split a link in a network with geometry by creating a new node along an existing link, modifying the original link and adding a new link.

Definition

```
CREATE FUNCTION ST_ModGeoLinkSplit
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    alink INTEGER,
    apoint ST_Point)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum cardinality of the CHARACTER VARYING network name.

Description

1) The function *ST_ModGeoLinkSplit(CHARACTER VARYING, INTEGER, ST_Point)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork,*
   b) an INTEGER value *alink,*
   c) an *ST_Point* value *apoint.*
2) For the function *ST_ModGeoLinkSplit(CHARACTER VARYING, INTEGER, ST_Point)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *alink* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If *apoint* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   d) If there is no row in the *anetwork*.*ST_NETLINK* view with an *LINK_ID* value equal to *alink*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent link*.
   e) Let *G* be equal to the *ST_Curve* value of *anetwork*.*ST_NETLINK.GEOMETRY* where *LINK_ID* is equal to *alink*. If *G* is equal to the null value, then an exception condition is raised: *SQL/MM Spatial exception – link has null geometry*.
   f) If *apoint.ST_Within(G)* is equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – point not on link*.

g) Otherwise:

i) generate a unique node id INTEGER value *newnode,*

ii) insert into the *anetwork.ST_NETNODE* view values (*newnode, apoint*),

iii) select from *anetwork.ST_NETLINK END_NODE* and *GEOMETRY* into *oldend* and *oldgeom* where *LINK_ID* is equal to *alink*,

iv) generate a unique link id INTEGER value *newlink,*

v) create two new *ST_Curve* values, *newgeom1* and *newgeom2* from *oldgeom*, breaking *oldgeom* at the location defined by *apoint*,

vi) update *anetwork.ST_NETLINK* where *LINK_ID* is equal to *alink:*

1) set the *END_NODE* value equal to *newnode,*

2) set the *GEOMETRY* value equal to *newgeom1,*

vii) insert into *anetwork.ST_NETLINK* values(*newlink, newnode, oldend, newgeom2),*

viii) return *newnode.*

3) The new and modified links have the same direction as the link being split.

4) To determine the new link ID value, select *LINK_ID* from anetwork.*ST_NETLINK* where *START_NODE* is equal to *newnode*.

5) All non-NULL geometry values in the *ST_NETNODE* and *ST_NETLINK* views in the same *anetwork* schema shall have the same spatial reference system identifier.

X+1.3.12 ST_NewLinkHeal Function

Purpose

Heal two links by deleting the node connecting them, deleting both links, and replacing them with a new link whose direction is the same as the first link provided.

Definition

```
CREATE FUNCTION ST_NewLinkHeal
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    alink INTEGER,
    anotherlink INTEGER)
   RETURNS INTEGER
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum cardinality of the CHARACTER VARYING network name.

Description

1) The function *ST_NewLinkHeal(CHARACTER VARYING, INTEGER, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork,*
   b) an INTEGER value *alink,*
   c) an INTEGER value *anotherlink.*
2) For the function *ST_NewLinkHeal(CHARACTER VARYING, INTEGER, INTEGER)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *alink* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If *anotherlink* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   d) If there is no row in the *anetwork*.*ST_NETLINK* view with an *LINK_ID* value equal to *alink*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent link*.
   e) If there is no row in the *anetwork*.*ST_NETLINK* view with an *LINK_ID* value equal to *anotherlink*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent link*.
   f) Check if links are connected:
      i) let *COMMONNODE* be an INTEGER value equal to 0 (zero),
      ii) let *CASE* be an INTEGER value,

iii) let S1 and E1 be INTEGER values equal to the *anetwork*.*ST_NETLINK.START_NODE* and *END_NODE* values, respectively, where *LINK_ID* is equal to *alink*,

iv) let S2 and E2 be INTEGER values equal to the *anetwork*.*ST_NETLINK.START_NODE* and *END_NODE* values, respectively, where *LINK_ID* is equal to *anotherlink*,

v) case:

1) if E1 is equal to S2, then set *CASE* equal to 1 (one) and *COMMONNODE* equal to E1,

2) if E1 is equal to E2, then set *CASE* equal to 2 and *COMMONNODE* equal to E1,

3) if S1 is equal to S2, then set *CASE* equal to 3 and *COMMONNODE* equal to S1,

