

Teradata Vantage™ - Geospatial Data Types

Release 17.10




July 2021

Copyright and Trademarks

Copyright © 2008 - 2021 by Teradata. All Rights Reserved.

All copyrights and trademarks used in Teradata documentation are the property of their respective owners. For more information, see [Trademark Information](#).

Product Safety

Safety type	Description
	Indicates a situation which, if not avoided, could result in damage to property, such as to equipment or data, but not related to personal injury.
	Indicates a hazardous situation which, if not avoided, could result in minor or moderate personal injury.
	Indicates a hazardous situation which, if not avoided, could result in death or serious personal injury.

Third-Party Materials

Non-Teradata (i.e., third-party) sites, documents or communications ("Third-party Materials") may be accessed or accessible (e.g., linked or posted) in or in connection with a Teradata site, document or communication. Such Third-party Materials are provided for your convenience only and do not imply any endorsement of any third party by Teradata or any endorsement of Teradata by such third party. Teradata is not responsible for the accuracy of any content contained within such Third-party Materials, which are provided on an "AS IS" basis by Teradata. Such third party is solely and directly responsible for its sites, documents and communications and any harm they may cause you or others.

Warranty Disclaimer

Except as may be provided in a separate written agreement with Teradata or required by applicable law, the information available from the Teradata Documentation website or contained in Teradata information products is provided on an "as-is" basis, without warranty of any kind, either express or implied, including the implied warranties of merchantability, fitness for a particular purpose, or noninfringement.

The information available from the Teradata Documentation website or contained in Teradata information products may contain references or cross-references to features, functions, products, or services that are not announced or available in your country. Such references do not imply that Teradata Corporation intends to announce such features, functions, products, or services in your country. Please consult your local Teradata Corporation representative for those features, functions, products, or services available in your country.

The information available from the Teradata Documentation website or contained in Teradata information products may be changed or updated by Teradata at any time without notice. Teradata may also make changes in the products or services described in this information at any time without notice.

Machine-Assisted Translation

Certain materials on this website have been translated using machine-assisted translation software/tools. Machine-assisted translations of any materials into languages other than English are intended solely as a convenience to the non-English-reading users and are not legally binding. Anybody relying on such information does so at his or her own risk. No automated translation is perfect nor is it intended to replace human translators. Teradata does not make any promises, assurances, or guarantees as to the accuracy of the machine-assisted translations provided. Teradata accepts no responsibility and shall not be liable for any damage or issues that may result from using such translations. Users are reminded to use the English contents.

Feedback

To maintain the quality of our products and services, e-mail your comments on the accuracy, clarity, organization, and value of this document to: docs@teradata.com.

Any comments or materials (collectively referred to as "Feedback") sent to Teradata Corporation will be deemed nonconfidential. Without any payment or other obligation of any kind and without any restriction of any kind, Teradata and its affiliates are hereby free to (1) reproduce, distribute, provide access to, publish, transmit, publicly display, publicly perform, and create derivative works of, the Feedback, (2) use any ideas, concepts, know-how, and techniques contained in such Feedback for any purpose whatsoever, including developing, manufacturing, and marketing products and services incorporating the Feedback, and (3) authorize others to do any or all of the above.

Contents

Chapter 1: Introduction to Geospatial Data Types	8
Changes and Additions	8
Chapter 2: Geospatial Data	9
Geospatial Information	9
SQL/MM Spatial	9
Geometry Types	9
Viewing Geospatial Data	12
Methods, Functions, and Stored Procedures	12
Geospatial Types and External Routines	14
Geospatial Indexes	14
Geospatial Statistics Collection	15
Geospatial Metadata	15
Terminology	16
System Requirements	16
Privileges	17
Chapter 3: Minimum Bounding Types and Methods	18
MBR	18
MBR Constructor	20
MBB	20
MBB Constructor	22
Intersects	23
XMin	24
YMin	24
ZMin	25
XMax	25
YMax	25
ZMax	26
Chapter 4: ST_Geometry Type	27
ST_Geometry	27
Geospatial Data Formats	31
ST_Geometry Type Transforms	42
Loading (Importing) and Unloading (Exporting) Geospatial Data	44
Chapter 5: ST_Geometry Constructors and Methods	46
ST_Geometry Constructor (BLOB Form)	46
ST_Geometry Constructor (CLOB Form)	46

ST_Geometry Constructor (VARBYTE Form)	47
ST_Geometry Constructor (VARCHAR Form)	48
Make_2D Method	49
MBB Method	50
SimplifyPreserveTopology Method	50
ST_3DDistance Method	51
ST_AsBinary Method	52
ST_AsText Method	53
ST_Boundary Method	54
ST_Buffer Method	55
ST_Centroid Method	56
ST_Contains Method	56
ST_ConvexHull Method	57
ST_CoordDim Method	58
ST_Crosses Method	59
ST_Difference Method	60
ST_Dimension Method	61
ST_Disjoint Method	62
ST_Distance Method	63
ST_Envelope Method	63
ST_Equals Method	64
ST_GeometryType Method	65
ST_Intersection Method	66
ST_Intersects Method	67
ST_Is3D Method	68
ST_IsEmpty Method	69
ST_IsSimple Method	69
ST_IsValid Method	70
ST_MBR Method	71
ST_MBR_Xmax Method [Deprecated]	71
ST_MBR_Xmin Method [Deprecated]	72
ST_MBR_Ymax Method [Deprecated]	72
ST_MBR_Ymin Method [Deprecated]	73
ST_MaxX Method	73
ST_MaxY Method	74
ST_MaxZ Method	75
ST_MinX Method	75
ST_MinY Method	76
ST_MinZ Method	76
ST_Overlaps Method	77
ST_Relate Method	78
ST_SRID Method	80
ST_SymDifference Method	80
ST_Touches Method	82
ST_Transform Method	83

ST_Union Method	84
ST_Within Method	86
ST_WKBTToSQL Method	87
ST_WKTTToSQL Method	88
Chapter 6: ST_Point Methods	90
ST_Geometry Constructor (ST_Point Form)	90
ST_SphericalBufferMBR	91
ST_SphericalDistance	92
ST_SpheroidalBufferMBR	93
ST_SpheroidalDistance	94
ST_X	95
ST_Y	96
ST_Z	97
Chapter 7: ST_LineString Methods	99
ST_3DIsClosed	99
ST_3DLength	99
ST_EndPoint	100
ST_IsClosed	101
ST_IsRing	101
ST_Length	102
ST_Line_Interpolate_Point	103
ST_NumPoints	103
ST_PointN	104
ST_StartPoint	105
Chapter 8: ST_Polygon Methods	106
ST_Area	106
ST_ExteriorRing	106
ST_InteriorRingN	108
ST_NumInteriorRing	108
ST_Perimeter	109
ST_PointOnSurface	110
Chapter 9: ST_GeomCollection Methods	111
ST_GeometryN	111
ST_NumGeometries	112
Chapter 10: ST_Geomsequence Methods	113
Clip	113
GetFinalT	113
GetInitT	114
GetUserFld	115
GetUserFldCount	116

HeadingN	116
LinkID	117
SpeedN	118
Chapter 11: Filtering Functions and Methods	121
Intersects_MBB	121
MBB_Filter	122
MBR_Filter	122
Within_MBB	123
Chapter 12: Embedded Services System Functions	125
AggGeomIntersection [Deprecated]	125
AggGeomUnion [Deprecated]	127
DataSize	128
FROM_MGRS	130
GeomFromGeoJSON	131
GeoJSONFromGeom	131
GeoSequenceFromRows	132
GeoSequenceToRows	136
PolygonSplit	138
TO_MGRS	142
Chapter 13: Geospatial UDFs	145
SphericalDistance	145
SpheroidalDistance	146
ST_GeomFromText	147
ST_GeomFromWKB	148
Chapter 14: Geospatial Table Operators	150
AggGeom	150
GeometryToRows	153
PolygonSplit	158
Chapter 15: Geospatial Predicates and the Optimizer	163
Geospatial Single-Table Predicates	163
Geospatial Join Predicates	167
Chapter 16: Geospatial Metadata (SYSSPATIAL Database)	172
GEOMETRY_COLUMNS Table	172
AddGeometryColumn Stored Procedure	174
AddGeometryColumn_3D Stored Procedure	177
DropGeometryColumn Stored Procedure	179
SPATIAL_REF_SYS Table	181
Appendix A: Notation Conventions	183

Appendix B: Tessellation 186

Appendix C: Additional Information 205

Introduction to Geospatial Data Types

Teradata Vantage™ is our flagship analytic platform offering, which evolved from our industry-leading Teradata® Database. Until references in content are updated to reflect this change, the term Teradata Database is synonymous with Teradata Vantage.

Advanced SQL Engine is a core capability of Teradata Vantage, based on our best-in-class Teradata Database. Advanced SQL refers to the ability to run advanced analytic functions beyond that of standard SQL.

For information on data type mapping between Advanced SQL Engine and ML Engine, see *Teradata Vantage™ User Guide*, B700-4002.

Teradata Vantage™ - Geospatial Data Types describes tools and database objects (including data types, functions, and methods) that provide an interface between Vantage and applications that manage, analyze, and display geographic information.

Changes and Additions

Date	Description
July 2021	Minor edits.

Geospatial Data

Geospatial Information

Geospatial information identifies the geographic location of features and boundaries on the planet. Geospatial types address the need to store, manage, retrieve, manipulate, and analyze geospatial information by governmental bodies and commercial enterprises. For example, local governments might use geospatial data for city planning, traffic management, or accident investigation.

You can use geospatial types to represent geometries having up to three dimensions.

Vantage geospatial data types define methods that perform geometric calculations and test for spatial relationships between two geospatial values.

SQL/MM Spatial

The Teradata geospatial implementation closely follows *ISO/IEC 13249-3, Information technology — Database languages — SQL Multimedia and Application Packages — Part 3: Spatial*, referred to in this document as *SQL/MM Spatial*.

Geometry Types

SQL/MM Spatial supports the following geometry types:

<ul style="list-style-type: none"> • ST_CircularString • ST_CompoundCurve • ST_Curve • ST_CurvePolygon • ST_GeomCollection 	<ul style="list-style-type: none"> • ST_Geometry • ST_LineString • ST_MultiCurve • ST_MultiLineString • ST_MultiPoint 	<ul style="list-style-type: none"> • ST_MultiPolygon • ST_MultiSurface • ST_Point • ST_Polygon • ST_Surface
---	--	--

ST_Geometry

Teradata provides the ST_Geometry data type for creating and manipulating geometric shapes in the database. ST_Geometry is implemented as a user-defined type (UDT). ST_Geometry is an instantiable type within Vantage. (Vantage UDTs do not support inheritance or subtyping.) You can use ST_Geometry as the data type of a table column to represent most of the geospatial types specified in the standard.

Type	Description
ST_Point	0-dimensional geometry that represents a single location in two-dimensional coordinate space.

Type	Description
ST_LineString	1-dimensional geometry usually stored as a sequence of points with a linear interpolation between points.
ST_Polygon	2-dimensional geometry consisting of one exterior boundary and zero or more interior boundaries, where each interior boundary defines a hole.
ST_GeomCollection	Collection of zero or more ST_Geometry values.
ST_MultiPoint	0-dimensional geometry collection where the elements are restricted to ST_Point values.
ST_MultiLineString	1-dimensional geometry collection where the elements are restricted to ST_LineString values.
ST_MultiPolygon	2-dimensional geometry collection where the elements are restricted to ST_Polygon values.
GeoSequence	Extension of ST_LineString that can contain tracking information, such as time stamps, in addition to geospatial information. GeoSequence is a Teradata extension to SQL/MM Spatial.

Here is an example of a table definition that has an ST_Geometry column:

```
CREATE TABLE sample_shapes (skey INTEGER, shape ST_Geometry);
```

Minimum Bounding Types

When working with geospatial data, it is often convenient to find the smallest rectangle or box shape that could contain a given 2D or 3D shape. Vantage includes methods and types that let you determine these "minimum bounding" shapes.

The ST_MBR() and MBB() methods on ST_Geometry return MBR and MBB types, respectively, that define these minimum bounding rectangle and box shapes. These are Teradata extensions to the SQL/MM Spatial standard.

Example: Finding the Minimum Bounding Box of Geometric Shapes

Consider the following table definition:

```
CREATE TABLE sample_MBRs (skey INTEGER, shape_mbr MBR);
```

The following SQL would obtain the MBR of each geometry in the shape column of the sample_shapes table using the ST_MBR method for ST_Geometry objects.

```
INSERT INTO sample_MBRs
SELECT skey, shape.ST_MBR()
FROM sample_shapes;
```

Geospatial Formats, Transforms, Loading, and Unloading

SQL/MM Spatial defines a well-known text (WKT) and a well-known binary (WKB) representation for each geospatial type. For details on WKT and WKB formats, see [Geospatial Data Formats](#).

Teradata provides several different "transform groups" for geospatial data that can automatically convert the data to different types and formats during load/import and unload/export operations between Vantage and client applications. The default transform group exports and imports geospatial data as CLOB types in the WKT format.

By default, client applications can load geospatial data into Vantage by passing a CLOB value in the WKT format. Applications that accept geospatial data in the WKT format can select directly from an ST_Geometry column. By default, the data type returned is a CLOB.

Applications that require geospatial data in the WKB format can select directly from a ST_Geometry column using the ST_AsBinary() method. The data type returned is a BLOB.

For client applications that use geospatial data in other types and formats, you can optionally use one of the other ST_Geospatial transforms that Teradata supplies. For more information about geospatial transform groups, see [ST_Geometry Type Transforms](#).

Vantage also defines ST_Geometry constructors and functions that client applications can call, passing in a WKT or WKB representation of a geospatial type and getting an ST_Geometry value as a result.

TDGeoImport and TDGeoExport Utilities

TDGeoImport and TDGeoExport are utilities that Teradata provides to interconvert between the database representation of geospatial data and formats compatible with the ESRI, MapInfo, and TIGER/Line data formats. For more information on these utilities, see *Teradata® Geospatial Utilities User Guide*, B035-2519.

Example: Inserting a Point Value into an ST_Geometry Column

Consider the previous definition of the sample_shapes table. The following example inserts a point value into the shape column using the WKT representation of a point:

```
INSERT INTO sample_shapes
VALUES (1001, 'POINT(10 20)');
```

Limitations

Geospatial data can have a maximum size of approximately 16MB, allowing for representations of approximately one million points.

It is a Vantage limit that you cannot create a table with more than six LOB columns. By default, ST_Geometry columns are LOBs, so they are subject to this limit. However, you can create ST_Geometry columns that are not LOBs by specifying a *maxlength* of less than or equal to 64,000 bytes with no INLINE LENGTH specification in the column type definition. That restricts the size of the ST_Geometry data to

a size that can be stored within a row, rather than in a LOB subtable. For more information on defining ST_Geometry columns, see [ST_GeometryType Method](#).

Additionally, Teradata load utilities that cannot directly load LOB types similarly cannot load ST_Geometry types unless you specify that ST_Geometry types use a non-default transform group. The transform group used for ST_Geometry data can be specified in the CREATE/MODIFY USER and CREATE/MODIFY PROFILE SQL statements. For more information on specifying transform groups, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

Viewing Geospatial Data

The following visualization tools display geospatial data stored as Vantage ST_Geometry types.

Visualization Tool	Description
MapServer	MapServer is an open source multi-platform Web mapping development environment developed at the University of Minnesota. It uses script files and form variables to configure the page output. It can be used to display geospatial data stored as Vantage ST_Geometry types from a web server running on Windows. MapServer interfaces with Teradata using a plug-in dll called MSTDPlugin.dll. The MapServer plug-in is available from Teradata Community Developer Zone.. Use MapServer when you have polygons or non-latitude, non-longitude data.
Google Maps	Use of Google Maps/Google Earth APIs requires a license. For more information, see Google Maps/Google Earth APIs Terms of Service . These APIs let you embed Google Maps in your web pages with JavaScript. Use Google Maps when you have point data or linestring data where the coordinates are longitude and latitude.

Methods, Functions, and Stored Procedures

SQL/MM Spatial defines constructor methods and instance methods for the geospatial types. The standard also defines user-defined functions (UDFs) to support geospatial types.

ST_Geometry Constructor Methods

To construct an instance of ST_Geometry that represents any of the supported subtypes, you can use one of several ST_Geometry constructor methods that Vantage implements. You can also use one of the UDFs that is defined in the SYSSPATIAL database. (For the best performance, use constructor methods instead of UDFs to construct instances of ST_Geometry.)

ST_Geometry Instance Methods

To perform operations on geospatial types, you can use the geospatial UDFs or the instance methods defined for ST_Geometry. (For the best performance, if an instance method is available that performs the same functionality as a UDF, use the instance method.)

Geospatial Predicates

The geospatial methods may be used to form geospatial predicates. For more information see [Geospatial Predicates and the Optimizer](#).

Methods Specific to the Subtype the ST_Geometry Type Represents

Some methods are specific to the subtype that the ST_Geometry type represents. For example, some methods only operate on ST_Geometry types that represent ST_Point values.

Other methods test the spatial relationship between geospatial types. Consider the following tables, where sample_cities represents a list of cities and sample_streets represents a list of streets.

```
CREATE TABLE sample_cities(
  skey INTEGER,
  cityName VARCHAR(40),
  cityShape ST_GEOMETRY);

CREATE TABLE sample_streets(
  skey INTEGER,
  streetName VARCHAR(40),
  streetShape ST_GEOMETRY);
```

The following requests insert polygon values into the sample_cities table and linestring values into the sample_streets table:

```
INSERT INTO sample_cities
VALUES(0, 'Oceanville', 'POLYGON((1 1, 1 3, 6 3, 6 0, 1 1))');

INSERT INTO sample_cities
VALUES(1, 'Seaside', 'POLYGON((10 10, 10 20, 20 20, 20 15, 10 10))');

INSERT INTO sample_streets
VALUES(1, 'Main Street', 'LINESTRING(2 2, 3 2, 4 1)');

INSERT INTO sample_streets
VALUES(1, 'Coast Blvd', 'LINESTRING(12 12, 18 17)');
```

The following query uses the ST_Within method to test if any of the streets are within any of the cities:

```
SELECT streetName, cityName
FROM sample_cities, sample_streets
WHERE streetShape.ST_Within(cityShape) = 1
ORDER BY cityName;
```

streetName

cityName

 Coast Blvd
 Main Street

Seaside
 Oceanville

Teradata Extensions to the SQL/MM Spatial Standard

Teradata also provides functions, methods, and stored procedures that are extensions to the SQL/MM Spatial standard. Some of the methods are defined for the MBR type and for the ST_Geometry type that represents a Geosequence value.

For details on the geospatial stored procedures, see [Introduction to Geospatial Data Types](#).

Geospatial Types and External Routines

You can write external routines (procedures and user-defined functions) that use or return geospatial types. For more information on writing external routines, see *Teradata Vantage™ - SQL External Routine Programming*, B035-1147.

Geospatial Indexes

Vantage supports creation of nonunique secondary indexes (NUSIs) on geospatial data columns. These indexes are implemented internally using a Hilbert R-tree structure. Geospatial indexes can greatly improve the performance of geospatial queries that contain single-table predicates or join predicates using the following geospatial methods:

- [MBB_Filter](#)
- [MBR_Filter](#)
- [ST_3DDistance Method](#)
- [ST_Contains Method](#)
- [ST_Crosses Method](#)
- [ST_Distance Method](#)
- [ST_Equals Method](#)
- [ST_Intersects Method](#)
- [ST_Overlaps Method](#)
- [ST_Touches Method](#)
- [ST_Within Method](#)
- [Within_MBB](#)
- [Intersects_MBB](#)
- [ST_Relate Method](#) (if this predicate is configured to execute one of the above methods)

For more information on how geospatial indexes and predicates can speed geospatial queries, see [Geospatial Predicates and the Optimizer](#).

The syntax used to create indexes on geospatial data columns is the same as the syntax used with CREATE INDEX and CREATE TABLE to create any NUSI with the following restrictions:

- Geospatial NUSIs must be single-column indexes.
- Geospatial NUSIs cannot be created on geospatial columns used in join indexes.
- The ALL keyword is not supported for geospatial indexes.
- The ORDER BY VALUES and ORDER BY HASH clauses are not supported for geospatial indexes.
- Geospatial indexes cannot be created on global temporary or volatile tables. Consequently, the TEMPORARY keyword is not supported for geospatial indexes.
- Geospatial indexes can be created for 3D data, however the z coordinate is ignored. The MBR of the geometry, which is based only on the x and y coordinates, is used to create the index.
- Because geospatial indexing is based on x and y coordinates, performance can be adversely affected by indexing if 3D geospatial objects being indexed are stacked in the z axis, with very little distribution in the x and y axes.

For more information about indexes, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144 and *Teradata Vantage™ - Database Design*, B035-1094. For more information on the Hilbert R-tree structure, see *Teradata Vantage™ - SQL Request and Transaction Processing*, B035-1142.

Geospatial Statistics Collection

Collecting statistics on geospatial columns can improve database performance with geospatial data by providing demographic data that can aid the Optimizer in generating the most efficient access and join plans. The basic syntax used for the COLLECT STATISTICS statement applied to geospatial data columns is the same as that used for non-geospatial data, with the following restrictions:

- The COLLECT STATISTICS USING options are not supported on geospatial columns.
- Automated statistics collection and management is not supported for geospatial columns.
- Because geospatial indexes are always NUSIs, the UNIQUE keyword cannot be used.
- The ALL and PARTITION options are not supported for geospatial data.
- The ORDER BY VALUES and ORDER BY HASH clauses are not supported for geospatial data.
- Statistics cannot be collected on geospatial columns in global temporary or volatile tables. Consequently, the TEMPORARY keyword is not supported for geospatial data.
- Statistics collection for geospatial data is supported only for single columns in user tables. It is not supported on geospatial data in views.
- Statistics cannot be collected on geospatial data in join or hash indexes.
- INITIATE INDEX ANALYSIS and RESTART INDEX ANALYSIS are not supported for geospatial indexes.
- If statistics are collected on 3D geospatial data, only the x and y coordinate data are considered. The z coordinate data is ignored. The output of the SHOW STATISTICS command for 3D geospatial data is unchanged from that of 2D geospatial data.

Geospatial Metadata

SQL/MM Spatial defines two tables (or views) that provide additional information about columns of type ST_Geometry and the spatial reference systems.

Table	Description
GEOMETRY_COLUMNS Table	Provides metadata for every table column defined as ST_Geometry. Teradata provides stored procedures to add and drop geometry metadata in this table. For more information on these procedures, see AddGeometryColumn Stored Procedure , AddGeometryColumn_3D Stored Procedure , and DropGeometryColumn Stored Procedure
SPATIAL_REF_SYS Table	Contains information about each spatial reference system. During installation, the Database Initialization Program (DIP) utility executes a script that populates the SPATIAL_REF_SYS table. For more information on DIP, see <i>Teradata Vantage™ - Database Utilities</i> , B035-1102.

The GEOMETRY_COLUMNS and SPATIAL_REF_SYS tables and associated stored procedures are defined in the SYSSPATIAL database.

For more information, see [Geospatial Metadata \(SYSSPATIAL Database\)](#).

Terminology

To simplify information that pertains to a subtype of ST_Geometry, this documentation uses the name of the subtype, although Teradata Vantage does not support actual instances of the subtype.

For example, consider a method that returns an ST_Geometry type that represents an ST_Point. Rather than discuss the return type as “an ST_Geometry type that represents an ST_Point” the topic discusses the return type as an ST_Point type.

System Requirements

Before you can use the ST_Geometry, MBR, and MBB data types, your system must meet the following requirements:

- You must have a C or C++ compiler on a Teradata node prior to running the Database Initialization Program (DIP) during installation.
- You must run the DIPGEO script file to create the database infrastructure to support geospatial data.

Note:

The DIPGEO script is normally run automatically as part of executing the DIPALL script of the DIP utility during system installation. For more information on the DIP utility, see *Teradata Vantage™ - Database Utilities*, B035-1102.

Privileges

Before you can use the geospatial types, functions, metadata tables, and stored procedures, you must be granted certain privileges.

Consider a user named GeoUser. The following statements grant the correct privileges:

- `GRANT UDTUSAGE ON SYSUDTLIB TO GeoUser;`
- `GRANT EXECUTE FUNCTION ON SYSSPATIAL TO GeoUser;`
- `GRANT SELECT ON SYSSPATIAL TO GeoUser;`
- `GRANT EXECUTE PROCEDURE ON SYSSPATIAL TO GeoUser;`

Minimum Bounding Types and Methods

This section describes the Vantage minimum bounding rectangle (MBR) and minimum bounding box (MBB) types and associated methods. These types define the smallest rectangle or box that can surround a 2D or 3D geometric object, respectively.

MBR

Defines the minimum sized rectangle that can surround a 2D geometry.

MBR Syntax

Use the following syntax to specify an MBR data type in a column definition.

```
[SYSUDTLIB.] MBR [ attribute [...] ]
```

Syntax Elements

SYSUDTLIB.

The database that contains the definition of the MBR data type.

attribute

A column defined as data type MBR supports the following data type attributes:

- NULL
- NOT NULL
- FORMAT
- TITLE
- NAMED
- DEFAULT NULL

For more information on using these data type attributes, see *Teradata Vantage™ - Data Types and Literals*, B035-1143.

An MBR column does not support column storage or column constraint attributes.

Usage Notes

Transforms

Client applications can use a VARCHAR to insert a value into an MBR column. The VARCHAR must have the following format:

```
(xmin, ymin, xmax, ymax)
```

where white space can appear anywhere.

For queries that select data from an MBR column, the database returns the MBR data as a VARCHAR(256) using the same format (xmin, ymin, xmax, ymax).

Consider the following definition of a table that defines an MBR column.

```
CREATE TABLE sample_MBRs (skey INTEGER, shape_mbr MBR);
```

A query that selects data from the shape_mbr column results in the database returning the column data as a VARCHAR(256):

```
SELECT *
FROM sample_MBRs;
```

SKEY	SHAPE_MBR
-----	-----
1001	(10, 3, 22, 7)

Casts

Vantage implements cast functionality that allows data type conversions between VARCHAR and MBR types. To convert a VARCHAR type to an MBR type, the VARCHAR must use the following format:

```
(xmin, ymin, xmax, ymax)
```

where white space can appear anywhere.

Consider the following table definitions.

```
CREATE TABLE sample_MBRs (skey INTEGER, shape_mbr MBR);
CREATE TABLE sample_data (mbr_data VARCHAR(256));
```

This example performs MBR to VARCHAR data type conversion:

```
INSERT INTO sample_data
SELECT CAST(shape_mbr as VARCHAR(256))
FROM sample_MBRs;
```

Ordering

The MBR data type is defined with an inherent ordering scheme. In general, however, ordering MBR types is not useful or meaningful.

MBR Constructor

Returns an MBR value.

MBR Constructor Syntax

```
MBR ( xmin, ymin, xmax, ymax )
```

Syntax Elements

xmin

A FLOAT value for the lower left X coordinate of the MBR.

ymin

A FLOAT value for the lower left Y coordinate of the MBR.

xmax

A FLOAT value for the upper right X coordinate of the MBR.

ymax

A FLOAT value for the upper right Y coordinate of the MBR.

Example: MBR Constructor

```
INSERT INTO sample_MBRs VALUES (0, NEW MBR(5, 5, 10, 10) );
```

MBB

Defines the minimum sized box that can surround a 3D geometry.

MBB Syntax

Use the following syntax to specify an MBB data type in a column definition.

```
[SYSUDTLIB.] MBB [ attribute [...] ]
```

Syntax Elements

SYSUDTLIB.

The database that contains the definition of the MBB data type.

attribute

A column defined as data type MBB supports the following data type attributes:

- NULL
- NOT NULL
- FORMAT
- TITLE
- NAMED
- DEFAULT NULL

For more information on using these data type attributes, see *Teradata Vantage™ - Data Types and Literals*, B035-1143.

An MBB column does not support column storage or column constraint attributes.

Usage Notes

Transforms

Client applications can use a VARCHAR to insert a value into an MBB column. The VARCHAR must have the following format:

```
(xmin, ymin, zmin, xmax, ymax, zmax)
```

where white space can appear anywhere.

For queries that select data from an MBB column, the system returns the MBB data as a VARCHAR(340) using the same format (xmin, ymin, xmax, ymax).

Consider the following definition of a table that defines an MBB column.

```
CREATE TABLE sample_MBBs (skey INTEGER, shape_mbb MBB);
```

A query that selects data from the `shape_mbr` column results in the system returning the column data as a `VARCHAR(340)`:

```
SELECT *
FROM sample_MBBs;

SKEY          SHAPE_MBB
-----
1002          (10, 3, 1, 22, 7, 2)
```

Casts

Vantage implements cast functionality that allows data type conversions between `VARCHAR` and `MBB` types. To convert a `VARCHAR` type to an `MBB` type, the `VARCHAR` must use the following format:

```
(xmin, ymin, zmin, xmax, ymax, zmax)
```

where white space can appear anywhere.

Consider the following table definitions.

```
CREATE TABLE sample_MBBs (skey INTEGER, shape_mbb MBB);
CREATE TABLE sample_data (mbb_data VARCHAR(340));
```

This example performs MBR to `VARCHAR` data type conversion:

```
INSERT INTO sample_data
SELECT CAST(shape_mbb as VARCHAR(340))
FROM sample_MBBs;
```

Ordering

The `MBB` data type is defined with an inherent ordering scheme. In general, however, ordering MBR types is not useful or meaningful.

MBB Constructor

Returns an `MBB` value.

MBB Constructor Syntax

```
MBB ( xmin, ymin, xmax, ymax, zmin, zmax )
```

Syntax Elements

xmin

A FLOAT value for the lower front left X coordinate of the MBB.

ymin

A FLOAT value for the lower front left Y coordinate of the MBB.

zmin

A FLOAT value for the lower front left Z coordinate of the MBB.

xmax

A FLOAT value for the upper back right X coordinate of the MBB.

ymax

A FLOAT value for the upper back right Y coordinate of the MBB.

zmax

A FLOAT value for the upper back right Z coordinate of the MBB.

Example: MBB Constructor

```
INSERT INTO sample_MBBs VALUES (0, NEW MBB(5, 5, 5, 10, 10, 10) );
```

Intersects

Tests if an MBR or MBB value spatially intersects another value of the same data type.

Intersects Syntax

```
Intersects ( other )
```

Syntax Elements

other

The other value, which must be the same data type on which the method is called.

If the value of the MBR or MBB spatially intersects *other*, Intersect returns an INTEGER value of 1; otherwise, it returns an INTEGER value of 0.

Examples: Intersects

```
SELECT skey
FROM sample_MBRs
WHERE shape_mbr.Intersects(NEW MBR(5, 5, 10, 10)) = 1;

SELECT skey
FROM sample_MBBs
WHERE shape_mbb.Intersects(NEW MBB(5, 5, 5, 10, 10, 10)) = 1;
```

XMin

Returns the minimum x coordinate value of an MBR or MBB (a FLOAT value):

- For an MBR this is the x coordinate of the lower left corner.
- For an MBB this is the x coordinate of the lower left front corner.

XMin Syntax

```
XMin ()
```

Examples: XMin

```
SELECT NEW MBR(0, 1, 10, 20).XMin();

SELECT NEW MBB(0, 1, 10, 10, 20, 10).XMin();
```

YMin

Returns the minimum y coordinate value of an MBR or MBB (a FLOAT value):

- For an MBR this is the y coordinate of the lower left corner.
- For an MBB this is the y coordinate of the lower left front corner.

YMin Syntax

```
YMin ()
```

Examples: YMin

```
SELECT NEW MBR(0, 1, 10, 20).YMin();

SELECT NEW MBB(0, 1, 10, 10, 20, 10).YMin();
```

ZMin

Returns the minimum z coordinate value of an MBB (a FLOAT value). This is the z coordinate of the lower left front corner.

ZMin Syntax

```
ZMin ()
```

Example: ZMin

```
SELECT NEW MBB(0, 1, 10, 10, 20, 10).ZMin();
```

XMax

Returns the maximum x coordinate value of an MBR or MBB (a FLOAT value):

- For an MBR this is the x coordinate of the upper right corner.
- For an MBB this is the x coordinate of the upper right back corner.

XMax Syntax

```
XMax ()
```

Examples: XMax

```
SELECT NEW MBR(0, 1, 10, 20).XMax();

SELECT NEW MBB(0, 1, 10, 10, 20, 10).XMax();
```

YMax

Returns the maximum y coordinate value of an MBR or MBB (a FLOAT value):

- For an MBR this is the y coordinate of the upper right corner.
- For an MBB this is the y coordinate of the upper right back corner.

YMax Syntax

```
YMax ()
```

Examples: YMax

```
SELECT NEW MBR(0, 1, 10, 20).YMax();
```

```
SELECT NEW MBB(0, 1, 10, 10, 20, 10).YMax();
```

ZMax

Returns the maximum z coordinate value of an MBB (a FLOAT value). This is the z coordinate of the upper right back corner.

