ISO/IEC JTC 1/SC 32/WG 4:STX-021r1
INCITS H2-2004-168r2
November 1, 2004
Authoritative file: stx021r1-Topo-Geo-and-Topo-Net-1-The-Concepts.pdf
Number of Pages: 22

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 1: The Concepts
Author: Paul Scarponcini
Status: SQL/MM—Part 3: Spatial Change Proposal
Source: Expert Contribution
Abstract:

References:

[1] ISO/IEC CD 13249-3:200x(E) – Text for CD Ballot, **Information technology – Database languages – SQL Multimedia and Application Packages — Part 3: Spatial**, June 30, 2004.

[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.

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 summarizes the approach proposed for addressing topology by providing the definitions and concepts relevant to this topic and therefore partially addresses the comments. Additional detail will need to be provided (in subsequent change proposals) in order to completely satisfy the comments.

1.2 Changes from Previous Version

This r2 H2 version (r1 MM version) of this change proposal resulted from comments received on the previous version. Changes are marked with a change bar in the left margin. Several issues have been addressed:

Unique ID generation – clarified that the unique ID's proposed for topological primitives are system generated and such generation is implementation-dependent.

Keeping Topo-Geo and Topo-Net in step – Support for both together would require keeping the two in step. The original proposed approach would require support for arrays. As the frequency of the requirement to support both is minimal, it is dropped from the proposal and left to individual database vendors or application developers who wish to add such capability beyond the requirements of this proposed standard. The result is that the Topo-Geo columns in the Topo-Net tables have been dropped.

Optional columns – the intent was to specify that some columns have optional values, i.e., allow NULL values. The column itself is mandatory. Wording has been changed to remove this confusion.

Mini-Topo approach – the original approach was more redundant than Mini-Topo in that two extra columns were proposed for the Topo-Geo Edge Table. These (previous left edge and previous right edge) have been dropped from this proposal. An implementation may choose to add them if they believe the trade-off in redundancy vs. computation is warranted. This new proposal includes an equivalent number of attributes proposed with the Mini-Topo approach, the only difference is that they are provided in three instead of four tables.

Mini-Topo trade-off – an example of the trade-off in the Mini-Topo Approach is provided: To determine the area of a feature having face topology primitives will require the system to determine the edges bounding it from the Edge table and then constructing an area geometry from the geometries of these edges from which the area could then be determined.

Other issues raised that have not (yet) been addressed include:

- compatibility with GML topology
- functions returning multiple values

- functions knowing schema name
- linking topological primitives to features (currently implementation-defined)
- defining "topologically consistent"
- for topology primitive splitting functions, updating one value and adding one new vs. deleting the old and adding two new (and similar handling for other updates)

1.3 Summary of Approach

SQL/MM Part 3: Spatial [1] currently addresses geometry. Considerable interest has been expressed in expanding this to include topology as well. To maintain harmony with ISO TC211 and the OpenGIS Consortium, this should be done consistent with ISO 19107 – Spatial Schema [2]. The coverage of topology in that document, however, is quite extensive. A simple subset of this is therefore appropriate as a first start.

In addition to harmonizing with ISO TC211 19107, there is also interest from ISO TC204, Intelligent Transportation Systems to develop an SQL implementation of their Graphic Data Files (GDF) standard, ISO 14825 [4]. Currently providing map information for in-car navigation systems, GDF has significantly more potential for providing map base data for a variety of transportation related applications. It is expected that an SQL implementation of GDF, and delivery via the leading RDBMS vendors, will greatly aid this realization. Because GDF is dependent on topology, SQL/MM will need to support topology for this to happen in a manner that will insure standardization across RDBMS vendors so that GDF data suppliers need only create a single, portable data set. The approach to topology taken by SQL/MM should therefore consider the GDF requirements for topology.

An approach to addressing topology was first presented in [3] and later expanded in [5]. Four options were presented, representing the possible combinations resulting from two key decisions. The first relates to the topology model itself: should it be modeled after the initial approach in [3] or be based upon the Mini-Topo model presented in [2]. The second choice is between implementations: relational in which topological primitives are stored as rows in tables or object-relational where the primitives are defined as user-defined (abstract) types.

After much debate, a compromise Relational solution was chosen to implement the Mini-Topo approach. The Mini-Topo model of Minimally Redundant Topology represents a compact model adopted by numerous commercially available products. It appears to meet the GDF requirements and is consistent with ISO TC211 19107. But its minimal redundancy requires more computation to "reconstruct" the information more readily available with the other approach. An example is the determination of the area of a feature having face topology primitives. This will require the system to determine the edges bounding it from the Edge table and then constructing an area geometry from the geometries of these edges from which the area could then be determined.

The Edge and Directed Edge tables from [5] are replaced by a single Edge table. Edges appear only once, instead of once for each direction. In addition to the Target Node (End Node), a Start Node is also persisted for each edge. This is equivalent to the target node of the reverse directed edge in [5]. The next (left face) directed edge was provided for in [5] for the edge and its reverse directed counterpart. This proposal provides for both of these as the next left (face) edge and the next right (face) edge for the single forward directed edge. (Persistence of the previous left and previous right edges originally proposed in the previous version of this document has been dropped. An implementation is free to add this beyond the requirements of the standard if it so desires.) The left face is provided for the forward directed edge as in [5] but the reverse directed edge's left face becomes the forward directed edge's right face. Edge ID's are integers. To reference an edge in the reverse direction, the ID is negated.

Because topological primitives are interdependent (e.g., nodes connect edges), an object-relational approach would require the use of REF types. As SQL support for REF types in not sufficient at this time, a more traditional fully relational (table) approach was selected. Where topological primitives have geometry, that geometry is specified with the object-relational geometry types In SQL/MM Part 3: Spatial [1].

It was also decided to propose two types of topology. An area based model is comprised of nodes, edges, and faces. As geometry can be assigned to each of these, the model is called Topology-Geometry (Topo-Geo). Topo-Geo assumes full, planar topology. A node must exist at the end of every edge and if two edges cross, they must be split at this point of intersection into two edges with an intervening node added. All topology is decomposed to its lowest level of primitives.