4) if S1 is equal to E2, then set *CASE* equal to 4 and *COMMONNODE* equal to S1,

5) otherwise, an exception condition is raised: *SQL/MM Spatial exception – non-connected links,*

g) If *COMMONNODE* is equal to the *anetwork.ST_NETLINK.START_NODE* or *anetwork*.*ST_NETLINK.END_NODE* value for any link other than the link whose *LINK_ID* is equal to either *alink* or *anotherlink*, then an exception condition is raised: *SQL/MM Spatial exception – other links connected*,

h) Otherwise:

i) delete from the *anetwork.ST_NETNODE* view where *NODE_ID* is equal to *COMMONNODE*,

ii) select from *anetwork.ST_NETLINK START_NODE, END_NODE,* and *GEOMETRY* into *oldstart1, oldend1* and *oldgeom1* where *LINK_ID* is equal to *alink*,

iii) select from *anetwork.ST_NETLINK START_NODE, END_NODE,* and *GEOMETRY* into *oldstart2, oldend2,* and *oldgeom2* where *LINK_ID* is equal to *anotherlink*,

iv) delete from *anetwork.ST_NETLINK* where *LINK_ID* is equal to *alink*,

v) delete from *anetwork.ST_NETLINK* where *LINK_ID* is equal to *anotherlink*,

vi) generate a unique link id INTEGER value *newlink,*

vii) case:

1) if *CASE* is equal to 1 (one) then:

A) case:

I) if *oldgeom1* or *oldgeom2* is equal to the null value, then set *newgeom* equal to the null value,

II) otherwise, create a new *ST_Curve* value *newgeom* from *oldgeom1* and *oldgeom2*, by connecting the end of *oldgeom1* to the start of *oldgeom2*,

B) insert into *anetwork.ST_NETLINK* values(*newlink, oldstart1, oldend2, newgeom*),

2) if *CASE* is equal to 2 then:

A) case:

I) if *oldgeom1* or *oldgeom2* is equal to the null value, then set *newgeom* equal to the null value,

II) otherwise, create a new *ST_Curve* value *newgeom* from *oldgeom1 and oldgeom2*, by reversing the direction of *oldgeom2* and connecting it to the end of *oldgeom1*,

B) insert into *anetwork.ST_NETLINK* values(*newlink, oldstart1, oldstart2, newgeom*),

3) if *CASE* is equal to 3 then:

A) case:

I) if *oldgeom1* or *oldgeom2* is equal to the null value, then set *newgeom* equal to the null value,

II) otherwise, create a new *ST_Curve* value *newgeom* from *oldgeom1 and oldgeom2*, by reversing the direction of *oldgeom2* and connecting *oldgeom1* to the new end,

B) insert into *anetwork.ST_NETLINK* values(*newlink, oldend2, oldend1, newgeom*),

4) if *CASE* is equal to 4 then:

A) case:

I) if *oldgeom1* or *oldgeom2* is equal to the null value, then set *newgeom* equal to the null value,

II) otherwise, create a new *ST_Curve* value *newgeom* from *oldgeom1 and oldgeom2*, by connecting *oldgeom1* to the end of *oldgeom2*,

B) insert into *anetwork.ST_NETLINK* values(*newlink, oldstart2, oldend1, newgeom*),

viii) return *newlink*.

3) The direction of the new link shall be the same as the direction of the first link supplied.

4) All non-NULL geometry values in the *ST_NETNODE* and *ST_NETLINK* views in the same *anetwork* schema shall have the same spatial reference system identifier.

X+1.3.13 ST_ModLinkHeal Procedure

Purpose

Heal two links by deleting the node connecting them, modifying the first link provided, and deleting the second link.