ZMax Syntax

```
ZMax ()
```

Example: ZMax

```
SELECT NEW MBB(0, 1, 10, 10, 20, 10).ZMax();
```

ST_Geometry Type

Teradata provides the ST_Geometry data type for creating and manipulating geometric shapes in the database. ST_Geometry is implemented as a user-defined type (UDT). ST_Geometry is an instantiable type within Vantage. (Vantage UDTs do not support inheritance or subtyping.) You can use ST_Geometry as the data type of a table column to represent most of the geospatial types specified in the standard.

ST_Geometry

Represents any of the following geometry types: ST_Point, ST_LineString, ST_Polygon, ST_GeomCollection, ST_MultiPoint, ST_MultiLineString, ST_MultiPolygon, and GeoSequence.

ST_Geometry Syntax

```
[SYSUDTLIB.] ST_GEOMETRY
[ ( maxlength ) ]
[ INLINE LENGTH integer ]
[ attribute [...] ]
```

Syntax Elements

SYSUDTLIB.

The name of the database in which all UDTs are created.

maxlength

A positive integer value followed by an optional multiplier.

maxlength specifies the maximum length of the data type in bytes. You can define a maximum length on a per instance basis. When specified, the data type is used in a manner analogous to the VARBYTE or BLOB data types.

The length specified only covers the actual data length. The actual storage sizes include additional header information.

The default value is approximately 16 MB.

Note:

You can find the actual size of ST_Geometry objects by using the [DataSize](#) system function.

INLINE LENGTH *integer*

A positive integer value which specifies the inline storage size. Data that is smaller than or equal to the inline storage size is stored inside the base row; otherwise, it is stored in a LOB subtable.

The inline length cannot be larger than *maxlength*.

The default value is approximately 10,000 bytes.

attribute

Appropriate data attributes.

An ST_GEOMETRY column supports the following attributes:

- NULL
- NOT NULL
- FORMAT
- TITLE
- NAMED
- DEFAULT NULL

For more information on using the data attributes, see *Teradata Vantage™ - Data Types and Literals*, B035-1143.

An ST_GEOMETRY column does not support column storage or column constraint attributes.

Usage Notes

Casts

The database implements cast functionality that allows data type conversions between the following types:

- ST_Geometry and VARCHAR(64000)
- ST_Geometry and CLOB

To cast a VARCHAR or CLOB to an ST_Geometry type, the VARCHAR or CLOB data must use the WKT format of any of the ST_Geometry subtypes. For example:

```
INSERT INTO sample_shapes
VALUES (1001, CAST('LINESTRING(1 1, 2 2, 3 3)' AS ST_Geometry));
```

For more information on the WKT format of the ST_Geometry type, see [Well-Known Text Format](#).

Ordering

Because the database requires ordering functionality for UDTs, an ordering definition exists for the ST_Geometry type. In general, however, ordering ST_Geometry types by their ordering definitions is not useful or meaningful.

To determine spatial relationships between geometries, for example whether one ST_Geometry type is equal to another, the best practice is to use the ST_Geometry spatial methods, for example `geom1.ST_Equals(geom2)`.

For more information on ordering functionality, see the CREATE ORDERING statement in *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

Limitations

The maximum size of an ST_Geometry type is approximately 16 MB, allowing for representations of approximately one million points.

The 16 MB limit applies to the following:

- Any WKB representation that is passed into the ST_Geometry type (for example, `cast`, ST_Geometry constructor method)
- Any WKB representation that is returned from the ST_Geometry type (for example, `cast`, ST_AsBinary)
- Any WKT representation that is passed into the ST_Geometry type (for example, ToSQL transform, `cast`, ST_Geometry constructor method)
- Any ST_Geometry method (for example, ST_Union, ST_Buffer) that produces a geometry with a WKB that exceeds the 16 MB limit is reported as an error.

WKT representations returned from the ST_Geometry type (for example, `cast`, ST_AsText) can exceed the approximate 16 MB limit. These cases typically occur when the corresponding WKB is large, but less than 16 MB, and the corresponding WKT exceeds 16 MB. Note that if the returned WKT representation exceeds 16 MB, this text representation of the geometry can no longer be input back into the ST_Geometry object, because it exceeds the size limit. In these cases, use the WKB representation instead.

Any restrictions that apply to a LOB type and to a UDT type also apply to the ST_Geometry types that are stored as LOBs. For example, for these types the maximum number of ST_Geometry columns in a table is the same as the maximum number of LOB columns in a table. Additionally, Teradata load utilities that cannot directly load LOB types similarly cannot load these ST_Geometry types.

Inline Storage Size

You can use the optional `INLINE LENGTH` specification to specify the inline storage size. When the data is smaller than or equal to the inline storage size, it is stored inside the base row. Otherwise, the data is stored as a LOB (large object).

If the inline length is equal to the maximum length specified for the data type, the data type is treated as a non-LOB type. In this case, the performance may be better because there is no LOB overhead. You may see some performance improvement especially when the data type is used with UDFs.

You can use the *maxlength* together with the `INLINE LENGTH` options to improve space management for geospatial data by forcing smaller geometries to be stored inline as non-LOB values. This stores the geospatial data within the table row itself, rather than in a separate LOB subtable. Furthermore, by specifying a small inline length for small geometries, Vantage reserves less space in the row, so more columns can be added to the table. Additionally, non-LOB geometries work with some load utilities that do not support LOBs.

Examples: ST_Geometry

Specifying an ST_Geometry column using the *maxlength* and `INLINE LENGTH` options

The following examples show different column data type specifications for ST_Geometry columns and describe how the geometry data is stored.

`ST_Geometry`

Because there is neither a *maxlength* nor an `INLINE LENGTH` specified, *maxlength* defaults to approximately 16 MB and `INLINE LENGTH` defaults to approximately 10000 bytes. Any geometry that is less than or equal to 10,000 bytes will be stored directly in the row, otherwise it will be stored as a LOB.

`ST_Geometry(250000)`

The maximum length of the ST_Geometry is 250,000 bytes. Since no `INLINE LENGTH` was specified, it defaults to 10,000 bytes. If this ST_Geometry is larger than 10,000 bytes, it will be stored as a LOB.

`ST_Geometry(250000) INLINE LENGTH 4000`

The maximum length of the ST_Geometry is 250,000 bytes. Because there is also an `INLINE LENGTH` specified, if the geometry data is less than or equal to 4,000 bytes it will be stored inline, within a row, otherwise it will be stored as a LOB.

`ST_Geometry(8000)`

Because the maximum length is specified without an `INLINE LENGTH`, and the maximum is less than the Teradata threshold for storing a column value as a LOB, the ST_Geometry values are always stored inline.

ST_Geometry(8000) INLINE LENGTH 2000

The maximum length of the ST_Geometry is 8,000, however, if it exceeds 2,000 bytes it will be stored as a LOB.

ST_Geometry INLINE LENGTH 2000

ST_Geometry values larger than 2,000 bytes will be stored as LOB values. The *maxlength* will be the maximum length possible, approximately 16 MB.

Geospatial Data Formats

SSQL/MM Spatial defines a well-known text (WKT) and a well-known binary (WKB) representation for each geospatial type. The database supports these standard formats.

The database also supports ESRI-compatible extended versions of these formats (EWKT and EWKB) that include SRID information together with the geometry data. To have geospatial data use these extended formats, specify that a user or profile use one of the "SRID" transform groups. For more information on geospatial transform groups, see [ST_Geometry Type Transforms](#).

Well-Known Text Format

Defines the format of geometry data when using character data to construct new ST_Geometry instances or transfer ST_Geometry values to and from client applications.

Syntax

point

```
POINT point_spec
```

linestring

```
LINESTRING linestring_spec
```

polygon

```
POLYGON polygon_spec
```

multipoint

```
MULTIPOINT { EMPTY | ( point_spec [,...] ) }
```

multilinestring

```
MULTILINESTRING { EMPTY | ( linestring_spec [,...] ) }
```

multipolygon

```
MULTIPOLYGON { EMPTY | ( polygon_spec [,...] ) }
```

geometrycollection

```
GEOMETRYCOLLECTION { EMPTY | ( geometry_collection_item [,...] ) }
```

geosequence

```
GEOSEQUENCE {
  EMPTY |
  ( ( x-y_pair [,...] ),
    ( ts [,...] ),
    ( linkID [,...] ),
    ( count, uf [,...] )
  )
}
```

point_spec

```
{ EMPTY | ( x-y_pair_opt_z ) }
```

linestring_spec

```
{ EMPTY | ( x-y_pair_opt_z [,...] ) }
```

polygon_spec

```
{ EMPTY | ( linestring_spec [, ...] ) }
```

geometry_collection_item

```
{ point |  
  linestring |  
  polygon |  
  multipoint |  
  multilinestring |  
  multipolygon |  
  geometry_collection |  
  geosequence  
}
```

x-y_pair

```
x y
```

x-y_pair_opt_z

```
x y [ z ]
```

Syntax Elements**EMPTY**

The empty set.

x

A numeric value that represents the x coordinate of a point.

y

A numeric value that represents the y coordinate of a point.

z

A numeric value that represents the z coordinate of a point in 3D space.

ts

A timestamp value with the following format:

```
yyyy-mm-dd hh:mi:ss.ms
```

The first timestamp value is associated with the first point, the second timestamp value is associated with the second point, and so forth.

You must specify *n* timestamp values, where *n* is the number of points in the geosequence.

linkID

A NUMERIC(18,0) value for the ID of the link on the road network for a point in the geosequence.

This value is reserved for a future release.

The first link ID value is associated with the first point, the second link ID value is associated with the second point, and so forth.

You must specify *n* link ID values, where *n* is the number of points in the geosequence.

count

An integer value that represents the number of *uf* elements for each point in the geosequence.

A value of 0 indicates that no *uf* elements appear after *count* in the character string.

uf

A user field that represents a FLOAT value to associate with a point. For example, certain tracking systems may associate velocity, direction, and acceleration values with each point.

You must specify *count* groups of *n* user field values (where *n* is the number of points in the geosequence). The first group provides the first user field values for each point, the second group provides the second user field values for each point, and so forth.

Usage Notes

Usage Notes

Vantage implements transform functionality that, by default, allows importing and exporting an ST_Geometry type to and from the server as a CLOB in WKT format. This means that a client application can use a CLOB to insert a value into an ST_Geometry column, provided the CLOB uses the WKT format of one of the geospatial subtypes that ST_Geometry can represent. Similarly, when a client application submits a query that selects data from an ST_Geometry column, by default, Vantage exports the type as a CLOB using the WKT format of the geometry that the ST_Geometry column represents.

Teradata also provides other transforms for the ST_Geometry type that allow for the import and export of geospatial data as other types and formats. For more information, see [ST_Geometry Type Transforms](#)

Examples: WKT Format for Geospatial Data

Consider the following table:

```
CREATE TABLE sample_shapes (skey INTEGER, shape ST_Geometry);
```

Here are some examples that show how to insert geometry values using WKT representation:

```
INSERT INTO sample_shapes
  VALUES (1001, 'POINT(10 20)');

INSERT INTO sample_shapes
  VALUES (1002, 'POINT EMPTY');

INSERT INTO sample_shapes
  VALUES (1003, 'LINESTRING(1 1, 2 2, 3 3, 4 4)');

INSERT INTO sample_shapes
  VALUES (1004, 'LINESTRING EMPTY');

INSERT INTO sample_shapes
  VALUES (1005, 'POLYGON((0 0, 0 20, 20 20, 20 0, 0 0),
                        (5 5, 5 10, 10 10, 10 5, 5 5))');

INSERT INTO sample_shapes
  VALUES (1006, 'MULTIPOINT((1 1), (1 3), (6 3), (10 5), (20 1))');

INSERT INTO sample_shapes
  VALUES (1007, 'MULTILINESTRING((1 1, 1 3, 6 3),
                                   (10 5, 20 1))');

INSERT INTO sample_shapes
  VALUES (1008, 'MULTIPOLYGON(((1 1, 1 3, 6 3, 6 0, 1 1)),
                                ((10 5, 10 10, 20 10, 20 5, 10 5)))');

INSERT INTO sample_shapes
  VALUES (1009, 'GEOMETRYCOLLECTION( POINT(10 10),
                                       POINT(30 30),
                                       LINESTRING(15 15, 20 20))');
```

```
INSERT INTO sample_shapes
VALUES (1010, 'GEOSEQUENCE( (10 20, 30 40, 50 60),
                             (2007-08-22 12:05:23.56,
                              2007-08-22 12:08:25.14,
                              2007-08-22 12:11:41.52),
                             (1, 2, 3),
                             (2, 10, 12, 11, 18, 21, 19) )' );

INSERT INTO sample_shapes
VALUES (1011, 'GEOSEQUENCE( (10 20, 30 40, 50 60),
                             (2008-03-17 10:34:03.53,
                              2008-03-17 10:38:25.21,
                              2008-03-17 10:41:41.48),
                             (1, 2, 3),
                             (0))' );

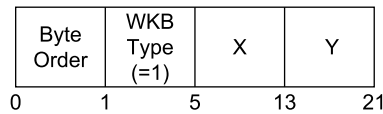
SELECT * FROM sample_shapes ORDER BY skey;
```

Well-Known Binary Format

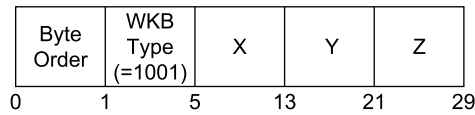
Defines the format of geometry data when using a byte stream to construct new ST_Geometry instances or transfer ST_Geometry values to and from client applications.

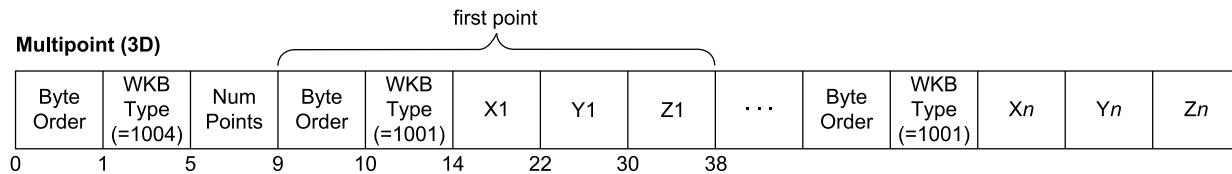
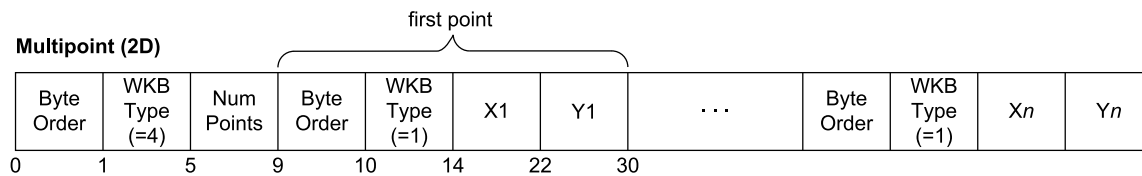
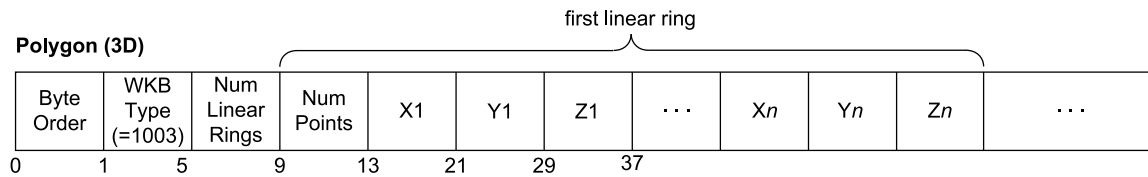
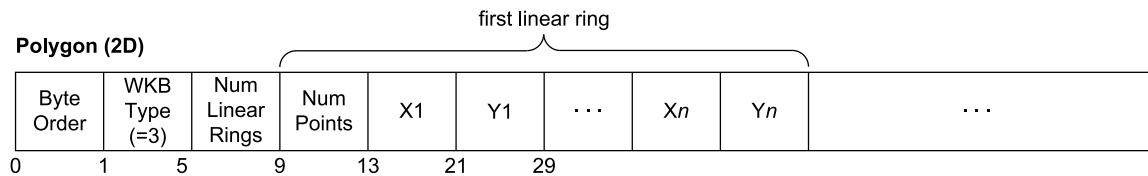
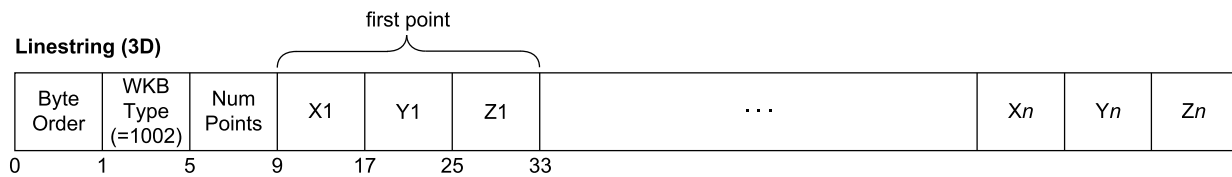
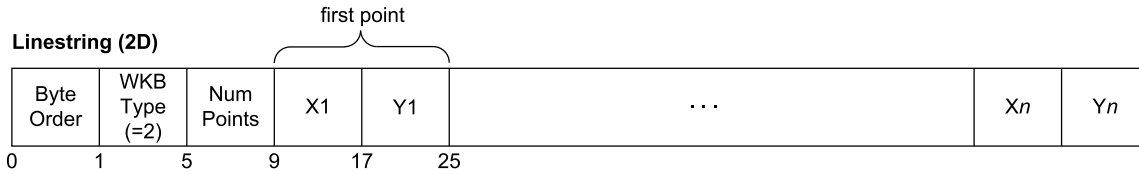
Format

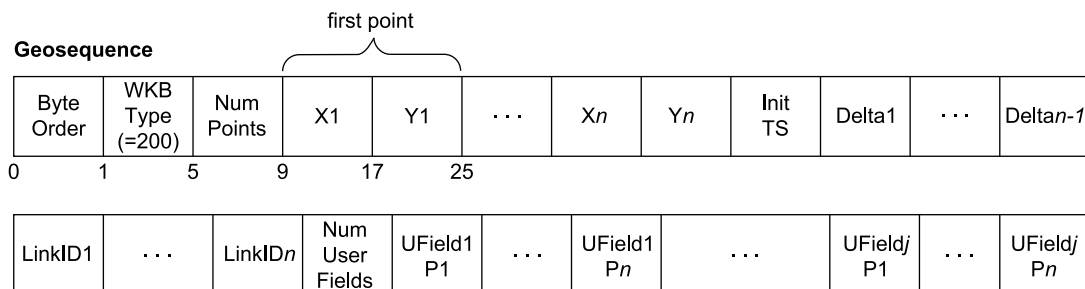
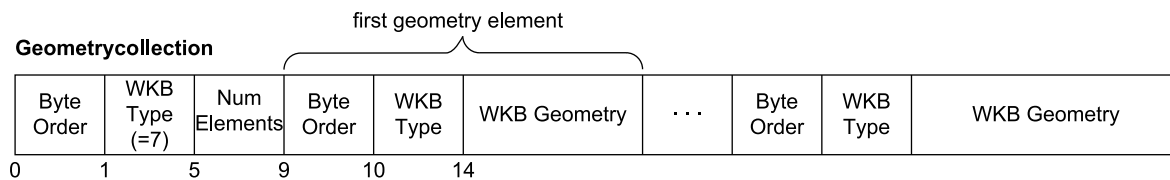
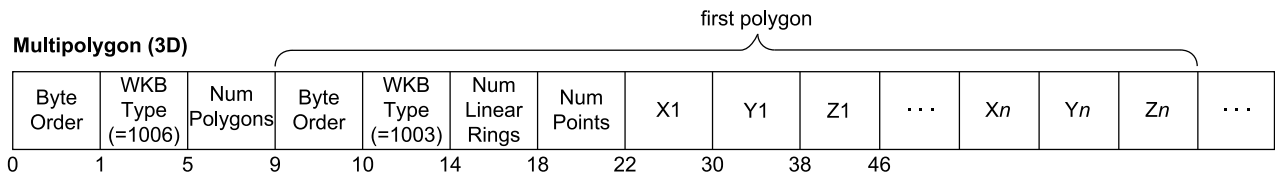
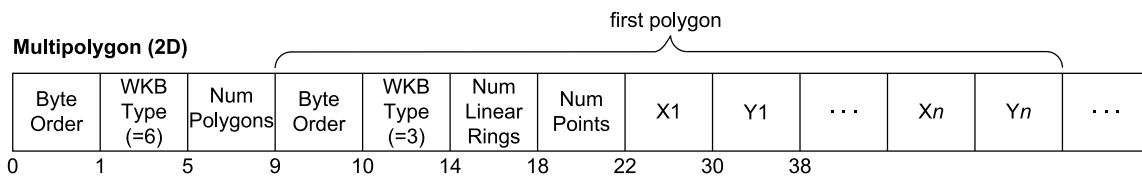
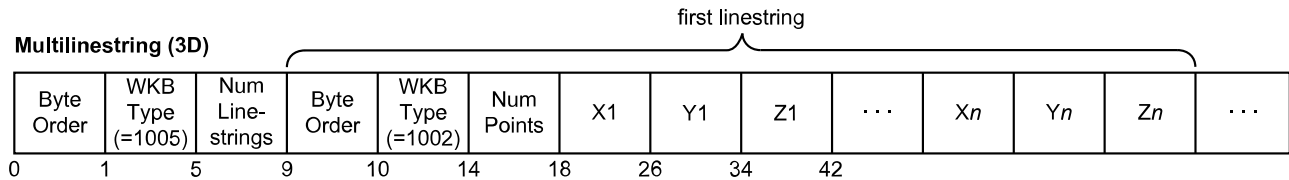
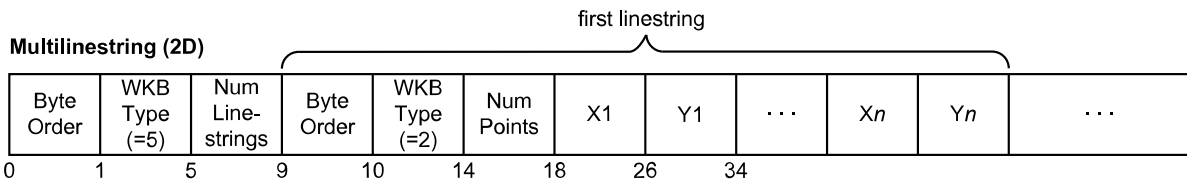
Point (2D)



Point (3D)







Field ...	Specifies ...
Byte Order	<p>an 8-bit value that indicates the binary representation of 32-bit unsigned integer values and 64-bit double precision values in the byte stream.</p> <p>0 indicates big endian. The first octet is the most significant octet. For double precision values, the sign bit is in the first octet.</p> <p>1 indicates little endian. The last octet is the most significant octet. For double precision values, the sign bit is in the last octet.</p>
WKB Type	<p>a 32-bit unsigned integer value that indicates the geometry represented by the byte stream.</p> <p>A value of:</p> <ul style="list-style-type: none"> • 1 means a point (2D). 1001 means a point (3D). • 2 means a linestring (2D). 1002 means a linestring (3D). • 3 means polygon (2D). 1003 means a polygon (3D). • 4 means multipoint (2D). 1004 means a multipoint (3D). • 5 means a multilinestring (2D). 1005 means a multilinestring (3D). • 6 means a multipolygon (2D). 1006 means a multipolygon (3D). • 7 means a geometry collection (2D). 1007 means a geometry collection (3D). • 200 means geosequence.
X	a 64-bit, double-precision value that represents the X coordinate value of an ST_Point value.
Y	a 64-bit, double-precision value that represents the Y coordinate value of an ST_Point value.
Z	a 64-bit, double-precision value that represents the Z coordinate value of an ST_Point value.
Num Points	a 32-bit, unsigned integer that specifies the number of points (X, Y coordinate pairs) that are to follow.
X1	a 64-bit, double precision value that represents the X coordinate value of the first point of a geometry.
Y1	a 64-bit, double precision value that represents the Y coordinate value of the first point of a geometry.
Z1	a 64-bit, double precision value that represents the Z coordinate value of the first point of a geometry.
X <i>n</i>	<p>a 64-bit, double precision value that represents the X coordinate value of the <i>n</i>th point of a geometry, where <i>n</i> is defined by the Num Points field that precedes the series of points.</p> <p>Note that if the preceding Num Points field specifies a value of 1, the series of points is defined by the X1, Y1, and Z1 fields only and does not require any more X or Y coordinates.</p>

Field ...	Specifies ...
Y <i>n</i>	a 64-bit, double precision value that represents the Y coordinate value of the <i>n</i> th point of a geometry, where <i>n</i> is defined by the Num Points field that precedes the series of points. Note that if the preceding Num Points field specifies a value of 1, the series of points is defined by the X1, Y1, and Z1 fields only and does not require any more X or Y coordinates.
Z <i>n</i>	a 64-bit, double precision value that represents the Z coordinate value of the <i>n</i> th point of a geometry, where <i>n</i> is defined by the Num Points field that precedes the series of points. Note that if the preceding Num Points field specifies a value of 1, the series of points is defined by the X1, Y1, and Z1 fields only and does not require any more X or Y coordinates.
Num Linear Rings	a 32-bit, unsigned integer that specifies the number of linear rings (closed and simple linestrings) that are to follow.
Num Linestrings	a 32-bit, unsigned integer that specifies the number of linestrings that are to follow.
Num Polygons	a 32-bit, unsigned integer that specifies the number of polygons that are to follow.
Num Elements	a 32-bit, unsigned integer that specifies the number of geometries that are to follow.
WKB Geometry	the fields that represent the geometry specified by the value of the preceding WKB Type field. For example, suppose the value of the WKB Type field in the first geometry element of a Geometry collection is 1, indicating that the first geometry is a point. The fields that follow must be X and Y.
Init TS	the initial timestamp associated with the first point of a geosequence, where the timestamp consists of the following fields: <ul style="list-style-type: none"> • Seconds is a 32-bit unsigned integer value where the first two digits represent the seconds value of the timestamp and the next six digits represent the fractional seconds of the timestamp. The valid range for the first two digits is 0 to 61. • Year is a 16-bit unsigned integer value that represents the year value of the timestamp. The valid range is from 1 to 9999. • Month is an 8-bit unsigned integer value that represents the month value of the timestamp. The valid range is from 1 to 12. • Day is an 8-bit unsigned integer value that represents the day value of the timestamp. The valid range is from 1 to 31, constrained by Gregorian calendar definitions. • Hour is an 8-bit unsigned integer value that represents the hour value of the timestamp. The valid range is from 0 to 23. • Minute is an 8-bit unsigned integer value that represents the minute value of the timestamp. The valid range is from 0 to 59.
Delta1	a 32-bit integer value that specifies the difference, in milliseconds, between the initial timestamp and the timestamp of the second point in the geosequence.
Delta <i>n-1</i>	a 32-bit integer value that specifies the difference, in milliseconds, between the penultimate timestamp and the timestamp of the last point in the geosequence.
LinkID1	a 64-bit NUMERIC(18,0) value that specifies the link ID of the first point of the geosequence. Link IDs are reserved for a future release.

Field ...	Specifies ...
LinkID n	a 64-bit NUMERIC(18,0) value that specifies the link ID of the n th point of the geosequence, where n is defined by the Num Points field. Link IDs are reserved for a future release.
Num User Fields	a 32-bit unsigned integer value that specifies the number of optional user fields of data for each point in the geosequence.
UField1 P1	a 64-bit double precision value that represents the first user field for the first point in the geosequence. Note that if the value of the Num User Fields field is zero, no user fields appear in the byte stream.
UField1 P n	a 64-bit double precision value that represents the first user field for the n th point in the geosequence, where n is defined by the Num Points field. Note that if the value of the Num User Fields field is zero, no user fields appear in the byte stream.
UField j P1	a 64-bit double precision value that represents the j th user field for the first point in the geosequence, where j is defined by the Num User Fields field. Note that if the value of the Num User Fields field is zero, no user fields appear in the byte stream.
UField j P n	a 64-bit double precision value that represents the j th user field for the n th point in the geosequence, where j is defined by the Num User Fields field and n is defined by the Num Points field. Note that if the value of the Num User Fields field is zero, no user fields appear in the byte stream.

Extended Well-Known Formats (ESRI-Compatible)

Extended Well-Known Text Format

The EWKT format simply prepends an SRID identifier to the existing WKT formats.

Syntax

SRID=*srid_value*;*WKT*

Syntax Elements

srid_value

An integer value greater than or equal to zero.

Usage Notes

As is the case for WKT, spaces are ignored.

To import or export geospatial data in EWKT format, you must use one of the "SRID" transform groups included with Vantage. For more information on transform groups, see [ST_Geometry Type Transforms](#).

Examples: EWKT Format for Geospatial Data

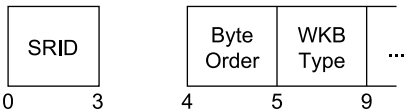
The following are valid geospatial data in EWKT format.

```
SRID=12;POINT(10 20)
SRID=35234;LINESTRING(10 20, 40 50)
SRID = 112 ; POINT(10 20)
```

Extended Well-Known Binary Format

Similar to the EWKT format, the EWKB format prepends an SRID value to the existing WKB format.

Format



Usage Notes

To import or export geospatial data in EWKB format, you must use one of the "SRID" transform groups included with Vantage. For more information on transform groups, see [ST_Geometry Type Transforms](#).

ST_Geometry Type Transforms

Learn about the default ST_Geometry transform, plus additional transform groups that support ST_Geometry data.

Default ST_Geometry Transform

By default, Vantage imports and exports geometry data to and from the server as CLOB data in the well-known text (WKT) standard format for geospatial data. Client applications can insert CLOB values in WKT format into an ST_Geometry column for the geospatial subtypes that ST_Geometry can represent.

Similarly, for queries that select data from an ST_Geometry column and export it to a client application, Vantage exports the data to the client application as CLOB values in WKT format.

Additional Transform Groups for ST_Geometry Data

To accommodate importing and exporting geospatial data from and to other types and formats, Vantage provides other transform groups for the ST_Geometry type.

- To override the default CLOB/WKT transform for geospatial data for the current session, use the SET TRANSFORM GROUP FOR TYPE statement.
- To override the default CLOB/WKT transform for geospatial data for a particular user or profile, specify the transform to use in the CREATE/MODIFY USER and CREATE/MODIFY PROFILE statements.

For more information about these SQL statements, see *Teradata Vantage™ - SQL Data Definition Language Syntax and Examples*, B035-1144.

The following table lists the transform groups that are available for the ST_Geometry type.

Transform Group	Import/Export Type	Default	Format
ST_WellKnownText	CLOB (Latin character set)	Yes	WKT This corresponds to the ST_GEOMETRY_IO transform group in previous database releases.
ST_WellKnownBinary	BLOB	No	Well-known binary (WKB)
TD_GEO_VARCHAR	VARCHAR(64000) (Latin character set)	No	WKT
TD_GEO_VARBYTE	VARBYTE(64000)	No	WKB
ST_WellKnownTextSRID	CLOB (Latin character set)	No	Extended WKT (WKT and SRID)
ST_WellKnownBinarySRID	BLOB	No	Extended WKB (WKB and SRID)
TD_GEO_VARCHAR_SRID	VARCHAR(64000) (Latin character set)	No	Extended WKT
TD_GEO_VARBYTE_SRID	VARBYTE(64000)	No	Extended WKB

For more information on the WKT, WKB, and extended formats, see [Geospatial Data Formats](#).

Transform Group Macros

You can use the following macros to find the transform group for a UDT (or CDT), or the transform group settings for a user, profile, or current session.

Macro	Description
SYSUDTLIB.HelpCurrentUserTransforms	Lists the transform group settings of the current logon user.
SYSUDTLIB.HelpCurrentSessionTransforms	Lists the transform group settings of the current session.
SYSUDTLIB.HelpUserTransforms(<i>User</i>)	Lists the transform group settings for a specific user.
SYSUDTLIB.HelpCurrentUDTTransform(<i>UDT</i>)	Lists the transform group settings of the current session for the specified UDT.
SYSUDTLIB.HelpUDTTransform(<i>User</i> , <i>UDT</i>)	Lists the transform group for a UDT for a user.
SYSUDTLIB.HelpProfileTransforms(<i>Profile</i>)	Lists the transform group settings for a specific profile.
SYSUDTLIB.HelpProfileTransform(<i>Profile</i> , <i>UDT</i>)	Lists the transform group for a UDT for a profile.

Example: Exporting Geometry Data as CLOB in WKT Format

Consider the following definition of a table that defines an ST_Geometry column for representing ST_Point values.

```
CREATE TABLE sample_points1 (skey INTEGER, point1 ST_Geometry);
```

By default, the following query issued through a client application results in the export of values in the point1 column using the WKT representation of a point:

```
SELECT *
FROM sample_points1;

SKEY          POINT1
-----
1001          POINT(10 20)
```

For more information on the WKT format of the ST_Geometry type, see [Well-Known Text Format](#).

Loading (Importing) and Unloading (Exporting) Geospatial Data

WKT Format

Applications that have geospatial data in the WKT format can insert the data into ST_Geometry columns by specifying the values as any of the following expressions:

- CLOB or VARCHAR constant in the WKT format

- NEW expression with the VARCHAR or CLOB form of the ST_Geometry constructor
- SYSSPATIAL.ST_GeomFromText function
- ST_Geometry.ST_WKTToSql method

Applications that require geospatial data in the WKT format can select directly from an ST_Geometry column using the default transform group for the ST_Geometry type. The returned data type is CLOB.

