

SAP HANA Platform 2.0 SPS 00  
Document Version: 1.0 – 2016-11-30

# SAP HANA Spatial Reference



# Content

<b>1</b>	<b>Introduction.</b>	<b>7</b>
<b>2</b>	<b>Spatial Data.</b>	<b>8</b>
2.1	Spatial Reference Systems (SRS) and Spatial Reference Identifiers (SRID).	9
2.2	Units of Measure.	10
2.3	Support for Spatial Data.	11
	Supported Spatial Data Types and Their Hierarchy.	11
	Supported Import and Export Formats for Spatial Data.	14
	Support for ESRI Shapefiles.	16
	Multidimensional Support.	17
2.4	Recommended Reading on Spatial Topics.	19
2.5	Create Spatial Columns (SQL).	19
2.6	Create a Table with Spatial Columns (SQL).	20
2.7	Spatial Data Type Syntax.	21
2.8	Advanced Spatial Topics.	22
	How Flat-Earth and Round-Earth Representations Work.	22
	How Snap-to-Grid and Tolerance Impact Spatial Calculations.	24
	How Polygon Ring Orientation Works.	28
	How Geometry Interiors, Exteriors, and Boundaries Work.	28
	How Spatial Comparisons Work.	29
	How Spatial Relationships Work.	29
	How Spatial Dimensions Work.	31
2.9	SpatialShapes Table.	32
<b>3</b>	<b>Accessing and Manipulating Spatial Data.</b>	<b>34</b>
3.1	ST_Geometry Type.	34
	ST_AsBinary Method.	36
	ST_AsEWKB Method.	37
	ST_AsEWKT Method.	37
	ST_AsGeoJSON Method.	38
	ST_AsSVG Method.	39
	ST_AsSVGAggr Method.	40
	ST_AsText Method.	41
	ST_AsWKB Method.	42
	ST_AsWKT Method.	43
	ST_Boundary Method.	44
	ST_Buffer Method.	44

ST_Contains Method.	46
ST_ConvexHull Method.	48
ST_ConvexHullAggr Method.	49
ST_CoordDim Method.	50
ST_CoveredBy Method.	52
ST_Covers Method.	53
ST_Crosses Method.	55
ST_Difference Method.	56
ST_Dimension Method.	57
ST_Disjoint Method.	58
ST_Distance Method.	60
ST_Envelope Method.	61
ST_EnvelopeAggr Method.	62
ST_Equals Method.	63
ST_GeomFromEWKB Method.	64
ST_GeomFromEWKT Method.	65
ST_GeomFromText Method.	66
ST_GeomFromWKB Method.	68
ST_GeomFromWKT Method.	69
ST_GeometryType Method.	70
ST_Intersection Method.	71
ST_Intersects Method.	72
ST_IntersectionAggr Method.	73
ST_IntersectsFilter Method.	74
ST_IntersectsRect Method.	74
ST_IntersectsRectPlanar Method.	76
ST_Is3D Method.	77
ST_IsEmpty Method.	78
ST_IsMeasured Method.	79
ST_IsSimple Method.	79
ST_IsValid Method.	80
ST_MMax Method.	81
ST_MMin Method.	81
ST_NumInteriorRing Method.	82
ST_NumInteriorRings Method.	83
ST_OrderingEquals Method.	83
ST_Overlaps Method.	85
ST_Perimeter Method.	87
ST_PointOnSurface Method.	88
ST_Relate Method.	89
ST_SRID Method.	90

ST_SRID(INT) Method.	91
ST_SnapToGrid Method.	92
ST_SymDifference Method.	93
ST_Touches Method.	94
ST_Transform Method.	96
ST_Union Method.	97
ST_UnionAggr Method.	98
ST_Within Method.	99
ST_WithinDistance Method.	100
ST_XMax Method.	102
ST_XMin Method.	103
ST_YMax Method.	104
ST_YMin Method.	105
ST_ZMax Method.	105
ST_ZMin Method.	106
3.2 ST_CircularString Type.	107
ST_CircularString Constructor.	108
3.3 ST_GeometryCollection Type.	111
ST_GeometryCollection Constructor.	112
ST_GeometryN Method.	115
ST_NumGeometries Method.	116
3.4 ST_LineString Type.	116
ST_LineString Constructor.	117
ST_EndPoint Method.	120
ST_IsClosed Method.	121
ST_IsRing Method.	122
ST_Length Method.	123
ST_NumPoints Method.	124
ST_PointN Method.	125
ST_StartPoint Method.	126
3.5 ST_MultiLineString Type.	126
ST_MultiLineString Constructor.	127
ST_IsClosed Method.	130
ST_Length Method.	131
3.6 ST_MultiPoint Type.	132
ST_MultiPoint Constructor.	133
3.7 ST_MultiPolygon Type.	136
ST_MultiPolygon Constructor.	136
ST_Area Method.	140
3.8 ST_Point Type.	141
ST_Point Constructor.	142

ST_M Method . . . . .	145
ST_X Method . . . . .	146
ST_Y Method . . . . .	147
ST_Z Method . . . . .	147
3.9 ST_Polygon Type . . . . .	148
ST_Polygon Constructor . . . . .	149
ST_Area Method . . . . .	152
ST_Centroid Method . . . . .	153
ST_ExteriorRing Method . . . . .	154
ST_InteriorRingN Method . . . . .	154
3.10 List of All Supported Methods . . . . .	155
3.11 List of All Supported Constructors . . . . .	162
3.12 List of Aggregate Methods . . . . .	163
3.13 List of Set Operation Methods . . . . .	164
3.14 List of Static Methods . . . . .	164
3.15 List of Spatial Predicates . . . . .	165
<b>4 Spatial Clustering . . . . .</b>	<b>168</b>
4.1 Introduction to Spatial Clustering . . . . .	168
4.2 Cluster Algorithms . . . . .	169
Grid . . . . .	169
K-Means . . . . .	172
DBSCAN . . . . .	173
4.3 SQL Syntax for Spatial Clustering . . . . .	175
Window Functions . . . . .	175
GROUP CLUSTER BY . . . . .	176
4.4 Methods for Spatial Clustering . . . . .	178
ST_ClusterID Method . . . . .	178
ST_ClusterCentroid Method . . . . .	179
ST_ClusterEnvelope Method . . . . .	180
4.5 Use Cases for Spatial Clustering . . . . .	180
<b>5 Appendix . . . . .</b>	<b>185</b>
5.1 SQL Statements . . . . .	185
CREATE SPATIAL REFERENCE SYSTEM Statement . . . . .	185
CREATE SPATIAL UNIT OF MEASURE Statement . . . . .	192
DROP SPATIAL REFERENCE SYSTEM Statement . . . . .	194
DROP SPATIAL UNIT OF MEASURE Statement . . . . .	194
5.2 Geocoding . . . . .	195
Create Index . . . . .	196
Modify Index . . . . .	198
Drop Index . . . . .	198

---

Maintain the Queue. . . . .	198
System Tables and Monitoring View. . . . .	199
Using SAP HANA Geocode. . . . .	200
Geocode Provider. . . . .	202
User Defined Geocode Provider. . . . .	203
Geocoding Errors and Messages. . . . .	204
5.3 Additional Services for SAP HANA Spatial. . . . .	204
Spatial Map Client, Content Viewer and Geo-Content. . . . .	205
Geospatial Metadata Installer. . . . .	214

# 1 Introduction

Column-oriented data structures and in-memory computing have developed into powerful components of today's enterprise applications. While the focus of these developments has primarily been on analyzing sales data, the potential for using these technologies to analyze geographic information is significant. Support for the processing of spatial data represents a key evolution in SAP HANA.

To deliver vastly improved performance and results in everything from modeling and storage to analysis and presentation of your spatial data, SAP HANA includes a multilayered spatial engine and supports spatial columns, spatial access methods, and spatial reference systems. With these enhanced GIS features, SAP HANA now provides a common database for both your business and spatial data.

SAP HANA spatial is an SAP HANA optional component.

## Caution

SAP HANA server software and tools can be used for several SAP HANA platform and options scenarios as well as the respective capabilities used in these scenarios. The availability of these is based on the available SAP HANA licenses and the SAP HANA landscape, including the type and version of the back-end systems the SAP HANA administration and development tools are connected to. There are several types of licenses available for SAP HANA. Depending on your SAP HANA installation license type, some of the features and tools described in the SAP HANA platform documentation may only be available in the SAP HANA options and capabilities, which may be released independently of an SAP HANA Platform Support Package Stack (SPS). Although various features included in SAP HANA options and capabilities are cited in the SAP HANA platform documentation, each SAP HANA edition governs the options and capabilities available. Based on this, customers do not necessarily have the right to use features included in SAP HANA options and capabilities. For customers to whom these license restrictions apply, the use of features included in SAP HANA options and capabilities in a production system requires purchasing the corresponding software license(s) from SAP. The documentation for the SAP HANA optional components is available in SAP Help Portal at [http://help.sap.com/hana\\_options](http://help.sap.com/hana_options). If you have additional questions about what your particular license provides, or wish to discuss licensing features available in SAP HANA options, please contact your SAP account team representative.

## 2 Spatial Data

**Spatial data** is data that describes the position, shape, and orientation of objects in a defined space. Spatial data is represented as 2D geometries in the form of points, line strings, and polygons. For example, the following image shows the state of Massachusetts, representing the union of polygons representing zip code regions.



Two common operations performed on spatial data are calculating the distance between geometries, and determining the union or intersection of multiple objects. These calculations are performed using predicates such as intersects, contains, and crosses.

The spatial data documentation assumes you already have some familiarity with spatial reference systems and with the spatial data you intend to work with.

The software provides storage and data management features for spatial data, allowing you to store information such as geographic locations, routing information, and shape data.

These information pieces are stored as points and various forms of polygons and lines in columns defined with a corresponding **spatial data type** (such as ST\_Point and ST\_Polygon). You use methods and constructors to access and manipulate the spatial data. The software also provides a set of SQL spatial functions designed for compatibility with other products.

### Example

#### How spatial data might be used

Spatial data support lets application developers' associate spatial information with their data. For example, a table representing companies could store the location of the company as a point, or store the delivery area for the company as a polygon. This could be represented in SQL as (returns 1):

```
SELECT NEW ST_Polygon('Polygon((0 0, 0 1, 1 1, 1 0, 0 0 ))').ST_Contains( NEW  
ST_POINT('Point(0.5 0.5)' ) ) FROM dummy;
```

### **i Note**

ST\_POINT is composed of longitude and latitude:

```
NEW ST_Point( <longitude> , <latitude> )
NEW ST_Point('POINT ( <longitude> <latitude> )')
```

## **2.1 Spatial Reference Systems (SRS) and Spatial Reference Identifiers (SRID)**

In the context of spatial databases, the defined space in which geometries are described is called a **spatial reference system (SRS)**. A spatial reference system defines, at minimum:

- Units of measure of the underlying coordinate system (degrees, meters, and so on)
- Maximum and minimum coordinates (also referred to as the bounds)
- Default linear unit of measure
- Whether the data is planar or spheroid data
- Projection information for transforming the data to other SRSs

Every spatial reference system has an identifier called a **Spatial Reference Identifier (SRID)**. When the database server performs operations like finding out if a geometry touches another geometry, it uses the SRID to look up the spatial reference system definition so that it can perform the calculations properly for that spatial reference system. Each SRID must be unique in a database.

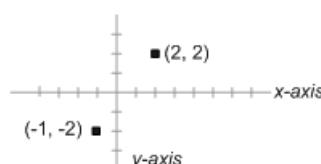
Spatial reference system information is stored in the ST\_SPATIAL\_REFERENCE\_SYSTEMS system view.

By default, the database server adds the following spatial reference systems to a new database:

#### **Default - SRID 0**

This is the default spatial reference system used when constructing a geometry and the SRID is not specified in the SQL and is not present in the value being loaded.

Default is a Cartesian spatial reference system that works with data on a flat, two dimensional plane. Any point on the plane can be defined using a single pair of x, y coordinates where x and y have the bounds -1,000,000 to 1,000,000. Distances are measured using perpendicular coordinate axis. This spatial reference system is assigned SRID of **0**.



Cartesian is a planar type of spatial reference system.

#### **WGS84 - SRID 4326**

The WGS84 standard provides a spheroidal reference surface for the Earth. It is the spatial reference system used by the Global Positioning System (GPS). The

coordinate origin of WGS 84 is the Earth's center, and is considered accurate up to  $\pm 1$  meter. WGS stands for World Geodetic System.

WGS 84 Coordinates are in degrees, where the first coordinate is longitude with bounds -180 to 180, and the second coordinate is latitude with bounds -90 to 90.

The default unit of measure for WGS 84 is METRE, and it is a round-Earth type of spatial reference system.

**WGS 84 (planar) - SRID 1000004326**

WGS 84 (planar) is similar to WGS 84 except that it uses equirectangular projection, which distorts length, area and other computations. For example, at the equator in both WGS 84 and WGS 84 (planar), 1 degree longitude is approximately 111 km. At 80 degrees north, 1 degree of longitude is approximately 19 km in WGS 84, but WGS 84 (planar) treats 1 degree of longitude as approximately 111 km at *all* latitudes. The amount of distortion of lengths in the WGS 84 (planar) is considerable—off by a factor of 10 or more—the distortion factor varies depending on the location of the geometries relative to the equator. Consequently, WGS 84 (planar) should not be used for distance and area calculations. It should only be used for relationship predicates such as ST\_Contains, ST\_Touches, ST\_Covers, and so on.

The default unit of measure for WGS 84 (planar) is planar DEGREE, and it is a flat-Earth type of spatial reference system.

**sa\_planar\_unbounded (unbounded (planar)) - SRID 2147483646**

For internal use only.

## 2.2 Units of Measure

Geographic features can be measured in degrees of latitude, radians, or other angular units of measure. Every spatial reference system must explicitly state the name of the unit in which geographic coordinates are measured, and must include the conversion from the specified unit to a radian.

If you are using a projected coordinate system, the individual coordinate values represent a linear distance along the surface of the Earth to a point. Coordinate values can be measured by the meter, foot, mile, or yard. The projected coordinate system must explicitly state the linear unit of measure in which the coordinate values are expressed.

The following units of measure are automatically installed in any new database:

<b>meter</b>	A linear unit of measure. Also known as International metre. SI standard unit. Defined by ISO 1000.
<b>metre</b>	A linear unit of measure. An alias for meter. SI standard unit. Defined by ISO 1000.
<b>radian</b>	An angular unit of measure. SI standard unit. Defined by ISO 1000:1992.
<b>degree</b>	An angular unit of measure ( $\pi() / 180.0$ radians).

**planar degree** A linear unit of measure. Defined as 60 nautical miles. A linear unit of measure used for geographic spatial reference systems with PLANAR line interpretation.

Unit of measure information is stored in the ST\_UNITS\_OF\_MEASURE system view.

## 2.3 Support for Spatial Data

The following sections describe the SAP HANA support for spatial data.

### 2.3.1 Supported Spatial Data Types and Their Hierarchy

Spatial support follows the SQL Multimedia (SQL/MM) standard for storing and accessing geospatial data.

A key component of this standard is the use of the ST\_Geometry type hierarchy to define how geospatial data is created. Within the hierarchy, the prefix ST is used for all data types (also referred to as classes or types). When a column is identified as a specific type, the values of the type and its subtypes can be stored in the column. For example, a column identified as ST\_Geometry can also store the ST\_LineString and ST\_MultiLineString values.

#### Descriptions of supported spatial data types

The following spatial data types are supported:

<b>Geometries</b>	The term geometry means the overarching type for objects such as points, linestrings, and polygons. The geometry type is the supertype for all supported spatial data types.
<b>Points</b>	A point defines a single location in space. A point geometry does not have length or area. A point always has an X and Y coordinate.  ST_Dimension returns 0 for non-empty points.  In GIS data, points are typically used to represent locations such as addresses, or geographic features such as a mountain.
<b>Multipoints</b>	A multipoint is a collection of individual points.  In GIS data, multipoints are typically used to represent a set of locations.
<b>Linestrings</b>	A linestring is geometry with a length, but without any area. ST_Dimension returns 1 for non-empty linestrings. Linestrings can be characterized by whether they are simple or not simple, closed or not closed. <b>Simple</b> means a linestring that does not cross itself. <b>Closed</b> means a linestring that starts and ends at the same point. For example, a ring is an example of simple, closed linestring.

In GIS data, linestrings are typically used to represent rivers, roads, or delivery routes.

**Multilinestrings** A multilinestring is a collection of linestrings.

In GIS data, multilinestrings are often used to represent geographic features like rivers or a highway network.

**Polygons** A polygon defines a region of space. A polygon is constructed from one exterior bounding ring that defines the outside of the region and zero or more interior rings which define holes in the region. A polygon has an associated area but no length.

ST\_Dimension returns 2 for non-empty polygons.

In GIS data, polygons are typically used to represent territories (counties, towns, states, and so on), lakes, and large geographic features such as parks.

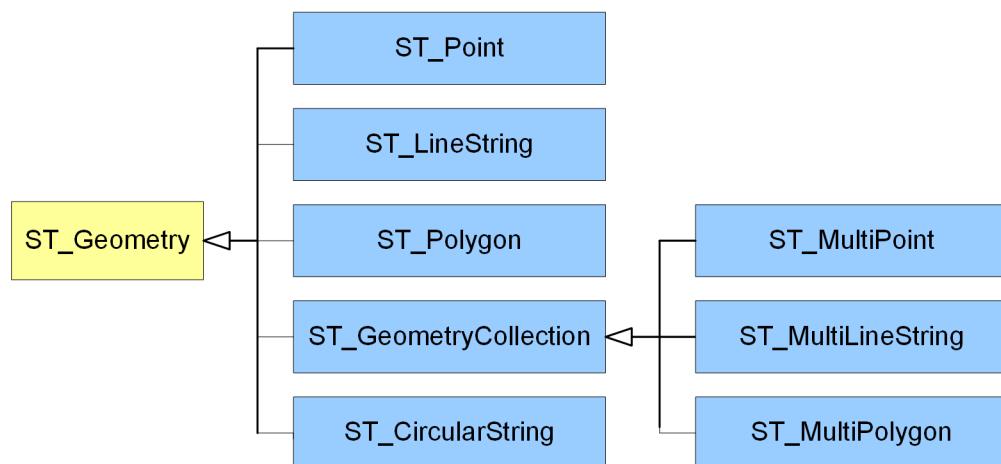
**Multipolygons** A multipolygon is a collection of zero or more polygons.

In GIS data, multipolygons are often used to represent territories made up of multiple regions (for example a state with islands), or geographic features such as a system of lakes.

**Circularstrings** A circularstring is a connected sequence of circular arc segments; much like a linestring with circular arcs between points.

## Spatial type hierarchy

The following diagram illustrates the hierarchy of the ST\_Geometry data types:



LEGEND:  Instantiable  Non-instantiable A B B is subtype of A

## Object-oriented properties of spatial data types

- A subtype (or derived type) is more specific than its supertype (or base type).
- A subtype inherits all methods from all supertypes. For example, ST\_Polygon values can call methods from the ST\_Geometry.
- A value of a subtype can be automatically converted to any of its supertypes. For example, an ST\_Point value can be used where a ST\_Geometry parameter is required, as in `point1.ST_Distance( point2 )`.
- A column or variable can store values of any subtype. For example, a column of type ST\_Geometry can store spatial values of any type.
- A column, variable, or expression with a declared type can be treated as, or cast to a subtype. For example, you can use the TREAT expression to change a ST\_Polygon value in a ST\_Geometry column named geom to have declared type ST\_Surface so you can call the ST\_Area method on it with `TREAT( geom AS ST_Surface ).ST_Area()`.

### 2.3.1.1 Supported Spatial Predicates

A predicate is a conditional expression that, combined with the logical operators AND and OR, makes up the set of conditions in a WHERE, HAVING, or ON clause, or in an IF or CASE expression, or in a CHECK constraint. In SQL, a predicate may evaluate to TRUE, FALSE. In many contexts, a predicate that evaluates to UNKNOWN is interpreted as FALSE.

Spatial predicates are implemented as member functions that return 0 or 1. To test a spatial predicate, your query should compare the result of the function to 1 or 0 using the = or <> operator. For example:

```
SELECT * FROM SpatialShapes WHERE shape.ST_IsEmpty() = 0;
```

You use predicates when querying spatial data to answer such questions as: how close together are two or more geometries? Do they intersect or overlap? Is one geometry contained within another? If you are a delivery company, for example, you may use predicates to determine if a customer is within a specific delivery area.

#### Related Information

[SpatialShapes Table \[page 32\]](#)

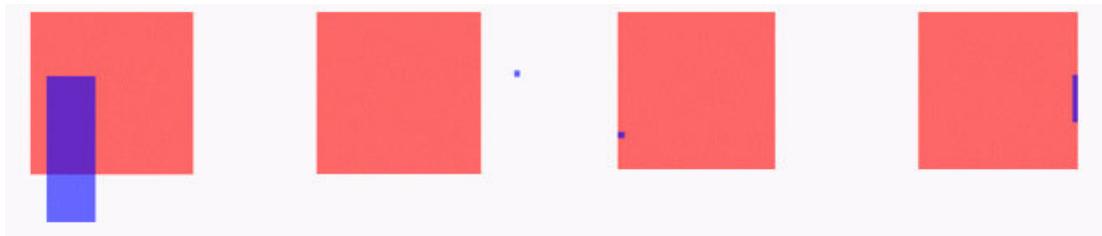
### 2.3.1.1.1 Intuitiveness of Spatial Predicates

Sometimes the outcome of a predicate is not intuitive.

Test special cases to make sure you are getting the results you want. For example, in order for a geometry to contain another geometry (`a.ST_Contains(b)=1`), or for a geometry to be within another geometry (`b.ST_Within(a)=1`), the interior of a and the interior of b must intersect, and no part of b can intersect the

exterior of a. However, there are some cases where you would expect a geometry to be considered contained or within another geometry, but it is not.

For example, the following return 0 (a is red) for `a.ST_Contains(b)` and `b.ST_Within(a)`:



Case one and two are obvious; the purple geometries are not completely within the red squares. Case three and four, however, are not as obvious. In both of these cases, the purple geometries are only on the boundary of the red squares. ST\_Contains does not consider the purple geometries to be within the red squares, even though they appear to be within them.

ST\_Covers and ST\_CoveredBy are similar predicates to ST\_Contains and ST\_Within. The difference is that ST\_Covers and ST\_CoveredBy do not require the interiors of the two geometries to intersect. Also, ST\_Covers and ST\_CoveredBy often have more intuitive results than ST\_Contains and ST\_Within.

If your predicate tests return a different result for cases than desired, consider using the ST\_Relate method to specify the exact relationship you are testing for.

## 2.3.2 Supported Import and Export Formats for Spatial Data

The following table lists the data and file formats supported for importing and exporting spatial data:

Table 1:

Data format	Import	Export	Description
Well Known Text (WKT)	Yes	Yes	<p>Geographic data expressed in ASCII text. This format is maintained by the Open Geospatial Consortium (OGC) as part of the Simple Features defined for the OpenGIS Implementation Specification for Geographic Information. See <a href="http://www.opengeospatial.org/standards/sfa">http://www.opengeospatial.org/standards/sfa</a>.</p> <p>Here is an example of how a point might be represented in WKT:</p> <div style="background-color: #f0f0f0; padding: 5px;">'POINT(1 1)'</div>
Well Known Binary (WKB)	Yes	Yes	<p>Geographic data expressed as binary streams. This format is maintained by the OGC as part of the Simple Features defined for the OpenGIS Implementation Specification for Geographic Information. See <a href="http://www.opengeospatial.org/standards/sfa">http://www.opengeospatial.org/standards/sfa</a>.</p> <p>Here is an example of how a point might be represented in WKB:</p> <div style="background-color: #f0f0f0; padding: 5px;">'01010000000000000000F03F000000000000F03F'</div>

Data format	Import	Export	Description
Extended Well Known Text (EWKT)	Yes	Yes	<p>WKT format, but with SRID information embedded. This format is maintained as part of PostGIS, the spatial database extension for PostgreSQL. See <a href="#">post-gis.refractions.net/</a>.</p> <pre>'srid=101;POINT (1 1)'</pre>
Extended Well Known Binary (EWKB)	Yes	Yes	<p>WKB format, but with SRID information embedded. This format is maintained as part of PostGIS, the spatial database extension for PostgreSQL. See <a href="#">post-gis.refractions.net/</a>.</p> <p>Here is an example of how a point might be represented in EWKB:</p> <pre>'01010000020040 0000000000000000 0F03F00000000000 00F03F'</pre>
ESRI shapefiles	Yes	No	<p>A popular geospatial vector data format for representing spatial objects in the form of shapefiles (several files that are used together to define the shape).</p>
GeoJSON	No	Yes	<p>Text format that uses name/value pairs, ordered lists of values, and conventions similar to those used in common programming languages such as C, C++, C#, Java, JavaScript, Perl, and Python.</p> <p>GeoJSON is a subset of the JSON standard and is used to encode geographic information. The GeoJSON standard is supported and the software provides the ST_AsGeoJSON method for converting SQL output to the GeoJSON format.</p> <p>Here is an example of how a point might be represented in GeoJSON:</p> <pre>{"x" : 1, "y" : 1, "spatialReference" : {"wkid" : 4326}}</pre> <p>For more information about the GeoJSON specification, see <a href="http://geojson.org/geojson-spec.html">http://geojson.org/geojson-spec.html</a>.</p>
Scalable Vector Graphic (SVG) files	No	Yes	<p>XML-based format used to represent twodimensional geometries. The SVG format is maintained by the World Wide Web Consortium (W3C). See <a href="http://www.w3.org/Graphics/SVG/">www.w3.org/Graphics/SVG/</a>.</p> <p>Here is an example of how a point might be represented in SVG:</p> <pre>&lt;rect width="1" height="1" fill="deepskyblue" stroke="black" strokeWidth="1" x="1" y="-1"/&gt;</pre>

## 2.3.3 Support for ESRI Shapefiles

The Environmental System Research Institute, Inc. (ESRI) shapefile format is supported. ESRI shapefiles are used to store geometry data and attribute information for the spatial features in a data set.

An ESRI shapefile includes at least three different files: .shp, .shx, and .dbf. The suffix for the main file is .shp, the suffix for the index file is .shx, and the suffix for the attribute columns is .dbf. All files share the same base name and are frequently combined in a single compressed file. The software can read all ESRI shapefiles with all shape types except MultiPatch. This includes shape types that include Z and M data.

The data in an ESRI shapefile usually contains multiple rows and columns. For example, the spatial tutorial loads a shapefile that contains zip code regions for Massachusetts. The shapefile contains one row for each zip code region, including the polygon information for the region. It also contains additional attributes (columns) for each zip code region, including the zip code name (for example, the string '02633') and other attributes.

### Related Information

<http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf> ↗

[Import ESRI Shapefiles Using Menu Functions \[page 17\]](#)

[Import ESRI Shapefiles Using SQL Commands \[page 16\]](#)

### 2.3.3.1 Import ESRI Shapefiles Using SQL Commands

You can import ESRI shapefiles using SQL commands.

To load ESRI shapefiles to a SAP HANA system perform these steps:

1. Create or acquire ESRI shapefiles.
2. Create a zip file containing the ESRI shapefiles.
3. Copy the zip file containing the ESRI shapefiles to the host on which the SAP HANA system runs.
4. Unzip the zip file which contains the ESRI shapefiles.
5. Make the files accessible to the <sid>adm user.
6. Call a SAP HANA studio, and log on to the SAP HANA System.
7. Open an SQL console.
8. Enter the SQL command to import the ESRI shapefiles.

To load a shapefile into a table, simply use the standard import statement with AS SHAPEFILE as follows:

```
IMPORT [<schema_name>.]<identifier> AS SHAPEFILE FROM <path> [WITH
<shapefile-option-list> ]
<schema_name>      ::= <identifier>
<path>              ::= <string_literal>
<shapefile_option_list> ::= <shapefile_option> [{, <shapefile_option_list>}]
<shapefile_option>  ::= CATALOG ONLY | REPLACE | SRID <srs-id> |
                           THREADS <number_of_threads>
<srs-id>            ::= <unsigned_integer>
<number_of_threads> ::= <unsigned_integer>
```

(**prefix**: filename without extension, **abs-path**: absolute path)

If the schema does not exist, it will be created before the import statement is performed.

#### Example

The following SQL commands could be used to import an ESRI shapefile:

```
IMPORT MY_SCHEMA.MY_TABLE AS SHAPEFILE FROM '/tmp/shapefile';
IMPORT MY_SCHEMA.MY_TABLE AS SHAPEFILE FROM '/tmp/shapefile' WITH REPLACE;
IMPORT MY_SCHEMA.MY_TABLE AS SHAPEFILE FROM '/tmp/shapefile' WITH SRID 4326;
IMPORT MY_SCHEMA.MY_TABLE AS SHAPEFILE FROM '/tmp/shapefile' WITH REPLACE SRID
4326;
IMPORT MY_SCHEMA.MY_TABLE AS SHAPEFILE FROM '/tmp/shapefile' WITH SRID 0
THREADS 4;
```

In the directory /tmp there are the following files: shapefile.shp, shapefile.shx, shapefile.dbf.

### 2.3.3.2 Import ESRI Shapefiles Using Menu Functions

You can import ESRI shapefiles using menu functions.

To load ESRI shapefiles to a SAP HANA system perform these steps:

1. Copy the ESRI shapefiles (for example `shapefile.shp`, `shapefile.shx`, `shapefile.dbf`) to the directory you have chosen, according to the import location (server, current client), you want to use, and provide the necessary access rights to the files.
2. Call a SAP HANA studio and log on to the SAP HANA system.
3. In the *File* menu, choose *Import*.
4. Expand the *SAP HANA* node.
5. Choose *ESRI Shapefiles*, and choose *Next*.
6. Choose the *Import Location*:
  - *Import ESRI shapefiles on server*  
You can choose the default directory on the server or enter a different directory, where the shapefiles are located on the server.
  - *Import ESRI shapefiles from current client*  
Enter the directory, where the shapefiles are located on the current client.
7. Choose *Next*.
8. Select the ESRI shapefiles you want to import, and choose *Next*.
9. Enter the name of the schema you want to import the ESRI shapefiles into, and choose the additional import options.
10. Choose *Finish*.

### 2.3.4 Multidimensional Support

SAP HANA Spatial supports multidimensional spatial data.

#### Note

SAP HANA spatial supports multidimensional spatial data for the column type `ST_GEOMETRY`.

The following dimension types are supported:

- 2D dimension (X, Y)
- 2D dimension with measure (X, Y, M)
- 3D dimension (X, Y, Z)
- 3D dimension with measure (X, Y, Z, M)

Multidimensional support is available for the following spatial data types (examples in WKT format):

- ST\_Point
  - 'Point (10 20)'
  - 'Point M(10 30 40)'
  - 'Point Z(10 30 60)'
  - 'Point ZM(10 20 30 50)'
- ST\_LineString
  - 'LineString (0 0, 5 10)'
  - 'LineString M(0 0 4, 5 10 6)'
  - 'LineString Z(0 0 7, 5 10 4)'
  - 'LineString ZM(0 0 3 6, 5 10 4 8)'
- ST\_Polygon
  - 'Polygon ((-5 -5, 5 -5, 0 5, -5 -5), (-2 -2, -2 0, 2 0, 2 -2, -2 -2))'
  - 'Polygon Z((-5 -5 6, 5 -5 6, 0 5 6, -5 -5 6), (-2 -2 1, -2 0 1, 2 0 1, 2 -2 1, -2 -2 1))'
  - 'Polygon M((-5 -5 6, 5 -5 6, 0 5 6, -5 -5 6), (-2 -2 1, -2 0 1, 2 0 1, 2 -2 1, -2 -2 1))'
  - 'Polygon ZM((-5 -5 6 4, 5 -5 6 4, 0 5 6 4, -5 -5 6 4), (-2 -2 1 4, -2 0 1 4, 2 0 1 4, 2 -2 1 4, -2 -2 1 4))'
- ST\_MultiPoint
  - 'MultiPoint ((10 10), (12 12), (14 10))'
  - 'MultiPoint Z((10 10 12), (12 12 14), (14 10 10))'
  - 'MultiPoint ZM((10 10 12 1), (12 12 14 1), (14 10 10 1))'
- ST\_MultiLineString
  - 'MultiLineString ((10 10, 12 12), (14 10, 16 12))'
  - 'MultiLineString Z((10 10 10, 12 12 12), (14 10 10, 16 12 12))'
  - 'MultiLineString ZM((10 10 10 10, 12 12 12 10), (14 10 12 13, 16 12 13 14))'
- ST\_MultiPolygon
  - 'MultiPolygon ((((-5 -5, 5 -5, 0 5, -5 -5), (-2 -2, -2 0, 2 0, 2 -2, -2 -2)), ((10 -5, 15 5, 5 5, 10 -5)))'
  - 'MultiPolygon M((((-5 -5 3, 5 -5 3, 0 5 3, -5 -5 3), (-2 -2 3, -2 0 3, 2 0 3, 2 -2 3, -2 -2 3)), ((10 -5 3, 15 5 3, 5 5 3, 10 -5 3)))'
  - 'MultiPolygon ZM((((-5 -5 4 1, 5 -5 7 1, 0 5 1 1, -5 -5 4 1), (-2 -2 9 1, -2 0 4 1, 2 0 4 1, 2 -2 1 1, -2 -2 9 1)), ((10 -5 2 1, 15 5 2 1, 5 5 3 1, 10 -5 2 1)))'
- ST\_GeometryCollection
  - 'GeometryCollection (LineString (5 10, 10 12, 15 10), Polygon ((10 -5, 15 5, 5 5, 10 -5)))'

- 'GeometryCollection Z(LineString Z(5 10 20, 10 12 25, 15 10 13), Polygon Z((10 -5 4, 15 5 6, 5 5 7, 10 -5 4)), Point Z(10 15 12))'

## 2.4 Recommended Reading on Spatial Topics

Various recommended readings on spatial topics are available.

- A brief overview of coordinates and their reference systems: <http://www.iogp.org/geomatics> ↗ -> Geodetic awareness (<http://www.iogp.org/pubs/373-01.pdf> ↗ )
- OGC OpenGIS Implementation Specification for Geographic information - Simple feature access: <http://www.opengeospatial.org/standards/sfs> ↗
- International Standard ISO/IEC 13249-3:2006: [http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=38651](http://www.iso.org/iso/catalogue_detail.htm?csnumber=38651) ↗
- Scalable Vector Graphics (SVG) 1.1 Specification: <http://www.w3.org/Graphics/SVG/> ↗
- JavaScript Object Notation (JSON): <http://json.org> ↗
- GeoJSON specification: <http://geojson.org/geojson-spec.html> ↗

## 2.5 Create Spatial Columns (SQL)

You can add spatial data to any table by adding a column that supports spatial data.

### Prerequisites

You must be the owner of the table, or have the ALTER privilege for the table, or alternatively have the ALTER ANY TABLE or ALTER ANY OBJECT system privilege.

#### i Note

The following column types are supported for column tables only:

- ST\_POINT  
Column type ST\_POINT supports 2D spatial data (X, Y) for spatial data type ST\_Point.
- ST\_GEOMETRY  
Column type ST\_GEOMETRY supports multidimensional spatial data for the following spatial data types: ST\_CircularString, ST\_GeometryCollection, ST\_LineString, ST\_MultiLineString, ST\_MultiPoint, ST\_MultiPolygon, ST\_Point, and ST\_Polygon.

## Procedure

1. Connect to the database.
2. Execute an ALTER TABLE statement.

## Results

A spatial column is added to the existing table.

### Example

The following statement adds a spatial column named Location to the SpatialShapes table. The new column is of spatial data type ST\_Point, and has a declared SRID of 1000004326, which is a flat-Earth spatial reference system.

```
ALTER TABLE SpatialShapes ADD (location ST_POINT(1000004326));
```

## Next Steps

You can place SRID constraints on the column in order to place restrictions on the values that can be stored in a spatial column.

## 2.6 Create a Table with Spatial Columns (SQL)

You create a table with columns which support spatial data.

You need to have the privilege for creating tables in a database schema.

### i Note

The following column types are supported for column tables only:

- ST\_POINT  
Column type ST\_POINT supports 2D spatial data (X, Y) for spatial data type ST\_Point.
- ST\_GEOMETRY  
Column type ST\_GEOMETRY supports multidimensional spatial data for the following spatial data types: ST\_CircularString, ST\_GeometryCollection, ST\_LineString, ST\_MultiLineString, ST\_MultiPoint, ST\_MultiPolygon, ST\_Point, and ST\_Polygon.

1. Connect to the database.
2. Execute a CREATE TABLE statement.

### Example

The following statement creates table `SpatialShapes_POINTS`, which contains column `SHAPE` with column type `ST_POINT`:

```
CREATE COLUMN TABLE MYSCHHEMA.SpatialShapes_POINTS
(
  ShapeID integer,
  SHAPE ST_Point
);
```

### Example

The following statement creates table `SpatialShapes_GEOMETRIES`, which contains column `SHAPE1` with column type `ST_POINT`, and column `SHAPE2` with column type `ST_GEOMETRY`:

```
CREATE COLUMN TABLE MYSCHHEMA.SpatialShapes_GEOMETRIES
(
  ShapeID integer,
  SHAPE1 ST_Point,
  SHAPE2 ST_Geometry
);
```

## 2.7 Spatial Data Type Syntax

The SQL/MM standard defines spatial data support in terms of user-defined extended types (UDTs) built on the ANSI/SQL CREATE TYPE statement. Although user-defined types are not supported, the spatial data support has been implemented as though they are supported.