Definition

```
CREATE PROCEDURE ST_ModLinkHeal
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    alink INTEGER,
    anotherlink INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum cardinality of the CHARACTER VARYING network name.

Description

1) The procedure *ST_ModLinkHeal(CHARACTER VARYING, INTEGER, INTEGER)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork,*
   b) an INTEGER value *alink,*
   c) an INTEGER value *anotherlink.*
2) For the procedure *ST_ModLinkHeal(CHARACTER VARYING, INTEGER, INTEGER)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If *alink* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   c) If *anotherlink* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   d) If there is no row in the *anetwork*.*ST_NETLINK* view with an *LINK_ID* value equal to *alink*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent link*.
   e) If there is no row in the *anetwork*.*ST_NETLINK* view with an *LINK_ID* value equal to *anotherlink*, then an exception condition is raised: *SQL/MM Spatial exception – non-existent link*.
   f) Check if links are connected:
      i) let *COMMONNODE* be an INTEGER value equal to 0 (zero),
      ii) let *CASE* be an INTEGER value,

iii) let S1 and E1 be INTEGER values equal to the *anetwork*.*ST_NETLINK.START_NODE* and *END_NODE* values, respectively, where *LINK_ID* is equal to *alink*,

iv) let S2 and E2 be INTEGER values equal to the *anetwork*.*ST_NETLINK.START_NODE* and *END_NODE* values, respectively, where *LINK_ID* is equal to *anotherlink*,

v) case:

1) if E1 is equal to S2, then set *CASE* equal to 1 (one) and *COMMONNODE* equal to E1,

2) if E1 is equal to E2, then set *CASE* equal to 2 and *COMMONNODE* equal to E1,

3) if S1 is equal to S2, then set *CASE* equal to 3 and *COMMONNODE* equal to S1,

4) if S1 is equal to E2, then set *CASE* equal to 4 and *COMMONNODE* equal to S1,

5) otherwise, an exception condition is raised: *SQL/MM Spatial exception – non-connected links,*

g) If *COMMONNODE* is equal to the *anetwork.ST_NETLINK.START_NODE* or *anetwork*.*ST_NETLINK.END_NODE* value for any link other than the link whose *LINK_ID* is equal to either *alink* or *anotherlink*, then an exception condition is raised: *SQL/MM Spatial exception – other links connected*,

h) Otherwise:

i) delete from the *anetwork.ST_NETNODE* view where *NODE_ID* is equal to *COMMONNODE*,

ii) select from *anetwork.ST_NETLINK GEOMETRY* into *oldgeom1* where *LINK_ID* is equal to *alink*,

iii) select from *anetwork.ST_NETLINK START_NODE, END_NODE,* and *GEOMETRY* into *oldstart2, oldend2,* and *oldgeom2* where *LINK_ID* is equal to *anotherlink*,

iv) delete from *anetwork.ST_NETLINK* where *LINK_ID* is equal to *anotherlink*,

v) case:

1) if *CASE* is equal to 1 (one) then:

A) case:

I) if *oldgeom1* or *oldgeom2* is equal to the null value, then set *newgeom* equal to the null value,

II) otherwise, create a new *ST_Curve* value *newgeom* from *oldgeom1 and oldgeom2*, by connecting the end of *oldgeom1* to the start of *oldgeom2*,

B) update *anetwork*.*ST_NETLINK* where *LINK_ID* is equal to *alink*:

I) set the *END_NODE* value equal to *oldend2,*

II) set the *GEOMETRY* value equal to *newgeom,*

2) if *CASE* is equal to 2 then:

A) case:

I) if *oldgeom1* or *oldgeom2* is equal to the null value, then set *newgeom* equal to the null value,

II) otherwise, create a new *ST_Curve* value *newgeom* from *oldgeom1 and oldgeom2*, by reversing the direction of *oldgeom2* and connecting it to the end of *oldgeom1*,

B) update *anetwork*.*ST_NETLINK* where *LINK_ID* is equal to *alink*:

I) set the *END_NODE* value equal to *oldstart2,*

II) set the *GEOMETRY* value equal to *newgeom,*

3) if *CASE* is equal to 3 then:

A) case:

I) if *oldgeom1* or *oldgeom2* is equal to the null value, then set *newgeom* equal to the null value,

II) otherwise, create a new *ST_Curve* value *newgeom* from *oldgeom1 and oldgeom2*, by reversing the direction of *oldgeom2* and connecting *oldgeom1* to the new end,

B) update *anetwork*.*ST_NETLINK* where *LINK_ID* is equal to *alink*:

I) set the *START_NODE* value equal to *oldend2,*

II) set the *GEOMETRY* value equal to *newgeom,*

4) if *CASE* is equal to 4 then:

A) case:

I) if *oldgeom1* or *oldgeom2* is equal to the null value, then set *newgeom* equal to the null value,

II) otherwise, create a new *ST_Curve* value *newgeom* from *oldgeom1 and oldgeom2*, by connecting *oldgeom1* to the end of *oldgeom2*,

B) update *anetwork*.*ST_NETLINK* where *LINK_ID* is equal to *alink*:

I) set the *START_NODE* value equal to *oldstart2,*

II) set the *GEOMETRY* value equal to *newgeom.*

3) All non-NULL geometry values in the *ST_NETNODE* and *ST_NETLINK* views in the same *anetwork* schema shall have the same spatial reference system identifier.

X+1.3.14 ST_LogiNetFromTGeo Procedure

Purpose

Create a logical Topology-Network from a Topology-Geometry.

Definition