Alternatively, you can use one of the other provided transform groups to load and unload geospatial data as other types or formats. For more information, see [ST_Geometry Type Transforms](#).

WKB Format

Applications that have geospatial data in the WKB format can insert the data into ST_Geometry columns by specifying the values as any of the following expressions:

- NEW expression with the VARBYTE or BLOB form of the ST_Geometry constructor
- SYSSPATIAL.ST_GeomFromWKB function
- ST_Geometry.ST_WKBToSql method

Applications that require geospatial data in the WKB format can directly select from a ST_Geometry column using the ST_AsBinary method using the default transform group for the ST_Geometry type. The returned data type is BLOB.

Alternatively, you can use one of the other provided transform groups to load and unload geospatial data as other types or formats. For more information, see [ST_Geometry Type Transforms](#).

TDGeoImport and TDGeoExport Utilities

TDGeoImport and TDGeoExport are utilities that Teradata provides to interconvert between the database representation of geospatial data and formats compatible with the ESRI, MapInfo, and TIGER/Line data formats. For more information on these utilities, see *Teradata® Geospatial Utilities User Guide*, B035-2519.

ST_Geometry Constructors and Methods

ST_Geometry Constructor (BLOB Form)

Constructor method that returns a specified ST_Geometry value.

Use this form of ST_Geometry to construct all ST_Geometry types.

ST_Geometry Constructor Syntax (BLOB Form)

```
ST_Geometry ( wkb [, asrid ] )
```

Syntax Elements

wkb

A BLOB for the well-known binary representation of the spatial type.

The maximum size of *wkb* is approximately 16 MB (the exact maximum is 16 MB - 1 KB), allowing for a representation of approximately one million points.

For details on well-known binary formats, see [Well-Known Binary Format](#).

asrid

An optional INTEGER value for a spatial reference system identifier.

If this argument is omitted, the spatial reference system identifier is set to 0.

Example: ST_Geometry Constructor (BLOB Form)

```
USING (a INTEGER,  
      b BLOB(60000))  
INSERT INTO sample_shapes VALUES (:a, NEW ST_Geometry(:b));
```

ST_Geometry Constructor (CLOB Form)

Constructor method that returns a specified ST_Geometry value.

Use this form of ST_Geometry to construct all ST_Geometry types.

ST_Geometry Constructor Syntax (CLOB Form)

```
ST_Geometry ( wkt [, asrid ] )
```

Syntax Elements

wkt

A CLOB for the well-known text representation of the spatial type.

The maximum size of *wkt* is approximately 16 MB (the exact maximum is 16 MB - 1 KB), allowing for a representation of approximately one million points.

For details on well-known text formats, see [Well-Known Text Format](#).

asrid

An optional INTEGER for a spatial reference system identifier.

If this argument is omitted, the spatial reference system identifier is set to 0.

Example: ST_Geometry Constructor (CLOB Form)

```
USING (a INTEGER,  
      b CLOB(60000))  
INSERT INTO sample_shapes VALUES (:a, NEW ST_Geometry(:b));
```

ST_Geometry Constructor (VARBYTE Form)

Constructor method that returns a specified ST_Geometry value.

Use this form of ST_Geometry to construct all ST_Geometry types.

ST_Geometry Constructor Syntax (VARBYTE Form)

```
ST_Geometry ( wkb [, asrid ] )
```

Syntax Elements

wkb

A VARBYTE(64000) for the well-known binary representation of the spatial type. The size of *wkb* cannot exceed 64000 bytes.

When a WKB representation is larger than 64000 bytes, use the BLOB version of the ST_Geometry constructor method.

For more information on well-known binary formats, see [Well-Known Binary Format](#).

asrid

An optional INTEGER value for the spatial reference system identifier.

If this argument is omitted, the spatial reference system identifier is set to 0.

Example: ST_Geometry Constructor (VARBYTE Form)

```
USING (a INTEGER,
      b VARBYTE(60000))
INSERT INTO sample_shapes VALUES (:a, NEW ST_Geometry(:b));
```

ST_Geometry Constructor (VARCHAR Form)

Constructor method that returns a specified ST_Geometry value.

Use this form of ST_Geometry to construct all ST_Geometry types.

ST_Geometry Constructor Syntax (VARCHAR Form)

```
ST_Geometry ( wkt [, asrid ] )
```

Syntax Elements

wkt

A VARCHAR(64000) for the well-known text representation of the spatial type. The size of the *wkt* argument cannot exceed 64000 bytes.

When a well-known text representation is larger than 64000 bytes, use the CLOB version of the ST_Geometry constructor method.

For more information on well-known text formats, see [Well-Known Text Format](#).

asrid

An optional INTEGER value for a spatial reference system identifier.

If this argument is omitted, the spatial reference system identifier is set to 0.

Example: ST_Geometry Constructor (VARCHAR Form)

```

USING (a INTEGER,
      b VARCHAR(60000))
INSERT INTO sample_shapes VALUES (:a, NEW ST_Geometry(:b));

```

Make_2D Method

Converts a 3D ST_Geometry value to a 2D ST_Geometry value, and optionally validates that the 2D geometry is well formed. This method strips the z coordinate from 3D geometries.

Valid Data Types

All ST_Geometry types.

Result Type

Returns the 2D version of the ST_Geometry value on which the method is called.

Make_2D Syntax

```
Make_2D ( validate )
```

Syntax Elements

validate

An INTEGER value that determines whether the converted 2D geometry is tested to ensure it is well formed:

- If *validate* is 1, the 2D geometry is tested. If it is not well formed, the system returns an error.
- If *validate* is not equal to 1 or is not specified, the 2D geometry is not tested.

Usage Notes

- The ST_IsValid method can be used to validate converted 2D geometries that were not validated by Make_2D.
- If the input geometry is already a 2D geometry value, this value is returned unchanged by this method.
- A well-formed 3D geometry value can become invalid during a 3D to 2D conversion. For example, if a 3D polygon lies in a plane parallel to the y axis, its line segments will self-intersect. Make_2D does not attempt to change the geometry type of the converted geometry from polygon to linestring to account for this, and an invalid 2D geometry can result.

Example: Make_2D

```
SELECT shape.Make_2D()
FROM sample_shapes;
```

MBB Method

Returns the 3D minimum bounding box (MBB) that encloses a 3D ST_Geometry.

Valid Data Types

All ST_Geometry types.

Result Type

MBB value.

MBB Syntax

```
MBB ( )
```

Usage Notes

MBB can return a two-dimensional MBB, if appropriate. Such an MBB has z coordinates that are all zero.

If the 3D input geometry is empty, all of the coordinates of the resulting MBB will be zero.

Example: MBB

```
SELECT shape.MBB()
FROM sample_shapes;
```

SimplifyPreserveTopology Method

Simplifies a 2D geometry by removing points that would fall within a specified distance tolerance.

Valid Data Types

- LineString
- MultiLineString
- Polygon
- MultiPolygon

The preceding types can be within Geometry Collections.

If called on another type of 2D geometry, SimplifyPreserveTopology returns it unchanged.

Result Type

Returns an ST_Geometry value.

SimplifyPreserveTopology Syntax

```
SimplifyPreserveTopology ( tolerance )
```

Syntax Elements

tolerance

A FLOAT that represents the distance tolerance for the simplification. The simplified line will never be farther away from the original line than the *tolerance* value.

Usage Notes

- The simplification always returns a valid geometry.
- Simplified geometries require less storage space and fewer spatial operations during geospatial manipulations. Consequently operations on simplified geometries generally perform faster.
- Smaller tolerance values result in a geometry closer to the input geometry, but will remove fewer vertices. Larger tolerance values will remove more vertices, but the resulting simplified geometry will be less similar to the original input.

Example: SimplifyPreserveTopology

```
SELECT shape.SimplifyPreserveTopology(0.1)
FROM sample_shapes;
```

ST_3DDistance Method

Returns the distance between two ST_Geometry values. ST_3DDistance considers z coordinate values, if they exist in the geometries passed to it.

Valid Data Types

- ST_Point
- ST_MultiPoint
- ST_LineString
- ST_MultiLineString

To be valid with ST_3DDistance, these shapes must include z coordinates.

Result Type

Returns a FLOAT value. The distance units are those of the two input geometries.

Returns 0 if the two geometries intersect.

Returns NULL if the *ageometry* argument is NULL.

ST_3DDistance Syntax

```
ST_3DDistance ( ageometry )
```

Syntax Elements***ageometry***

The other ST_Geometry value.

Usage Notes

- Both geometries must include z coordinates.
- Both geometries must use the same spatial reference system (have the same SRS ID).
- If a geospatial index is defined on the geospatial data column to which this method is applied, the index will only be used if method appears in the WHERE clause of a query such that the distance between the geometries is less than or equal to a constant. For example:

```
... WHERE geom.ST_3DDistance(otherGeom) < 10

... WHERE 10 >= geom.ST_3DDistance(otherGeom)
```

Example: ST_3DDistance

```
INSERT INTO sample_shapes VALUES(1,'Point(10 20 30)');
INSERT INTO sample_shapes VALUES(2,
  'MultiPoint((10 20 30),(40 50 60), (70 80 80))');
INSERT INTO sample_shapes VALUES(3,
  'Linestring(10 20 30,40 50 60, 70 80 80)');
INSERT INTO sample_shapes VALUES(4,
  'MultiLinestring((10 20 30,40 50 60), (70 80 80, 90 100 110))');

SELECT key
FROM sample_shapes
WHERE shape.ST_3DDistance('POINT(10 20 30)') < 1E0;
```

ST_AsBinary Method

Returns the well-known binary (WKB) representation of an ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Result Type

Returns a BLOB value.

The maximum size of the return value is 16 MB (the exact maximum is 16 MB - 1 KB).

For details on WKB formats, see [Well-Known Binary Format](#).

ST_AsBinary Syntax

```
ST_AsBinary ()
```

Example: ST_AsBinary

```
CREATE TABLE blob_table (bkey INTEGER, b1 blob);

INSERT INTO blob_table
SELECT skey, shape.ST_AsBinary()
FROM sample_shapes
WHERE skey < 1009;
```

ST_AsText Method

Returns the well-known text (WKT) representation of an ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Result Type

Returns a CLOB value.

For details on WKT formats, see [Well-Known Text Format](#).

ST_AsText Syntax

```
ST_AsText ()
```

Example: ST_AsText

```
CREATE TABLE sample_streets(  
  skey INTEGER,  
  streetName VARCHAR(40),  
  streetShape ST_GEOMETRY);  
  
INSERT INTO sample_streets  
  VALUES(1, 'Main Street', 'LINESTRING(2 2, 3 2, 4 1)');  
INSERT INTO sample_streets  
  VALUES(1, 'Coast Blvd', 'LINESTRING(12 12, 18 17)');  
  
SELECT streetShape.ST_AsText()  
FROM sample_streets  
WHERE skey < 1001;
```

ST_Boundary Method

Returns the boundary of the ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Result Type

Returns an ST_Geometry value.

ST_AsBoundary Syntax

```
ST_AsBoundary ( )
```

Usage Notes

The boundary of an ST_Geometry value is a set of ST_Geometry values of the next lower dimension.

IF the ST_Geometry represents ...	THEN the boundary ...
an ST_Point or ST_MultPoint value	is the empty set.
a nonclosed ST_LineString or GeoSequence value	consists of the start and end ST_Point values.
a closed ST_LineString or GeoSequence value	is empty.

IF the ST_Geometry represents ...	THEN the boundary ...
an ST_MultiLineString value	consists of ST_Point values that are in the boundaries of an odd number of its element ST_LineString values.
an ST_Polygon value	consists of its set of linear rings.
an ST_MultiPolygon value	consists of the set of linear rings of its ST_Polygon values.
an ST_GeomCollection where the interiors of the geometries in the collection are disjoint	consists of geometry values drawn from the boundaries of the element geometries, where the geometry values are in the boundaries of an odd number of element geometries.

Example: ST_Boundary

```
SELECT skey
FROM sample_shapes
WHERE shape.ST_Boundary().ST_IsEmpty() = 1;
```

ST_Buffer Method

Returns all points whose distance from an ST_Geometry value is less than or equal to a specified distance.

Valid Data Types

All ST_Geometry types.

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is dropped, and only x and y coordinates are returned.

Result Type

Returns an ST_Geometry value.

ST_Buffer Syntax

```
ST_Buffer ( adistance )
```

Syntax Elements

adistance

A FLOAT for the distance from the geometry value to the buffer.

Usage Notes

The *adistance* argument is measured in the linear units of measure in the spatial reference system of the ST_Geometry value.

Example: ST_Buffer

```
SELECT shape.ST_Buffer(1E1)
FROM sample_shapes
WHERE skey = 1098;
```

ST_Centroid Method

Returns the mathematical centroid of an ST_Polygon or ST_MultiPolygon value.

Valid Data Types

ST_Polygon or ST_MultiPolygon.

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is dropped, and only x and y coordinates are returned.

Result Type

Returns an ST_Point value.

ST_Centroid Syntax

```
ST_Centroid ()
```

Example: ST_Centroid

```
SELECT cityName, cityShape.ST_Centroid()
FROM sample_cities;
```

ST_Contains Method

Tests if an ST_Geometry value spatially contains another ST_Geometry value.

Valid Data Types

All ST_Geometry types except geometry collections.

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is ignored in method calculations.

Result Type

Returns an INTEGER value:

- 1 if the input geometry contains *ageometry*
- 0 if the input geometry does not contain *ageometry*

ST_Contains Syntax

```
ST_Contains ( ageometry )
```

Syntax Elements

ageometry

The other ST_Geometry value.

Usage Notes

Using a geospatial index can greatly improve the performance of queries that use this method.

Example: ST_Contains

```

SELECT streetName, cityName
FROM sample_cities, sample_streets
WHERE cityShape.ST_Contains(streetShape) = 1
ORDER BY cityName;
```

ST_ConvexHull Method

Returns the convex hull of the ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is ignored in method calculations. Consequently, any z coordinates returned by this method should be ignored. Teradata recommends using the `Make_2D` method to strip out the z coordinates of the return value.

Result Type

Returns an ST_Geometry value.

ST_ConvexHull Syntax

```
ST_ConvexHull ()
```

Example: ST_ConvexHull

```
SELECT cityName, cityShape.ST_ConvexHull()  
FROM sample_cities;
```

ST_CoordDim Method

Returns the coordinate dimension of a geometry.

Valid Data Types

Any ST_Geometry type.

Result Type

Returns a SMALLINT value:

- 1, if the input geometry is 1D
- 2, if the input geometry is 2D
- 3, if the input geometry is 3D

ST_CoordDim Syntax

```
ST_CoordDim ()
```

Usage Notes

The coordinate dimension is the same as the coordinate dimension of the spatial reference system for the ST_Geometry value.

Example: ST_CoordDim

```
SELECT sample_shapes.shape.ST_CoordDim();
```

ST_Crosses Method

Tests if an ST_Geometry value spatially crosses another ST_Geometry value.

Valid Data Types

All ST_Geometry types except geometry collections.

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is ignored in method calculations.

Result Type

Returns an INTEGER value:

- 1, if the geometries cross
- 0, if the geometries do not cross

ST_Crosses Syntax

```
ST_Crosses ( ageometry )
```

Syntax Elements

ageometry

The other ST_Geometry value.

Usage Notes

Using a geospatial index can greatly improve the performance of queries that use this method.

Example: ST_Crosses

```
SELECT streetName
FROM sample_streets
WHERE streetShape.ST_Crosses(
```

```
NEW ST_Geometry('LINESTRING(2 2, 3 2, 4 1)') = 1
ORDER BY streetName;
```

ST_Difference Method

Returns an ST_Geometry value that represents the point set difference of two ST_Geometry values.

Valid Data Types

All ST_Geometry types except geometry collections.

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is ignored in method calculations. Consequently, any z coordinates returned by this method should be ignored. Teradata recommends using the Make_2D method to strip out the z coordinates of the return value.

Result Type

Returns an ST_Geometry type where the representation is one of the possible set of types in the following table, depending on the parameter types.

a — b	∅	ST_ Point	ST_ LineString, GeoSequence	ST_ Polygon	ST_ MultiPoint	ST_ MultiLineString	ST_ MultiPolygon
∅	∅	∅	∅	∅	∅	∅	∅
ST_Point	R01	R09	R09	R09	R09	R09	R09
ST_ LineString, GeoSequence	R02	R02	R08	R08	R02	R08	R08
ST_Polygon	R03	R03	R03	R14	R14	R14	R14
ST_MultiPoint	R04	R13	R13	R13	R13	R13	R13
ST_ MultiLineString	R05	R05	R08	R08	R05	R08	R08
ST_ MultiPolygon	R06	R06	R06	R14	R06	R05	R06

Where:

R01 = ST_Point
R02 = ST_LineString
R03 = ST_Polygon

R06 = ST_MultiPolygon
R08 = ∅, ST_LineString, ST_MultiLineString
R09 = ∅, ST_Point
R13 = ∅, ST_Point, ST_MultiPoint

R04 = ST_MultiPoint R05 = ST_MultiLineString	R14 = Ø, ST_Polygon, ST_MultiPolygon
---	--------------------------------------

Vantage converts GeoSequence types that are involved in the ST_Difference method to ST_LineString values. Therefore, ST_Difference never returns a GeoSequence type.

ST_Difference Syntax

```
ST_Difference ( ageometry )
```

Syntax Elements

ageometry

The other ST_Geometry value.

Example: ST_Difference

```
SELECT shape.ST_Difference(NEW ST_Geometry('LINESTRING(2 2, 3 2, 4 1)'))
FROM sample_shapes
WHERE skey = 1067;
```

ST_Dimension Method

Returns the dimension of the ST_Geometry type.

Valid Data Types

All ST_Geometry types.

Result Type

Returns a SMALLINT value:

- 0, for a 0-dimensional geometry
- 1, for a 1D geometry
- 2, for a 2D geometry
- -1, if the input geometry is empty

ST_Dimension Syntax

```
ST_Dimension ()
```

Example: ST_Dimension

```
SELECT skey
FROM sample_shapes
WHERE shape.ST_Dimension() = 1
ORDER BY skey;
```

ST_Disjoint Method

Tests if an ST_Geometry value is spatially disjoint from another ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is ignored in method calculations.

Result Type

Returns an INTEGER value:

- 1, if the geometries are spatially disjoint
- 0, if the geometries are not spatially disjoint

ST_Disjoint Syntax

```
ST_Disjoint ( ageometry )
```

Syntax Elements

ageometry

The other ST_Geometry value.

Example: ST_Disjoint

```
SELECT streetName
FROM sample_streets
WHERE streetShape.ST_Disjoint(
    NEW ST_Geometry('LINESTRING(2 2, 3 2, 4 1)')) = 0
ORDER BY streetName;
```

ST_Distance Method

Returns the distance between two ST_Geometry values.

Valid Data Types

All ST_Geometry types.

Result Type

Returns a FLOAT value.

ST_Distance Syntax

```
ST_Distance ( ageometry )
```

Syntax Elements

ageometry

The other ST_Geometry value.

Usage Notes

- ST_Distance returns NULL if the geometry value passed to it is the empty set.
- Both geometries must use the same spatial reference system (have the same SRS ID).
- If a geospatial index is defined on the geospatial data column to which this method is applied, the index will only be used if method appears in the WHERE clause of a query such that the distance between the geometries is less than or equal to a constant. For example:

```
... WHERE geom.ST_Distance(otherGeom) < 10
```

```
... WHERE 10 >= geom.ST_Distance(otherGeom)
```

Example: ST_Distance

```
SELECT skey
FROM sample_shapes
WHERE shape.ST_Distance('LINESTRING(2 2, 3 2, 4 1)')
< 1E0;
```

ST_Envelope Method

Returns the bounding rectangle for the ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is dropped, and only x and y coordinates are returned.

Result Type

Returns an ST_Geometry value.

ST_Envelope Syntax

```
ST_Envelope ( )
```

Example: ST_Envelope

```
SELECT streetName, streetShape.ST_Envelope()
FROM sample_streets
WHERE skey = 1044;
```

ST_Equals Method

Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Result Type

Returns an INTEGER value:

- 1, if the geometries are spatially equal
- 0, if the geometries are not spatially equal

ST_Equals Syntax

```
ST_Equals ( ageometry )
```

Syntax Elements

ageometry

The other ST_Geometry value.

Usage Notes

Using a geospatial index can greatly improve the performance of queries that use this method.

Example: ST_Equals

```

SELECT streetName
FROM sample_streets
WHERE streetShape.ST_Equals(
    NEW ST_Geometry('LINESTRING(2 2, 3 2, 4 1)')) = 1
ORDER BY streetName;

```

ST_GeometryType Method

Returns the ST_Geometry type of the ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Result Type

Returns a VARCHAR(128) value that indicates the ST_Geometry type. The return value can be any of these strings:

- 'ST_Point'
- 'ST_LineString'
- 'ST_Polygon'
- 'ST_MultiPoint'
- 'ST_MultiLineString'
- 'ST_MultiPolygon'
- 'ST_GeomCollection'
- 'GeoSequence'

ST_GeometryType Syntax

```
ST_GeometryType ( )
```

Example: ST_GeometryType

```
SELECT skey
FROM sample_shapes
WHERE shape.ST_GeometryType() = 'ST_LineString';
```

ST_Intersection Method

Returns an ST_Geometry type where the value represents the point set intersection of two ST_Geometry values.

Valid Data Types

All ST_Geometry types, including GeoSequence, but not the ST_GeomCollection type.

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is ignored in method calculations. Consequently, any z coordinates returned by this method should be ignored. Teradata recommends using the Make_2D method to strip out the z coordinates of the return value.

Result Type

The type that the ST_Geometry return value represents is one from the possible set of types in the following table, depending on the argument types.

$a \cap b$	\emptyset	ST_Point	ST_LineString, GeoSequence	ST_Polygon	ST_MultiPoint	ST_MultiLineString	ST_MultiPolygon
\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset
ST_Point	\emptyset	R09	R09	R09	R09	R09	R09
ST_LineString, GeoSequence	\emptyset	R09	R11	R11	R13	R11	R11
ST_Polygon	\emptyset	R09	R11	R12	R13	R11	R12
ST_MultiPoint	\emptyset	R09	R13	R13	R13	R13	R13
ST_MultiLineString	\emptyset	R09	R11	R11	R13	R11	R11
ST_MultiPolygon	\emptyset	R09	R11	R12	R13	R11	R12

Where:

R09 = Ø, ST_Point	R12 = Ø, ST_Point, ST_LineString, ST_Polygon, ST_MultiPoint, ST_MultiLineString, ST_MultiPolygon
R11 = Ø, ST_Point, ST_LineString, ST_MultiPoint, ST_MultiLineString	R13 = Ø, ST_Point, ST_MultiPoint

Vantage converts GeoSequence types that are involved in the ST_Intersection method to ST_LineString values. Therefore, ST_Intersection never returns a GeoSequence type.

ST_Intersection Syntax

```
ST_Intersection ( ageometry )
```

Syntax Elements

ageometry
The other ST_Geometry value.

Example: ST_Intersection

```
INSERT INTO sample_shapes
SELECT skey, new ST_Geometry(
  'LINESTRING (2 2,3 2,4 1)').ST_Intersection(streetShape)
FROM sample_streets
WHERE new ST_Geometry(
  'MultiPoint((2 2),(3 2))').ST_Intersects(streetShape) = 1;
```

ST_Intersects Method

Tests if an ST_Geometry value spatially intersects another ST_Geometry value.

Valid Data Types

All ST_Geometry types except geometry collections.

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is ignored in method calculations.

Result Type

Returns an INTEGER value:

- 1, if the geometries intersect

- 0, if the geometries do not intersect

ST_Intersects Syntax

```
ST_Intersects ( ageometry )
```

Syntax Elements

ageometry

The other ST_Geometry value.

Usage Notes

Using a geospatial index can greatly improve the performance of queries that use this method.

Example: ST_Intersects

```
SELECT streetName
FROM sample_streets
WHERE streetShape.ST_Intersects(
    NEW ST_Geometry('LINESTRING(2 2, 3 2, 4 1)')) = 1
ORDER BY streetName;
```

ST_Is3D Method

Tests if an ST_Geometry value has z coordinate values.

Valid Data Types

All ST_Geometry types.

Result Type

Returns an INTEGER value:

- 1, if the ST_Geometry contains z coordinates
- 0, if the ST_Geometry does not contain z coordinates

ST_Is3D Syntax

```
ST_Is3D ( )
```

Example: ST_Is3D

```
SELECT skey
FROM sample_shapes
WHERE shape.ST_Is3D() = 1
ORDER BY skey;
```

ST_IsEmpty Method

Tests if an ST_Geometry value corresponds to the empty set.

Valid Data Types

All ST_Geometry types.

Result Type

Returns an INTEGER value:

- 1, if the geometry is empty
- 0, if the geometry is not empty

ST_IsEmpty Syntax

```
ST_IsEmpty ()
```

Example: ST_IsEmpty

```
SELECT skey
FROM sample_shapes
WHERE shape.ST_IsEmpty() = 1
ORDER BY skey;
```

ST_IsSimple Method

Tests if an ST_Geometry value has no anomalous geometric points, such as self intersection tangency.

Valid Data Types

Any ST_Geometry type.

Result Type

Returns an INTEGER value:

- 1, if the geometry is simple, with no anomalous points
- 0, if the geometry is not simple

ST_IsSimple Syntax

```
ST_IsSimple ()
```

Example: ST_IsSimple

```
SELECT skey
FROM sample_shapes
WHERE shape.ST_IsSimple() = 0
ORDER BY skey;
```

ST_IsValid Method

Tests if an ST_Geometry value is well formed.

Valid Data Types

All ST_Geometry types.

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is ignored in method calculations.

Result Type

Returns an INTEGER value:

- 1, if the geometry is valid
- 0, if the geometry is not valid

ST_IsValid Syntax

```
ST_IsValid ()
```

Example: ST_IsValid

```
SELECT skey
FROM sample_shapes
```

```
WHERE shape.ST_IsValid() = 0
ORDER BY skey;
```

ST_MBR Method

Returns the minimum bounding rectangle (MBR) of an ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Result Type

Returns an MBR value.

ST_MBR Syntax

```
ST_MBR ( )
```

Example: ST_MBR

```
INSERT INTO sample_MBRs
SELECT skey, shape.ST_MBR()
FROM sample_shapes;
```

ST_MBR_Xmax Method [Deprecated]

Note:

This deprecated method was replaced by the ANSI-compliant method [ST_MaxX Method](#).

Returns the upper right X coordinate of the minimum bounding rectangle (MBR) of an ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Result Type

Returns a FLOAT value.

ST_MBR_Xmax Syntax

```
ST_MBR_Xmax ( )
```

Example: ST_MBR_Xmax

```
SELECT cityName
FROM sample_cities
WHERE cityShape.ST_MBR_Xmax() > 9E1;
```

ST_MBR_Xmin Method [Deprecated]

Note:

This deprecated method was replaced by the ANSI-compliant method [ST_MinX Method](#).

Returns the lower left X coordinate of the minimum bounding rectangle (MBR) of an ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Result Type

Returns a FLOAT value.

ST_MBR_Xmin Syntax

```
ST_MBR_Xmin ()
```

Example: ST_MBR_Xmin

```
SELECT cityName
FROM sample_cities
WHERE cityShape.ST_MBR_Xmin() < 9E1;
```

ST_MBR_Ymax Method [Deprecated]

Note:

This deprecated method was replaced by the ANSI-compliant method [ST_MaxY Method](#).

Returns the upper right Y coordinate of the minimum bounding rectangle (MBR) of an ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Result Type

Returns a FLOAT value.

ST_MBR_Ymax Syntax

```
ST_MBR_Ymax ()
```

Example: ST_MBR_Ymax

```
SELECT cityName
FROM sample_cities
WHERE cityShape.ST_MBR_Ymax() > 9E1;
```

ST_MBR_Ymin Method [Deprecated]**Note:**

This deprecated method was replaced by the ANSI-compliant method [ST_MinY Method](#).

Returns the lower left Y coordinate of the minimum bounding rectangle (MBR) of an ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Result Type

Returns a FLOAT value.

ST_MBR_Ymin Syntax

```
ST_MBR_Ymin ()
```

Example: ST_MBR_Ymin

```
SELECT cityName
FROM sample_cities
WHERE cityShape.ST_MBR_Ymin() < 9E1;
```

ST_MaxX Method

Returns the maximum x coordinate of an ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Result Type

Returns a FLOAT value.

Returns NULL if the ST_Geometry is an empty set.

ST_MaxX Syntax

```
ST_MaxX ()
```

Example: ST_MaxX

```
SELECT shape.ST_MaxX()  
FROM sample_shapes;
```

ST_MaxY Method

Returns the maximum y coordinate of an ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Result Type

Returns a FLOAT value.

Returns NULL if the ST_Geometry is an empty set.

ST_MaxY Syntax

```
ST_MaxY ()
```

Example: ST_MaxY

```
SELECT shape.ST_MaxY()  
FROM sample_shapes;
```

ST_MaxZ Method

Returns the maximum z coordinate of an ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Result Type

Returns a FLOAT value.

Returns NULL if the ST_Geometry is an empty set.

ST_MaxZ Syntax

```
ST_MaxZ ()
```

Example: ST_MaxZ

```
SELECT shape.ST_MaxZ()  
FROM sample_shapes;
```

ST_MinX Method

Returns the minimum x coordinate of an ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Result Type

Returns a FLOAT value.

Returns NULL if the ST_Geometry is an empty set.

ST_MinX Syntax

```
ST_MinX ()
```

Example: ST_MinX

```
SELECT shape.ST_MinX()  
FROM sample_shapes;
```

ST_MinY Method

Returns the minimum y coordinate of an ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Result Type

Returns a FLOAT value.

Returns NULL if the ST_Geometry is an empty set.

ST_MinY Syntax

```
ST_MinY ()
```

Example: ST_MinY

```
SELECT shape.ST_MinY()  
FROM sample_shapes;
```

ST_MinZ Method

Returns the minimum z coordinate of an ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Result Type

Returns a FLOAT value.

Returns NULL if the ST_Geometry is an empty set.

ST_MinZ Syntax

```
ST_MinZ ( )
```

Example: ST_MinZ

```
SELECT shape.ST_MinZ()
FROM sample_shapes;
```

ST_Overlaps Method

Tests if an ST_Geometry value spatially overlaps another ST_Geometry value.

Valid Data Types

ST_Overlaps is valid on all ST_Geometry types, with the exception of geometry collections.

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is ignored in method calculations.

Result Type

Returns an INTEGER value:

- 1, if the geometries overlap
- 0, if the geometries do not overlap

ST_Overlaps Syntax

```
ST_Overlaps ( ageometry )
```

Syntax Elements

ageometry

The other ST_Geometry value.

Usage Notes

Using a geospatial index can greatly improve the performance of queries that use this method.

Example: ST_Overlaps

```
SELECT cityName
FROM sample_cities
WHERE cityShape.ST_Overlaps(
    NEW ST_Geometry('POLYGON((1 1, 1 3, 6 3, 6 0, 1 1))')) = 1;
```

ST_Relate Method

Tests if an ST_Geometry value is spatially related to another ST_Geometry value.

Valid Data Types

All ST_Geometry types except geometry collections.

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is ignored in method calculations.

Result Type

Returns an INTEGER value:

- 1, if the spatial relationship between the geometries corresponds to one of the acceptable values represented by *amatrix*
- 0, if the spatial relationship between the geometries does not correspond to one of the acceptable values represented by *amatrix*

ST_Relate Syntax

```
ST_Relate ( ageometry, amatrix )
```

Syntax Elements

ageometry

The other ST_Geometry value.

amatrix

A CHARACTER(9) value where each character corresponds to a cell in the DE-9IM pattern matrix.

The mapping for *amatrix*, as defined by SQL/MM Spatial, is as follows:

Position	DE-9IM Cell
1	$(\text{Interior}(\text{SELF}) \cap \text{Interior}(\text{ageometry})).\text{ST_Dimension}$
2	$(\text{Interior}(\text{SELF}) \cap \text{Boundary}(\text{ageometry})).\text{ST_Dimension}$
3	$(\text{Interior}(\text{SELF}) \cap \text{Exterior}(\text{ageometry})).\text{ST_Dimension}$
4	$(\text{Boundary}(\text{SELF}) \cap \text{Interior}(\text{ageometry})).\text{ST_Dimension}$
5	$(\text{Boundary}(\text{SELF}) \cap \text{Boundary}(\text{ageometry})).\text{ST_Dimension}$
6	$(\text{Boundary}(\text{SELF}) \cap \text{Exterior}(\text{ageometry})).\text{ST_Dimension}$
7	$(\text{Exterior}(\text{SELF}) \cap \text{Interior}(\text{ageometry})).\text{ST_Dimension}$
8	$(\text{Exterior}(\text{SELF}) \cap \text{Boundary}(\text{ageometry})).\text{ST_Dimension}$
9	$(\text{Exterior}(\text{SELF}) \cap \text{Exterior}(\text{ageometry})).\text{ST_Dimension}$

Valid values for each character are: 'T', 'F', '0', '1', '2', and '*'.