For linear applications, the Topology-Network (Topo-Net) model includes node and link topological primitives. In a spatial Topo-Net, links have Spatial geometries associated with them. Otherwise, the resultant network is a logical network (topology without geometry). Topo-Net only requires nodes where it would be possible to change link, i.e., at link ends. Unlike Topo-Geo, there is no requirement that nodes exist where links cross, e.g. at grade-separated crossings. As there is no area primitive, features would need to be bounded by links or by introducing a Topo-Geo level.

This latest version drops the ability to reference Topo-Geo primitives in a separate column in the Topo-Net Tables. This would have allowed an alternative (indirect) method of specifying geometries for Topo-Net topologies but at the expense of keeping the two topological models in step. The frequency with which this would be implemented for a given installation is probably minimal and therefore the requirement to offer this capability is removed from the standard. An implementation or application is free to add this if they so choose, but should then deal with the issues of keeping the two in step.

ISO TC204 supports three types of topology. Topo-Geo is compatible with its full, planar topology approach. Topo-Net is compatible with its non-planar connectivity topology approach. Non-explicit topology can be accommodated by existing SLQ/MM Part 3: Spatial ST_Geometry types.

The r1 (H2) version of this paper added <schema qualified name> to identify tables supporting a particular Topo-Geo called <topology-name> or Topo-Net called <network-name>. This necessitated changing the Topo-Net tables to “NETNODE” in place of “NODE” and, for consistency, “NETLINK” for “LINK”. Whether this needs t be passed to functions as an input parameter is to be determined.

Unique IDs of topological primitives are to be system generated. How an implementation implements this is implementation-defined.

1.4 Requirements

Topological relationships may be realizable in three ways: via features, geometry, or topological primitives. An intersection feature could be attributed with the road segments which participate and the road segment features can be attributed with their terminating intersections. It would then be possible to determine a route through a transportation network of road segment and intersection features by tracing through these features.

Alternatively, each feature can have an associated geometry. Using geometric calculations, it would then be possible to determine possible routes from assumed connectivity based on coincident location.

The third approach would be to associate topological primitives to features, such as nodes for intersections and edges (links) to road segments. Routing would then be achieved via the resultant graph of nodes and edges, independent of geometry.

SQL can be used to address the feature approach. SQL/MM Spatial supports the geometries necessary for the second approach. This proposed extension to SQL/MM Spatial addresses the data and functions necessary to enable the third approach.

As stated in ISO 19107 [2], topological complexes can be used to reduce the computational intensity of geometric calculations. They can also be used to relate feature instances independent of their geometry. The proposed approach therefore suggests topological structures which closely parallel the geometry types in SQL/MM Spatial [1].

A topological complex consists of a collection of topological primitives. These can include nodes, edges, and faces as the zero-, one-, and two-dimensional topological primitives which parallel geometric points, curves, and surfaces.

With the current geometry model in [1], a real world object can be represented as a feature. Typically, information about the feature is stored as columns in a table. A column can be specified to be of type ST_Geometry (or one of its subtypes). For a given row in the table, the value in this column defines the geometry of the feature instance specified by this row. Multiple geometry columns are permitted to enable multiple geometric representations for the same feature instance. But geometries cannot be shared between feature instances. The geometry of the boundary shared by two adjoining land parcels, for example, must be repeated for each parcel. Geometric calculations are required to determine if the two parcels are adjacent (i.e., share a common boundary).

The proposed topological model allows one or more columns in a feature table to be specified as the feature topology. The value in this column specifies the topological primitives which comprise the topology of the feature instance for the given row in the table. Topological primitives (ST_Node, ST_Edge, and ST_Face) can be shared across topologies. A topological primitive can have a geometry (of an appropriate ST_Geometry subtype).

Each feature may require multiple topological primitives (e.g., a road may be defined by a list of contiguous links). The topology column therefore needs to either be an array of (sometimes negated) topological primitive ID's or a topology ID which expands to a set (or sequence) of rows in another table in which each row has the unique identifier of a contributing topological primitive. For now, the approach taken is implementation-defined.

1.4.1 Topo-Geo Sample Data

The proposed standard extension for topology needs to provide for the storage and maintenance of topological data types. The following example will be used to convey these data requirements. The real world objects represented on the map in Figure 1 include three land parcels, a park, a fountain, and seven roads. All road intersections are at grade except where Road 5 crosses over Road 2.

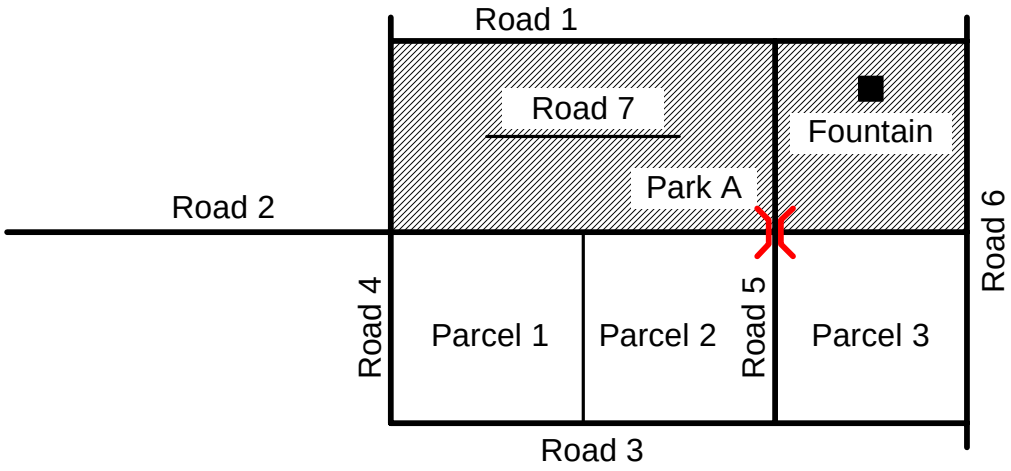

Figure 1. Map of real world objects

The topological primitives required to model the real world objects shown above are shown in the figure below. Each edge has a start and end node. For example, edge E1 goes from node N1 to node N2. This direction is consistent with the order of the points used to define the geometry of the edge. A face is bounded by a list of edges, specified in a counter-clockwise order. The edge list for face F1 would be (-E1, E3, E7, E8, -E4), where a negative sign prefix indicates that the face uses the edge in the direction opposite its stored direction. Edge E17 is an isolated edge. Node N15 is an isolated node. Node N7 is required in spite of the grade separation since Topo-Geo is a full planar topology model.