```
CREATE PROCEDURE ST_LogiNetFromTGeo
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    atopology CHARACTER VARYING(ST_MaxTopologyName))
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum cardinality of the CHARACTER VARYING network name.

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The procedure *ST_LogiNetFromTGeo(CHARACTER VARYING, CHARACTER VARYING)* takes the following input parameters:

   a) a CHARACTER VARYING value *anetwork,*

   b) a CHARACTER VARYING value *atopology.*

2) For the procedure *ST_LogiNetFromTGeo(CHARACTER VARYING, CHARACTER VARYING):*

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   c) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.SCHEMATA* WHERE *SCHEMA_NAME* = *'anetwork'* returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent schema*.

   d) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.SCHEMATA* WHERE *SCHEMA_NAME* = *'atopology'* returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent schema*.

   e) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.VIEWS* WHERE *TABLE_SCHEMA* = *'anetwork'* AND *TABLE_NAME* = 'ST_NETNODE' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

   f) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.VIEWS* WHERE *TABLE_SCHEMA* = *'anetwork'* AND *TABLE_NAME* = 'ST_NETLINK' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

g) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.VIEWS* WHERE *TABLE_SCHEMA* = *'atopology'* AND *TABLE_NAME* = 'ST_NODE' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

h) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.VIEWS* WHERE *TABLE_SCHEMA* = *'atopology'* AND *TABLE_NAME* = 'ST_EDGE' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

i) If SELECT COUNT(*) FROM *anetwork.ST_NETNODE* returns a value greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-empty view*.

j) If SELECT COUNT(*) FROM *anetwork.ST_NETLINK* returns a value greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-empty view*.

k) Otherwise:

   i) for nodes in *atopology*, insert into *anetwork.ST_NETNODE* select *NODE_ID*, NULL from *atopology.ST_NODE*,

   ii) for edges in *atopology*, insert into *anetwork.ST_NETLINK* select *EDGE_ID*, *START_NODE, END_NODE,* NULL from *atopology.ST_EDGE*.

X+1.3.15 ST_SpatNetFromTGeo Procedure

Purpose

Create a spatial Topology-Network from a Topology-Geometry.

Definition

```
CREATE PROCEDURE ST_SpatNetFromTGeo
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    atopology CHARACTER VARYING(ST_MaxTopologyName))
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum cardinality of the CHARACTER VARYING network name.

1) *ST_MaxTopologyName* is the implementation-defined maximum cardinality of the CHARACTER VARYING topology name.

Description

1) The procedure *ST_SpatNetFromTGeo(CHARACTER VARYING, CHARACTER VARYING)* takes the following input parameters:

   a) a CHARACTER VARYING value *anetwork,*

   b) a CHARACTER VARYING value *atopology.*

2) For the procedure *ST_SpatNetFromTGeo(CHARACTER VARYING, CHARACTER VARYING):*

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If *atopology* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   c) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.SCHEMATA* WHERE *SCHEMA_NAME* = *'anetwork'* returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent schema*.

   d) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.SCHEMATA* WHERE *SCHEMA_NAME* = *'atopology'* returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent schema*.

   e) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.VIEWS* WHERE *TABLE_SCHEMA* = *'anetwork'* AND *TABLE_NAME* = 'ST_NETNODE' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

   f) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.VIEWS* WHERE *TABLE_SCHEMA* = *'anetwork'* AND *TABLE_NAME* = 'ST_NETLINK' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

g) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.VIEWS* WHERE *TABLE_SCHEMA* = *'atopology'* AND *TABLE_NAME* = 'ST_NODE' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

h) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.VIEWS* WHERE *TABLE_SCHEMA* = *'atopology'* AND *TABLE_NAME* = 'ST_EDGE' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

i) If SELECT COUNT(*) FROM *anetwork.ST_NETNODE* returns a value greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-empty view*.

j) If SELECT COUNT(*) FROM *anetwork.ST_NETLINK* returns a value greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-empty view*.

k) Otherwise:

   i) for nodes in *atopology*, insert into *anetwork.ST_NETNODE* select *NODE_ID*, *GEOMETRY* from *atopology.ST_NODE*,

   ii) for edges in *atopology*, insert into *anetwork.ST_NETLINK* select *EDGE_ID*, *START_NODE, END_NODE, GEOMETRY* from *atopology.ST_EDGE*.

3) All non-NULL geometry values in the *ST_NETNODE* and *ST_NETLINK* views in the same *anetwork* schema shall have the same spatial reference system identifier.

X+1.3.16 ST_SpatNetFromGeom Procedure

Purpose

Create a Topology-Network from a collection of geometry values.

Definition

