Administration: Spatial Data

SAP Sybase IQ 16.0 SP04

DOCUMENT ID: DC01964-01-1604-01
LAST REVISED: May 2014

Contents

Restrictions and Limitations

Before working with spatial data, spatial references systems, and spatial units of measure in SAP® Sybase® IQ, familiarize yourself with 3D method restrictions and implications for performance and referential integrity.

2006 ISO Standard

3D methods are not supported, although you can store Z and M dimensions. The 2006 ISO Standard supports 2D spatial methods (X and Y dimensions) only.

Spatial Data Must be Stored in IQ Catalog Store Tables

Spatial data, spatial references systems, and spatial units of measure can be used only in the catalog store. The IQ main store cannot interpret spatial data, or store spatial data. You can query spatial data in the catalog store and join with an IQ main store table, but the join must be on a non-spatial column.

For example, suppose you want to associate an ST_Point with each of your customers, stored in an IQ main store table. Because IQ main store tables cannot store ST_Points, you must create a separate IQ catalog store table to hold points:

IQ main store table	Customer(CustID, CustName, …)
IQ catalog store table	CustPoints(CustID, Point)

Consider a scenario where you have an ST_Polygon P, and you want a query to find all customers within P. Assume P is a connection variable that has been populated either from a constant or by a previous query of an IQ catalog store table.

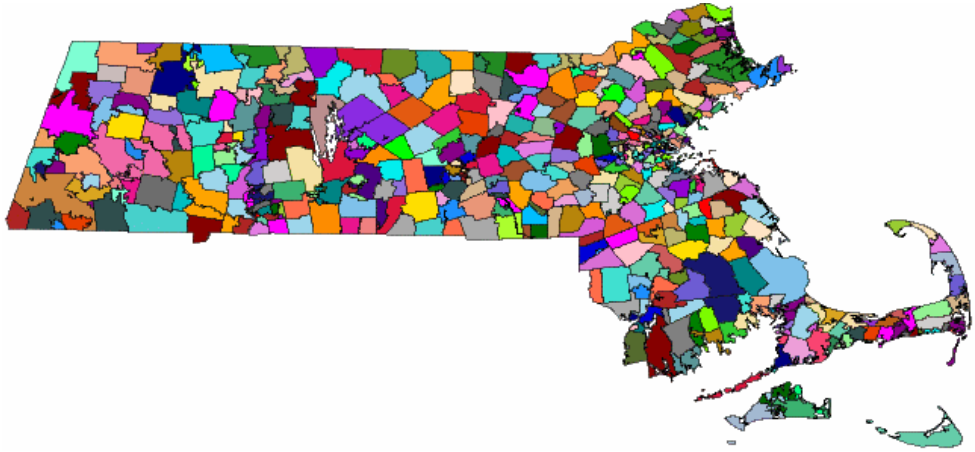
```
Select C.*
From Customer C, CustPoints CP
Where C.CustID = CP.CustID
And CP.Point.ST_Within( P ) = 1
```

Since this query streams the Customer table from the IQ main store to the IQ catalog store, performance is impacted. CIS functional compensation performance considerations apply.

Referential integrity constraints are not maintained across the IQ main store / IQ catalog store bridge. Ensure that row inserts/deletes in one table are reflected in the joined table. Every CustPoints row, for example, must correspond to a Customer row. If you delete Customer rows, you must delete the corresponding CustPoints rows.

Spatial data

Spatial data is data that describes the position, shape, and orientation of objects in a defined space. Spatial data in SAP Sybase IQ is represented as 2D geometries in the form of points, curves (line strings and strings of circular arcs), 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.

Note: Spatial data support for 32-bit Windows and 32-bit Linux requires a CPU that supports SSE2 instructions. This support is available with Intel Pentium 4 or later (released in 2001) and AMD Opteron or later (released in 2003).

Example of how spatial data might be used

Spatial data support in SAP Sybase IQ 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:

```
CREATE TABLE Locations(
   ID INT,
   ManagerName CHAR(16),
   StoreName CHAR(16),
   Address ST_Point,
   DeliveryArea ST_Polygon )
```

The spatial data type ST_Point in the example represents a single point, and ST_Polygon represents an arbitrary polygon. With this schema, the application could show all company locations on a map, or find out if a company delivers to a particular address using a query similar to the following:

```
CREATE VARIABLE @pt ST_Point;
SET @pt = ST_Geometry::ST_GeomFromText( 'POINT(1 1)' );

SELECT * FROM Locations
WHERE DeliveryArea.ST_Contains( @pt ) = 1
```

SAP Sybase IQ 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. SAP Sybase IQ also provides a set of SQL spatial functions designed for compatibility with other products.

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:

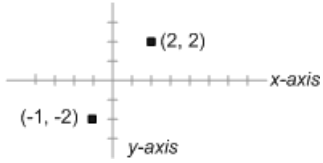
- Units of measure of the underlying coordinate system (degrees, meters, and so on)
- Maximum and minimum coordinates (also referred to as the bounds)
- Default linear unit of measure
- Whether the data is planar or spheroid data
- Projection information for transforming the data to other SRSs

Every spatial reference system has an identifier called a **Spatial Reference Identifier (SRID)**. When SAP Sybase IQ 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. In an SAP Sybase IQ database, each SRID must be unique.

By default, SAP Sybase IQ 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.

- **WGS 84 - SRID 4326 –** The WGS 84 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 ±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 SRID 4326 and 1000004326, 1 degree longitude is approximately 111 km. At 80 degrees north, 1 degree of longitude is approximately 19 km in SRID 4326, but SRID 1000004326 treats 1 degree of longitude as approximately 111 km at *all* latitudes. The amount of distortion of lengths in the SRID 1000004326 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, SRID 1000004326 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 DEGREE, and it is a flat-Earth type of spatial reference system.
- **sa_planar_unbounded - SRID 2,147,483,646 –** For internal use only.
- **sa_octahedral_gnomonic - SRID 2,147,483,647 –** For internal use only.

Since you can define a spatial reference system however you want and can assign any SRID number, the spatial reference system definition (projection, coordinate system, and so on) must accompany the data as it moves between databases or is converted to other SRSs. For example, when you unload spatial data to WKT, the definition for the spatial reference system is included at the beginning of the file.

Installing additional spatial reference systems using the sa_install_feature system procedure

SAP Sybase IQ also provides thousands of predefined SRSs for use. However, these SRSs are not installed in the database by default when you create a new database. You use the sa_install_feature system procedure to add them.

You can find descriptions of these additional spatial reference systems at *spatialreference.org* and *www.epsg-registry.org/*.

Determining the list of spatial reference systems currently in the database
Spatial reference system information is stored in the ISYSSPATIALREFERENCESYSTEM system table. The SRIDs for the SRSs are used as primary key values in this table. The database server uses SRID values to look up the configuration information for a spatial reference system so that it can interpret the otherwise abstract spatial coordinates as real positions on the Earth.

You can find the list of spatial reference systems by querying the ST_SPATIAL_REFERENCE_SYSTEMS consolidated view. Each row in this view defines a spatial reference system.

Compatibility with popular mapping applications
Some popular web mapping and visualization applications such as Google Earth, Bing Maps, and ArcGIS Online, use a spatial reference system with a Mercator projection that is based on a spherical model of the Earth. This spherical model ignores the flattening at the Earth's poles and can lead to errors of up to 800m in position and up to 0.7 percent in scale, but it also allows applications to perform projections more efficiently.

In the past, commercial applications assigned SRID 900913 to this spatial reference system. However, EPSG has since released this projection as SRID 3857. For compatibility with applications requiring 900913, you can do the following:

1. Use the sa_install_feature system procedure to install all of the spatial reference systems provided by SAP Sybase IQ (including SRID 3857).
2. Perform `dbunload -n` to get the 3857 SRID definition (the dbunload utility is not provided with SAP Sybase IQ).

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 SAP Sybase IQ database:

- **meter –** A linear unit of measure. Also known as International metre. SI standard unit. Defined by ISO 1000.

- **metre –** A linear unit of measure. An alias for meter. SI standard unit. Defined by ISO 1000.
- **radian –** An angular unit of measure. SI standard unit. Defined by ISO 1000:1992.
- **degree –** 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.

Installing additional predefined units of measure

The sa_install_feature system procedure adds additional predefined units of measure not installed by default in a new database.

Prerequisites

None.

Task

Execute the following statement to install all of the predefined units of measure:

```
CALL sa_install_feature('st_geometry_predefined_uom');
```

All additional units of measure are installed.

Next

You can create a spatial reference system that uses the unit of measure.

You can find descriptions of these additional units of measure at *www.epsg-registry.org/*. On the web page, type the name of the unit of measure in the `Name` field, pick `Unit of Measure (UOM)` from the `Type` field, and then click `Search`.