### Instantiating instances of a UDT

You can instantiate a value of a user-defined type by calling a constructor as follows:

```
NEW <type-name>(<argument-list>)
```

For example, a query could contain the following to instantiate two `ST_Point` values:

```
SELECT NEW ST_Point(), NEW ST_Point(3,4) FROM dummy;
```

The database server matches `<argument-list>` against defined constructors using the normal overload resolution rules. An error is returned in the following situations:

- If `NEW` is used with a type that is not a user-defined type
- If the user-defined type is not instantiable (for example, `ST_Geometry` is not an instantiable type).
- If there is no overload that matches the supplied argument types

## Using instance methods

User defined types can have instance methods defined. Instance methods are invoked on a value of the type as follows:

```
<value-expression>.<method-name>( <argument-list> )
```

For example, the following fictitious example selects the X coordinate:

```
SELECT NEW ST_Point(2.25, 3.41).ST_X() FROM dummy;
```

If there was a user ID called CenterPoint, the database server would consider `CenterPoint.ST_X()` to be **ambiguous**. This is because the statement could mean "call the user-defined function `ST_X` owned by user `CenterPoint`" (the incorrect intention of the statement), or it could mean "call the `ST_X` method on the `Massdata.CenterPoint` column" (the correct meaning). The database server resolves the ambiguity by first performing a case-insensitive search for a user named `CenterPoint`. If one is found, the database server proceeds as though a user-defined function called `ST_X` and owned by user `CenterPoint` is being called. If no user is found, the database server treats the construct as a method call and calls the `ST_X` method on the `Massdata.CenterPoint` column.

An instance method invocation gives an error in the following cases:

- If the declared type of the `<value-expression>` is not a user-defined type
- If the named method is not defined in the declared type of `<value-expression>` or one of its supertypes
- If `<argument-list>` does not match one of the defined overloads for the named method.

## 2.8 Advanced Spatial Topics

This section contains advanced spatial topics.

### 2.8.1 How Flat-Earth and Round-Earth Representations Work

SAP HANA supports both flat-Earth and round-Earth representations.

**Flat-Earth** reference systems project all or a portion of the surface of the Earth to a flat, two dimensional plane (planar), and use a simple 2D Euclidean geometry. Lines between points are straight (except for circularstrings), and geometries cannot wrap over the edge (cross the dateline).

**Round-Earth** spatial reference systems use an ellipsoid to represent the Earth. Points are mapped to the ellipsoid for computations, all lines follow the shortest path and arc toward the pole, and geometries can cross the date line. The following characteristics are true for Round-Earth representations:

- For intersections, the line segments of the geometries are interpreted as parts of great circles on a sphere.
- A rectangle defined by two points LL and UR will be identical to the following polygon: `POLYGON ((X(LL) Y(LL), X(UR) Y(LL), X(UR) Y(UR), X(LL) Y(UR), X(LL) Y(LL)))`
- Distance calculations use the Vincenty's formulae on the ellipsoid using geodesy parameters of the SRS.

For information about which methods support Round-Earth representation, see *List of All Supported Methods* in the SAP HANA Spatial Reference.

**Both flat-Earth and round-Earth** representations have their limitations. There is not a single ideal map projection that best represents all features of the Earth, and depending on the location of an object on the Earth, distortions may affect its area, shape, distance, or direction.

#### Limitations of round-Earth spatial reference systems

- When working with a round-Earth spatial reference system such as WGS84, many operations are not available. For example, computing distance is restricted to points or collections of points.
- Some predicates and set operations are also not available.
- Circularstrings are not allowed in round-Earth spatial reference systems.
- Computations in round-Earth spatial reference systems are more expensive than the corresponding computation in a flat-Earth spatial reference system.
- The ST\_IntersectsRect method currently does not allow to cover more than 1 hemisphere (full globe support).
- Polygon rings are automatically reordered if they are not in counter-clockwise orientation. This prevents polygons covering more than one hemisphere as well.

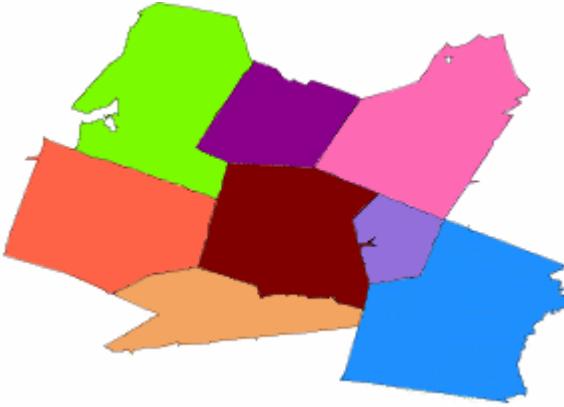
#### Limitations of flat-Earth spatial reference systems

A flat-Earth spatial reference system is a planar spatial reference system that has a projection defined for it. Projection resolves distortion issues that occur when using a flat-Earth spatial reference system to operate on round-Earth data. For example of the distortion that occurs if projection is not used, the next two images show the same group of zip code regions in Massachusetts. The first image shows the data in a SRID 3586, which is a projected planar spatial reference system specifically for Massachusetts data. The second image shows the data in a planar spatial reference system without projection (SRID 1000004326). The distortion manifests itself in the second image as larger-than-actual distances, lengths, and areas that cause the image to appear horizontally stretched.

##### Projected planar spatial reference system (SRID 3586)



##### Planar spatial reference system without projection (SRID 1000004326)



While more calculations are possible in flat-Earth spatial reference systems, calculations are only accurate for areas of bounded size, due to the effect of projection.

You can project round-Earth data to a flat-Earth spatial reference system to perform distance computations with reasonable accuracy provided you are working within distances of a few hundred kilometers. To project the data to a planar projected spatial reference system, you use the `ST_Transform` method.

## Related Information

[List of All Supported Methods \[page 155\]](#)

### 2.8.2 How Snap-to-Grid and Tolerance Impact Spatial Calculations

Spatial calculations are impacted by **Snap-to-grid** and **Tolerance**.

**Snap-to-grid** is the action of positioning the points in a geometry so they align with intersection points on a grid. When aligning a point with the **grid**, the X and Y values may be shifted by a small amount - similar to rounding. In the context of spatial data, a grid is a framework of lines that is laid down over a two-dimensional representation of a spatial reference system. The database server uses a square grid.

As a simplistic example of snap-to-grid, if the grid size is 0.2, then the line from `Point( 14.2321, 28.3262 )` to `Point( 15.3721, 27.1128 )` would be snapped to the line from `Point( 14.2, 28.4 )` to `Point( 15.4, 27.2 )`. Grid size is typically much smaller than this simplistic example, however, so the loss of precision is much less.

By default, the database server automatically sets the grid size so that 12 significant digits can be stored for every point within the X and Y bounds of a spatial reference system. For example, if the range of X values is from -180 to 180, and the range of Y values is from -90 to 90, the database server sets the grid size to `1e-9` (`0.000000001`). That is, the distance between both horizontal and vertical grid lines is `1e-9`. The intersection points of the grid lines represent all the points that can be represented in the spatial reference system. When a geometry is created or loaded, each point's X and Y coordinates are snapped to the nearest points on the grid.

**Tolerance** defines the distance within which two points or parts of geometries are considered equal. This can be thought of as all geometries being represented by points and lines drawn by a marker with a thick tip, where the thickness is equal to the tolerance. Any parts that touch when drawn by this thick marker are considered

equal within tolerance. If two points are exactly equal to tolerance apart, they are considered not equal within tolerance.

As a simplistic example of tolerance, if the tolerance is 0.5, then Point( 14.2, 28.4 ) and Point( 14.4, 28.2 ) are considered equal. This is because the distance between the two points (in the same units as X and Y) is about 0.283, which is less than the tolerance. Tolerance is typically much smaller than this simplistic example, however.

Tolerance can cause extremely small geometries to become invalid. Lines which have length less than tolerance are invalid (because the points are equivalent), and similarly polygons where all points are equal within tolerance are considered invalid.

Snap-to-grid and tolerance are set on the spatial reference system and are always specified in same units as the X and Y (or Longitude and Latitude) coordinates. Snap-to-grid and tolerance work together to overcome issues with inexact arithmetic and imprecise data. However, be aware of how their behavior can impact the results of spatial operations.

### Note

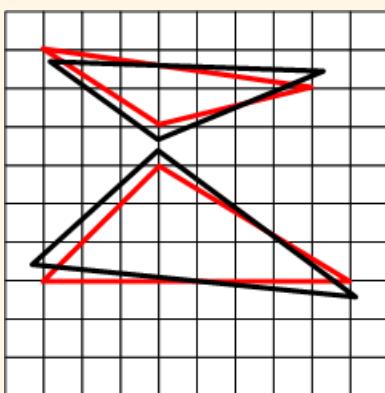
For planar spatial reference systems, setting grid size to 0 is never recommended as it can result in incorrect results from spatial operations.

The following examples illustrate the impact of grid size and tolerance settings on spatial calculations.

### Example

#### **Example 1: Snap-to-grid impacts intersection results**

Two triangles (shown in black) are loaded into a spatial reference system where tolerance is set to grid size, and the grid in the diagram is based on the grid size. The red triangles represent the black triangles after the triangle vertexes are snapped to the grid. Notice how the original triangles (black) are well within tolerance of each other, whereas the snapped versions in red do not. ST\_Intersects returns 0 for these two geometries. If tolerance was larger than the grid size, ST\_Intersects would return 1 for these two geometries.



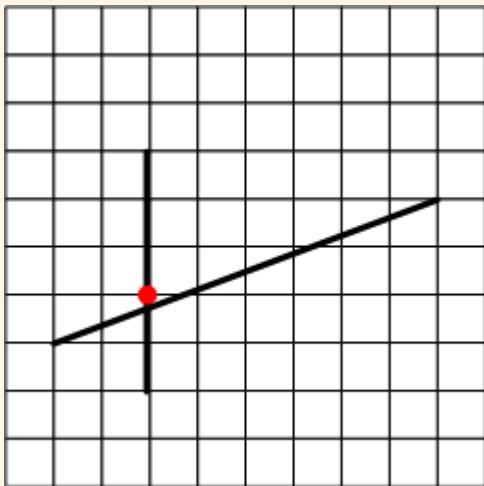
### Example

#### **Example 2: Tolerance impacts intersection results**

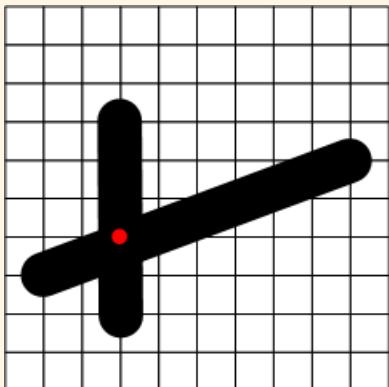
In the following example, two lines lie in a spatial reference system where tolerance is set to 0. The intersection point of the two lines is snapped to the nearest vertex in the grid. Since tolerance is set to 0, a test to determine if the intersection point of the two lines intersects the diagonal line returns false.

In other words, the following expression returns 0 when tolerance is 0:

```
vertical_line.ST_Intersection( diagonal_line ).ST_Intersects( diagonal_line )
```



Setting the tolerance to grid size (the default), however, causes the intersection point to be inside the thick diagonal line. So a test of whether the intersection point intersects the diagonal line within tolerance would pass:

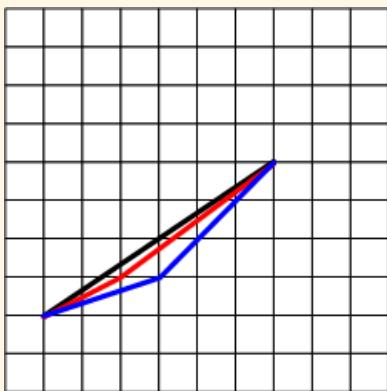




## Example

### Example 3: Tolerance and transitivity

In spatial calculations when tolerance is in use, transitivity does not necessarily hold. For example, suppose you have the following three lines in a spatial reference system where the tolerance is equal to the grid size:



The ST\_Equals method considers the black and red lines to be equivalent within tolerance, and the red and blue lines to be equivalent within tolerance but the black line and the blue line are not equivalent within tolerance. ST\_Equals is not transitive.

ST\_OrderingEquals considers each of these lines to be different, and ST\_OrderingEquals is transitive.



## Example

### Example 4: Impact of grid and tolerance settings on imprecise data

Suppose you have data in a projected planar spatial reference system which is mostly accurate to within 10 centimeters, and always accurate to within 10 meters. You have three choices:

1. Use the default grid size and tolerance that the database server selects, which is normally greater than the precision of your data. Although this provides maximum precision, predicates such as ST\_Intersects, ST\_Touches, and ST\_Equals may give results that are different than expected for some geometries, depending on the accuracy of the geometry values. For example, two adjacent polygons that share a border with each other may not return true for ST\_Intersects if the leftmost polygon has border data a few meters to the left of the rightmost polygon.
2. Set the grid size to be small enough to represent the most accuracy in any of your data (10 centimeters, in this case) and at least four times smaller than the tolerance, and set tolerance to represent the distance to which your data is always accurate to (10 meters, in this case). This strategy means your data is stored without losing any precision, and that predicates will give the expected result even though the data is only accurate within 10 meters.
3. Set grid size and tolerance to the precision of your data (10 meters, in this case). This way your data is snapped to within the precision of your data, but for data that is more accurate than 10 meters the additional accuracy is lost.

In many cases predicates will give the expected results but in some cases they will not. For example, if two points are within 10 centimeters of each other but near the midway point of the grid intersections, one point will snap one way and the other point will snap the other way, resulting in the points being about 10 meters apart. For this reason, setting grid size and tolerance to match the precision of your data is not recommended in this case.

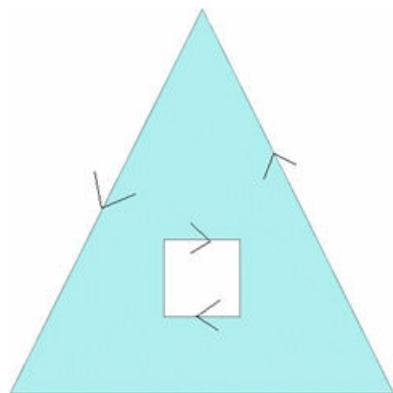
## 2.8.3 How Polygon Ring Orientation Works

Internally, the database server interprets polygons by looking at the orientation of the constituent rings.

As one travels a ring in the order of the defined points, the inside of the polygon is on the left side of the ring. The same rules are applied in PLANAR and ROUND EARTH spatial reference systems. In most cases, outer rings are in counter-clockwise orientation and interior rings are in the opposite (clockwise) orientation. The exception is for rings that contain the north or south pole in ROUND EARTH.

By default, polygons are automatically reoriented if they are created with a different ring orientation than the internal ring orientation.

For example, if you create a polygon and specify the points in a clockwise order `Polygon((0 0, 5 10, 10 0, 0 0), (4 2, 4 4, 6 4, 6 2, 4 2))`, the database server automatically rearranges the points to be in counter-clockwise rotation, as follows: `Polygon((0 0, 10 0, 5 10, 0 0), (4 2, 4 4, 6 4, 6 2, 4 2))`.



If the inner ring was specified before the outer ring, the outer ring would appear as the first ring

In order for polygon reorientation to work in round-Earth spatial reference systems, polygons are limited to 160° in diameter.

## 2.8.4 How Geometry Interiors, Exteriors, and Boundaries Work

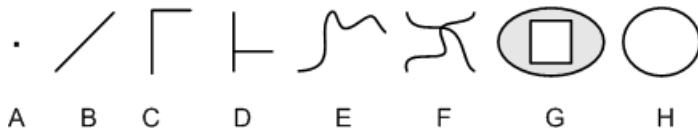
The **interior** of a geometry is all points that are part of the geometry except the boundary.

The **exterior** of a geometry is all points that are not part of the geometry. This can include the space inside an interior ring, for example in the case of a polygon with a hole. Similarly, the space both inside and outside a linestring ring is considered the exterior.

The **boundary** of a geometry is what is returned by the ST\_Boundary method.

Knowing the boundary of a geometry helps when comparing to another geometry to determine how the two geometries are related. However, while all geometries have an interior and an exterior, not all geometries have a boundary, nor are their boundaries always intuitive.

Here are some cases of geometries where the boundary may not be intuitive:



<b>Point</b>	A point (such as A) has no boundary.
<b>Lines and linestrings</b>	The boundary for lines and linestrings (B, C, D, E, F) are their endpoints. Geometries B, C, and E have two end points for a boundary. Geometry D has four end points for a boundary, and geometry F has four.
<b>Polygon</b>	The boundary for a polygon (such as G) is its outer ring and any inner rings.
<b>Rings</b>	A ring—a curve where the start point is the same as the end point and there are no self-intersections (such as H)—has no boundary.

## 2.8.5 How Spatial Comparisons Work

There are two methods you can use to test whether a geometry is equal to another geometry: ST\_Equals, and ST\_OrderingEquals. These methods perform the comparison differently, and return a different result.

<b>ST_Equals</b>	The order in which points are specified does not matter, and point comparison takes tolerance into account. Geometries are considered equal if they occupy the same space, within tolerance. For example, if two linestrings occupy the same space, yet one is defined with more points, they are still considered equal.
<b>ST_OrderingEquals</b>	With ST_OrderingEquals, the two geometries must contain the same hierarchy of objects with the exact same points in the same order to be considered equal under ST_OrderingEquals. That is, the two geometries must be exactly the same.

To illustrate the difference in results when comparisons are made using ST\_Equals versus ST\_OrderingEquals, consider the following lines. ST\_Equals considers them all equal (assuming line C is within tolerance). However, ST\_OrderingEquals does not consider any of them equal.

1 •————— A—————• 2	LineString( 0.0 0.0, 4.0 0.0 )
2 •————— B—————• 1	LineString( 4.0 0.0, 0.0 0.0 )
1 •————— C—————• 2	LineString( 0.0 0.0, 4.0 0.000001 )
1 •————— D—————• 3	LineString( 0.0 0.0, 1.0 0.0, 4.0 0.0 )
1 •————— E—————• 2	MultiLineString(( 0.0 0.0, 4.0 0.0 ))

## 2.8.6 How Spatial Relationships Work

For best performance, use methods like ST\_Within, or ST\_Touches to test single, specific relationships between geometries. However, if you have more than one relationship to test, ST\_Relate can be a better

method, since you can test for several relationships at once. ST\_Relate is also good when you want to test for a different interpretation of a predicate.

The most common use of ST\_Relate is as a predicate, where you specify the exact relationship(s) to test for. However, you can also use ST\_Relate to determine all possible relationships between two geometries.

#### Predicate use of ST\_Relate

ST\_Relate assesses how geometries are related by performing intersection tests of their interiors, boundaries, and exteriors. The relationship between the geometries is then described in a 9-character string in DE-9IM (Dimensionally Extended 9 Intersection Model) format, where each character of the string represents the dimension of the result of an intersection test.

When you use ST\_Relate as a predicate, you pass a DE-9IM string reflecting intersection results to test for. If the geometries satisfy the conditions in the DE-9IM string you specified, then ST\_Relate returns a 1. If the conditions are not satisfied, then 0 is returned. If either or both of the geometries is NULL, then NULL is returned.

The 9-character DE-9IM string is a flattened representation of a pair-wise matrix of the intersection tests between interiors, boundaries, and exteriors. The next table shows the 9 intersection tests in the order they are performed: left to right, top to bottom:

Table 2:

	<b>g2 interior</b>	<b>g2 boundary</b>	<b>g2 exterior</b>
<b>g1 interior</b>	Interior(g1) $\cap$ Interior(g2)	Interior(g1) $\cap$ Boundary(g2)	Interior(g1) $\cap$ Exterior(g2)
<b>g1 boundary</b>	Boundary(g1) $\cap$ Interior(g2)	Boundary(g1) $\cap$ Boundary(g2)	Boundary(g1) $\cap$ Exterior(g2)
<b>g1 exterior</b>	Exterior(g1) $\cap$ Interior(g2)	Exterior(g1) $\cap$ Boundary(g2)	Exterior(g1) $\cap$ Exterior(g2)

When you specify the DE-9IM string, you can specify \*, 0, 1, 2, T, or F for any of the 9 characters. These values refer to the number of dimensions of the geometry created by the intersection.

Table 3:

<b>When you specify:</b>	<b>The intersection test result must return:</b>
T	one of: 0, 1, 2 (an intersection of any dimension)
F	-1
*	-1, 0, 1, 2 (any value)
0	0
1	1
2	2

Suppose you want to test whether a geometry is within another geometry using ST\_Relate and a custom DE-9IM string for the within predicate:

```
SELECT NEW ST_Polygon('Polygon(( 2 3, 8 3, 4 8, 2 3 ))').
ST_Relate(NEW ST_Polygon('Polygon((-3 3, 3 3, 3 6, -3 6, -3 3))'), 'T*F**F***' )
FROM dummy;
```

This is equivalent to asking ST\_Relate to look for the following conditions when performing the intersection tests:

Table 4:

	g2 interior	g2 boundary	g2 exterior
g1 interior	one of: 0, 1, 2	one of: 0, 1, 2, -1	-1
g1 boundary	one of: 0, 1, 2, -1	one of: 0, 1, 2, -1	-1
g1 exterior	one of: 0, 1, 2, -1	one of: 0, 1, 2, -1	one of: 0, 1, 2, -1

When you execute the query, however, ST\_Relate returns 0 indicating that the first geometry is not within the second geometry.

#### Non-predicate use of ST\_Relate

The non-predicate use of ST\_Relate returns the full relationship between two geometries.

For example, suppose you have the same two geometries used in the previous example and you want to know how they are related. You would execute the following statement in Interactive SQL to return the DE-9IM string defining their relationship.

```
SELECT NEW ST_Polygon('Polygon(( 2 3, 8 3, 4 8, 2 3 ))').  
ST_Relate(NEW ST_Polygon('Polygon((-3 3, 3 3, 3 6, -3 6, -3 3))')) FROM dummy;
```

ST\_Relate returns the DE-9IM string, 212111212.

The matrix view of this value shows that there are many points of intersection:

Table 5:

	g2 interior	g2 boundary	g2 exterior
g1 interior	2	1	2
g1 boundary	1	1	1
g1 exterior	2	1	2

## 2.8.7 How Spatial Dimensions Work

As well as having distinct properties of its own, each of the geometry subtypes inherits properties from the ST\_Geometry supertype.

A geometry subtype has one of the following dimensional values:

- -1 – A value of -1 indicates that the geometry is empty (it does not contain any points).
- 0 – A value of 0 indicates the geometry has no length or area. The subtypes ST\_Point and ST\_MultiPoint have dimensional values of 0. A point represents a geometric feature that can be represented by a single pair of coordinates, and a cluster of unconnected points represents a multipoint feature.
- 1 – A value of 1 indicates the geometry has length but no area. The set of subtypes that have a dimension of 1 is ST\_LineString, or collection types containing this type. In GIS data, these geometries of dimension 1 are used to define linear features such as streams, branching river systems, and road segments.
- 2 – A value of 2 indicates the geometry has area. The set of subtypes that have a dimension of 2 are subtypes of ST\_Surface (ST\_Polygon), or collection types containing these types. Polygons and

multipolygons represent geometric features with perimeters that enclose a defined area such as lakes or parks.

The dimension of a geometry is not related to the number of coordinate dimensions of each point in a geometry.

A single ST\_GeometryCollection can contain geometries of different dimensions, and the highest dimension geometry is returned.

## 2.9 SpatialShapes Table

Several examples in this documentation refer to spatial shapes. The following table is used in those examples to further explain some functions.

You can use the following syntax to create a test table:

```
CREATE COLUMN TABLE SpatialShapes
(
    ShapeID integer,
    shape ST_GEOMETRY
);
-- a set of points
INSERT INTO SpatialShapes VALUES(1, NEW ST_POINT('POINT(2.5 3.0)' ));
INSERT INTO SpatialShapes VALUES(2, NEW ST_POINT('POINT(3.0 4.5)' ));
INSERT INTO SpatialShapes VALUES(3, NEW ST_POINT('POINT(3.0 6.0)' ));
INSERT INTO SpatialShapes VALUES(4, NEW ST_POINT('POINT(4.0 6.0)' ));
INSERT INTO SpatialShapes VALUES(5, NEW ST_POINT());
-- a set of linestrings
INSERT INTO SpatialShapes VALUES(6, NEW ST_LINESTRING('LINESTRING(3.0 3.0, 5.0
4.0, 6.0 3.0)' ));
INSERT INTO SpatialShapes VALUES(7, NEW ST_LINESTRING('LINESTRING(4.0 4.0, 6.0
5.0, 7.0 4.0)' ));
INSERT INTO SpatialShapes VALUES(8, NEW ST_LINESTRING('LINESTRING(7.0 5.0, 9.0
7.0)' ));
INSERT INTO SpatialShapes VALUES(9, NEW ST_LINESTRING('LINESTRING(7.0 3.0, 8.0
5.0)' ));
INSERT INTO SpatialShapes VALUES(10, NEW ST_LINESTRING());
-- a set of polygons
INSERT INTO SpatialShapes VALUES(11, NEW ST_POLYGON('POLYGON((6.0 7.0, 10.0 3.0,
10.0 10.0, 6.0 7.0))' ));
INSERT INTO SpatialShapes VALUES(12, NEW ST_POLYGON('POLYGON((4.0 5.0, 5.0 3.0,
6.0 5.0, 4.0 5.0))' ));
INSERT INTO SpatialShapes VALUES(13, NEW ST_POLYGON('POLYGON((1.0 1.0, 1.0 6.0,
6.0 6.0, 6.0 1.0, 1.0 1.0))' ));
INSERT INTO SpatialShapes VALUES(14, NEW ST_POLYGON('POLYGON((1.0 3.0, 1.0 4.0,
5.0 4.0, 5.0 3.0, 1.0 3.0))' ));
INSERT INTO SpatialShapes VALUES(15, NEW ST_POLYGON());
```

### i Note

The ST\_Geometry/ST\_Point column is using the default spatial reference system 0 (Euclidean space). If you want to use a different spatial reference system (for instance 1000004326) you have to specify it explicitly at table creation time:

```
CREATE COLUMN TABLE SpatialShapes
(
    ShapeID integer,
    shape ST_GEOMETRY(1000004326)
```

```
) ;
```

### Note

In descriptions of the methods for accessing and manipulatig spatial data there are several examples based on the entries in the table `SpatialShapes`. So make sure the table `SpatialShapes` exists, when you use these examples.

# 3 Accessing and Manipulating Spatial Data

This section describes the types, methods, and constructors you can use to access, manipulate, and analyze spatial data. The spatial data types can be considered like data types or classes. Each spatial data type has associated methods and constructors you use to access the data.

## 3.1 ST\_Geometry Type

The ST\_Geometry type is the maximal supertype of the geometry type hierarchy.

### Direct subtypes

[ST\\_CircularString \[page 107\]](#)  
[ST\\_GeometryCollection \[page 111\]](#)  
[ST\\_LineString \[page 116\]](#)  
[ST\\_MultiLineString \[page 126\]](#)  
[ST\\_MultiPoint \[page 132\]](#)  
[ST\\_MultiPolygon \[page 136\]](#)  
[ST\\_Point \[page 141\]](#)  
[ST\\_Polygon \[page 148\]](#)

### Methods

- Methods of ST\_Geometry:

Table 6:

<a href="#">ST_AsBinary [page 36]</a>	<a href="#">ST_AsEWKB [page 37]</a>	<a href="#">ST_AsEWKT [page 37]</a>	<a href="#">ST_AsGeoJSON [page 38]</a>
<a href="#">ST_AsSVG [page 39]</a>	<a href="#">ST_AsSVGAgr [page 40]</a>	<a href="#">ST_AsText [page 41]</a>	<a href="#">ST_AsWKB [page 42]</a>
<a href="#">ST_AsWKT [page 43]</a>	<a href="#">ST_Boundary [page 44]</a>	<a href="#">ST_Buffer [page 44]</a>	<a href="#">ST_Contains [page 46]</a>
<a href="#">ST_ConvexHull [page 48]</a>	<a href="#">ST_ConvexHullAggr [page 49]</a>	<a href="#">ST_CoordDim [page 50]</a>	<a href="#">ST_CoveredBy [page 52]</a>
<a href="#">ST_Covers [page 53]</a>	<a href="#">ST_Crosses [page 55]</a>	<a href="#">ST_Difference [page 56]</a>	<a href="#">ST_Dimension [page 57]</a>

<a href="#">ST_Disjoint [page 58]</a>	<a href="#">ST_Distance [page 60]</a>	<a href="#">ST_Envelope [page 61]</a>	<a href="#">ST_EnvelopeAggr [page 62]</a>
<a href="#">ST_Equals [page 63]</a>	<a href="#">ST_GeomFromEWKB [page 64]</a>	<a href="#">ST_GeomFromEWKT [page 65]</a>	<a href="#">ST_GeomFromText [page 66]</a>
<a href="#">ST_GeomFromWKB [page 68]</a>	<a href="#">ST_GeomFromWKT [page 69]</a>	<a href="#">ST_GeometryType [page 70]</a>	<a href="#">ST_Intersection [page 71]</a>
<a href="#">ST_Intersects [page 72]</a>	<a href="#">ST_IntersectionAggr [page 73]</a>	<a href="#">ST_IntersectsFilter [page 74]</a>	<a href="#">ST_IntersectsRect [page 74]</a>
<a href="#">ST_Is3D [page 77]</a>	<a href="#">ST_IsEmpty [page 78]</a>	<a href="#">ST_IsMeasured [page 79]</a>	<a href="#">ST_IsSimple [page 79]</a>
<a href="#">ST_IsValid [page 80]</a>	<a href="#">ST_MMax [page 81]</a>	<a href="#">ST_MMin [page 81]</a>	<a href="#">ST_NumInteriorRings [page 83]</a>
<a href="#">ST_NumInteriorRing [page 82]</a>	<a href="#">ST_OrderingEquals [page 83]</a>	<a href="#">ST_Overlaps [page 85]</a>	<a href="#">ST_Perimeter [page 87]</a>
<a href="#">ST_PointOnSurface [page 88]</a>	<a href="#">ST_Relate [page 89]</a>	<a href="#">ST_SRID [page 90]</a>	<a href="#">ST_SRID(INT) [page 91]</a>
<a href="#">ST_SnapToGrid [page 92]</a>	<a href="#">ST_SymDifference [page 93]</a>	<a href="#">ST_Touches [page 94]</a>	<a href="#">ST_Transform [page 96]</a>
<a href="#">ST_Union [page 97]</a>	<a href="#">ST_UnionAggr [page 98]</a>	<a href="#">ST_Within [page 99]</a>	<a href="#">ST_WithinDistance [page 100]</a>
<a href="#">ST_XMax [page 102]</a>	<a href="#">ST_XMin [page 103]</a>	<a href="#">ST_YMax [page 104]</a>	<a href="#">ST_YMin [page 105]</a>
<a href="#">ST_ZMax [page 105]</a>	<a href="#">ST_ZMin [page 106]</a>		

## Remarks

The ST\_Geometry type is the maximal supertype of the geometry type hierarchy. The ST\_Geometry type supports methods that can be applied to any spatial value. The ST\_Geometry type cannot be instantiated; instead, a subtype should be instantiated. When working with original formats (WKT or WKB), you can use methods such as ST\_GeomFromText/ST\_GeomFromWKB to instantiate the appropriate concrete type representing the value in the original format.

All of the values in an ST\_Geometry value are in the same spatial reference system. The ST\_SRID method can be used to retrieve the spatial reference system associated with the value.

Columns of type ST\_Geometry or any of its subtypes cannot be included in a primary key, unique index, or unique constraint.

## Related Information

[ST\\_SRID Method \[page 90\]](#)

## 3.1.1 ST\_AsBinary Method

Returns the WKB representation of an ST\_Geometry value.

### Syntax

```
<geometry-expression>.ST_AsBinary()
```

### Returns

**BLOB (Binary Large Object)** Returns the WKB representation of the `<geometry-expression>`.

### Remarks

The ST\_AsBinary method returns a binary string representing the geometry in WKB format.

#### Example

The following returns the result 01010000000000000000F03F0000000000000040.

```
SELECT NEW ST_Point( 1.0, 2.0 ).ST_AsBinary() FROM dummy;
```

### Related Information

[ST\\_AsWKB Method \[page 42\]](#)

### **3.1.2 ST\_AsEWKB Method**

Returns the extended WKB representation of an ST\_Geometry value.

## Syntax

```
<geometry-expression>.ST_AsEWKB()
```

## Returns

**BLOB (Binary Large Object)** Returns a binary string containing the extended WKB representation of the <geometry-expression>.

## Remarks

The `ST_AsEWKB` method returns a binary string representing the geometry in extended WKB format.

## Example

```
SELECT NEW ST Point(0.0,0.0).ST AsEWKB() FROM dummy;
```

### 3.1.3 ST\_AsEWKT Method

Returns the extended WKT representation of an ST\_Geometry value.

## Syntax

```
<geometry-expression>.ST_AsEWKT()
```

## Returns

**CLOB (Character Large Object)** Returns a string containing the extended WKT representation of the `<geometry-expression>`.

## Remarks

The ST\_AsEWKT method returns a string representing the geometry in extended WKT format.

### Example

The following example returns the result SRID=0;POINT (2 4).

```
SELECT NEW ST_Point(2.0,4.0).ST_AsEWKT() FROM dummy;
```

## 3.1.4 ST\_AsGeoJSON Method

Returns a string representing a geometry in JSON format.

## Syntax

```
<geometry-expression>.ST_AsGeoJSON()
```

## Returns

**CLOB (Character Large Object)** Returns the GeoJSON representation of the `<geometry-expression>`.

## Remarks

The GeoJSON standard defines a geospatial interchange format based on the JavaScript Object Notation (JSON). This format is suited to web-based applications and it can provide a format that is more concise and easier to interpret than WKT or WKB. See <http://geojson.org/geojson-spec.html>.

The ST\_AsGeoJSON method returns a text string representing the geometry.

### Example

The following returns the result {"type": "Point", "coordinates": [1.0, 2.0]}.

```
SELECT NEW ST_Point( 1.0, 2.0 ).ST_AsGeoJSON() FROM dummy;
```

## 3.1.5 ST\_AsSVG Method

Returns an SVG figure representing a geometry value.

### Syntax

```
<geometry-expression>.ST_AsSVG([<parameter>[,...]])
```

### Parameters

Table 7:

Name	Type	Description
parameter	VARCHAR(128)	A key value pair defining the parameters to use when converting the geometry-expression to an SVG representation.

The following parameters can be specified:

Table 8:

Parameter	Values	Description
Approximate	<yes   no>	Defines, if the geometry should have reduced details
Attribute	<text>	Additional SVG attributes
DecimalDigits	<integer>	Precision of coordinate
PathDataOnly	<yes   no>	Only paths are created
RandomFill	<yes   no>	Randomly assign fill color

Parameter	Values	Description
Relative	<yes   no>	Start point of the plot

## Returns

CLOB (Character Large Object)      Returns an SVG rendering of the [`<geometry-expression>`](#).

### Example

The following returns a complete SVG document with polygons filled with random colors.

```
SELECT NEW ST_Polygon( 'Polygon(( 0 0, 0 20, 60 10, 0 0 ))' ).ST_AsSVG() FROM dummy;
```

The following returns a complete SVG document with polygons with no fill and precision 3.

```
SELECT NEW ST_Polygon( 'Polygon(( 0 0, 0 20, 60 10, 0 0 ))' ).ST_AsSVG(RandomFill=>'no', DecimalDigits=>3) FROM dummy;
```

The following returns a complete SVG document with polygons filled with random colors and precision 3.

```
SELECT NEW ST_Polygon( 'Polygon(( 0 0, 0 20, 60 10, 0 0 ))' ).ST_AsSVG(RandomFill=>'yes', DecimalDigits=>3) FROM dummy;
```

## 3.1.6 ST\_AsSVGAgr Method

Returns a complete or partial SVG document which renders the geometries in a group.

## Syntax

```
ST_AsSVGAgr(<geometry_column>)
```

## Parameters

Table 9:

Name	Type	Description
geometry_column	ST_Geometry	The geometry value to contribute to the SVG figure. Typically this is a column.

## Returns

**LONG VARCHAR** Returns a complete or partial SVG document which renders the geometries in a group.

### Example

The following returns a complete SVG document with polygons filled with random colors.

```
SELECT ST_AsSVGAggr(Shape) FROM SpatialShapes;
```

## 3.1.7 ST\_AsText Method

Returns the text representation of an ST\_Geometry value.

## Syntax

```
<geometry-expression>.ST_AsText()
```

## Returns

**CLOB (Character Large Object)** Returns the text representation of the `<geometry-expression>`.

## Remarks

The ST\_AsText method returns a text string representing the geometry in well known text format (WKT).

### Example

The following returns the result POINT (1 2).

```
SELECT NEW ST_Point( 1.0, 2.0 ).ST_AsText() FROM dummy;
```

## Related Information

[ST\\_AsWKT Method \[page 43\]](#)

### 3.1.8 ST\_AsWKB Method

Returns the WKB representation of an ST\_Geometry value.

## Syntax

```
<geometry-expression>.ST_AsWKB()
```

## Returns

**BLOB (Binary Large Object)** Returns the WKB representation of the `<geometry-expression>`.

## Remarks

The ST\_AsWKB method returns a binary string representing the geometry in WKB format.