```
CREATE PROCEDURE ST_SpatNetFromGeom
   (anetwork CHARACTER VARYING(ST_MaxNetworkName),
    ageomcollection ST_GeomCollection)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum cardinality of the CHARACTER VARYING network name.

Description

1) The procedure *ST_SpatNetFromGeom(CHARACTER VARYING, ST_GeomCollection)* takes the following input parameters:

   a) a CHARACTER VARYING value *anetwork,*

   b) an *ST_GeomCollection* value *ageomcollection.*

2) For the procedure *ST_SpatNetFromGeom(CHARACTER VARYING, ST_GeomCollection):*

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   b) If *ageomcollection* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.

   c) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.SCHEMATA* WHERE *SCHEMA_NAME* = *'anetwork'* returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent schema*.

   d) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.VIEWS* WHERE *TABLE_SCHEMA* = *'anetwork'* AND *TABLE_NAME* = 'ST_NETNODE' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

   e) If SELECT COUNT(*) FROM *INFORMATION_SCHEMA.VIEWS* WHERE *TABLE_SCHEMA* = *'anetwork'* AND *TABLE_NAME* = 'ST_NETLINK' returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-existent view*.

   f) If SELECT COUNT(*) FROM *anetwork.ST_NETNODE* returns a value greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-empty view*.

   g) If SELECT COUNT(*) FROM *anetwork.ST_NETLINK* returns a value greater than 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – non-empty view*.

h) Otherwise:

i) using the geometry values in *ageomcollection*, create a corresponding set of network primitives (nodes and links),

ii) for each node,

1) let *nodeid* be the automatically generated next available node id INTEGER value,

2) let *geometry* be the *ST_Point* value which specifies the node location,

3) insert into *anetwork.ST_NETNODE* values(*nodeid, geometry),*

iii) for each link,

1) let *linkid* be the automatically generated next available link id INTEGER value,

2) let *startnode* be the INTEGER value equal to the node id of the node at the start of the link,

3) let *endnode* be the INTEGER value equal to the node id of the node at the end of the link,

4) let *geometry* be the *ST_Curve* value which represents the geometry of the link, in the same direction as the link,

5) insert into *anetwork.ST_NETLINK* values(*linkid, startnode, endnode, geometry).*

3) All non-NULL geometry values in the *ST_NETNODE* and *ST_NETLINK* views in the same *anetwork* schema shall have the same spatial reference system identifier.

X+1.3.17 ST_ValidLogicalNet Function

Purpose

Return a table containing network inconsistencies for a logical Topology-Network.