The topological primitives contained in the topologies of each real world object can be determined by comparing the two figures. For example, the Road 2 topology is comprised of edges -E6, E7, E8, and E9. Notice that edges may be included in their defined direction or in the opposite direction. Land parcel 1 is represented by face F3. The park includes faces F1 and F2.

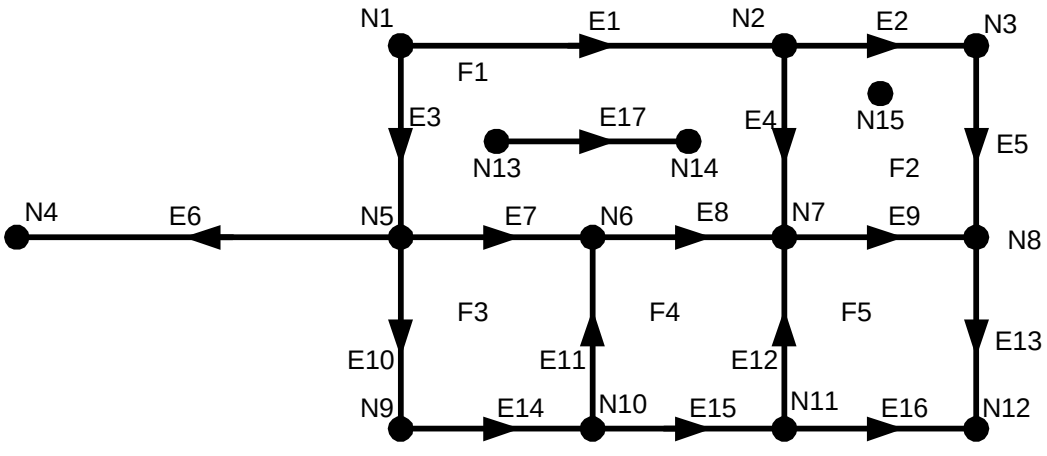

Figure 2. Topo-Geo topological primitives for the map example

1.4.2 Topo-Geo Representation

This proposal calls for the Topology-Geometry to be stored in three tables: <topology-name>.ST_NODE, <topology-name>.ST_EDGE, and <topology-name>.ST_FACE in the same schema whose schema name is <topology-name>. These tables are shown below, populated with the values necessary to represent the node, edge, and face topological primitives in the above example. For geometry values, a grid is added to the Topo-Geo from Figure 2:

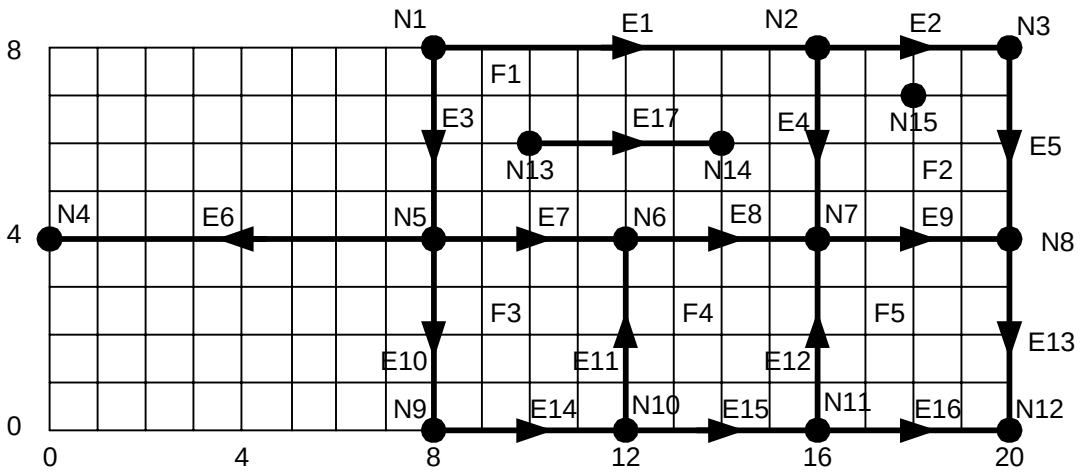

Figure 3. Topological primitives for the map example with geometry grid

The node table has columns for mandatory, unique node ID (integer), containing face ID (integer) for isolated nodes only, and ST_Point geometry (depicted here as (x,y) coordinate values):

Node ID	Face ID	Geometry
1		(8, 8)
2		(16, 8)
3		(20, 8)
4		(0, 4)
5		(8, 4)
6		(12, 4)
7		(16, 4)
8		(20, 4)
9		(8, 0)
10		(12, 0)
11		(16, 0)
12		(20, 0)
13		(10, 6)
14		(14, 6)
15	2	(18, 7)

Table 1. ST_<topology-name>_NODE Table

The edge table has columns for mandatory integer ID for the edge (unique), the start and end nodes, next left and right face edges (moving counterclockwise around the face boundary), and left and right faces, and a mandatory ST_Curve geometry column, again shown here with (x,y) coordinates.