SAP Sybase IQ support for spatial data

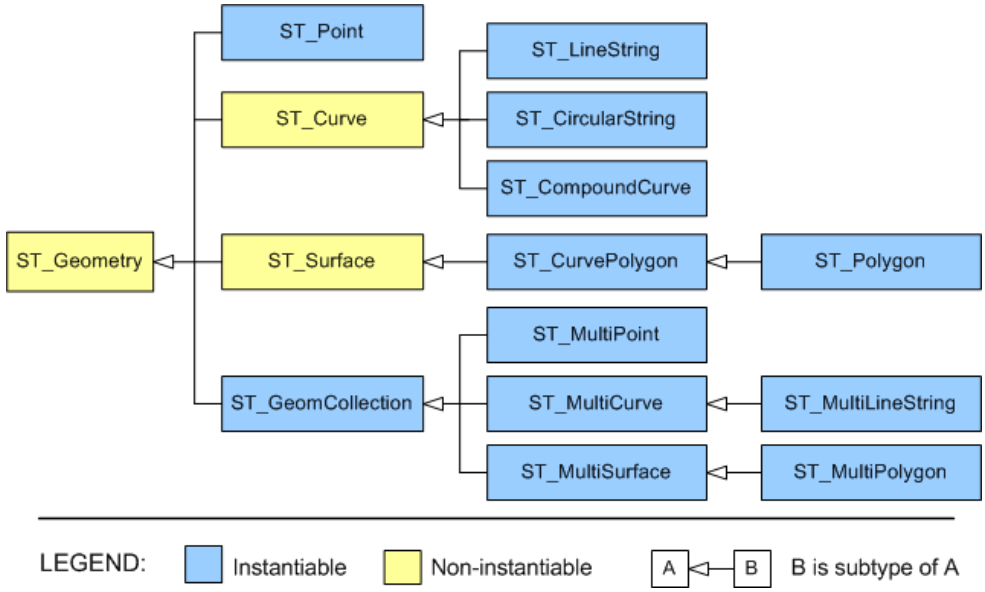
The following sections describe the SAP Sybase IQ support for spatial data.

Supported spatial data types and their hierarchy

SAP Sybase IQ 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_GeomCollection can also store the ST_MultiPoint, ST_MultiSurface, ST_MultiCurve, ST_MultiPolygon, and ST_MultiLineString values.

The following diagram illustrates the hierarchy of the ST_Geometry data types and their subtypes:



The types on the left are supertypes (or base types) for the subtypes (or derived types) on the right.

Descriptions of supported spatial data types

SAP Sybase IQ supports the following spatial data types:

- **Points –** 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.
- **Linestrings –** 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. **Simple** means a linestring that does not cross itself. **Closed** 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.
- **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.

- **Circularstrings –** A circularstring is a connected sequence of circular arc segments; much like a linestring with circular arcs between points.
- **Compound curves –** A compound curve is a connected sequence of circularstrings or linestrings.
- **Curve polygons –** A curve polygon is a more general polygon that may have circular arc boundary segments.
- **Geometries –** 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.
- **Geometry collections –** A geometry collection is a collection of one or more geometries (such as points, lines, polygons, and so on).
- **Multipoints –** A multipoint is a collection of individual points.

  In GIS data, multipoints are typically used to represent a set of locations.
- **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.
- **Multilinestring –** A multilinestring is a collection of linestrings.

  In GIS data, multilinestrings are often used to represent geographic features like rivers or a highway network.
- **Multisurfaces –** A multisurface is a collection of curve polygons.

Object-oriented properties of spatial data types

- A subtype (or derived type) is more specific than its supertype (or base type). For example, ST_LineString is a more specific type of ST_Curve.
- A subtype inherits all methods from all supertypes. For example, ST_Polygon values can call methods from the ST_Geometry, ST_Surface and ST_CurvePolygon supertypes.
- 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 a values of any subtype. For example, a column of type ST_Geometry(SRID=4326) 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()`.

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:

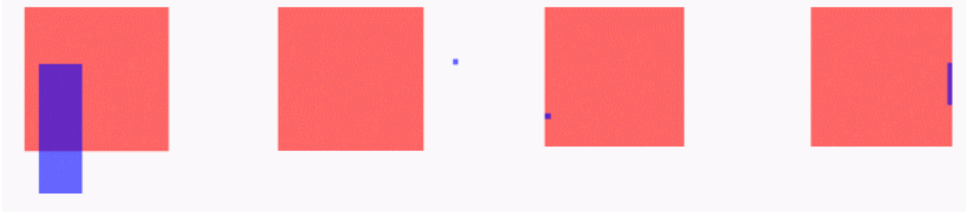
```
SELECT * FROM SpatialShapes WHERE geometry.ST_IsEmpty() = 0;
```

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.

Intuitiveness of spatial predicates

Sometimes the outcome of a predicate is not intuitive, so you should 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.

Compliance with spatial standards

SAP Sybase IQ spatial complies with the following standards:

- **International Organization for Standardization (ISO) –** SAP Sybase IQ geometries conform to the ISO standards for defining spatial user-types, routines, schemas, and for processing spatial data. SAP Sybase IQ conforms to the specific recommendations made by the International Standard ISO/IEC 13249-3:2006. See *http://www.iso.org/iso/catalogue_detail.htm?csnumber=38651*.
- **Open Geospatial Consortium (OGC) Geometry Model –** SAP Sybase IQ geometries conform to the OGC OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 2: SQL option version 1.2.0 (OGC 06-104r3). See *http://www.opengeospatial.org/standards/sfs*.

  SAP Sybase IQ uses the standards recommended by the OGC to ensure that spatial information can be shared between different vendors and applications.

  To ensure compatibility with SAP Sybase IQ spatial geometries, it is recommended that you adhere to the standards specified by the OGC.
- **SQL Multimedia (SQL/MM) –** SAP Sybase IQ follows the SQL/MM standard, and uses the prefix `ST_` for all method and function names.

  SQL/MM is an international standard that defines how to store, retrieve, and process spatial data using SQL. Spatial data type hierarchies such as ST_Geometry are one of the methods used to retrieve spatial data. The ST_Geometry hierarchy includes a number of subtypes such as ST_Point, ST_Curve, and ST_Polygon. With the SQL/MM standard, every spatial value included in a query must be defined in the same spatial reference system.

Special notes on support and compliance

This section describes any special notes about SAP Sybase IQ support of spatial data including unsupported features and notable behavioral differences with other database products.

- **Geographies and geometries –** Some vendors distinguish spatial objects by whether they are **geographies** (pertaining to objects on a round-Earth) or **geometries** (objects on a plane or a flat-Earth). In SAP Sybase IQ, all spatial objects are considered to be geometries, and the object's SRID indicates whether it is being operated on in a round-Earth or flat-Earth (planar) spatial reference system.
- **Unsupported methods –**

  ST_Buffer method
  ST_LocateAlong method
  ST_LocateBetween method
  ST_Segmentize method

ST_Simplify method
ST_Distance_Spheroid method
ST_Length_Spheroid method

Supported import and export formats for spatial data

The following table lists the data and file formats supported by SAP Sybase IQ for importing and exporting spatial data:

Data format	Import	Export	Description
Well Known Text (WKT)	Yes	Yes	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 *www.opengeospatial.org/standards/sfa*. Here is an example of how a point might be represented in WKT: `'POINT(1 1)'`

Data format	Import	Export	Description
Well Known Binary (WKB)	Yes	Yes	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 *www.opengeospatial.org/standards/sfa*. Here is an example of how a point might be represented in WKB: `'01010000000000000000F03F0000000000000F03F'`
Extended Well Known Text (EWKT)	Yes	Yes	WKT format, but with SRID information embedded. This format is maintained as part of PostGIS, the spatial database extension for PostgreSQL. See *postgis.refractions.net/*. Here is an example of how a point might be represented in EWKT: `'srid=101;POINT(1 1)'`

Data format	Import	Export	Description
Extended Well Known Binary (EWKB)	Yes	Yes	WKB format, but with SRID information embedded. This format is maintained as part of PostGIS, the spatial database extension for PostgreSQL. See *postgis.refractions.net/*. Here is an example of how a point might be represented in EWKB: `'01010000020040000000000000000F03F0000000000000F03F'`
Geographic Markup Language (GML)	No	Yes	XML grammar used to represent geographic spatial data. This standard is maintained by the Open Geospatial Consortium (OGC), and is intended for the exchange of geographic data over the internet. See *www.opengeospatial.org/standards/gml*. Here is an example of how a point might be represented in GML: `<gml:Point> <gml:coordinates>1,1</gml:coordinates> </gml:Point>`

Data format	Import	Export	Description
KML	No	Yes	Formerly Google Keyhole Markup Language, this XML grammar is used to represent geographic data including visualization and navigation aids and the ability to annotate maps and images. Google proposed this standard to the OGC. The OGC accepted it as an open standard which it now calls KML. See *www.opengeospatial.org/standards/kml*. Here is an example of how a point might be represented in KML: `<Point> <coordinates>1,0</coordinates> </Point>`
ESRI shapefiles	Yes	No	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).