### Example

The following returns the result 0101000000000000000000F03F0000000000000040.

```
SELECT NEW ST_Point( 1.0, 2.0 ).ST_AsWKB() FROM dummy;
```

## Related Information

[ST\\_AsBinary Method \[page 36\]](#)

### 3.1.9 ST\_AsWKT Method

Returns the WKT representation of an ST\_Geometry value.

#### Syntax

```
<geometry-expression>.ST_AsWKT()
```

#### Returns

CLOB (Character Large Object)      Returns the WKT representation of the <geometry-expression>.

#### Remarks

The ST\_AsWKT method returns a text string representing the geometry in well known text format (WKT).

##### Example

The following returns the result POINT (1 2).

```
SELECT NEW ST_Point( 1.0, 2.0 ).ST_AsWKT() FROM dummy;
```

## Related Information

[ST\\_AsText Method \[page 41\]](#)

## 3.1.10 ST\_Boundary Method

Returns the boundary of the geometry value.

### Syntax

```
<geometry-expression>.ST_Boundary()
```

### Returns

**ST\_Geometry** Returns a geometry value representing the boundary of the `<geometry-expression>`.

#### Example

The following example constructs a geometry collection containing a polygon and a linestring and returns the boundary for the collection: GEOMETRYCOLLECTION (LINESTRING (0 0,3 0,3 3,0 3,0),POINT (0 7),POINT (4 4)).

```
SELECT NEW ST_GeometryCollection('GeometryCollection (Polygon ((0 0, 3 0, 3 3, 0 3, 0 0)), LineString (0 7, 0 4, 4 4))').ST_Boundary().ST_AsText() FROM dummy;
```

## 3.1.11 ST\_Buffer Method

Returns the ST\_Geometry value that represents all points whose distance from any point of an ST\_Geometry value is less than or equal to a specified distance in the given units.

### Syntax

```
<geometry-expression>.ST_Buffer(<distance>[, <unit_name>])
```

## Parameters

Table 10:

Name	Type	Description
distance	DOUBLE	The distance the buffer should be from the geometry value.
unit_name	VARCHAR(128)	The units in which the distance parameter should be interpreted. Defaults to the unit of the spatial reference system. The unit name must match the UNIT_NAME column of a row in the ST_UNITS_OF_MEASURE view where UNIT_TYPE is 'LINEAR'.

## Returns

**ST\_Geometry** Returns the ST\_Geometry value representing all points within the specified distance of the [`<geometry-expression>`](#).

The spatial reference system identifier of the result is the same as the spatial reference system of the [`<geometry-expression>`](#).

## Remarks

The ST\_Buffer method generates a geometry that expands a geometry by the specified distance. This method can be used, for example, to find all points in geometry A that are within a specified distance of geometry B. The distance parameter must be a positive value. This method will return an error if distance is negative. If the distance parameter is equal to 0, the original geometry is returned. The ST\_Buffer method is best used only when the actual buffer geometry is required. Determining whether two geometries are within a specified distance of each other should be done using ST\_WithinDistance instead.

**i Note**

If the [`<geometry-expression>`](#) is an empty geometry (`ST_IsEmpty()=1`), then this method returns NULL.

**i Note**

This method cannot be used with geometries in round-Earth spatial reference systems.

## Example

The following example shows the buffer of radius 1 computed on a polygon with 2 rings.

```
SELECT NEW ST_Polygon('Polygon((1 2, 10 2, 10 10, 1 2),(9 3, 5 3, 9 9, 9 3))').ST_Buffer(1.0) FROM dummy;
```

The following example returns the buffer of radius 1 around a single point (1, 2). The buffer of a single point is a polygon which is an approximation of a circle.

```
SELECT NEW ST_Point(1,2).ST_Buffer(1.0).ST_AsText() FROM dummy;
```

The following examples returns the buffer of a straight line in well known binary format:

```
SELECT NEW ST_LineString('LineString(1 2, 2 2)').ST_Buffer(1.0) FROM dummy;
```

The following example determines which part of the line from (0, 2) to (4, 2) is within 1 unit of the point (2, 2.5). The result is LINESTRING (1.137272 2, 2.863421 2).

```
SELECT NEW ST_Point(2, 2.5).ST_Buffer(1.0).ST_Intersection( NEW ST_LineString('Linestring(0 2, 4 2)').ST_AsText()) FROM dummy;
```

## 3.1.12 ST\_Contains Method

Tests if a geometry value spatially contains another geometry value.

### Syntax

```
<geometry-expression>.ST_Contains(<geo2>)
```

### Parameters

Table 11:

Name	Type	Description
geo2	ST_Geometry	The other geometry value that is to be compared to the <geometry-expression>.

## Returns

**INTEGER** Returns 1 if the `<geometry-expression>` contains `<geo2>`, otherwise 0. If `<geo2>` is NULL, then NULL is returned.

## Remarks

The ST\_Contains method tests if the `<geometry-expression>` completely contains `<geo2>` and there is one or more interior points of `<geo2>` that lies in the interior of the `<geometry-expression>`.

`<geometry-expression>.ST_Contains( <geo2> )` is equivalent to `<geo2>.ST_Within( <geometry-expression> )`.

The ST\_Contains and ST\_Covers methods are similar. The difference is that ST\_Covers does not require intersecting interior points.

### i Note

This method cannot be used with geometries in round-Earth spatial reference system.

### Example

The following example tests if a polygon contains a point. The polygon completely contains the point, and the interior of the point (the point itself) intersects the interior of the polygon, so the example returns 1.

```
SELECT NEW ST_Polygon( 'Polygon(( 0 0, 2 0, 1 2, 0 0 ))' ).ST_Contains( NEW  
ST_Point( 1, 1 ) ) FROM dummy;
```

The following example tests if a polygon contains a line. The polygon completely contains the line, but the interior of the line and the interior of the polygon do not intersect (the line only intersects the polygon on the polygon's boundary, and the boundary is not part of the interior), so the example returns 0. If ST\_Covers was used in place of ST\_Contains, ST\_Covers would return 1.

```
SELECT NEW ST_Polygon( 'Polygon(( 0 0, 2 0, 1 2, 0 0 ))' ).ST_Contains( NEW  
ST_LineString( 'LineString( 0 0, 1 0 )' ) ) FROM dummy;
```

### 3.1.13 ST\_ConvexHull Method

Returns the convex hull of the geometry value.

#### Syntax

```
<geometry-expression>.ST_ConvexHull()
```

#### Returns

**ST\_Geometry** If the geometry value is NULL or an empty value, then NULL is returned. Otherwise, the convex hull of the geometry value is returned.

The spatial reference system identifier of the result is the same as the spatial reference system of the `<geometry-expression>`.

#### Remarks

The convex hull of a geometry is the smallest convex geometry that contains all of the points in the geometry.

The convex hull may be visualized by imagining an elastic band stretched to enclose all of the points in the geometry. When released, the elastic band takes the shape of the convex hull.

If the geometry consists of a single point, the point is returned. If all of the points of the geometry lie on a single straight line segment, a linestring is returned. Otherwise, a convex polygon is returned.

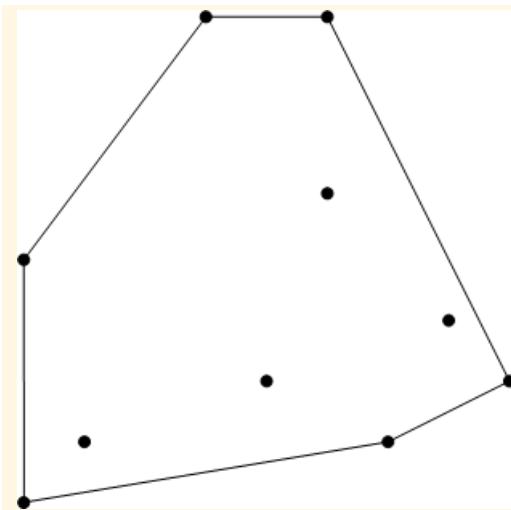
The convex hull can serve as an approximation of the original geometry. When testing a spatial relationship, the convex hull can serve as a quick pre-filter because if there is no spatial intersection with the convex hull then there can be no intersection with the original geometry.

##### i Note

If the `<geometry-expression>` is an empty geometry (`ST_IsEmpty()=1`), then this method returns NULL.

##### Example

The following example shows the convex hull computed from 10 points. The resulting hull is the result `Polygon ((1 1, 7 2, 9 3, 6 9, 4 9, 1 5, 1 1))`.



```
SELECT NEW ST_MultiPoint('MultiPoint( (1 1), (2 2), (5 3), (7 2), (9 3), (8 4), (6 6), (6 9), (4 9), (1 5) )').ST_ConvexHull().ST_AsText() FROM dummy;
```

The following example returns the single point (0,0). The convex hull of a single point is a point.

```
SELECT NEW ST_Point(0,0).ST_ConvexHull().ST_AsText() FROM dummy;
```

The following example returns the result `LineString (0 0, 3 3)`. The convex hull of a single straight line is a linestring with a single segment.

```
SELECT NEW ST_LineString('LineString(0 0,1 1,2 2,3 3)').ST_ConvexHull().ST_AsText() FROM dummy;
```

### 3.1.14 ST\_ConvexHullAggr Method

Returns the convex hull for all of the geometries in a group.

#### Syntax

```
ST_ConvexHullAggr(<geometry_column>)
```

## Parameters

Table 12:

Name	Type	Description
geometry_column	ST_Geometry	The geometry values to generate the convex hull. This is normally a column.

## Returns

**ST\_Geometry** Returns the convex hull for all the geometries in a group.

### Example

The following example returns: POLYGON ((10 3,10 10,6 7,4 5,5 3,10 3)).

```
SELECT ST_ConvexHullAggr( Shape ).ST_AsText() FROM SpatialShapes WHERE ShapeID  
in (11, 12);
```

## 3.1.15 ST\_CoordDim Method

Returns the number of coordinate dimensions stored with each point of the ST\_Geometry value.

## Syntax

```
<geometry-expression>.ST_CoordDim()
```

## Returns

**SMALLINT** Returns a value between 2 and 4 indicating the number of coordinate dimensions stored with each point of the ST\_Geometry value.

## Remarks

The ST\_CoordDim method returns the number of coordinates stored within each point in the geometry. All geometries have at least two coordinate dimensions. For geographic spatial reference systems, these are the

latitude and longitude of the point. For other spatial reference system, these coordinates are the X and Y positions of the point.

Geometries can optionally have Z and M values associated with each of the points in the geometry. These additional coordinate values are not considered when computing spatial relations or set operations, but they can be used to record additional information. For example, the measure value (M) can be used to record the pollution at various points within a geometry. The Z value usually is used to indicate elevation, but that interpretation is not imposed by the database server.

The following values may be returned by the ST\_CoordDim method:

- **2** - The geometry contains only two coordinates (either latitude/longitude or X/Y).
- **3** - The geometry contains one additional coordinate (either Z or M) for each point.
- **4** - The geometry contains two additional coordinate (both Z and M) for each point.

### i Note

Spatial operations that combine geometries using set operations do not preserve any Z or M values associated with the points of the geometry.

### i Note

By default, ST\_CoordDim uses the original format for a geometry, if it is available. Otherwise, the internal format is used. See the STORAGE FORMAT clause of the CREATE SPATIAL REFERENCE SYSTEM statement.

### Example

The following example returns the result 2.

```
SELECT NEW ST_Point(1.0, 1.0).ST_CoordDim() FROM dummy;
```

The following example returns the result 3.

```
SELECT NEW ST_Point('Point M (1 1 1)').ST_CoordDim() FROM dummy;
```

The following example returns the result 4.

```
SELECT NEW ST_Point('Point ZM (1 1 1 1)').ST_CoordDim() FROM dummy;
```

## Related Information

[CREATE SPATIAL REFERENCE SYSTEM Statement \[page 185\]](#)

## 3.1.16 ST\_CoveredBy Method

Tests if a geometry value is spatially covered by another geometry value.

### Syntax

```
<geometry-expression>.ST_CoveredBy(<geo2>)
```

### Parameters

Table 13:

Name	Type	Description
geo2	ST_Geometry	The other geometry value that is to be compared to the <geometry-expression>.

### Returns

**INTEGER** Returns 1 if the <geometry-expression> covers <geo2>, otherwise 0. If <geo2> is NULL, then NULL is returned.

### Remarks

The ST\_CoveredBy method tests if the <geometry-expression> is completely covered by <geo2>.

<geometry-expression>.ST\_CoveredBy( <geo2> ) is equivalent to <geo2>.ST\_Covers( <geometry-expression> ).

This predicate is similar to ST\_Within except for one subtle difference. The ST\_Within predicate requires that one or more interior points of the <geometry-expression> lie in the interior of <geo2>. For ST\_CoveredBy(), the method returns 1 if no point of the <geometry-expression> lies outside of <geo2>, regardless of whether interior points of the two geometries intersect. ST\_CoveredBy can be used with geometries in round-Earth spatial reference systems.



## Example

The following example tests if a point is covered by a polygon. The point is completely covered by the polygon so the example returns 1.

```
SELECT NEW ST_Point( 1, 1 ).ST_CoveredBy( NEW  
ST_Polygon( 'Polygon(( 0 0, 2 0, 1 2, 0 0 ))' ) ) FROM dummy;
```

The following example tests if a line is covered by a polygon. The line is completely covered by the polygon so the example returns 1. If ST\_Within was used in place of ST\_CoveredBy, ST\_Within would return 0.

```
SELECT NEW ST_LineString( 'LineString( 0 0, 1 0 )' ).ST_CoveredBy( NEW  
ST_Polygon( 'Polygon(( 0 0, 2 0, 1 2, 0 0 ))' ) ) FROM dummy;
```

The following example lists the ShapeIDs where the given point is within the Shape geometry. The following example lists the ShapeIDs where the given point is within the Shape geometry. This example returns the results 7, 13, and 14.

```
SELECT ShapeID FROM SpatialShapes WHERE NEW ST_Point( 4,  
4 ).ST_CoveredBy( Shape ) = 1 ORDER BY ShapeID;
```

## 3.1.17 ST\_Covers Method

Tests if a geometry value spatially covers another geometry value.

### Syntax

```
<geometry-expression>.ST_Covers(<geo2>)
```

### Parameters

Table 14:

Name	Type	Description
geo2	ST_Geometry	The other geometry value that is to be compared to the <geometry-expression>.

## Returns

**INTEGER** Returns 1 if the `<geometry-expression>` covers `<geo2>`, otherwise 0. If `<geo2>` is NULL, then NULL is returned.

## Remarks

The ST\_Covers method tests if the `<geometry-expression>` completely covers `<geo2>`. `<geometry-expression>.ST_Covers(<geo2>)` is equivalent to `<geo2>.ST_CoveredBy(<geometry-expression>)`.

This predicate is similar to ST\_Contains except for one subtle difference. The ST\_Contains predicate requires that one or more interior points of `<geo2>` lie in the interior of the geometry-expression. For ST\_Covers(), the method returns 1 if no point of `<geo2>` lies outside of the geometry-expression. Also, ST\_Covers can be used with geometries in round-Earth spatial reference systems, while ST\_Contains cannot.

### i Note

If the `<geometry-expression>` contains circularstrings, then these are interpolated to line strings.

### Example

The following example tests if a polygon covers a point. The polygon completely covers the point so the example returns 1.

```
SELECT NEW ST_Polygon( 'Polygon(( 0 0, 2 0, 1 2, 0 0 ))' )
    .ST_Covers( NEW ST_Point( 1, 1 ) ) FROM dummy;
```

The following example tests if a polygon covers a line. The polygon completely covers the line so the example returns 1. If ST\_Contains was used in place of ST\_Covers, ST\_Contains would return 0.

```
SELECT NEW ST_Polygon( 'Polygon(( 0 0, 2 0, 1 2, 0 0 ))' )
    .ST_Covers( NEW ST_LineString( 'LineString( 0 0, 1 0 )' ) ) FROM dummy;
```

The following example lists the ShapeIDs where the given polygon covers each Shape geometry. The following example lists the ShapeIDs where the given polygon covers each Shape geometry. This example returns the result 1.

```
SELECT ShapeID FROM SpatialShapes WHERE NEW ST_Polygon('Polygon ((-9 -9, 9 -9,
0 9, -9 -9), (-2 -2, -2 -1, -1 -1, -1 -2, -2 -2))').ST_Covers( Shape ) = 1
ORDER BY ShapeID;
```

## 3.1.18 ST\_Crosses Method

Tests if a geometry value crosses another geometry value.

### Syntax

```
<geometry-expression>.ST_Crosses(<geo2>)
```

### Parameters

Table 15:

Name	Type	Description
geo2	ST_Geometry	The other geometry value that is to be compared to the <geometry-expression>.

### Returns

**INTEGER** Returns 1 if the <geometry-expression> crosses <geo2>, otherwise 0. Returns NULL if <geometry-expression> is a polygon or multipolygon, or if <geo2> is a point or multipoint. If <geo2> is NULL, then NULL is returned.

### Remarks

Tests if a geometry value crosses another geometry value.

When both <geometry-expression> and <geo2> are linestrings or multilinestrings, they cross each other if their interiors intersect at one or more points. If the intersection results in a linestring or multilinestring, the geometries do not cross. If all of the intersecting points are boundary points, the geometries do not cross.

When <geometry-expression> has lower dimension than <geo2>, then <geometry-expression> crosses <geo2> if part of <geometry-expression> is on the interior of <geo2> and part of <geometry-expression> is on the exterior of <geo2>.

More precisely, <geometry-expression>.ST\_Crosses(<geo2>) returns 1 when the following is TRUE:

```
( <geometry-expression>.ST_Dimension() = 1 AND <geo2>.ST_Dimension() = 1 AND  
<geometry-expression>.ST_Relate( <geo2>, '0*****1' ) = 1 ) OR( <geometry-
```

```
<expression>.ST_Dimension() < <geo2>.ST_Dimension() AND <geometry-expression>.ST_Relate( <geo2>, 'T*T*****' ) = 1 )
```

### Example

The following example returns the result 1.

```
SELECT NEW ST_LineString( 'LineString( 0 0, 2 2 )' ).ST_Crosses( NEW ST_LineString( 'LineString( 0 2, 2 0 )' ) ) FROM dummy;
```

The following examples returns the result 0 because the interiors of the two lines do not intersect (the only intersection is at the first linestring boundary).

```
SELECT NEW ST_LineString( 'LineString( 0 1, 2 1 )' ).ST_Crosses( NEW ST_LineString( 'LineString( 0 0, 2 0 )' ) ) FROM dummy;
```

The following example returns NULL because the first geometry is a polygon.

```
SELECT NEW ST_Polygon( 'Polygon(( 0 0, 0 1, 1 0, 0 0 ))' ).ST_Crosses( NEW ST_LineString( 'LineString( 0 0, 2 0 )' ) ) FROM dummy;
```

## 3.1.19 ST\_Difference Method

Returns the geometry value that represents the point set difference of two geometries.

### Syntax

```
<geometry-expression>.ST_Difference(<geo2>)
```

### Parameters

Table 16:

Name	Type	Description
geo2	ST_Geometry	The other geometry value that is to be subtracted from the <geometry-expression>.

## Returns

**ST\_Geometry** Returns the geometry value that represents the point set difference of two geometries.

The spatial reference system identifier of the result is the same as the spatial reference system of the `<geometry-expression>`.

## Remarks

The ST\_Difference method finds the spatial difference of two geometries. A point is included in the result if it is present in the geometry-expression but not present in `<geo2>`.

Unlike other spatial set operations (ST\_Union, ST\_Intersection, and ST\_SymDifference), the ST\_Difference method is not symmetric: the method can give a different answer for `A.ST_Difference( B )` and `B.ST_Difference( A )`.

### Example

The following example shows the difference between the two objects.

```
SELECT NEW ST_Polygon( 'Polygon( (-1 -0.25, 1 -0.25, 1 2.25, -1 2.25, -1 -0.25) )' ).ST_Difference(NEW ST_Polygon('Polygon ((-5 -5, 5 -5, 0 5, -5 -5), (-2 -2, -2 0, 2 -2, -2 -2))')) from dummy;
```

## 3.1.20 ST\_Dimension Method

Returns the dimension of the ST\_Geometry value. Points have dimension 0, lines have dimension 1, and polygons have dimension 2. Any empty geometry has dimension -1.

## Syntax

```
<geometry-expression>.ST_Dimension()
```

## Returns

**SMALLINT** Returns the dimension of the `<geometry-expression>` as a SMALLINT between -1 and 2.

## Remarks

The ST\_Dimension method returns the spatial dimension occupied by a geometry. The following values may be returned:

- 1 The geometry corresponds to the empty set.
- 0 The geometry consists only of individual points (for example, an ST\_Point or ST\_MultiPoint).
- 1 The geometry contains at least one linestring and no polygons (for example, an ST\_LineString).
- 2 The geometry consists of at least one polygon (for example, an ST\_Polygon or ST\_MultiPolygon).

When computing the dimension of a collection, the largest dimension of any element is returned. For example, if a geometry collection contains a linestring and a point, ST\_Dimension returns 1 for the collection.

### Example

The following example returns the result 0.

```
SELECT NEW ST_Point(1.0,1.0).ST_Dimension() FROM dummy;
```

The following example returns the result 1.

```
SELECT NEW ST_LineString('LineString( 0 0, 1 1 )').ST_Dimension() FROM dummy;
```

## 3.1.21 ST\_Disjoint Method

Test if a geometry value is spatially disjoint from another value.

### Syntax

```
<geometry-expression>.ST_Disjoint(<geo2>)
```

## Parameters

Table 17:

Name	Type	Description
geo2	ST_Geometry	The other geometry value that is to be compared to the <code>&lt;geometry-expression&gt;</code> .

## Returns

**INTEGER** Returns 1 if the geometry-expression is spatially disjoint from `<geo2>`, otherwise 0. If `<geo2>` is NULL, then NULL is returned.

## Remarks

Tests if a geometry value is spatially disjoint from another value. Two geometries are disjoint if their intersection is empty. In other words, they are disjoint if there is no point anywhere in geometry-expression that is also in `<geo2>`."

`<geometry-expression>.ST_Disjoint( <geo2> ) = 1` is equivalent to `<geometry-expression>.ST_Intersects( <geo2> ) = 0`.

### i Note

If the `<geometry-expression>` contains circularstrings, then these are interpolated to line strings.

### ⊕ Example

The following example returns a result with one row for each shape that has no points in common with the specified triangle.

```
SELECT ShapeID FROM SpatialShapes WHERE
NEW ST_Polygon( 'Polygon((0 0, 5 0, 0 5, 0 0))' ).ST_Disjoint( Shape ) = 1
ORDER BY ShapeID;
```

The example returns the following SHAPEIDs: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 15.

## 3.1.22 ST\_Distance Method

Returns the smallest distance between the `<geometry-expression>` and the specified geometry value.

### Syntax

```
<geometry-expression>.ST_Distance(<geo2>,<unit_name>)
```

### Parameters

Table 18:

Name	Type	Description
geo2	ST_Geometry	The other geometry value whose distance is to be measured from the <code>&lt;geometry-expression&gt;</code> .
unit_name	VARCHAR(128)	The units in which the distance parameter should be interpreted. Defaults to the unit of the spatial reference system. The unit name must match the UNIT_NAME column of a row in the ST_UNITS_OF_MEASURE view where UNIT_TYPE is 'LINEAR'.

### Returns

**DOUBLE** Returns the smallest distance between the `<geometry-expression>` and `<geo2>`. If either `<geometry-expression>` or `<geo2>` is empty, then NULL is returned.

### Remarks

The ST\_Distance method computes the shortest distance between two geometries. For planar spatial reference systems, the distance is calculated as the Cartesian distance within the plane, computed in the linear units of measure for the associated spatial reference system. For round-Earth spatial reference systems, the distance is computed taking the curvature of the Earth's surface into account using the ellipsoid parameters in the spatial reference system definition.



## Example

The following returns the result 5.

```
SELECT NEW ST_Point(0, 0).ST_Distance( NEW ST_Point(5, 0) ) FROM dummy;
```

The following returns the result 5.

```
SELECT NEW ST_LineString('LineString(0 0, 10 0)').ST_Distance( NEW ST_Point(0, 5) ) FROM dummy;
```

The following returns the result 44,553338819890925.

```
SELECT NEW ST_Polygon('Polygon((-3 3, 3 3, 3 6, -3 6, -3 3))').ST_Distance( NEW ST_Point(10, 50) ) FROM dummy;
```

The following returns the distance between SAP headquarter to the Kalipeh (SRS 4326):  
352,6532242777006 (meter).

```
SELECT NEW ST_Point('POINT (8.641612 49.293703)', 4326).ST_Distance(NEW ST_Point('POINT (8.645850 49.295243)', 4326), 'meter') FROM dummy;
```

The following returns the distance between SAP headquarter to the Kalipeh (SRS 4326):  
385,66625577176353 (yard).

```
SELECT NEW ST_Point('POINT (8.641612 49.293703)', 4326).ST_Distance(NEW ST_Point('POINT (8.645850 49.295243)', 4326), 'yard') FROM dummy;
```

The following returns the distance between SAP headquarter to the Kalipeh (SRS 1000004326):  
501,0544104319325 (meter).

```
SELECT NEW ST_Point('POINT (8.641612 49.293703)', 1000004326).ST_Distance(NEW ST_Point('POINT (8.645850 49.295243)', 1000004326), 'meter') FROM dummy;
```

The following returns the distance between SAP headquarter to the Kalipeh (SRS 1000004326):  
547,9597664391213 (yard).

```
SELECT NEW ST_Point('POINT (8.641612 49.293703)', 1000004326).ST_Distance(NEW ST_Point('POINT (8.645850 49.295243)', 1000004326), 'yard') FROM dummy;
```

## 3.1.23 ST\_Envelope Method

Returns the bounding rectangle for the geometry value.

## Syntax

```
<geometry-expression>.ST_Envelope()
```

## Returns

**ST\_Polygon** Returns a polygon that is the bounding rectangle for the [`<geometry-expression>`](#).

The spatial reference system identifier of the result is the same as the spatial reference system of the [`<geometry-expression>`](#).

## Remarks

The ST\_Envelope method constructs a polygon that is an axis-aligned bounding rectangle for the [`<geometry-expression>`](#). The envelope covers the entire geometry, and it can be used as a simple approximation for the geometry.

### Note

If the [`<geometry-expression>`](#) is an empty geometry (`ST_IsEmpty()=1`), then this method returns NULL.

### Example

The following returns the result `Polygon ((0 0, 1 0, 1 1, 0 1, 0 0))` in binary format.

```
SELECT NEW ST_LineString('LineString(0 0, 1 1)').ST_Envelope() FROM dummy;
```

## 3.1.24 ST\_EnvelopeAggr Method

Returns the bounding rectangle for all of the geometries in a group.

## Syntax

```
ST_EnvelopeAggr(<geometry_column>)
```

## Parameters

Table 19:

Name	Type	Description
geometry_column	ST_Geometry	The geometry values to generate the bounding rectangle. This is normally a column.

## Returns

**ST\_Polygon** Returns a polygon that is the bounding rectangle for all the geometries in a group.

### Example

The following example returns POLYGON ((2.5 3, 9 3, 9 7, 2.5 7, 2.5 3)).

```
SELECT ST_EnvelopeAggr( Shape ).ST_AsText() FROM SpatialShapes WHERE ShapeID <= 10;
```

## 3.1.25 ST\_Equals Method

Tests if an ST\_Geometry value is spatially equal to another ST\_Geometry value.

## Syntax

```
<geometry-expression>.ST_Equals(<geo2>)
```

## Parameters

Table 20:

Name	Type	Description
geo2	ST_Geometry	The other geometry value that is to be compared to the <code>&lt;geometry-expression&gt;</code> .

## Returns

**INTEGER** Returns 1 if the two geometry values are spatially equal, otherwise 0. If `<geo2>` is NULL, then NULL is returned.

## Remarks

Tests if an ST\_Geometry value is equal to another ST\_Geometry value.

The test for spatial equality is performed by first comparing the bounding rectangles of the two geometries. If they are not equal within tolerance, the two geometries are considered not to be equal, and 0 is returned. Otherwise, the database server returns 1 if `<geometry-expression>.ST_SymDifference( geo2 )` is the empty set, otherwise it returns 0.

The SQL/MM standard defines ST\_Equals only in terms of ST\_SymDifference, without an additional bounding box comparison. There are some geometries that generate an empty result with ST\_SymDifference while their bounding boxes are not equal. These geometries would be considered equal by the SQL/MM standard but are not considered equal in the software. This difference can arise if one or both geometries contain spikes or punctures.

Two geometry values can be considered equal even though they have different representations. For example, two linestrings may have opposite orientations but contain the same set of points in space. These two linestrings are considered equal by ST\_Equals but not by ST\_OrderingEquals.

ST\_Equals may be limited by the resolution of the spatial reference system or the accuracy of the data.

### Example

The following example returns the result 1, indicating that the two linestrings are equal even though they contain a different number of points specified in a different order, and the intermediate point is not exactly on the line. The intermediate point is about 3.33e-7 away from the line with only two points, but that distance less than the tolerance 1e-6 for the "Default" spatial reference system (SRID 0).

```
SELECT NEW ST_LineString( 'LineString( 0 0, 0.333333 1, 1  
3 )' ).ST_Equals( NEW ST_LineString( 'LineString( 1 3, 0 0 )' ) ) FROM dummy;
```

## 3.1.26 ST\_GeomFromEWKB Method

Constructs a geometry from a string representation.

## Syntax

```
ST_GeomFromEWKB( <ewkb> )
```

## Parameters

Table 21:

Name	Type	Description
ewkb	VARBINARY	A string containing the extended WKB representation of a geometry.

## Returns

**ST\_Geometry** Returns a geometry value of the appropriate type based on the source string

## Remarks

Parses a string containing an extended WKB representation of a geometry and creates a geometry value of the appropriate type. This method is used by the server when evaluating a cast from a hexadecimal character string to a geometry type.

## Example

The following example returns the result POINT (0 0).

### 3.1.27 ST\_GeomFromEWKT Method

Constructs a geometry from a string representation

## Syntax

`ST_GeomFromEWKT(<wkt>)`

## Parameters

Table 22:

Name	Type	Description
wkt	VARCHAR	A string containing the extended WKT representation of a geometry value.

## Returns

**ST\_Geometry** Returns a geometry value of the appropriate type based on the source string.

## Remarks

Parses a string containing an extended WKT representation of a geometry and creates a geometry value of the appropriate type. This method is used by the server when evaluating a cast from a character string to a geometry type.

### Example

The following example returns the point POINT (2 4).

```
SELECT ST_GeomFromEWKT('SRID=0;POINT (2.0 4.0)').ST_AsText() FROM dummy;
```

## 3.1.28 ST\_GeomFromText Method

Constructs a geometry from a character string representation.

## Syntax

```
ST_GeomFromText(<character-string>[, <srid>])
```

## Parameters

Table 23:

Name	Type	Description
character-string	LONG VARCHAR	A string containing the text representation of a geometry.
srid	INT	The SRID of the result. If not specified and the input string does not contain a SRID, the default is 0.

## Returns

**ST\_Geometry** Returns a geometry value of the appropriate type based on the source string.

The spatial reference system identifier of the result is given by parameter `<srid>`.

## Remarks

Parses a text string representing a geometry and creates a geometry value of the appropriate type. This method is used by the server when evaluating a cast from a character string to a geometry type.

The server detects the format of the input string. Some input formats contain a SRID definition. If provided, the `<srid>` parameter must match any value taken from the input string.

### Example

The following example returns the result `LineString (1 2, 5 7)`.

```
SELECT ST_GeomFromText( 'LineString( 1 2, 5 7 )', 4326 ).ST_AsText() FROM dummy;
```

## Related Information

[ST\\_GeomFromText Method \[page 66\]](#)

[ST\\_GeomFromWKT Method \[page 69\]](#)

## 3.1.29 ST\_GeomFromWKB Method

Constructs a geometry from a string representation.

### Syntax

```
ST_GeomFromWKB (<wkb>[, <srid>])
```

### Parameters

Table 24:

Name	Type	Description
wkb	VARBINARY	A string containing the WKB representation of a geometry value.
srid	INTEGER	The SRID of the result. If not specified, the default is 0.

### Returns

**ST\_Geometry** Returns a geometry value of the appropriate type based on the source string.

The spatial reference system identifier of the result is given by parameter `<srid>`.

### Remarks

Parses a string containing a WKB representation of a geometry and creates a geometry value of the appropriate type. This method is used by the server when evaluating a cast from a hexadecimal character string to a geometry type. Some input formats contain an SRID definition.

#### Example

The following example returns the result `POINT (2 4)`.

```
SELECT
  ST_GeomFromWKB(x'01010000000000000000000000000040000000000000001040').ST_AsText() FROM
  dummy;
```

## 3.1.30 ST\_GeomFromWKT Method

Constructs a geometry from a string representation.

### Syntax

```
ST_GeomFromWKT (<wkt>[, <srid>])
```

### Parameters

Table 25:

Name	Type	Description
wkt	VARCHAR	A string containing the WKT representation of a geometry value.
srid	INTEGER	The SRID of the result. If not specified, the default is 0.

### Returns

**ST\_Geometry** Returns a geometry value of the appropriate type based on the source string.

The spatial reference system identifier of the result is given by parameter `<srid>`.

### Remarks

Parses a string containing a WKT representation of a geometry and creates a geometry value of the appropriate type. This method is used by the server when evaluating a cast from a character string to a geometry type. Some input formats contain an SRID definition.

#### Example

The following example returns the result `POINT (2 4)`.

```
SELECT ST_GeomFromWKT('POINT (2.0 4.0)').ST_AsText() FROM dummy;
```

## Related Information

[ST\\_GeomFromText Method \[page 66\]](#)

### 3.1.31 ST\_GeometryType Method

Returns the name of the type of the ST\_Geometry value.

#### Syntax

```
<geometry-expression>.ST_GeometryType()
```

#### Returns

**VARCHAR(128)** Returns the data type of the geometry value as a text string. This method can be used to determine the dynamic type of a value.

#### Remarks

The ST\_GeometryType method returns a string containing the specific type name of <geometry-expression>.

#### Example

The following returns the result ST\_Point.

```
SELECT NEW ST_Point( 1, 2 ).ST_GeometryType() FROM dummy;
```

## 3.1.32 ST\_Intersection Method

Returns the geometry value that represents the point set intersection of two geometries.

### Syntax

```
<geometry-expression>.ST_Intersection(<geo2>)
```

### Parameters

Table 26:

Name	Type	Description
geo2	ST_Geometry	The other geometry value that is to be intersected with the <geometry-expression>.

### Returns

**ST\_Geometry** Returns the geometry value that represents the point set intersection of two geometries.

The spatial reference system identifier of the result is the same as the spatial reference system of the <geometry-expression>.

### Remarks

The ST\_Intersection method finds the spatial intersection of two geometries. A point is included in the intersection if it is present in both of the input geometries. If the two geometries don't share any common points, the result is an empty geometry.

#### Example

The following intersection between a triangle and a square returns the result POLYGON ((1 1, 3 1, 3 2, 2.5 3, 1.5 3, 1 2, 1 1)), a six-sided polygon.

```
SELECT NEW ST_Polygon('Polygon ((0 0, 4 0, 2 4, 0 0))').ST_Intersection( NEW ST_Polygon('Polygon ((1 1, 3 1, 3 3, 1 3, 1 1))') ).ST_AsText() FROM dummy;
```

### 3.1.33 ST\_Intersects Method

Test if a geometry value spatially intersects another value.

#### Syntax

```
<geometry-expression>.ST_Intersects(<geo2>)
```

#### Parameters

Table 27:

Name	Type	Description
geo2	ST_Geometry	The other geometry value that is to be compared to the <geometry-expression>.

#### Returns

**INTEGER** Returns 1 if the <geometry-expression> spatially intersects with <geo2>, otherwise 0. If <geo2> is NULL, then NULL is returned.

#### Remarks

Tests if a geometry value spatially intersects another value. Two geometries intersect if they share one or more common points.

##### Example

The following returns the result 1 because the two linestrings intersect.

```
SELECT NEW ST_LineString('LineString(0 0,2 0)').ST_Intersects( NEW  
ST_LineString('LineString(1 -1,1 1)' ) ) FROM dummy;
```

The following returns the result 0 because the two linestrings do not intersect.

```
SELECT NEW ST_LineString('LineString(0 0,2 0)').ST_Intersects( NEW  
ST_LineString('LineString(1 1,1 2)' ) ) FROM dummy;
```

### 3.1.34 ST\_IntersectionAggr Method

Returns the spatial intersection of all of the geometries in a group.

#### Syntax

```
ST_IntersectionAggr(<geometry_column>)
```

#### Parameters

Table 28:

Name	Type	Description
geometry_column	ST_Geometry	The geometry values to generate the spatial intersection. This is normally a column.

#### Returns

**ST\_Geometry** Returns a geometry that is the spatial intersection for all the geometries in a group.

#### Example

The following example returns an empty geometry collection, because there is no spatial intersection for all the geometries in the selected group.

```
SELECT ST_IntersectionAggr( Shape ).ST_AsText() FROM SpatialShapes WHERE  
ShapeID IN ( 11, 12 );
```

The following example returns: POLYGON ((1 4,1 3,5 3,5 4,1 4)).

```
SELECT ST_IntersectionAggr( Shape ).ST_AsText() FROM SpatialShapes WHERE  
ShapeID IN ( 13, 14 );
```

### 3.1.35 ST\_IntersectsFilter Method

A quick test if the two geometries intersect.

#### Syntax

```
<geometry-expression>.ST_IntersectsFilter(<geo2>)
```

#### Parameters

Table 29:

Name	Type	Description
geo2	ST_Geometry	The other geometry value that is to be compared to the <geometry-expression>.