The value specifies the set of acceptable values for an intersection at that given cell. The meaning is as follows.

Cell Value	Intersection Set Results
'T'	{ 0, 1, 2 }
'F'	{ -1 }
'0'	{ 0 }
'1'	{ 1 }
'2'	{ 2 }
'**'	{ -1, 0, 1, 2 }

Usage Notes

Using a geospatial index can greatly improve the performance of queries that use this method.

Example: ST_Relate

```
SELECT streetName
FROM sample_streets
WHERE streetShape.ST_Relate(
    NEW ST_Geometry('LINESTRING(2 2, 3 2, 4 1)'), '0*****') = 1
ORDER BY streetName;
```

ST_SRID Method

Get and set the spatial reference system identifier of the ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Result Type

- If no *asrid* argument is passed, returns an integer identifying the spatial reference system of the geometry.
- If an *asrid* argument is passed, returns the ST_Geometry with the spatial reference system identifier set to the specified spatial reference system

ST_SRID Syntax

```
ST_SRID ( [ asrid ] )
```

Syntax Elements

asrid

An INTEGER value for the spatial reference system identifier.

Example: ST_SRID

```
SELECT sample_shapes.shape.ST_SRID();
```

ST_SymDifference Method

Returns an ST_Geometry value that represents the point set symmetric difference of two ST_Geometry values.

Valid Data Types

All ST_Geometry types except geometry collections.

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is ignored in method calculations. Consequently, any z coordinates returned by this method should be ignored. Teradata recommends using the Make_2D method to strip out the z coordinates of the return value.

Result Type

The type that the ST_Geometry return type represents is one from the possible set of types in the following table, depending on the parameter types.

\cup (b — a)	\emptyset	ST_Point	ST_LineString, GeoSequence	ST_Polygon	ST_MultiPoint	ST_MultiLineString	ST_MultiPolygon
\emptyset	\emptyset	R01	R02	R03	R04	R05	R06
ST_Point	R01	R10	R15	R22	R10	R17	R19
ST_LineString, GeoSequence	R02	R15	R08	R21	R15	R08	R18
ST_Polygon	R03	R22	R21	R14	R22	R21	R14
ST_MultiPoint	R04	R10	R15	R22	R10	R17	R19
ST_MultiLineString	R05	R17	R08	R21	R17	R08	R18
ST_MultiPolygon	R06	R19	R18	R14	R19	R18	R14

Where:

R01 = ST_Point R02 = ST_LineString R03 = ST_Polygon R04 = ST_MultiPoint R05 = ST_MultiLineString R06 = ST_MultiPolygon R08 = \emptyset , ST_LineString, ST_MultiLineString R10 = \emptyset , ST_MultiPoint R14 = \emptyset , ST_Polygon, ST_MultiPolygon R15 = ST_LineString, ST_GeomCollection of ST_Point and ST_LineString values	R17 = ST_MultiLineString, ST_GeomCollection of ST_Point and ST_LineString values R18 = ST_MultiPolygon, ST_GeomCollection of ST_LineString and ST_Polygon values R19 = ST_MultiPolygon, ST_GeomCollection of ST_Point and ST_Polygon values R21 = ST_Polygon, ST_GeomCollection of ST_LineString and ST_Polygon values R22 = ST_Polygon, ST_GeomCollection of ST_Point and ST_Polygon values
---	--

Vantage converts GeoSequence types that are involved in the ST_SymDifference method to ST_LineString values. Therefore, ST_SymDifference never returns a GeoSequence type.

ST_SymDifference Syntax

```
ST_SymDifference ( ageometry )
```

Syntax Elements

ageometry

The other ST_Geometry value.

Example: ST_SymDifference

```

SELECT shape.ST_SymDifference(
    NEW ST_Geometry('LINESTRING(2 2, 3 2, 4 1)'))
FROM sample_shapes
WHERE skey = 1067;

```

ST_Touches Method

Tests if an ST_Geometry value spatially touches another ST_Geometry value.

Valid Data Types

All ST_Geometry types except geometry collections.

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is ignored in method calculations.

Result Type

Returns an INTEGER value:

- 1, if the geometries touch
- 0, if the geometries do not touch

ST_Touches Syntax

```
ST_Touches ( ageometry )
```

Syntax Elements

ageometry

The other ST_Geometry value.

Usage Notes

Using a geospatial index can greatly improve the performance of queries that use this method.

Example: ST_Touches

```
SELECT streetName
FROM sample_streets
WHERE streetShape.ST_Touches(
    NEW ST_Geometry('LINESTRING(2 2, 3 2, 4 1)')) = 1
ORDER BY streetName;
```

ST_Transform Method

Returns an ST_Geometry value transformed to the specified spatial reference system.

Valid Data Types

All ST_Geometry types.

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is ignored in method calculations. Consequently, any z coordinates returned by this method should be ignored. Teradata recommends using the Make_2D method to strip out the z coordinates of the return value.

Result Type

Returns an ST_Geometry value.

ST_Transform Syntax

```
ST_Transform ( [ toSRSid, ] toWktSRS, fromWktSRS )
```

Syntax Elements

toSRSid

An INTEGER that is the identifier of the spatial reference system returned by the method.

toWktSRS

A VARCHAR(2048) for the WKT representation of the spatial reference system to transform to.

fromWktSRS

A VARCHAR(2048) for the WKT representation of the spatial reference system to assign to the geometry value (without any transformation) before transforming it to the spatial reference system specified by *toWktSRS*.

Usage Notes

The SRTEXT column of the SYSSPATIAL.SPATIAL_REF_SYS metadata table contains WKT representations of spatial reference systems.

Examples: ST_Transform

```
SELECT R.skey, R.shape.ST_Transform(X.srtext, Y.srtext)
FROM sample_shapes R,
     SYSSPATIAL.SPATIAL_REF_SYS X,
     SYSSPATIAL.SPATIAL_REF_SYS Y
WHERE X.AUTH_SRID = 4326 and Y.AUTH_SRID = 3149;
```

The following example demonstrates the use of the *toSRSid* argument.

```
CREATE TABLE customers(pkey INTEGER, point ST_Geometry);
INSERT INTO customers VALUES(0, new ST_Geometry('POINT(-89.39 43.09)', 1619));
INSERT INTO customers VALUES(1, new ST_Geometry('POINT(-87.65 41.90)', 1619));

CREATE TABLE transformed_customers2(pkey INTEGER, point ST_Geometry);
INSERT INTO transformed_customers2
SELECT pkey,
       point.ST_Transform(3054, X.srtext, Y.srtext)
FROM customers,
     SYSSPATIAL.SPATIAL_REF_SYS X,
     SYSSPATIAL.SPATIAL_REF_SYS Y
WHERE X.AUTH_SRID = 32616 AND -- UTM 16 / WGS84
      Y.AUTH_SRID = 4326      -- WGS84
;
;SELECT pkey,
       point.ST_SRID(),
       point
FROM transformed_customers2
ORDER BY pkey;
```

pkey	point.ST_SRID()	point
0	3054	POINT (305475.753263251972385 4773581.672131018)
1	3054	POINT (446084.218845848692581 4638877.682686116)

ST_Union Method

Returns an ST_Geometry value that represents the point set union of two ST_Geometry values.

Valid Data Types

All ST_Geometry types except geometry collections.

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is ignored in method calculations. Consequently, any z coordinates returned by this method should be ignored. Teradata recommends using the Make_2D method to strip out the z coordinates of the return value.

Result Type

The type that the ST_Geometry return type represents is one from the possible set of types in the following table, depending on the parameter types.

a U b	Ø	ST_Point	ST_LineString, GeoSequence	ST_Polygon	ST_MultiPoint	ST_MultiLineString	ST_MultiPolygon
Ø	Ø	R01	R02	R03	R04	R05	R06
ST_Point	R01	R20	R15	R22	R04	R17	R19
ST_LineString, GeoSequence	R02	R15	R16	R21	R15	R16	R18
ST_Polygon	R03	R22	R21	R23	R22	R21	R23
ST_MultiPoint	R04	R04	R15	R22	R04	R17	R19
ST_MultiLineString	R05	R17	R16	R21	R17	R16	R18
ST_MultiPolygon	R06	R19	R18	R23	R19	R18	R23

Where:

R09 = ST_Point
 R02 = ST_LineString
 R03 = ST_Polygon
 R04 = ST_MultiPoint
 R05 = ST_MultiLineString
 R06 = ST_MultiPolygon
 R15 = ST_LineString, ST_GeomCollection of ST_Point and ST_LineString values
 R16 = ST_LineString, ST_MultiLineString
 R17 = ST_MultiLineString, ST_GeomCollection of ST_Point and ST_LineString values

R18 = ST_MultiPolygon, ST_GeomCollection of ST_LineString and ST_Polygon values
 R19 = ST_MultiPolygon, ST_GeomCollection of ST_Point and ST_Polygon values
 R20 = ST_Point, ST_MultiPoint
 R21 = ST_Polygon, ST_GeomCollection of ST_LineString and ST_Polygon values
 R22 = ST_Polygon, ST_GeomCollection of ST_Point ST_Polygon values
 R23 = ST_Polygon, ST_MultiPolygon

Vantage converts GeoSequence types that are involved in the ST_Union method to ST_LineString values. Therefore, ST_Union never returns a GeoSequence type.

ST_Union Syntax

```
ST_Union ( ageometry )
```

Syntax Elements

ageometry

The other ST_Geometry value.

Example: ST_Union

```
SELECT shape.ST_Union(NEW ST_Geometry('LINESTRING(2 2, 3 2, 4 1)'))
FROM sample_shapes
WHERE skey = 1067;
```

ST_Within Method

Tests if an ST_Geometry value is spatially within another ST_Geometry value.

Valid Data Types

All ST_Geometry types except geometry collections.

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is ignored in method calculations.

Result Type

Returns an INTEGER value:

- 1, if the geometry is within *ageometry*
- 0, if the geometry is not within *ageometry*

ST_Within Syntax

```
ST_Within ( ageometry )
```

Syntax Elements

ageometry

The other ST_Geometry value.

Usage Notes

Using a geospatial index can greatly improve the performance of queries that use this method.

Example: ST_Within

```
SELECT streetName, cityName
FROM sample_cities, sample_streets
WHERE streetShape.ST_Within(cityShape) = 1
ORDER BY cityName;
```

ST_WKBToSQL Method

Returns a specified ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Result Type

Returns an ST_Geometry value.

ST_WKBToSQL Syntax

```
ST_WKBToSQL ( awkb )
```

Syntax Elements

awkb

The geometry value specified in the well-known binary (WKB) format.

The data type of *awkb* is BLOB.

The maximum size of *awkb* is approximately 16 MB (the exact maximum is 16 MB - 1 KB), allowing for a representation of approximately one million points.

For more information on WKB formats, see [Well-Known Binary Format](#).

Usage Notes

Vantage provides this method because it is in the SQL/MM Spatial standard, where it is defined to implement transform functionality that allows importing the ST_Geometry type to the server.

For best performance, use a constructor method instead of ST_WKBToSQL to construct an instance of ST_Geometry.

Example: ST_WKBToSQL

```

USING (a INTEGER,
      b BLOB(60000))
INSERT INTO sample_shapes VALUES (:a,
    new ST_Geometry('POINT(1 2)').ST_WKBToSQL(:b));

```

ST_WKTToSQL Method

Returns a specified ST_Geometry value.

Valid Data Types

All ST_Geometry types.

Result Type

Returns an ST_Geometry value.

ST_WKTToSQL Syntax

```
ST_WKTToSQL ( awkt )
```

Syntax Elements

awkt

A geometry value specified in the well-known text (WKT) format.

The data type of *awkt* is CLOB.

The maximum size of *awkt* is approximately 16 MB (the exact maximum is 16 MB - 1 KB), allowing for a representation of approximately one million points.

For details on WKT formats, see [Well-Known Text Format](#).

Usage Notes

Vantage provides this method because it is in the SQL/MM Spatial standard, where it is defined to implement transform functionality that allows importing the ST_Geometry type to the server.

For best performance, use a constructor method instead of ST_WKTToSQL to construct an instance of ST_Geometry.

Example: ST_WKTToSQL

```
USING (a INTEGER,  
       b CLOB(60000))  
INSERT INTO sample_shapes VALUES (:a,  
    new ST_Geometry('POINT(1 2)').ST_WKTToSQL(:b));
```

ST_Point Methods

ST_Geometry Constructor (ST_Point Form)

Returns an ST_Geometry value that represents a 2D or 3D ST_Point.

Valid Data Types

Use this constructor to construct an ST_Geometry that represents a 2D or 3D ST_Point.

ST_Geometry Constructor Syntax (ST_Point Form)

```
ST_Geometry ( geomtype, xcoord, ycoord, [, zcoord ] [, asrid ] )
```

Syntax Elements

geomtype

A VARCHAR(80) for the type of spatial object to create.

The geomtype argument must be set to 'ST_POINT'.

xcoord

A FLOAT value for the x coordinate of the point. The value should be entered in floating point format.

ycoord

A FLOAT value for the y coordinate of the point. The value should be entered in floating point format.

zcoord

A FLOAT value for the z coordinate of the point, if the point is represented in 3D space. The value should be entered in floating point format.

Note:

If you specify a *zcoord*, you must also specify an *asrid*.

asrid

An optional INTEGER value for the spatial reference system identifier.

If this argument is omitted, the spatial reference system identifier is set to 0.

Note:

asrid is required if you also pass a *zcoord* argument.

Example: ST_Geometry Constructor (ST_Point Form)

```
INSERT INTO sample_shapes
VALUES (1101, NEW ST_Geometry('ST_POINT', 1E1, 2E1));

SELECT NEW ST_Geometry('ST_Point', 10.0, 20.0);
SELECT NEW ST_Geometry('ST_Point', 10.0, 20.0, 1699);
SELECT NEW ST_Geometry('ST_Point', 10.0, 20.0, 5.0, 1699);
```

ST_SphericalBufferMBR

Returns an MBR that represents the minimum and maximum latitude and longitude of all points within a given distance from a point. The earth is modeled as a sphere.

Valid Data Types

ST_Point

Result Type

Returns an MBR that represents the minimum and maximum latitude and longitude of all points within a given distance from the point.

ST_SphericalBufferMBR Syntax

```
ST_SphericalBufferMBR ( distance [, radius ] )
```

Syntax Elements

distance

A FLOAT value for the distance, in meters, from the point to calculate the MBR.

radius

An optional FLOAT value for the radius of the sphere used to model the earth. If omitted, the method uses a value of 6371000.0 meters.

Usage Notes

ST_SphericalBufferMBR and ST_SpheroidalBufferMBR basically perform the same function, but with different performance and accuracy. ST_SphericalBufferMBR models the earth as a simple sphere, so it performs better than ST_SpheroidalBufferMBR, but with less accuracy. ST_SpheroidalBufferMBR models the earth as a spheroid, so it returns a more accurate MBR, but it runs slower than ST_SphericalBufferMBR.

The returned MBR cannot cross the longitude value of +/-180 or the latitude value of +/-90. The MBR cannot cover more than 180 degrees of latitude or longitude on the sphere or spheroid.

Example: ST_SphericalBufferMBR

This example uses ST_SphericalBufferMBR to calculate an MBR that is 5000 meters from a point at longitude 100 and latitude 30:

```
SELECT
  NEW ST_Geometry('POINT(100 30)').ST_SphericalBufferMBR( 5000.0 );
```

ST_SphericalDistance

Returns the spherical distance between two spherical coordinates on the planet using the Haversine Formula. Both coordinates must be specified as ST_Point values.

Valid Data Types

ST_Point

Result Type

Returns a FLOAT.

ST_SphericalDistance Syntax

```
ST_SphericalDistance ( apoint )
```

Syntax Elements

apoint

An ST_Geometry for the other spherical coordinate.

Example: ST_SphericalDistance

```
CREATE TABLE sample_points1 (skey1 INTEGER, point1 ST_Geometry);
CREATE TABLE sample_points2 (skey2 INTEGER, point2 ST_Geometry);

INSERT INTO sample_points1 VALUES (101, 'POINT(10 20)');
INSERT INTO sample_points1 VALUES (102, 'POINT(10 50)');

INSERT INTO sample_points2 VALUES (1001, 'POINT(20 30)');
INSERT INTO sample_points2 VALUES (1002, 'POINT(20 40)');

SELECT skey1, skey2, point1.ST_SphericalDistance(point2)
FROM sample_points1, sample_points2
ORDER BY skey1;
```

ST_SpheroidalBufferMBR

Returns an MBR that represents the minimum and maximum latitude and longitude of all points within a given distance from the point. The earth is modeled as a spheroid.

Valid Data Types

ST_Point

Result Type

Returns an MBR that represents the minimum and maximum latitude and longitude of all points within a given distance from the point.

ST_SpheroidalBufferMBR Syntax

```
ST_SpheroidalBufferMBR ( distance [, semimajor, invflattening ] )
```

Syntax Elements

distance

A FLOAT value for the distance, in meters, from the point to calculate the MBR.

semimajor

An optional FLOAT value for the length, in meters, of the semimajor axis of the spheroid. If omitted, the method uses the WGS84 value of 6,378,137.0 meters.

invflattening

An optional FLOAT value for the inverse flattening of the spheroid. If omitted, the method uses the WGS84 value of 298.257223563.

Usage Notes

For the MBR calculation, the planet is modeled as a spheroid. If you do not pass in values for the *semimajor* and *invflattening* arguments, the computation uses the semimajor axis and the inverse flattening ratio from the World Geodetic System, WGS84. A value of 6,378,137.0 meters is used for the semimajor axis, and a value of 298.257223563 is used for the inverse flattening ratio.

ST_SphericalBufferMBR and ST_SpheroidalBufferMBR perform a similar function, but with different performance and accuracy. ST_SphericalBufferMBR models the earth as a simple sphere, so it performs better than ST_SpheroidalBufferMBR, but with less accuracy. ST_SpheroidalBufferMBR models the earth as a spheroid, so it returns a more accurate MBR, but it runs slower than ST_SphericalBufferMBR.

The returned MBR cannot cross the longitude value of +/-180 or the latitude value of +/-90. The MBR cannot cover more than 180 degrees of latitude or longitude on the sphere or spheroid.

Example: ST_SpheroidalBufferMBR

This example uses ST_SpheroidalBufferMBR to calculate an MBR that is 5000 meters from a point at longitude 100 and latitude 30:

```
SELECT
  NEW ST_Geometry('POINT(100 30)').ST_SpheroidalBufferMBR( 5000.0 );
```

ST_SpheroidalDistance

Returns the distance, in meters, between two spherical coordinates.

Valid Data Types

ST_Point

Result Type

Returns a FLOAT.

ST_SpheroidalDistance Syntax

```
ST_SpheroidalDistance ( apoint [, semimajor, invflattening ] )
```

Syntax Elements

apoint

An ST_Geometry for the other coordinate.

semimajor

An optional FLOAT value for the length in meters of the semimajor axis of the spheroid.

invflattening

An optional FLOAT value for the inverse flattening ratio of the spheroid.

Usage Notes

For the distance calculation, the planet is modeled as a spheroid.

If you do not pass in values for the *semimajor* and *invflattening* arguments, the computation uses the semimajor axis and the inverse flattening ratio from the World Geodetic System, WGS84. A value of 6,378,137.0 meters is used for the semimajor axis, and a value of 298.257223563 is used for the inverse flattening ratio.

Distance calculations between antipodal or nearly antipodal points using the ST_SpheroidalDistance method may fail to converge, generating an error. For these distance calculations, use instead the ST_SphericalDistance method.

Example: ST_SpheroidalDistance

The following two examples show how to use the ST_SpheroidalDistance method to get the spheroidal distance between Madison (-89.39, 43.09) and Chicago (-87.65, 41.90):

```
CREATE TABLE sample_points1 (skey1 INTEGER, point1 ST_Geometry);

INSERT INTO sample_points1 VALUES (101, 'POINT(-89.39 43.09)');

SELECT point1.ST_SpheroidalDistance(
    new ST_Geometry(
        'ST_POINT(-87.65 41.90)'), 6378137, 298.257223563)
FROM sample_points1;

SELECT point1.ST_SpheroidalDistance(
    new ST_Geometry('POINT(-87.65 41.90)') )
FROM sample_points1;
```

ST_X

Get and set the X coordinate of an ST_Point value.

Valid Data Types

ST_Point

Result Type

IF you pass in ...	THEN ST_X returns ...
no argument	a FLOAT value for the X coordinate.
the <i>xcoord</i> argument	an ST_Geometry value where the X coordinate is set to <i>xcoord</i> .
NULL	NULL

ST_X Syntax

```
ST_X ( [ xcoord ] )
```

Syntax Elements

xcoord

A FLOAT value for the new X coordinate of the ST_Point value.

Example: ST_X

```
SELECT skey1
FROM sample_points1
WHERE (point1.ST_X() < 1E-2);
```

ST_Y

Get and set the Y coordinate of an ST_Point value.

Valid Data Types

ST_Point

Result Type

IF you pass in ...	THEN ST_Y returns ...
no argument	a FLOAT value for the Y coordinate.

IF you pass in ...	THEN ST_Y returns ...
the <i>ycoord</i> argument	an ST_Geometry value where the Y coordinate is set to <i>ycoord</i> .
NULL	NULL

ST_Y Syntax

```
ST_Y ( [ ycoord ] )
```

Syntax Elements

ycoord

A FLOAT value for the new Y coordinate of the ST_Point value.

Example: ST_Y

```
SELECT skey1
FROM sample_points1
WHERE (point1.ST_Y() < 1E-2);
```

ST_Z

Get and set the z coordinate of an ST_Point value.

Valid Data Types

ST_Point

Result Type

IF you pass in ...	THEN ST_Z returns ...
no argument	a FLOAT value for the z coordinate.
the <i>zcoord</i> argument	an ST_Geometry value where the z coordinate is set to <i>zcoord</i> .
NULL	NULL

If the input geometry is a non-point, the system returns an error.

If the input geometry is a 2D point, and no *zcoord* argument is passed, the system returns an error.

ST_Z Syntax

```
ST_Z ( [ zcoord ] )
```

Syntax Elements

zcoord

A FLOAT value for the new z coordinate of the ST_Point value.

Usage Notes

This method can be used to convert a 2D point to a 3D point if a *zcoord* argument is passed.

Example: ST_Z

```
UPDATE sample_shapes  
SET shape=shape.ST_Z(35.2)  
WHERE skey = 1;
```

ST_LineString Methods

ST_3DIsClosed

Tests whether a 3D LineString or 3D MultiLineString is closed, taking into account the z coordinates in the calculation.

Valid Data Types

LineString and MultiLineString types that include z coordinates

Result Type

Returns an INTEGER value:

- 1 if the 3D LineString or 3D MultiLineString is closed.
- 0 if the 3D LineString or 3D MultiLineString is not closed or is empty.

If the input geometry is not a 3D LineString or 3D MultiLineString, ST_3DIsClosed returns an error.

ST_3DIsClosed Syntax

```
ST_3DIsClosed ()
```

Example: ST_3DIsClosed

```
CREATE TABLE sample_shapes(skey INTEGER, shape ST_Geometry);
INSERT sample_shapes(9901,
    'Linestring(0 0 0, 0 5 5, 5 5 5, 5 0 5, 0 0 0)');
INSERT sample_shapes(9902,
    'Linestring(0 0 0, 1 1 1, 2 2 2, 3 3 3, 4 4 4)');
SELECT Shape.ST_3DIsClosed()
FROM sample_shapes;
```

ST_3DLength

Returns the length of a 3D LineString or MultiLineString, taking into account the z coordinates in the calculation.

Valid Data Types

LineString and MultiLineString types that include z coordinates

Result Type

Returns a DOUBLE PRECISION value.

Returns 0 if the LineString or MultiLineString is empty.

ST_3DLength Syntax

```
ST_3DLength ()
```

Example: ST_3DLength

```
CREATE TABLE sample_shapes(skey INTEGER, shape ST_Geometry);
INSERT sample_shapes(9901,
    'Linestring(0 0 0, 0 5 5, 5 5 5, 5 0 5, 0 0 0)');
INSERT sample_shapes(9902,
    'Linestring(0 0 0, 1 1 1, 2 2 2, 3 3 3, 4 4 4)');

SELECT Shape.ST_3DLength()
FROM sample_shapes;
```

ST_EndPoint

Returns the end point of an ST_LineString or GeoSequence value.

Valid Data Types

ST_LineString and GeoSequence

Result Type

Returns an ST_Geometry value that represents the end point of the ST_LineString.

Returns NULL if the input geometry value is the empty set.

ST_EndPoint Syntax

```
ST_EndPoint ()
```

Usage Notes

This method returns a point having a z coordinate if the input geometry is 3-dimensional.

Example: ST_EndPoint

```
SELECT streetName, streetShape.ST_EndPoint()
FROM sample_streets
WHERE skey = 1044;
```

ST_IsClosed

Tests if an ST_Geometry type that represents an ST_LineString, GeoSequence, or ST_MultiLineString value is closed.

Valid Data Types

ST_LineString, GeoSequence, and ST_MultiLineString

Result Type

Returns an INTEGER value:

- 1, if the ST_LineString, GeoSequence, or ST_LineString components of an ST_MultiLineString are closed.
- 0, if the ST_LineString, GeoSequence, or ST_LineString components of an ST_MultiLineString are not closed, or if the input geometry is empty.

ST_IsClosed Syntax

```
ST_IsClosed ()
```

Usage Notes

An ST_LineString value is closed if the start point of the ST_LineString value is equal to the end point.

Example: ST_IsClosed

```
SELECT skey
FROM sample_shapes
WHERE shape.ST_IsClosed() = 0
ORDER BY skey;
```

ST_IsRing

Tests if an ST_Geometry type that represents an ST_LineString or a GeoSequence value is a ring.

Valid Data Types

ST_LineString or GeoSequence

Result Type

Returns an INTEGER value:

- 1 if the ST_LineString or GeoSequence value is simple (has no anomalous geometric points, such as self intersection tangency) and is closed (the start point of the ST_LineString value is equal to the end point)
- 0 in all other cases.

ST_IsRing Syntax

```
ST_IsRing ()
```

Usage Notes

ST_IsRing returns zero if the geometry value passed to it is the empty set.

Example: ST_IsRing

```
SELECT skey
FROM sample_shapes
WHERE shape.ST_IsRing() = 0
ORDER BY skey;
```

ST_Length

Returns the length measurement of an ST_Geometry type that represents an ST_LineString, GeoSequence, or ST_MultiLineString value.

Valid Data Types

ST_LineString, GeoSequence, and ST_MultiLineString

Result Type

Returns a FLOAT value.

ST_Length Syntax

```
ST_Length ()
```

Example: ST_Length

```
SELECT streetName
FROM sample_streets
WHERE streetShape.ST_Length() < 5E1
ORDER BY streetName;
```

ST_Line_Interpolate_Point

Returns a point interpolated along an ST_Geometry type that represents an ST_LineString or GeoSequence value, given a proportional distance along that line.

Valid Data Types

ST_LineString and GeoSequence

Result Type

Returns an ST_Point value.

ST_Line_Interpolate_Point Syntax

```
ST_Line_Interpolate_Point ( proportion )
```

Syntax Elements

proportion

A FLOAT value between 0 and 1 that represents the fraction of the total 2-dimensional length of the ST_LineString where the point must be located.

Example: ST_Line_Interpolate_Point

```
SELECT streetShape.ST_Line_Interpolate_Point(3E-1)
FROM sample_streets
WHERE skey = 1331;
```

ST_NumPoints

Returns the number of points in an ST_Geometry type that represents an ST_LineString or GeoSequence value.

Valid Data Types

ST_LineString and GeoSequence

Result Type

Returns an INTEGER value.

ST_NumPoints Syntax

```
ST_NumPoints ( )
```

Example: ST_NumPoints

```
SELECT skey
FROM sample_shapes
WHERE shape.ST_NumPoints() < 40
ORDER BY skey;
```

ST_PointN

Returns the specified point from an ST_Geometry type that represents an ST_LineString or GeoSequence value.

Valid Data Types

ST_LineString and GeoSequence

Result Type

Returns an ST_Geometry value.

ST_PointN Syntax

```
ST_PointN ( aposition )
```

Syntax Elements***aposition***

An INTEGER value for the requested point, where the position of the first point is 1.

Usage Notes

This method returns a point having a z coordinate if the input geometry is 3-dimensional.

Example: ST_PointN

```
SELECT shape.ST_PointN(2)
FROM sample_shapes
WHERE skey = 9902;
```

ST_StartPoint

Returns the start point of an ST_LineString or GeoSequence value.

Valid Data Types

ST_LineString and GeoSequence

Result Type

Returns an ST_Geometry value.

ST_StartPoint Syntax

```
ST_StartPoint ()
```

Usage Notes

This method returns a point having a z coordinate if the input geometry is 3-dimensional.

Example: ST_StartPoint

```
SELECT streetName, streetShape.ST_StartPoint()
FROM sample_streets
WHERE skey = 1044;
```

ST_Polygon Methods

A polygon consists of one exterior ring and zero or more interior rings. The interior rings must lie completely within the exterior ring. A MultiPolygon is a group of non-overlapping polygons.

For information on the PolygonSplit method, see [PolygonSplit](#).

ST_Area

Returns the area measurement of an ST_Polygon or ST_MultiPolygon. For ST_MultiPolygon, returns the sum of the area measurements of the component polygons.

Valid Data Types

ST_Polygon and ST_MultiPolygon

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is ignored in method calculations. Consequently, any z coordinates returned by this method should be ignored. Teradata recommends using the Make_2D method to strip out the z coordinates of the return value.

Result Type

Returns a FLOAT value.

ST_Area Syntax

```
ST_Area ()
```

Example: ST_Area

```
CREATE TABLE sample_shapes (skey INTEGER, shape ST_Geometry);

INSERT sample_shapes
  (1, 'Polygon((30 30, 30 60, 60 60, 60 30, 30 30))');

SELECT skey
FROM sample_shapes
WHERE (shape.ST_Area() < 100.5);
```

ST_ExteriorRing

Get and set the exterior ring of an ST_Geometry type that represents an ST_Polygon value.

Valid Data Types

ST_Polygon

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is ignored in method calculations. Consequently, any z coordinates returned by this method should be ignored. Teradata recommends using the Make_2D method to strip out the z coordinates of the return value.

Result Type

Returns an ST_Geometry value.

IF you ...	THEN ST_ExteriorRing returns ...
do not pass in the <i>acurve</i> argument	an ST_LineString of the exterior ring of the ST_Polygon.
pass in the <i>acurve</i> argument	the new ST_Polygon with the exterior ring set to the value of the <i>acurve</i> argument.

ST_ExteriorRing Syntax

```
ST_ExteriorRing ( [ acurve ] )
```

Syntax Elements

acurve

An ST_Geometry type that represents an ST_LineString for the new exterior ring of the polygon.

Usage Notes

ST_ExteriorRing returns an error if the geometry value passed to it is the empty set.

Example: ST_ExteriorRing

```
SELECT cityName, cityShape.ST_ExteriorRing()
FROM sample_cities
WHERE skey = 1044;
```

ST_InteriorRingN

Returns the specified interior ring of an ST_Geometry type that represents an ST_Polygon value.

Valid Data Types

ST_Polygon

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is ignored in method calculations. Consequently, any z coordinates returned by this method should be ignored. Teradata recommends using the Make_2D method to strip out the z coordinates of the return value.

Result Type

Returns an ST_LineString.

ST_InteriorRingN Syntax

```
ST_InteriorRingN ( [ aposition ] )
```

Syntax Elements

aposition

An INTEGER value for the requested interior ring, where the position of the first interior ring is 1.

Example: ST_InteriorRingN

```
SELECT shape.ST_InteriorRingN(2)
FROM sample_shapes
WHERE skey = 992;
```

ST_NumInteriorRing

Returns the number of interior rings of an ST_Geometry type that represents an ST_Polygon value.

Valid Data Types

ST_Polygon

Result Type

Returns an INTEGER value.

ST_NumInteriorRing Syntax

```
ST_NumInteriorRing ()
```

Example: ST_NumInteriorRing

```
SELECT skey
FROM sample_shapes
WHERE shape.ST_NumInteriorRing() < 4
ORDER BY skey;
```

ST_Perimeter

Returns the boundary length of an ST_Polygon, or the sum of the boundary lengths of the component polygons of an ST_MultiPolygon.

Valid Data Types

ST_Polygon or ST_MultiPolygon

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is ignored in method calculations.

Result Type

Returns a FLOAT value.

ST_Perimeter Syntax

```
ST_Perimeter ()
```

Example: ST_Perimeter

```
SELECT cityName
FROM sample_cities
WHERE cityShape.ST_Perimeter() < 5E1
ORDER BY cityName;
```

ST_PointOnSurface

Returns a point that is guaranteed to spatially intersect an ST_Polygon, or at least one of the component polygons of an ST_MultiPolygon.

Valid Data Types

ST_Polygon and ST_MultiPolygon

Note:

This method can be called on 3D geometries (those that include z coordinates). However, the z coordinate is dropped, and only x and y coordinates are returned.

Result Type

Returns an ST_Geometry value.