Edge ID	Start Node ID	End Node ID	Next Left Edge	Next Right Edge	Left Face ID	Right Face ID	Geometry
1	1	2	2	3	0	1	(8, 8), (16, 8)
2	2	3	5	4	0	2	(16, 8), (20, 8)
3	1	5	7	1	1	0	(8, 8), (8, 4)
4	2	7	9	-1	1	2	(16, 8), (16, 4)
5	3	8	13	-2	0	2	(20, 8), (20, 4)
6	5	4	-6	-3	0	0	(8, 4), (0, 4)
7	5	6	8	10	1	3	(8, 4), (12, 4)
8	6	7	-4	-11	1	4	(12, 4), (16, 4)
9	7	8	-5	-12	2	5	(16, 4), (20, 4)
10	5	8	14	6	3	0	(8, 4), (8, 0)
11	10	6	-7	15	3	4	(12, 0), (12, 4)
12	11	7	-8	16	4	5	(16, 0), (16, 4)
13	8	12	-16	-9	0	5	(20, 4), (20, 0)
14	9	10	11	-10	3	0	(8, 0), (12, 0)
15	10	11	12	-14	4	0	(12, 0), (16, 0)
16	11	12	-13	-15	5	0	(16, 0), (20, 0)
17	13	14	-17	17	1	1	(10, 6), (14, 6)

Table 2. <topology-name>.ST_EDGE Table

The face table has columns for the mandatory, unique face ID (integer) and optional minimum bounding rectangle geometry of type ST_Polygon:

Face ID	MBR Geometry
0	(0, 0), (20, 0), (20, 8), (0, 8), (0, 0)
1	(8, 4), (16, 4), (16, 8), (8, 8), (8, 4)
2	(16, 4), (20, 4), (20, 8), (16, 8), (16, 4)
3	(8, 0), (12, 0), (12, 4), (8, 4), (8, 0)
4	(12, 0), (16, 0), (16, 4), (12, 4), (12, 0)
5	(16, 0), (20, 0), (20, 4), (16, 4), (16, 0)

Table 3. ST_<topology-name>_FACE Table

1.4.3 Topo-Net Sample Problem

Now consider the same map of real world objects from Figure 2, but abstracting only the transportation features. This includes the seven roads and the fountain, considered to be a point of interest, as shown in Figure 4.

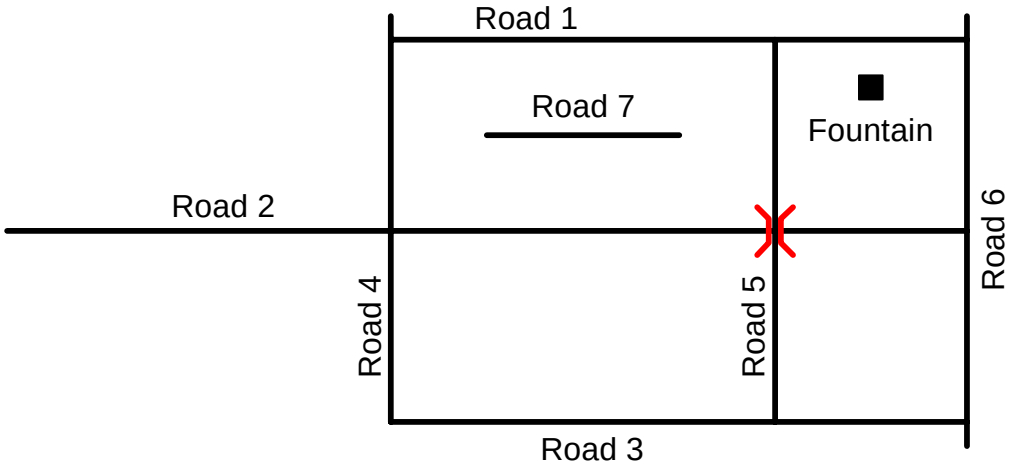

Figure 4. Map of real world transportation objects

The network primitives required to model these features which comprise the Topo-Net consist of nodes and links. Links are similar to Topo-Geo edges and have therefore been given link IDs similar to the edge IDs in Figure 2. Unlike edges, links can cross without an intervening node if inter-link access is prohibited. This is why there is no node N7 and a single link, L4, replaces edges E12 and –E4. Because edge E11 only functioned as a parcel boundary, it is not needed here. Without it, Node N6 and N10 are not needed and a single link, L7, replaces edges E7, E8, and E9. Extraneous node N9 is also missing so Link L10 replaces edges E10, E14, and E15.

As with Topo-Geo, Topo-Net primitives can be used by features. For example, Road 2 would be modeled by Topo-Net links –L6 and L7. As before, the negative sign indicates that a link participates in a feature in the direction opposite to how it is defined, i.e., from its end node to its start node. Road 2 has a directional sense, in this case from West to East (left to right in the Figure). The direction of Link 7 is consistent with this; Link 6 is contrary. Note that these directional senses for the road and links is independent of the direction of travel along the road and links. Direction of travel can be defined as a separate attribute of the road or link at the discretion of the database designer and would therefore be implemented as an additional column in the feature or link tables.

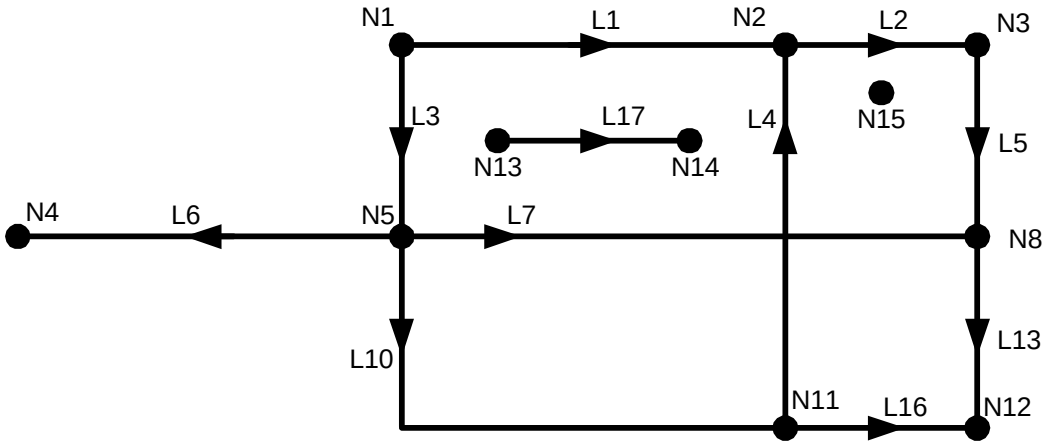

Figure 5. Topo-Net network primitives for the map example

1.4.4 Topo-Net Representation

This proposal calls for the Topology-Network to be stored in two tables: <network-name>.ST_NETNODE and <network-name>.ST_NETLINK in the <network-name> schema. These tables are shown below, populated with the values necessary to represent the node and edge network primitives in the above example. For a logical network, no geometries are included. For a spatial network, geometry values can be included directly as ST_Geometry values of the appropriate type (ST_Point and ST_Curve for nodes and links, respectively) or indirectly by including the corresponding Topo-Geo nodes and edges which themselves have geometries. For reference, a geometry grid is added to the Topo-Net from Figure 5:

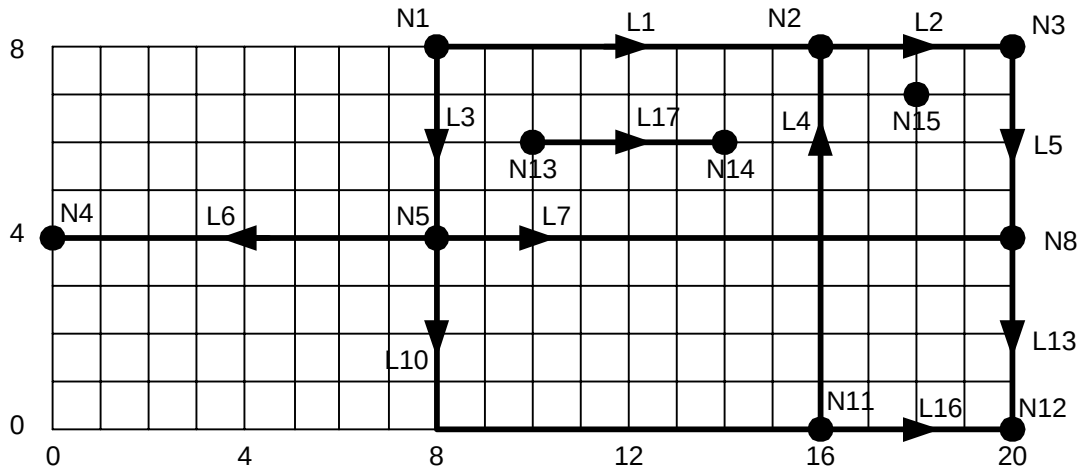

Figure 6. Network primitives for the map example with geometry grid

The node table has columns for mandatory, unique node ID (integer), and optional ST_Point geometry. A logical network will have NULL values for Geometry. A spatial network will have non-NULL values for Geometry.

Node ID	Geometry
1	**(8, 8)**
2	**(16, 8)**
3	**(20, 8)**
4	**(0, 4)**
5	**(8, 4)**
8	**(20, 4)**
11	**(16, 0)**
12	**(20, 0)**
13	**(10, 6)**
14	**(14, 6)**
15	(18, 7)

Table 4. <network-name>.ST_NETNODE Table

The link table has columns for mandatory integer IDs for the link (unique) and start and end nodes, and optional ST_Curve geometry values.

Link ID	Start Node ID	End Node ID	Geometry
1	1	2	**(8, 8), (16, 8)**
2	2	3	**(16, 8), (20, 8)**
3	1	5	**(8, 8), (8, 4)**
4	11	2	**(16, 0), (16, 8)**
5	3	8	**(20, 8), (20, 4)**
6	5	4	**(8, 4), (0, 4)**
7	5	8	**(8, 4), (20, 4)**
10	5	11	**(8, 4), (8, 0), (16, 0)**
13	8	12	**(20, 4), (20, 0)**
16	11	12	**(16, 0), (20, 0)**
17	13	14	(10, 6), (14, 6)

Table 5. <network-name>.ST_NETLINK Table

1.4.5 Functions

In addition to satisfying the data requirements outlined by example above, the proposed standard topology extension needs to support basic topological and network functions. These basic functions can then be utilized by an application to solve real world problems, such as routing.

Because topological and network primitives are proposed to be stored in tables, direct SQL select access [6] can satisfy most of the query functions listed in [5], such as selecting all edges incident on a node. Other more complex query results (e.g., an ordered list of the edges bounding a face) require explicit programmatic support and therefore standard functions are included in this proposal.

Also provided are functions for updating topology data. Updates to the Topo-Geo topology or Topo-Net network tables must be restricted to these specified functions to insure topological integrity is maintained.

1.4.6 Completeness

This change proposal is the first in a series of change proposals to address topology. It only addresses definitions and concepts. Assuming favorable response for this document, additional change proposal documents will be tabled to complete the topology proposal.

1.5 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".

1.6 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/STX_2004_11_St.Croix_USVI_USA

Use the following filename to get the paper:

stx021-Topo-Geo-and-Topo-Net-1-The-Concepts.pdf *PDF*

2 Proposal

2.1 Add the following new subclauses to subclause 3.1.2 “Definitions provided in Part 3”:

3.1.2.14+1
degree of a node
the number of links connecting to the node

3.1.2.16+1
directed link
directed topological object that represents an association between a link and one of its orientations

3.1.2.27+1
isolated edge
an edge which is not part of the boundary of any face and is not connected to any other edge

3.1.2.27+2
isolated link
a link which is not connected to any other link

3.1.2.27+1
link
one-dimensional topological primitive that represents a relationship between two nodes

3.1.2.27+2
logical network
a network which contains connectivity information but no geometric information

3.1.2.29+1
network
set of nodes and links

3.1.2.39+1
spatial network
a network which contains both connectivity and geometric information

2.2 Add the following new subclauses to subclause 3.1.4 "Definitions taken from ISO 19107":

c+1) connected node

j+1) directed edge

j+2) directed face

j+3) directed topological object

j+4) edge

j+5) end node

i+1) face

q+1) isolated node

r+1) node

s+1) start node

u+1) topological object

u+2) topological primitive

2.3 Add the following new subclauses to Clause 4 Concepts:

4.X Topology-Geometry

The following tables are defined: <topology-name>.ST_NODE, <topology-name>.ST_EDGE, and <topology-name>.ST_FACE.