#### Returns

**INTEGER** Returns 1 if the <geometry-expression> spatially intersects with <geo2>, otherwise 0. If <geo2> is NULL, then NULL is returned.

### 3.1.36 ST\_IntersectsRect Method

Test if a geometry intersects a rectangle.

#### Syntax

```
<geometry-expression>.ST_IntersectsRect(<pmin>,<pmax>)
```

## Parameters

Table 30:

Name	Type	Description
pmin	ST_Point	The minimum point value that is to be compared to the <geometry-expression>.
pmax	ST_Point	The maximum point value that is to be compared to the <geometry-expression>.

## Returns

**INTEGER** Returns 1 if the <geometry-expression> intersects with the specified rectangle, otherwise 0. If <pmin> or <pmax> is NULL, then NULL is returned.

## Remarks

The ST\_IntersectsRect method tests if a geometry intersects with a specified axis-aligned bounding rectangle.

Therefore, this method can be useful for writing window queries to find all geometries that intersect a given axis-aligned rectangle.

### Example

The following returns the result 0 because the linestring does not intersect the provided window.

```
SELECT NEW ST_LineString( 'LineString( 0 0, 10 0, 10  
10 )' ).ST_IntersectsRect( NEW ST_Point( 4, 4 ), NEW ST_Point( 6, 6 ) ) FROM  
dummy;
```

The following returns the result 1 because the linestring does intersect the provided window.

```
SELECT NEW ST_LineString( 'LineString( 0 4, 10 4, 10  
10 )' ).ST_IntersectsRect( NEW ST_Point( 4, 4 ), NEW ST_Point( 6, 6 ) ) FROM  
dummy;
```

### 3.1.37 ST\_IntersectsRectPlanar Method

Test if intersects a rectangle.

#### Syntax

```
<geometry-expression>.ST_IntersectsRectPlanar(<pmin>,<pmax>)
```

#### Parameters

Table 31:

Name	Type	Description
pmin	ST_Point	The minimum point value to be compared to the <geometry-expression>.
pmax	ST_Point	The maximum point value to be compared to the <geometry-expression>.

#### Returns

**INTEGER** Returns 1 if the <geometry-expression> intersects with the specified rectangle, otherwise 0. If <pmin> or <pmax> is NULL, then NULL is returned.

#### Remarks

In planar spatial reference systems (see *Spatial Reference Systems (SRS) and Spatial Reference Identifiers (SRID)*), this method returns the same result as ST\_IntersectsRect.

In round-earth spatial reference systems, this method re-interprets <geometry-expression> as planar and checks if the expression intersects the rectangle, which is also re-interpreted as planar. This method can be useful to perform a window query if round-earth geometries are displayed as flat geometries.

## Example

Returns 0 as the line string crosses the dateline in round-earth reference systems and does not contain the point (0, 0).

```
SELECT ST_GeomFromText('LineString(179 0, -179 0)',  
4326).ST_IntersectsRect(ST_GeomFromText('POINT(-1 -1)', 4326),  
ST_GeomFromText('POINT(1 1)', 4326)) FROM DUMMY;
```

Returns 1 as the line-string is re-interpreted as flat geometry. It contains the point (0, 0) and does not cross the dateline anymore.

```
SELECT ST_GeomFromText('LineString(179 0, -179 0)',  
4326).ST_IntersectsRectPlanar(ST_GeomFromText('POINT(-1 -1)', 4326),  
ST_GeomFromText('POINT(1 1)', 4326)) FROM DUMMY;
```

## Related Information

[Spatial Reference Systems \(SRS\) and Spatial Reference Identifiers \(SRID\) \[page 9\]](#)

### 3.1.38 ST\_Is3D Method

Determines if the geometry value has Z coordinate values.

## Syntax

```
<geometry-expression>.ST_Is3D()
```

## Returns

**INTEGER** Returns 1 if the geometry value has Z coordinate values, otherwise 0. If `<geometry-expression>` is NULL, then NULL is returned.

## Example

The following returns the result 0 because the point does not contain a Z value.

```
SELECT NEW ST_POINT('POINT(0 0)').ST_Is3D() FROM dummy;
```

The following returns the result 1.

```
SELECT NEW ST_POINT('POINT Z(0 0 0)').ST_Is3D() FROM dummy;
```

The following returns the result 0 because the point does not contain a Z value.

```
SELECT NEW ST_POINT('POINT M(0 0 0)').ST_Is3D() FROM dummy;
```

The following returns the result 1.

```
SELECT NEW ST_POINT('POINT ZM(0 0 0 0)').ST_Is3D() FROM dummy;
```

### 3.1.39 ST\_IsEmpty Method

Determines whether the geometry value represents an empty set.

#### Syntax

```
<geometry-expression>.ST_IsEmpty()
```

#### Returns

**INTEGER** Returns 1 if the geometry value is empty, otherwise 0. If `<geometry-expression>` is NULL, then NULL is returned.

#### Example

The following example returns the result 1.

```
SELECT NEW ST_LineString().ST_IsEmpty() FROM dummy;
```

## 3.1.40 ST\_IsMeasured Method

Determines if the geometry value has associated measure values.

### Syntax

```
<geometry-expression>.ST_IsMeasured()
```

### Returns

**INTEGER** Returns 1 if the geometry value has measure values, otherwise 0. If `<geometry-expression>` is NULL, then NULL is returned.

#### Example

The following example returns the result 1.

```
SELECT ST_GeomFromText( 'LineString M( 1 2 4, 5 7 3 )' ).ST_IsMeasured() FROM dummy;
```

The following example returns the result 0.

```
SELECT ST_GeomFromText( 'LineString Z( 1 2 4, 5 7 3 )' ).ST_IsMeasured() FROM dummy;
```

The following example returns the result 0.

```
SELECT count(*) FROM SpatialShapes WHERE Shape.ST_IsMeasured() = 1;
```

## 3.1.41 ST\_IsSimple Method

Determines whether the geometry value is simple (containing no self-intersections or other irregularities).

### Syntax

```
<geometry-expression>.ST_IsSimple()
```

## Returns

**INTEGER** Returns 1 if the geometry value is simple, otherwise 0. If `<geometry-expression>` is NULL, then NULL is returned.

### Example

The following returns the result 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15.

```
SELECT Shape.ST_IsSimple() FROM SpatialShapes;
```

## 3.1.42 ST\_IsValid Method

Determines whether the geometry is a valid spatial object.

## Syntax

```
<geometry-expression>.ST_IsValid()
```

## Returns

**INTEGER** Returns 1 if the geometry value is valid, otherwise 0. If `<geometry-expression>` is NULL, then NULL is returned.

## Remarks

By default, the server does not validate spatial data as it is created or imported from other formats. The `ST_IsValid` method can be used to verify that the imported data represents a geometry that is valid. Operations on invalid geometries return undefined results.

### Example

The following returns the result 0 because the polygon contains a bow tie (the ring has a self-intersection).

```
SELECT NEW ST_Polygon( 'Polygon(( 0 0, 4 0, 4 5, 0 -1, 0 0 ))' ).ST_IsValid()  
FROM dummy;
```

The following returns the result 0 because the polygons within the geometry self-intersect at a polygon. Self-intersections of a geometry collection at finite number of points is considered valid.

```
SELECT NEW ST_MultiPolygon( 'MultiPolygon((( 0 0, 2 0, 1 2, 0 0 )),((0 2, 1 0, 2 2, 0 2)))' ).ST_IsValid() FROM dummy;
```

### 3.1.43 ST\_MMax Method

Retrieves the maximum M coordinate value of a geometry.

#### Syntax

```
<geometry-expression>.ST_MMax()
```

#### Returns

**DOUBLE** Returns the maximum M coordinate value of the `<geometry-expression>`.

#### Example

The following example returns the result 8.

```
SELECT NEW ST_LineString( 'LineString ZM( 1 2 3 4, 5 6 7 8 )' ).ST_MMax() FROM dummy;
```

### 3.1.44 ST\_MMin Method

Retrieves the minimum M coordinate value of a geometry.

#### Syntax

```
<geometry-expression>.ST_MMin()
```

## Returns

**DOUBLE** Returns the minimum M coordinate value of the <geometry-expression>.

### Example

The following example returns the result 4.

```
SELECT NEW ST_LineString( 'LineString ZM( 1 2 3 4, 5 6 7 8 )' ).ST_MMin() FROM dummy;
```

## 3.1.45 ST\_NumInteriorRing Method

Returns the number of interior rings in the polygon.

## Syntax

```
<polygon-expression>.ST_NumInteriorRing()
```

## Returns

**INT** Returns the number of interior rings in the polygon.

### Example

The following example returns the result 1.

```
SELECT NEW ST_Polygon('Polygon ((0 0, 3 0, 3 3, 0 3, 0 0), (1 1, 2 1, 2 2, 1 1))').ST_NumInteriorRing() FROM dummy;
```

## Related Information

[ST\\_NumInteriorRings Method \[page 83\]](#)

## 3.1.46 ST\_NumInteriorRings Method

Returns the number of interior rings in the polygon.

### Syntax

```
<polygon-expression>.ST_NumInteriorRings()
```

### Returns

**INT** Returns the number of interior rings in the polygon.

#### Example

The following example returns the result 1.

```
SELECT NEW ST_Polygon('Polygon ((0 0, 3 0, 3 3, 0 3, 0 0), (1 1, 2 1, 2 2, 1 1))').ST_NumInteriorRings() FROM dummy;
```

### Related Information

[ST\\_NumInteriorRing Method \[page 82\]](#)

## 3.1.47 ST\_OrderingEquals Method

Tests if a geometry is identical to another geometry.

### Syntax

```
<geometry-expression>.ST_OrderingEquals(<geo2>)
```

## Parameters

Table 32:

Name	Type	Description
geo2	ST_Geometry	The other geometry value that is to be compared to the <geometry-expression>.

## Returns

**INTEGER** Returns 1 if the two geometry values are exactly equal, otherwise 0. If <geo2> is NULL, then NULL is returned.

## Remarks

Tests if an ST\_Geometry value is identical to another ST\_Geometry value. The two geometries must contain the same hierarchy of objects with the exact same points in the same order to be considered equal under ST\_OrderingEquals.

The ST\_OrderingEquals method differs from ST\_Equals in that it considers the orientation of linestrings. Two linestrings can contain exactly the same points but in opposite orders. These two linestrings are considered equal with ST\_Equals but unequal with ST\_OrderingEquals. Additionally, ST\_OrderingEquals requires that each point in both geometries is exactly equal, not just equal within the tolerance-distance specified by the spatial reference system.

The ST\_OrderingEquals method defines the semantics used for comparison predicates (= and <>), IN list predicates, DISTINCT, and GROUP BY. If you wish to compare if two spatial values are spatially equal (contain the same set of points in set), you can use the ST\_Equals method.

### i Note

The SQL/MM standard defines ST\_OrderingEquals to return a relative ordering, with 0 returned if two geometries are spatially equal (according to ST\_Equals) and 1 if they are not equal. The software follows industry practice and differs from SQL/MM in that it returns a Boolean with 1 indicating the geometries are equal and 0 indicating they are different. Further, the ST\_OrderingEquals implementation differs from SQL/MM because it tests that values are identical (same hierarchy of objects in the same order) instead of spatially equal (same set of points in space).



## Example

The following returns the result 1 because the two linestrings contain exactly the same points in the same order.

```
SELECT NEW ST_LineString('LineString(0 0, 1 0)').ST_OrderingEquals(NEW  
ST_LineString('LineString(0 0, 1 0)')) FROM dummy;
```

The following returns the result 0 because the two linestrings are defined in different orders (even though the two linestrings are spatially equal (ST\_Equals is 1)).

```
SELECT NEW ST_LineString('LineString(0 0, 1 0)').ST_OrderingEquals(NEW  
ST_LineString('LineString(1 0, 0 0)')) FROM dummy;
```

### 3.1.48 ST\_Overlaps Method

Tests if a geometry value overlaps another geometry value.

#### Syntax

```
<geometry-expression>.ST_Overlaps(<geo2>)
```

#### Parameters

Table 33:

Name	Type	Description
geo2	ST_Geometry	The other geometry value that is to be compared to the <geometry-expression>.

#### Returns

**INTEGER** Returns 1 if the <geometry-expression> overlaps <geo2>, otherwise 0. Returns NULL if <geometry-expression> and <geo2> have different dimensions. If <geometry-expression> or <geo2> is NULL, then NULL is returned.

## Remarks

Two geometries overlap if the following conditions are all true:

- Both geometries have the same dimension.
- The intersection of <geometry-expression> and <geo2> geometries has the same dimension as <geometry-expression>.
- Neither of the original geometries is a subset of the other.

More precisely, <geometry-expression>.ST\_Overlaps(<geo2>) returns 1 when the following is TRUE:

```
<geometry-expression>.ST_Dimension() = <geo2>.ST_Dimension()
AND <geometry-expression>.ST_Intersection( <geo2> ).ST_Dimension() =
<geometry-expression>.ST_Dimension()
AND <geometry-expression>.ST_Covers( <geo2> ) = 0
AND <geo2>.ST_Covers( <geometry-expression> ) = 0
```

### i Note

If the <geometry-expression> contains circularstrings, then these are interpolated to line strings.

### i Note

This method cannot be used with geometries in round-Earth spatial reference system.



### Example

The following returns the result 1 since the intersection of the two linestrings is also a linestring, and neither geometry is a subset of the other.

```
SELECT NEW ST_LineString( 'LineString( 0 0, 5 0 )' ).ST_Overlaps( NEW
ST_LineString( 'LineString( 2 0, 3 0, 3 3 )' ) ) FROM dummy;
```

The following returns the result NULL since the linestring and point have different dimension.

```
SELECT NEW ST_LineString( 'LineString( 0 0, 5 0 )' ).ST_Overlaps( NEW
ST_Point( 1, 0 ) ) FROM dummy;
```

The following returns the result 0 since the point is a subset of the multipoint.

```
SELECT NEW ST_MultiPoint( 'MultiPoint(( 2 3 ), ( 1 0 ))' ).ST_Overlaps( NEW
ST_Point( 1, 0 ) ) FROM dummy;
```

The following returns the result 12, 14.

```
SELECT ShapeID FROM SpatialShapes WHERE NEW ST_Polygon('Polygon(( 3 3, 3 6, 6
6, 3 3 ))').ST_Overlaps( Shape ) = 1 ORDER BY ShapeID;
```

### 3.1.49 ST\_Perimeter Method

Computes the perimeter of the multi-surface in the specified unit.

#### Syntax

```
<multisurface-expression>.ST_Perimeter([<unit_name>])
```

#### Parameters

Table 34:

Name	Type	Description
unit_name	VARCHAR(128)	The unit in which the perimeter should be computed. Defaults to the unit of the spatial reference system. The unit name must match the UNIT_NAME column of a row in the ST_UNITS_OF_MEASURE view where UNIT_TYPE is 'LINEAR'.

#### Returns

**DOUBLE** Returns the perimeter of the multi-surface.

#### Remarks

The ST\_Perimeter method returns the length of the perimeter of a multi-surface in the units identified by the `<unit_name>` parameter.

If the multi-surface contains Z values, these are not considered when computing the perimeter of the geometry.

The perimeter of a polygon includes the length of all rings (exterior and interior).

##### i Note

If the `<multisurface-expression>` is an empty geometry (`ST_IsEmpty()=1`), then this method returns `NULL`.

### Note

By default, ST\_Perimeter uses the original format for a geometry, if it is available. Otherwise, the internal format is used. See the STORAGE FORMAT clause of the CREATE SPATIAL REFERENCE SYSTEM statement.

### Example

The following example creates a multi-surface containing two polygons and uses ST\_Perimeter to find the length of the perimeter, returning the result 44.

```
SELECT NEW ST_MultiPolygon('MultiPolygon (((0 0, 1 0, 1 1, 0 1, 0 0)), ((10  
10, 20 10, 20 20, 10 20, 10 10)))').ST_Perimeter() FROM dummy;
```

The following example creates a multi-surface containing two polygons and uses ST\_Perimeter to find the length of the perimeter, returning the result 48,118985126859144.

```
SELECT NEW ST_MultiPolygon('MultiPolygon (((0 0, 1 0, 1 1, 0 1, 0 0)), ((10 10,  
20 10, 20 20, 10 20, 10 10)))').ST_Perimeter('yard') FROM dummy;
```

## 3.1.50 ST\_PointOnSurface Method

Returns an ST\_Point value that is guaranteed to spatially intersect the ST\_Surface value.

### Syntax

```
<surface-expression>.ST_PointOnSurface()
```

### Returns

**ST\_Point** If the surface is the empty set, returns NULL. Otherwise, returns an ST\_Point value guaranteed to spatially intersect the surface.

### Example

The following returns a point that intersects the polygon.

```
SELECT NEW ST_Polygon('POLYGON((0 0, 0 1.25, 1.25 1.25, 1.25  
0 0))').ST_PointOnSurface().ST_AsText() FROM dummy;
```

## 3.1.51 ST\_Relate Method

Tests if a geometry value is spatially related to another geometry value as specified by the intersection matrix.

The ST\_Relate method uses a 9-character string from the Dimensionally Extended 9 Intersection Model (DE-9IM) to describe the pair-wise relationship between two spatial data items. For example, the ST\_Relate method determines if an intersection occurs between the geometries, and the geometry of the resulting intersection, if it exists.

### Syntax

```
<geometry-expression>.ST_Relate (<geo2>)
```

### Parameters

Table 35:

Name	Type	Description
geo2	ST_Geometry	The second geometry value that is to be compared to the <geometry-expression>.

### Returns

**VARCHAR** Returns a 9-character string representing a matrix in the dimensionally extended 9 intersection model. Each character in the 9-character string represents the type of intersection at one of the nine possible intersections between the interior, boundary, and exterior of the two geometries. If <geo2> is NULL, then NULL is returned.

### Remarks

Tests if a geometry value is spatially related to another geometry value by testing for intersection between the interior, boundary, and exterior of two geometries, as specified by the intersection matrix.

The intersection matrix is returned as a string. While it is possible to use this method variant to test a spatial relationship by examining the returned string, it is more efficient to test relationships by passing a pattern string as second parameter or by using specific spatial predicates such as ST\_Contains or ST\_Intersects.

### Note

If the <geometry-expression> contains circularstrings, then these are interpolated to line strings.

### Note

This method cannot be used with geometries in round-Earth spatial reference system.

### Example

The following example returns the result 1F2001102.

```
SELECT NEW ST_Polygon( 'Polygon(( 0 0, 2 0, 0 2, 0 0 ))' )  
.ST_Relate( NEW ST_LineString( 'LineString( 0 1, 5 1 )' ) ) from dummy;
```

## Related Information

[How Spatial Relationships Work \[page 29\]](#)

## 3.1.52 ST\_SRID Method

Returns the SRID of the geometry.

### Syntax

```
<geometry-expression>.ST_SRID()
```

### Returns

**INT** Returns the SRID of the geometry.

### Remarks

Returns the SRID of the geometry. Every geometry is associated with a spatial reference system.

### Example

The following returns the result 0, indicating the point has the SRID 0, corresponding to the 'Default' spatial reference system.

```
SELECT NEW ST_Point().ST_SRID() FROM dummy;
```

## 3.1.53 ST\_SRID(INT) Method

Changes the spatial reference system associated with the geometry without modifying any of the values.

### Syntax

```
<geometry-expression>.ST_SRID([<srid>])
```

### Parameters

Table 36:

Name	Type	Description
srid	INT	The SRID to use for the result.

### Returns

**ST\_Geometry** Returns a copy of the geometry value with the specified spatial reference system.

The spatial reference system identifier of the result is given by parameter `<srid>`.

### Remarks

The ST\_SRID method creates a copy of a `<geometry-expression>` with the SRID specified by srid parameter. ST\_SRID can be used when both the source and destination spatial reference systems have the same coordinate system.

If you are transforming a geometry between two spatial reference systems that have different coordinate systems, you should use the ST\_Transform method.

### Note

By default, ST\_SRID uses the original format for a geometry, if it is available. Otherwise, the internal format is used. See the STORAGE FORMAT clause of the CREATE SPATIAL REFERENCE SYSTEM statement.

### Example

The following example returns the result SRID=1000004326; Point (-118 34).

```
SELECT NEW ST_Point('Point (-118 34)', 4326).ST_SRID( 1000004326 ).ST_AsEWKT()  
FROM dummy;
```

## 3.1.54 ST\_SnapToGrid Method

Returns a copy of the geometry with all points snapped to the specified grid.

### Syntax

```
<geometry-expression>.ST_SnapToGrid(<cell-size>)
```

### Parameters

Table 37:

Name	Type	Description
cell-size	DOUBLE	The cell size for the grid.

### Returns

**ST\_Geometry** Returns the geometry with all points snapped to the grid.

The spatial reference system identifier of the result is the same as the spatial reference system of the `<geometry-expression>`.

## Remarks

The ST\_SnapToGrid method can be used to reduce the precision of data by snapping all points in a geometry to a grid defined by the origin and cell size.

The X and Y coordinates are snapped to the grid; any Z and M values are unchanged.

### Note

Reducing precision may cause the resulting geometry to have different properties. For example, it may cause a simple linestring to cross itself, or it may generate an invalid geometry.

### Note

Each spatial reference system defines a grid that all geometries are automatically snapped to. You cannot store more precision than this predefined grid.

### Example

The following example returns the result LineString (1.536133 6.439453, 2.173828 6.100586).

```
SELECT NEW ST_LineString( 'LineString( 1.5358 6.4391, 2.17401  
6.10018 )' ).ST_SnapToGrid( 0.001 ).ST_AsText() FROM dummy;
```

Each X and Y coordinate is shifted to the closest grid point using a grid size of approximately 0.001. The actual grid size used is not exactly the grid size specified.

## 3.1.55 ST\_SymDifference Method

Returns the geometry value that represents the point set symmetric difference of two geometries.

### Syntax

```
<geometry-expression>.ST_SymDifference(<geo2>)
```

## Parameters

Table 38:

Name	Type	Description
geo2	ST_Geometry	The other geometry value that is to be subtracted from the <a href="#"><code>&lt;geometry-expression&gt;</code></a> to find the symmetric difference.

## Returns

**ST\_Geometry** Returns the geometry value that represents the point set symmetric difference of two geometries.

The spatial reference system identifier of the result is the same as the spatial reference system of the [`<geometry-expression>`](#).

## Remarks

The ST\_SymDifference method finds the symmetric difference of two geometries. The symmetric difference consists of all of those points that are in only one of the two geometries. If the two geometry values consist of the same points, the ST\_SymDifference method returns an empty geometry.

### Example

The following example shows the difference between the two objects.

```
SELECT NEW ST_Polygon( 'Polygon( -1 -0.25, 1 -0.25, 1 2.25, -1 2.25, -1 -0.25 )' ).ST_SymDifference(NEW ST_Polygon('Polygon ((-5 -5, 5 -5, 0 5, -5 -5), (-2 -2, -2 0, 2 0, 2 -2, -2 -2))')) from dummy;
```

## 3.1.56 ST\_Touches Method

Tests if a geometry value spatially touches another geometry value.

## Syntax

```
<geometry-expression>.ST_Touches(<geo2>)
```

## Parameters

Table 39:

Name	Type	Description
geo2	ST_Geometry	The other geometry value that is to be compared to the <geometry-expression>.

## Returns

**INTEGER** Returns 1 if the <geometry-expression> touches <geo2>, otherwise 0. Returns NULL if both <geometry-expression> and <geo2> have dimension 0. If <geometry-expression> or <geo2> is NULL, then NULL is returned.

## Remarks

Tests if a geometry value spatially touches another geometry value. Two geometries spatially touch if their interiors do not intersect but one or more boundary points from one value intersects the interior or boundary of the other value.

### i Note

This method cannot be used with geometries in round-Earth spatial reference system.

### Example

The following example returns 1.

```
SELECT NEW ST_LINestring('LINESTRING(7.0 5.0, 9.0 7.0)')  
.ST_Touches( NEW ST_LINestring('LINESTRING(7.0 5.0, 8.0 5.0)') ) FROM dummy;
```

## 3.1.57 ST\_Transform Method

Creates a copy of the geometry value transformed into the specified spatial reference system.

### Syntax

```
<geometry-expression>.ST_Transform(<srid>)
```

### Parameters

Table 40:

Name	Type	Description
srid	INT	The SRID of the result.

### Returns

**ST\_Geometry** Returns a copy of the geometry value transformed into the specified spatial reference system.

### Remarks

The ST\_Transform method transforms `<geometry-expression>` from its spatial reference system to the specified spatial reference system using the transform definition of both spatial reference systems.

A transformation only takes place, if both spatial reference systems exist and if both spatial reference systems have a transform definition.

For spatial reference systems 0 a transformation does not take place, because spatial reference systems 0 does not have a transform definition.

#### Example

The following example returns the result Point (10, 30) in binary format.

```
SELECT NEW ST_Point('POINT(-86 36)',4326).ST_Transform(1000004326) from dummy;
```

## 3.1.58 ST\_Union Method

Returns the geometry value that represents the point set union of two geometries.

### Syntax

```
<geometry-expression>.ST_Union(<geo2>)
```

### Parameters

Table 41:

Name	Type	Description
geo2	ST_Geometry	The other geometry value that is to be unioned with the <geometry-expression>.

### Returns

**ST\_Geometry** Returns the geometry value that represents the point set union of two geometries.  
The spatial reference system identifier of the result is the same as the spatial reference system of the <geometry-expression>.

### Remarks

The ST\_Union method finds the spatial union of two geometries. A point is included in the union if it is present in either of the two input geometries.

#### i Note

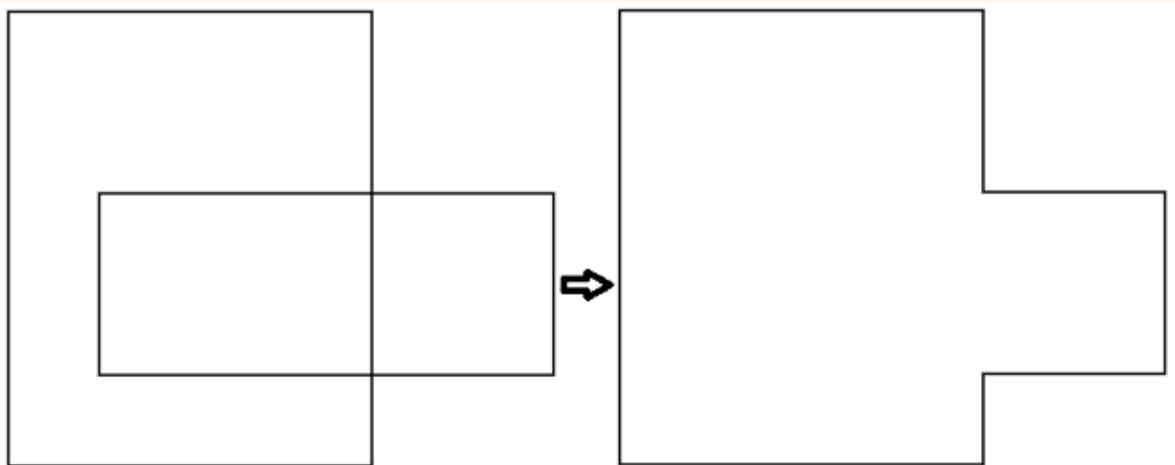
If the <geometry-expression> contains circularstrings, then these are interpolated to line strings.

#### Example

The following example shows union of two overlapping rectangles:

```
SELECT NEW ST_Polygon( 'Polygon( (1 1, 1 6, 5 6, 5 1, 1 1) )' ).ST_Union( NEW ST_Polygon( 'Polygon( (2 2, 2 4, 7 4, 7 2, 2 2) )' ) ).ST_AsText() FROM dummy;
```

The union of the two rectangles is a polygon (POLYGON ((5 4, 5 6, 1 6, 1 1, 5 1, 5 2, 7 2, 7 4, 5 4))):



### 3.1.59 ST\_UnionAggr Method

Returns the spatial union of all of the geometries in a group.

#### Syntax

```
ST_UnionAggr(<geometry_column>)
```

#### Parameters

Table 42:

Name	Type	Description
geometry_column	ST_Geometry	The geometry values to generate the spatial union. Typically this is a column.

#### Returns

**ST\_Geometry** Returns a geometry that is the spatial union for all the geometries in a group.

## Example

The following example returns the result MULTIPOINT ((2.5 3), (3 4.5)).

```
SELECT ST_UnionAggr( Shape ).ST_AsText() FROM SpatialShapes WHERE ShapeID in  
(1,2);
```

## 3.1.60 ST\_Within Method

Tests if a geometry value is spatially contained within another geometry value.

### Syntax

```
<geometry-expression>.ST_Within(<geo2>)
```

### Parameters

Table 43:

Name	Type	Description
geo2	ST_Geometry	The other geometry value that is to be compared to the <geometry-expression>.

### Returns

**INTEGER** Returns 1 if <geometry-expression> is within <geo2>, otherwise 0. If <geometry-expression> or <geo2> is NULL, then NULL is returned.

### Remarks

The ST\_Within method tests if the <geometry-expression> is completely within <geo2> and there is one or more interior points of <geo2> that lies in the interior of the <geometry-expression>.

<geometry-expression>.ST\_Within(<geo2>) is equivalent to <geo2>.ST\_Contains(<geometry-expression>).

The ST\_Within and ST\_CoveredBy methods are similar. The difference is that ST\_CoveredBy does not require intersecting interior points.

#### Note

If the <geometry-expression> contains circularstrings, then these are interpolated to line strings.

#### Note

This method cannot be used with geometries in round-Earth spatial reference system.

#### Example

The following example tests if a point is within a polygon. The point is completely within the polygon, and the interior of the point (the point itself) intersects the interior of the polygon, so the example returns 1.

```
SELECT NEW ST_Point(1, 1).ST_Within( NEW ST_Polygon( 'Polygon(( 0 0, 2 0, 1 2, 0 0 ))' ) ) FROM dummy;
```

The following example tests if a line is within a polygon. The line is completely within the polygon, but the interior of the line and the interior of the polygon do not intersect (the line only intersects the polygon on the polygon's boundary, and the boundary is not part of the interior), so the example returns 0. If ST\_CoveredBy was used in place of ST\_Within, ST\_CoveredBy would return 1.

```
SELECT NEW ST_LineString( 'LineString( 0 0, 1 0 )' ).ST_Within( NEW ST_Polygon( 'Polygon(( 0 0, 2 0, 1 2, 0 0 ))' ) ) FROM dummy;
```

## 3.1.61 ST\_WithinDistance Method

Test if two geometries are within a specified distance of each other.

### Syntax

```
<geometry-expression>.ST_WithinDistance(<geo2>,<distance>,<unit_name>)
```

## Parameters

Table 44:

Name	Type	Description
geo2	ST_Geometry	The other geometry value whose distance is to be measured from the <geometry-expression>.
distance	DOUBLE	The distance the two geometries should be within.
unit_name	VARCHAR(128)	The units in which the distance parameter should be interpreted. Defaults to the unit of the spatial reference system. The unit name must match the UNIT_NAME column of a row in the ST_UNITS_OF_MEASURE view where UNIT_TYPE is 'LINEAR'.

## Returns

**INTEGER** Returns 1 if <geometry-expression> and <geo2> are within the specified distance of each other, otherwise 0. If any of the input parameters is NULL, then NULL is returned.

## Remarks

The ST\_WithinDistance method tests if the smallest distance between two geometries does not exceed a specified distance, taking tolerance into consideration.

More precisely, let <d> denote the smallest distance between <geometry-expression> and <geo2>. The expression <geometry-expression>.ST\_WithinDistance( <geo2>, <distance>) evaluates to 1 if either <d> <= <distance> or if <d> exceeds <distance> by a length that is less than the tolerance of the associated spatial reference system.

### i Note

This method can only be used with geometries in round-Earth spatial reference system for point to point calculations.

### Example

The following example returns 1 because the two points are within 1 unit of distance.

```
SELECT NEW ST_Point(0,0).ST_WithinDistance( NEW ST_Point(1,0), 1 ) FROM dummy;
```

The following example returns 0 because the two points are not within 1 unit of distance.

```
SELECT NEW ST_Point(0,0).ST_WithinDistance( NEW ST_Point(1,1), 1 ) FROM dummy;
```

The following example returns 0 because the two points are not within 4 units of distance.

```
SELECT NEW ST_Point( 'Point(0 0)' ).ST_WithinDistance(NEW ST_Point( 'Point(3.1 4.1)' ), 4) FROM dummy;
```

The following example returns 0 because the two points are not within 6 units of distance.

```
SELECT NEW ST_Point( 'Point(0 0)' ).ST_WithinDistance(NEW ST_Point( 'Point(3.1 4.1)' ), 6) FROM dummy;
```

The following example returns 0 because the distance between Walldorf and Heidelberg is not less or equal 25 meters.

```
SELECT NEW ST_Point('Point(8.641493 49.293679)', 4326).ST_WithinDistance(new ST_Point('Point(8.682241 49.407571)', 4326), 25, 'meter') FROM dummy;
```

The following example returns 1 because the distance between Walldorf and Heidelberg is less or equal 25 kilometers.

```
SELECT NEW ST_Point('Point(8.641493 49.293679)', 4326).ST_WithinDistance(new ST_Point('Point(8.682241 49.407571)', 4326), 25, 'kilometer') FROM dummy;
```

## 3.1.62 ST\_XMax Method

Retrieves the maximum X coordinate value of a geometry.

### Syntax

```
<geometry-expression>.ST_XMax()
```

### Returns

**DOUBLE** Returns the maximum X coordinate value of the `<geometry-expression>`.

## Remarks

Returns the maximum X coordinate value of the `<geometry-expression>`. This is computed by comparing the X attribute of all points in the geometry.

### i Note

If the `<geometry-expression>` is an empty geometry (`ST_IsEmpty()=1`), then this method returns `NULL`.

### Example

The following returns the result 3.

```
SELECT NEW ST_LineString('LineString(1 2, 3 4)').ST_XMax() FROM dummy;
```

## 3.1.63 ST\_XMin Method

Retrieves the minimum X coordinate value of a geometry.

## Syntax

```
<geometry-expression>.ST_XMin()
```

## Returns

**DOUBLE** Returns the minimum X coordinate value of the `<geometry-expression>`.

## Remarks

Returns the minimum X coordinate value of the `<geometry-expression>`. This is computed by comparing the X attribute of all points in the geometry.

### i Note

If the `<geometry-expression>` is an empty geometry (`ST_IsEmpty()=1`), then this method returns `NULL`.

### Example

The following returns the result 1.

```
SELECT NEW ST_LineString('LineString(1 2, 3 4)').ST_XMin() FROM dummy;
```

## 3.1.64 ST\_YMax Method

Retrieves the maximum Y coordinate value of a geometry.

### Syntax

```
<geometry-expression>.ST_YMax()
```

### Returns

**DOUBLE** Returns the maximum Y coordinate value of the [`<geometry-expression>`](#).

### Remarks

Returns the maximum Y coordinate value of the [`<geometry-expression>`](#). This is computed by comparing the Y attribute of all points in the geometry.

#### Note

If the [`<geometry-expression>`](#) is an empty geometry (`ST_IsEmpty()=1`), then this method returns NULL.

### Example

The following returns the result 4.

```
SELECT NEW ST_LineString('LineString(1 2, 3 4)').ST_YMax() FROM dummy;
```

## 3.1.65 ST\_YMin Method

Retrieves the minimum Y coordinate value of a geometry.

### Syntax

```
<geometry-expression>.ST_YMin()
```

### Returns

**DOUBLE** Returns the minimum Y coordinate value of the `<geometry-expression>`.

### Remarks

Returns the minimum Y coordinate value of the `<geometry-expression>`. This is computed by comparing the Y attribute of all points in the geometry.

#### i Note

If the `<geometry-expression>` is an empty geometry (`ST_IsEmpty()=1`), then this method returns NULL.

#### Example

The following returns the result 2.

```
SELECT NEW ST_LineString('LineString(1 2, 3 4)').ST_YMin() FROM dummy;
```

## 3.1.66 ST\_ZMax Method

Retrieves the maximum Z coordinate value of a geometry.

### Syntax

```
<geometry-expression>.ST_ZMax()
```

## Returns

**DOUBLE** Returns the maximum Z coordinate value of the `<geometry-expression>`.

## Remarks

Returns the maximum Z coordinate value of the `<geometry-expression>`. This is computed by comparing the Z attribute of all points in the geometry.

### i Note

If the `<geometry-expression>` is an empty geometry (`ST_IsEmpty()=1`), then this method returns NULL.

### Example

#### Example

The following returns the result 10.

```
SELECT NEW ST_LineString('LineString Z(1 2 10, 3 4 7, 5 6 9.9)').ST_ZMax()  
FROM dummy;
```

## 3.1.67 ST\_ZMin Method

Retrieves the minimum Z coordinate value of a geometry.

## Syntax

```
<geometry-expression>.ST_ZMin()
```

## Returns

**DOUBLE** Returns the minimum Z coordinate value of the `<geometry-expression>`.

## Remarks

Returns the minimum Z coordinate value of the <geometry-expression>. This is computed by comparing the Z attribute of all points in the geometry.

### i Note

If the <geometry-expression> is an empty geometry (ST\_IsEmpty()=1), then this method returns NULL.

### Example

#### Example

The following returns the result 7.

```
SELECT NEW ST_LineString('LineString Z(1 2 10, 3 4 7, 5 6 9.9)').ST_ZMin()  
FROM dummy;
```

## 3.2 ST\_CircularString Type

The ST\_CircularString type is a subtype of ST\_Geometry that uses circular line segments between control points.