Data format	Import	Export	Description
GeoJSON	No	Yes	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. GeoJSON is a subset of the JSON standard and is used to encode geographic information. SAP Sybase IQ supports the GeoJSON standard and provides the ST_AsGeoJSON method for converting SQL output to the GeoJSON format. Here is an example of how a point might be represented in GeoJSON: `{"x" : 1, "y" : 1, "spatialReference" : {"wkid" : 4326}}` For more information about the GeoJSON specification, see *geojson.org/geojson-spec.html*.

Data format	Import	Export	Description
Scalable Vector Graphic (SVG) files	No	Yes	XML-based format used to represent two-dimensional geometries. The SVG format is maintained by the World Wide Web Consortium (W3C). See *www.w3.org/Graphics/SVG/*. Here is an example of how a point might be represented in SVG: `<rect width="1" height="1" fill="deepskyblue" stroke="black" strokewidth="1" x="1" y="-1"/>`

Support for ESRI shapefiles

SAP Sybase IQ supports the Environmental System Research Institute, Inc. (ESRI) shapefile format. 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. SAP Sybase IQ 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.

The simplest ways to load a shapefile into a table are with the Interactive SQL **Import Wizard**, or the st_geometry_load_shapefile system procedure. Both of these tools create a table with appropriate columns and load the data from the shapefile.

You can also load shapefiles using the LOAD TABLE and INPUT statements, but you must already have created the table with the appropriate columns before performing the load operation.

To find the columns needed when loading data using the LOAD TABLE or INPUT statements, you can use the sa_describe_shapefile system procedure.

For more information about ESRI shapefiles, see *http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf*.

Recommended reading on spatial topics

For a good primer on the different approaches that are used to map and measure the earth's surface (geodesy), and the major concepts surrounding coordinate (or spatial) reference systems, go to *www.epsg.org/guides/index.html* and select Geodetic Awareness.

OGC OpenGIS Implementation Specification for Geographic information - Simple feature access: *www.opengeospatial.org/standards/sfs*

International Standard ISO/IEC 13249-3:2006: *www.iso.org/iso/catalogue_detail.htm?csnumber=38651*

Scalable Vector Graphics (SVG) 1.1 Specification: *www.w3.org/Graphics/SVG/*

Geographic Markup Language (GML) specification: *www.opengeospatial.org/standards/gml*

KML specification: *www.opengeospatial.org/standards/kml*

JavaScript Object Notation (JSON): *json.org*

GeoJSON specification: *geojson.org/geojson-spec.html*

Creating a spatial column (SQL)

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

Prerequisites

- The table must be in a catalog store table created using the IN SYSTEM clause.
- You must be the owner of the table, or have ALTER privilege on the table, or have the ALTER ANY TABLE or ALTER ANY OBJECT system privilege.
- Table must be in the catalog store table created with the "IN SYSTEM" clause.

Task

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

A spatial column is added to the existing table.

Next

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

Indexes on spatial columns

When creating a spatial index, use the CREATE INDEX statement or **Create Index Wizard** as you would when creating an index for any other data type. However, when creating indexes on spatial data, it is recommended that you do not include more than one spatial column in the index, and that you position the spatial column last in the index definition.

Also, to include a spatial column in an index, the column must have a SRID constraint.

Indexes on spatial data can reduce the cost of evaluating relationships between geometries. For example, suppose that you are considering changing the boundaries of your sales regions and want to determine the impact on existing customers. To determine which customers are located within a proposed sales region, you could use the ST_Within method to compare a point representing each customer address to a polygon representing the sales region. Without any index, the database server must test every address point in the Customer table against the sales region polygon to determine if it should be returned in the result, which could be expensive if the Customer table is large, and inefficient if the sales region is small. An index including the address point of each customer may help to return results faster. If a predicate can be added to the query relating the sales region to the states which it overlaps, results might be obtained even faster using an index that includes both the state code and the address point.

Spatial queries *may* benefit from a clustered index, but other uses of the table need to be considered before deciding to use a clustered index. You should consider, and test, the types of queries that are likely to be performed to see whether performance improves with clustered indexes.

While you can create text indexes on a spatial column, they offer no advantage over regular indexes; regular indexes are recommended instead.

Note: Spatial columns cannot be included in a primary key, unique index, or unique constraint.

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 SAP Sybase IQ does

not support user-defined types, the SAP Sybase IQ 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)
```

SAP Sybase IQ 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 of the Massdata.CenterPoint column:

```
SELECT CenterPoint.ST_X() FROM Massdata;
```

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.

Using static methods

In addition to instance methods, the ANSI/SQL standard allows user-defined types to have static methods associated with them. These are invoked using the following syntax:

```
type-name::method-name( argument-list )
```

For example, the following instantiates an ST_Point by parsing text:

```
SELECT ST_Geometry::ST_GeomFromText('POINT( 5 6 )')
```

A static method invocation gives an error in the following cases:

- If the declared type of *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

Using static aggregate methods (SAP Sybase IQ extension)

As an extension to ANSI/SQL, SAP Sybase IQ supports static methods that implement user-defined aggregates. For example:

```
SELECT ST_Geometry::ST_AsSVGAggr(T.geo) FROM table T
```

All of the overloads for a static method must be aggregate or none of them may be aggregate.

A static aggregate method invocation gives an error in the following cases:

- If a static method invocation would give an error
- If a built-in aggregate function would give an error
- If a WINDOW clause is specified

Using type predicates

The ANSI/SQL standard defines type predicates that allow a statement to examine the concrete type (also called object type in other languages) of a value. The syntax is as follows:

```
value  IS [ NOT ] OF ( [ ONLY ] type-name,...)
```

If *value* is NULL, the predicate returns UNKNOWN. Otherwise, the concrete type of *value* is compared to each of the elements in the *type-name* list. If ONLY is specified, there is a match if the concrete type is exactly the specified type. Otherwise, there is a match if the concrete type is the specified type or any derived type (subtype).

If the concrete type of *value* matches one of the elements in the list, TRUE is returned, otherwise FALSE.

The following example returns all rows where the Shape column value has the concrete type ST_Curve or one if its subtypes (ST_LineString, ST_CircularString, or ST_CompoundCurve):

```
SELECT * FROM SpatialShapes WHERE Shape IS OF ( ST_Curve );
```

Using the TREAT expression for subtypes

The ANSI/SQL standard defines a subtype treatment expression that allows the declared type of an expression to be efficiently changed from a supertype to a subtype. This can be used when you know the concrete type (also called object type in other languages) of the expression is the specified subtype or a subtype of the specified subtype. This is more efficient that using

the CAST function since the CAST function makes a copy of the value, while TREAT does not make a copy. The syntax is as follows:

```
TREAT( value-expression AS target-subtype )
```

If no error condition is raised, the result is the *value-expression* with declared type of *target-subtype*.

The subtype treatment expression gives an error in the following cases:

- If *value-expression* is not a user-defined type
- If *target-subtype* is not a subtype of the declared type of *value-expression*
- If the dynamic type of *value-expression* is not a subtype of *target-subtype*

The following example effectively changes the declared type of the ST_Geometry Shape column to the ST_Curve subtype so that the ST_Curve type's ST_Length method can be called:

```
SELECT ShapeID, TREAT( Shape AS ST_Curve ).ST_Length() FROM
SpatialShapes WHERE Shape IS OF ( ST_Curve );
```

How to create geometries

There are several methods for creating geometries in a database:

- **Load from Well Known Text (WKT) or Well Known Binary (WKB) formats –** You can load or insert data in WKT or WKB formats. These formats are defined by the OGC, and all spatial database vendors support them. SAP Sybase IQ performs automatic conversion from these formats to geometry types.
- **Load from ESRI shapefiles –** You can load data from ESRI shapefiles into a new or existing table. There are a number of ways to do this.
- **Use a SELECT...FROM OPENSTRING statement –** You can execute a SELECT... FROM OPENSTRING statement on a file containing the spatial data. For example:

  ```
  INSERT INTO world_cities( country, city, point )
     SELECT country, city, NEW ST_Point( longitude, latitude, 4326 )
        FROM OPENSTRING( FILE 'capitalcities.csv' )
             WITH(
                   country   CHAR(100),
                   city      CHAR(100),
                   latitude  DOUBLE,
                   longitude DOUBLE )
  ```

- **Create coordinate points by combining latitude and longitude values –** You can combine latitude and longitude data to create a coordinate of spatial data type ST_Point. For example, if you had a table that already has latitude and longitude columns, you can create an ST_Point column that holds the values as a point using a statement similar to the following:

```
ALTER TABLE my_table
   ADD point AS ST_Point(SRID=4326)
   COMPUTE( NEW ST_Point( longitude, latitude, 4326 ) );
```

- **Create geometries using constructors and static methods –** You can create geometries using constructors and static methods.

Viewing spatial data as images (Interactive SQL)

In Interactive SQL, you can view a geometry as an image using the **Spatial Preview** tab to understand what the data in the database represents.

Prerequisites

You must have SELECT privilege on the table you are selecting from, or the SELECT ANY TABLE system privilege.

Task

Each instance of Interactive SQL is associated with a different connection to a database. When you open an instance of the **Spatial Viewer** from within Interactive SQL, that instance of **Spatial Viewer** remains associated with that instance of Interactive SQL, and shares the connection to the database.

When you execute a query in the **Spatial Viewer**, if you attempt to execute a query in the associated instance of Interactive SQL, you get an error. Likewise, if you have multiple instances of the **Spatial Viewer** open that were created by the same instance of Interactive SQL, only one of those instances can execute a query at a time; the other instances must wait for the query to finish.

Note: By default, Interactive SQL truncates values in the **Results** pane to 256 characters. If Interactive SQL returns an error indicating that the full column value could not be read, increase the truncation value. To do this, click **Tools** » **Options** and click **SAP Sybase IQ** in the left pane. On the **Results** tab, change **Truncation Length** to a high value, such as 5000. Click **OK** to save your changes, execute query again, and then double-click the row again.

1. Connect to your database in Interactive SQL.
2. Execute a query to select spatial data from a table. For example:
   ```
   SELECT * FROM owner.spatial-table;
   ```
3. Double-click any value in the Shapes column in the **Results** pane to open the value in the **Value** window.

   The value is displayed as text on the **Text** tab of the **Value** window.
4. Click the **Spatial Preview** tab to see the geometry as a Scalable Vector Graphic (SVG).

The geometry is displayed as a Scalable Vector Graphic (SVG).

Next

The spatial data can be viewed as geometry by using the **Previous Row** and **Next Row** buttons to view other rows in the result set.

Viewing spatial data as images (Spatial Viewer)

You can view multiple geometries as an image to understand what the data in the database represents using the Spatial Viewer.

Prerequisites

You must have SELECT privilege on the table you are selecting from, or the SELECT ANY TABLE system privilege.

Task

The order of rows in a result matter to how the image appears in the **Spatial Viewer** because the image is drawn in the order in which the rows are processed, with the most recent appearing on the top. Shapes that occur later in a result set can obscure ones that occur earlier in the result set.

1. Connect to your database in Interactive SQL, click **Tools** » **Spatial Viewer**.
2. In the **Spatial Viewer**, execute a query similar to the following in the **SQL** pane and then click **Execute**:
   ```
   SELECT * FROM GROUPO.SpatialShapes;
   ```
3. Use the **Draw Outlined Polygons** tool to remove the coloring from the polygons in a drawing to reveal the outline of all shapes. This tool is located beneath the image, near the controls for saving, zooming, and panning.

All of the geometries in the result set are displayed in the **Results** area as one image.

Loading spatial data from a Well Known Text (WKT) file

You can add spatial data to a table by using a Well Known Text file (WKT) that contains text that can be used to load spatial data into a database and be represented as geometry.

Prerequisites

The privileges required to load data depend on the -gl server option. If the -gl option is set to ALL, one of the following must be true:

you are the owner of the table

you have LOAD privilege on the table
you have the LOAD ANY TABLE system privilege
you have the ALTER ANY TABLE system privilege

If the -gl option is set to DBA, you must the LOAD ANY TABLE or ALTER ANY TABLE system privilege.

If the -gl option is set to NONE, LOAD TABLE is not permitted.

When loading from a file on a client computer:

- READ CLIENT FILE privilege is also required.
- Read privileges are required on the directory being read from.
- The allow_read_client_file database option must be enabled.
- The read_client_file secure feature must be enabled.

Task

1. Create a file that contains spatial data in WKT format that you can load into the database.

   The file can be in any format supported by the LOAD TABLE statement.
2. In Interactive SQL, connect to your database.
3. Create a table and load the data from the file into using a statement similar to the following:

```
DROP TABLE IF EXISTS SA_WKT;
CREATE TABLE SA_WKT (
    description CHAR(24),
    sample_geometry ST_Geometry(SRID=1000004326)
);