The rows in these tables define topological primitives of type node, edge, and face, respectively for the Topology-Geometry (ST_Topo-Geo) called <topology-name>. Nodes and edges have associated ST_Geometries. Faces may have a minimum bounding rectangle geometry for spatial indexing.

Each topological primitive has an ID, unique within the respective table, which allows the primitive to be referenced from another (e.g., feature, ST_Topo-Geo, or ST_Topo-Net) table.

4.X.1 <topology-name>.ST_NODE

The <topology-name>.ST_NODE table provides storage for the node type of topological primitives (ST_Node) contained in the <topology-name> Topology-Geometry. An ST_Node has a mandatory, unique node ID of type integer and a geometry of type ST_Point. If ST_Node is an isolated node, it has a containing face, identified by an <topology-name>.ST_FACE.ID.

4.X.1.1 Functions on <topology-name>.ST_NODE only

1) ST_AddIsoNode: for the provided topology-name, optional face ID, and ST_Point geometry, inserts a row into the <topology-name>.ST_NODE table, returning the automatically generated, unique integer node ID. If a face ID is provided, the ST_Point geometry must be within the geometry of the face or an exception is raised. If another node in the <topology-name>.ST_NODE table exists at the ST_Point location, an exception is raised.
2) ST_MoveIsoNode: for the provided topology-name, node ID, and ST_Point geometry, updates the ST_Point geometry value. If the node is a connected node or if another node in the <topology-name>.ST_NODE table exists at the new location, an exception is raised.

3) ST_RemoveIsoNode: deletes the row for the isolated node identified by the provided topology-name and node ID. If the node is a connected node, an exception is raised.

4.X.2 <topology-name>.ST_EDGE

The <topology-name>.ST_EDGE table provides storage for the edge type of topological primitives (ST_Edge) contained in the <topology-name> Topology-Geometry. An ST_Edge has a mandatory unique edge ID of type integer; mandatory node ID's of type integer for the start and end nodes; mandatory edge ID's of type integer for the next left face and next right face edges; mandatory face ID's of type integer for the left and right faces; and a mandatory geometry of type ST_Curve. An isolated edge will have its containing face as both its left and right faces. The next right face edge will have an ID equal to the edge ID of the isolated edge and the next left face edge will have the negative of the isolated edge's ID.

Start and end node ID's are immutable. To change the start or end node of an edge, the edge must be removed and a new one must be created.

4.X.2.1 Functions on <topology-name>.ST_EDGE only

1) ST_AddIsoEdge: for the provided topology-name, start and end node IDs, and ST_Curve geometry, inserts a row into the <topology-name>.ST_EDGE table, returning the automatically generated, unique integer edge ID. The next right face ID is set equal to the new edge ID and the next left face edge is set equal to the negative of this value. The left and right face IDs are set equal to the containing face ID of the start and end node.

   An exception is raised for any of the following conditions:

   a) if the start and end nodes are not distinct, isolated nodes in the <topology-name>.ST_NODE table,

   b) if the start and end node containing face IDs are not equal,

   c) if the ST_Point geometry of the start node does not equal the ST_StartPoint of the ST_Curve geometry of the edge,

   d) if the ST_Point geometry of the end node does not equal the ST_EndPoint of the ST_Curve geometry of the edge,

   e) if the ST_Curve geometry is not within the geometry of the containing face of the start and end nodes,

   f) if the ST_Curve geometry intersects the ST_Point geometry of any isolated node other than the start and end node, or

   g) if the ST_Curve geometry intersects the ST_Curve geometry of any other edge in the <topology-name>.ST_EDGE table.

2) ST_GetFaceEdges: for the provided topology-name and face ID, returns the integer edge IDs of the edges which bound the face, in counterclockwise order. Edge IDs will be negated in the query result if the face is right of the edge when looking in the direction of the edge from start to end node.

3) ST_ChangeEdgeGeom: for the provided topology-name, edge ID, and ST_Curve geometry, updates the ST_Curve geometry value.

   An exception is raised for any of the following conditions:

   a) if the ST_StartPoint of the new ST_Curve geometry is not equal to the ST_StartPoint of the existing ST_Curve geometry,

b) if the ST_EndPoint of the new ST_Curve geometry is not equal to the ST_EndPoint of the existing ST_Curve geometry,

c) if the interior of the new ST_Curve geometry intersects the ST_Point geometry of any isolated node in the <topology-name>.ST_NODE table, or

d) if the interior of the new ST_Curve geometry intersects the ST_Curve geometry of any other edge in the <topology-name>.ST_EDGE table.

4) ST_RemoveIsoEdge: deletes the row for the isolated edge identified by the provided topology-name and edge ID. The start and end nodes are not removed from the <topology-name>.ST_NODE table. If the edge is a not an isolated edge, an exception is raised.

4.X.2.2 Functions on <topology-name>.ST_NODE and <topology-name>.ST_EDGE

1) ST_SplitEdge: for the provided topology-name, edge ID, and ST_Point geometry,

a) inserts a row into the <topology-name>.ST_NODE table, with geometry equal to the input ST_Point value,

b) returns the automatically generated, unique integer node ID,

c) deletes the row in the <topology-name>.ST_EDGE table for the edge identified by the provided topology-name and edge ID, and

d) inserts two rows into the <topology-name>.ST_EDGE table for the two new resultant edges, deriving appropriate node, edge, and face values from the deleted edge,

e) creates ST_Curve geometries for the two new edges by splitting the geometry of the split edge at the ST_Point location,

f) returns the two automatically generated, unique integer edge IDs, and

g) makes 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.

Both new edges have the same direction as the edge being split. An exception is raised for any of the following conditions:

a) if the edge identified by the edge ID does not exist in the <topology-name>.ST_EDGE table,

b) if the ST_Point geometry is not within the ST_Curve geometry of the identified edge, or

c) if a node already exists in the <topology-name>.ST_NODE table at the input ST_Point geometry location.