The first three points define a segment as follows. The first point is the start point of the segment. The second point is any point on the segment other than the start and end point. The third point is the end point of the segment. Subsequent segments are defined by two points only (intermediate and end point). The start point is taken to be the end point of the preceding segment.

A circularstring can be a complete circle with three points, if the start and end points are coincident. In this case, the intermediate point is the midpoint of the segment. If the start, intermediate and end points are collinear, the segment is a straight line segment between the start and end point. A circularstring with exactly three points is a circular arc. A circular ring is a circularstring that is both closed and simple.

Circularstrings are not allowed in round-Earth spatial reference systems. Attempting to create one for SRID 4326 for example returns an error.

### Direct supertype

[ST\\_Geometry \[page 34\]](#)

## Constructor

[ST\\_CircularString \[page 108\]](#)

### 3.2.1 ST\_CircularString Constructor

Constructs a circularstring.

#### Overload list

Table 45:

Name	Description
<a href="#">ST_CircularString() [page 108]</a>	Constructs a circularstring representing the empty set.
<a href="#">ST_CircularString(BLOB [.INT]) [page 109]</a>	Constructs a circularstring from Well Known Binary (WKB).
<a href="#">ST_CircularString(CLOB [.INT]) [page 110]</a>	Constructs a circularstring from Well Known Text (WKT).

#### 3.2.1.1 ST\_CircularString() Constructor

Constructs a circularstring representing the empty set.

#### Syntax

```
NEW ST_CircularString()
```

#### Example

Constructs a circularstring representing the empty set.

```
SELECT NEW ST_CircularString() FROM dummy;
```

### 3.2.1.2 ST\_CircularString(BLOB [,INT]) Constructor

Constructs a circularstring from Well Known Binary (WKB).

#### Syntax

```
NEW ST_CircularString(BLOB <wkb> [, INT <srid>)
```

#### Parameters

Table 46:

Name	Type	Description
wkb	BLOB	A string containing the text representation of a circularstring in Well Known Text (WKT) format.
srid	INT	The SRID of the result. If not specified and the input string does not contain a SRID, the default is 0.

#### Example

The following returns CircularString (5 10, 10 12, 15 10) in binary format.

```
SELECT NEW  
ST_CircularString(x'01080000000300000000000000000014400000000000002440000000000  
000244000000000000028400000000000002e40000000000002440') FROM dummy;
```

The following returns CircularString (5 10, 10 12, 15 10) in binary format.

```
SELECT NEW  
ST_CircularString(x'01080000000300000000000000000014400000000000002440000000000  
000244000000000000028400000000000002e40000000000002440', 1000004326) FROM  
dummy;
```

### 3.2.1.3 ST\_CircularString(CLOB [,INT]) Constructor

Constructs a circularstring from Well Known Text (WKT).

#### Syntax

```
NEW ST_CircularString(CLOB <wkt> [, INT <srid>])
```

#### Parameters

Table 47:

Name	Type	Description
wkt	CLOB	A string containing the text representation of a circularstring in Well Known Text (WKT) format.
srid	INT	The SRID of the result. If not specified and the input string does not contain a SRID, the default is 0.

#### Example

The following returns CircularString (0 0, 1 1, 0 2) in binary format.

```
SELECT NEW ST_CircularString('CIRCULARSTRING(0 0, 1 1, 0 2)') FROM dummy;
```

The following returns CircularString (0 0, 1 1, 0 2) in binary format.

```
SELECT NEW ST_CircularString('CIRCULARSTRING(0 0, 1 1, 0 2)', 1000004326) FROM dummy;
```

## 3.3 ST\_GeometryCollection Type

An ST\_GeometryCollection is a collection of zero or more ST\_Geometry values.

### Direct supertype

[ST\\_Geometry \[page 34\]](#)

### Constructor

[ST\\_GeometryCollection \[page 112\]](#)

### Methods

Methods of ST\_GeometryCollection

Table 48:

<a href="#">ST_GeometryN [page 115]</a>	<a href="#">ST_NumGeometries [page 116]</a>
---	---

### Remarks

An ST\_GeometryCollection is a collection of zero or more ST\_Geometry values. All of the values are in the same spatial reference system as the collection value. The ST\_GeometryCollection type can contain a heterogeneous collection of objects (for example, points, lines, and polygons). Sub-types of ST\_GeometryCollection can be used to restrict the collection to certain geometry types. The dimension of the geometry collection value is the largest dimension of its constituents. A geometry collection is simple if all of the constituents are simple and no two constituent geometries intersect except possibly at their boundaries.

### 3.3.1 ST\_GeometryCollection Constructor

Constructs a geometry collection.

#### Overload list

Table 49:

Name	Description
<a href="#">ST_GeometryCollection() [page 112]</a>	Constructs a geometry collection representing the empty set.
<a href="#">ST_GeometryCollection(LONG VARCHAR [.INT]) [page 113]</a>	Constructs a geometry collection from a text representation.
<a href="#">ST_GeometryCollection(LONG BINARY [.INT]) [page 114]</a>	Constructs a geometry collection from Well Known Binary (WKB).

#### 3.3.1.1 ST\_GeometryCollection() Constructor

Constructs a geometry collection representing the empty set.

#### Syntax

```
NEW ST_GeometryCollection()
```

#### Example

The following returns 1, indicating the value is empty.

```
SELECT NEW ST_GeometryCollection().ST_IsEmpty() FROM dummy;
```

### 3.3.1.2 ST\_GeometryCollection(LONG VARCHAR [,INT]) Constructor

Constructs a geometry collection from a text representation.

#### Syntax

```
NEW ST_GeometryCollection(<text-representation> [, INT <srid>])
```

#### Parameters

Table 50:

Name	Type	Description
text-representation	LONG VARCHAR	A string containing the text representation of a multi linestring in Well Known Text (WKT) format.
srid	INT	The SRID of the result. If not specified and the input string does not contain a SRID, the default is 0.

#### Remarks

Constructs a multi linestring from a character string representation.

##### Example

The following returns GeometryCollection (LineString(5 10, 10 12, 15 10), Polygon ((10 -5, 15 5, 5 5, 10 -5))) in binary format.

```
SELECT New ST_GeometryCollection( 'GeometryCollection (LineString(5 10, 10 12, 15 10), Polygon ((10 -5, 15 5, 5 5, 10 -5)))' ) FROM dummy;
```

The following returns the geometry collection in binary format.

```
SELECT New ST_GeometryCollection( 'GeometryCollection Z(LineString Z(5 10 20, 10 12 25, 15 10 13), Polygon Z((10 -5 4, 15 5 6, 5 5 7, 10 -5 4)), Point Z(10 15 12))' ) FROM dummy;
```

The following returns GeometryCollection (LineString(5 10, 10 12, 15 10), Polygon ((10 -5, 15 5, 5 5, 10 -5))) in binary format.

```
SELECT New ST_GeometryCollection( 'GeometryCollection (LineString(5 10, 10 12, 15 10), Polygon ((10 -5, 15 5, 5 5, 10 -5)))', 4326) FROM dummy;
```

### 3.3.1.3 ST\_GeometryCollection(LONG BINARY [,INT]) Constructor

Constructs a geometry collection from Well Known Binary (WKB).

#### Syntax

```
NEW ST_GeometryCollection(<wkb> [,INT <srid>])
```

#### Parameters

Table 51:

Name	Type	Description
wkb	LONG BINARY	A string containing the binary representation of a multi linestring in Well Known Binary (WKB) format.
srid	INT	The SRID of the result. If not specified and the input string does not contain a SRID, the default is 0.

#### Remarks

Constructs an ST\_GeometryCollection value from a binary string representation.

#### Example

The following returns GeometryCollection (Point (10 20)) in binary format.

```
SELECT NEW
ST_GeometryCollection(x'0107000000100000010100000000000000002440000000000000
03440') FROM dummy;
```

The following returns GeometryCollection (Point (10 20)) in binary format.

### 3.3.2 ST\_GeometryN Method

Returns the nth geometry in the geometry collection.

## Syntax

`<geometrycollection-expression>.ST_GeometryN(<n>)`

## Parameters

Table 52:

Name	Type	Description
n	INT	The position of the element to return, from 1 to geometrycollectionexpression. ST_NumGeometries().

## Returns

**ST\_Geometry** Returns the nth geometry in the geometry collection

## Example

The following example returns the result POLYGON ((10 -5, 15 5, 5 5, 10 -5)).

```
SELECT NEW ST_GeometryCollection('GeometryCollection (LineString(5 10, 10 12, 15 10), Polygon ((10 -5, 15 5, 5 5, 10 -5)))').ST_GeometryN( 2 ).ST_AsText()
FROM dummy;
```

The following example returns the result POLYGON ((10 -5, 15 5, 5 5, 5 -10, -5))

```
SELECT NEW ST_GeometryCollection('GeometryCollection (LineString(5 10, 10 12,  
15 10), Polygon ((10 -5, 15 5, 5 5, 10 -5)))',  
1000004326).ST_GeometryN( 2 ).ST_AsText() FROM dummy;
```

### 3.3.3 ST\_NumGeometries Method

Returns the number of geometries contained in the geometry collection.

#### Syntax

```
<geometrycollection-expression>.ST_NumGeometries()
```

#### Returns

**INT** Returns the number of geometries stored in this collection.

#### Example

The following example returns the result 3.

```
SELECT NEW ST_MultiPoint('MultiPoint ((10 10), (12 12), (14 10))').ST_NumGeometries() FROM dummy;
```

The following example returns the result 3.

```
SELECT NEW ST_MultiPoint('MultiPoint ((10 10), (12 12), (14 10))',  
1000004326).ST_NumGeometries() FROM dummy;
```

## 3.4 ST\_LineString Type

The ST\_LineString type is used to represent a multi-segment line using straight line segments between control points.

#### Direct supertype

[ST\\_Geometry \[page 34\]](#)

## Constructor

[ST\\_LineString \[page 117\]](#)

## Methods

- Methods of ST\_LineString:

Table 53:

<a href="#">ST_EndPoint [page 120]</a>	<a href="#">ST_IsClosed [page 121]</a>	<a href="#">ST_IsRing [page 122]</a>	<a href="#">ST_Length [page 123]</a>
<a href="#">ST_NumPoints [page 124]</a>	<a href="#">ST_PointN [page 125]</a>	<a href="#">ST_StartPoint [page 126]</a>	

- All methods of ST\_Geometry

## Remarks

The ST\_LineString type is used to represent a multi-segment line using straight line segments between control points. Each consecutive pair of points is joined with a straight line segment.

A line is an ST\_LineString value with exactly two points. A linear ring is an ST\_LineString value which is closed and simple.

### 3.4.1 ST\_LineString Constructor

Constructs a linestring.

## Overload list

Table 54:

Name	Description
<a href="#">ST_LineString() [page 118]</a>	Constructs a linestring representing the empty set.
<a href="#">ST_LineString(LONG VARCHAR [,INT]) [page 118]</a>	Constructs a linestring from a text representation.
<a href="#">ST_LineString(LONG BINARY [,INT]) [page 119]</a>	Constructs a linestring from Well Known Binary (WKB).

### 3.4.1.1 ST\_LineString() Constructor

Constructs a linestring representing the empty set.

#### Syntax

```
NEW ST_LineString()
```

##### Example

The following returns 1, indicating the value is empty.

```
SELECT NEW ST_LineString().ST_IsEmpty() FROM dummy;
```

### 3.4.1.2 ST\_LineString(LONG VARCHAR [,INT]) Constructor

Constructs a linestring from a text representation.

#### Syntax

```
NEW ST_LineString(<text-representation> [,INT <srid>])
```

#### Parameters

Table 55:

Name	Type	Description
text-representation	LONG VARCHAR	A string containing the text representation of a linestring in Well Known Text (WKT) format.
srid	INT	The SRID of the result. If not specified and the input string does not contain a SRID, the default is 0.

## Remarks

Constructs a linestring from a character string representation.

 Example

The following returns `LineString (0 0, 5 10)` in binary format.

```
SELECT NEW ST_LineString('LineString (0 0, 5 10)') FROM dummy;
```

## The following returns

```
SELECT NEW ST_LineString('LineString M(0 0 4, 5 10 6)') FROM dummy;
```

The following returns

```
SELECT NEW ST_LineString('LineString Z(0 0 7, 5 10 4)') FROM dummy;
```

## The following returns

```
SELECT NEW ST_LineString('LineString ZM(0 0 3 6, 5 10 4 8)') FROM dummy;
```

## The following returns

```
SELECT NEW ST_LineString('LineString (0 0, 5 10)', 4326) FROM dummy;
```

### 3.4.1.3 ST\_LineString(LONG BINARY [,INT]) Constructor

Constructs a linestring from Well Known Binary (WKB).

## Syntax

```
NEW ST_LineString(<wkb> [ INT <srid>)
```

## Parameters

Table 56:

Name	Type	Description
wkb	LONG BINARY	A string containing the binary representation of a linestring in Well Known Binary (WKB) format.
srid	INT	The SRID of the result. If not specified and the input string does not contain a SRID, the default is 0.

## Remarks

Constructs a linestring from a binary string representation.

## Example

The following returns LineString (0 0, 5 10) in binary format.

The following returns LineString (0 0, 5 10) in binary format.

### 3.4.2 ST EndPoint Method

Returns an ST\_Point value that is the end point.

## Syntax

```
<linestring-expression>.ST_EndPoint()
```

## Returns

**ST\_Point** If the linestring is an empty set, returns NULL. Otherwise, returns the end point of the linestring.

The spatial reference system identifier of the result is the same as the spatial reference system of the <linestring-expression>.



### Example

The following example returns the result Point (5 10) in binary format.

```
SELECT NEW ST_LineString( 'LineString(0 0, 5 5, 5 10)' ).ST_EndPoint() FROM dummy;
```

The following example returns the result Point (5 10) in binary format.

```
SELECT NEW ST_LineString( 'LineString(0 0, 5 5, 5 10)', 4326 ).ST_EndPoint() FROM dummy;
```

## 3.4.3 ST\_IsClosed Method

Test if the linestring is closed. A linestring is closed if the start and end points are coincident.

## Syntax

```
<linestring-expression>.ST_IsClosed()
```

## Returns

**INTEGER** Returns 1 if the linestring is closed (and non-empty). Otherwise, returns 0. If <linestring-expression> is NULL, then NULL is returned.



### Example

The following returns "1" because the linestring is closed.

```
SELECT NEW ST_LineString( 'LineString(0 0, 5 5, 5 0, 0 0)' ).ST_IsClosed() FROM dummy;
```

The following returns "1" because the linestring is closed.

```
SELECT NEW ST_LineString( 'LineString(0 0, 5 5, 5 0, 0 0)',  
4326).ST_IsClosed() FROM dummy;
```

### 3.4.4 ST\_IsRing Method

Tests if the linestring is a ring. A linestring is a ring if it is closed and simple (no self-intersections).

#### Syntax

```
<linestring-expression>.ST_IsRing()
```

#### Returns

**INTEGER** Returns 1 if the linestring is a ring (and non-empty). Otherwise, returns 0. If *<linestring-expression>* is NULL, then NULL is returned.

#### Example

The following returns "1" because the linestring is a ring.

```
SELECT NEW ST_LineString( 'LineString(0 0, 5 5, 5 0, 0 0)' ).ST_IsRing() FROM  
dummy;
```

The following returns "1" because the linestring is a ring.

```
SELECT NEW ST_LineString( 'LineString(0 0, 5 5, 5 0, 0 0)', 4326).ST_IsRing()  
FROM dummy;
```

## 3.4.5 ST\_Length Method

Returns the length of the linestring.

### Syntax

```
<linestring-expression>.ST_Length(<unit_name>)
```

### Parameters

Table 57:

Name	Type	Description
unit_name	VARCHAR(128)	The units in which the distance parameter should be interpreted. Defaults to the unit of the spatial reference system. The unit name must match the UNIT_NAME column of a row in the ST_UNITS_OF_MEASURE view where UNIT_TYPE is 'LINEAR'.

### Returns

**DOUBLE** If the linestring is an empty set, returns NULL. Otherwise, returns the length of the linestring.

### Remarks

The ST\_Length method returns the length of a linestring. If the linestring is empty, then NULL is returned.

If the linestring contains Z values, these are not considered when computing the length of the geometry.

#### i Note

If the `<linestring-expression>` is an empty geometry (`ST_IsEmpty()=1`), then this method returns NULL.

## Example

The following example returns the result 2.

```
SELECT NEW ST_LineString( 'LineString(1 0, 1 1, 2 1)' ).ST_Length() FROM dummy;
```

The following example returns the result 2.

```
SELECT NEW ST_LineString( 'LineString(1 0, 1 1, 2 1)', 4326 ).ST_Length() FROM dummy;
```

## 3.4.6 ST\_NumPoints Method

Returns the number of points defining the linestring.

## Syntax

```
<linestring-expression>.ST_NumPoints()
```

## Returns

**INT** Returns NULL if the linestring value is empty, otherwise the number of points in the value.

## Example

The following returns the result 5.

```
SELECT NEW ST_LineString('LineString(0 0, 1 0, 1 1, 0 1, 0 0)').ST_NumPoints() FROM dummy;
```

The following returns the result 5.

```
SELECT NEW ST_LineString('LineString(0 0, 1 0, 1 1, 0 1, 0 0 )', 4326 ).ST_NumPoints() FROM dummy;
```

## 3.4.7 ST\_PointN Method

Returns the `<n>`th point in the linestring.

### Syntax

```
<linestring-expression>.ST_PointN(<n>)
```

### Parameters

Table 58:

Name	Type	Description
n	INT	The position of the element to return, from 1 to <code>&lt;linestring-expression&gt;.ST_NumPoints()</code> .

### Returns

**ST\_Point** If the value of `<linestring-expression>` is the empty set, returns NULL. If the specified position `<n>` is less than 1 or greater than the number of points, returns NULL. Otherwise, returns the ST\_Point value at position n.

The spatial reference system identifier of the result is the same as the spatial reference system of the `<linestring-expression>`.

#### Example

The following returns the result Point (1 0) in binary format.

```
SELECT NEW ST_LineString('LineString(0 0, 1 0, 1 1, 0 1, 0 0 )').ST_PointN(2)  
FROM dummy;
```

The following returns the result Point (1 0) in binary format.

```
SELECT NEW ST_LineString('LineString(0 0, 1 0, 1 1, 0 1, 0 0 )',  
4326).ST_PointN(2) FROM dummy;
```

## 3.4.8 ST\_StartPoint Method

Returns an ST\_Point value that is the starting point.

### Syntax

```
<linestring-expression>.ST_StartPoint()
```

### Returns

**ST\_Point** If the linestring is an empty set, returns NULL. Otherwise, returns the start point of the linestring.

The spatial reference system identifier of the result is the same as the spatial reference system of the `<linestring-expression>`.



#### Example

The following example returns the result Point (0 0) in binary format.

```
SELECT NEW ST_LineString( 'LineString(0 0, 5 5, 5 10)').ST_StartPoint() FROM dummy;
```

The following example returns the result Point (0 0) in binary format.

```
SELECT NEW ST_LineString( 'LineString(0 0, 5 5, 5 10)', 4326).ST_StartPoint() FROM dummy;
```

## 3.5 ST\_MultiLineString Type

An ST\_MultiLineString is a collection of zero or more ST\_LineString values, and all of the linestrings are within the spatial reference system.

### Direct supertype

[ST\\_Geometry \[page 34\]](#)

## Constructor

[ST\\_MultiLineString \[page 127\]](#)

## Methods

- Methods of ST\_MultiLineString:

Table 59:

<a href="#">ST_IsClosed [page 121]</a>	<a href="#">ST_Length [page 131]</a>
--	--------------------------------------

- All methods of ST\_Geometry

### 3.5.1 ST\_MultiLineString Constructor

Constructs a multi linestring.

#### Overload list

Table 60:

Name	Description
<a href="#">ST_MultiLineString() [page 127]</a>	Constructs a multi linestring representing the empty set.
<a href="#">ST_MultiLineString(LONG VARCHAR [,INT]) [page 128]</a>	Constructs a multi linestring from a text representation.
<a href="#">ST_MultiLineString(LONG BINARY [,INT]) [page 129]</a>	Constructs a multi linestring from Well Known Binary (WKB).

#### 3.5.1.1 ST\_MultiLineString() Constructor

Constructs a multi linestring representing the empty set.

#### Syntax

```
NEW ST_MultiLineString()
```

### Example

The following returns 1, indicating the value is empty.

```
SELECT NEW ST_MultiLineString().ST_IsEmpty() FROM dummy;
```

## 3.5.1.2 **ST\_MultiLineString(LONG VARCHAR [,INT])** Constructor

Constructs a multi linestring from a text representation.

### Syntax

```
NEW ST_MultiLineString(<text-representation> [, INT <srid>])
```

### Parameters

Table 61:

Name	Type	Description
text-representation	LONG VARCHAR	A string containing the text representation of a multi linestring in Well Known Text (WKT) format.
srid	INT	The SRID of the result. If not specified and the input string does not contain a SRID, the default is 0.

### Remarks

Constructs a multi linestring from a character string representation.

### Example

The following returns MultiLineString ((10 10, 12 12), (14 10, 16 12)) in binary format.

```
SELECT NEW ST_MultiLineString('MultiLineString ((10 10, 12 12), (14 10, 16 12))') FROM dummy;
```

The following returns the multi linestring in binary format.

```
SELECT NEW ST_MultiLineString('MultiLineString Z((10 10 10, 12 12 12), (14 10  
10, 16 12 12))') FROM dummy;
```

The following returns the multi linestring in binary format.

```
SELECT NEW ST_MultiLineString('MultiLineString ZM((10 10 10 10, 12 12 12 10),  
(14 10 12 13, 16 12 13 14))') FROM dummy;
```

The following returns MultiLineString ((10 10, 12 12), (14 10, 16 12)) in binary format.

```
SELECT NEW ST_MultiLineString('MultiLineString ((10 10, 12 12), (14 10, 16  
12))', 4326) FROM dummy;
```

### 3.5.1.3 ST\_MultiLineString(LONG BINARY [,INT]) Constructor

Constructs a multi linestring from Well Known Binary (WKB).

#### Syntax

```
NEW ST_MultiLineString(<wkb> [,INT <srid>])
```

#### Parameters

Table 62:

Name	Type	Description
wkb	LONG BINARY	A string containing the binary representation of a multi linestring in Well Known Binary (WKB) format.
srid	INT	The SRID of the result. If not specified and the input string does not contain a SRID, the default is 0.

#### Remarks

Constructs a multi linestring from a binary string representation.

 Example

The following returns MultiLineString ((10 10, 12 12)) in binary format.

The following returns MultiLineString ((10 10, 12 12)) in binary format.

```
SELECT NEW  
ST_MultiLineString(x'0105000000010000000102000000200000000000000000000000244000000  
0000002440000000000000284000000000000002840', 4326) FROM dummy;
```

### **3.5.2 ST\_IsClosed Method**

Tests if the value is closed. A linestring is closed if the start and end points are coincident. A multilinestring is closed if it is non-empty and has an empty boundary.

## Syntax

```
<multilinestring-expression>.ST_IsClosed()
```

## Returns

**INTEGER** Returns 1 if the multilinestring is closed, otherwise 0. If [<multilinestring-expression>](#) is NULL, then NULL is returned.

 Example

The following returns the result 0 because the `LineString` has two points and therefore is not closed.

```
SELECT NEW ST_LineString('LineString(0 0, 1 0)').ST_IsClosed() FROM dummy;
```

The following returns the result 0 because the `LineString` has two points and therefore is not closed.

```
SELECT NEW ST_LineString('LineString(0 0, 1 0)', 4326).ST_IsClosed() FROM dummy;
```

The following returns the result 1 because the `LineString` is closed.

```
SELECT NEW ST_LineString('LineString(0 0, 1 0, 1 1, 0 0)').ST_IsClosed() FROM dummy;
```

The following returns the result 1 because the LineString is closed.

```
SELECT NEW ST_LineString('LineString(0 0, 1 0, 1 1, 0 0)', 4326).ST_IsClosed()  
FROM dummy;
```

### 3.5.3 ST\_Length Method

Returns the length measurement of all the linestrings in the multilinestring.

#### Syntax

```
<multilinestring-expression>.ST_Length(<unit_name>)
```

#### Parameters

Table 63:

Name	Type	Description
unit_name	VARCHAR(128)	The units in which the distance parameter should be interpreted. Defaults to the unit of the spatial reference system. The unit name must match the UNIT_NAME column of a row in the ST_UNITS_OF_MEASURE view where UNIT_TYPE is 'LINEAR'.

#### Returns

**DOUBLE** Returns the length measurement of the ST\_LineString ST\_MultiLineString value.

#### Remarks

The ST\_Length method returns the length of a multilinestring. The length of a multilinestring is the sum of the lengths of the contained linestrings. If the multilinestring is empty, then NULL is returned.

If the multilinestring contains Z values, these are not considered when computing the length of the geometry.

### Note

If the <multilinestring-expression> is an empty geometry (ST\_IsEmpty()=1), then this method returns NULL.

### Example

The following example creates a multilinestring with two elements and uses ST\_Length to find the length of the geometry, returning the value the result 2.

```
SELECT NEW ST_MultiLineString('MultiLineString ((0 0, 1 0), (0 1, 1  
1))').ST_Length() FROM dummy;
```

## 3.6 ST\_MultiPoint Type

An ST\_MultiPoint is a collection of zero or more ST\_Point values, and all of the points are within the spatial reference system.

### Direct supertype

[ST\\_Geometry \[page 34\]](#)

### Constructor

[ST\\_MultiPoint \[page 133\]](#)

## 3.6.1 ST\_MultiPoint Constructor

Constructs a multi-point.

### Overload list

Table 64:

Name	Description
<a href="#">ST_MultiPoint() [page 133]</a>	Constructs a multi-point representing the empty set.
<a href="#">ST_MultiPoint(LONG VARCHAR [,INT]) [page 133]</a>	Constructs a multi-point from a text representation.
<a href="#">ST_MultiPoint(LONG BINARY [,INT]) [page 135]</a>	Constructs a multi-point from Well Known Binary (WKB).

### 3.6.1.1 ST\_MultiPoint() Constructor

Constructs a multi-point representing the empty set.

#### Syntax

```
NEW ST_MultiPoint()
```

#### Example

The following returns 1, indicating the value is empty.

```
SELECT NEW ST_MultiPoint().ST_IsEmpty() FROM dummy;
```

### 3.6.1.2 ST\_MultiPoint(LONG VARCHAR [,INT]) Constructor

Constructs a multi-point from a text representation.

#### Syntax

```
NEW ST_MultiPoint(<text-representation> [,INT <srid>])
```

## Parameters

Table 65:

Name	Type	Description
text-representation	LONG VARCHAR	A string containing the text representation of a multi-point in Well Known Text (WKT) format.
srid	INT	The SRID of the result. If not specified and the input string does not contain a SRID, the default is 0.

## Remarks

Constructs a multi-point from a character string representation.

### Example

The following returns MultiPoint ((10 10), (12 12), (14 10)) in binary format.

```
SELECT NEW ST_MultiPoint('MultiPoint ((10 10), (12 12), (14 10))') FROM dummy;
```

The following returns the multi-point in binary format.

```
SELECT NEW ST_MultiPoint('MultiPoint Z((10 10 12), (12 12 14), (14 10 10))') FROM dummy;
```

The following returns the multi-point in binary format.

```
SELECT NEW ST_MultiPoint('MultiPoint ZM((10 10 12 1), (12 12 14 1), (14 10 10 1))') FROM dummy;
```

The following returns multi-point in binary format.

```
SELECT NEW ST_MultiPoint('MultiPoint ((10 10), (12 12), (14 10))', 4326) FROM dummy;
```

### **3.6.1.3 ST\_MultiPoint(LONG BINARY [,INT]) Constructor**

Constructs a multi-point from Well Known Binary (WKB).

## Syntax

```
NEW ST_MultiPoint(<wkb> [, INT <srid>])
```

## Parameters

Table 66:

Name	Type	Description
wkb	LONG BINARY	A string containing the binary representation of a multi-point in Well Known Binary (WKB) format.
srid	INT	The SRID of the result. If not specified and the input string does not contain a SRID, the default is 0.

## Remarks

Constructs a multi-point from a binary string representation.

## Example

The following returns MultiPoint ((10 10), (12 12), (14 10)) in binary format.

```
SELECT NEW  
ST_MultiPoint(x'0104000000030000001010000000000000000000000244000000000000002440010  
10000000000000000000002840000000000000028400101000000000000000000002c4000000000000024  
40!') FROM dummy;
```

The following returns MultiPoint ((10, 10), (12, 12), (14, 10)) in binary format

## 3.7 ST\_MultiPolygon Type

An ST\_MultiPolygon is a collection of zero or more ST\_Polygon values, and all of the polygons are within the same spatial reference system.

### Direct supertype

[ST\\_Geometry \[page 34\]](#)

### Constructor

[ST\\_MultiPolygon \[page 136\]](#)

### Methods

- Methods of ST\_MultiPolygon:

Table 67:

[ST\\_Area \[page 140\]](#)

- All methods of ST\_Geometry

### 3.7.1 ST\_MultiPolygon Constructor

Constructs a multi polygon.

### Overload list

Table 68:

Name	Description
<a href="#">ST_MultiPolygon() [page 137]</a>	Constructs a multi polygon representing the empty set.
<a href="#">ST_MultiPolygon(LONG VARCHAR [.INT]) [page 137]</a>	Constructs a multi polygon from a text representation.

Name	Description
<a href="#">ST_MultiPolygon(LONG BINARY [,INT]) [page 139]</a>	Constructs a multi polygon from Well Known Binary (WKB).

### 3.7.1.1 ST\_MultiPolygon() Constructor

Constructs a multi polygon representing the empty set.

#### Syntax

```
NEW ST_MultiPolygon()
```

##### Example

The following returns 1, indicating the value is empty.

```
SELECT NEW ST_MultiPolygon().ST_IsEmpty() FROM dummy;
```

### 3.7.1.2 ST\_MultiPolygon(LONG VARCHAR [,INT]) Constructor

Constructs a multi polygon from a text representation.

#### Syntax

```
NEW ST_MultiPolygon(<text-representation> [,INT <srid>])
```

## Parameters

Table 69:

Name	Type	Description
text-representation	LONG VARCHAR	A string containing the text representation of a multi polygon in Well Known Text (WKT) format.
srid	INT	The SRID of the result. If not specified and the input string does not contain a SRID, the default is 0.

## Remarks

Constructs a multi polygon from a character string representation.

### Example

The following returns MultiPolygon ((((-5 -5, 5 -5, 0 5, -5 -5), (-2 -2, -2 0, 2 0, 2 -2, -2 -2)), ((10 -5, 15 5, 5 5, 10 -5))) in binary format.

```
SELECT NEW ST_MultiPolygon('MultiPolygon ((((-5 -5, 5 -5, 0 5, -5 -5), (-2 -2, -2 0, 2 0, 2 -2, -2 -2)), ((10 -5, 15 5, 5 5, 10 -5)))') FROM dummy;
```

The following returns multipolygon in binary format.

```
SELECT NEW ST_MultiPolygon('MultiPolygon M((((-5 -5 3, 5 -5 3, 0 5 3, -5 -5 3), (-2 -2 3, -2 0 3, 2 0 3, 2 -2 3, -2 -2 3)), ((10 -5 3, 15 5 3, 5 5 3, 10 -5 3)))') FROM dummy;
```

The following returns multipolygon in binary format.

```
SELECT NEW ST_MultiPolygon('MultiPolygon Z((((-5 -5 4, 5 -5 7, 0 5 1, -5 -5 4), (-2 -2 9, -2 0 4, 2 0 4, 2 -2 1, -2 -2 9)), ((10 -5 2, 15 5 2, 5 5 3, 10 -5 2)))') FROM dummy;
```

The following returns multipolygon in binary format.

```
SELECT NEW ST_MultiPolygon('MultiPolygon ZM((((-5 -5 4 1, 5 -5 7 1, 0 5 1 1, -5 -5 4 1), (-2 -2 9 1, -2 0 4 1, 2 0 4 1, 2 -2 1 1, -2 -2 9 1)), ((10 -5 2 1, 15 5 2 1, 5 5 3 1, 10 -5 2 1)))') FROM dummy;
```

The following returns MultiPolygon ((((-5 -5, 5 -5, 0 5, -5 -5), (-2 -2, -2 0, 2 0, 2 -2, -2 -2)), ((10 -5, 15 5, 5 5, 10 -5))).

```
SELECT NEW ST_MultiPolygon('MultiPolygon ((((-5 -5, 5 -5, 0 5, -5 -5), (-2 -2, -2 0, 2 0, 2 -2, -2 -2)), ((10 -5, 15 5, 5 5, 10 -5))), 4326) FROM dummy;
```

### 3.7.1.3 ST\_MultiPolygon(LONG BINARY [,INT]) Constructor

Constructs a multi polygon from Well Known Binary (WKB).

#### Syntax

```
NEW ST_MultiPolygon(<wkb> [, INT <srid>])
```

#### Parameters

Table 70:

Name	Type	Description
wkb	LONG BINARY	A string containing the binary representation of a multi polygon in Well Known Binary (WKB) format.
srid	INT	The SRID of the result. If not specified and the input string does not contain a SRID, the default is 0.

#### Remarks

Constructs a multi polygon from a binary string representation.

##### Example

The following returns MultiPolygon (((10 -5, 15 5, 5 5, 10 -5))) in binary format.

```
SELECT NEW
ST_MultiPolygon(x'0106000000100000010300000010000004000000000000000024400
000000000014c00000000000002e40000000000001440000000000001440000000000001440
00000000000244000000000000014c0') FROM dummy;
```

The following returns MultiPolygon (((10 -5, 15 5, 5 5, 10 -5))) in binary format.

```
SELECT NEW
ST_MultiPolygon(x'0106000000100000010300000010000004000000000000000024400
0000000000014c00000000000002e40000000000001440000000000001440000000000001440
00000000000244000000000000014c0', 1000004326) FROM dummy;
```

## 3.7.2 ST\_Area Method

Computes the area of the multipolygon.

### Syntax

```
<multipolygon-expression>.ST_Area(<unit_name>)
```

### Parameters

Table 71:

Name	Type	Description
unit_name	VARCHAR(128)	The units in which the distance parameter should be interpreted. Defaults to the unit of the spatial reference system. The unit name must match the UNIT_NAME column of a row in the ST_UNITS_OF_MEASURE view where UNIT_TYPE is 'LINEAR'.

### Returns

**DOUBLE** Returns the area of the multipolygon.

### Remarks

Computes the area of the multipolygon. The area of the multipolygon is the sum of the areas of the contained polygons.

#### Example

The following returns the result 4.

```
SELECT NEW ST_MultiPolygon( 'MultiPolygon((( 0 0, 2 0, 1 2, 0 0 )),((10 2, 11  
0, 12 2, 10 2)))' )  
.ST_Area() FROM dummy;
```

The following returns the result 4.

```
SELECT NEW ST_MultiPolygon( 'MultiPolygon((( 0 0, 2 0, 1 2, 0 0 )),((10 2, 11  
0, 12 2, 10 2)))', 1000004326).ST_Area() FROM dummy;
```

## 3.8 ST\_Point Type

The ST\_Point type is a 0-dimensional geometry and represents a single location.

### Direct supertype

[ST\\_Geometry \[page 34\]](#)

### Constructor

[ST\\_Point \[page 142\]](#)

### Methods

- Methods of ST\_Point:

Table 72:

<a href="#">ST_M [page 145]</a>	<a href="#">ST_X [page 146]</a>	<a href="#">ST_Y [page 147]</a>	<a href="#">ST_Z [page 147]</a>
---------------------------------	---------------------------------	---------------------------------	---------------------------------

- All methods of ST\_Geometry

## 3.8.1 ST\_Point Constructor

Constructs a point.

### Overload list

Table 73:

Name	Description
<a href="#">ST_Point() [page 142]</a>	Constructs a point representing the empty set.
<a href="#">ST_Point(LONG VARCHAR [,INT]) [page 143]</a>	Constructs a point from a text representation.
<a href="#">ST_Point(LONG BINARY [,INT]) [page 144]</a>	Constructs a point from Well Known Binary (WKB).
<a href="#">ST_Point(DOUBLE,DOUBLE) [page 145]</a>	Constructs a 2D point from X,Y coordinates.

### 3.8.1.1 ST\_Point() Constructor

Constructs a point representing the empty set.

### Syntax

```
NEW ST_Point()
```

#### Example

The following returns 1, indicating the value is empty.

```
SELECT NEW ST_Point().ST_IsEmpty() FROM dummy;
```

### 3.8.1.2 ST\_Point(LONG VARCHAR [,INT]) Constructor

Constructs a point from a text representation.

#### Syntax

```
NEW ST_Point(<wkt> [,INT <srid>])
```

#### Parameters

Table 74:

Name	Type	Description
<wkt>	LONG VARCHAR	A string containing the text representation of a point in Well Known Text (WKT) format which is composed of x/longitude and y/latitude.
srid	INT	The SRID of the result. If not specified and the input string does not contain a SRID, the default is 0.

#### Remarks

Constructs a point from a character string representation.

##### Example

The following returns 01010000000000000000000024400000000000003440.

```
SELECT NEW ST_Point('Point (10 20)' ) FROM dummy;
```

The following returns 01E90300000000000000000024400000000000034400000000000003E40.

```
SELECT NEW ST_Point('Point Z(10 20 30)' ) FROM dummy;
```

The following returns 01D1070000000000000000002440000000000003E40000000000004440.

```
SELECT NEW ST_Point('Point M(10 30 40)' ) FROM dummy;
```

The following returns

01B90B000000000000000024400000000000034400000000000003E40000000000004940.

```
SELECT NEW ST_Point('Point ZM(10 20 30 50)') FROM dummy;
```

The following returns 01010000000000000000002440000000000003440.

```
SELECT NEW ST_Point('Point (10 20)', 4326) FROM dummy;
```

### 3.8.1.3 ST\_Point(LONG BINARY [,INT]) Constructor

Constructs a point from Well Known Binary (WKB).