LOAD TABLE SA_WKT FROM 'C:\\Documents and Settings\\All Users\
\Documents\\SAP Sybase IQ 16\\Samples\\wktgeometries.csv'
DELIMITED BY ',';
```

   The data is loaded into the table.

The spatial data is successfully loaded from the WKT file.

Next

You can view the data in Interactive SQL using the **Spatial Viewer**.

Create or Manage a Spatial Reference System

Use Interactive SQL or SAP Control Center to create and manage SAP Sybase IQ spatial reference units of measure.

The unit of measure that you want to associate with the SRS must already exist.

With SAP Control Center, you can create a spatial reference system (SRS) that uses an existing one as a template and then edit the settings. Therefore you should choose a spatial reference system that is similar to the one you want to create.

To create a spatial reference system requires one of:

- MANAGE ANY SPATIAL OBJECT system privilege.
- CREATE ANY OBJECT system privilege.

To modify a spatial reference system requires one of:

- You are the owner of the spatial reference system.
- ALTER privilege on the spatial reference system
- MANAGE ANY SPATIAL OBJECT system privilege
- ALTER ANY OBJECT system privilege.

To delete a spatial reference system requires one of:

- MANAGE ANY SPATIAL OBJECT system privilege.
- DROP ANY OBJECT system privilege.
- You own the spatial references system.

Create or Manage a Spatial Unit of Measure

Several units of measure are installed with the software. If the installed units of measure are not appropriate for your data, you can create your own.

Use Interactive SQL or SAP Control Center to create and manage SAP Sybase IQ spatial units of measure.

To create a spatial unit of measure requires one of:

- MANAGE ANY SPATIAL OBJECT system privilege.
- CREATE ANY OBJECT system privilege.

To delete a spatial unit of measure requires one of:

- MANAGE ANY SPATIAL OBJECT system privilege.
- DROP ANY OBJECT system privilege.
- You own the spatial unit of measure.

Advanced spatial topics

This section contains advanced spatial topics.

How flat-Earth and round-Earth representations work

SAP Sybase IQ 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.

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 WGS 84, 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.

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.

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.

How snap-to-grid and tolerance impact spatial calculations

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. SAP Sybase IQ 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, SAP Sybase IQ 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 line represents 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.

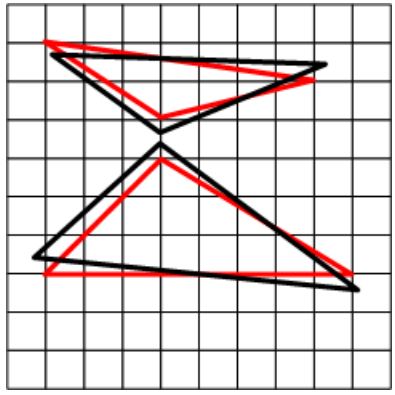
Snap-to-grid and tolerance are set on the spatial reference system and are always specified in same units as the X and Y (or Longitude and Latitude) coordinates. Snap-to-grid and tolerance work together to overcome issues with inexact arithmetic and imprecise data. However, you should be aware of how 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. For round-Earth spatial reference systems, grid size and tolerance must be set to 0. SAP Sybase IQ uses fixed grid size and tolerance on an internal projection when performing round-Earth operations.

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

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.

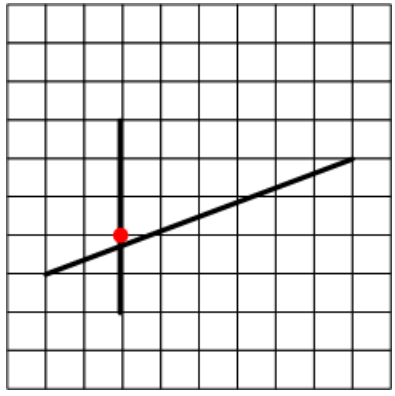


Example 2: Tolerance impacts intersection results

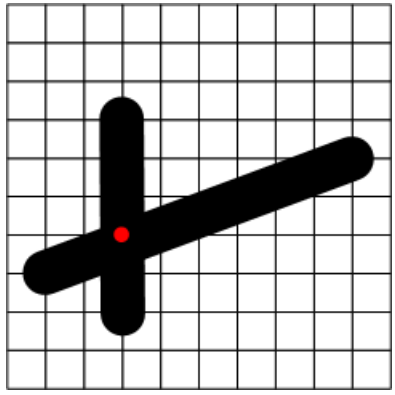
In the following example, two lines lie in a spatial reference system where tolerance is set to 0. The intersection point of the two lines is snapped to the nearest vertex in the grid. Since tolerance is set to 0, a test to determine if the intersection point of the two lines intersects the diagonal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

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

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

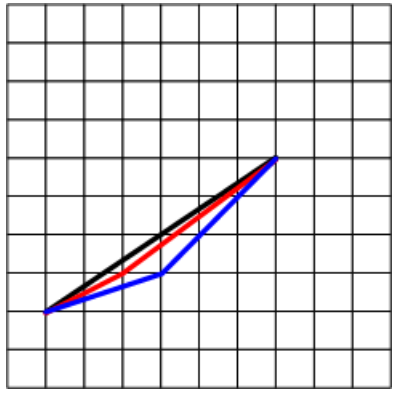


Setting the tolerance to grid size (the default), however, causes the intersection point to be inside the thick diagonal line. So a test of whether the intersection point intersects the diagonal line within tolerance would pass:



Example 3: Tolerance and transitivity

In spatial calculations when tolerance is in use, transitivity does not necessary 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 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 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 SAP Sybase IQ 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.

How interpolation impacts spatial calculations

Interpolation is the process of using known points in a geometry to approximate unknown points. Several spatial methods and predicates use interpolation when the calculations involve circular arcs. Interpolation turns a circular arc into a sequence of straight lines. For example, a circularstring representing a quarter arc might be interpolated as a linestring with 11 control points.

Interpolation example

1. In Interactive SQL, connect to the sample database execute the following statement to create a variable called arc in which you will store a circularstring:

   ```
   CREATE VARIABLE arc ST_CircularString;
   ```
2. Execute the following statement to create a circularstring and store it in the arc variable:

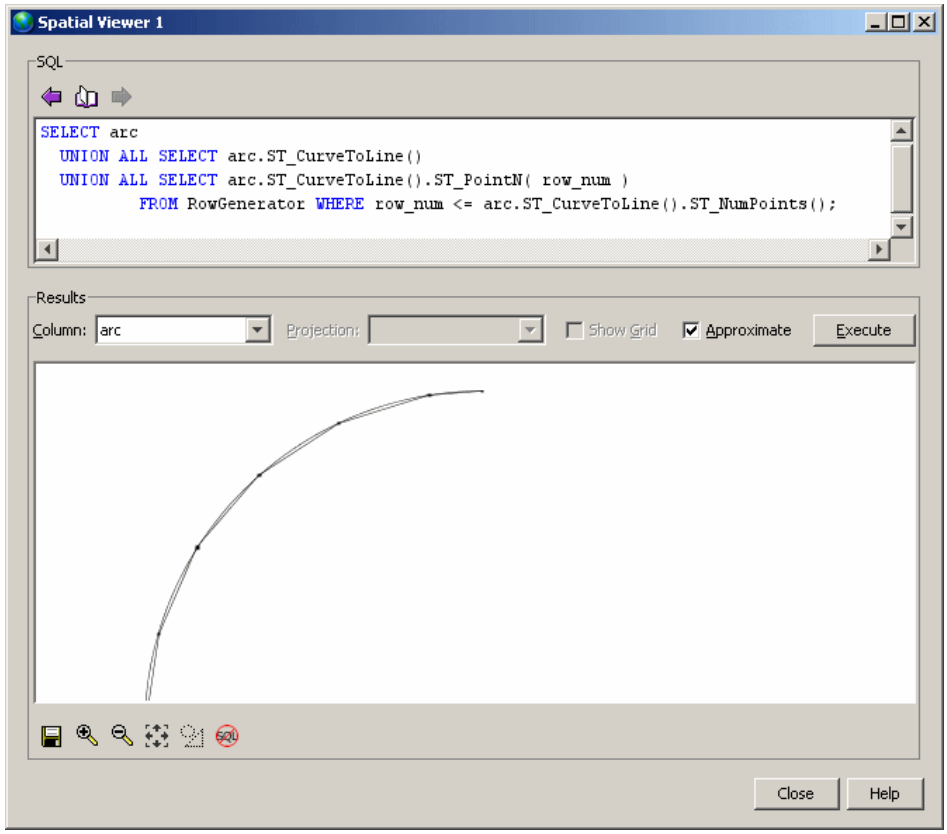
   ```
   SET arc = NEW ST_CircularString( 'CircularString( -1 0, -0.707107
   0.707107, 0 1 )' );
   ```
3. Execute the following statement to temporarily set the relative tolerance to 1% using the st_geometry_interpolation option.

   ```
   SET TEMPORARY OPTION st_geometry_interpolation = 'relative-
   tolerance-percent=1';
   ```

   Setting relative tolerance to 1% is optional, but makes the effects of interpolation more visible for the purposes of this example.
4. Open the **Spatial Viewer** (in Interactive SQL, click **Tools** » **Spatial Viewer**) and execute the following query to view the circularstring:

   ```
   SELECT arc
     UNION ALL SELECT arc.ST_CurveToLine()
   ```

```
  UNION ALL SELECT arc.ST_CurveToLine().ST_PointN( row_num )
  FROM RowGenerator WHERE row_num <=
arc.ST_CurveToLine().ST_NumPoints();
```



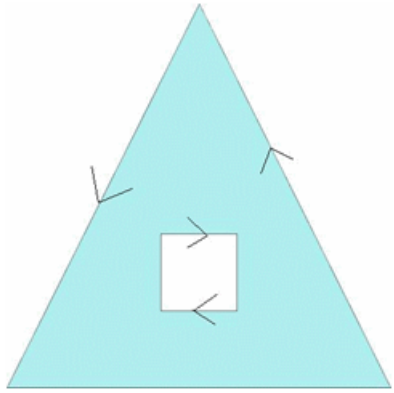
Notice how the arc is broken into a sequence of linestring. Since relative tolerance was set to 1%, each line segment shows up as a line that bows in from the true arc. The maximum distance between the interpolated line string and the true arc is 1% of the radius of the arc.

How polygon ring orientation works

Internally, SAP Sybase IQ 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 SAP Sybase IQ internal ring orientation. Use the POLYGON FORMAT clause of the CREATE SPATIAL REFERENCE SYSTEM statement to specify the orientation of polygon rings of the input data. This should only be done if all input data for the spatial reference system uses the same ring orientation. The polygon format can also be specified on some polygon and multisurface constructors.

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.

How geometry interiors, exteriors, and boundaries work

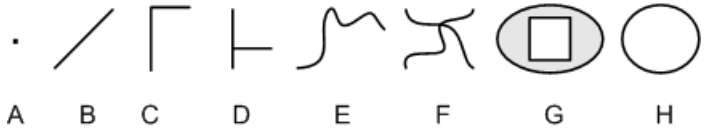
The **interior** of a geometry is all points that are part of the geometry except the boundary.

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:



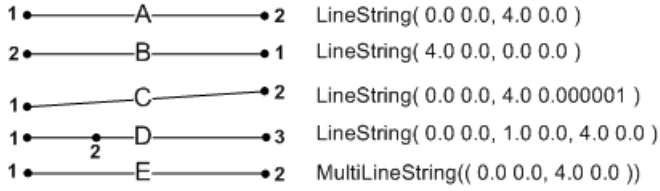
- **Point –** A point (such as A) has no boundary.
- **Lines and curves –** The boundary for lines and curves (B, C, D, E, F) are their endpoints. Geometries B, C, and E have two end points for a boundary. Geometry D has four end points for a boundary, and geometry F has four.
- **Polygon –** The boundary for a polygon (such as G) is its outer ring and any inner rings.
- **Rings –** 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.

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.

- **ST_Equals –** 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.
- **ST_OrderingEquals –** 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 need to 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.



How SAP Sybase IQ performs comparisons of geometries

The database server uses ST_OrderingEquals to perform operations such as GROUP BY and DISTINCT.

For example, when processing the following query the server considers two rows to be equal if the two shape expressions have ST_OrderingEquals() = 1:

```
SELECT DISTINCT Shape FROM GROUPO.SpatialShapes;
```

SQL statements can compare two geometries using the equal to operator (=), or not equal to operator (<> or !=), including search conditions with a subquery and the ANY or ALL keyword. Geometries can also be used in an IN search condition. For example, `geom1 IN (geom-expr1, geom-expr2, geom-expr3)`. For all of these search conditions, equality is evaluated using the ST_OrderingEquals semantics.

You cannot use other comparison operators to determine if one geometry is less than or greater than another (for example, `geom1 < geom2` is not accepted). This means you cannot include geometry expressions in an ORDER BY clause. However, you can test for membership in a set.

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:

	g2 interior	**g2 boundary**	**g2 exterior**
g1 interior	`Interior(g1) ∩ Interi-or(g2)`	`Interior(g1) ∩ Boun-dary(g2)`	`Interior(g1) ∩ Exteri-or(g2)`

g1 boundary	`Boundary(g1) ∩ Interi-or(g2)`	`Boundary(g1) ∩ Boun-dary(g2)`	`Boundary(g1) ∩ Exteri-or(g2)`
g1 exterior	`Exterior(g1) ∩ Interi-or(g2)`	`Exterior(g1) ∩ Boun-dary(g2)`	`Exterior(g1) ∩ Exteri-or(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.

When you specify:	The intersection test result must re-turn:
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***' );
```