2) ST_HealEdges: for the provided topology-name and two edge IDs,

a) deletes the row in the <topology-name>.ST_NODE table corresponding to the node shared by the two identified edges,

b) deletes the two rows in the <topology-name>.ST_EDGE table identified by the input edge IDs,

c) inserts a new row into the <topology-name>.ST_EDGE table for the resultant edge, deriving appropriate node, edge, and face values from the deleted edges,

d) creates an ST_Curve geometry for the new edge from the geometries of the two deleted edges,

e) returns the automatically generated, unique integer edge ID for the new edge, and

f) makes any necessary updates to the next left and right face edge IDs for any edges incident on the start and end nodes of the edges being healed.

The direction of the new edge shall be the same as the direction of the first supplied edge. An exception is raised for any of the following conditions:

a) if either edge identified by the edge IDs does not exist in the <topology-name>.ST_EDGE table,

b) if the two edges do not share a common node, or

c) if additional edges also share the common node.

4.X.3 <topology-name>.ST_FACE

The <topology-name>.ST_FACE table provides storage for the face type of topological primitives (ST_Face) contained in the <topology-name> Topology-Geometry. An ST_Face has a mandatory, unique face ID of type integer and an optional MBR (minimum bounding rectangle) geometry of type ST_Polygon.

4.X.3.1 Functions on <topology-name>.ST_EDGE and <topology-name>.ST_FACE

1) ST_AddEdge: for the provided topology-name, start and end node IDs, and ST_Curve geometry,

   a) inserts a row into the <topology-name>.ST_EDGE table, with start and end nodes as specified, automatically determined next edges and left and right faces, and geometry equal to the input ST_Curve value,

   b) returns the automatically generated, unique integer edge ID, and

   c) if the new edge splits a face, then

      i) deletes the row in the <topology-name>.ST_FACE table corresponding to the face being split,

      ii) inserts two rows into the <topology-name>.ST_FACE table for the two new resultant faces,

      iii) creates ST_Polygon MBR geometries for the two new faces,

      iv) returns the two automatically generated, unique integer face IDs, and

      v) updates the appropriate next left and right face edge IDs and left and right face IDs for edges bounding the face being split.

   An exception is raised for any of the following conditions:

   a) if either the start or end nodes identified do not exist in the <topology-name>.ST_NODE table,

   b) if the ST_StartPoint of the new ST_Curve geometry is not equal to the ST_Point value of the start node geometry,

   c) if the ST_EndPoint of the new ST_Curve geometry is not equal to the ST_Point value of the end node geometry,

   d) if the interior of the new ST_Curve geometry intersects the ST_Point geometry of any isolated node in the <topology-name>.ST_NODE table,

   e) if the interior of the new ST_Curve geometry intersects the ST_Curve geometry of any other edge in the <topology-name>.ST_EDGE table, or

   f) if an edge already exists in the <topology-name>.ST_EDGE table with the same terminal nodes and geometry.

2) ST_RemoveEdge: for the provided topology-name and edge ID,

   a) deletes the row in the <topology-name>.ST_EDGE table identified by the edge ID, and

   b) if the edge removal results in the healing of two faces, then

      i) deletes the two rows in the <topology-name>.ST_FACE table corresponding to the faces being healed,

      ii) inserts a new row into the <topology-name>.ST_FACE table for the new resultant face,

iii) creates an ST_Polygon MBR geometry for the new face,

iv) returns the automatically generated, unique integer face ID, and

v) updates the appropriate next left and right face edge IDs and left and right face IDs for edges bounding the faces being healed.

The start and end nodes of the deleted edge remain in the <topology-name>.ST_NODE table. An exception is raised if the edge identified by the edge ID does not exist in the <topology-name>.ST_EDGE table.

4.X.3.2 Functions on <topology-name>.ST_NODE, <topology-name>.ST_EDGE and <topology-name>.ST_FACE

1) ST_CreateTopoGeo: for the provided topology-name, and ST_GeomCollection, populates the <topology-name>.ST_NODE, <topology-name>.ST_EDGE, and <topology-name>.ST_FACE tables from the geometry values in the ST_GeomCollection. An exception is raised if any of these three tables do not already exist or if they already contain any rows.

2) ST_ValidateTopoGeo: for the provided topology-name, determines whether the rows in the <topology-name>.ST_NODE, <topology-name>.ST_EDGE, and <topology-name>.ST_FACE tables are topologically consistent.

4.X+1 Topology-Network

The following tables are defined: <network-name>.ST_NETNODE and <network-name>.ST_NETLINK.

The rows in these tables define network primitives of type node and link, respectively for the Topology-Network (ST_Topo-Net) called <network-name>. These network primitives may have associated ST_Geometries.

Each topological primitive has an ID, unique within the respective table, which allows the primitive to be referenced from another (e.g., feature or ST_Topo-Net) table.

4.X+1.1 <network-name>.ST_NETNODE

The <network-name>.ST_NETNODE table provides storage for the node type of network primitives (ST_Node) contained in the <network-name> Topology-Network. An ST_Node has a mandatory, unique node ID of type integer and an optional geometry of type ST_Point.

4.X+1.1.1 Functions on <network-name>.ST_NETNODE only

1) ST_AddIsoNode: for the provided network-name, and optional ST_Point geometry, inserts a row into the <network-name>.ST_NETNODE table, returning the automatically generated, unique integer node ID. If another node in the <network-name>.ST_NETNODE table exists at the ST_Point location, an exception is raised.
2) ST_MoveIsoNode: for the provided network-name, node ID, and ST_Point geometry, updates the ST_Point geometry value. If the node is a connected node or if another node in the <network-name>.ST_NETNODE table already exists at the ST_Point location.
3) ST_RemoveIsoNode: deletes the row for the isolated node identified by the provided network-name and node ID. If the node is a connected node, an exception is raised.

4.X+1.2 <network-name>.ST_NETLINK

The <network-name>.ST_NETLINK table provides storage for the link type of network primitives (ST_Link) contained in the <network-name> Topology-Network. An ST_Link has a mandatory unique link ID of type integer; mandatory node IDs of type integer for the start and end nodes; and an optional geometry of type ST_Curve.

Start and end node ID's are immutable. To change the start or end node of an link, the link must be removed and a new one must be created.