#### Syntax

```
NEW ST_Point(<wkb> [,INT <srid>])
```

#### Parameters

Table 75:

Name	Type	Description
wkb	LONG BINARY	A string containing the binary representation of a point in Well Known Binary (WKB) format.
srid	INT	The SRID of the result. If not specified and the input string does not contain a SRID, the default is 0.

#### Remarks

Constructs a point from a binary string representation.

#### Example

The following returns 010100000000000000000024400000000000003440.

```
SELECT NEW ST_Point(x'010100000000000000000024400000000000003440') FROM dummy;
```

The following returns 01010000000000000000000024400000000000003440.

```
SELECT NEW ST_Point(x'0101000000000000000000002440000000000003440', 4326) FROM dummy;
```

### 3.8.1.4 ST\_Point(DOUBLE,DOUBLE) Constructor

Constructs a 2D point from X,Y coordinates.

#### Syntax

```
NEW ST_Point(<x>,<y>)
```

#### Parameters

Table 76:

Name	Type	Description
<x>	DOUBLE	The x/longitude coordinate value.
<y>	DOUBLE	The y/latitude coordinate value.

#### Example

The following returns the result 0101000000000000000000002440000000000003440.

```
SELECT NEW ST_Point(10.0,20.0) FROM dummy;
```

### 3.8.2 ST\_M Method

Returns the measure value of the ST\_POINT value.

#### Syntax

```
<point-expression>.ST_M()
```

## Returns

**DOUBLE** Returns the measure value of the ST\_Point value.

### Example

#### Example

The following example returns 70.

```
SELECT NEW ST_Point('Point M(10 40 70)').ST_M() FROM dummy;
```

## 3.8.3 ST\_X Method

Returns the X coordinate of the ST\_Point value.

### i Note

If the <point-expression> is an empty geometry (ST\_IsEmpty()=1), then this method returns NULL.

## Syntax

```
<point-expression>.ST_X()
```

## Returns

**DOUBLE** Returns the X coordinate of the ST\_Point value.

### Example

The following example returns the result 1.

```
SELECT NEW ST_Point('Point (1 2)').ST_X() FROM dummy;
```

## 3.8.4 ST\_Y Method

Returns the Y coordinate of the ST\_Point value.

### i Note

If the <point-expression> is an empty geometry (ST\_IsEmpty()=1), then this method returns NULL.

## Syntax

```
<point-expression>.ST_Y()
```

## Returns

**DOUBLE** Returns the Y coordinate of the ST\_Point value.

### Example

The following example returns the result 2.

```
SELECT NEW ST_Point( 'Point (1 2)' ).ST_Y() FROM dummy;
```

## 3.8.5 ST\_Z Method

Returns the Z coordinate of the ST\_Point value.

### i Note

If the <point-expression> is an empty geometry (ST\_IsEmpty()=1), then this method returns NULL.

## Syntax

```
<point-expression>.ST_Z()
```

## Returns

**DOUBLE** Returns the Z coordinate of the ST\_Point value.

### Example

#### Example

The following example returns the result 3.

```
SELECT NEW ST_Point( 'Point Z(1 2 3)').ST_Z() FROM dummy;
```

## 3.9 ST\_Polygon Type

An ST\_Polygon defines an area of space using an exterior ring and one or more interior rings, all defined using ST\_LineString.

### Direct supertype

[ST\\_Geometry \[page 34\]](#)

### Constructor

[ST\\_Polygon \[page 149\]](#)

### Methods

- Methods of ST\_Polygon:

Table 77:

<a href="#">ST_Area [page 152]</a>	<a href="#">ST_Centroid [page 153]</a>	<a href="#">ST_ExteriorRing [page 154]</a>	<a href="#">ST_InteriorRingN [page 154]</a>
------------------------------------	--	--	---

- All methods of ST\_Geometry

## 3.9.1 ST\_Polygon Constructor

Constructs a polygon.

### Overload list

Table 78:

Name	Description
<a href="#">ST_Polygon() [page 149]</a>	Constructs a polygon representing the empty set.
<a href="#">ST_Polygon(LONG VARCHAR [,INT]) [page 149]</a>	Constructs a polygon from a text representation.
<a href="#">ST_Polygon(LONG BINARY [,INT]) [page 151]</a>	Constructs a polygon from Well Known Binary (WKB).

### 3.9.1.1 ST\_Polygon() Constructor

Constructs a polygon representing the empty set.

#### Syntax

```
NEW ST_Polygon()
```

#### Example

The following returns 1, indicating the value is empty.

```
SELECT NEW ST_Polygon().ST_IsEmpty() FROM dummy;
```

### 3.9.1.2 ST\_Polygon(LONG VARCHAR [,INT]) Constructor

Constructs a polygon from a text representation.

#### Syntax

```
NEW ST_Polygon(<text-representation> [,INT <srid>])
```

## Parameters

Table 79:

Name	Type	Description
text-representation	LONG VARCHAR	A string containing the text representation of a polygon in Well Known Text (WKT) format.
srid	INT	The SRID of the result. If not specified and the input string does not contain a SRID, the default is 0.

## Remarks

Constructs a polygon from a character string representation.

### Example

The following returns Polygon ((-5 -5, 5 -5, 0 5, -5 -5), (-2 -2, -2 0, 2 0, 2 -2, -2 -2)) in binary format.

```
SELECT NEW ST_Polygon('Polygon ((-5 -5, 5 -5, 0 5, -5 -5), (-2 -2, -2 0, 2 0, 2 -2, -2 -2))') FROM dummy;
```

The following returns the polygon in binary format.

```
SELECT NEW ST_Polygon('Polygon Z((-5 -5 6, 5 -5 6, 0 5 6, -5 -5 6), (-2 -2 1, -2 0 1, 2 0 1, 2 -2 1, -2 -2 1))') FROM dummy;
```

The following returns the polygon in binary format.

```
SELECT NEW ST_Polygon('Polygon M((-5 -5 6, 5 -5 6, 0 5 6, -5 -5 6), (-2 -2 1, -2 0 1, 2 0 1, 2 -2 1, -2 -2 1))') FROM dummy;
```

The following returns the polygon in binary format.

```
SELECT NEW ST_Polygon('Polygon ZM((-5 -5 6 4, 5 -5 6 4, 0 5 6 4, -5 -5 6 4), (-2 -2 1 4, -2 0 1 4, 2 0 1 4, 2 -2 1 4, -2 -2 1 4))') FROM dummy;
```

The following returns the polygon in binary format.

```
SELECT NEW ST_Polygon('Polygon ((-5 -5, 5 -5, 0 5, -5 -5), (-2 -2, -2 0, 2 0, 2 -2, -2 -2))', 4326) FROM dummy;
```

The following returns the polygon in binary format.

```
SELECT NEW ST_Polygon('Polygon ((-5 -5, 5 -5, 0 5, -5 -5), (-2 -2, -2 0, 2 0, 2 -2, -2 -2))', 1000004326) FROM dummy;
```

### 3.9.1.3 ST\_Polygon(LONG BINARY [,INT]) Constructor

Constructs a polygon from Well Known Binary (WKB).

#### Syntax

```
NEW ST_Polygon(<wkb> [, INT <srid>])
```

#### Parameters

Table 80:

Name	Type	Description
wkb	LONG BINARY	A string containing the binary representation of a polygon in Well Known Binary (WKB) format.
srid	INT	The SRID of the result. If not specified and the input string does not contain a SRID, the default is 0.

#### Remarks

Constructs a polygon from a binary string representation.

##### Example

The following returns Polygon ((10 -5, 15 5, 5 5, 10 -5)) in binary format.

```
SELECT NEW
ST_Polygon(x'010300000001000000400000000000000000244000000000000014c000000000
00002e4000000000000014400000000000001440000000000000144000000000000024400000000
000014c0') FROM dummy;
```

The following returns Polygon ((10 -5, 15 5, 5 5, 10 -5)) in binary format.

```
SELECT NEW
ST_Polygon(x'010300000001000000400000000000000000244000000000000014c000000000
00002e4000000000000014400000000000001440000000000000144000000000000024400000000
000014c0', 1000004326) FROM dummy;
```

## 3.9.2 ST\_Area Method

Calculates the area of a polygon.

### Syntax

```
<polygon-expression>.ST_Area(<unit_name>)
```

### Parameters

Table 81:

Name	Type	Description
unit_name	VARCHAR(128)	The units in which the distance parameter should be interpreted. Defaults to the unit of the spatial reference system. The unit name must match the UNIT_NAME column of a row in the ST_UNITS_OF_MEASURE view where UNIT_TYPE is 'LINEAR'.

### Returns

**DOUBLE** Returns the area of the polygon.

### Remarks

The ST\_Area method computes the area of a polygon.

#### Example

The following returns the result 1 as the area of the unit square.

```
SELECT NEW ST_Polygon('Polygon((0 0, 1 0, 1 1, 0 1, 0 0))').ST_Area() FROM dummy;
```

The following returns the result 1 as the area of the unit square.

```
SELECT NEW ST_Polygon('Polygon((0 0, 1 0, 1 1, 0 1, 0 0))', 1000004326).ST_Area() FROM dummy;
```

### 3.9.3 ST\_Centroid Method

Returns the ST\_Point value that is the mathematical centroid of the polygon value.

#### Syntax

```
<polygon-expression>.ST_Centroid()
```

#### Returns

**ST\_Point** If the polygon is the empty set, returns NULL. Otherwise, returns the mathematical centroid of the polygon.

The spatial reference system identifier of the result is the same as the spatial reference system of the `<polygon-expression>`.

#### Remarks

Returns the ST\_Point value that is the mathematical centroid of the polygon value. This point will not necessarily be a point on the polygon.

##### Example

The following returns the result 0101000000000000000000001440ABAAAAAAA1240.

```
SELECT NEW ST_Polygon('Polygon ((3 3, 8 3, 4 8, 3 3))').ST_Centroid() FROM dummy;
```

The following returns the result 0101000000000000000000001440ABAAAAAAA1240.

```
SELECT NEW ST_Polygon('Polygon ((3 3, 8 3, 4 8, 3 3))',  
1000004326).ST_Centroid() FROM dummy;
```

## 3.9.4 ST\_ExteriorRing Method

Returns the exterior ring of the polygon.

### Syntax

```
<polygon-expression>.ST_ExteriorRing()
```

### Returns

**ST\_LineString** Returns a line string representing the exterior ring of the POLYGON geometry.

#### Example

The following example returns the polygon with the specified external ring.

```
SELECT NEW ST_Polygon('Polygon ((0 0, 3 0, 3 3, 0 3, 0 0), (1 1, 2 1, 2 2, 1 1))').ST_ExteriorRing() FROM dummy;
```

## 3.9.5 ST\_InteriorRingN Method

Returns the nth interior ring in the polygon.

### Syntax

```
<polygon-expression>.ST_InteriorRingN(<n>)
```

## Parameters

Table 82:

Name	Type	Description
n	INT	The position of the element to return, from 1 to <code>polygonexpression</code> . <code>ST_NumInteriorRing()</code> .

## Returns

### ST\_LineString

Returns the nth interior ring in the polygon.

#### Example

The following example returns the inner ring as LINESTRING (1 1,2 2,2 1,1 1).

```
SELECT NEW ST_Polygon('Polygon ((0 0, 3 0, 3 3, 0 3, 0 0), (1 1, 2 1, 2 2, 1 1))').ST_InteriorRingN(1).ST_AsText() FROM dummy;
```

## 3.10 List of All Supported Methods

The following is a list of all supported spatial methods.

Table 83:

Method	Type	Round-Earth	Description
<a href="#">ST_Area [page 140]</a>	<a href="#">ST_MultiPolygon [page 136]</a>	not supported	Computes the area of the multipolygon.
<a href="#">ST_Area [page 152]</a>	<a href="#">ST_Polygon [page 148]</a>	not supported	Calculates the area of a polygon.
<a href="#">ST_AsBinary [page 36]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the WKB representation of an ST_Geometry value.
<a href="#">ST_AsEWKB [page 37]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the extended WKB representation of an ST_Geometry value.

Method	Type	Round-Earth	Description
<a href="#">ST_AsEWKT [page 37]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the extended WKT representation of an ST_Geometry value.
<a href="#">ST_AsGeoJSON [page 38]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns a string representing a geometry in JSON format.
<a href="#">ST_AsSVG [page 39]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns an SVG figure representing a geometry value.
<a href="#">ST_AsSVGAggr [page 40]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns a complete or partial SVG document which renders the geometries in a group.
<a href="#">ST_AsText [page 41]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the text representation of an ST_Geometry value.
<a href="#">ST_AsWKB [page 42]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the WKB representation of an ST_Geometry value.
<a href="#">ST_AsWKT [page 43]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the WKT representation of an ST_Geometry value.
<a href="#">ST_Boundary [page 44]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Returns the boundary of the geometry value.
<a href="#">ST_Buffer [page 44]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Returns the ST_Geometry value that represents all points whose distance from any point of an ST_Geometry value is less than or equal to a specified distance in the given units.
<a href="#">ST_Centroid [page 153]</a>	<a href="#">ST_Polygon [page 148]</a>	not supported	Returns the ST_Point value that is the mathematical centroid of the polygon value.
<a href="#">ST_Contains [page 46]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Tests if a geometry value spatially contains another geometry value.
<a href="#">ST_ConvexHull [page 48]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Returns the convex hull of the geometry value.

Method	Type	Round-Earth	Description
<a href="#">ST_ConvexHullAggr [page 49]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Returns the convex hull for all of the geometries in a group.
<a href="#">ST_CoordDim [page 50]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the number of coordinate dimensions stored with each point of the ST_Geometry value.
<a href="#">ST_CoveredBy [page 52]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Tests if a geometry value is spatially covered by another geometry value.
<a href="#">ST_Covers [page 53]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Tests if a geometry value spatially covers another geometry value.
<a href="#">ST_Crosses [page 55]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Tests if a geometry value crosses another geometry value.
<a href="#">ST_Difference [page 56]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the geometry value that represents the point set difference of two geometries.
<a href="#">ST_Dimension [page 57]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the dimension of the ST_Geometry value. Points have dimension 0, lines have dimension 1, and polygons have dimension 2. Any empty geometry has dimension -1.
<a href="#">ST_Disjoint [page 58]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Test if a geometry value is spatially disjoint from another value.
<a href="#">ST_Distance [page 60]</a>	<a href="#">ST_Geometry [page 34]</a>	partially supported (only point-point)	Returns the smallest distance between the <geometry-expression> and the specified geometry value.
<a href="#">ST_EndPoint [page 120]</a>	<a href="#">ST_LineString [page 116]</a>	supported	Returns an ST_Point value that is the end point.
<a href="#">ST_Envelope [page 61]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Returns the bounding rectangle for the geometry value.

Method	Type	Round-Earth	Description
<a href="#">ST_EnvelopeAggr [page 62]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Returns the bounding rectangle for all of the geometries in a group.
<a href="#">ST_Equals [page 63]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.
<a href="#">ST_ExteriorRing [page 154]</a>	<a href="#">ST_Polygon [page 148]</a>	supported	Returns the exterior ring of the polygon.
<a href="#">ST_GeomFromEWKB [page 64]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Constructs a geometry from a string representation.
<a href="#">ST_GeomFromEWKT [page 65]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Constructs a geometry from a string representation.
<a href="#">ST_GeomFromText [page 66]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Constructs a geometry from a character string representation.
<a href="#">ST_GeomFromWKB [page 68]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Constructs a geometry from a string representation.
<a href="#">ST_GeomFromWKT [page 69]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Constructs a geometry from a string representation.
<a href="#">ST_GeometryN [page 115]</a>	<a href="#">ST_GeometryCollection [page 111]</a>	supported	Returns the nth geometry in the geometry collection.
<a href="#">ST_GeometryType [page 70]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the name of the type of the ST_Geometry value.
<a href="#">ST_InteriorRingN [page 154]</a>	<a href="#">ST_Polygon [page 148]</a>	supported	Returns the nth interior ring in the polygon.
<a href="#">ST_Intersection [page 71]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the geometry value that represents the point set intersection of two geometries.
<a href="#">ST_Intersects [page 72]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Test if a geometry value spatially intersects another value.
<a href="#">ST_IntersectionAggr [page 73]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the spatial intersection of all of the geometries in a group.

Method	Type	Round-Earth	Description
<a href="#">ST_IntersectsFilter [page 74]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	An inexpensive test if the two geometries might intersect.
<a href="#">ST_IntersectsRect [page 74]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Test if a geometry intersects a rectangle.
<a href="#">ST_Is3D [page 77]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Determines if the geometry value has Z coordinate values.
<a href="#">ST_IsClosed [page 121]</a>	<a href="#">ST_LineString [page 116]</a>	supported	Test if the linestring is closed. A linestring is closed if the start and end points are coincident.
<a href="#">ST_IsClosed [page 130]</a>	<a href="#">ST_MultiLineString [page 126]</a>	supported	Tests if the value is closed. A linestring is closed if the start and end points are coincident. A multilinestring is closed if it is non-empty and has an empty boundary.
<a href="#">ST_IsEmpty [page 78]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Determines whether the geometry value represents an empty set.
<a href="#">ST_IsMeasured [page 79]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Determines if the geometry value has associated measure values.
<a href="#">ST_IsRing [page 122]</a>	<a href="#">ST_LineString [page 116]</a>	supported	Tests if the linestring is a ring. A linestring is a ring if it is closed and simple (no self intersections).
<a href="#">ST_IsSimple [page 79]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Determines whether the geometry value is simple (containing no self intersections or other irregularities).
<a href="#">ST_IsValid [page 80]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Determines whether the geometry is a valid spatial object.
<a href="#">ST_Length [page 123]</a>	<a href="#">ST_LineString [page 116]</a>	supported	Retrieves the length measurement.

Method	Type	Round-Earth	Description
<a href="#">ST_Length [page 131]</a>	<a href="#">ST_MultiLineString [page 126]</a>	supported	Returns the length measurement of all the linestrings in the multilinestring.
<a href="#">ST_M [page 145]</a>	<a href="#">ST_Point [page 141]</a>	supported	Retrieves or modifies the M coordinate value of a point.
<a href="#">ST_MMax [page 81]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Retrieves the maximum M coordinate value of a geometry.
<a href="#">ST_MMin [page 81]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Retrieves the minimum M coordinate value of a geometry.
<a href="#">ST_NumGeometries [page 116]</a>	<a href="#">ST_GeometryCollection [page 111]</a>	supported	Returns the number of geometries contained in the geometry collection.
<a href="#">ST_NumInteriorRing [page 82]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the number of interior rings in the polygon.
<a href="#">ST_NumInteriorRings [page 83]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the number of interior rings in the polygon.
<a href="#">ST_NumPoints [page 124]</a>	<a href="#">ST_LineString [page 116]</a>	supported	Returns the number of points defining the linestring.
<a href="#">ST_OrderingEquals [page 83]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Tests if a geometry is identical to another geometry.
<a href="#">ST_Overlaps [page 85]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Tests if a geometry value overlaps another geometry value.
<a href="#">ST_Perimeter [page 87]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Computes the perimeter of the multi-surface in the specified unit.
<a href="#">ST_PointOnSurface [page 88]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns an ST_Point value that is guaranteed to spatially intersect the ST_Surface value.
<a href="#">ST_PointN [page 125]</a>	<a href="#">ST_LineString [page 116]</a>	supported	Returns the $< n >$ th point in the linestring.
<a href="#">ST_Relate [page 89]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Tests if a geometry value is spatially related to another geometry value as specified by the intersection matrix.

Method	Type	Round-Earth	Description
<a href="#">ST_SRID [page 90]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Retrieves or modifies the spatial reference system associated with the geometry value.
<a href="#">ST_SRID(INT) [page 91]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Changes the spatial reference system associated with the geometry without modifying any of the values.
<a href="#">ST_SnapToGrid [page 92]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns a copy of the geometry with all points snapped to the specified grid.
<a href="#">ST_StartPoint [page 126]</a>	<a href="#">ST_LineString [page 116]</a>	supported	Returns an ST_Point value that is the starting point.
<a href="#">ST_SymDifference [page 93]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the geometry value that represents the point set symmetric difference of two geometries.
<a href="#">ST_Touches [page 94]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Tests if a geometry value spatially touches another geometry value.
<a href="#">ST_Transform [page 96]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Creates a copy of the geometry value transformed into the specified spatial reference system.
<a href="#">ST_Union [page 97]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the geometry value that represents the point set union of two geometries.
<a href="#">ST_UnionAggr [page 98]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the spatial union of all of the geometries in a group.
<a href="#">ST_Within [page 99]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Tests if a geometry value is spatially contained within another geometry value.
<a href="#">ST_WithinDistance [page 100]</a>	<a href="#">ST_Geometry [page 34]</a>	partially supported (only point-point)	Test if two geometries are within a specified distance of each other.
<a href="#">ST_X [page 146]</a>	<a href="#">ST_Point [page 141]</a>	supported	Retrieves or modifies the X coordinate value of a point.

Method	Type	Round-Earth	Description
<a href="#">ST_XMax [page 102]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Retrieves the maximum X coordinate value of a geometry.
<a href="#">ST_XMin [page 103]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Retrieves the minimum X coordinate value of a geometry.
<a href="#">ST_Y [page 147]</a>	<a href="#">ST_Point [page 141]</a>	supported	Retrieves or modifies the Y coordinate value of a point.
<a href="#">ST_YMax [page 104]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Retrieves the maximum Y coordinate value of a geometry.
<a href="#">ST_YMin [page 105]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Retrieves the minimum Y coordinate value of a geometry.
<a href="#">ST_Z [page 147]</a>	<a href="#">ST_Point [page 141]</a>	supported	Retrieves or modifies the Z coordinate value of a point.
<a href="#">ST_ZMax [page 105]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Retrieves the maximum Z coordinate value of a geometry.
<a href="#">ST_ZMin [page 106]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Retrieves the minimum Z coordinate value of a geometry.

## 3.11 List of All Supported Constructors

The following is a list of all supported spatial constructors.

Table 84:

Constructor	Description
<a href="#">ST_CircularString [page 107]</a>	Constructs a circularstring.
<a href="#">ST_GeometryCollection [page 111]</a>	Constructs a geometry collection.
<a href="#">ST_LineString [page 116]</a>	Constructs a linestring.
<a href="#">ST_MultiLineString [page 126]</a>	Constructs a multilinestring.
<a href="#">ST_MultiPoint [page 132]</a>	Constructs a multipoint.
<a href="#">ST_MultiPolygon [page 136]</a>	Constructs a multipolygon.

Constructor	Description
<a href="#">ST_Point [page 141]</a>	Constructs a point.
<a href="#">ST_Polygon [page 148]</a>	Constructs a polygon.

## 3.12 List of Aggregate Methods

There are many aggregate methods available for use with spatial data.

Table 85:

Method	Type	Round-Earth	Description
<a href="#">ST_AsSVGAggr [page 40]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns a complete or partial SVG document which renders the geometries in a group.
<a href="#">ST_ConvexHullAggr [page 49]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Returns the convex hull for all of the geometries in a group.
<a href="#">ST_EnvelopeAggr [page 62]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Returns the bounding rectangle for all of the geometries in a group.
<a href="#">ST_IntersectionAggr [page 73]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the spatial intersection of all of the geometries in a group.
<a href="#">ST_UnionAggr [page 98]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the spatial union of all of the geometries in a group.

## 3.13 List of Set Operation Methods

The following is a list of set operation methods available for use with spatial data.

Table 86:

Method	Type	Round-Earth	Description
<a href="#">ST_Difference [page 56]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the geometry value that represents the point set difference of two geometries.
<a href="#">ST_Intersection [page 71]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the geometry value that represents the point set intersection of two geometries.
<a href="#">ST_IntersectionAggr [page 73]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the spatial intersection of all of the geometries in a group.
<a href="#">ST_SymDifference [page 93]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the geometry value that represents the point set symmetric difference of two geometries.
<a href="#">ST_Union [page 97]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the geometry value that represents the point set union of two geometries.
<a href="#">ST_UnionAggr [page 98]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the spatial union of all of the geometries in a group.

## 3.14 List of Static Methods

There are many static methods available for use with spatial data.

Table 87:

Method	Type	Round-Earth	Description
<a href="#">ST_AsSVGAggr [page 40]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns a complete or partial SVG document which renders the geometries in a group.

Method	Type	Round-Earth	Description
<a href="#">ST_ConvexHullAggr [page 49]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Returns the convex hull for all of the geometries in a group.
<a href="#">ST_EnvelopeAggr [page 62]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Returns the bounding rectangle for all of the geometries in a group.
<a href="#">ST_GeomFromText [page 66]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Constructs a geometry from a character string representation.
<a href="#">ST_GeomFromWKB [page 68]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Constructs a geometry from a string representation.
<a href="#">ST_GeomFromWKT [page 69]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Constructs a geometry from a string representation.
<a href="#">ST_IntersectionAggr [page 73]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the spatial intersection of all of the geometries in a group.
<a href="#">ST_UnionAggr [page 98]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the spatial union of all of the geometries in a group.

## 3.15 List of Spatial Predicates

The following is a list of predicate methods available for use with spatial data.

Table 88:

Method	Type	Round-Earth	Description
<a href="#">ST_Contains [page 46]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Tests if a geometry value spatially contains another geometry value.
<a href="#">ST_CoveredBy [page 52]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Tests if a geometry value is spatially covered by another geometry value.
<a href="#">ST_Covers [page 53]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Tests if a geometry value spatially covers another geometry value.

Method	Type	Round-Earth	Description
<a href="#">ST_Crosses [page 55]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Tests if a geometry value crosses another geometry value.
<a href="#">ST_Disjoint [page 58]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Test if a geometry value is spatially disjoint from another value.
<a href="#">ST_Equals [page 63]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.
<a href="#">ST_Intersects [page 72]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the geometry value that represents the point set intersection of two geometries.
<a href="#">ST_Intersects [page 72]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Test if a geometry value spatially intersects another value.
<a href="#">ST_IntersectionAggr [page 73]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Returns the spatial intersection of all of the geometries in a group.
<a href="#">ST_IntersectsFilter [page 74]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	An inexpensive test if the two geometries might intersect.
<a href="#">ST_IntersectsRect [page 74]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Test if a geometry intersects a rectangle.
<a href="#">ST_OrderingEquals [page 83]</a>	<a href="#">ST_Geometry [page 34]</a>	supported	Tests if a geometry is identical to another geometry.
<a href="#">ST_Overlaps [page 85]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Tests if a geometry value overlaps another geometry value.
<a href="#">ST_Relate [page 89]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Tests if a geometry value is spatially related to another geometry value as specified by the intersection matrix.
<a href="#">ST_Touches [page 94]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Tests if a geometry value spatially touches another geometry value.
<a href="#">ST_Within [page 99]</a>	<a href="#">ST_Geometry [page 34]</a>	not supported	Tests if a geometry value is spatially contained within another geometry value.

Method	Type	Round-Earth	Description
<a href="#">ST_WithinDistance [page 100]</a>	<a href="#">ST_Geometry [page 34]</a>	partially supported (only point-point)	Test if two geometries are within a specified distance of each other.

# 4 Spatial Clustering

SAP HANA spatial provides spatial clustering using the algorithms grid, k-means, and DBSCAN. Spatial clustering can be performed on a set of geospatial points (referred to hereafter as the original data set) in the SAP HANA system.

## 4.1 Introduction to Spatial Clustering

Spatial clustering groups a set of points with regard to certain criteria into clusters. A cluster is a partition of the original point set.

**i** Note

Spatial clustering for SAP HANA spatial is available for two-dimensional points in flat-earth spatial reference systems (for example WGS 84 (planar) - SRID 1000004326).

The grouping criteria depend on the clustering algorithm that is applied on the points and on the parameterization of the cluster algorithm. SAP spatial provides the following algorithms:

- Grid
- K-means
- DBSCAN

Clustering provides many insights into the data set. Most of the time operators are interested in the following:

- Association of each data point to a cluster (identified by a cluster ID)
- Characteristics of each cluster, for example:
  - Centroid
  - Envelope
- Aggregates on data associated with the points located in a cluster:
  - Number of points
  - Average measurement ( average household income level for example)
  - Aggregation functions (ST\_ConvexHullAggr, ST\_EnvelopeAggr, ST\_Intersection\_Aggr)

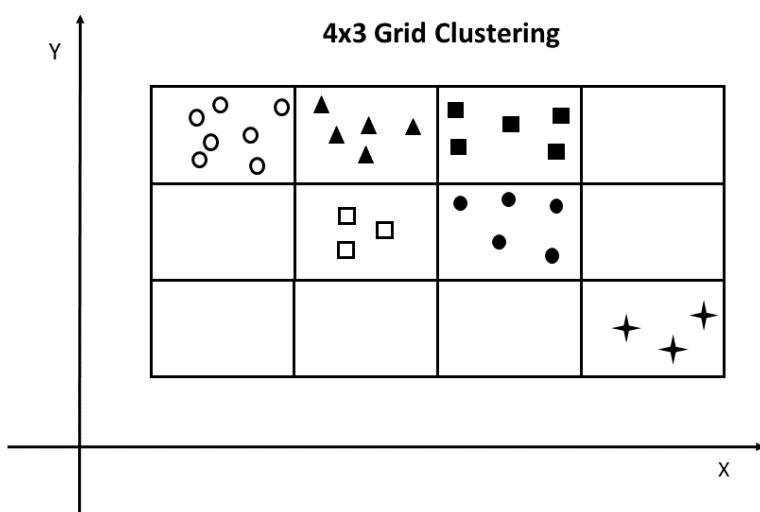
## 4.2 Cluster Algorithms

SAP HANA spatial supports several cluster algorithms.

### 4.2.1 Grid

SAP HANA spatial supports a grid-based aggregation. The following methods are available for the grid algorithm: ST\_ClusterID(), ST\_ClusterEnvelope().

Grid clustering provides a quick and easy way to use clustering. It is useful for providing a first impression. For deeper analysis, you can use the other algorithms.



The mapping for the grid clustering is defined with respect to the grid. A grid is a rectangle that can be further divided into a set of inner rectangles (cells). Rather than intersecting the grid, these cells tessellate it. Every cell has a unique cluster identifier. You can specify the number of grid cells in the X and Y direction. You can also specify the location of the grid.

#### i Note

If no boundaries are defined, the cluster grid is created automatically. In this case all points belong to clusters with ST\_ClusterID greater than 0.

If boundaries are defined, points could be located outside the defined grid. These points are called outliers and are assigned to the outlier cluster with ST\_ClusterID 0. ST\_ClusterEnvelope() for the outlier cluster is NULL.

SAP HANA spatial grid clustering uses the following parameters:

- **X CELLS <int> Y CELLS <int>** (mandatory)

The number of cells is X times Y

The number of clusters can be less than X times Y, because cells without points do not count as clusters.

For X = 10 and Y = 10, the situation could be:

- 100 = number of cells
- 81 = number of clusters (19 cells do not contain points)

The boundary of the overall grid rectangle is determined by the points to be investigated.

- **BETWEEN <left-point> AND <right-point>** (optional, X values)

Determines the X values of the overall grid rectangle regardless of the points to be investigated

- **BETWEEN <lower-point> AND <upper-point>** (optional, Y values)

Determines the Y values of the overall grid rectangle regardless of the points to be investigated

The following combinations of the mandatory and optional parameters are possible:

- **X CELLS <int> Y CELLS <int>**

(example: X CELLS 50 Y CELLS 50)

- **X BETWEEN <left-point> AND <right-point> CELLS <int> Y CELLS <int>**

(example: X BETWEEN 0 AND 100 CELLS 50 Y CELLS 50)

- **X CELLS <int> Y BETWEEN <lower-point> AND <upper-point> CELLS <int>**

(example: X CELLS 50 Y BETWEEN 0 AND 200 CELLS 50)

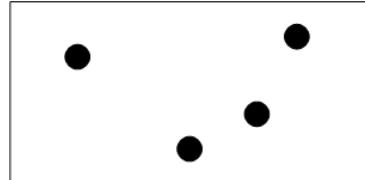
- **X BETWEEN <left-point> AND <right-point> CELLS <int> Y BETWEEN <lower-point> AND <upper-point> CELLS <int>**

(example: X BETWEEN 0 AND 100 CELLS 50 Y BETWEEN 0 AND 200 CELLS 50)

### Assignment rules for cluster points

Points which are inside a cell and not at a boundary are assigned to this cell. The assignment of points on the boundary need to be defined. The assignment of points is made according to the following rules (points with black fill belong to cell; points with white fill do not belong to the cell):

Table 89:

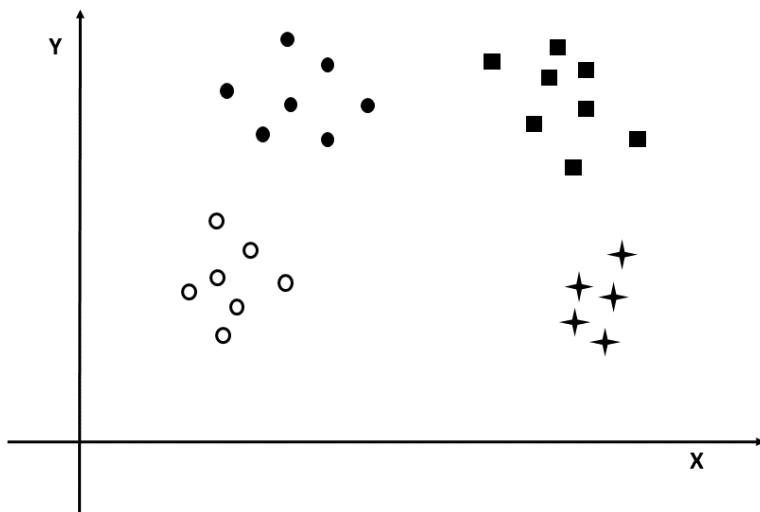
Position	Rule	Graphic
Points inside a cell	All points inside a cell belong to the cell.	

Position	Rule	Graphic
Boundary points of any cell	<p>All points on the left boundary of the cell except the point with the highest Y value belong to the cell. The point with the highest Y value belongs to the neighbor above.</p> <p>All points on the lower boundary of the cell except the point with the highest X value belong to the cell. The point with the highest X value belongs to the neighbor to the right.</p>	
Boundary points of all cells of the upper boundary of the overall grid rectangle (no neighbor above) except the upper right cell (see last rule)	All points on the upper boundary except the point with the highest X value belong to the cell.	
Boundary points of all cells of the right-hand boundary of the overall grid rectangle (no neighbor to the right) except the upper right cell (see last rule)	All points on the right-hand boundary of a cell belong to the cell, except the one with the highest Y value.	
Boundary points of the upper right cell (no neighbor above and no neighbor to the right)	All boundary points belong to the cell.	
Points completely outside the defined grid (outliers)	All outliers are assigned to the outlier cluster with ST_ClusterID 0.	

## 4.2.2 K-Means

SAP HANA spatial supports k-means clustering. The following methods are available for the k-means algorithm: ST\_ClusterID(), ST\_ClusterCentroid().

K-means is a clustering algorithm. It is best suited for spherical clusters. K-means is centroid based. The higher complexity of the algorithm provides better insights.



K-means tries to find an assignment of points to clusters, so that the sum of squared distances of the points to the center of the cluster they belong to is minimal. The initialization is performed randomly. RANDOM PARTITION starts by assigning every point to one of the random k clusters. FORGY starts with the random choice of k cluster centers.

The iterations now start. They try to find a better solution according to the sum of the squared distances. The iterations stops when MAXITERATIONS is reached, or the change of the sum of squared distances between two iterations is below the THRESHOLD.

### i Note

K-means is not deterministic. k-means might therefore produce different results with the same parameters on the same data set.