ST_PointOnSurface Syntax

```
ST_PointOnSurface ()
```

Example: ST_PointOnSurface

```
SELECT cityShape.ST_PointOnSurface()  
FROM sample_cities  
WHERE skey = 1;
```

ST_GeomCollection Methods

ST_GeometryN

Returns the geometry of one component member of a composite geometry type (ST_GeomCollection, ST_MultiPoint, ST_MultiLineString, or ST_MultiPolygon). The element to be returned is specified by position within the collection.

Valid Data Types

ST_GeomCollection, ST_MultiPoint, ST_MultiLineString, and ST_MultiPolygon

Result Type

Returns an ST_Geometry value.

ST_GeometryN Syntax

```
ST_GeometryN ( aposition )
```

Syntax Elements

aposition

An INTEGER value representing the position in the collection of the geometry to be returned. The position of the first element in the collection is number 1.

Example: ST_GeometryN

```
CREATE TABLE sample_shapes (skey INTEGER, shape ST_Geometry);

INSERT INTO sample_shapes VALUES
(1001, 'MultiPoint((10 20 30),(40 50 60),(70 80 90))');

INSERT INTO sample_shapes VALUES
(1002, 'MultiLineString((10 20 30,10 20 50,10 20 70,10 20 90),
(5 5 5,10 10 10,15 15 15,20 20 20,25 25 25,30 30 30))');

INSERT INTO sample_shapes VALUES
(1003, 'MultiPolygon(((0 0 0,0 20 20,20 20 20,20 0 20,0 0 0)),
((50 50 50,50 90 90,90 90 90,90 50 90,50 50 50)))');
```

```

INSERT INTO sample_shapes VALUES
(1004, 'Geometrycollection(Point(10 20 30),Linestring(1 2 3,4 5 6,7 8 9),
Polygon((0 0 0,0 40 40,40 40 40,40 0 40,0 0 0)))');

SELECT shape.ST_GeometryN(2)
FROM sample_shapes
WHERE skey = 1001;

```

ST_NumGeometries

Returns the number of distinct geometries that constitute a composite geometry type (ST_GeomCollection, ST_MultiPoint, ST_MultiLineString, or ST_MultiPolygon).

Valid Data Types

ST_GeomCollection, ST_MultiPoint, ST_MultiLineString, and ST_MultiPolygon

Result Type

Returns an INTEGER value.

ST_NumGeometries Syntax

```
ST_NumGeometries ()
```

Example: ST_NumGeometries

```

SELECT skey
FROM sample_shapes
WHERE shape.ST_NumGeometries() < 4
ORDER BY skey;

```

ST_Geomsequence Methods

Clip

Returns a GeoSequence type containing the subset of points and associated data that lie between the two timestamp input arguments.

Valid Data Types

GeoSequence

Result Type

Returns GeoSequence.

Clip Syntax

```
Clip ( startT, endT )
```

Syntax Elements

startT

A TIMESTAMP value for the beginning timestamp.

endT

A TIMESTAMP value for the ending timestamp.

Example: Clip

```
CREATE TABLE sample_shapes(skey INTEGER, shape ST_Geometry);

INSERT INTO sample_shapes ( 100,'GeoSequence( (10 10, 15 15, -2 0)
,(2007-03-14 01:35:00, 2007-03-14 01:35:05, 2007-03-14 01:35:08),
(1222, 1223, 1224),(2, 12.1, 3.14159, 2.78128, -10, -11, 100.1))' );

SELECT shape.Clip(TIMESTAMP '2008-07-31 10:36:02.123456',
                  TIMESTAMP '2008-07-31 11:36:02.123456')
FROM sample_shapes;
```

GetFinalT

Returns the TimeStamp of the last point of a GeoSequence.

Valid Data Types

GeoSequence

Result Type

Returns a TIMESTAMP value.

GetFinalT Syntax

```
GetFinalT ()
```

Example: GetFinalT

```
CREATE TABLE sample_shapes(skey INTEGER, shape ST_Geometry);

INSERT INTO sample_shapes ( 100,'GeoSequence( (10 10, 15 15, -2 0) ,
(2007-03-14 01:35:00, 2007-03-14 01:35:05, 2007-03-14 01:35:08),
(1222, 1223, 1224),(2, 12.1, 3.14159, 2.78128, -10, -11, 100.1))' );

SELECT skey
FROM sample_shapes
WHERE (EXTRACT(HOUR FROM shape.GetFinalT()) < 12)
ORDER BY skey;
```

GetInitT

Returns the TimeStamp of the first point of a GeoSequence.

Valid Data Types

GeoSequence

Result Type

Returns a TIMESTAMP value.

GetInitT Syntax

```
GetInitT ()
```

Example: GetInitT

```
CREATE TABLE sample_shapes(skey INTEGER, shape ST_Geometry);

INSERT INTO sample_shapes ( 100,'GeoSequence( (10 10, 15 15, -2 0) ,
(2007-03-14 01:35:00, 2007-03-14 01:35:05, 2007-03-14 01:35:08),
(1222, 1223, 1224),(2, 12.1, 3.14159, 2.78128, -10, -11, 100.1))' );

SELECT skey
FROM sample_shapes
WHERE (EXTRACT(HOUR FROM shape.GetInitT())) < 12)
ORDER BY skey;
```

GetUserFld

Returns the user field specified by *fldIndex* for the point specified by *index* for a GeoSequence type.

Valid Data Types

GeoSequence

Result Type

Returns a FLOAT value.

GetUserFld Syntax

```
GetUserFld ( fldIndex, index )
```

Syntax Elements

fldIndex

The index of the user field within the point, where the index of the first user field is 1.

The data type of *fldIndex* is INTEGER.

index

The index of the point within the GeoSequence type, where the index of the first point is 1.

The data type of *index* is INTEGER.

Example: GetUserFld

```
CREATE TABLE sample_shapes(skey INTEGER, shape ST_Geometry);

INSERT INTO sample_shapes ( 100,'GeoSequence( (10 10, 15 15, -2 0) ,
(2007-03-14 01:35:00, 2007-03-14 01:35:05, 2007-03-14 01:35:08),
(1222, 1223, 1224),(2, 12.1, 3.14159, 2.78128, -10, -11, 100.1))' );

SELECT skey
FROM sample_shapes
WHERE (shape.GetUserFld(1,2) < 1E2)
ORDER BY skey;
```

GetUserFldCount

Returns the number of user fields associated with each point of a GeoSequence.

Valid Data Types

GeoSequence

Result Type

Returns an INTEGER value.

GetUserFldCount Syntax

```
GetUserFldCount ()
```

Example: GetUserFldCount

```
CREATE TABLE sample_shapes(skey INTEGER, shape ST_Geometry);

INSERT INTO sample_shapes ( 100,'GeoSequence( (10 10, 15 15, -2 0) ,
(2007-03-14 01:35:00, 2007-03-14 01:35:05, 2007-03-14 01:35:08),
(1222, 1223, 1224),(2, 12.1, 3.14159, 2.78128, -10, -11, 100.1))' );

SELECT skey
FROM sample_shapes
WHERE (shape.GetUserFldCount() < 3)
ORDER BY skey;
```

HeadingN

Returns the heading for the specified point of a GeoSequence.

Valid Data Types

GeoSequence

Result Type

Returns a FLOAT value. The value is calculated as the angle (in degrees) between a vertical line and the line segment from the specified point to the next point clockwise from the North.

HeadingN Syntax

```
HeadingN ( index )
```

Syntax Elements

index

The index of the point to return the heading of, where the index of the first point in the GeoSequence is 1.

The data type of *index* is INTEGER.

Example: HeadingN

```
CREATE TABLE sample_shapes(skey INTEGER, shape ST_Geometry);

INSERT INTO sample_shapes ( 100, 'GeoSequence( (10 10, 15 15, -2 0) ,
(2007-03-14 01:35:00, 2007-03-14 01:35:05, 2007-03-14 01:35:08),
(1222, 1223, 1224),(2, 12.1, 3.14159, 2.78128, -10, -11, 100.1))' );

SELECT skey
FROM sample_shapes
WHERE ((shape.HeadingN(2)) < 3.8e2)
ORDER BY skey;
```

LinkID

Get and set the link ID of a specified point in a GeoSequence.

Valid Data Types

GeoSequence

Result Type

IF you pass in ...	THEN LinkID returns ...
<i>index</i>	a DECIMAL(18,0) value for the link ID of the specified point.
<i>index</i> and <i>linkID</i>	an ST_Geometry value that represents a GeoSequence where the link ID for the specified point is set to <i>linkID</i> .

LinkID Syntax

```
LinkID ( index [, linkID ] )
```

Syntax Elements***index***

The index of the point in the GeoSequence, where the index of the first point is 1.

The data type of *index* is INTEGER.

linkID

The new link ID of specified point in the GeoSequence.

The data type of *linkID* is DECIMAL(18,0).

Example: LinkID

```
CREATE TABLE sample_shapes(skey INTEGER, shape ST_Geometry);

INSERT INTO sample_shapes ( 100, 'GeoSequence( (10 10, 15 15, -2 0) ,
(2007-03-14 01:35:00, 2007-03-14 01:35:05, 2007-03-14 01:35:08),
(1222, 1223, 1224),(2, 12.1, 3.14159, 2.78128, -10, -11, 100.1))' );

SELECT skey
FROM sample_shapes
WHERE ((shape.LinkID(1)) < 9000.0)
ORDER BY skey;
```

SpeedN

Returns the approximate speed at a specified point (SpeedN(*index* INTEGER)) or between two points (SpeedN(*iBegin* INTEGER, *iEnd* INTEGER)) for a GeoSequence type.

Valid Data Types

GeoSequence

Result Type

Returns a FLOAT value.

IF you pass in ...	THEN ...
<i>iBegin</i> and <i>iEnd</i>	the speed is calculated as the distance between the two points (along the LineString) divided by the time between them.
<i>index</i>	if the point is: <ul style="list-style-type: none"> • The first point, the distance between the first and second points is divided by the time between them. • The last point, the distance between the second to the last and last points is divided by the time between them. • Any other point, the distance between the previous point and the next point (along the LineString) is divided by the time difference between the previous point and the next point.

The units used for distance are the same units that are used by the coordinate system for the GeoSequence value, and the time units are in hours.

SpeedN Syntax

```
SpeedN ( { index | iBegin, iEnd } )
```

Syntax Elements

index

An INTEGER for the index of the point within the GeoSequence type, where the index of the first point is 1.

iBegin

An INTEGER for the index of the starting point for which to calculate the speed, where the index of the first point in a GeoSequence type is 1.

iEnd

An INTEGER for the index of the ending point for which to calculate the speed, where the index of the first point in a GeoSequence is 1.

Example: SpeedN

```
CREATE TABLE sample_shapes(skey INTEGER, shape ST_Geometry);

INSERT INTO sample_shapes ( 100, 'GeoSequence( (10 10, 15 15, -2 0) ,
(2007-03-14 01:35:00, 2007-03-14 01:35:05, 2007-03-14 01:35:08),
(1222, 1223, 1224),(2, 12.1, 3.14159, 2.78128, -10, -11, 100.1))' );

SELECT skey
FROM sample_shapes
WHERE ((shape.SpeedN(2,3)) < 3000e1)
ORDER BY skey;
```

Filtering Functions and Methods

This section describes methods that perform spatial filtering operations. Vantage accomplishes 2D spatial filtering using minimum bounding rectangles (MBRs). Three-dimensional filtering employs volumes represented as minimum bounding boxes (MBBs). For the `MBB_Filter` method, two 3D geometries are converted automatically to MBBs for the filtering calculation. For the `Intersects_MBB` and `Within_MBB` methods, a 3D geometry is compared to an MBB.

Intersects_MBB

Tests whether a 3D geometry spatially intersects a specified MBB.

Valid Data Types

3D Point, 3D MultiPoint, 3D LineString, 3D MultiLineString

Result Type

Returns an INTEGER value:

- 1, if the input 3D geometry intersects the MBB argument.
- 0, if the input 3D geometry does not intersect the MBB argument.

Returns an error if the input geometry is not 3D (does not have a z coordinate).

Intersects_MBB Syntax

```
Intersects_MBB ( aMBB )
```

Syntax Elements

aMBB

A minimum bounding box.

Example: Intersects_MBB

```
CREATE TABLE sample_shapes(skey INTEGER, shape ST_Geometry);
INSERT sample_shapes VALUES(1001, 'Point(10 20 30)');
INSERT sample_shapes VALUES(1002,
'LineString(0 0 0, 1 1 1, 2 2 2, 3 3 3, 4 4 4)');
INSERT sample_shapes VALUES(1003,
'MultiPoint((10 20 30), (40 50 60), (70 80 90))');
```

```

INSERT sample_shapes VALUES(1004,
  'MultiLinestring((0 0 0, 1 1 1, 2 2 2, 3 3 3, 4 4 4),
(30 30 30, 30 30 50, 30 30 70, 30 30 90))');

SELECT shape.Intersects_MBB(new MBB(0,0,0,20,20,20))
FROM sample_shapes;

```

MBB_Filter

Tests whether the MBBs of two 3D geometries spatially intersect.

Valid Data Types

All 3D ST_Geometry types.

Result Type

Returns an INTEGER value:

- 1, if the MBB of the input 3D geometry intersects the MBB of the geometry passed to the method.
- 0, if the MBB of the input 3D geometry does not intersect the MBB of the geometry passed to the method.

Returns an error if either geometry is not 3D (does not have a z coordinate).

MBB_Filter Syntax

```
MBB_Filter ( othergeom )
```

Syntax Elements

othergeom

A 3D ST_Geometry.

Example: MBB_Filter

```

SELECT shape.MBB_Filter(
  'Polygon((0 0 0, 0 20 20, 20 20 20, 20 0 20, 0 0 0))')
FROM sample_shapes;

```

MBR_Filter

Tests whether the MBRs of two 2D geometries spatially intersect.

Valid Data Types

All 2D ST_Geometry types.

Although MBR_Filter will accept 3D geometries, the z coordinates are ignored in the calculations.

Result Type

Returns an INTEGER value:

- 1, if the MBR of the input geometry intersects the MBR of the geometry passed to the method.
- 0, if the MBR of the input geometry does not intersect the MBR of the geometry passed to the method.

MBR_Filter Syntax

```
MBR_Filter ( othergeom )
```

Syntax Elements

othergeom

A 2D ST_Geometry.

Example: MBR_Filter

```
INSERT INTO sample_shapes VALUES (1001, 'POINT(10 20)');
INSERT INTO sample_shapes VALUES (1002, CAST(
  'LINESTRING(1 1, 2 2, 3 3)' AS ST_Geometry));

SELECT shape.MBR_Filter('MultiPoint((10 20), (30 40), (50 60))')
FROM sample_shapes;
```

Within_MBB

Tests whether a 3D geometry is spatially within the bounds of a specified MBB.

Valid Data Types

All 3D ST_Geometry types.

Result Type

Returns an INTEGER value:

- 1, if the input 3D geometry is spatially within MBB argument.
- 0, if the input 3D geometry is not spatially within the MBB argument.

Returns an error if the input geometry is not 3D (does not have a z coordinate).

Within_MBB Syntax

```
Within_MBB ( aMBB )
```

Syntax Elements

aMBB

A minimum bounding box.

Example: Within_MBB

```
CREATE TABLE sample_shapes(skey INTEGER, shape ST_Geometry);
INSERT sample_shapes VALUES(1001, 'Point(10 20 30)');
INSERT sample_shapes VALUES(1002,
  'Linestring(0 0 0, 1 1 1, 2 2 2, 3 3 3, 4 4 4)');
INSERT sample_shapes VALUES(1003,
  'MultiPoint((10 20 30), (40 50 60), (70 80 90))');
INSERT sample_shapes VALUES(1004,
  'MultiLinestring((0 0 0, 1 1 1, 2 2 2, 3 3 3, 4 4 4),
  (30 30 30, 30 30 50, 30 30 70, 30 30 90))');

SELECT shape.Within_MBB(new MBB(0,0,0,20,20,20))
FROM sample_shapes;
```

Embedded Services System Functions

For best performance, when a geospatial system function exists that performs the same functionality as a geospatial UDF, use the system function.

If your SQL statement specifies the name of a geospatial system function that is also the name of a geospatial UDF, Vantage calls the system function. To call the UDF, qualify the name of the UDF with the SYSSPATIAL database name.

AggGeomIntersection [Deprecated]

Note:

This deprecated method was replaced by the [AggGeom](#) table operator, which provides better performance and no size restriction.

Aggregate function that returns the intersection of all objects in the aggregation group.

Result Type

Returns an ST_Geometry value.

Returns "GEOMETRYCOLLECTION EMPTY" if there is no intersection between the objects resulting from any step of the aggregation.

NULL values are ignored in the aggregation. If all values in the aggregation are null, the return value is NULL.

Note:

Because the order in which the rows are processed and per-AMP results are returned for aggregation can vary from run to run, the ordering of the values in the results can vary between different instances of identical queries. Although the order of the individual values in the returned results can vary, the overall results from the query will be the same when an identical query is run on the same data.

AggGeomIntersection Syntax

```
[TD_SYSFNLIB.] AggGeomIntersection ( ageometry )
```

Syntax Elements

ageometry

A constant or column expression for which the intersection is to be computed.

The data type of *ageometry* is ST_Geometry.

Usage Notes

Restrictions

The input geometry cannot be larger than 64000 bytes.

If the WKB representation of the resulting geometry becomes larger than 64000 bytes during the aggregation, the system reports an error.

This function does not support geometry collections.

Example: AggGeomIntersection

These examples use the AggGeomIntersection function to get the spatial intersection of the shape column values in rows of the sample_shapes table:

```
CREATE TABLE sample_shapes(skey INTEGER, shape ST_Geometry);

INSERT INTO sample_shapes(1001,'MultiPoint((10 20 30),(40 50 60))');
INSERT INTO sample_shapes(1002,'Point(10 20 30)');
INSERT INTO sample_shapes(1003,
  'Polygon((100 100 100, 100 500 500, 500 500 500, 500 100 500, 100 100 100))');
INSERT INTO sample_shapes(1004, 'Linestring (100 500 500, 500 100 500)');
INSERT INTO sample_shapes(1005,
  'Polygon((500 500 500, 500 1000 1000, 1000 1000 1000, 1000 500 1000, 500
  500 500))');
INSERT INTO sample_shapes(1006, 'Polygon((0 0 0, 0 50 50, 50 50 50, 50 0 50, 0
  0 0))');
INSERT INTO sample_shapes(1007,
  'Polygon((50 50 50, 50 100 100, 100 100 100, 100 50 100, 50 50 50))');

select AggGeomIntersection(shape) from sample_shapes;

AggGeomIntersection(shape)
-----
GEOMETRYCOLLECTION EMPTY

select AggGeomIntersection(shape) from sample_shapes where skey in (1001, 1002);

AggGeomIntersection(shape)
-----
POINT (10 20 30)

select AggGeomIntersection(shape) from sample_shapes where skey in (1003, 1004);

AggGeomIntersection(shape)
-----
LINESTRING (100 500 500,500 100 500)

select AggGeomIntersection(shape) from sample_shapes where skey in (1003,1005);

AggGeomIntersection(shape)
-----
```

```
POINT (500 500 500)

select AggGeomIntersection(shape) from sample_shapes where skey in (1006,1007);

AggGeomIntersection(shape)
-----
POINT (50 50 50)
```

AggGeomUnion [Deprecated]

Note:

This deprecated method was replaced by the [AggGeom](#) table operator, which provides better performance and no size restriction.

Aggregate function that returns the union of all spatial objects in the aggregation group.

Result Type

Returns an ST_Geometry value.

NULL arguments are not included in the aggregation. If all values in the aggregation are null, the return value is NULL.

Note:

Because the order in which the rows are processed and per-AMP results are returned for aggregation can vary from run to run, the ordering of the values in the results can vary between different instances of identical queries. Although the order of the individual values in the returned results can vary, the overall results from the query will be the same when an identical query is run on the same data.

AggGeomUnion Syntax

```
[TD_SYSFNLIB.] AggGeomUnion ( ageometry )
```

Syntax Elements

ageometry

A constant or column expression for which the union is to be computed.

The data type of *ageometry* is ST_Geometry.

Usage Notes

During the final phase of aggregation, each AMP sends a portion of the aggregated results to be combined into a final aggregation. The order in which this happens can vary from run to run, so actual results can vary per run.

Restrictions

The input geometry cannot be larger than 64000 bytes.

If the WKB representation of the resulting geometry becomes larger than 64000 bytes during the aggregation, the system reports an error.

This function does not support geometry collections.

Example: AggGeomUnion

This example uses the AggGeomUnion function to get the spatial union of the shape column in rows of the sample_shapes table:

```
CREATE TABLE sample_shapes(skey INTEGER, shape ST_Geometry);

INSERT INTO sample_shapes(1001,'MultiPoint((10 20 30),(40 50 60))');
INSERT INTO sample_shapes(1002,'Point(10 20 30)');
INSERT INTO sample_shapes(1003,
'Polygon((100 100 100, 100 500 500, 500 500 500, 500 100 500, 100 100 100))');
INSERT INTO sample_shapes(1004, 'Linestring (100 500 500, 500 100 500)');
INSERT INTO sample_shapes(1005,
'Polygon((500 500 500, 500 1000 1000, 1000 1000 1000, 1000 500 1000, 500
500 500))');
INSERT INTO sample_shapes(1006, 'Polygon((0 0 0, 0 50 50, 50 50 50, 50 0 50, 0
0 0))');
INSERT INTO sample_shapes(1007,
'Polygon((50 50 50, 50 100 100, 100 100 100, 100 50 100, 50 50 50))');

select AggGeomUnion(shape) from sample_shapes where skey in (1001, 1002);

AggGeomUnion(shape)
-----
MULTIPOINT (10 20 30,40 50 60)

select AggGeomUnion(shape) from sample_shapes where skey in (1003, 1004);

AggGeomUnion(shape)
-----
POLYGON ((100 100 100,100 500 500,500 500 500,500 100 500,100 100 100))

select AggGeomUnion(shape) from sample_shapes where skey in (1003,1005,1006,1007);

AggGeomUnion(shape)
-----
MULTIPOLYGON (((50 50 50,50 100 100,100 100 100,100 50 100,50 50 50)),
((500 500 500,500 1000 1000,1000 1000 1000,1000 500 1000,500 500 500)),
((0 0 0,0 50 50,50 50 50,50 0 50,0 0 0)),
((100 100 100,100 500 500,500 500 500,500 100 500,100 100 100)))
```

DataSize

Returns the data length in bytes of any of the following Teradata variable maximum length complex data types:

- DATASET
- JSON
- ST_Geometry
- XML

Returns a BIGINT that is the size in bytes of the data object passed in to the function.

DataSize Syntax

```
[TD_SYSFNLIB.] DataSize (var_max_length_cdt)
```

Syntax Elements

var_max_length_cdt

A DATASET, JSON, ST_Geometry, or XML data type object.

DataSize Examples

The following examples use JSON data types.

```
SELECT TD_SYSFNLIB.DataSize (NEW JSON ('{"name" : "Mitzy", "age" : 3}'));
```

```
datasize( NEW JSON('{"name" : "Mitzy", "age" : 3}', LATIN))
```

```
-----  
29
```

```
CREATE TABLE JSON_table (id INTEGER, jsn JSON INLINE LENGTH 1000);
```

```
INSERT INTO JSON_table VALUES (100, '{"name" : "Mitzy", "age" : 3}');
```

```
INSERT INTO JSON_table VALUES (200, '{"name" : "Rover", "age" : 5}');
```

```
INSERT INTO JSON_table VALUES (300, '{"name" : "Princess", "age" : 4.5}');
```

```
SELECT * FROM JSON_table ORDER BY id;
```

```
id jsn
```

```
-----  
100 {"name" : "Mitzy", "age" : 3}
```

```
200 {"name" : "Rover", "age" : 5}
```

```
300 {"name" : "Princess", "age" : 4.5}
```

```
SELECT id, TD_SYSFNLIB.DataSize (jsn) FROM JSON_table ORDER BY id;
```

id	datasize(jsn)
-----	-----
100	29
200	29
300	34

FROM_MGRS

Converts a coordinate in the Military Grid Reference System (MGRS) to an ST_GEOMETRY point object.

Note:

This function was developed using GEOTRANS, a product of the National Geospatial-Intelligence Agency (NGA) and U.S. Army Engineering Research and Development Center.

Result Type

Returns an ST_POINT type object in the desired spatial reference system.

FROM_MGRS Syntax

```
[TD_SYSFNLIB.] FROM_MGRS ( MGRS_coord, toWktSRS, SRID )
```

Syntax Elements

MGRS_coord

A string describing a coordinate in MGRS

toWktSRS

The well known text (WKT) representation of the target spatial reference system. The supported coordinate systems are UTM, UPS, and Geodetic.

SRID

The ID used for the spatial reference system specified by *toWktSRS*. It is used to set the ID inside the resulting ST_Geometry type.

Argument Types and Rules

The data type of an argument that you pass to a geospatial system function does not need to exactly match the corresponding declared argument type as described in the function description. If the argument type is a compatible type, as listed in the following table, the system tries to implicitly cast the argument.

Declared Argument Type	Compatible Argument Types
DECIMAL	BYTEINT, SMALLINT, INTEGER, or BIGINT
FLOAT	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL, or NUMERIC
INTEGER	BYTEINT or SMALLINT
VARCHAR	CHAR

To pass an argument where the data type is not an exact match and is not a compatible type according to the preceding table, you must explicitly convert the argument.

Usage Notes

MGRS coordinates employ one of two lettering schemes to denote row identifiers in the grid system: MGRS-Old or MGRS-New. During conversions to and from MGRS, the MGRS lettering scheme in the returned coordinates (for the `TO_MGRS` function) or expected as input (to the `FROM_MGRS` function) depends on the spatial reference system from or to which the coordinates are being converted. If the spatial reference system is based on the Bessel, Clarke 1866, or Clarke 1880 reference ellipsoid, the MGRS coordinates use the MGRS-Old lettering scheme. Conversion to or from all other spatial reference systems use the MGRS-New scheme.

Example: FROM_MGRS

This example converts coordinates in the MGRS system to Geodetic coordinates in the “WGS 84” spatial reference system.

```
SELECT FROM_MGRS(
  '36TTQ6355387329',
  (SELECT SRTEXT FROM SYSSPATIAL.SPATIAL_REF_SYS WHERE SRID = 1619),
  1619);
```

Returns:

```
FROM_MGRS('36TTQ6355387329',<Scalar Subquery>,1619)
-----
POINT (29.999987897221764 44.999995138532547)
```

GeomFromGeoJSON

Converts a JSON document that conforms to the GeoJSON standards into an `ST_Geometry` object. For more information about this system function, see *Teradata Vantage™ - JSON Data Type*, B035-1150.

GeoJSONFromGeom

Converts an ST_Geometry object into a JSON document that conforms to the GeoJSON standards. For more information on this system function, see *Teradata Vantage™ - JSON Data Type*, B035-1150.

GeoSequenceFromRows

Table function that takes a table, where the rows have data for each point, and returns a GeoSequence object.

GeoSequenceFromRows Syntax

```
[TD_SYSFNLIB.] GeoSequenceFromRows (
  in_key,
  pcount,
  point_index,
  x,
  y,
  ts,
  Link_id,
  user_fld_1,
  user_fld_2,
  user_fld_3,
  user_fld_4,
  user_fld_5,
  user_fld_6,
  user_fld_7,
  user_fld_8,
  user_fld_9,
  user_fld_10
)
```

Syntax Elements

in_key

A key that identifies this GeoSequence type. All points in the same GeoSequence object have the same *in_key* value.

The data type of *in_key* is DECIMAL(18,0).

pcount

The number of points in the GeoSequence.

The function produces a result row when it sees *pcount* rows for the same *in_key* value.

The data type of *pcount* is INTEGER.

point_index

The index of this point in the sequence, where the index of the first point is 1.

The data type of *point_index* is INTEGER.

x,y

The X and Y coordinates of the point.

The data type of *x* and *y* is FLOAT.

ts

A timestamp for the point.

The data type of *ts* is TIMESTAMP.

Link_id

The Link ID of the point.

The data type of *Link_id* is DECIMAL(18,0).

user_fld_1 to user_fld_10

User fields of data for this point.

The data type of *user_fld_1* to *user_fld_10* is FLOAT.

A NULL value indicates that the user field is not used.

Argument Types and Rules

The data type of an argument that you pass to a geospatial system function does not need to exactly match the corresponding declared argument type as described in the function description. If the argument type is a compatible type, as listed in the following table, the system tries to implicitly cast the argument.

Declared Argument Type	Compatible Argument Types
DECIMAL	BYTEINT, SMALLINT, INTEGER, or BIGINT
FLOAT	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL, or NUMERIC
INTEGER	BYTEINT or SMALLINT
VARCHAR	CHAR

To pass an argument where the data type is not an exact match and is not a compatible type according to the preceding table, you must explicitly convert the argument.

Result Rows

Returns the following columns:

Column Name	Data Type	Description
out_key	DECIMAL(18,0)	A key that is passed back and can be associated with a trip ID.
geom	ST_Geometry	An ST_Geometry that represents a GeoSequence object.

Usage Notes

Input rows must be partitioned by *in_key* and sorted by the *ts* timestamp value.

Example: GeoSequenceFromRows

```
CREATE TABLE TripRowInput(
    trip_id INTEGER NOT NULL,
    pi_AMP INTEGER NOT NULL,
    flag INTEGER NOT NULL,
    pcount INTEGER NOT NULL,
    ppi# INTEGER NOT NULL,
    seq INTEGER,
    x FLOAT,
    y FLOAT,
    t TIMESTAMP,
    link_id NUMERIC(18),
    speed FLOAT,
    accel FLOAT,
    heading FLOAT)
PRIMARY INDEX(trip_id)
PARTITION BY ppi#;

INSERT INTO TripRowInput(100, 200, 300, 400, 500, 600, 1.11, 2.11,
'2014-01-07 13:29:49.200000', 12112121, 150.2, 110.1, 120);

CREATE TABLE TempFromRowsInput(
    trip_id INTEGER NOT NULL,
    pi_AMP INTEGER NOT NULL,
    flag INTEGER NOT NULL,
    pcount INTEGER NOT NULL,
```

```

    ppi# INTEGER NOT NULL,
    seq INTEGER,
    x FLOAT,
    y FLOAT,
    t TIMESTAMP,
    link_id NUMERIC(18),
    speed FLOAT,
    accel FLOAT,
    heading FLOAT)
PRIMARY INDEX(trip_id)
PARTITION BY ppi#;
INSERT INTO TempFromRowsInput
SELECT trip_id, pi_AMP, flag,
    COUNT(*) OVER (PARTITION BY pi_AMP, trip_id ORDER BY trip_id)
        AS pcount,
    SUM (flag) OVER (PARTITION BY pi_AMP ORDER BY trip_id, flag
        DESC ROWS UNBOUNDED PRECEDING) AS ppi#,
    seq, x, y, t, link_id, speed, accel, heading
FROM (
    SELECT trip_id,
        HASHAMP(HASHBUCKET(HASHROW(trip_id))) AS pi_AMP,
        CASE WHEN trip_id = MAX(trip_id) OVER
            (PARTITION BY pi_AMP ORDER BY trip_id
                ROWS BETWEEN 1 PRECEDING AND 1 PRECEDING)
            THEN 0
            ELSE 1
        END AS flag,
        seq, x, y, t, link_id, speed, accel, heading
    FROM TripRowInput) d
QUALIFY ppi# > 0 and ppi# <= 64000;

CREATE TABLE trips AS (
    SELECT T.trip_id, R.geom
    FROM (
        SELECT trip_id, pcount, seq, x, y, t, link_id, speed, accel,
            heading, flag
        FROM TempFromRowsInput) T,
        TABLE (GeoSequenceFromRows(T.trip_id, T.pcount, T.seq,
            T.x, T.y, T.t, T.link_id, T.speed, T.accel, T.heading,
            NULL, NULL, NULL, NULL, NULL, NULL, NULL)) R
    WHERE R.out_key = T.trip_id
    QUALIFY( COUNT(*) OVER (PARTITION BY T.trip_id

```

```
ORDER BY T.trip_id ROWS UNBOUNDED PRECEDING) = 1 )
) WITH DATA;
```

GeoSequenceToRows

Table function that takes a GeoSequence input argument, represented as an ST_Geometry type, and returns table rows where each row is a point in the GeoSequence object. Output rows are fixed in size and allow up to ten user fields.

GeoSequenceToRows Syntax

```
[TD_SYSNLIB.] GeoSequenceToRows ( id, ageometry )
```

Syntax Elements

id

An ID that can be used to associate the rows in the table with a single GeoSequence object.

The data type of *id* is DECIMAL(18,0).

ageometry

An ST_Geometry value that represents the GeoSequence object.

Argument Types and Rules

The data type of an argument that you pass to a geospatial system function does not need to exactly match the corresponding declared argument type as described in the function description. If the argument type is a compatible type, as listed in the following table, the system tries to implicitly cast the argument.

Declared Argument Type	Compatible Argument Types
DECIMAL	BYTEINT, SMALLINT, INTEGER, or BIGINT
FLOAT	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL, or NUMERIC
INTEGER	BYTEINT or SMALLINT
VARCHAR	CHAR

To pass an argument where the data type is not an exact match and is not a compatible type according to the preceding table, you must explicitly convert the argument.