Definition

```
CREATE FUNCTION ST_ValidLogicalNet
   (anetwork CHARACTER VARYING(ST_MaxNetworkName))
   RETURNS TABLE
      (error CHARACTER VARYING(30),
       primitive1 INTEGER,
       primitive2 INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum cardinality of the CHARACTER VARYING network name.

Description

1) The function *ST_ValidLogicalNet(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork.*
2) For the function *ST_ValidLogicalNet(CHARACTER VARYING)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELECT COUNT(*) FROM *anetwork.ST_NETNODE* returns a value equal to 0 (zero) and SELECT COUNT(*) FROM *anetwork.ST_NETLINK* returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – empty network*.
   c) Otherwise,
      i) let T be a table value consisting of the following three columns:
         1) a column *error* of type CHARACTER VARYING, which identifies the type of inconsistency found in *anetwork*,
         2) a column *primitive1* of type INTEGER, which contains the node or link id of the first offending network primitive,
         2) a column *primitive2* of type INTEGER, which contains the node or link id of the second offending network primitive,
      ii) for each node with a non-NULL geometry value:
         1) let *E* be the CHARACTER VARYING value equal to “node has geometry”,

2) let *N1* be the INTEGER value equal to *anetwork.ST_NETNODE.NODE_ID* for a node in *anetwork*,

3) let *G1* be the *ST_Point* value equal to *anetwork.ST_NETNODE.GEOMETRY* where *NODE_ID* is equal to *N1*,

4) let *N2* be equal to the null value,

5) for all values of *N1* such that *G1* is not equal to the null value, insert into *T* values (*E, N1, N2*),

iii) for each link with a non-NULL geometry value:

1) let *E* be the CHARACTER VARYING value equal to “link has geometry”,

2) let *L1* be the INTEGER value equal to *anetwork.ST_NETLINK.LINK_ID* for a link in *anetwork*,

3) let *G1* be the *ST_Curve* value equal to *anetwork.ST_NETLINK.GEOMETRY* where *LINK_ID* is equal to *L1*,

4) let *L2* be equal to the null value,

5) for all values of *L1* such that *G1* is not equal to the null value, insert into *T* values (*E, L1, L2*),

X+1.3.18 ST_ValidSpatialNet Function

Purpose

Return a table containing network inconsistencies for a spatial Topology-Network.

Definition