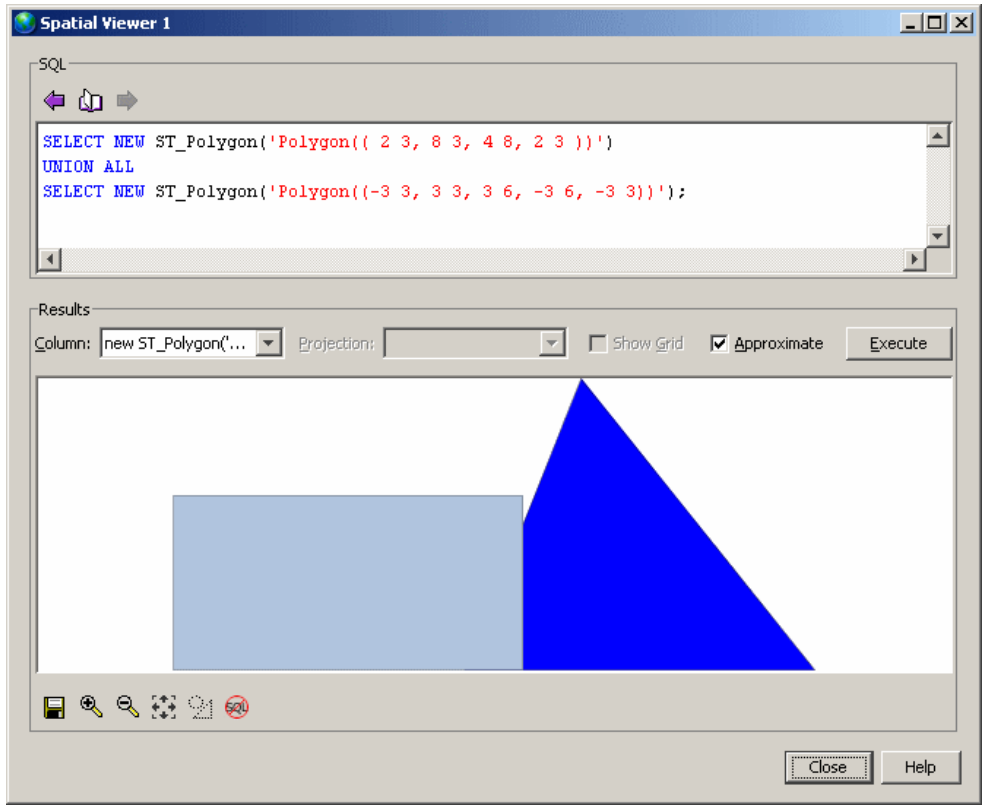
This is equivalent to asking ST_Relate to look for the following conditions when performing the intersection tests:

	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.

To view the two geometries and compare their appearance to what is being tested, execute the following statement in the Interactive SQL Spatial Viewer (**Tools** » **Spatial Viewer**):

```
SELECT NEW ST_Polygon('Polygon(( 2 3, 8 3, 4 8, 2 3 ))')
UNION ALL
SELECT NEW ST_Polygon('Polygon((-3 3, 3 3, 3 6, -3 6, -3 3))');
```



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))'));
```

ST_Relate returns the DE-9IM string, `212111212`.

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

	g2 interior	g2 boundary	g2 exterior
g1 interior	2	1	2

g1 boundary	1	1	1
g1 exterior	2	1	2

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 are subtypes of ST_Curve (ST_LineString, ST_CircularString, and ST_CompoundCurve), or collection types containing these types, but no surfaces. 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 and ST_CurvePolygon), 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_GeomCollection can contain geometries of different dimensions, and the highest dimension geometry is returned

Tutorial: Experimenting with the spatial features

This tutorial allows you to experiment with some of the spatial features in SAP Sybase IQ. To do so, you will first load an ESRI shapefile into your sample database (iqdemo.db) to give you some valid spatial data to experiment with.

The tutorial is broken into the following parts:

Lesson 1: Install additional units of measure and spatial reference systems
Lesson 2: Download the ESRI shapefile data
Lesson 3: Load the ESRI shapefile data
Lesson 4: Query spatial data
Lesson 5: Output spatial data to SVG
Lesson 6: Project spatial data

Privileges
To perform this tutorial, you must have the following privileges:

MANAGE ANY SPATIAL OBJECT system privilege
CREATE TABLE system privilege
WRITE FILE system privilege
SELECT privilege on the GROUPO.SpatialContacts table

Lesson 1: Install additional units of measure and spatial reference systems

This lesson shows you how to use the sa_install_feature system procedure to install many predefined units of measure and spatial reference systems you will need later in this tutorial.

Prerequisites

This lesson assumes you have the roles and privileges listed in the Privileges section at the start of this tutorial: Tutorial: Experimenting with the spatial features.

Task

1. Using Interactive SQL, start and connect to the sample database (iqdemo.db).

   The sample database is located in `"%ALLUSERPROFILE%"\SybaseIQ\demo`.

2. Execute the following statement in Interactive SQL:

   ```
   CALL sa_install_feature( 'st_geometry_predefined_srs' );
   ```

   When the statement finishes, the additional units of measure and spatial reference systems have been installed.
3. To determine the units of measure installed in your database, execute the following query in Interactive SQL:

   ```
   SELECT * FROM ST_UNITS_OF_MEASURE;
   ```
4. To determine the spatial reference systems installed in your database, execute the following query in Interactive SQL:

   ```
   SELECT * FROM ST_SPATIAL_REFERENCE_SYSTEMS;
   ```

The list of installed spatial reference systems is returned.

Lesson 2: Download the ESRI shapefile data

Prerequisites

This lesson assumes you have completed all preceding lessons. See Lesson 1: Install additional units of measure and spatial reference systems.

This lesson assumes you have the roles and privileges listed in the Privileges section at the start of this tutorial: Tutorial: Experimenting with the spatial features.

Task

1. Create a local directory called `c:\temp\massdata`.
2. Go to the following URL: *http://www2.census.gov/cgi-bin/shapefiles2009/national-files*
3. On the right side of the page, in the **State- and County-based Shapefiles** dropdown, click **Massachusetts**, and then click **Submit**.
4. On the left side of the page, click **5-Digit ZIP Code Tabulation Area (2002)**, and then click **Download Selected Files**.
5. When prompted, save the zip file, `multiple_tiger_files.zip`, to `c:\temp\massdata`, and extract its contents. This creates a subdirectory called 25_MASSACHUSETTS containing another zip file called `tl_2009_25_zcta5.zip`.
6. Extract the contents of `tl_2009_25_zcta5.zip` to `C:\temp\massdata`.

This lesson unpacks five files, including an ESRI shapefile (`.shp`) that you can use to load the spatial data into the database.

Lesson 3: Load the ESRI shapefile data

This lesson shows you how to determine the columns in the ESRI shapefile and use that information to create a table that you will load the data into.

Prerequisites

This lesson assumes you have completed all preceding lessons. See Lesson 1: Install additional units of measure and spatial reference systems.

This lesson assumes you have the roles and privileges listed in the Privileges section at the start of this tutorial: Tutorial: Experimenting with the spatial features.

Task

If you have difficulty running any of the steps due to privilege problems, ask your administrator what value the -gl database option is set to, and then read the privilege section of the st_geometry_load_shapefile system procedure to determine the corresponding privileges you need.

1. Since spatial data is associated with a specific spatial reference system, when you load data into the database, you must load it into the same spatial reference system, or at least one with an equivalent definition. To find out the spatial reference system information for the ESRI shapefile, open the project file, `c:\temp\massdata\tl_2009_25_zcta5.prj`, in a text editor. This file contains the spatial reference system information you need.

   ```
   GEOGCS["GCS_North_American_1983", DATUM["D_North_American_1983",
   SPHEROID["GRS_1980",6378137,298.257222101]],
   PRIMEM["Greenwich",0],UNIT["Degree",0.017453292519943295]]
   ```

   The string **GCS_North_American_1983** is the name of the spatial reference system associated with the data.

2. A quick query of the ST_SPATIAL_REFERENCE_SYSTEMS view, `SELECT * FROM ST_SPATIAL_REFERENCE_SYSTEMS WHERE srs_name='GCS_North_American_1983';`, reveals that this name is not present in the list of predefined SRSs. However, you can query for a spatial reference system with the same definition and use it instead:

   ```
   SELECT *
   FROM ST_SPATIAL_REFERENCE_SYSTEMS
   WHERE definition LIKE '%1983%'
   AND definition LIKE 'GEOGCS%';
   ```

   The query returns a single spatial reference system, NAD83 with SRID **4269** that has the same definition. This is the SRID you will assign to the data you load from the shapefile.

3. In Interactive SQL, execute the following statement to create a table called Massdata, load the shapefile into the table, and assign SRID 4269 to the data. he load may take a minute.