Result Rows

Returns the following columns:

Column Name	Data Type	Description
out_key	DECIMAL(18,0)	A key that is passed back. This can be used to associate all points in the table with this key to a single GeoSequence type.
point_index	INTEGER	The index of this point in the sequence, where the index of the first point is 1.
x	FLOAT	X coordinate of the point.
y	FLOAT	Y coordinate of the point.
ts	TIMESTAMP	Timestamp of the point.
Link_id	DECIMAL(18,0)	Link ID of the point.
UserFld1	FLOAT	User fields of data for this point.
UserFld2		
UserFld3		
UserFld4		
UserFld5		
UserFld6		
UserFld7		
UserFld8		
UserFld9		
UserFld10		

Example: GeoSequenceToRows

```
CREATE TABLE sample_shapes(skey INTEGER, shape ST_Geometry);

INSERT INTO sample_shapes
( 100,
  'GeoSequence( (10 10, 15 15, -2 0),
                (2007-03-14 01:35:00, 2007-03-14 01:35:05, 2007-03-14 01:35:08),
                (1222, 1223, 1224),
                (2, 12.1, 3.14159, 2.78128, -10, -11, 100.1))' );
```

```
CREATE TABLE sample_trips(pkey FLOAT, point_index INTEGER, x FLOAT, y FLOAT,
ts TIMESTAMP,
Link_id DECIMAL(18,0), UserFld1 FLOAT, UserFld2 FLOAT, UserFld3 FLOAT);

INSERT INTO sample_trips
SELECT out_key, point_index, x, y, ts, Link_id, UserFld1, UserFld2, UserFld3
FROM TABLE (GeoSequenceToRows(9601., sample_shapes.shape)) AS ts;
```

PolygonSplit

Table operator that accepts a table containing ST_Geometry Polygon types and recursively processes qualifying polygons into sets of smaller sub-polygons, each having fewer vertices than the original polygon. Taken together, the sub-polygons closely approximate the shape and area of the original polygon.

If the input polygon exceeds a specified maximum number of vertices, PolygonSplit divides the polygon into four sub-polygons, one occupying each quadrant of the input polygon. The sub-polygons thus formed are similarly, recursively split until each resulting polygon has fewer than the specified maximum number of vertices.

The input table must include an ST_Geometry column to hold a polygon and an identifier column to hold a unique identifier for each polygon. The sub-polygons that are created all share the identifier of the original polygon that was split.

PolygonSplit Syntax

```
[TD_SYSFNLIB.] PolygonSplit (
  ON { [ database_name.. ] table_name |

      ( SELECT polygon_ID, geom [, max_vertices ]
        FROM [ database_name.. ] table_name
      )
  }
) [AS] correlation_name [ ( column_list )]
```

Syntax Elements

polygon_ID

A numeric or character value that uniquely identifies the polygon or multipolygon to be split. This same value is returned for each of the sub-polygons that is returned as a result of the splitting, and can be used to associate the sub-polygons with the original polygon

polygon_ID can be any valid Teradata numeric or character data type except CLOB.

If *polygon_ID* is NULL for a given polygon, the results will show corresponding NULLs for the IDs of the unchanged polygon or for the sub-polygons that are generated.

geom

The polygon or multipolygon geometry to be split.

PolygonSplit accepts all types of ST_Geometry, but only 2D polygons and 2D multipolygons are split. Other geometries are returned unchanged.

If *geom* is NULL, PolygonSplit returns NULL for that geometry.

max_vertices

The maximum number of vertices a returned sub-polygon can include. If the original polygon or multipolygon has fewer than this number of vertices, it is returned unchanged.

max_vertices must be greater than or equal to 10. The default is 300.

If *max_vertices* is NULL, PolygonSplit defaults to 300.

[database_name.]***table_name***

The name and optional database location of the table that contains the polygon and multipolygon geometries to be split.

correlation_name

A name to be assigned to the table returned by the table operator.

column_list

Optional list of custom names for the columns returned by the table operator.

These names override the column names that would be otherwise generated by the operator. If you specify any column names, you must provide names for all columns returned. If more than one column is returned by the operator, separate the names with commas.

Result Rows

Returns the a table having the following columns:

Column Name	Data Type	Description
out_polygon_ID	Matches type of <i>polygon_ID</i> input parameter	All sub-polygons that result from the splitting of a polygon or multipolygon will share the ID of the input polygon or multipolygon. Geometries that are unchanged by PolygonSplit will retain their original ID in the results table.
sub_polygon_ID	INTEGER	Uniquely identifies each subpolygon that came from a particular input polygon or multipolygon. The subpolygons are numbered sequentially as they are created. Numbering starts at 0.

Column Name	Data Type	Description
split_geom	ST_Geometry	A sub-polygon that results from splitting a polygon or multipolygon. For input polygons and multipolygons that have fewer than the maximum specified number of vertices, and for all other geometries, this column contains the original, unchanged input geometry.

Usage Notes

The splitting process is subject to precision and rounding errors that are inherent with floating-point arithmetic. Consequently, there can be small discrepancies between the area of the original polygon and the sum of the areas of the resulting sub-polygons.

Example: PolygonSplit

```
CREATE TABLE feature_tbl(poly_id INTEGER, geom ST_Geometry);

INSERT INTO feature_tbl VALUES(100, new ST_Geometry('POLYGON ((50
50,50 60,50 70,50 80,50 90,50 100,50 110,50 120,50 130,50 140,50
150,50 160,50 170,50 180,50 190,50 200,50 210,50 220,50 230,50 240,50
250,50 260,50 270,50 280,50 290,50 300,50 310,50 320,50 330,50 340,50
350,50 360,50 370,50 380,50 390,50 400,50 410,50 420,50 430,50 440,50
450,50 460,50 470,50 480,50 490,50 500,50 510,50 520,50 530,50 540,50
550,50 560,50 570,50 580,50 590,50 600,50 610,50 620,50 630,50 640,50
650,50 660,60 670,50 680,50 690,50 700,50 710,50 720,50 730,50 740,50
750,50 760,50 770,50 780,50 790,50 800,50 810,50 820,50 830,50 840,50
850,50 860,50 870,50 880,50 890,50 900,50 910,50 920,50 930,50 940,50
950,50 960,50 970,50 980,50 990,50 1000,50 1010,50 1020,50 1030,50
1040,50 1050,50 1060,50 1070,50 1080,50 1090,50 1100,50 1110,50
1120,50 1130,50 1140,50 1150,50 1160,50 1170,50 1180,50 1190,50
1200,60 1200,70 1200,80 1200,90 1200,100 1200,110 1200,120 1200,130
1200,140 1200,150 1200,160 1200,170 1200,180 1200,190 1200,200
1200,210 1200,220 1200,230 1200,240 1200,250 1200,260 1200,270
1200,280 1200,290 1200,300 1200,310 1200,320 1200,330 1200,340
1200,350 1200,360 1200,370 1200,380 1200,390 1200,400 1200,410
1200,420 1200,430 1200,440 1200,450 1200,460 1200,470 1200,480
1200,490 1200,500 1200,510 1200,520 1200,530 1200,540 1200,550
1200,560 1200,570 1200,580 1200,590 1200,600 1200,610 1200,620
1200,630 1200,640 1200,650 1200,660 1200,670 1200,680 1200,690
1200,700 1200,710 1200,720 1200,730 1200,740 1200,750 1200,760
1200,770 1200,780 1200,790 1200,800 1200,810 1200,820 1200,830
1200,840 1200,850 1200,860 1200,870 1200,880 1200,890 1200,900
1200,910 1200,920 1200,930 1200,940 1200,950 1200,960 1200,970
```

```

1200,980 1200,990 1200,1000 1200,1010 1200,1020 1200,1030 1200,1040
1200,1050 1200,1060 1200,1070 1200,1080 1200,1090 1200,1100 1200,1110
1200,1120 1200,1130 1200,1140 1200,1150 1200,1160 1200,1170 1200,1180
1200,1190 1200,1200 1200,1200 1190,1200 1180,1200 1170,1200 1160,1200
1150,1200 1140,1200 1130,1200 1120,1200 1110,1200 1100,1200 1090,1200
1080,1200 1070,1200 1060,1200 1050,1200 1040,1200 1030,1200 1020,1200
1010,1200 1000,1200 990,1200 980,1200 970,1200 960,1200 950,1200
940,1200 930,1200 920,1200 910,1200 900,1200 890,1200 880,1200
870,1200 860,1200 850,1200 840,1200 830,1200 820,1200 810,1200
800,1200 790,1200 780,1200 770,1200 760,1200 750,1200 740,1200
730,1200 720,1200 710,1200 700,1200 690,1200 680,1200 670,1200
660,1200 650,1200 640,1200 630,1200 620,1200 610,1200 600,1200
590,1200 580,1200 570,1200 560,1200 550,1200 540,1200 530,1200
520,1200 510,1200 500,1200 490,1200 480,1200 470,1200 460,1200
450,1200 440,1200 430,1200 420,1200 410,1200 400,1200 390,1200
380,1200 370,1200 360,1200 350,1200 340,1200 330,1200 320,1200
310,1200 300,1200 290,1200 280,1200 270,1200 260,1200 250,1200
240,1200 230,1200 220,1200 210,1200 200,1200 190,1200 180,1200
170,1200 160,1200 150,1200 140,1200 130,1200 120,1200 110,1200
100,1200 90,1200 80,1200 70,1200 60,1200 50,1190 50,1180 50,1170
50,1160 50,1150 50,1140 50,1130 50,1120 50,1110 50,1100 50,1090
50,1080 50,1070 50,1060 50,1050 50,1040 50,1030 50,1020 50,1010
50,1000 50,990 50,980 50,970 50,960 50,950 50,940 50,930 50,920
50,910 50,900 50,890 50,880 50,870 50,860 50,850 50,840 50,830
50,820 50,810 50,800 50,790 50,780 50,770 50,760 50,750 50,740
50,730 50,720 50,710 50,700 50,690 50,680 50,670 50,660 50,650
50,640 50,630 50,620 50,610 50,600 50,590 50,580 50,570 50,560
50,550 50,540 50,530 50,520 50,510 50,500 50,490 50,480 50,470
50,460 50,450 50,440 50,430 50,420 50,410 50,400 50,390 50,380
50,370 50,360 50,350 50,340 50,330 50,320 50,310 50,300 50,290
50,280 50,270 50,260 50,250 50,240 50,230 50,220 50,210 50,200
50,190 50,180 50,170 50,160 50,150 50,140 50,130 50,120 50,110
50,100 50,90 50,80 50,70 50,60 50,50 50))'))';

```

```

SELECT SplitTable.poly_id, SplitTable.sub_poly_id, SplitTable.splitGeom
FROM PolygonSplit
(ON
  (
    SELECT poly_id, geom, cast(400 AS INTEGER)
    FROM feature_tbl
    WHERE poly_id=100
  )
)

```

```
AS SplitTable(poly_id, sub_poly_id, splitGeom)
ORDER BY 1,2,3;
```

The following output shows that the polygon with an id of 100 was split into four pieces. The actual vertex values that define each polygon are not shown, and are instead replaced with "...".

poly_id	sub_poly_id	splitGeom
-----	-----	-----
100	0	POLYGON ((...))
100	1	POLYGON ((...))
100	2	POLYGON ((...))
100	3	POLYGON ((...))

TO_MGRS

Converts an ST_GEOMETRY point object to a coordinate in the Military Grid Reference System (MGRS).

Note:

This function was developed using GEOTRANS, a product of the National Geospatial-Intelligence Agency (NGA) and U.S. Army Engineering Research and Development Center.

Result Type

Returns a VARCHAR(20), LATIN character set MGRS coordinate string with the specified precision.

TO_MGRS Syntax

```
[TD_SYSNLIB.] TO_MGRS ( ageometry, fromWktSRS, precision )
```

Syntax Elements

ageometry

An ST_POINT type object.

fromWktSRS

The WKT representation of the source spatial reference system. The supported coordinate systems are UTM, UPS, and Geodetic.

precision

The degree of precision of the resulting MGRS coordinate string. The value should be an integer in the range of 0 through 5, where 0 represents 100,000 meter precision, and 5 represents 1 meter precision.

Argument Types and Rules

The data type of an argument that you pass to a geospatial system function does not need to exactly match the corresponding declared argument type as described in the function description. If the argument type is a compatible type, as listed in the following table, the system tries to implicitly cast the argument.

Declared Argument Type	Compatible Argument Types
DECIMAL	BYTEINT, SMALLINT, INTEGER, or BIGINT
FLOAT	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL, or NUMERIC
INTEGER	BYTEINT or SMALLINT
VARCHAR	CHAR

To pass an argument where the data type is not an exact match and is not a compatible type according to the preceding table, you must explicitly convert the argument.

Usage Notes

Vantage follows the MGRS convention of truncating, rather than rounding values during the conversion. The net effect of the truncation is to shift points to the south and west (the lowest point in each rectangle).

MGRS coordinates employ one of two lettering schemes to denote row identifiers in the grid system: MGRS-Old or MGRS-New. During conversions to and from MGRS, the MGRS lettering scheme in the returned coordinates (for the TO_MGRS function) or expected as input (to the FROM_MGRS function) depends on the spatial reference system from or to which the coordinates are being converted. If the spatial reference system is based on the Bessel, Clarke 1866, or Clarke 1880 reference ellipsoid, the MGRS coordinates use the MGRS-Old lettering scheme. Conversion to or from all other spatial reference systems use the MGRS-New scheme.

Example: TO_MGRS

This example converts Geodetic coordinates in the “WGS 84” spatial reference system to MGRS.

```
SELECT TO_MGRS(
  new ST_GEOMETRY('ST_POINT', 30, 45),
  (select SRTEXT from sysspatial.spatial_ref_sys where srid = 1619),
  5);
```

Returns:

```
TO_MGRS(NEW ST_GEOMETRY('ST_POINT', 30, 45), <Scalar Subquery>, 5 )
```

```
36TTQ6355387329
```

Geospatial UDFs

This section describes the Vantage geospatial UDFs. These are geospatial functions that have been implemented by Teradata using the Vantage user-defined function infrastructure.

For best performance, follow these practices:

- When ST_Geometry defines an instance method that performs the same functionality as a geospatial UDF, use the instance method instead of the UDF.
- When a geospatial system function exists that performs the same functionality as a geospatial UDF, use the system function instead of the UDF.

If your SQL request specifies the name of a geospatial system function that is also the name of a geospatial UDF, Vantage calls the system function. To call the UDF, qualify the name of the UDF with the SYSSPATIAL database name.

SphericalDistance

Returns the spherical distance, in meters, between two spherical coordinates on the planet using the Haversine Formula.

See also [ST_SphericalDistance](#).

Result Type

Returns a FLOAT value.

SphericalDistance Syntax

```
SYSSPATIAL.SphericalDistance ( lat1, lon1, lat2, lon2 )
```

Syntax Elements

lat1, *lon1*

The latitude and longitude of the first point on the planet.

The data type of *lat1* and *lon1* is FLOAT.

lat2, *lon2*

The latitude and longitude of the second point on the planet.

The data type of *lat2* and *lon2* is FLOAT.

Example: SphericalDistance

```
SELECT SYSSPATIAL.SphericalDistance(10, 20, 20, 30);
```

SpheroidalDistance

Returns the distance, in meters, between two spherical coordinates.

See also [ST_SpheroidalDistance](#).

Result Type

Returns a FLOAT value.

SpheroidalDistance Syntax

```
SYSSPATIAL.SpheroidalDistance ( lat1, lon1, lat2, lon2 [, semimajor, invf ] )
```

Syntax Elements

lat1, lon1

the latitude and longitude of the first point. The data type of both is FLOAT.

lat2,lon2

the latitude and longitude of the second point.

The data type of *lat2* and *lon2* is FLOAT.

semimajor

an optional FLOAT for the length in meters of the semimajor axis of the spheroid.

invf

an optional FLOAT for the inverse flattening ratio of the spheroid.

Usage Notes

For the distance calculation, the planet is modeled as a spheroid.

If you do not pass in values for the *semimajor* and *invf* arguments, the computation uses the semimajor axis and the inverse flattening ratio from the World Geodetic System, WGS84. A value of 6,378,137.0 meters is used for the semimajor axis, and a value of 298.257223563 is used for the inverse flattening ratio.

Distance calculations between antipodal or nearly antipodal points using the SYSSPATIAL.SpheroidalDistance UDF may fail to converge, generating a system error. For these distance calculations use instead the SYSSPATIAL.SphericalDistance UDF.

Example

The following two examples show how to use the SpheroidalDistance function to get the spheroidal distance between Madison (-89.39, 43.09) and Chicago (-87.65, 41.90):

```
SELECT SYSSPATIAL.SpheroidalDistance( -89.39, 43.09, -87.65, 41.90,
                                       6378137, 298.257223563);

SELECT SYSSPATIAL.SpheroidalDistance( -89.39, 43.09, -87.65, 41.90);
```

ST_GeomFromText

User-defined function that returns a specified ST_Geometry value.

Result Type

Returns an ST_Geometry value.

ST_GeomFromText Syntax

```
SYSSPATIAL.ST_GeomFromText ( awkt [, asrid ] )
```

Syntax Elements

awkt

a VARCHAR(64000) for the well-known text (WKT) representation of the spatial type. The size of *awkt* cannot exceed 64000 bytes.

When a WKT representation is larger than 64000 bytes, use the ST_Geometry constructor method (CLOB version).

For details on WKT formats, see [Well-Known Text Format](#).

asrid

an optional INTEGER for a spatial reference system identifier.

If this argument is omitted, the spatial reference system identifier is set to 0.

Usage Notes

For best performance, use the `ST_Geometry` constructor method instead of this UDF.

The `ST_GeomFromText` UDF does not support system time information, so times for geosequence types are not automatically adjusted for the system or session time zone. For geosequence types, you should use the constructor method instead of this UDF.

Example

```
CREATE TABLE varchar_tab1 (a int, b VARCHAR (60000));
INSERT INTO varchar_tab1 VALUES (1001, 'Point(10 20 30)');
CREATE TABLE sample_shapes(skey INTEGER, shape ST_Geometry);
INSERT INTO sample_shapes
SELECT a, SYSSPATIAL.ST_GeomFromText(b) FROM varchar_tab1;
SELECT * FROM sample_shapes;
```

ST_GeomFromWKB

User-defined function that returns a specified `ST_Geometry` value.

Result Type

Returns an `ST_Geometry` value.

ST_GeomFromWKB Syntax

```
SYSSPATIAL.ST_GeomFromWKB ( awkb [, asrid ] )
```

Syntax Elements

awkb

a `VARBYTE(64000)` for the well-known binary (WKB) representation of the spatial type. The size of *awkb* cannot exceed 64000 bytes.

When a WKB representation is larger than 64000 bytes, use the `ST_Geometry` constructor method (BLOB version).

For more information on WKB formats, see [Well-Known Binary Format](#).

asrid

an `INTEGER` value for a spatial reference system identifier.

If this argument is omitted, the spatial reference system identifier is set to 0.

Usage Notes

For best performance, use the ST_Geometry constructor method instead of this UDF.

Example

```
CREATE TABLE varbyte_tab1 (a int, b VARBYTE (6000));
INSERT INTO varbyte_tab1 VALUES (1002, NEW ST_Geometry(
  'Point(10 20 30)').ST_Asbinary());
INSERT INTO sample_shapes
SELECT a, SYSSPATIAL.ST_GeomFromWKB(b) FROM varbyte_tab1;
SELECT * FROM sample_shapes;
```

Geospatial Table Operators

Table operators accept a table or table expression as input and generate a table as output. They can only be specified in the FROM clause of SELECT statements.

AggGeom

Returns a union or intersection of a group of ST_Geometry objects.

Note:

This table operator replaces the deprecated [AggGeomUnion \[Deprecated\]](#) and [AggGeomIntersection \[Deprecated\]](#) functions.

Returned Values

The kind of geometry objects in the ST_Geometry type column returned by the table operator matches the type of geometry objects that were aggregated.

For Intersection operations, "GEOMETRYCOLLECTION EMPTY" is returned if there is no intersection between the objects.

If the Well-Known Binary (WKB) representation of the resulting geometry becomes larger than the maximum size geometry (~16MB), the system reports an error.

Non-Empty geometry collections are not supported, so any union or intersection that results in a collection, as either an intermediate or final result generates an error. Involving one will result in an error.

Valid Data Types

ST_Geometry objects that represent Points, LineStrings, Polygons, MultiPoints, MultiLineStrings, MultiPolygons, and GeometryCollections

AggGeom Syntax

```
[TD_SYSFNLIB.] AggGeom (
  ON ( SELECT ageometry [, part_column_list ] FROM [ database_name. ] table_name )
  [ PARTITION BY part_column_list ]
  [ USING OPERATION ( 'value' ) ]
) [AS] correlation_name [ ( column_list ) ]
```

Syntax Elements

ageometry

A constant or column expression for which the aggregation is to be computed. The data type of ageometry is ST_Geometry.

part_column_list

The column name set by which rows are to be partitioned before being passed to the operator. If a PARTITION BY clause is used, the list of columns must match the *part_column_list* that was passed to the table operator.

database_name

table_name

The table that contains the geometry type column for which the aggregation is to be computed.

value

The type of aggregation operation desired, either Union or Intersection. Union is the default. The value must be delimited by apostrophe (single-quote) characters.

correlation_name

A name to be assigned to the table returned by the table operator.

column_list

Optional list of custom names for the columns returned by the table operator. These names override the column names that would be otherwise generated by the operator. If you specify any column names, you must provide names for all columns returned. If more than one column is returned by the operator, separate the names with commas.

Usage Notes

- The first input column in the ON clause must have data type ST_GEOMETRY and contain the objects to be aggregated. After that, any number of partitioning columns may be included in the ON clause and they will be written as is to the output. The PARTITION BY clause must specify all columns other than the first or the table operator will return an error.
- NULL arguments are not included in aggregation operations. If all values in the aggregation are NULL, no rows are returned.
- If the PARTITION BY clause is omitted, each AMP performs a local aggregation operation on its rows. One output row is produced per AMP.
- Whenever input rows are redistributed to other AMPs, as caused by a PARTITION BY clause, the order of the redistribution can vary from run to run. Consequently, the geometries produced by the

aggregation can vary per run. Nevertheless, the result geometries will be equivalent, but there may be differences in the ordering of points within geometries. For example, they might display different starting and ending point for a polygon.

Similarly, the structure of the geometries can vary, for example, a single LineString can represent two LineStrings or a MultiLineString. Because the union or intersection steps can occur in a different order from run to run, precision errors that can normally happen in the computation of these operations can vary slightly.

Examples: AggGeom

AggGeom can be used to aggregate by a partition value (or list). This example performs an aggregate union of geometries by zip code. If you invoke AggGeom with the PARTITION BY clause on the zip code column, all of the rows are redistributed by zip code, returning one output row per zip code.

```
CREATE TABLE geom_table(pkey integer, zipcode CHAR(5), geom
ST_Geometry);

INSERT INTO geom_table VALUES(
  0, '92127', new ST_Geometry('LineString(10 10, 20 20)'));
INSERT INTO geom_table VALUES(
  1, '92127', new ST_Geometry('LineString(20 20, 30 30, 40 5)'));
INSERT INTO geom_table VALUES(
  2, '90338', new ST_Geometry('LineString(40 5, 50 50)'));

SELECT zipcode, geom
FROM AggGeom( ON (
  SELECT geom, zipcode
  FROM geom_table)
PARTITION BY zipcode
USING Operation('Union') ) L;

zipcode geom
-----
90338    LINESTRING (40 5,50 50)
92127    MULTILINESTRING ((20 20,30 30,40 5),(10 10,20 20))
```

This second example aggregates all geometry values using a local aggregation and a final global aggregation.

The AggGeom operator can be invoked twice within a statement to perform a single aggregation of all input geometry values. If you invoke AggGeom once from SQL without the PARTITION BY clause, it will perform a local aggregation on each AMP, and one row will be returned from each amp.

To do a final union from all AMPs, nest a second call to AggGeom. The inner call to AggGeom performs the local aggregation on each AMP, and the outer call will do a final aggregation of the local aggregations and produce a single result row that represents the union of all geometries.

Use the Partition BY clause with a constant value to perform the final aggregation on a single amp. By specifying a constant value, like the number 1, the locally aggregated rows from each AMP will be in the same partition, so will all be redistributed to the same AMP for the final aggregation. A single output row is returned.

You could also aggregate all values with a single call to AggGeom by partitioning by a constant value, but that would re-distribute all rows to a single AMP to perform the aggregation. This would not take advantage of the parallel processing capability of Vantage. By using two calls to AggGeom, the AMPs all perform local aggregations in parallel, and only the final aggregation is performed on a single AMP.

```
CREATE TABLE geom_table(pkey integer,
                        zipcode CHAR(5),
                        geom ST_Geometry);

INSERT INTO geom_table VALUES(
  0, '92127', new ST_Geometry('LineString(10 10, 20 20)'));
INSERT INTO geom_table VALUES(
  1, '92127', new ST_Geometry('LineString(20 20, 30 30, 40 5)'));
INSERT INTO geom_table VALUES(
  2, '90338', new ST_Geometry('LineString(40 5, 50 50)'));

SELECT *
FROM AggGeom( ON (
  SELECT L.*, 1 as p from AggGeom( ON (
    SELECT geom from geom_table)
  USING Operation('Union') ) L)
PARTITION BY p
) G;

geom
-----
MULTILINESTRING ((40 5,50 50),(20 20,30 30,40 5),(10 10,20 20))
```

GeometryToRows

Takes ST_Geometry objects and produces an output row for each point in the geometry object.

Valid Data Types

ST_Geometry objects that represent Points, LineStrings, Polygons, MultiPoints, MultiLineStrings, and MultiPolygons.

GeometryToRows Syntax

```
[TD_SYSFNLIB.] GeometryToRows (
  ON { [ database_name. ] table_name | ( query_expression ) }
  [ USING ErrOnUnsupportedGeometry ('value') ]
) [AS] correlation_name [ ( column_list ) ]
```

Syntax Elements

database_name

table_name

A table that contains the ST_Geometry types, the component points of which are returned as individual rows in the table returned by the table operator. The table must be defined to have two or three columns, as described for the *query_expression* subquery: one or two ID columns and an ST_Geometry type column.

query_expression

```
SELECT id1, [ id1, ] id1 FROM [ database_name. ] table_name
```

A SELECT subquery that provides input data for the table operator.

value

The table that contains the geometry type column for which the aggregation is to be computed.

value

Whether errors are reported for unsupported geometries that are passed to the table function. *value* can be either Yes or No. The default is No. The value must be delimited by apostrophe (single-quote) characters.

When *value* is No, or this argument is omitted, unsupported geometries are ignored.

correlation_name

A name to be assigned to the table returned by the table operator.

id1

A numeric or character type column that uniquely identifies each geometry object in the table passed to the operator. *id1* can be any valid Teradata numeric or character data type except CLOB. If *id1* is NULL for a particular input geometry object, the rows returned by the table operator for this geometry object show corresponding NULLs in their out_geom_id1 column.

id2

A numeric or character type column that, combined with *id1*, uniquely identifies each geometry object in the table passed to the operator. *id2* can be any valid Teradata numeric or character data type except CLOB. If *id2* is NULL for a particular input geometry object, the rows returned by the table operator for this geometry object show corresponding NULLs in their out_geom_id2 column.

If an *id2* column is not passed to the table operator, an out_geom_id2 column is not returned.

ageometry

An ST_Geometry type column that contains geometry objects. For each component point of each object, the table operator returns a separate row that uniquely identifies the object, the geometric type, and the x, y, and z coordinates that locate the point in space.

correlation_name

name to be assigned to the table returned by the table operator.

column_list

Optional list of custom names for the columns returned by the table operator. These names override the column names that would be otherwise generated by the operator. If you specify any column names, you must provide names for all columns returned. If more than one column is returned by the operator, separate the names with commas.

Return Type

The GeometryToRows table operator returns an eight or nine column table with the following columns.

Column Name	Description
<i>Geometry object identifier column</i>	The value of the input <i>id1</i> argument, which is passed directly to the output of the GeometryToRows operator. Identifies the points that are returned for a particular input geometry. The type of this column matches the type of <i>id1</i> in the input table.

Column Name	Description
	If the SELECT statement that uses the table operator does not specify a column list in the AS clause, the name of this column matches the name of the <i>id1</i> argument to the table operator.
<i>Optional second geometry object identifier column</i>	If the input table included an <i>id2</i> column to further identify the input geometry objects, that value is passed through the table operator and appears in a second column in the returned table. The this column matches the type of <i>id2</i> in the input table. If the input table does not have an <i>id2</i> column, there is no corresponding column in the table returned by the GeometryToRows table operator.
element_id	An INTEGER value that identifies the particular component element of a Multi type geometry to which the point represented by this row belongs. Individual elements in a Multi geometry are numbered starting from 1. For example, if a MutliLineString contains two LineStrings, the points for the first LineString will have an element_id of 1 and the points from the second LineString will have an element_id of 2. LineString will have an element_id of 2.
ring_id	An INTEGER value that identifies the particular polygon ring of a Polygon or MultiPolygon to which the point represented by this row belongs. Individual rings for a polygon are numbered starting from 1, with the exterior ring being number 1, and the interior rings numbered starting from 2. If the geometry object is not a Polygon or MultiPolygon, the ring_id will be set to NULL.
point_id	An INTEGER value that reflects the ordering of the point within the geometry. point_id numbering starts with 1. The points for each geometry are numbered in the order in which they occur within the geometry.
geomType	The geometry type of the input geometry. The possible values are ST_Point, ST_MultiPoint, ST_LineString, ST_MultiLineString, ST_Polygon, or ST_MultiPolygon. If the input geometry is NULL, this value will be NULL. The data type of this value is VARCHAR(30) CHARACTER SET LATIN.
x	The x coordinate value for the point represented by the row. This is a DOUBLE PRECISION value. If the input geometry is an empty set, this value is NULL.
y	The y coordinate value for the point represented by the row. This is a DOUBLE PRECISION value. If the input geometry is an empty set, this value is NULL.
z	The z coordinate value for the point represented by the row. This is a DOUBLE PRECISION value. If the geometry is a 2D geometry, or if the input geometry is an empty set, this value is NULL.

Usage Notes

- The first and last points in a polygon will be the same point.

- A polygon consists of one exterior ring and zero or more interior rings. The interior rings must lie completely within the exterior ring. A MultiPolygon is a group of non-overlapping polygons.
- The element ID of each point identifies which element the point belongs to in a Multi-type geometry. For example, for a MultiLineString that contains two LineStrings, the first LineString will be element 1 and the second LineString will be element 2.

For non-Multi geometries, the element ID is always 1.

- If an input geometry value is NULL, the corresponding output row that is returned will have NULLs in the geomType, x, y, and z columns.
- Due to the hierarchical nature of the ID columns, you can use an ORDER BY clause to keep the rows associated with each passed-in geometry together, and grouped according to the sub-geometries of Multi types. See the example below.

Example: GeometryToRows

```
CREATE TABLE geo_table(pkey integer, geom_id VARCHAR(20), geom ST_Geometry);

INSERT INTO geo_table VALUES(0, 'S101', new ST_Geometry('Point(10 20 5)'));
INSERT INTO geo_table VALUES(1, 'S102', new ST_Geometry('LineString(10 10,
20 20)'));
INSERT INTO geo_table VALUES(2, 'S103', new ST_Geometry(
  'Polygon((0 0, 0 10, 10 10, 10 0, 0 0))'));
INSERT INTO geo_table VALUES(3, 'S104', new ST_Geometry('LineString(EMPTY)'));
INSERT INTO geo_table VALUES(4, 'S105', NULL);
INSERT INTO geo_table VALUES(5, 'S106', new ST_Geometry(
  'MultiLineString((10 10, 20 20),(30 30, 40 40, 50 50))'));

INSERT INTO geo_table VALUES(6, 'S107', new ST_Geometry(
  'Polygon((0 0, 0 10, 10 10, 10 0, 0 0), (2 2, 2 8, 8 8, 8 2, 2 2))'));

SELECT PointsTable.geom_id1 (FORMAT 'X(8)'),
       PointsTable.element_id (FORMAT 'Z'),
       PointsTable.ring_id (FORMAT 'Z'),
       PointsTable.point_id (FORMAT 'Z'),
       PointsTable.geomType (FORMAT 'X(15)'),
       PointsTable.x (FORMAT '99.99'),
       PointsTable.y (FORMAT '99.99'),
       PointsTable.z (FORMAT '99.99')

FROM GeometryToRows(ON (SELECT geom_id, geom FROM geo_table) )
AS PointsTable (geom_id1, element_id, ring_id, point_id, geomType, x, y, z)
ORDER BY 1, 2, 3, 4;
```

The output would look like the following.

geom_id1	element_id	ring_id	point_id	geomType	x	y	z
S101	1	?	1	POINT	10.00	20.00	05.00
S102	1	?	1	LINESTRING	10.00	10.00	?
S102	1	?	2	LINESTRING	20.00	20.00	?
S103	1	1	1	POLYGON	00.00	00.00	?
S103	1	1	2	POLYGON	00.00	10.00	?
S103	1	1	3	POLYGON	10.00	10.00	?
S103	1	1	4	POLYGON	10.00	00.00	?
S103	1	1	5	POLYGON	00.00	00.00	?
S104	1	?	1	LINESTRING	?	?	?
S105	1	?	1	?	?	?	?
S106	1	?	1	MULTILINESTRING	10.00	10.00	?
S106	1	?	2	MULTILINESTRING	20.00	20.00	?
S106	2	?	1	MULTILINESTRING	30.00	30.00	?
S106	2	?	2	MULTILINESTRING	40.00	40.00	?
S106	2	?	3	MULTILINESTRING	50.00	50.00	?
S107	1	1	1	POLYGON	00.00	00.00	?
S107	1	1	2	POLYGON	00.00	10.00	?
S107	1	1	3	POLYGON	10.00	10.00	?
S107	1	1	4	POLYGON	10.00	00.00	?
S107	1	1	5	POLYGON	00.00	00.00	?
S107	1	2	1	POLYGON	02.00	02.00	?
S107	1	2	2	POLYGON	02.00	08.00	?
S107	1	2	3	POLYGON	08.00	08.00	?
S107	1	2	4	POLYGON	08.00	02.00	?
S107	1	2	5	POLYGON	02.00	02.00	?