```
CREATE FUNCTION ST_ValidSpatialNet
   (atopology CHARACTER VARYING(ST_MaxNetworkName))
   RETURNS TABLE
      (error CHARACTER VARYING(30),
       primitive1 INTEGER,
       primitive2 INTEGER)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- See Description
      --
   END
```

Definitional Rules

1) *ST_MaxNetworkName* is the implementation-defined maximum cardinality of the CHARACTER VARYING network name.

Description

1) The function *ST_ValidSpatialNet(CHARACTER VARYING)* takes the following input parameters:
   a) a CHARACTER VARYING value *anetwork.*
2) For the function *ST_ValidSpatialNet(CHARACTER VARYING)*:

   Case:

   a) If *anetwork* is the null value, then an exception condition is raised: *SQL/MM Spatial exception – null argument*.
   b) If SELECT COUNT(*) FROM *anetwork.ST_NETNODE* returns a value equal to 0 (zero) and SELECT COUNT(*) FROM *anetwork.ST_NETLINK* returns a value equal to 0 (zero), then an exception condition is raised: *SQL/MM Spatial exception – empty network*.
   c) Otherwise,
      i) let T be a table value consisting of the following three columns:
         1) a column *error* of type CHARACTER VARYING, which identifies the type of inconsistency found in *anetwork*,
         2) a column *primitive1* of type INTEGER, which contains the node or link id of the first offending network primitive,
         2) a column *primitive2* of type INTEGER, which contains the node or link id of the second offending network primitive,
      ii) for each node with a null geometry value:
         1) let *E* be the CHARACTER VARYING value equal to “missing node geometry”,

2) let *N1* be the INTEGER value equal to *anetwork.ST_NETNODE.NODE_ID* for a node in *anetwork*,
3) let *G1* be the *ST_Point* value equal to *anetwork.ST_NETNODE.GEOMETRY* where *NODE_ID* is equal to *N1*,
4) let *N2* be equal to the null value,
5) for all values of *N1* such that *G1* is equal to the null value, insert into *T* values (*E, N1, N2*),

iii) for each link with a null geometry value:

1) let *E* be the CHARACTER VARYING value equal to “missing link geometry”,
2) let *L1* be the INTEGER value equal to *anetwork.ST_NETLINK.LINK_ID* for a link in *anetwork*,
3) let *G1* be the *ST_Curve* value equal to *anetwork.ST_NETLINK.GEOMETRY* where *LINK_ID* is equal to *L1*,
4) let *L2* be equal to the null value,
5) for all values of *L1* such that *G1* is equal to the null value, insert into *T* values (*E, L1, L2*),

iv) for each link having a geometry with a start point not equal to the geometry of its start node:

1) let *E* be the CHARACTER VARYING value equal to “geometry start mis-match”,
2) let *L1* be the INTEGER value equal to *anetwork.ST_NETLINK_ID* for a link in *anetwork*,
3) let *G1* be the *ST_Curve* value equal to *anetwork.ST_NETLINK.GEOMETRY* where *LINK_ID* is equal to *L1*,
4) let *N2* be the INTEGER value equal to *anetwork.ST_NETLINK.START_NODE* where *LINK_ID* is equal to *L1*,
5) let *G2* be the *ST_Point* value equal to *anetwork.ST_NETNODE.GEOMETRY* where *NODE_ID* is equal to *N2*,
6) for all values of *L1* and *N2* such that the value returned by *G2.ST_Equals(G1.ST_StartPoint())* is not equal to 1 (one), insert into *T* values (*E, L1, N2*),

v) for each link having a geometry with an end point not equal to the geometry of its end node:

1) let *E* be the CHARACTER VARYING value equal to “geometry end mis-match”,
2) let *L1* be the INTEGER value equal to *anetwork.ST_NETLINK_ID* for a link in *anetwork*,
3) let *G1* be the *ST_Curve* value equal to *anetwork.ST_NETLINK.GEOMETRY* where *LINK_ID* is equal to *L1*,
4) let *N2* be the INTEGER value equal to *anetwork.ST_NETLINK.START_NODE* where *LINK_ID* is equal to *L1*,
5) let *G2* be the *ST_Point* value equal to *anetwork.ST_NETNODE.GEOMETRY* where *NODE_ID* is equal to *N2*,
6) for all values of *L1* and *N2* such that the value returned by *G2.ST_Equals(G1.ST_EndPoint())* is not equal to 1 (one), insert into *T* values (*E, L1, N2*),

vi) if all geometries do not have the same spatial reference system identifier:

1) let *E* be the CHARACTER VARYING value equal to “mixed SRIDs”,
2) let *N1* be the INTEGER value equal to *anetwork.ST_NETNODE.NODE_ID* for a node in *anetwork*,
3) let *G1* be the *ST_Point* value equal to *anetwork.ST_NETNODE.GEOMETRY* where *NODE_ID* is equal to *N1,*
4) let *S1* be the INTEGER value returned by *G1.ST_SRID()*,
5) let *N2* be the INTEGER value equal to *anetwork.ST_NETNODE.NODE_ID* for another node in *anetwork*,
6) let *G2* be the *ST_Point* value equal to *anetwork.ST_NETNODE.GEOMETRY* where *NODE_ID* is equal to *N2,*
7) let *S2* be the INTEGER value returned by *G2.ST_SRID()*,
8) let *L3* be the INTEGER value equal to *anetwork.ST_NETLINK.LINK_ID* for a link in *anetwork*,
9) let *G3* be the *ST_Curve* value equal to *anetwork.ST_NETLINK.GEOMETRY* where *LINK_ID* is equal to *L3,*
10) let *S3* be the INTEGER value returned by *G3.ST_SRID()*,
11) if any value of *N2* or *L3* has a corresponding value of *S2* or *S3*, respectively, such that *S2* is not equal to *S1* or *S3* is not equal to *S1*, insert into *T* values (*E,* NULL, NULL).

3 Checklist

Concepts	Yes
Static methods	Yes
Constructor methods	Yes
Cast definitions	No
Ordering definitions	No
SQL Transforms definitions	No
Conformance Clause	*
New Status Codes	*
Closing Possible Problems	Yes
Any new Possible Problems	No
18-Character identifiers	Yes
Full data type names for basetypes	Yes
Implementation-defined elements	No
Implementation-defined Meta-variables	*
Implementation-dependent elements	No
Implementation-dependent Meta-variables	No
Deprecated items	No
Incompatibilities	No
Part 3: Spatial Only	
ST_Geometry Type Hierarchy	n/a
Well-known Text Support	n/a
Well-known Binary Support	n/a
GML Support	*
Empty Set Support	n/a

***** Items marked with an asterisk will be addressed in a future change proposal.

End of Paper.