In SAP HANA spatial k-means uses the following parameters:

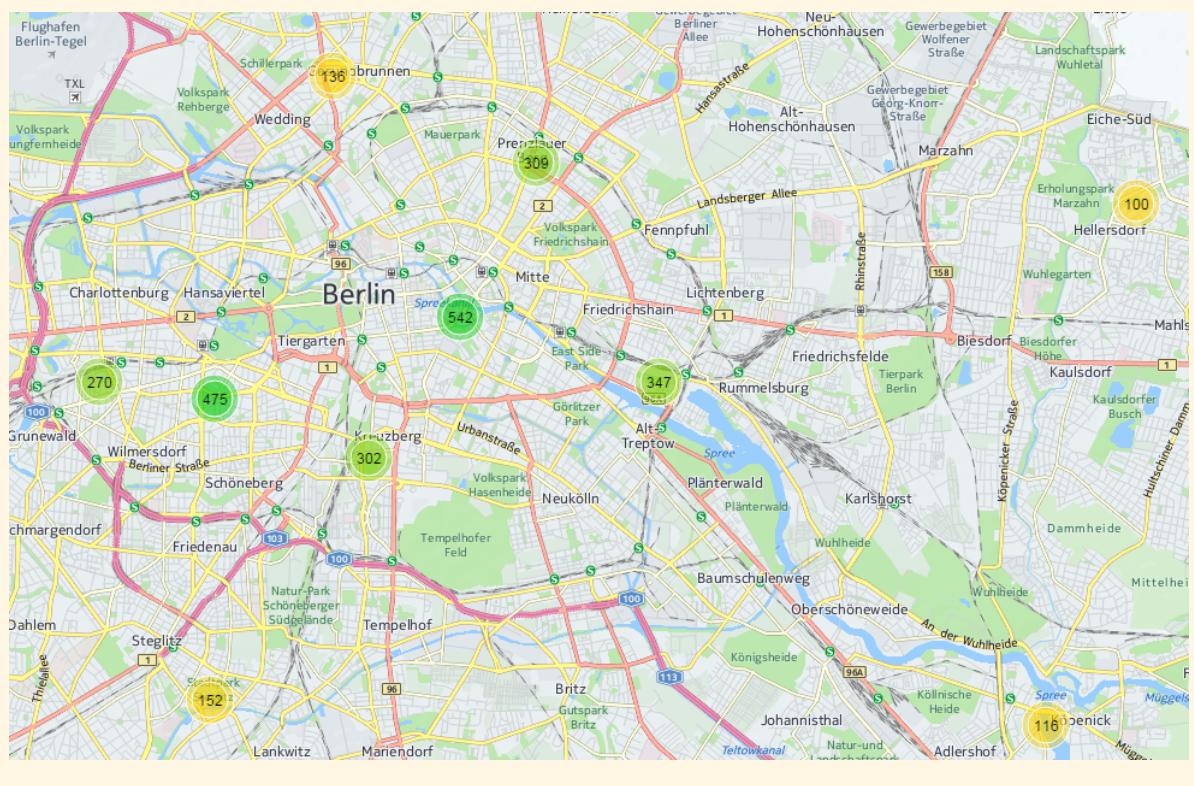
- **CLUSTERS <int>** (mandatory)  
Number of clusters to be created
- **MAXITERATIONS <int>** (optional, default 128)  
Maximum number of iterations to be performed
- **THRESHOLD <real>** (optional, default 0)  
Threshold for the change of the sum of the squared distances between two iterations
- **INIT RANDOM PARTITION** or **INIT FORGY** (optional, default FORGY)

Initialization method; FORGY is the recommended method

For more information about k-means, see [k-means clustering](#)

### Example

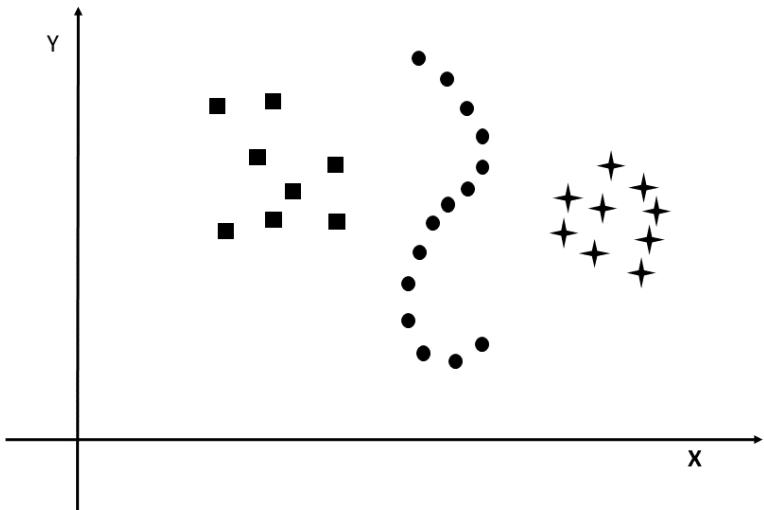
The following example shows clusters of restaurants in Berlin:



### 4.2.3 DBSCAN

SAP HANA spatial supports DBSCAN clustering. The following method is available for the DBSCAN algorithm: ST\_ClusterID().

Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm. DBSCAN is best suited for non-spherical clusters. The higher complexity of the algorithm provides better insights.



The set of points with a distance of less than or equal to EPS are called the **neighborhood** of this point. There are the following classes of points:

- If a data point has at least MINPTS data points in its neighborhood, including itself, it is called a **core point**.
- If a data point is within the neighborhood of a core point, but not a core point itself, it is called a **border point**.
- All other points, i.e. points that are not within the neighborhood of a core point, are called **outliers** or **noise**.

The points are grouped into clusters according to these rules:

- Two core points  $x$  and  $y$  belong to the same cluster if - and only if - there is a sequence of core points  $x = x_1, x_2, \dots, x_n = y$ , such that the distance of  $x_{(n-1)}$  and  $x_n$  is less than or equal to EPS; i.e. every point is in the neighborhood of its successor and predecessor.
- Border points are assigned to the same cluster that a core point in their neighborhood is assigned to. Note that this might not be uniquely defined; there might be more than one core point in the border points neighborhood.
- Outliers are assigned to the outlier cluster with ST\_ClusterID 0.

In SAP HANA, spatial DBSCAN uses the following parameters:

- **EPS <int>** (mandatory)  
Neighborhood radius
- **MINPTS <int>** (mandatory)  
The number of data points that must be contained in the neighborhood of a point in order to make it a core point

For more information about DBSCAN, see [DBSCAN](#)

## 4.3 SQL Syntax for Spatial Clustering

For spatial clustering, special clauses are provided in a SELECT statement.

### 4.3.1 Window Functions

Clustering functions are supported as SQL window functions.

#### Syntax

```
<spatial_window_function> ::= <spatial_window_func_type>
OVER ( <window_cluster_by_clause> [<window_order_by_clause>] )
```

The spatial window function clause is part of the select clause of a subquery. Example:

```
SELECT <spatial_window_function> <from_clause> ...
```

The spatial window function clause consists of:

- Spatial window function type
- Window cluster by clause
- Window order by clause

#### Spatial window function type

```
<spatial_window_func_type> ::= <window_func>           |
                             <aggregate_func>        |
                             <cluster_metadata_func>
```

There are the following spatial window function types:

- Window functions
- Aggregate functions
- Cluster metadata functions

```
<window_func> ::= all existing window_function_types
                  (for example ROW_NUMBER, RANK, FIRST_VALUE, NTH_VALUE, ...)
```

For a complete list of supported window functions, see *Window Functions in SAP HANA SQL and System Views Reference*.

```
<aggregate_func> ::= COUNT(*) | <agg_name> ( <expression> )
<agg_name> ::= COUNT | MIN | MAX | SUM | AVG | STDDEV | VAR
```

The spatial aggregation functions are also supported (for example ST\_AsSVGAgr). For information about the spatial aggregation functions, see the SAP HANA Spatial Reference.

```
<cluster_metadata_func> ::= ST_ClusterId() |  
                           ST_ClusterCentroid() |  
                           ST_ClusterEnvelope()
```

## Window cluster by clause

```
<cluster_by_clause> ::= GROUP CLUSTER BY <spatial_column_name>  
                        USING <cluster_algorithm>
```

The `<spatial_column_name>` is a table column of type ST\_POINT, or of type ST\_GEOMETRY containing only points.

```
<cluster_algorithm> ::= <dbscan> | <grid> | <kmeans>  
<dbscan> ::= DBSCAN  
              EPS <double>  
              MINPTS <integer>  
<grid>   ::= GRID  
              X [BETWEEN <left-point> AND <right-point>] CELLS <integer>  
              Y [BETWEEN <lower-point> AND <upper-point>] CELLS <integer>  
<kmeans> ::= KMEANS  
              CLUSTERS <integer>  
              [MAXITERATIONS <integer>]  
              [THRESHOLD <integer>]  
              [INIT RANDOM PARTITION | INIT FORGY]
```

## Window order by clause

```
<window_order_by_clause> ::= ORDER BY {<window_order_by_expression>}
```

```
<window_order_by_expression> ::= <expression> [ASC | DESC]  
                                [NULLS FIRST | NULLS LAST]  
                                [, <window_order_by_expression>]
```

## 4.3.2 GROUP CLUSTER BY

Clustering functions are supported as a GROUP CLUSTER BY clause.

### Syntax

```
<subquery> ::= <select_clause>
```

```
<from_clause>
[<where_clause>]
[<group_by_clause> | <cluster_by_clause>]
[<having_clause>]
[<set_operator> <subquery> [{, <set_operator> <subquery>}...]]
[<order_by_clause>]
[<limit>]
```

For spatial clustering, the CLUSTER BY clause has been added.

For more information, see *Select in the SAP HANA SQL and System Views Reference*.

## SELECT clause

In the select clause, you can use the methods for spatial clustering (ST\_ClusterID(), ST\_ClusterCentroid(), ST\_ClusterEnvelope()) that can be used to investigate the data provided by the cluster algorithms.

### Cluster by clause

```
<cluster_by_clause> ::= GROUP CLUSTER BY <spatial_column_name>
                      USING <cluster_algorithm>
```

The `<spatial_column_name>` is a table column of type ST\_POINT, or of type ST\_GEOMETRY containing only points.

### Cluster algorithm

```
<cluster_algorithm> ::= <dbSCAN> | <grid> | <kmeans>
<dbSCAN> ::= DBSCAN
            EPS <double>
            MINPTS <integer>
<grid>   ::= GRID
            X [BETWEEN <left-point> AND <right-point>] CELLS <integer>
            Y [BETWEEN <lower-point> AND <upper-point>] CELLS <integer>
<kmeans> ::= KMEANS
            CLUSTERS <integer>
            [MAXITERATIONS <integer>]
            [THRESHOLD <integer>]
            [INIT RANDOM PARTITION | INIT FORGY]
```

SAP HANA spatial provides spatial clustering using the algorithms grid, k-means, and DBSCAN.

## 4.4 Methods for Spatial Clustering

SAP HANA spatial provides methods to perform spatial clustering.

The following methods are provided:

- ST\_ClusterID() - cluster algorithms Grid, K-Means, DBSCAN
- ST\_ClusterCentroid() - cluster algorithm K-Means
- ST\_ClusterEnvelope() - cluster algorithm Grid

### 4.4.1 ST\_ClusterID Method

Returns the cluster ID of a given point. This method works for all cluster algorithms (Grid, K-Means, DBSCAN).

#### Syntax

```
ST_ClusterID()
```

#### Parameters

Table 90:

Name	Type	Description
Table column	ST_Point	Method ST_ClusterID operates on table columns of type ST_Point.

#### Returns

**Metadata** Returns the cluster ID for a given point.

#### Example

The following example returns cluster IDs and the number of points for each cluster:

```
SELECT ST_ClusterID() AS cluster_id, COUNT(*) AS counter
FROM CLUSTER.VENDING_MACHINES
GROUP CLUSTER BY LOCATION
USING GRID X CELLS 10 Y CELLS 10;
```

## 4.4.2 ST\_ClusterCentroid Method

Computes the ST\_Point. This is the mathematical centroid of a cluster. This method works for the following cluster algorithm: K-Means.

### Syntax

```
ST_ClusterCentroid()
```

### Parameters

Table 91:

Name	Type	Description
Table column	ST_Point	Method ST_ClusterCentroid operates on table columns of type ST_Point.

### Returns

**Metadata** Returns the centroid of a cluster.

#### Example

The following example returns cluster centroids and the number of points per cluster:

```
SELECT ST_ClusterCentroid() AS centroidcluster,  
COUNT(*) AS counter  
FROM CLUSTER.VENDING_MACHINES  
GROUP CLUSTER BY LOCATION  
USING KMEANS  
CLUSTERS 5  
MAXITERATIONS 10  
THRESHOLD 0.01  
INIT RANDOM PARTITION;
```

### 4.4.3 ST\_ClusterEnvelope Method

Returns the geometry of a cell (inner rectangle). This method works for the following cluster algorithm: Grid.

#### Syntax

```
ST_ClusterEnvelope ()
```

#### Parameters

Table 92:

Name	Type	Description
Table column	ST_Point	The method ST_Envelope operates on table columns of type ST_Point.

#### Returns

**Metadata** Returns the envelope of a cluster.

#### Example

The following example returns the cluster IDs and the cluster envelopes:

```
SELECT ST_ClusterID() AS cluster_id,
       ST_ClusterEnvelope() AS clusterenvelope
  FROM CLUSTER.VENDING_MACHINES
 GROUP CLUSTER BY LOCATION
  USING GRID X CELLS 10 Y CELLS 10;
```

## 4.5 Use Cases for Spatial Clustering

Spatial clustering provides metadata, so that geographical information can be used to group data.

You can investigate and group data by many different attributes. You can investigate the income of households or the revenue of vending machines for example. Columns containing attributes (for example CITY, STATE, COUNTRY) used for grouping data (GROUP BY/PARTITION BY) provide basic information, but this might not be sufficient.

With spatial clustering (GROUP CLUSTER BY, CLUSTER BY) you can use geographical information together with other attributes (income and revenue for example) to cluster data.

## Finding clusters with high or low household income

A company wants to establish a new chain of luxury stores and a new chain of discounters in the United States. To find suitable locations for their luxury stores, the company would like to identify areas with a preponderance of high-income households. For their discounters, they would like to find areas with a preponderance of low-income households.

The company has the data for a total of 120 million households in the United States, including their location and income. The management team wants to visualize areas with a preponderance of high-income households and areas with a preponderance of low-income households on a map.

A simplified table containing the household data could be defined as following:

```
CREATE COLUMN TABLE HOUSEHOLDS (
    HHID      INT PRIMARY KEY,
    STATE     CHAR(2),
    LOCATION  ST_POINT,
    INCOME    DECIMAL(11, 2));
```

The following statements show examples of how the necessary information can be retrieved using GROUP BY or CLUSTER BY.

### GROUP BY

A grouping on a pre-defined area (in this case state) could be defined as:

```
SELECT
    STATE,
    COUNT(*)      AS number_of_households,
    AVG(INCOME)   AS household_income
FROM HOUSEHOLDS
WHERE INCOME > 120000
GROUP BY STATE
HAVING COUNT(*) >= 1000;
```

This query finds the number of households with an income of more than \$120,000 per state and the average income of these households with more than \$120,000 income as long as there are at least 1,000 of these households in the state. The STATE is the key to further information of this area. The shape and the centroid of the state could be located in an extra table accessed via the state abbreviation.

### GROUP CLUSTER BY

With spatial clustering, you investigate the information about the household income in conjunction with a cluster algorithm. The following query provides information about the cluster centroids, which can give you a first impression of the situation:

```
SELECT
    ST_ClusterId()      AS cluster_id,
    ST_ClusterCentroid() AS centroid,
    COUNT(*)            AS number_of_households,
    AVG(INCOME)         AS average_cluster_income
FROM HOUSEHOLDS
WHERE INCOME > 120000
```

```
GROUP CLUSTER BY LOCATION USING KMEANS CLUSTERS 100  
HAVING COUNT(*) >= 300;
```

With the following statement, you can combine all centroids to one scalable vector graphic:

```
SELECT ST_AsSVGAgr(centroid) from  
(SELECT  
    ST_ClusterId()      AS cluster_id,  
    ST_ClusterCentroid() AS centroid,  
    COUNT(*)             AS number_of_households,  
    AVG(INCOME)          AS average_cluster_income  
FROM HOUSEHOLDS  
WHERE INCOME > 120000  
GROUP CLUSTER BY LOCATION USING KMEANS CLUSTERS 100  
HAVING COUNT(*) >= 300);
```

## Finding clusters with high or low vending machine revenue

A vending machine supplier has its vending machines distributed and operating across the United States. The company knows the location of each vending machine and its revenue from the past year.

It would like to identify lucrative areas where the vending machines produced higher revenue and non-lucrative areas where its machines produced low revenue. It wants to visualize these areas on a map. It also wants to be able to zoom into these areas, and be able to retrieve detailed information, such as the ID and revenue of each vending machine.

A simplified table containing the vending machine data could be defined as:

```
CREATE COLUMN TABLE VENDING_MACHINES (  
    VM_ID      INT PRIMARY KEY,  
    STATE       CHAR(2),  
    LOCATION    ST_POINT,  
    REVENUE    DECIMAL(11,2));
```

The following statements show examples of how the required information can be retrieved using GROUP BY/PARTITION BY or CLUSTER BY in a window function.

### GROUP BY/PARTITION BY

With GROUP BY/PARTITION BY, you can investigate the data on a pre-defined group (in this case state).

The following query finds the number and average revenue of vending machines per state, which have less than \$60,000 revenue.

```
SELECT  
    STATE,  
    COUNT(*)      AS number_of_vending_machines,  
    AVG(REVENUE)  AS average_revenue  
FROM VENDING_MACHINES  
WHERE REVENUE < 60000  
GROUP BY STATE;
```

The following query finds the rank of vending machines per state, which have less than \$60,000 revenue, and ranks them by revenue in ascending order.

```
SELECT  
    STATE,
```

```

        RANK() OVER (PARTITION BY STATE ORDER BY REVENUE ASC) AS RANK,
        REVENUE,
        VM_ID
    FROM VENDING_MACHINES
    WHERE REVENUE < 60000;

```

## CLUSTER BY (window functions)

You can perform the spatial clustering as a window function. The cluster by clause is located in the select clause.

In the following query, the clusters for vending machines with less than \$60,000 are determined:

```

SELECT
    ST_ClusterID() OVER (CLUSTER BY LOCATION USING DBSCAN EPS 100 MINPTS 5)
    AS cluster_id,
    REVENUE, VM_ID, LOCATION
FROM VENDING_MACHINES
WHERE REVENUE < 60000
ORDER BY cluster_id;

```

In the following query, ST\_ClusterID() and RANK() are combined. In the query in brackets, ST\_ClusterID() is performed as a window function, while in the surrounding query RANK() is performed on cluster\_id. The vending machines are grouped by cluster and ranked inside the clusters by revenue:

```

SELECT cluster_id, REVENUE, VM_ID,
        RANK() OVER (PARTITION BY cluster_id ORDER BY REVENUE ASC) AS rank,
        LOCATION
    FROM (SELECT ST_ClusterID() OVER (CLUSTER BY LOCATION USING DBSCAN EPS 100
    MINPTS 5)
        AS cluster_id,
        REVENUE, VM_ID, LOCATION
        FROM VENDING_MACHINES
        WHERE REVENUE < 60000)
    ORDER BY cluster_id;

```

In the following query ST\_ClusterID() and RANK() are combined in the select clause. ST\_ClusterID() (CLUSTER BY LOCATION ...) and RANK() (CLUSTER BY LOCATION ... ORDER BY REVENUE) are performed as window functions. The vending machines are grouped by cluster and ranked inside the clusters by revenue:

```

SELECT
    ST_ClusterID()
    OVER (CLUSTER BY LOCATION USING DBSCAN EPS 100 MINPTS 5) AS cluster_id,
    RANK()
    OVER (CLUSTER BY LOCATION USING DBSCAN EPS 100 MINPTS 5
    ORDER BY REVENUE ASC) AS rank,
    REVENUE, VM_ID, LOCATION
    FROM VENDING_MACHINES
    WHERE REVENUE < 60000;

```

## Comparison of GROUP BY and CLUSTER BY

The way spatial clustering and window functions are used is similar. This is a comparison using the examples above:

1. You group your data set by a column (for example STATE) or you determine geographical clusters using an algorithm on a column containing geospatial points.

2. Within the group or the cluster, you order the data by a column (for example REVENUE).
3. Within the group or the cluster, you rank the data (starting with 1 for the highest REVENUE for example ).

**i Note**

The main difference between the two methods is:

- GROUP BY groups the data set using information contained in the column(s).
- CLUSTER BY splits the data set into clusters, which are determined by geospatial point data using an algorithm.

Further investigations on the data set are similar. Using spatial clustering provides a benefit however, because it helps to investigate data from a geospatial point of view as well.

# 5 Appendix

The appendix provides additional information.

## 5.1 SQL Statements

Reference material for SQL statements mentioned in this document.

### 5.1.1 CREATE SPATIAL REFERENCE SYSTEM Statement

Creates a spatial reference system.

#### Syntax

```
CREATE SPATIAL REFERENCE SYSTEM <srs-name>
[ srs-attribute ] [ srs-attribute ... ]
srs-attribute - (back to Syntax)
SRID <srs-id>
| DEFINITION { <definition-string> | NULL }
| ORGANIZATION
{ <organization-name> IDENTIFIED BY <organization-srs-id> | NULL }
| TRANSFORM DEFINITION { <transform-definition-string> | NULL }
| LINEAR UNIT OF MEASURE <linear-unit-name>
| ANGULAR UNIT OF MEASURE { <angular-unit-name> | NULL }
| TYPE { ROUND EARTH | PLANAR }
| COORDINATE <coordinate-name>
{ UNBOUNDED | BETWEEN <low-number> AND <high-number> }
| ELLIPSOID SEMI MAJOR AXIS <semi-major-axis-length>
{ SEMI MINOR AXIS <semi-minor-axis-length> |
  INVERSE FLATTENING <inverse-flattening-ratio> }
| TOLERANCE { <tolerance-distance> | DEFAULT }
| SNAP TO GRID { grid-size | DEFAULT }
| AXIS ORDER axis-order
| POLYGON FORMAT polygon-format
| STORAGE FORMAT storage-format
grid-size - (back to srs-attribute)
DOUBLE : usually between 0 and 1
axis-order - (back to srs-attribute)
{ 'x/y/z/m' | 'long/lat/z/m' | 'lat/long/z/m' }
polygon-format - (back to srs-attribute)
{ 'CounterClockWise' | 'Clockwise' | 'EvenOdd' }
storage-format - (back to srs-attribute)
{ 'Internal' | 'Original' | 'Mixed' }
```

## Parameters

Table 93:

Name	Description
IDENTIFIED BY	<p>The SRID (srs-id) for the spatial reference system.</p> <p>If the spatial reference system is defined by an organization with an organization-srs-id, then srs-id should be set to that value.</p> <p>If the IDENTIFIED BY clause is not specified, then the SRID defaults to the organization srs-id defined by either the ORGANIZATION clause or the DEFINITION clause. If neither clause defines an organization-srs-id that could be used as a default SRID, an error is returned.</p> <p>When the spatial reference system is based on a well known coordinate system, but has a different geodesic interpretation, set the srs-id value to be 1000000000 (one billion) plus the well known value. For example, the SRID for a planar interpretation of the geodetic spatial reference system WGS 84 (ID 4326) would be 1000004326.</p> <p>With the exception of SRID 0, spatial reference systems provided by SAP HANA that are not based on well known systems are given a SRID of 2000000000 (two billion) and above. The range of SRID values from 2000000000 to 2147483647 is reserved by SAP HANA and you should not create SRIDs in this range.</p> <p>To reduce the possibility of choosing a SRID that is reserved by a defining authority such as OGC or by other vendors, you should not choose a SRID in the range 0 - 32767 (reserved by EPSG), or in the range 2147483547 - 2147483647.</p> <p>Also, since the SRID is stored as a signed 32-bit integer, the number cannot exceed <math>(2^{31})-1</math> or 2147483647.</p>
DEFINITION	<p>Set the coordinate system definition.</p> <p><code>&lt;definition-string&gt;</code> is a string in the Spatial Reference System Well Known Text syntax as defined by SQL/MM and OGC.</p>
ORGANIZATION	Information about the organization that created the spatial reference system that the spatial reference system is based on.
TRANSFORM DEFINITION	A description of the transform to use for the spatial reference system. Currently, only the PROJ.4 transform is supported. The transform definition is used by the ST_Transform method when transforming data between spatial reference systems. Some transforms may still be possible even if there is no transform definition-string defined.

Name	Description
LINEAR UNIT OF MEASURE	<p>The linear unit of measure for the spatial reference system.</p> <p>The value you specify must match a linear unit of measure defined in the ST_UNITS_OF_MEASURE system view.</p> <p>If this clause is not specified, and is not defined in the DEFINITION clause, the default is METRE. To add predefined units of measure to the database, use the sa_install_feature system procedure.</p> <p>To add custom units of measure to the database, use the CREATE SPATIAL UNIT OF MEASURE statement.</p> <div style="background-color: #ffffcc; padding: 10px;"> <p><b>i Note</b></p> <p>While both METRE and METER are accepted spellings, METRE is preferred as it conforms to the SQL/MM standard.</p> </div>
ANGULAR UNIT OF MEASURE	<p>The angular unit of measure for the spatial reference system.</p> <p>The value you specify must match an angular unit of measure defined in the ST_UNITS_OF_MEASURE system table.</p> <p>If this clause is not specified, and is not defined in the DEFINITION clause, the default is DEGREE for geographic spatial reference systems and NULL for non-geographic spatial reference systems.</p> <p>The angular unit of measure must be non-NULL for geographic spatial reference systems and it must be NULL for non-geographic spatial reference systems.</p> <p>To add custom units of measure to the database, use the CREATE SPATIAL UNIT OF MEASURE statement.</p>

Name	Description
TYPE	<p>Control how the SRS interprets lines between points.</p> <p>For geographic spatial reference systems, the TYPE clause can specify either ROUND EARTH (the default) or PLANAR. The ROUND EARTH model interprets lines between points as great elliptic arcs. Given two points on the surface of the Earth, a plane is selected that intersects the two points and the center of the Earth. This plane intersects the Earth, and the line between the two points is the shortest distance along this intersection.</p> <p>For two points that lie directly opposite each other, there is not a single unique plane that intersects the two points and the center of the Earth. Line segments connecting these antipodal points are not valid and give an error in the ROUND EARTH model.</p> <p>The ROUNDEARTH model treats the Earth as a spheroid and selects lines that follow the curvature of the Earth. In some cases, it may be necessary to use a planar model where a line between two points is interpreted as a straight line in the equirectangular projection where <math>x=long</math>, <math>y=lat</math>.</p> <p>In the following example, the blue line shows the line interpretation used in the ROUND EARTH model and the red line shows the corresponding PLANAR model.</p> <p>The PLANAR model may be used to match the interpretation used by other products. The PLANAR model may also be useful because there are some limitations for methods that are not supported in the ROUND EARTH model (such as ST_Area, ST_ConvexHull) and some are partially supported (ST_Distance only supported between point geometries). Geometries based on circularstrings are not supported in ROUND EARTH spatial reference systems.</p> <p>For non-geographic SRSs, the type must be PLANAR (and that is the default if the TYPE clause is not specified and either the DEFINITION clause is not specified or it uses a non-geographic definition).</p>

Name	Description
COORDINATE	<p>The bounds on the spatial reference system's dimensions.</p> <p>Coordinate name is the name of the coordinate system used by the spatial reference system. For nongeographic coordinate systems, coordinate-name can be x, y, or m. For geographic coordinate systems, coordinate-name can be LATITUDE, LONGITUDE, z, or m.</p> <p>Specify UNBOUNDED to place no bounds on the dimensions. Use the BETWEEN clause to set low and high bounds.</p> <p>The X and Y coordinates must have associated bounds. For geographic spatial reference systems, the longitude coordinate is bounded between -180 and 180 degrees and the latitude coordinate is bounded between -90 and 90 degrees by default unless the COORDINATE clause overrides these settings. For non-geographic spatial reference systems, the CREATE statement must specify bounds for both X and Y coordinates.</p> <p>LATITUDE and LONGITUDE are used for geographic coordinate systems. The bounds for LATITUDE and LONGITUDE default to the entire Earth, if not specified.</p>
ELLIPSOID	<p>The values to use for representing the Earth as an ellipsoid for spatial reference systems of type ROUND EARTH. If the DEFINITION clause is present, it can specify ellipsoid definition.</p> <p>If the ELLIPSOID clause is specified, it overrides this default ellipsoid.</p> <p>The Earth is not a perfect sphere because the rotation of the Earth causes a flattening so that the distance from the center of the Earth to the North or South pole is less than the distance from the center to the equator. For this reason, the Earth is modeled as an ellipsoid with different values for the semi-major axis (distance from center to equator) and semi-minor axis (distance from center to the pole). It is most common to define an ellipsoid using the semi-major axis and the inverse flattening, but it can instead be specified using the semi-minor axis (for example, this approach must be used when a perfect sphere is used to approximate the Earth). The semi-major and semi-minor axes are defined in the linear units of the spatial reference system, and the inverse flattening (<math>1/f</math>) is a ratio:</p> <div style="background-color: #f0f0f0; padding: 5px;"> <math display="block">1/f = (\text{semi-major-axis}) / (\text{semi-major-axis} - \text{semi-minor-axis})</math> </div> <p>SAP HANA uses the ellipsoid definition when computing distance in geographic spatial reference systems.</p>
SNAP TO GRID	<p>Flat-Earth (planar) spatial reference systems, use the SNAP TO GRID clause to define the size of the grid SAP HANA uses when performing calculations.</p> <p>By default, SAP HANA selects a grid size so that 12 significant digits can be stored at all points in the space bounds for X and Y. For example, if a spatial reference system bounds X between -180 and 180 and Y between -90 and 90, then a grid size of 0.000000001 (1E-9) is selected.</p>

Name	Description
TOLERANCE	<p>Flat-Earth (planar) spatial reference systems, use the TOLERANCE clause to specify the precision to use when comparing points.</p> <p>If the distance between two points is less than tolerance-distance, the two points are considered equal. Setting tolerance-distance allows you to control the tolerance for imprecision in the input data or limited internal precision. By default, tolerance-distance is set to be equal to grid-size.</p> <p>When set to 0, two points must be exactly equal to be considered equal.</p> <p>For round-Earth spatial reference systems, TOLERANCE must be set to 0.</p>
POLYGON FORMAT	<p>Internally, SAP HANA interprets polygons by looking at the orientation of the constituent rings. As one travels a ring in the order of the defined points, the inside of the polygon is on the left side of the ring. The same rules are applied in PLANAR and ROUND EARTH spatial reference systems. The interpretation used by SAP HANA is a common but not universal interpretation. Some products use the exact opposite orientation, and some products do not rely on ring orientation to interpret polygons. The POLYGONFORMAT clause can be used to select a polygon interpretation that matches the input data, as needed. The following values are supported:</p> <ul style="list-style-type: none"> <li>• <b>CounterClockwise</b> – Input follows SAP HANA's internal interpretation: the inside of the polygon is on the left side while following ring orientation.</li> <li>• <b>Clockwise</b> – Input follows the opposite of SAP HANA's approach: the inside of the polygon is on the right side while following ring orientation.</li> <li>• <b>EvenOdd</b> – (default) The orientation of rings is ignored and the inside of the polygon is instead determined by looking at the nesting of the rings, with the exterior ring being the largest ring and interior rings being smaller rings inside this ring. A ray is traced from a point within the rings and radiating outward crossing all rings. If the number the ring being crossed is an even number, it is an outer ring. If it is odd, it is an inner ring.</li> </ul>
STORAGE FORMAT	<p>STORAGE FORMAT – control what is stored when spatial data is loaded into the database. Possible values are:</p> <ul style="list-style-type: none"> <li>• <b>Internal</b> – SAP HANA stores only the normalized representation. Specify this when the original input characteristics do not need to be reproduced. This is the default for planar spatial reference systems (TYPE PLANAR).</li> <li>• <b>Original</b> – SAP HANA stores only the original representation. The original input characteristics can be reproduced, but all operations on the stored values must repeat normalization steps, possibly slowing down operations on the data.</li> <li>• <b>Mixed</b> – SAP HANA stores the internal version and, if it is different from the original version, SAP HANA stores the original version as well. By storing both versions, the original representation characteristics can be reproduced and SAP HANA operations on stored values do not need to repeat normalization steps. However, storage requirements may increase significantly because potentially two representations are being stored for each geometry. Mixed is the default format for round-Earth spatial reference systems (TYPE ROUND EARTH).</li> </ul>



## Example

Creates a spatial reference system named <mySpatialRS>:

```
CREATE SPATIAL REFERENCE SYSTEM "mySpatialRS"
IDENTIFIED BY 1000026980
LINEAR UNIT OF MEASURE "meter"
TYPE PLANAR
COORDINATE X BETWEEN 171266.736269555 AND 831044.757769222
COORDINATE Y BETWEEN 524881.608973277 AND 691571.125115319
DEFINITION 'PROJCS["NAD83 / Kentucky South",
GEOGCS["NAD83",
DATUM["North_American_Datum_1983",
SPHEROID["GRS 1980",
6378137,298.257222101,AUTHORITY["EPSG","7019"]],,
AUTHORITY["EPSG","6269"]],,
PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],,
UNIT["degree",0.0174532951994328,AUTHORITY["EPSG","9122"]],,
AUTHORITY["EPSG","4269"]],,
UNIT["metre",1,AUTHORITY["EPSG","9001"]],,
PROJECTION["Lambert_Conformal_Conic_2SP"],
PARAMETER["standard_parallel_1",37.93333333333333],
PARAMETER["standard_parallel_2",36.73333333333333],
PARAMETER["latitude_of_origin",36.3333333333334],
PARAMETER["central_meridian",-85.75],
PARAMETER["false_easting",500000],
PARAMETER["false_northing",500000],
AUTHORITY["EPSG","26980"]],
AXIS["X",EAST],
AXIS["Y",NORTH]]'
TRANSFORM DEFINITION '+proj=lcc
+lat_1=37.93333333333333 +lat_2=36.73333333333333
+lat_0=36.3333333333334 +lon_0=-85.75 +x_0=500000
+y_0=500000 +ellps=GRS80 +datum=NAD83 +units=m +no_defs';
```

## Usage

For a geographic spatial reference system, you can specify both a LINEAR and an ANGULAR unit of measure; otherwise for non-geographic, you specify only a LINEAR unit of measure. The LINEAR unit of measure is used for computing distance between points and areas. The ANGULAR unit of measure tells how the angular latitude/longitude are interpreted and is NULL for projected coordinate systems, non-NULL for geographic coordinate systems.

All derived geometries returned by operations are normalized.

When working with data that is being synchronized with a non-SQL Anywhere database, STORAGE FORMAT should be set to either 'Original' or 'Mixed' so that the original characteristics of the data can be preserved.

## Permissions

Requires one of:

- MANAGE ANY SPATIAL OBJECT system privilege.

- CREATE ANY OBJECT system privilege.

## 5.1.2 CREATE SPATIAL UNIT OF MEASURE Statement

Creates a spatial unit of measurement.

### Syntax

```
CREATE SPATIAL UNIT OF MEASURE <identifier>
TYPE { LINEAR | ANGULAR }
[ CONVERT USING <number> ]
```

### Parameters

Table 94:

Name	Description
TYPE	Defines whether the unit of measure is used for angles (ANGULAR) or distances (LINEAR).
CONVERT USING	The conversion factor for the spatial unit relative to the base unit. For linear units, the base unit is METRE. For angular units, the base unit is RADIANS.

### Example

The following examples create different types of spatial units of measure:

```
CREATE SPATIAL UNIT OF MEASURE "meter" TYPE LINEAR convert using 1;

CREATE SPATIAL UNIT OF MEASURE "radian" TYPE ANGULAR convert using 1;

CREATE SPATIAL UNIT OF MEASURE "degree" TYPE ANGULAR convert using 0.017453293;

CREATE SPATIAL UNIT OF MEASURE "planar degree" TYPE LINEAR convert using 111120;

CREATE SPATIAL UNIT OF MEASURE "kilometre" TYPE LINEAR convert using 1000;

CREATE SPATIAL UNIT OF MEASURE "yard" TYPE LINEAR convert using 0.9144;
```



## Example

Creates a spatial unit of measure named <Test>:

```
CREATE SPATIAL UNIT OF MEASURE Test
  TYPE LINEAR
  CONVERT USING 15;
```

## Usage

The CONVERT USING clause is used to define how to convert a measurement in the defined unit of measure to the base unit of measure (radians or meters). The measurement is multiplied by the supplied conversion factor to get a value in the base unit of measure. For example, a measurement of 512 millimeters would be multiplied by a conversion factor of 0.001 to get a measurement of 0.512 metres.

Spatial reference systems always include a linear unit of measure to be used when calculating distances (ST\_Distance or ST\_Length), or area. For example, if the linear unit of measure for a spatial reference system is miles, then the area unit used is square miles. In some cases, spatial methods accept an optional parameter that specifies the linear unit of measure to use. For example, if the linear unit of measure for a spatial reference system is in miles, you could retrieve the distance between two geometries in meters by using the optional parameter 'metre'.

For projected coordinate systems, the X and Y coordinates are specified in the linear unit of the spatial reference system. For geographic coordinate systems, the latitude and longitude are specified in the angular units of measure associated with the spatial reference system. In many cases, this angular unit of measure is degrees but any valid angular unit of measure can be used.

You can use the sa\_install\_feature system procedure to add predefined units of measure to your database.

## Permissions

Requires one of:

- MANAGE ANY SPATIAL OBJECT system privilege.
- CREATE ANY OBJECT system privilege.

## 5.1.3 DROP SPATIAL REFERENCE SYSTEM Statement

Drops a spatial reference system.

### Syntax

```
DROP SPATIAL REFERENCE SYSTEM <srs-name>
```

### Parameters

Table 95:

Name	Description
<srs-name>	Identifier of the spatial reference system that has to be dropped.

### Remarks

You can only drop a spatial reference system, if it is not referenced by other objects.

#### Example

The following example drops a fictitious spatial reference system named <Test>.

```
DROP SPATIAL REFERENCE SYSTEM Test;
```

## 5.1.4 DROP SPATIAL UNIT OF MEASURE Statement

Drops a spatial unit of measurement.

### Syntax

```
DROP SPATIAL UNIT OF MEASURE <identifier>
```

## Parameters

Table 96:

Name	Description
<identifier>	Identifier of the unit of measurement that has to be dropped.

## Remarks

You can only drop a spatial unit of measurement, if it is not referenced by other objects.

### Example

The following example drops a fictitious spatial unit of measurement named <Test>.

```
DROP SPATIAL UNIT OF MEASURE Test;
```

## 5.2 Geocoding

Geocoding is the process of converting address information into geographic locations. Since current business data is associated with address information without location information, geocoding is crucial to enable geographic analysis of this data.