Notice how the ORDER BY clause causes the rows to be arranged by ID values, keeping the rows for each geometry and sub-geometry (for Multi types) together and organized hierarchically by geometries and their sub-geometries. Otherwise, the rows would be returned in a random order, and the relationships would be more difficult to discern.

PolygonSplit

Table operator that accepts a table containing ST_Geometry Polygon types and recursively processes qualifying polygons into sets of smaller sub-polygons, each having fewer vertices than the original polygon. Taken together, the sub-polygons closely approximate the shape and area of the original polygon.

If the input polygon exceeds a specified maximum number of vertices, PolygonSplit divides the polygon into four sub-polygons, one occupying each quadrant of the input polygon. The sub-polygons thus formed

are similarly, recursively split until each resulting polygon has fewer than the specified maximum number of vertices.

The input table must include an ST_Geometry column to hold a polygon and an identifier column to hold a unique identifier for each polygon. The sub-polygons that are created all share the identifier of the original polygon that was split.

PolygonSplit Syntax

```
[TD_SYSFNLIB.] PolygonSplit (
  ON { [ database_name.. ] table_name |

      ( SELECT polygon_ID, geom [, max_vertices ]
        FROM [ database_name.. ] table_name
      )
  }
) [AS] correlation_name [ ( column_list )
```

Syntax Elements

polygon_ID

A numeric or character value that uniquely identifies the polygon or multipolygon to be split. This same value is returned for each of the sub-polygons that is returned as a result of the splitting, and can be used to associate the sub-polygons with the original polygon.

polygon_ID can be any valid Teradata numeric or character data type except CLOB.

If *polygon_ID* is NULL for a given polygon, the results will show corresponding NULLs for the IDs of the unchanged polygon or for the sub-polygons that are generated.

geom

The polygon or multipolygon geometry to be split.

PolygonSplit accepts all types of ST_Geometry, but only 2D polygons and 2D multipolygons are split. Other geometries are returned unchanged.

If *geom* is NULL, PolygonSplit returns NULL for that geometry.

max_vertices

The maximum number of vertices a returned sub-polygon can include. If the original polygon or multipolygon has fewer than this number of vertices, it is returned unchanged.

max_vertices must be greater than or equal to 10. The default is 300.

If *max_vertices* is NULL, PolygonSplit defaults to 300.

[*database_name*.]

table_name

The name and optional database location of the table that contains the polygon and multipolygon geometries to be split.

correlation_name

A name to be assigned to the table returned by the table operator.

column_list

Optional list of custom names for the columns returned by the table operator.

These names override the column names that would be otherwise generated by the operator. If you specify any column names, you must provide names for all columns returned. If more than one column is returned by the operator, separate the names with commas.

Result Rows

Returns the a table having the following columns:

Column Name	Data Type	Description
out_polygon_ID	Matches type of <i>polygon_ID</i> input parameter	All sub-polygons that result from the splitting of a polygon or multipolygon will share the ID of the input polygon or multipolygon. Geometries that are unchanged by PolygonSplit will retain their original ID in the results table.
sub_polygon_ID	INTEGER	Uniquely identifies each subpolygon that came from a particular input polygon or multipolygon. The subpolygons are numbered sequentially as they are created. Numbering starts at 0.
split_geom	ST_Geometry	A sub-polygon that results from splitting a polygon or multipolygon. For input polygons and multipolygons that have fewer than the maximum specified number of vertices, and for all other geometries, this column contains the original, unchanged input geometry.

Usage Notes

The splitting process is subject to precision and rounding errors that are inherent with floating-point arithmetic. Consequently, there can be small discrepancies between the area of the original polygon and the sum of the areas of the resulting sub-polygons.

Example: PolygonSplit

```
CREATE TABLE feature_tbl(poly_id INTEGER, geom ST_Geometry);

INSERT INTO feature_tbl VALUES(100, new ST_Geometry('POLYGON ((50
50,50 60,50 70,50 80,50 90,50 100,50 110,50 120,50 130,50 140,50
150,50 160,50 170,50 180,50 190,50 200,50 210,50 220,50 230,50 240,50
250,50 260,50 270,50 280,50 290,50 300,50 310,50 320,50 330,50 340,50
350,50 360,50 370,50 380,50 390,50 400,50 410,50 420,50 430,50 440,50
450,50 460,50 470,50 480,50 490,50 500,50 510,50 520,50 530,50 540,50
550,50 560,50 570,50 580,50 590,50 600,50 610,50 620,50 630,50 640,50
650,50 660,60 670,50 680,50 690,50 700,50 710,50 720,50 730,50 740,50
750,50 760,50 770,50 780,50 790,50 800,50 810,50 820,50 830,50 840,50
850,50 860,50 870,50 880,50 890,50 900,50 910,50 920,50 930,50 940,50
950,50 960,50 970,50 980,50 990,50 1000,50 1010,50 1020,50 1030,50
1040,50 1050,50 1060,50 1070,50 1080,50 1090,50 1100,50 1110,50
1120,50 1130,50 1140,50 1150,50 1160,50 1170,50 1180,50 1190,50
1200,60 1200,70 1200,80 1200,90 1200,100 1200,110 1200,120 1200,130
1200,140 1200,150 1200,160 1200,170 1200,180 1200,190 1200,200
1200,210 1200,220 1200,230 1200,240 1200,250 1200,260 1200,270
1200,280 1200,290 1200,300 1200,310 1200,320 1200,330 1200,340
1200,350 1200,360 1200,370 1200,380 1200,390 1200,400 1200,410
1200,420 1200,430 1200,440 1200,450 1200,460 1200,470 1200,480
1200,490 1200,500 1200,510 1200,520 1200,530 1200,540 1200,550
1200,560 1200,570 1200,580 1200,590 1200,600 1200,610 1200,620
1200,630 1200,640 1200,650 1200,660 1200,670 1200,680 1200,690
1200,700 1200,710 1200,720 1200,730 1200,740 1200,750 1200,760
1200,770 1200,780 1200,790 1200,800 1200,810 1200,820 1200,830
1200,840 1200,850 1200,860 1200,870 1200,880 1200,890 1200,900
1200,910 1200,920 1200,930 1200,940 1200,950 1200,960 1200,970
1200,980 1200,990 1200,1000 1200,1010 1200,1020 1200,1030 1200,1040
1200,1050 1200,1060 1200,1070 1200,1080 1200,1090 1200,1100 1200,1110
1200,1120 1200,1130 1200,1140 1200,1150 1200,1160 1200,1170 1200,1180
1200,1190 1200,1200 1200,1200 1190,1200 1180,1200 1170,1200 1160,1200
1150,1200 1140,1200 1130,1200 1120,1200 1110,1200 1100,1200 1090,1200
1080,1200 1070,1200 1060,1200 1050,1200 1040,1200 1030,1200 1020,1200
1010,1200 1000,1200 990,1200 980,1200 970,1200 960,1200 950,1200
940,1200 930,1200 920,1200 910,1200 900,1200 890,1200 880,1200
870,1200 860,1200 850,1200 840,1200 830,1200 820,1200 810,1200
800,1200 790,1200 780,1200 770,1200 760,1200 750,1200 740,1200
730,1200 720,1200 710,1200 700,1200 690,1200 680,1200 670,1200
660,1200 650,1200 640,1200 630,1200 620,1200 610,1200 600,1200
590,1200 580,1200 570,1200 560,1200 550,1200 540,1200 530,1200
```

```

520,1200 510,1200 500,1200 490,1200 480,1200 470,1200 460,1200
450,1200 440,1200 430,1200 420,1200 410,1200 400,1200 390,1200
380,1200 370,1200 360,1200 350,1200 340,1200 330,1200 320,1200
310,1200 300,1200 290,1200 280,1200 270,1200 260,1200 250,1200
240,1200 230,1200 220,1200 210,1200 200,1200 190,1200 180,1200
170,1200 160,1200 150,1200 140,1200 130,1200 120,1200 110,1200
100,1200 90,1200 80,1200 70,1200 60,1200 50,1190 50,1180 50,1170
50,1160 50,1150 50,1140 50,1130 50,1120 50,1110 50,1100 50,1090
50,1080 50,1070 50,1060 50,1050 50,1040 50,1030 50,1020 50,1010
50,1000 50,990 50,980 50,970 50,960 50,950 50,940 50,930 50,920
50,910 50,900 50,890 50,880 50,870 50,860 50,850 50,840 50,830
50,820 50,810 50,800 50,790 50,780 50,770 50,760 50,750 50,740
50,730 50,720 50,710 50,700 50,690 50,680 50,670 50,660 50,650
50,640 50,630 50,620 50,610 50,600 50,590 50,580 50,570 50,560
50,550 50,540 50,530 50,520 50,510 50,500 50,490 50,480 50,470
50,460 50,450 50,440 50,430 50,420 50,410 50,400 50,390 50,380
50,370 50,360 50,350 50,340 50,330 50,320 50,310 50,300 50,290
50,280 50,270 50,260 50,250 50,240 50,230 50,220 50,210 50,200
50,190 50,180 50,170 50,160 50,150 50,140 50,130 50,120 50,110
50,100 50,90 50,80 50,70 50,60 50,50 50))'))';

```

```

SELECT SplitTable.poly_id, SplitTable.sub_poly_id, SplitTable.splitGeom
FROM PolygonSplit
  (ON
    (
      SELECT poly_id, geom, cast(400 AS INTEGER)
      FROM feature_tbl
      WHERE poly_id=100
    )
  )
AS SplitTable(poly_id, sub_poly_id, splitGeom)
ORDER BY 1,2,3;

```

The following output shows that the polygon with an id of 100 was split into four pieces. The actual vertex values that define each polygon are not shown, and are instead replaced with "..."

poly_id	sub_poly_id	splitGeom
-----	-----	-----
100	0	POLYGON ((...))
100	1	POLYGON ((...))
100	2	POLYGON ((...))
100	3	POLYGON ((...))

Geospatial Predicates and the Optimizer

Geospatial NUSIs can be used by the Teradata Optimizer to speed access to geospatial data for queries that use single-table predicates and join predicates. For more information on Geospatial Indexes, see [Geospatial Indexes](#).

The following geospatial methods can be used to form predicates:

- [MBR_Filter](#)
- [ST_3DDistance Method](#)
- [ST_Contains Method](#)
- [ST_Crosses Method](#)
- [ST_Distance Method](#)
- [ST_Equals Method](#)
- [ST_Intersects Method](#)
- [ST_Overlaps Method](#)
- [ST_Touches Method](#)
- [ST_Within Method](#)
- [Within_MBB](#)
- [Intersects_MBB](#)
- [ST_Relate Method](#) (if this predicate is configured to execute one of the above methods)

Note:

Predicates that use the ST_Distance and ST_3DDistance methods have a slightly different form than those that use the other listed methods.

Geospatial Single-Table Predicates

Geospatial single-table predicates cause the optimizer to evaluate a geospatial index, if one exists, as a potential access path to the geospatial data. In these circumstances, the index can greatly speed access to the data and execution of the query.

Geospatial Single-Table Predicate Syntax

Single-Table Predicate

For single-table predicates except distance methods ST_Distance and ST_3DDistance:

```
SELECT a, b FROM TableName WHERE {
    TableName.GeoCol.SupportedGeoMethod (GeospatialLiteralExpression) = 1 |
```

```

    GeospatialLiteralExpression.SupportedGeoMethod ( TableName.GeoCol) = 1 |
    TableName.GeoCol IS NULL
}

```

Note:

The expression must be set to evaluate to 1 (true) for the first two forms to qualify as single-table predicates.

Single-Table Distance Predicate

For distance methods ST_Distance and ST_3DDistance:

```

SELECT a, b FROM TableName WHERE {
    TableName.GeoCol.SupportedGeoDistanceMethod ( GeospatialLiteralExpression)
    { < | <= } DistanceLiteral |

    GeospatialLiteralExpression.SupportedGeoDistanceMethod( TableName.GeoCol)
    { < | <= } DistanceLiteral
}

```

Syntax Elements***TableName***

Table containing a geospatial data column.

GeoCol

Geospatial data column defined as one of the ST_GEOMETRY types.

SupportedGeoMethod

One of the geospatial methods listed above, excluding ST_Distance and ST_3DDistance.

SupportedGeoDistanceMethod

Either ST_Distance or ST_3DDistance.

GeospatialLiteralExpression

An arbitrary geospatial literal expression that can be folded (simplified) in to a geospatial literal value.

DistanceLiteral

A floating point value representing a distance.

Usage Notes

How the Optimizer Determines Index Utilization

The Teradata Optimizer weighs the cost and benefit of using the alternative data paths to execute a query.

If a table has a secondary index, the Optimizer can use the index when executing a query against that table, rather than performing a more costly full table scan. By using the index, the Optimizer can reduce the number of rows that must be considered for predicate qualification, usually resulting in faster access to the necessary data.

Although use of the index can reduce the amount of data that needs to be scanned, there is a cost to using the index due to writing, reading, and manipulating an auxiliary Row ID spool.

The Optimizer uses factors such as the following to judge whether to use an available index:

- The selectivity of the single-table predicate determines how many rows of the table are selected. It determines the cardinality of the Row ID spool. The Optimizer is more likely to use an index to satisfy a query that uses more selective single-table predicates.

The accuracy with which the Optimizer can determine the selectivity of the predicate depends on whether statistics have been collected on the geospatial column, and whether the collected statistics reflect the current data demographics.

- The size of the Row ID Spool row, which depends on the number of columns present in the selectivity list or WHERE-clause. This factor, together with the selectivity, determines the number of IO blocks present in the Row ID spool.
- The size of the table rows. In general, larger rows make it more cost effective for the Optimizer to use the index.

For more information on how the Teradata Optimizer chooses execution paths for queries, see *Teradata Vantage™ - SQL Request and Transaction Processing*, B035-1142.

Examples: Geospatial Single-Table Predicates

The following examples show the use of geospatial single-table predicates that enable the optimizer to use a geospatial index for data access.

```
CREATE TABLE StatT2
(a INT,b CHAR(20000),sp ST_Geometry)
INDEX(sp);

INSERT StatT2(1, 'Teradata01','Point(10 10 10)');
```

The number of rows inserted into the table, and the nature of the Geospatial Data itself will play a significant role in whether the Optimizer chooses to use the index.

```

SELECT a,b(format 'x(10)'), sp
FROM StatT2
WHERE StatT2.sp.ST_WITHIN(CAST(
'POLYGON((0 0 0, 0 50 50, 50 50 50, 50 0 50, 0 0 0))'
AS ST_Geometry)) = 1;

SELECT a,b (format 'x(10)'), sp
FROM StatT2
WHERE StatT2.sp.ST_WITHIN(NEW ST_GEOMETRY(
'POLYGON((0 0 0, 0 50 50, 50 50 50, 50 0 50, 0 0 0))')) = 1;

SELECT a,b (format 'x(10)'), sp
FROM StatT2
WHERE StatT2.sp.ST_WITHIN(NEW ST_GEOMETRY(
'POLYGON((0 0, 0 20, 20 20, 20 0, 0 0))').ST_INTERSECTION(
'POLYGON((0 0, 0 15, 15 15, 15 0, 0 0))')) = 1;

```

The following shows an example of EXPLAIN output showing the optimizer utilizing a geospatial index.

Explanation

```

-----
1) First, we lock JOSHUA.StatT2 for read on a reserved RowHash to
prevent global deadlock.
2) Next, we lock JOSHUA.StatT2 for read.
3) We do an all-AMPs SUM step to aggregate from JOSHUA.StatT2  by way
  of Spatial index # 4 "Retrieving from CDT indexed column
JOSHUA.StatT2.sp via CDT key expression NEW ST_GEOMETRY (
'POLYGON((0 0, 0 20, 20 20, 20 0, 0 0))'(VARCHAR(64000), CHARACTER
SET LATIN, NOT CASESPECIFIC)).ST_INTERSECTION (CAST(('POLYGON((9
9, 9 18, 18 18, 9 9))'(VARCHAR(64000), CHARACTER SET LATIN, NOT
CASESPECIFIC)) AS SYSUDTLIB.ST_Geometry)).ST_MBR ()"with a
residual condition of ("(JOSHUA.StatT2.sp .ST_WITHIN (NEW
ST_GEOMETRY ('POLYGON((0 0, 0 20, 20 20, 20 0, 0
0))'(VARCHAR(64000), CHARACTER SET LATIN, NOT
CASESPECIFIC)).ST_INTERSECTION (CAST(('POLYGON((9 9, 9 18, 18 18,
9 9))'(VARCHAR(64000), CHARACTER SET LATIN, NOT CASESPECIFIC)) AS
SYSUDTLIB.ST_Geometry))))= 1"), and the grouping identifier in
field 1027. Aggregate Intermediate Results are computed globally,
then placed in Spool 3. The size of Spool 3 is estimated with no
confidence to be 3 rows (63 bytes). The estimated time for this
step is 0.07 seconds.
4) We do an all-AMPs RETRIEVE step from Spool 3 (Last Use) by way of
an all-rows scan into Spool 1 (group_amps), which is built locally

```

on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with no confidence to be 3 rows (96 bytes). The estimated time for this step is 0.04 seconds.

5) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.11 seconds.

Geospatial Join Predicates

Geospatial join predicates cause the optimizer to consider executing a nested join to speed access to the data required to satisfy the geospatial query.

Geospatial Join Predicate Syntax

Join Predicate

For join predicates except distance methods ST_Distance and ST_3DDistance (where *GeoColExpression* contains a column from *Table2*):

```
SELECT a, b FROM {
  Table1, Table2 WHERE Table1.GeoCol.SupportedGeoMethod
    ( { Table2.GeoCol | GeoColExpression | Table1.GeoCol } ) = 1 |

  Table2 JOIN Table1 ON Table1.GeoCol.SupportedGeoMethod( Table2.GeoCol)=1
}
```

Note:

The expressions must be set to evaluate to 1 (true) for these to qualify as join predicates.

Join Distance Predicate

For distance methods ST_Distance and ST_3DDistance:

```
SELECT a, b FROM {
  Table1, Table2 WHERE
    Table1.GeoCol.SupportedGeoDistanceMethod
      ( { Table2.GeoCol | GeoColExpression | Table1.GeoCol } { < | <= }
      DistanceLiteral |

  Table2 JOIN Table1 ON Table1.GeoCol.SupportedGeoDistanceMethod
```

```
(Table2.GeoCol) { < | <= } DistanceLiteral
}
```

Syntax Elements

Table1

Table2

Table containing geospatial data columns.

GeoCol

An ST_Geometry column.

Note:

At least one side of the join, the owner expression or the argument, must be an ST_Geometry column that has a geospatial index.

GeoColExpression

An expression that evaluates to an ST_Geometry value and contains one reference to a column in one of the tables being joined.

SupportedGeoMethod

Any of the geospatial methods, excluding ST_Distance and ST_3DDistance, listed in [Geospatial Predicates and the Optimizer](#).

SupportedGeoDistanceMethod

Either ST_Distance or ST_3DDistance.

DistanceLiteral

A floating point value representing a distance.

Usage Notes

How the Optimizer Determines Nested Join Utilization

The Teradata Optimizer weighs the costs against the benefits of using a nested join strategy for satisfying a query. The Optimizer uses factors such as the following to judge whether to construct a nested join:

- The cardinality and size of the duplicated table.

If there is a geospatial index defined on only one of the tables involved in the join operation, the Optimizer will duplicate the non-indexed table.

If there is a geospatial index on both tables, the optimizer will calculate the costs of duplicating each table and using the index of the other table, and choose the most cost-effective combination.

- The cardinality and size of the row ID spool.

The number of rows populating the row ID spool depends on the selectivity of the join.

The row size of a row lying within the row ID spool depends on the number of columns belonging to the duplicated table, which were referenced in the query.

For more information on how the Teradata Optimizer chooses execution paths for queries, see *Teradata Vantage™ - SQL Request and Transaction Processing*, B035-1142.

Geospatial Join Predicate Examples

Example: Using Geospatial Join Predicates

The following example shows the use of geospatial join predicates to determine which points in one table lie within polygons defined in a second table. For such a query the Teradata Optimizer would consider creating a nested join to speed data access.

```
CREATE TABLE DB_TEST.T1
(a INTEGER, b char(20000), GeoCol ST_Geometry)
INDEX (GeoCol);

CREATE TABLE DB_TEST.T2
(pkey INTEGER, buffer char(20000), Geom ST_Geometry)
INDEX (Geom);

INSERT INTO DB_TEST.T1 (1, 'Teradata01', 'POINT(10 20)');
INSERT INTO DB_TEST.T1 (1, 'Teradata02', 'POINT(20 30)');

INSERT INTO DB_TEST.T2 (1, 'Teradata01', 'POLYGON((3 3, 3 8, 8 8, 8 3, 3 3))');
INSERT INTO DB_TEST.T2 (1, 'Teradata02', 'POLYGON((15 15, 15 35, 35 35, 35 15, 15 15))');
```

```
SELECT a,b (format 'x(12)'), geocol
FROM T2
INNER JOIN T1 ON T1.GeoCol.ST_WITHIN(T2.Geom)= 1;
```

a	b	GeoCol
1	Teradata02	POINT (20 30)

The number of rows inserted into the table and the nature of the geospatial data itself determines whether the Optimizer chooses to use a geospatial index. In this simple example, with only two rows in each table, the geospatial index would probably not be used. With significantly larger tables or more complex shapes, and depending on the nature and configuration of the database, the Optimizer could use the index to speed query processing. If it did use the index, the EXPLAIN would look similar to this, where step 5 shows the geospatial index use:

Explanation

-
- 1) First, we lock DB_TEST.T1 for read on a reserved RowHash to prevent global deadlock.
 - 2) Next, we lock DB_TEST.T2 for read on a reserved RowHash to prevent global deadlock.
 - 3) We lock DB_TEST.T1 for read, and we lock DB_TEST.T2 for read.
 - 4) We do an all-AMPs RETRIEVE step from DB_TEST.T2 by way of an all-rows scan with a condition of ("DB_TEST.T2.b > 1") into Spool 2 (all_amps), which is duplicated on all AMPs. The size of Spool 2 is estimated with no confidence to be 116 rows (1,168,468 bytes). The estimated time for this step is 0.67 seconds.
 - 5) We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of an all-rows scan, which is joined to DB_TEST.T1 by way of Spatial index # 4 "Retrieving from CDT indexed column DB_TEST.T1.sp via CDT key expression {LeftTable}.sp .ST_MBR ()"extracting row ids only. Spool 2 and DB_TEST.T1 are joined using a nested join, with a join condition of ("(1=1)"). The result goes into Spool 3 (all_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 3 by field Id 1. The size of Spool 3 is estimated with no confidence to be 1 row (10,083 bytes). The estimated time for this step is 0.01 seconds.
 - 6) We do an all-AMPs JOIN step from Spool 3 (Last Use) by way of an all-rows scan, which is joined to DB_TEST.T1 by way of an all-rows scan with a condition of ("DB_TEST.T1.b > 1"). Spool 3 and DB_TEST.T1 are joined using a row id join, with a join condition of ("(sp .ST_WITHIN ({RightTable}.sp))= 1"). The result goes into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with no confidence to be 1 row (95 bytes). The estimated time for this step is 0.05 seconds.
 - 7) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.73 seconds.

Using Both an ST_Geometry Column and a Column Expression in a Geospatial Join

The following example shows that one side of a geospatial join, either the owner expression or the argument, can be a column expression that contains a column and evaluates to an ST_Geometry value. The other side of the join must be a geospatial column that has a geospatial index. The example uses the ST_DISTANCE predicate. `g2.geo.ST_Buffer(1)` is an expression containing a geospatial column

from g2 that generates an ST_Geometry object. g1.geo is an ST_Geometry column in g1 that has a geospatial index.

```
SELECT * FROM geotable g1, geotable2 g2
WHERE g1.geo.ST_Distance (g2.geo.ST_Buffer(1)) < 5
ORDER BY g1.a;
```

Geospatial Metadata (SYSSPATIAL Database)

The SYSSPATIAL database contains two tables that provide additional information about columns of type ST_Geometry and the spatial reference systems. These tables follow the definitions provided by the SQL/MM Spatial standard.

Table	Description
GEOMETRY_COLUMNS Table	Provides metadata for every table column defined as ST_Geometry. Teradata provides stored procedures to add and drop geometry metadata in this table. For more information on these procedures, see AddGeometryColumn Stored Procedure , AddGeometryColumn_3D Stored Procedure , and DropGeometryColumn Stored Procedure
SPATIAL_REF_SYS Table	Contains information about each spatial reference system. During installation, the Database Initialization Program (DIP) utility executes a script that populates the SPATIAL_REF_SYS table. For more information on DIP, see <i>Teradata Vantage™ - Database Utilities</i> , B035-1102.

GEOMETRY_COLUMNS Table

Provides metadata for every table column defined as ST_Geometry.

Syntax

```
CREATE TABLE SYSSPATIAL.GEOMETRY_COLUMNS, Fallback (
  F_TABLE_CATALOG CHARACTER VARYING(256) CHARACTER SET LATIN NOT NULL,
  F_TABLE_SCHEMA CHARACTER VARYING(128) CHARACTER SET UNICODE NOT NULL,
  F_TABLE_NAME CHARACTER VARYING(128) CHARACTER SET UNICODE NOT NULL,
  F_GEOMETRY_COLUMN CHARACTER VARYING(128) CHARACTER SET UNICODE NOT NULL,
  COORD_DIMENSION INTEGER,
  SRID INTEGER REFERENCES SPATIAL_REF_SYS,
  GEOM_TYPE CHARACTER VARYING(30) CHARACTER SET LATIN NOT NULL,
  UxMin FLOAT,
  UyMin FLOAT,
  UxMax FLOAT,
  UyMax FLOAT,
  CONSTRAINT GC_PK PRIMARY KEY
    (F_TABLE_CATALOG, F_TABLE_SCHEMA, F_TABLE_NAME, F_GEOMETRY_COLUMN)
);
```

Syntax Elements**F_TABLE_CATALOG**

Catalog name (can be "" for Teradata).

F_TABLE_SCHEMA

Database name.

F_TABLE_NAME

Table name.

F_GEOMETRY_COLUMN

Column name that is declared as ST_Geometry.

COORD_DIMENSION

Number of coordinates used in the ST_Geometry values, usually corresponding to the number of dimensions in the spatial reference system.

SRID

Spatial reference system identifier used for the coordinate geometry in this table. It is a foreign key reference to the SPATIAL_REF_SYS table.

GEOM_TYPE

Geometry type that this column represents. Valid values:

- GEOMETRY
- POINT
- LINESTRING
- POLYGON
- GEOMETRYCOLLECTION
- MULTIPOINT
- MULTILINESTRING
- MULTIPOLYGON
- GEOSEQUENCE

UxMin

Lower left X coordinate of the MBR for the universe that includes all values that this column might hold.

UyMin

Lower left Y coordinate of the MBR for the universe that includes all values that this column might hold.

UxMax

Upper right X coordinate of the MBR for the universe that includes all values that this column might hold.

UyMax

Upper right Y coordinate of the MBR for the universe that includes all values that this column might hold.

Usage Notes

Teradata provides stored procedures to add and drop geometry metadata in the GEOMETRY_COLUMNS table. For details, see [AddGeometryColumn Stored Procedure](#), [AddGeometryColumn_3D Stored Procedure](#), and [DropGeometryColumn Stored Procedure](#).

AddGeometryColumn Stored Procedure

Add geometry metadata to the SYSSPATIAL.GEOMETRY_COLUMNS table for a new ST_Geometry column.

Result Type

VARCHAR(1024)

AddGeometryColumn Syntax

```
[SYSSPATIAL.] AddGeometryColumn (
  catalog_name
  schema_name
  table_name
  geom_column_name
  srid
  geom_type
  UxMin_v
  UyMin_v
  UxMax_v
  UyMax_v
)
```

Syntax Elements

catalog_name

The name of the catalog.

The value can be an empty string (") for Teradata.

schema_name

The name of the database in which the table specified by *table_name* is defined.

table_name

The name of the table that has a column defined as an ST_Geometry type.

geom_column_name

The name of the column in the table specified by *table_name* that is defined as an ST_Geometry type.

srid

The spatial reference system identifier used for the coordinate geometry in the *table_name* table.

geom_type

The geometry type that column specified by *geom_column_name* represents.

Valid values include:

- 'ST_Geometry'
- 'ST_Point'
- 'ST_LineString'
- 'ST_Polygon'
- 'ST_GeometryCollection'
- 'ST_MultiPoint'
- 'ST_MultiLineString'
- 'ST_MultiPolygon'
- 'GeoSequence'

UxMin_v

The lower left X coordinate of the MBR of the universe for all values in the new ST_Geometry column.

UyMin_v

The lower left Y coordinate of the MBR of the universe for all values in the new ST_Geometry column.

UxMax_v

The upper right X coordinate of the MBR of the universe for all values in the new ST_Geometry column.

UyMax_v

The upper right Y coordinate of the MBR of the universe for all values in the new ST_Geometry column.

Usage Notes

You can optionally use this stored procedure to add metadata to the GEOMETRY_COLUMNS table in the SYSSPATIAL database when you create a table that has a column defined as an ST_Geometry type.

Stored procedures are invoked using the CALL statement. For more information, see *Teradata Vantage™ - SQL Data Manipulation Language*, B035-1146.

Example: AddGeometryColumn

```
CREATE TABLE lakes (
  fid INTEGER NOT NULL,
  name VARCHAR(64),
  shore ST_Geometry ) UNIQUE PRIMARY INDEX (fid);

CALL SYSSPATIAL.AddGeometryColumn('', 'OpenGISU', 'lakes', 'shore', 988,
  'ST_Polygon', 0.0, 1.0, 1000.0, 1001.0, s );

resultString
-----
shore in table OpenGISU.lakes was successfully added.

CREATE TABLE road_segments (
  fid INTEGER NOT NULL,
  name VARCHAR(64),
  centerline ST_Geometry ) UNIQUE PRIMARY INDEX (fid);

CALL SYSSPATIAL.AddGeometryColumn( '', 'OpenGISU', 'road_segments',
  'centerline', 988, 'ST_LineString', 0E0, 1E0, 1E3, 1.001E3, s );

resultString
```

```
-----
centerline in table OpenGISU.road_segments was successfully added.
```

AddGeometryColumn_3D Stored Procedure

Add geometry metadata to the SYSSPATIAL.GEOMETRY_COLUMNS table for a new ST_Geometry column, and specifies the number of coordinates used in the ST_Geometry values.

Result Type

VARCHAR(1024)

AddGeometryColumn_3D Syntax

```
[SYSSPATIAL.] AddGeometryColumn_3D (
    catalog_name
    schema_name
    table_name
    geom_column_name
    dimensions
    srid
    geom_type
    UxMin_v
    UyMin_v
    UxMax_v
    UyMax_v
)
```

Syntax Elements

catalog_name

The name of the catalog.

The value can be an empty string ("") for Teradata.

schema_name

The name of the database in which the table specified by *table_name* is defined.

table_name

The name of the table that has a column defined as an ST_Geometry type.

geom_column_name

The name of the column in the table specified by *table_name* that is defined as an ST_Geometry type.

dimensions

The coordinate dimension. This is the number of coordinates used in the ST_Geometry values, usually corresponding to the number of dimensions in the spatial reference system.

srid

The spatial reference system identifier used for the coordinate geometry in the *table_name* table.

geom_type

The geometry type that column specified by *geom_column_name* represents.

Valid values include:

- 'ST_Geometry'
- 'ST_Point'
- 'ST_LineString'
- 'ST_Polygon'
- 'ST_GeometryCollection'
- 'ST_MultiPoint'
- 'ST_MultiLineString'
- 'ST_MultiPolygon'
- 'GeoSequence'

UxMin_v

The lower left X coordinate of the MBR of the universe for all values in the new ST_Geometry column.

UyMin_v

The lower left Y coordinate of the MBR of the universe for all values in the new ST_Geometry column.

UxMax_v

The upper right X coordinate of the MBR of the universe for all values in the new ST_Geometry column.