4.X+1.2.1 Functions on <network-name>.ST_NETLINK only

1) ST_AddIsoLink: for the provided network-name, start and end node IDs, and optional ST_Curve geometry, inserts a row into the <network-name>.ST_NETLINK table, returning the automatically generated, unique integer link ID.

   An exception is raised for any of the following conditions:

   a) if the start and end nodes are not existing isolated nodes in the <network-name>.ST_NETNODE table,
   b) if an ST_Curve geometry is provided, then
      i) if the start node has a geometry and the location thereby specified does not equal the location of the ST_StartPoint of the proposed ST_Curve geometry of the link,

ii) if the end node has a geometry and the location thereby specified does not equal the location of the ST_EndPoint of the proposed ST_Curve geometry of the link, or

2) ST_ChangeLinkGeom: for the provided network-name, link ID, and ST_Curve geometry, updates the ST_Curve geometry value.

   An exception is raised if the link identified by the provided link ID has a non-NULL geometry value and

   a) if the ST_StartPoint of the new ST_Curve geometry is not equal to the ST_StartPoint of the existing ST_Curve geometry, or

   b) if the ST_EndPoint of the new ST_Curve geometry is not equal to the ST_EndPoint of the existing ST_Curve geometry,

3) ST_RemoveIsoLink: deletes the row for the isolated link identified by the provided network-name and link ID. The start and end nodes are not removed from the <network-name>.ST_NETNODE table. If the link is a not an isolated link, an exception is raised.

4.X+1.2.2 Functions on <network-name>.ST_NETNODE and <network-name>.ST_NETLINK

1) ST_SplitLogLink: for the provided network-name and link ID,

   a) inserts a row into the <network-name>.ST_NETNODE table, with NULL values for Geometry,

   b) returns the automatically generated, unique integer node ID,

   c) deletes the row in the <network-name>.ST_NETLINK table for the link identified by the provided network-name and link ID,

   d) inserts two rows into the <network-name>.ST_NETLINK table for the two new resultant links, deriving appropriate start and end node IDs from the deleted link and newly generated node, and

   e) returns the two automatically generated, unique integer link IDs.

   Both new links have the same direction as the link being split. An exception is raised if the link identified by the link ID does not exist in the <network-name>.ST_NETLINK table,

2) ST_SplitGeomLink: for the provided network-name, link ID, and ST_Point geometry,

   a) inserts a row into the <network-name>.ST_NETNODE table, with geometry equal to the input ST_Point value,

   b) returns the automatically generated, unique integer node ID,

   c) deletes the row in the <network-name>.ST_NETLINK table for the link identified by the provided network-name and link ID,

   d) inserts two rows into the <network-name>.ST_NETLINK table for the two new resultant links, deriving appropriate start and end node IDs from the deleted link and newly generated node,

   e) creates ST_Curve geometries for the two new links by splitting the geometry of the split link at the ST_Point location, and

   f) returns the two automatically generated, unique integer link IDs.

   Both new links have the same direction as the link being split. An exception is raised for any of the following conditions:

   a) if the link identified by the link ID does not exist in the <network-name>.ST_NETLINK table,

   b) if the ST_Point geometry is not within the ST_Curve geometry of the identified link,

   c) if a node already exists in the <network-name>.ST_NETNODE table at the input ST_Point geometry location, or

   d) if the link identified by the link ID contains a null geometry value.

3) ST_HealLinks: for the provided network-name and two link IDs,

   a) if no other links start or end at the node shared by the two identified links, deletes the row in the <network-name>.ST_NETNODE table corresponding to the node shared by the two identified links,

   b) deletes the two rows in the <network-name>.ST_NETLINK table identified by the input link IDs,

   c) inserts a new row into the <network-name>.ST_NETLINK table for the resultant link, deriving appropriate start and end node IDs from the deleted links,

   d) if the two links have geometry, creates an ST_Curve geometry for the new link from the geometries of the two deleted links,

   e) returns the automatically generated, unique integer link ID for the new link.

   The direction of the new link shall be the same as the direction of the first supplied link. An exception is raised for any of the following conditions:

   a) if either link identified by the link IDs does not exist in the <network-name>.ST_NETLINK table, or

   b) if the two links do not share a common node.

4) ST_LogiNetFromTGeo: for the provided network-name, and topology-name, creates a logical network by populating the <network-name>.ST_NETNODE and <network-name>.ST_NETLINK tables from the Topo-Geo values identified by the provided topology-name. A Topo-Net node will be created for each Topo-Geo node. A Topo-Net link will be created for each Topo-Geo edge. A logical network will result, with nodes and links having geometry values set to NULL. An exception is raised if either of the two Topo-Net tables do not already exist or if they already contain any rows or if any of the three Topo-Geo tables do not exist.

5) ST_SpatNetFromTGeo: for the provided network-name, and topology-name, creates a logical network by populating the <network-name>.ST_NETNODE and <network-name>.ST_NETLINK tables from the Topo-Geo values identified by the provided topology-name. A Topo-Net node will be created for each Topo-Geo node. A Topo-Net link will be created for each Topo-Geo edge. A spatial network will result, with nodes and links having geometry values obtained from their corresponding Topo-Geo primitives. An exception is raised if either of the two Topo-Net tables do not already exist or if they already contain any rows or if any of the three Topo-Geo tables do not exist.

6) ST_SpatNetFromGeom: for the provided network-name, and ST_GeomCollection, creates a spatial network by populating the <network-name>.ST_NETNODE and <network-name>.ST_NETLINK tables from the geometry values in the ST_GeomCollection. A node will be created wherever links start, end, or cross. A spatial network will result, with nodes and links having geometry values. An exception is raised if either of these two tables do not already exist or if they already contain any rows.

7) ST_ValidLogicalNet: for the provided network-name, determines whether the rows in the <network-name>.ST_NETNODE and <network-name>.ST_NETLINK tables constitute a consistent logical network.

8) ST_ValidSpatialNet: for the provided network-name, determines whether the rows in the <network-name>.ST_NETNODE and <network-name>.ST_NETLINK tables constitute a consistent spatial network with geometries.

3 Checklist

Concepts	Yes
Static methods	
Constructor methods	
Cast definitions	

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

End of Paper.