For geocoding, you use the index type `GEOCODE`, which is similar to the `FULLTEXT` index. To create, maintain and delete an index for geocoding, SQL statements are used.

### Geocoding Using SAP HANA Smart Data Quality Within SAP HANA Spatial

Use the SAP HANA smart data quality option to provide geocoding information. This is a solution that has high performance with no third party data and no need to use a web service, and it is not necessary to create a flowgraph using Web-based Development Workbench. You may need a subscription to reference data from SAP.

### Geocoding Using Third Party Data

Geocoding can also be provided by an external provider such as Nokia HERE or TOMTOM, while the outbound connectivity required is maintained and executed asynchronously in the XS engine. The index server provides

jobs for geocoding, which are triggered by the table update. The geocoding processing is very similar to text processing, with the difference that multiple columns are used as an input, only asynchronous processing is supported, and the geocode column (the column which holds the converted addresses) is visible. The GEOCODE column is visible for searches, but must not be altered or dropped.

To specify which columns correspond semantically to an address, each individual part must be explicitly defined.

## Geocoding Using SAP HANA Smart Data Quality Outside of SAP HANA Spatial

SAP HANA smart data quality provides native geocoding and reverse geocoding capabilities in SAP HANA Web-based Development Workbench. You can create flowgraphs using the Geocode node. You may need to license additional software options as well as a subscription to reference data from SAP, which is used by smart data quality to perform the geocoding and reverse geocoding operations. SAP HANA smart data quality also includes data cleansing capabilities which is able to cleanse and enhance address information to improve address geocoding capabilities natively in SAP HANA.

To find more information about how to use these capabilities natively in SAP HANA refer to the [SAP HANA Smart Data Integration and SAP HANA Smart Data Quality Configuration Guide](#), and navigate to

▶ [Transforming Data](#) ▶ [Geocode](#) ▷

### 5.2.1 Create Index

To create a geocoding index, use the following SQL statement:

```
CREATE GEOCODE INDEX <name> ON <table_name> (
<column_name> COUNTRY,
<column_name> STATE,
<column_name> COUNTY,
<column_name> CITY,
<column_name> POSTAL_CODE,
<column_name> DISTRICT,
<column_name> STREET,
<column_name> HOUSE_NUMBER,
<column_name> ADDRESS_LINE,
<column_name> GEOCODE )
ASYNC[HRONOUS] [FLUSH [QUEUE] [EVERY <n> MINUTES] [[OR] AFTER <m> ROWS]]]
```

#### i Note

When using a third-party geocode provider, at least one address column must be specified in addition to the GEOCODE column. Specifying any additional columns is optional. When using the SAP HANA smart data quality geocode, then you must map STREET, LOCALITY, POSTAL\_CODE and COUNTRY.

Table 97:

Name	Description
COUNTRY	This column holds the country name of an address. The data type has to be character like (internally represented as string). If the column contains a value, it will be used for geocoding. If there is no value, or the value is null, the country is ignored. The country name may be fully spelled, the 2-character ISO country code, or the 3-character ISO country code.
STATE	This column holds the state name of an address. The data type has to be character like (internally represented as string). If the column contains a value, it will be used for geocoding. If there is no value, or the value is null, the state is ignored.
COUNTY	This column holds the county name of an address. The data type has to be character like (internally represented as string). If the column contains a value it will be used for geocoding. If there is no value, or the value is null, the county is ignored.
CITY	This column holds the city name of an address. The data type has to be character like (internally represented as string). If the column contains a value it will be used for geocoding. If there is no value, or the value is null, the city is ignored.
POSTAL_CODE	This column holds the postal code of an address. The data type has to be character like (internally represented as string). If the column contains a value it will be used for geocoding. If there is no value, or the value is null, the postal code is ignored.
DISTRICT	This column holds the district name of an address. The data type has to be character like (internally represented as string). If the column contains a value it will be used for geocoding. If there is no value, or the value is null, the district is ignored.
STREET	This column holds the primary portion of the street address, such as the components that make up the name of the street such as the directional (North, East, and so on). The data type has to be character like (internally represented as string). If the column contains a value it will be used for geocoding. If there is no value, or the value is null, the street is ignored.
HOUSE_NUMBER	This column holds the house number of an address. The house number may be either located in this column, or combined with the street data in the STREET column. The data type has to be character like (internally represented as string). If the column contains a value it will be used for geocoding. If there is no value, or the value is null, the house number is ignored.
ADDRESS_LINE	This column holds a several address parts in a single value such as the building name, floor number, or unit number. Sometimes this data is located in the STREET column. The data type has to be character like (internally represented as string). If the column contains a value it will be used for geocoding; in case of no or null value the address line is ignored. The ADDRESS_LINE is required as address data is not fully structured in most cases. In extreme cases, a single field stores the entire address information.
GEOCODE	The name of the generated point column that holds the geo-coded location. The data type of the GEOCODE column is ST_POINT.
ASYNC: EVERY <n> MINUTES	Preprocessing and insertion into geocode index does not take place immediately upon insert-call, but rather every n minutes.
ASYNC: AFTER <m> ROWS	Preprocessing and insertion into geocode index does not take place immediately upon insert-call, but rather after m new rows have been inserted.

## 5.2.2 Modify Index

Changing the criteria is only permitted for queue processing.

```
ALTER GEOCODE INDEX <name> ASYNC[HRONOUS] [FLUSH [QUEUE] [EVERY <n> MINUTES]
[[OR] AFTER <m> ROWS]]]
```

## 5.2.3 Drop Index

The SQL statement to drop a geocoding index is similar to the statement for the FULLTEXT index.

```
DROP GEOCODE INDEX <name>
```

## 5.2.4 Maintain the Queue

The SQL statements to maintain the geocoding queue are similar to the statements for the FULLTEXT index.

Stop queue processing:

```
ALTER GEOCODE INDEX <name> SUSPEND QUEUE
```

Continue processing of queue content:

```
ALTER GEOCODE INDEX <name> ACTIVATE QUEUE
```

Force processing of queue content:

```
ALTER GEOCODE INDEX <name> FLUSH QUEUE
```

Retry failed geocoding jobs:

```
CALL RETRY_DOCUMENT_INDEXING (<schema>,<name>,<error code>)
```

### i Note

Only error code 0 is allowed. If the <error code> is 0, any errors will be queued for retry.

## 5.2.5 System Tables and Monitoring View

Information about geocoding activities is stored in system tables and a monitoring view.

### System Table INDEXES

A geocode index is registered in the `INDEXES` table with type `GEOCODE`.

### System Table INDEX\_COLUMNS

Columns that are part of a geocode index are registered in the `INDEX_COLUMNS` table and are distinguished by the constraint column.

Table 98:

Index part	Constraint
COUNTRY	GEOCODE COUNTRY
STATE	GEOCODE STATE
CITY	GEOCODE CITY
POSTAL_CODE	GEOCODE POSTAL_CODE
DISTRICT	GEOCODE DISTRICT
STREET	GEOCODE STREET
HOUSE_NUMBER	GEOCODE HOUSE_NUMBER
ADDRESS_LINE	GEOCODE ADDRESS_LINE
GEOCODE	GEOCODE

### System Table GEOCODE\_INDEXES

Detailed information about the geocode indexes in a system is registered in the `GEOCODE_INDEXES` table.

Table 99:

Columns name	Data type	Description
SCHEMA_NAME	NVARCHAR(256)	Schema name
TABLE_NAME	NVARCHAR(256)	Table name
TABLE_OID	BIGINT	Object ID of the table
INDEX_NAME	NVARCHAR(256)	Name of the geocode index
INDEX_OID	BIGINT	Object ID of the geocode index

Columns name	Data type	Description
FLUSH_AFTER_ROWS	INTEGER	Used to store change-tracking behavior of the geocode index
FLUSH_EVERY_MINUTES	INTEGER	Used to store change-tracking behavior of the geocode index

## Monitoring View M\_INDEXING\_QUEUES

View M\_INDEXING\_QUEUES lists queues of all types. Queues related to geocoding are registered with type GEOCODE.

### 5.2.6 Using SAP HANA Geocode

Set SAP HANA spatial to use Geocode from SAP HANA smart data quality.

You can use the SAP HANA smart data quality geocode feature, rather than using a third party provider such as TOMTOM or Nokia HERE. The SAP HANA smart data quality geocode is technology created at SAP and offers high performance without the need for a web service. You may need a subscription to access the directory reference data from SAP.

### Download Directories

The accuracy of the point represented by the latitude and longitude coordinates generated by the Geocode node is based on the completeness of the address being input and how well it matches to the geocode reference data, which is purchased separately. You will need to download the latest directories from the SAP Support Portal, and sign in with your S-User ID and password.

#### i Note

You will also need to enable the script server to use the directories.

For information about downloading and deploying the directories, see "Smart Data Quality Directories" in the *SAP HANA Smart Data Integration and SAP HANA Smart Data Quality Administration Guide*.

### Map Input Columns

At a minimum, you will need to map STREET, LOCALITY, POSTAL\_CODE and COUNTRY. The more columns that you map, the more likely you will have an assignment. In fact, if Geocode is unable to find a house-level assignment, then the output will be blank, even if Geocode was able to assign at the postcode or locality level. Here is a list of the supported input columns in alphabetical order.

- ADDRESS\_LINE
- CITY
- COUNTRY
- DISTRICT
- HOUSE\_NUMBER
- POSTAL\_CODE
- STATE

Depending on the input data, we recommend mapping the input data in the following ways.

At a minimum map these columns.

Table 100:

Input data columns	Mapped columns
STREET (if you have a single free-form address column)	STREET
CITY	LOCALITY
POSTCODE	POSTAL_CODE
COUNTRY	COUNTRY

If your data has the following columns, map them as shown here.

**i Note**

Do not map these columns to COUNTRY.

Table 101:

Input data columns	Mapped columns
REGION	REGION
SUBCITY	DISTRICT

If your data has two free-form address columns, map them as shown here.

For example, if you have the data "Main St" in the Address1 column, and the data "100" in the Address2 column, you would map Address1 to Street because there are many addresses on Main Street; then map Address2 to House\_Number because it contains the most specific information. In another example, if you have the data "100 Main St" in Address1 and the data "Suite 300" in the Address2 column, you would map Address1 to Street and Address2 to House\_Number.

Table 102:

Input data columns	Mapped columns
Address1 (the column with more broad information)	STREET
Address2 (the column with the more specific information)	HOUSE_NUMBER

If your data has three free-form address columns, map them from the broadest information to the most specific, similar to the example above.

Table 103:

Input data columns	Mapped columns
ADDRESS1 (broad information)	STREET
ADDRESS2 (more specific information)	HOUSE_NUMBER
ADDRESS3 (most specific information)	ADDRESS_LINE

**i Note**

Geographic location points are assigned based on the street name and house number. Extra information such as apartment or suite numbers, floor numbers or building names are not used, so those types of discrete columns do not need to be mapped.

## 5.2.7 Geocode Provider

The geocoding function that is integrated in SAP HANA uses the XS engine.

A geocode provider is used to translate textual address data into geographic locations. Geocode providers are supplied by delivery units and are written in JavaScript. To define the currently used provider, a configuration parameter is introduced.

### To use SAP HANA smart data quality geocode:

```
ALTER SYSTEM ALTER CONFIGURATION ('xsengine.ini','SYSTEM') set  
('geocoding','provider') = 'local' WITH RECONFIGURE
```

The maximum rows to process default setting is 1000, and the processing timeout value is 60 seconds. If you have multiple cores that you can use for processing, you can increase these values respectively with the following code.

```
ALTER SYSTEM ALTER CONFIGURATION ('xsengine.ini','SYSTEM')  
set('geocoding','max_rows_per_execution') = '100000' WITH RECONFIGURE
```

```
ALTER SYSTEM ALTER CONFIGURATION ('xsengine.ini','SYSTEM')  
set('geocoding','execution_timeout') = '1000' WITH RECONFIGURE
```

See the topic "Configuring the Operation Cache" in the *SAP HANA Smart Data Integration and SAP HANA Smart Data Quality Administration Guide* for details.

### To use a third party geocode provider:

```
ALTER SYSTEM ALTER CONFIGURATION ('xsengine.ini','SYSTEM') set  
('geocoding','provider') = '<content_path>' WITH RECONFIGURE
```

### Note

This parameter is not set by default in the case of newly imported and activated delivery units. It therefore has to be set manually. In general, <content\_path> refers to an xsjs script provided by the delivery unit, for example sap/hana/spatial/geocoding/<NameOfTheProvider>/geocodeindex.xsjs.

## 5.2.8 User Defined Geocode Provider

A geocode provider can be created to use third-party data that can be used by SAP HANA, provided that a number of conditions are met.

The basic structure of a geocode provider script has to contain the following JavaScript code:

```
function doGeocoding(input) { // [1]
    var geocodes = [], errors = [];
    for( var i = 0; i < input.entries.length; ++i ) {
        // do the geocoding for entry at index i [2]
        // if no error for entry at index i {
        //     geocodes.push( "A valid WKT POINT" ); // [3]
        // } else {
        //     geocodes.push( null ) // [4]
        //     errors.push( 400 )
        // }
    }
    return { "geocodes" : geocodes, "errors" : errors}; // [5]
} // [6]
```

The following conditions have to be met:

1. The name of the function is fixed and has to be `doGeocoding`. It takes exactly one input argument. If the input object is valid, it contains a single array attribute, named `<entries>`.
2. The `<input.entries>` array contains a sequence of addresses that should be processed. The number of entries that have to be processed can vary per call. Each entry object has the same attributes as (and no more than) the column names that have been provided during the `create_geocode_index` statement. Possible attributes are: `<address_line>`, `<city>`, `<country>`, `<county>`, `<district>`, `<house_number>`, `<postal_code>`, `<state>` and `<street>`. For instance: `<input.entries[i].house_number>`.
3. Addresses that could be successfully geocoded have to be inserted in the `<geocodes>` array in the same order as they are extracted from the `<input.entries>` array. The type has to be a valid string WKT representation of point geometry.
4. If an address cannot be geocoded, or a different error occurs that is only related to this single entry, this can only be represented as an empty string or `null` in the `<geocodes>` array. The `<errors>` array must be filled with a meaningful error code for this case. Filling the `<errors>` array for valid geocodes is prohibited and will cause an error. Only integers are allowed as error codes. Tracing calls are still possible and can be used to give more details about the current state.
5. The return object has to have two attributes. The array called `<geocodes>` and the array called `<errors>`. Other attributes are ignored. If a general error occurs during execution, returning `null` is a valid option.

6. The whole script is interpreted before the `doGeocoding` function is called. If other instructions are present, make sure that they do not interfere with the geocoding task. Querying the database is disabled, and will result in a privileges error exception.

**i Note**

Beside the xsjs-script with the `doGeocoding` function, there has to be at least one empty “`.xsapp`” file in the same folder, otherwise the script is unknown to the system.

## 5.2.9 Geocoding Errors and Messages

Geocoding errors and messages are handled as follows:

- If an address cannot be geocoded for any reason, this will be represented by the `POINT EMPTY` entity that is written into the `geocode` column for the failed row. This row will not be processed again until an address column in the `geocode` index is modified, the `geocode` column is updated to `null`, or the `RETRY_DOCUMENT_INDEXING` procedure is called.
- In case of any other errors related to the `geocode` provider or the queue processing of `geocode` indices, trace errors will be written, indicating the possible root cause. This can be recognized by the keyword `geocoding` available for the index server as well as the XS engine.

## 5.3 Additional Services for SAP HANA Spatial

Customers who have licensed SAP HANA (SAP HANA PLATFORM EDITION, SAP HANA ENTERPRISE EDITION, SAP HANA BASE EDITION with the SAP HANA SPATIAL OPTION, or derived licenses thereof) can use the voluntary map content and base map services (spatial map client, geo-content, spatial content viewer) provided by Nokia/HERE. These additional services and content are voluntarily provided by SAP and can be withdrawn, postponed, or suspended at any time. Customers who have licensed SAP HANA (SAP HANA PLATFORM EDITION, SAP HANA ENTERPRISE EDITION, SAP HANA BASE EDITION with the SAP HANA SPATIAL OPTION, or derived licenses thereof) can use this voluntary map content and base map services at no additional fee or license cost. Use of SAP HANA Spatial Engine is also governed by the Software Use Rights Agreement ([Browse SAP agreements](#) -> See agreements (your region) -> Access documents (Software Use Rights Agreements)). For more information about SAP HANA Spatial, see [SAP Note 2091935 - SAP HANA Spatial](#).

The following additional services for SAP HANA Spatial are provided on SAP Support Portal:

- Spatial map client, content viewer and geo-content
- Geospatial Metadata Installer

## Related Information

[SAP Note 1928222 - SAP HANA Spatial Delivery Units](#)

## 5.3.1 Spatial Map Client, Content Viewer and Geo-Content

The following additional services for SAP HANA Spatial are provided on SAP Support Portal:

- **Spatial map client**

The spatial map client provides an interface to use HERE maps.

- **Spatial content viewer**

The spatial content viewer is a simple web application that allows you to view geo-content in the predefined geo models (database schemas) called SAP\_SPATIAL and SAP\_SPATIAL\_POSTAL. You need this to view the geo-content on HERE maps. To use the spatial content viewer, you need the spatial map client.

- **Geo-Content**

The geo-content provides generalized administration boundaries (GAB) and postal codes (POC). The geo-content for GAB is provided in CSV files, whereas the geo-content for POC is provided in shapefiles.

 Note

Some countries (for example China) do not allow use of all administration hierarchies. For these countries, only a subset of the geo-content is provided.

### 5.3.1.1 Download Files from SAP Support Portal

In the SAP Software Download Center, you can access content for the additional services for SAP HANA Spatial.

1. Call a Web browser.
2. Open the SAP Software Download Center.
3. Choose *Software Downloads*.

4. Search for and download the following objects:
  - "Map Client"  
Download the HCOSPATIALMAPC00\_0-<nnn>.ZIP file and unpack it. Put the HCOSPATIALMAPC.tgz file on your client so that you can import it using the SAP HANA studio.
  - "Content Viewer"  
Download the HCOSPATIALCV00\_0-<nnn>.ZIP file and unpack it. Put the HCOSPATIALCV.tgz file on your client so that you can import it using the SAP HANA studio.
  - "GAB" (Generalized Administration Boundaries)  
Download the .SAR files that you need and unpack them. Put the CSV files on the SAP HANA host, where you can import them into the SAP HANA database.
  - "POC" (Postal Codes)  
Download the .SAR files that you need and unpack them. Put the shapefiles on the SAP HANA host, where you can import them into the SAP HANA database.  
In addition to the unpacked shapefiles for POC, you will also find product guides and release notes for the unpacked shapefiles.

## Related Information

[SAP Software Download Center](#)

## 5.3.1.2 Import the Map Client and the Spatial Content Viewer

Import the delivery units containing the map client and the content viewer.

1. Open the SAP HANA studio.
2. Choose *File* > *Import*.
3. Choose *SAP HANA Content* > *Delivery Unit*.
4. Select *Client*.
5. Import the delivery units:
  - o Browse for file HCOSPATIALMAPC.tgz, select it and choose *Finish*.
  - o Browse for file HCOSPATIALCV.tgz, select it and choose *Finish*.

For more information, see *Deploy a Delivery Unit Archive (\*.tgz)* in the *SAP HANA Master Guide*.

### Note

If you are using the `hdbalm` command to import the map client and the content viewer, you might encounter problems with the versions of the archives. An update might be rejected because `hdbalm` recognizes an older version in the archives you want to import. In this case you can enforce the update by using the `hdbalm` option `ALLOW_DU_DOWNGRADE`. The reason for this behavior is a change in the structuring of the version numbers.

See also the attachment to [SAP Note 2165826 - SAP HANA Platform SPS 10 Release Note](#).

## Related Information

[SAP HANA Master Guide](#)

[SAP Note 2165826 - SAP HANA Platform SPS 10 Release Note](#)

### 5.3.1.3 Create Database Schemas and Tables

For the Generalized Administration Boundaries (GAB) and the Postal Codes (POC), you need to create different database schemas and tables.

#### 5.3.1.3.1 Create Database Schema and Table (GAB)

The database schema for GAB is SAP\_SPATIAL.

**i Note**

The database schema can also be created by the content viewer. For more information, see *Use Content Viewer (GAB)*.

The database schema contains the following tables:

- AREA
  - AREA\_DESCRIPTION
  - HIERARCHY
  - HIERARCHY\_DESCRIPTION
  - SHAPE
  - SHAPE\_DISPLAY
1. Call the SAP HANA studio.
  2. Open an SQL console.
  3. Create the database schema and the tables with the following SQL commands:

```
create schema SAP_SPATIAL;
create column table SAP_SPATIAL.AREA (
    area_id bigint,
    spatial_hierarchy_id VARCHAR(15),
    package VARCHAR(3),
    name NVARCHAR(256),
    level TINYINT,
    parent_area_id bigint,
    PRIMARY KEY(area_id, spatial_hierarchy_id)
);
create column table SAP_SPATIAL.AREA_DESCRIPTION (
    area_id bigint,
    langu VARCHAR(3),
    description NVARCHAR(256),
    PRIMARY KEY(area_id, langu)
);
create column table SAP_SPATIAL.HIERARCHY (
    spatial_hierarchy_id VARCHAR(15),
    name NVARCHAR(256),
    PRIMARY KEY(spatial_hierarchy_id)
);
create column table SAP_SPATIAL.HIERARCHY_DESCRIPTION (
    spatial_hierarchy_id VARCHAR(15),
    langu VARCHAR(3),
    description NVARCHAR(256),
    PRIMARY KEY(spatial_hierarchy_id, langu)
);
create column table SAP_SPATIAL.SHAPE (
```

```

area_id bigint,
detail_level tinyint,
external_id VARCHAR(20),
shape ST_GEOMETRY(1000004326),
PRIMARY KEY(area_id, detail_level)
);
create column table SAP_SPATIAL.SHAPE_DISPLAY (
use_case_id VARCHAR(15),
zoom_level tinyint,
area_id bigint,
display_detail_level tinyint,
PRIMARY KEY(use_case_id, zoom_level, area_id)
);

```

### **i Note**

The field LEVEL in the table SAP\_SPATIAL.AREA has the same meaning in every country, but not every country has entries on every level:

- 1 - country
- 2 - state
- 3 - county
- 4 - city
- 5 - district

## Related Information

[Use the Content Viewer \(GAB\) \[page 213\]](#)

### 5.3.1.3.2 Create Database Schema and Table (POC)

The database schema for POC is SAP\_SPATIAL\_POSTAL.

The database schema contains the GENERALIZED table.

1. Call the SAP HANA studio.
2. Open an SQL console.
3. Create the database schema and the table with the following SQL commands:

```

CREATE SCHEMA SAP_SPATIAL_POSTAL;
CREATE COLUMN TABLE "SAP_SPATIAL_POSTAL"."GENERALIZED" (
"POSTCODE" VARCHAR(10),
"ISO_CTRY" VARCHAR(3),
"ADMIN1" VARCHAR(80),
"ADMIN2" VARCHAR(80),
"ADMIN3" VARCHAR(80),
"ADMIN4" VARCHAR(80),
"ADMIN5" VARCHAR(80),
"AREA" DOUBLE CS_DOUBLE,
"SHAPE" ST_Geometry(0) CS_Geometry) UNLOAD PRIORITY 5 AUTO MERGE;

```

### Note

Some of the shapefiles contain the AREA column, while others do not. This difference has to be taken into account during the import into the SAP\_SPATIAL\_POSTAL.GENERALIZED table.

## 5.3.1.4 Import Geo-Content

Two different import methods are used depending on whether the geo-content is for Generalized Administration Boundaries (GAB) or for Postal Codes (POC).

### 5.3.1.4.1 Import Geo-Content (GAB)

The geo-content for Generalized Administration Boundaries (GAB) needs to be imported into the database tables.

The geo-content for GAB is contained in CSV files. To import the CSV files for GAB, perform the following steps:

1. Put the CSV files on the host.
2. Perform IMPORT commands like this:

```
IMPORT FROM CSV FILE '<path and filename>' INTO SAP_SPATIAL.<TABLE>
WITH THREADS 10
RECORD DELIMITED BY '\n'
FIELD DELIMITED BY ';'
OPTIONALLY ENCLOSED BY '"';
ERROR LOG '<filepath and filename>';
```

Perform the import for all tables belonging to database schema SAP\_SPATIAL and for all geo-content that you want to import. This is an example of importing for the table SAP\_SPATIAL.AREA for EUROPE:

```
IMPORT FROM CSV FILE '/usr/sap/ABC/HDB50/work/GAB/EU_Q214/AREA.csv' INTO
SAP_SPATIAL.AREA
WITH THREADS 10
RECORD DELIMITED BY '\n'
FIELD DELIMITED BY ';'
OPTIONALLY ENCLOSED
BY '"';
ERROR LOG '/usr/sap/ABC/HDB50/work/GAB/EU_Q214/AREA.err';
```

### Caution

Choose the number of threads in accordance with your hardware.

### Note

By default, the files have to be in the work folder of the SAP HANA system: /usr/sap/<SID>/HDB<instance\_number>/work/. If configuration parameter enable\_csv\_import\_path\_filter

(indexserver.ini, nameserver.ini) is set to `false`, the files can be located anywhere on the SAP HANA host.

### 5.3.1.4.2 Import Geo-Content (POC)

The geo-content for Postal Codes (POC) needs to be imported into the database table.

The geo-content for POC is contained in various kinds of shapefiles. Check the geo-content and choose the shapefiles you need. The following shapefiles are offered for Japan for example:

- JPN\_2012Q1\_PCB\_PLY\_GEN
- JPN\_2012Q1\_PCB\_PLY\_UNGEN
- JPN\_2012Q1\_PCB PTS

#### i Note

In addition to the unpacked shapefiles for POC, you will also find product guides and release notes for the unpacked shapefiles.

The following section describes how to import the shapefiles with the generalized content:

`<country>_<year>_<quarter>_PCB_PLY_GEN` (for example `BEL_2013Q1_PCB_PLY_GEN`). The import is performed in the same way for the other types of shapefiles.

To import the shapefiles for POC, perform the following steps:

1. Put the shapefiles on the host.
2. Perform this step for all shapefiles you need.

Perform IMPORT commands like this:

```
IMPORT "SAP_SPATIAL_POSTAL"."<country>_GEN" AS SHAPEFILE FROM '<path and filename>';
```

This is an example for importing shapefile `BEL_2013Q1_PCB_PLY_GEN`:

```
IMPORT "SAP_SPATIAL_POSTAL"."BEL_GEN" AS SHAPEFILE FROM  
'/usr/sap/ABC/HDB507/work/POC/BEL_2013Q1_PCB/shp/BEL_2013Q1_PCB_PLY_GEN';
```

#### i Note

By default, the files have to be in the work folder of the SAP HANA system: `/usr/sap/<SID>/HDB<instance_number>/work/`. If configuration parameter `enable_csv_import_path_filter` (indexserver.ini, nameserver.ini) is set to `false`, the files can be located anywhere on the SAP HANA host.

3. Insert the data from the tables you just imported into table `SAP_SPATIAL_POSTAL.GENERALIZED`.  
This is an example for inserting two tables that do not have the AREA column:

```
insert into "SAP_SPATIAL_POSTAL"."GENERALIZED" (postcode, iso_ctry, admin1,  
admin2, admin3, admin4, admin5, shape) (  
    select postcode, iso_ctry, admin1, admin2, admin3, admin4, admin5, shape  
    from "SAP_SPATIAL_POSTAL"."BEL_GEN"  
    union all
```

```
select postcode, iso_ctry, admin1, admin2, admin3, admin4, admin5, shape  
from "SAP_SPATIAL_POSTAL"."NDL_GEN"  
);
```

For tables that have the AREA column, the insert statement is:

```
insert into "SAP_SPATIAL_POSTAL"."GENERALIZED" (postcode, iso_ctry, admin1,  
admin2, admin3, admin4, admin5, area, shape) (  
    select postcode, iso_ctry, admin1, admin2, admin3, admin4, admin5, area,  
    shape from "SAP_SPATIAL_POSTAL"."CAN_GEN"  
);
```

### 5.3.1.5 Create a User to View Geo-Content

To view the geo-content, you need a dedicated database user.

1. Call the SAP HANA studio.
2. In the Systems view, log on to the SAP HANA system.
3. In the Systems view, open the context menu for ► **Security** ► **Users** ▾.
4. Choose *New User* and make the required entries.
5. Grant the user Object Privileges for the following database schemas:
  - SAP\_SPATIAL
  - SAP\_SPATIAL\_POSTAL
6. Choose *Deploy* to save the user.

### 5.3.1.6 Assign the User for the Content Viewer

Assign the user that you created to view the geo-content.

1. Open a Web browser.
2. Start the SAP HANA XS Administration Tool.

The SAP HANA XS Administration Tool is available on the SAP HANA XS Web server at the following URL:

[http://<host>:80<instance\\_number>/sap/hana/xs/admin/](http://<host>:80<instance_number>/sap/hana/xs/admin/) (for example <http://myhost:8050/sap/hana/xs/admin/>).

**i Note**

You need XS Admin rights to assign the user.

3. Choose ► **XS Administration Tools** ▾.
4. Choose ► **XS Artifact Administration** ▾.
5. Choose ► **Application Objects** ► **Packages** ► **sap** ► **hana** ► **spatial** ► **contentViewer** ▾.
6. Select ► **query.xssqlcc** ▾.
7. Choose ► **Edit** ▾.
8. Enter user name and password of the user that you created to view the geo-content.
9. Choose ► **Save** ▾.

## Related Information

[Create a User to View Geo-Content \[page 212\]](#)

### 5.3.1.7 Use the Spatial Content Viewer

Different content viewers are provided for the Generalized Administration Boundaries (GAB) and for the Postal Codes (POC).

#### 5.3.1.7.1 Use the Content Viewer (GAB)

Use the content viewer to view GAB geo content.

##### Note

If you have not created the database schema SAP\_SPATIAL and imported the data yet, the viewer displays the message "SAP\_SPATIAL not found. Please create it **now!**". If you choose "**now!**", the content viewer creates the database schema and the tables. You then have to import the data, as described in *Import Geo-Content (GAB)*. The object privileges are automatically granted to the user you have created to view the geo content.

1. Open a Web browser.

2. Start the *Geo Content Viewer*.

The *Geo Content viewer* tool is available on the SAP HANA XS Web server at the following URL: `http://<host>:80<instance_number>/sap/hana/spatial/contentViewer/index.html`, (for example `http://myhost:8050/sap/hana/spatial/contentViewer/index.html`).

3. Choose .

You can make the following selections:

- Country
- Level
- Area Name
- Detail Level
- Zoom Level
- Jump to polygon (checked/not checked)
- Clear map (checked/not checked)

##### Note

If no database schema SAP\_SPATIAL has been created, or if the database tables do not contain any data, the viewer displays an error message.

4. Choose *Submit* to perform your selection.

## 5.3.1.7.2 Use the Content Viewer (POC)

Use the content viewer to view GAB geo-content.

1. Open a Web browser.
2. Start the *Content Viewer*.

The *Content Viewer* tool is available on the SAP HANA XS Web server at the following URL: `http://<host>:80<instance_number>/sap/hana/spatial/contentViewer/index.html`, (for example `http://myhost:8050/sap/hana/spatial/contentViewer/index.html`).

3. Choose  *Postal Content Viewer*.

You can make the following selections:

- Country
- Level
- Area Name
- Zoom Level
- Jump to polygon (checked/not checked)
- Clear map (checked/not checked)

 **Note**

If no database schema SAP\_SPATIAL\_POSTAL has been created, or if the database table does not contain any data, the viewer displays an error message.

4. Choose *Submit* to perform your selection.

## 5.3.2 Geospatial Metadata Installer

You can use the Geospatial Metadata Installer to install standard EPSG spatial reference system (SRS) and unit of measure (UOM) definitions.

### 5.3.2.1 Download File from SAP Support Portal

In the SAP Software Download Center, you can access content for the additional services for SAP HANA Spatial.

1. Call a Web browser.
2. Open the SAP Software Download Center.
3. Choose *Software Downloads*.

4. Search for and download the following objects:
  - o "SPATIAL METADATA"
 

Download the HCOSPATIALMI00\_0-<nnn>.ZIP file and unpack it. Put the HCOSPATIALMI.tgz file on your client so that you can import it using the SAP HANA studio.

## Related Information

[SAP Software Download Center](#)

### 5.3.2.2 Import the Geospatial Metadata Installer

Import the delivery unit.

1. Open the SAP HANA studio.
2. Choose **File > Import**.
3. Choose **SAP HANA Content > Delivery Unit**.
4. Select **Client**.
5. Import the delivery units:
  - o Browse for file HCOSPATIALMI.tgz, select it and choose *Finish*.

### 5.3.2.3 Create Database Users

Create database users to use the Geospatial Metadata Installer.

1. Call the SAP HANA studio.
2. In the Systems view, log on to the SAP HANA system.
3. In the Systems view, open the context menu for ► *Security* ► *Users* ▾.
4. Choose *New User* and make the required entries.  
Create the following users:
  - One user (for example RESTRICTED\_USER) to use the Geospatial Metadata Installer
  - One user (for example CONNECTOR) to establish the required SQLCC connection and modify the defined database object
5. Grant the users the following privileges:
  - RESTRICTED\_USER
    - APPLICATION PRIVILEGE: "sap.hana.spatial.metadataInstaller::Execute"
  - CONNECTOR
    - SYSTEM PRIVILEGE: CATALOG READ
    - OBJECT PRIVILEGE: ST\_SPATIAL\_REFERENCE\_SYSTEMS (DB object type "view") with privilege "SELECT"
    - OBJECT PRIVILEGE: ST\_UNITS\_OF\_MEASURE (DB object type "view") with privilege "SELECT"
    - OBJECT PRIVILEGE: ST\_GEOOMETRY\_COLUMNS (DB object type "view") with privilege "SELECT"
    - OBJECT PRIVILEGE: METADATAINSTALLER (DB object type "schema") with all privileges
6. Choose *Deploy* for each user (RESTRICTED\_USER and CONNECTOR) in order to save the user.

### 5.3.2.4 Assign User for the Geospatial Metadata Installer

Assign the user to view the Geospatial Metadata Installer.

1. Open a web browser.
2. Start the *SAP HANA XS Administration Tool*.  
The *SAP HANA XS Administration Tool* is available on the SAP HANA XS Web server at the following URL:  
`http://<host>:<instance_number>/sap/hana/xs/admin/` (for example `http://myhost:8050/sap/hana/xs/admin/`).

#### i Note

You need XS Admin rights to assign the user.

3. Choose ► *XS Administration Tools* ▾.
4. Choose ► *XS Artifact Administration* ▾.
5. Choose ► *Application Objects* ► *Packages* ► *sap* ► *hana* ► *spatial* ► *metadataInstaller* ► *xsjs* ▾.
6. Select ► *query.xssqlcc* ▾.
7. Choose ► *Edit* ▾.
8. Enter the user name and password of the user (for example CONNECTOR) that you created to establish the SQLCC connection.

9. Choose  Save.

### 5.3.2.5 Use the Geospatial Metadata Installer

Use the Geospatial Metadata Installer.

1. Open a web browser.
2. Start the *Geospatial Metadata Installer*.  
The *Geospatial Metadata Installer* tool is available on the SAP HANA XS Web server at the following URL:  
`http://<host>:80<instance_number>/sap/hana/spatial/metadataInstaller/index.html`,  
(for example `http://myhost:8050/sap/hana/spatial/metadataInstaller/index.html`).
3. Enter the user credentials (for example RESTRICTED\_USER) to log on to the *Geospatial Metadata Installer*.
4. You can use the following functions:
  - View Current State of Spatial Metadata  
This function shows the following information:
    - How many Spatial Reference Systems (SRS) and Spatial Units of Measure (UOM) currently are stored in the database.
    - How many and which SRS and UOM currently are active, i.e. in use in the database. These cannot be updated.
    - How often these SRS and UOM are used at present.
  - Start Update Immediately  
This function installs standard EPSG Spatial reference system (SRS) and unit of measure (UOM) definitions in your SAP HANA system.

# Important Disclaimers and Legal Information

## Coding Samples

Any software coding and/or code lines / strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended to better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, unless damages were caused by SAP intentionally or by SAP's gross negligence.

## Accessibility

The information contained in the SAP documentation represents SAP's current view of accessibility criteria as of the date of publication; it is in no way intended to be a binding guideline on how to ensure accessibility of software products. SAP in particular disclaims any liability in relation to this document. This disclaimer, however, does not apply in cases of willful misconduct or gross negligence of SAP. Furthermore, this document does not result in any direct or indirect contractual obligations of SAP.

## Gender-Neutral Language

As far as possible, SAP documentation is gender neutral. Depending on the context, the reader is addressed directly with "you", or a gender-neutral noun (such as "sales person" or "working days") is used. If when referring to members of both sexes, however, the third-person singular cannot be avoided or a gender-neutral noun does not exist, SAP reserves the right to use the masculine form of the noun and pronoun. This is to ensure that the documentation remains comprehensible.

## Internet Hyperlinks

The SAP documentation may contain hyperlinks to the Internet. These hyperlinks are intended to serve as a hint about where to find related information. SAP does not warrant the availability and correctness of this related information or the ability of this information to serve a particular purpose. SAP shall not be liable for any damages caused by the use of related information unless damages have been caused by SAP's gross negligence or willful misconduct. All links are categorized for transparency (see: <http://help.sap.com/disclaimer>).





**go.sap.com/registration/  
contact.html**

© 2016 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see <http://www.sap.com/corporate-en/legal/copyright/index.epx> for additional trademark information and notices.