UyMax_v

The upper right Y coordinate of the MBR of the universe for all values in the new ST_Geometry column.

Usage Notes

You can optionally use this stored procedure to add metadata to the GEOMETRY_COLUMNS table in the SYSSPATIAL database when you create a table that has a column defined as an ST_Geometry type.

Stored procedures are invoked using the CALL statement. For more information, see *Teradata Vantage™ - SQL Data Manipulation Language*, B035-1146.

Example: AddGeometryColumn_3D

```
CREATE TABLE lakes (
  fid INTEGER NOT NULL,
  name VARCHAR(64),
  shore ST_Geometry ) UNIQUE PRIMARY INDEX (fid);

CALL SYSSPATIAL.AddGeometryColumn_3D('','OpenGISU','lakes','shore',3,
  988,'ST_Polygon', 0.0, 1.0, 1000.0, 1001.0, s );

resultString
-----
shore in table OpenGISU.lakes was successfully added.

CREATE TABLE road_segments (
  fid INTEGER NOT NULL,
  name VARCHAR(64),
  centerline ST_Geometry ) UNIQUE PRIMARY INDEX (fid);

CALL SYSSPATIAL.AddGeometryColumn_3D( '', 'OpenGISU', 'road_segments',
  'centerline', 3, 988, 'ST_LineString', 0E0, 1E0, 1E3, 1.001E3, s );

resultString
-----
centerline in table OpenGISU.road_segments was successfully added.
```

DropGeometryColumn Stored Procedure

Drop geometry metadata from the SYSSPATIAL.GEOMETRY_COLUMNS table for an ST_Geometry column

Result Type

VARCHAR(1024)

DropGeometryColumn Syntax

```
[SYSSPATIAL.] DropGeometryColumn (
  catalog_name
  schema_name
  table_name
  geom_column_name
)
```

Syntax Elements***catalog_name***

The name of the catalog.

The value can be an empty string (") for Teradata.

schema_name

The name of the database in which the table specified by *table_name* is defined.

table_name

The name of the table that has a column defined as an ST_Geometry type.

geom_column_name

The name of the column in the table specified by *table_name* that is defined as an ST_Geometry type.

Usage Notes

You can optionally use this stored procedure to drop metadata from the SYSSPATIAL.GEOMETRY_COLUMNS table when you drop or alter a table that has a column defined as an ST_Geometry type.

Stored procedures are invoked using the CALL statement. For more information, see *Teradata Vantage™ - SQL Data Manipulation Language*, B035-1146.

Example: DropGeometryColumn

The following examples correspond to the examples used for [AddGeometryColumn Stored Procedure](#).

```
CALL SYSSPATIAL.DropGeometryColumn('', 'OpenGISU', 'lakes', 'shore', s);

CALL SYSSPATIAL.DropGeometryColumn('', 'OpenGISU', 'road_segments',
    'centerline', s );
```

SPATIAL_REF_SYS Table

Contains information about each spatial reference system.

Syntax

```
CREATE TABLE SYSSPATIAL.SPATIAL_REF_SYS, FALLBACK (
    SRID INTEGER NOT NULL PRIMARY KEY,
    AUTH_NAME CHARACTER VARYING(256) CHARACTER SET LATIN,
    AUTH_SRID INTEGER,
    SRTEXT CHARACTER VARYING(2048) CHARACTER SET LATIN)
PRIMARY INDEX (AUTH_SRID);
```

Syntax Elements

SRID

Spatial reference system identifier.

AUTH_NAME

Name of the standard or standards body that is being cited for this reference system.

AUTH_SRID

Identifier of the spatial reference as defined by the authority cited in AUTH_NAME.

The table specifies a nonunique primary index (NUPI) on this column.

SRTEXT

WKT representation of a geographic (latitude-longitude), a projected (X, Y), or a geocentric (X, Y, Z) coordinate system.

The coordinate system is composed of several items, where each item has a keyword in uppercase followed by the defining, comma-delimited, parameters of the item in brackets.

The list of keywords is:

- DATUM
- GEOCCS
- GEOGCS
- PROJCS

- PARAMETER
- PRIMEM
- PROJECTION
- SPHEROID
- UNIT

Notation Conventions

Mathematical Set Operation Notation

The following table defines mathematical set operation notation used in this document.

Symbol	Definition
\emptyset	Empty set
\cap	Mathematical set intersection
\cup	Mathematical set union
$-$	Mathematical set difference

The Dimensionally Extended 9 Intersection Model

A large number of spatial relationships between two ST_Geometry values can be based on testing for intersections between the interior, boundary, and exterior of the two values. For example, two ST_Geometry values are disjoint if neither the interiors nor the boundaries of either value intersect.

The intersections of any interior, boundary, or exterior of two ST_Geometry values can result in a set of ST_Geometry values of mixed dimension. For example, the intersection of the boundaries of two ST_Polygon values may consist of an ST_Point value and an ST_LineString value.

SQL/MM Spatial uses a dimensionally extended nine intersection model (DE-9IM) that expresses spatial relationships among ST_Geometry values as pair-wise intersections of their interior, boundary, and exterior with consideration for the dimension of the resulting intersections.

In the following 3-by-3 matrix, $I(a)$, $B(a)$, and $E(a)$ represent the interior, boundary, and exterior of ST_Geometry value a , and $I(b)$, $B(b)$, and $E(b)$ represent the interior, boundary, and exterior of ST_Geometry value b . The ST_Dimension method returns a value of -1, 0, 1, or 2, where a value of -1 corresponds to the dimension of the empty set (\emptyset).

	Interior	Boundary	Exterior
Interior	$(I(a) \cap I(b)).ST_Dimension$	$(I(a) \cap B(b)).ST_Dimension$	$(I(a) \cap E(b)).ST_Dimension$
Boundary	$(B(a) \cap I(b)).ST_Dimension$	$(B(a) \cap B(b)).ST_Dimension$	$(B(a) \cap E(b)).ST_Dimension$
Exterior	$(E(a) \cap I(b)).ST_Dimension$	$(E(a) \cap B(b)).ST_Dimension$	$(E(a) \cap E(b)).ST_Dimension$

How to Read Syntax

This document uses the following syntax conventions.

Syntax Convention	Meaning
KEYWORD	Keyword. Spell exactly as shown. Many environments are case-insensitive. Syntax shows keywords in uppercase unless operating system restrictions require them to be lowercase or mixed-case.
<i>variable</i>	Variable. Replace with actual value.
<i>number</i>	String of one or more digits. Do not use commas in numbers with more than three digits. Example: 10045
[x]	x is optional.
[x y]	You can specify x, y, or nothing.
{ x y }	You must specify either x or y.
x [...]	You can repeat x, separating occurrences with spaces. Example: x x x See note after table.
x [, ...]	You can repeat x, separating occurrences with commas. Example: x, x, x See note after table.
x [<i>delimiter</i> ...]	You can repeat x, separating occurrences with specified delimiter. Examples: <ul style="list-style-type: none"> If <i>delimiter</i> is semicolon: x; x; x If <i>delimiter</i> is { , OR }, you can do either of the following: <ul style="list-style-type: none"> x, x, x x OR x OR x See note after table.

Note:

You can repeat only the immediately preceding item. For example, if the syntax is:

```
KEYWORD x [...]
```

You can repeat x. Do not repeat KEYWORD.

If there is no white space between x and the delimiter, the repeatable item is x and the delimiter. For example, if the syntax is:

```
[ x, [...] ] y
```

- You can omit x: y
 - You can specify x once: x, y
 - You can repeat x and the delimiter: x, x, x, y
-

Tessellation

Tessellation and tessellation functions can improve database performance with geospatial data when geospatial indexes are not available. With the introduction of geospatial indexing in , tessellation usually is not necessary. Although Teradata geospatial indexing implements a geospatial nested join capability, there are other types of geospatial joins which may be better suited for indexing certain classes of geospatial objects. (For more information on geospatial nested joins, see [Geospatial Predicates and the Optimizer](#).) For this reason, there may be times when tessellation may yield better performance for queries involving geospatial data, rather than using a geospatial index driven nested join. Tessellation can also be used when a geospatial index is not available.

Tessellation remains available for instances when tessellation is a better choice than using a geospatial index. This section describes the tessellation functions and methods that are still available in Vantage.

The SELECT request shown in [Methods Specific to the Subtype the ST_Geometry Type Represents](#) specified a join condition of `streetShape.ST_Within(cityShape) = 1` to find all of the streets within a table of cities. To analyze the join without the aid of geospatial indexes, Vantage needs to check every street against every city, which is a cross product. This can be costly for geospatial types, especially since they can be complex and irregular in shape. A lot of comparisons will be done even if the geospatial objects are not close to each other.

Tessellation is a technique where the object universe is described as a grid with a specified grid size. The tiles within the grid are referred to as cells. Grids can be multilevel, with each level a finer granularity.

Here is an example of a single-level grid (of unspecified size) that contains 16 cells.

12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3

To use tessellation, the geospatial objects (for example, streets and cities) of the join are converted into minimum bounding rectangles (MBRs) and placed within this grid (multilevel if desired). During a join operation, instead of comparing every street to every city, only the streets and cities that lie within the same

grid cells are compared. The query can use a tessellation method on the geospatial objects to return the cell ids and grid level where the geospatial objects lie.

The sections that follow describe the tessellation functions and methods available in the database.

Tessellate_Index Method

Returns the smallest cell that contains this object. The Tessellate_Index method is based on the concept of a multiple level g_nx by g_ny grid. The return value is suitable to use as an index key.

Valid Data Types

All ST_Geometry types

Tessellate_Index Syntax

```
Tessellate_Index (
  u_xmin, u_ymin, u_xmax, u_ymax,
  g_nx, g_ny, levels, scale, shift
)
```

Syntax Elements

u_xmin

The coordinates of the universe of interest.

The data type is FLOAT.

u_ymin

The coordinates of the universe of interest.

The data type is FLOAT.

u_xmax,

The coordinates of the universe of interest.

The data type is FLOAT.

u_ymax

The coordinates of the universe of interest.

The data type is FLOAT.

g_nx

The number of grid cells to divide the universe into in the X dimensions.

The data type is INTEGER.

g_ny

The number of grid cells to divide the universe into in the Y dimensions.

The data type is INTEGER.

levels

The number of levels in the two-dimensional grid. There are always *levels* + 1 levels, where level 0 is the entire universe. Higher levels are more granular.

The data type is INTEGER and the range of values is 1 to 15.

scale

The scaling factor between grid levels. For example, if *g_nx* * *g_ny* is 100x100 and level is 2 and scale is 0.1, you have a 3 level grid (100x100, 10x10, 1x1).

The data type of *scale* is FLOAT and the value must be greater than 0.0 and less than 1.0.

shift

An INTEGER value that represents the number of times to shift the grid at each level. If the value of *shift* is 0, the method performs no shifting. If the value of *shift* is 1, the method creates four grids for each level, where it shifts each of the four grids in a unique manner.

Shifting the grid helps to avoid assigning smaller spatial objects to large cells if they cross a grid cell boundary.

Result Type

The cell ID that is returned is an INTEGER value that codes for the cell number and level. The upper 28 bits identifies the cell number, and the lower four bits identifies the level.

Cell ID:

Cell Number	Level
-------------	-------

<----- 28 bits -----><-- 4 bits -->

To derive the cell number from the returned cell ID value using SQL, use:

```
return_value / 16
```

To derive the level from the returned cell ID using SQL, use:

```
return_value MOD 16
```

Here is an example that shows the cell numbering of a single-level grid (of unspecified size) that contains 16 cells.

12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3

Example: Tessellate_Index Method

```
CREATE TABLE sample_shapes(skey INTEGER, shape ST_Geometry);

INSERT INTO sample_shapes
  values(1067, NEW ST_Geometry('LineString(-43.5 20.2, -43.6 20.3)'));

CREATE TABLE shapes_index
  (skey INTEGER
   ,cellid INTEGER);

INSERT INTO shapes_index
SELECT skey,
       shape.Tessellate_Index(-180, 0, 0, 90, 500, 500, 1, 0.01, 0)
FROM sample_shapes;
```

Tessellate Function

Table function that returns the set of cell IDs that intersect an object rectangle. The function is based on the concept of a single level g_{nx} by g_{ny} grid. The return value is suitable to use as an equijoin bind term.

Tessellate Syntax

```
Tessellate (
  in_key,
  o_xmin, o_ymin, o_xmax, o_ymax
  u_xmin, u_ymin, u_xmax, u_ymax,
  g_nx, g_ny
)
```

Syntax Elements

in_key

A key that is passed back in the result rows to allow joining back to the proper object.

The data type of *in_key* is DECIMAL(18,0) or VARCHAR(32). The data type of the argument you pass in determines the data type of the *out_key* column in the result row.

o_xmin, o_ymin, o_xmax, o_ymax

The lower left and upper right coordinates of the object rectangle.

The data type of *o_xmin*, *o_ymin*, *o_xmax*, and *o_ymax* is FLOAT.

u_xmin, u_ymin, u_xmax, u_ymax

The lower left and upper right coordinates of the universe of interest.

The data type of *u_xmin*, *u_ymin*, *u_xmax*, and *u_ymax* is FLOAT.

g_nx, g_ny

The number of grid cells to divide the universe into in the X and Y dimensions.

The data type of *g_nx* and *g_ny* is INTEGER.

Argument Types and Rules

If any input argument is NULL, Tessellate returns an error.

For details on the rules for argument data types, see [Embedded Services System Functions](#).

Result Rows

The columns in the table that Tessellate returns are as follows.

Column Name	Data Type	Description																
out_key	DECIMAL(18,0)	A copy of the in_key input argument.																
	VARCHAR(32)	The data type of out_key matches the data type of the in_key argument.																
cellid	INTEGER	<p>Cell ID that intersects the object rectangle.</p> <p>Cell IDs are numbered from left to right, 0 to $n-1$, where $n = g_nx * g_ny$. Cell 0 is in the lower left corner. Here is an example that shows the cell numbering of a single-level grid (of unspecified size) that contains 16 cells.</p> <table><tr><td>12</td><td>13</td><td>14</td><td>15</td></tr><tr><td>8</td><td>9</td><td>10</td><td>11</td></tr><tr><td>4</td><td>5</td><td>6</td><td>7</td></tr><tr><td>0</td><td>1</td><td>2</td><td>3</td></tr></table>	12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
12	13	14	15															
8	9	10	11															
4	5	6	7															
0	1	2	3															

Example: Tessellate Function

Consider two tables, P500000 and R5000, each of which contains a column that is a geospatial shape (column A in table P500000 and column B in table R5000).

For simplicity, the two tables have MBRs as their shapes, but any type of shape can be used by passing the MBR of the shape to the tessellation function and then doing the overlap comparison between the two shapes via the ST_Overlaps method in the final WHERE clause.

The following statement counts the number of objects that overlap between the two tables.

```

SELECT COUNT(*)
FROM ( SELECT *
      FROM P500000 A
        ,TABLE(tessellate(A.rkey, A.xmin, A.ymin, A.xmax, A.ymax,
                        0, 0, 100, 100, 100, 100 ) )T1
      WHERE A.rkey = T1.out_key ) S1
, ( SELECT *
    FROM R5000 B

```

```

        ,TABLE( Tessellate(B.rkey, B.xmin, B.ymin, B.xmax, B.ymax,
                           0, 0, 100, 100, 100, 100 ) )T2
    WHERE B.rkey = T2.out_key )S2
WHERE S1.cellid = S2.cellid
    AND S1.xmax >= S2.xmin and S1.xmin <= S2.xmax
    AND S1.ymax >= S2.ymin and S1.ymin <= S2.ymax;

```

Here is how the tessellation works in the preceding example.

1. We define a tessellation grid that is large enough to contain the universe of shapes within our two tables and we define a granularity that is reasonable given the typical size of a spatial object. In this case, the grid is 100 by 100 in size as defined by a universe MBR of 0,0 to 100,100 and the number of grid cells in both the x and y direction is defined to be 100.
2. In the first SELECT statement, a row at a time is selected from P500000. For each row, the tessellate table function is called to return a table of cell ids that contain the spatial object. This produces a table (S1) of the spatial objects and the cells that contain them. For example, if a row from P500000 is processed and the spatial object is in four cells, four rows are added to S1, each with the same spatial object and a cell id. This continues for all rows in P500000.
3. The second SELECT statement is processed on table R5000 in the same way as the first select statement with the results going into S2.
4. Now there are two tables, S1 and S2. Next, the final WHERE clause does a join between S1 and S2 on the cell id and the overlaps computation, which is simple for an MBR shape. This produces a row if the two shapes overlap. If we assume that the overlaps computation can be expensive, the performance gain comes from the fact that only spatial objects that are in the same cell are compared via the overlaps computation. The overlaps computation is never executed for objects that are not physically close to one another.

Tessellate_Search Function

Table function that returns all cells at each level of the grid that contain the object passed in. The function is based on the concept of a multilevel g_nx by g_ny grid. The returned cell IDs are suitable for using as search keys for the tessellate_index generated key.

Tessellate_Search Syntax

```

Tessellate_Search (
    in_key,
    o_xmin, o_ymin, o_xmax, o_ymax
    u_xmin, u_ymin, u_xmax, u_ymax,
    g_nx, g_ny, levels, scale, shift
)

```

Syntax Elements

in_key

A key that is passed back in the result rows to allow joining back to the proper object.

The data type of *in_key* is DECIMAL(18,0) or VARCHAR(32). The data type of the argument you pass in determines the data type of the *out_key* column in the result row.

o_xmin, o_ymin, o_xmax, o_ymax

The coordinates of the object rectangle.

The data type of *o_xmin*, *o_ymin*, *o_xmax*, and *o_ymax* is FLOAT.

u_xmin, u_ymin, u_xmax, u_ymax

The coordinates of the universe of interest.

The data type of *u_xmin*, *u_ymin*, *u_xmax*, and *u_ymax* is FLOAT.

g_nx, g_ny

The number of grid cells to divide the universe into in the X and Y dimensions. The maximum number of grid cells is 2^{28} .

The data type of *g_nx* and *g_ny* is INTEGER.

If *g_nx* or *g_ny* is less than or equal to zero, Tessellate_Search returns an error.

levels

The number of levels in the two-dimensional grid. There are always *levels* + 1 levels, where level 0 is the entire universe. Higher levels are more granular.

The data type of *levels* is INTEGER and the range of values is 1 to 15.

scale

The scaling factor between grid levels. For example, if *g_nx* * *g_ny* is 100x100 and level is 2 and scale is 0.1, you have a 3 level grid (100x100, 10x10, 1x1).

The data type of *scale* is FLOAT.

The value of *scale* must be greater than 0.0 and less than 1.0.

shift

A numeric value that represents the number of times to shift the grid at each level. If the value of *shift* is 0, the method performs no shifting. If the value of *shift* is 1, the method creates four grids for each level, where it shifts each of the four grids in a unique manner.

Shifting the grid helps to avoid assigning smaller spatial objects to large cells if they cross a grid cell boundary.

The data type of *shift* is INTEGER.

Argument Types and Rules

If any input argument is NULL, Tessellate_Search returns an error.

For details on the rules for argument data types, see [Embedded Services System Functions](#).

Result Rows

The columns in the table that Tessellate_Search returns are as follows.

Column Name	Data Type	Description
<i>out_key</i>	DECIMAL(18,0)	A copy of the <i>in_key</i> input argument. The data type of <i>out_key</i> matches the data type of the <i>in_key</i> argument.
	VARCHAR(32)	
<i>cellid</i>	INTEGER	<p>The INTEGER value that is returned codes for the cell number and level. The upper 28 bits identifies the cell number, and the lower four bits identifies the level. Use:</p> $cellid / 16$ <p>to derive the cell number and:</p> $cellid \text{ MOD } 16$ <p>to derive the level.</p> <p>Cell IDs are numbered from left to right, 0 to $n-1$, where $n = g_{nx} * g_{ny}$. Cell 0 is in the lower left corner. Here is an example that shows the cell numbering of a single-level grid (of unspecified size) that contains 16 cells.</p>

Column Name	Data Type	Description																
		<table><tr><td>12</td><td>13</td><td>14</td><td>15</td></tr><tr><td>8</td><td>9</td><td>10</td><td>11</td></tr><tr><td>4</td><td>5</td><td>6</td><td>7</td></tr><tr><td>0</td><td>1</td><td>2</td><td>3</td></tr></table>	12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
12	13	14	15															
8	9	10	11															
4	5	6	7															
0	1	2	3															

Example: Tessellate Search Function

```
CREATE TABLE sample_cities(
  skey INTEGER, cityName VARCHAR(200), cityShape ST_Geometry);

INSERT INTO sample_cities values(
  1067, New ST_Geometry('Polygon(
    (-43.5 20.2, -43.5 21.5, -40.5 21.5, -40.5 20.2, -43.5 20.2))'));
```

Consider the data in the cities_index table that is inserted using the Tessellate_Index method:

```
CREATE TABLE cities_index
  (skey INTEGER
   ,cellid INTEGER);

INSERT INTO cities_index
SELECT skey,
  cityShape.Tessellate_Index(-180, 0, 0, 90, 500, 500, 1, 0.01, 0)
FROM sample_cities;
```

The following statement uses the Tessellate_Search function with the same universe coordinates and grid specifications that were used by Tessellate_Index:

```
SELECT c.skey, c.cityName, s.streetName, s.streetShape
FROM sample_cities c
```

```

,cities_index ci
,(SELECT streetName, skey, streetShape ,streetShape.ST_MBR_Xmin()
    ,streetShape.ST_MBR_Ymin(), streetShape.ST_MBR_Xmax()
    ,streetShape.ST_MBR_Ymax()
 FROM sample_streets)
 AS s (streetName, skey, streetShape, xmin, ymin, xmax, ymax)
,TABLE ( Tessellate_Search (
    s.skey
    ,s.xmin, s.ymin, s.xmax, s.ymax
    ,-180, 0, 0, 90
    ,500,500
    ,1, 0.01, 0)) AS t
WHERE c.skey = ci.skey
AND ci.cellid = t.cellid
AND t.out_key = s.skey;

```

Tessellate UDF

Table function that returns the set of cell IDs that intersect an object rectangle. The function is based on the concept of a single level g_nx by g_ny grid. The return value is suitable to use as an equijoin bind term.

Tessellate UDF Syntax

```

SYSSPATIAL.Tessellate (
    in_key,
    o_xmin, o_ymin, o_xmax, o_ymax
    u_xmin, u_ymin, u_xmax, u_ymax,
    g_nx, g_ny
)

```

Syntax Elements

in_key

A DECIMAL(18,0) value for a key that is passed back in the result rows to allow joining back to the proper object.

o_xmin, o_ymin, o_xmax, o_ymax

The coordinates of the object rectangle.

The data type of *o_xmin*, *o_ymin*, *o_xmax*, and *o_ymax* is FLOAT.

u_xmin, u_ymin, u_xmax, u_ymax

The coordinates of the universe of interest.

The data type of *u_xmin*, *u_ymin*, *u_xmax*, and *u_ymax* is FLOAT.

g_nx, g_ny

The number of grid cells to divide the universe into in the X and Y dimensions.

The data type of *g_nx* and *g_ny* is INTEGER.

Result Rows

The columns in the table that Tessellate returns are as follows.

Column Name	Data Type	Description
<i>out_key</i>	DECIMAL(18,0)	A copy of the <i>in_key</i> input argument.
<i>cellid</i>	INTEGER	Cell ID that intersects the object rectangle.

Here is an example that shows the cell numbering of a single-level grid (of unspecified size) that contains 16 cells.

12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3

Example: tessellate UDF

Consider two tables, P500000 and R5000, each of which contains a column that is a geospatial shape (column A in table P500000 and column B in table R5000).

For simplicity, the two tables have MBRs as their shapes, but any type of shape can be used by passing the MBR of the shape to the tessellation function and then doing the overlap comparison between the two shapes via the ST_Overlaps method in the final WHERE clause.

The following statement counts the number of objects that overlap between the two tables.

```

SELECT COUNT(*)
FROM ( SELECT *
      FROM P500000 A
      ,TABLE(SYSSPATIAL.Tessellate(A.rkey, A.xmin, A.ymin,
                                   A.xmax, A.ymax, 0, 0, 100, 100, 100, 100 ) )T1
      WHERE A.rkey = T1.out_key ) S1
, ( SELECT *
    FROM R5000 B
    ,TABLE(SYSSPATIAL.Tessellate(B.rkey, B.xmin, B.ymin,
                                   B.xmax, B.ymax, 0, 0, 100, 100, 100, 100 ) )T2
    WHERE B.rkey = T2.out_key )S2
WHERE S1.cellid = S2.cellid
AND S1.xmax >= S2.xmin and S1.xmin <= S2.xmax
AND S1.ymax >= S2.ymin and S1.ymin <= S2.ymax;

```

Here is how the tessellation works in the preceding example.

- We define a tessellation grid that is large enough to contain the universe of shapes within our two tables and we define a granularity that is reasonable given the typical size of a spatial object. In this case, the grid is 100x100 in size as defined by a universe MBR of 0,0 to 100,100 and the “number of grids” in both the x and y direction is defined to be 100.
- In the first SELECT statement, a row at a time is selected from P500000. For each row, the tessellate table function is called to return a table of cell ids that contain the spatial object. This produces a table (S1) of the spatial objects and the cells that contain them. For example, if a row from P500000 is processed and the spatial object is in four cells, four rows are added to S1, each with the same spatial object and a cell id. This continues for all rows in P500000.
- The second SELECT statement is processed on table R5000 in the same way as the first select statement with the results going into S2.
- Now there are two tables, S1 and S2. Next, the final WHERE clause does a join between S1 and S2 on the cell id and the overlaps computation, which is simple for an MBR shape. This produces a row if the two shapes overlap. If we assume that the overlaps computation can be expensive, the performance gain comes from the fact that only spatial objects that are in the same cell are compared via the overlaps computation. The overlaps computation is never executed for objects that are not physically close to one another.

Tessellate_Index UDF

Returns the smallest cell that contains the object passed in. The Tessellate_Index function is based on the concept of a multiple level g_nx by g_ny grid. The return value is suitable to use as an index key.

Tessellate_Index UDF Syntax

```
SYSSPATIAL.Tessellate_Index (
  o_xmin, o_ymin, o_xmax, o_ymax
  u_xmin, u_ymin, u_xmax, u_ymax,
  g_nx, g_ny, levels, scale, shift
)
```

Syntax Elements

o_xmin, o_ymin, o_xmax, o_ymax

The coordinates of the object rectangle.

The data type of *o_xmin*, *o_ymin*, *o_xmax*, and *o_ymax* is FLOAT.

u_xmin, u_ymin, u_xmax, u_ymax

The coordinates of the universe of interest.

The data type of *u_xmin*, *u_ymin*, *u_xmax*, and *u_ymax* is FLOAT.

g_nx, g_ny

The number of grid cells to divide the universe into in the X and Y dimensions.

The data type of *g_nx* and *g_ny* is INTEGER.

levels

The number of levels in the two-dimensional grid. There are always *levels* + 1 levels, where level 0 is the entire universe. Higher levels are more granular.

The data type is INTEGER and the range of values is 1 to 15.

scale

The scaling factor between grid levels. For example, if *g_nx* * *g_ny* is 100x100 and level is 2 and scale is 0.1, you have a 3 level grid (100x100, 10x10, 1x1).

The data type of *scale* is FLOAT and the value must be greater than 0.0 and less than 1.0.

shift

An INTEGER value that represents the number of times to shift the grid at each level. If the value of *shift* is 0, the method performs no shifting. If the value of *shift* is 1, the method creates four grids for each level, where it shifts each of the four grids in a unique manner.

Shifting the grid helps to avoid assigning smaller spatial objects to large cells if they cross a grid cell boundary.

Return Type

The cell ID that is returned is an INTEGER value that codes for the cell number and level. The upper 28 bits identifies the cell number, and the lower four bits identifies the level.

Cell ID:

Cell Number	Level
-------------	-------

<----- 28 bits -----><-- 4 bits -->

To derive the cell number from the returned cell ID value using SQL, use:

```
return_value / 16
```

To derive the level from the returned cell ID using SQL, use:

```
return_value MOD 16
```

Here is an example that shows the cell numbering of a single-level grid (of unspecified size) that contains 16 cells.

12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3

Usage Notes

This function is also available as an ST_Geometry method, which offers better performance.

Example: Tessellate_Index UDF

```
CREATE TABLE shapes_index
  (skey INTEGER
   ,cellID INTEGER);

INSERT INTO shapes_index
SELECT SYSSPATIAL.Tessellate_Index(shape.ST_MBR_Xmin(),
  shape.ST_MBR_Ymin(), shape.ST_MBR_Xmax(), shape.ST_MBR_Ymax(),
  -180, 0, 0, 90, 500, 500, 1, 0.01, 0)
FROM sample_shapes;
```

Tessellate_Search UDF

Table function that returns all cells at each level of the grid that contain the object passed in. The function is based on the concept of a multilevel g_nx by g_ny grid. The returned cell IDs are suitable for using as search keys for the Tessellate_Index generated key.

Tessellate_Search UDF Syntax

```
SYSSPATIAL.Tessellate_Search (
  in_key,
  o_xmin, o_ymin, o_xmax, o_ymax
  u_xmin, u_ymin, u_xmax, u_ymax,
  g_nx, g_ny, levels, scale, shift
)
```

Syntax Elements

in_key

A DECIMAL(18,0) for a key that is passed back in the result rows to allow joining back to the proper object.

o_xmin, o_ymin, o_xmax, o_ymax

The coordinates of the object rectangle.

The data type of *o_xmin*, *o_ymin*, *o_xmax*, and *o_ymax* is FLOAT.

u_xmin, u_ymin, u_xmax, u_ymax

The coordinates of the universe of interest.

The data type of *u_xmin*, *u_ymin*, *u_xmax*, and *u_ymax* is FLOAT.

g_nx, g_ny

The number of grid cells to divide the universe into in the X and Y dimensions.

The data type of *g_nx* and *g_ny* is INTEGER.

levels

An INTEGER value for the number of levels in the two-dimensional grid. There are always *levels* + 1 levels, where level 0 is the entire universe. Higher levels are more granular.

The range of values is 1 to 15.

scale

The scaling factor between grid levels. For example, if *g_nx* * *g_ny* is 100x100 and level is 2 and scale is 0.1, you have a 3 level grid (100x100, 10x10, 1x1).

The value of scale must be greater than 0.0 and less than 1.0.

The data type of *scale* is FLOAT.

shift

An INTEGER value that represents the number of times to shift the grid at each level. If the value of *shift* is 0, the method performs no shifting. If the value of *shift* is 1, the method creates four grids for each level, where it shifts each of the four grids in a unique manner.

Shifting the grid helps to avoid assigning smaller spatial objects to large cells if they cross a grid cell boundary.

Result Rows

The columns in the table that `Tessellate_Search` returns are as follows.

Column Name	Data Type	Description
<i>out_key</i>	DECIMAL(18,0)	A copy of the <i>in_key</i> input argument.
<i>cellID</i>	INTEGER	<p>The INTEGER value that is returned codes for the cell number and level. The upper 28 bits identifies the cell number, and the lower four bits identifies the level. Use:</p> $cellid / 16$ <p>to derive the cell number and:</p> $cellid \text{ MOD } 16$

Column Name	Data Type	Description
		to derive the level.

Here is an example that shows the cell numbering of a single-level grid (of unspecified size) that contains 16 cells.

12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3

Example: Tessellate_Search UDF

Consider the data in the cities_index table that is inserted using the Tessellate_Index method:

```
CREATE TABLE cities_index
(skey INTEGER
,cellid INTEGER);

INSERT INTO cities_index
SELECT skey,
       cityShape.Tessellate_Index(-180, 0, 0, 90, 500, 500, 1, 0.01, 0)
FROM sample_cities;
```

The following statement uses the Tessellate_Search function with the same universe coordinates and grid specifications that were used by tessellate_index:

```
SELECT c.skey, c.cityName, s.streetName, s.streetShape
FROM sample_cities c
     ,cities_index ci
     ,(SELECT streetName, skey, streetShape ,streetShape.ST_MBR_Xmin()
        ,streetShape.ST_MBR_Ymin(), streetShape.ST_MBR_Xmax()
```

```
        ,streetShape.ST_MBR_Ymax()  
FROM sample_streets)  
AS s (streetName, skey, streetShape, xmin, ymin, xmax, ymax)  
,TABLE (SYSSPATIAL.Tessellate_Search (  
        s.skey  
        ,s.xmin, s.ymin, s.xmax, s.ymax  
        ,-180, 0, 0, 90  
        ,500,500  
        ,1, 0.01, 0)) AS t  
WHERE c.skey = ci.skey  
AND ci.cellid = t.cellid  
AND t.out_key = s.skey;
```

Additional Information

Teradata Links

Link	Description
https://docs.teradata.com/	Search Teradata Documentation, customize content to your needs, and download PDFs. Customers: Log in to access Orange Books.
https://support.teradata.com	One-stop source for Teradata community support, software downloads, and product information. Log in for customer access to: <ul style="list-style-type: none">• Community support• Software updates• Knowledge articles
https://www.teradata.com/University/Overview	Teradata education network
https://support.teradata.com/community	Link to Teradata community