```
CALL st_geometry_load_shapefile ( 'c:\\temp\\massdata\
\tl_2009_25_zcta5.shp',
4269,
'Massdata' );
```

Note: The **Import Wizard** also supports loading data from shapefiles.

4. In Interactive SQL, query the table to view the data that was in the shapefile:

```
SELECT * FROM Massdata;
```

Each row in the results represents data for a zip code region.

The geometry column holds the shape information of the zip code region as either a polygon (one area) or multipolygon (two or more noncontiguous areas).

5. The ZCTA5CE column holds zip codes. To make it easier to refer to this column later in the tutorial, execute the following ALTER TABLE statement n Interactive SQL to change the column name to `ZIP`:

```
ALTER TABLE Massdata
RENAME ZCTA5CE TO ZIP;
```

6. The two columns, INTPTLON and INTPTLAT, represent the X and Y coordinates for the center points of the zip code regions. Execute the following ALTER TABLE statement in Interactive SQL to create a column called CenterPoint of type ST_Point, and to turn each X and Y set into a value in CenterPoint.

```
ALTER TABLE Massdata
ADD CenterPoint AS ST_Point(SRID=4269)
COMPUTE( new ST_Point( CAST( INTPTLON AS DOUBLE ), CAST( INTPTLAT
AS DOUBLE ), 4269 ) );
```

Now, each ST_Point value in Massdata.CenterPoint represents the center point of the zip code region stored in Massdata.geometry.

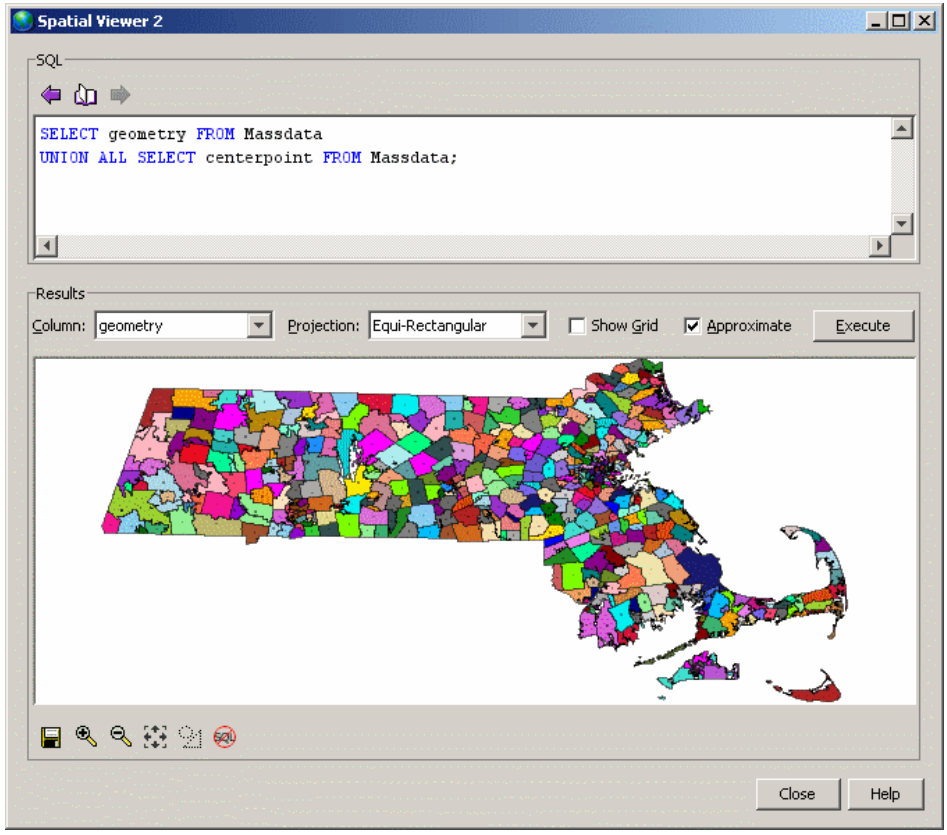
7. To view an individual geometry (a zip code region) as a shape, double-click any value except the first one in Massdata.geometry and then click the **Spatial Preview** tab of the **Value Of Column** window.

If you receive an error saying the value is to large, or suggesting you include a primary key in the results, it is because the value has been truncated for display purposes in Interactive SQL. To fix this, you can either modify the query to include the primary key column in the results, or adjust the **Truncation Length** setting for Interactive SQL. Changing the setting is recommended if you don't want to have to include the primary key each time you query for geometries with the intent to view them in Interactive SQL.

To change the **Truncation Length** setting for Interactive SQL, click **Tools** » **Options** » **SAP Sybase IQ**, set **Truncation Length** to a high number such as 100000.

8. To view the entire data set as one shape, click **Tools** » **Spatial Viewer** to open the SAP Sybase IQ **Spatial Viewer** and execute the following query in Interactive SQL:

```
SELECT geometry FROM Massdata
UNION ALL SELECT CenterPoint FROM Massdata;
```



The ESRI shapefile data is loaded.

Lesson 4: Query spatial data

This lesson shows you how to use some of the spatial methods to query the data in a meaningful context. You will also learn how to calculate distances, which requires you to add units of measurement to your database.

Prerequisites

This lesson assumes you have completed all preceding lessons. See Lesson 1: Install additional units of measure and spatial reference systems.

This lesson assumes you have the roles and privileges listed in the Privileges section at the start of this tutorial: Tutorial: Experimenting with the spatial features.

Task

The queries are performed on one or both of the SpatialContacts and Massdata tables. The SpatialContacts, which was already present in your database, holds names and contact information for people—many of whom live in Massachusetts.

1. In Interactive SQL, create a variable named `@Mass_01775` to hold the associated geometry for the zip code region 01775.

   ```
   CREATE VARIABLE @Mass_01775 ST_Geometry;
   SELECT geometry INTO @Mass_01775
   FROM Massdata
   WHERE ZIP = '01775';
   ```

2. Suppose you want to find all contacts in SpatialContacts in the zip code area 01775 and surrounding zip code areas. For this, you can use the ST_Intersects method, which returns geometries that intersects with, or are the same as, the specified geometry. You would execute the following statement in Interactive SQL:

   ```
   SELECT c.Surname, c.GivenName, c.Street, c.City, c.PostalCode,
   z.geometry
   FROM Massdata z, GROUPO.SpatialContacts c
   WHERE
   c.PostalCode = z.ZIP
   AND z.geometry.ST_Intersects( @Mass_01775 ) = 1;
   ```

3. All rows in Massdata.geometry are associated with the same spatial reference system (SRID 4269) because you assigned SRID 4269 when you created the geometry column and loaded data into it.

   However, it is also possible to create an **undeclared** ST_Geometry column (that is, without assigning a SRID to it). This may be necessary if you intend store spatial values that have different SRSs associated to them in a single column. When operations are performed on these values, the spatial reference system associated with each value is used.

   One danger of having an undeclared column, is that the database server does not prevent you from changing a spatial reference system that is associated with data in an undeclared column.

   If the column has a declared SRID, however, the database server does not allow you to modify the spatial reference system associated with the data. You must first unload and then truncate the data in the declared column, change the spatial reference system, and then reload the data.

   You can use the ST_SRID method to determine the SRID associated with values in a column, regardless of whether it is declared or not. For example, the following statement shows the SRID assigned to each row in the Massdata.geometry column:

   ```
   SELECT geometry.ST_SRID()
   FROM Massdata;
   ```

4. You can use the ST_CoveredBy method to check that a geometry is completely contained within another geometry. For example, Massdata.CenterPoint (ST_Point type) contains the latitude/longitude coordinates of the center of the zip code area, while Massdata.geometry contains the polygon reflecting the zip code area. You can do a quick

check to make sure that no CenterPoint value has been set outside its zip code area by executing the following query in Interactive SQL:

```
SELECT * FROM Massdata
WHERE NOT(CenterPoint.ST_CoveredBy(geometry) = 1);
```

No rows are returned, indicating that all CenterPoint values are contained within their associated geometries in Massdata.geometry. This check does not validate that they are the true center, of course. You would need to project the data to a flat-Earth spatial reference system and check the CenterPoint values using the ST_Centroid method. You will learn about projection later in this tutorial.

5. You can use the ST_Distance method to measure the distance between the center point of the zip code areas. For example, suppose you want the list of zip code within 100 miles of zip code area 01775. You could execute the following query in Interactive SQL:

```
SELECT c.PostalCode, c.City,
       z.CenterPoint.ST_Distance( ( SELECT CenterPoint
                            FROM Massdata WHERE ZIP = '01775' ),
                          'Statute mile' ) dist,
       z.CenterPoint
FROM Massdata z, GROUPO.SpatialContacts c
WHERE c.PostalCode = z.ZIP
  AND dist <= 100
ORDER BY dist;
```

6. If knowing the exact distance is not important, you could construct the query using the ST_WithinDistance method instead, which can offer better performance for certain datasets (in particular, for large geometries):

```
SELECT c.PostalCode, c.City, z.CenterPoint
FROM Massdata z, GROUPO.SpatialContacts c
WHERE c.PostalCode = z.ZIP
  AND z.CenterPoint.ST_WithinDistance( ( SELECT CenterPoint
                            FROM Massdata WHERE ZIP = '01775' ),
                          100, 'Statute mile' ) = 1
ORDER BY c.PostalCode;
```

The queries are executed on the spatial data.

Lesson 5: Output spatial data to SVG

In this lesson, you create an SVG document to view a multipolygon expressed in WKT. You can export geometries to SVG format for viewing in Interactive SQL or in an SVG-compatible application.

Prerequisites

This lesson assumes you have completed all preceding lessons. See Lesson 1: Install additional units of measure and spatial reference systems.

This lesson assumes you have the roles and privileges listed in the Privileges section at the start of this tutorial: Tutorial: Experimenting with the spatial features.

Task

1. In Interactive SQL, execute the following statement to create a variable with an example geometry:

```
CREATE OR REPLACE VARIABLE @svg_geom
ST_Polygon = (NEW ST_Polygon('Polygon ((1 1, 5 1, 5 5, 1 5, 1 1),
(2 2, 2 3, 3 3, 3 2, 2 2))'));
```

2. In Interactive SQL, execute the following SELECT statement to call the ST_AsSVG method:

```
SELECT @svg_geom.ST_AsSVG() AS svg;
```

   The result set has a single row that is an SVG image. You can view the image using the **SVG Preview** feature in Interactive SQL. To do this, double-click the result row, and select the **SVG Preview** tab. You should see a square geometry inside of another square geometry.

   Note: By default, Interactive SQL truncates values in the **Results** pane to 256 characters. If Interactive SQL returns an error indicating that the full column value could not be read, increase the truncation value. To do this, click **Tools** » **Options** and click **SAP Sybase IQ** in the left pane. On the **Results** tab, change **Truncation Length** to a high value, such as 5000. Click **OK** to save your changes, execute query again, and then double-click the row again.

3. The previous step described how to preview an SVG image within Interactive SQL. However, it may be more useful to write the resulting SVG to a file so that it can be read by an external application. You could use the xp_write_file system procedure or the WRITE_CLIENT_FILE function [String] to write to a file relative to either the database server or the client computer. In this example, you will use the OUTPUT statement [Interactive SQL].

   In Interactive SQL, execute the following SELECT statement to call the ST_AsSVG method and output the geometry to a file named `myPolygon.svg`:

```
SELECT @svg_geom.ST_AsSVG();
OUTPUT TO 'c:\\temp\\massdata\\myPolygon.svg'
QUOTE ''
ESCAPES OFF
FORMAT TEXT
```

   You must include the `QUOTE ''` and `ESCAPES OFF` clauses, otherwise line return characters and single quotes are inserted in the XML to preserve whitespace, causing the output to be an invalid SVG file.

4. Open the SVG in a web browser or application that supports viewing SVG images. Alternatively, you can open the SVG in a text editor to view the XML for the geometry.
5. The ST_AsSVG method generates an SVG image from a single geometry. In some cases, you want to generate an SVG image including all of the shapes in a group. The ST_AsSVGAggr method is an aggregate function that combines multiple geometries into a single SVG image. First, using Interactive SQL, create a variable to hold the SVG image and generate it using the ST_AsSVGAggr method.

```
CREATE OR REPLACE VARIABLE @svg XML;
SELECT ST_Geometry::ST_AsSVGAggr( geometry,
'attribute=fill="black"' )
INTO @svg
FROM Massdata;
```

The `@svg` variable now holds an SVG image representing all of the zip code regions in the Massdata table. The `'attribute=fill="black"'` specifies the fill color that is used for the generated image. If not specified, the database server chooses a random fill color. Now that you have a variable containing the SVG image you are interested in, you can write it to a file for viewing by other applications. Execute the following statement in Interactive SQL to write the SVG image to a file relative to the database server.

```
CALL xp_write_file( 'c:\\temp\\Massdata.svg', @svg );
```

The WRITE_CLIENT_FILE function could also be used to write a file relative to the client application, but additional steps may be required to ensure appropriate privileges are enabled. If you open the SVG image in an application that supports SVG data, you should see an image like the following:

The image is not uniformly black; there are small gaps between the borders of adjacent zip code regions. These are actually white lines between the geometries and is characteristic of the way the SVG is rendered. There are not really any gaps in the data. Larger white lines are rivers and lakes.

The geometry has been viewed as an SVG.

Lesson 6: Project spatial data

This lesson shows you how to project data into a spatial reference system that uses the flat-Earth model so that you can calculate area and distance measurements.

Prerequisites

This lesson assumes you have completed all preceding lessons. See Lesson 1: Install additional units of measure and spatial reference systems.

This lesson assumes you have the roles and privileges listed in the Privileges section at the start of this tutorial: Tutorial: Experimenting with the spatial features.

Task

The spatial values in Massdata were assigned SRID 4269 (NAD83 spatial reference system) when you loaded the data into the database from the ESRI shapefile. SRID 4269 is a round-Earth spatial reference system. However, calculations such as the area of geometries and some spatial predicates are not supported in the round-Earth model. If your data is currently associated with a round-Earth spatial reference system, you can create a new spatial column that projects the values into a flat-Earth spatial reference system, and then perform your calculations on that column.

1. To measure the area of polygons representing the zip code areas, you must project the data in Massdata.geometry to a flat-Earth spatial reference system.

   To select an appropriate SRID to project the data in Massdata.geometry into, use Interactive SQL to query the ST_SPATIAL_REFERENCE_SYSTEMS consolidated view for a SRID containing the word Massachusetts, as follows:

   ```
   SELECT * FROM ST_SPATIAL_REFERENCE_SYSTEMS WHERE srs_name LIKE
   '%massachusetts%';
   ```

   This returns several SRIDs suitable for use with the Massachusetts data. For the purpose of this tutorial, **3586** will be used.

2. You must now create a column, Massdata.proj_geometry, into which you will project the geometries into 3586 using the ST_Transform method. To do so, execute the following statement in Interactive SQL:

   ```
   ALTER TABLE Massdata
   ADD proj_geometry
    AS ST_Geometry(SRID=3586)
    COMPUTE( geometry.ST_Transform( 3586 ) );
   ```

3. You can compute the area using the Massdata.proj_geometry. For example, execute the following statement in Interactive SQL:

   ```
   SELECT zip, proj_geometry.ST_ToMultiPolygon().ST_Area('Statute
   Mile') AS area
   ```

```
FROM Massdata
ORDER BY area DESC;
```

Note: ST_Area is not supported on round-Earth spatial reference systems and ST_Distance is supported but only between point geometries.

4. To see the impact that projecting to another spatial reference system has on calculations of distance, you can use the following query to compute the distance between the center points of the zip codes using the round-Earth model (more precise) or the projected flat-Earth model. Both models agree fairly well for this data because the projection selected is suitable for the data set.

```
SELECT M1.zip, M2.zip,
       M1.CenterPoint.ST_Distance( M2.CenterPoint, 'Statute
Mile' ) dist_round_earth,

M1.CenterPoint.ST_Transform( 3586 ).ST_Distance( M2.CenterPoint.S
T_Transform( 3586 ),
       'Statute Mile' ) dist_flat_earth
FROM Massdata M1, Massdata M2
WHERE M1.ZIP = '01775'
ORDER BY dist_round_earth DESC;
```

5. Suppose you want to find neighboring zip code areas that border the zip code area 01775. To do this, you would use the ST_Touches method. The ST_Touches method compares geometries to see if one geometry touches another geometry without overlapping in any way. The results for ST_Touches do not include the row for zip code 01775 (unlike the ST_Intersects method).

```
CREATE OR REPLACE VARIABLE @Mass_01775 ST_Geometry;
SELECT geometry INTO @Mass_01775
FROM Massdata
WHERE ZIP = '01775';

SELECT record_number, proj_geometry
FROM Massdata
WHERE
proj_geometry.ST_Touches( @Mass_01775.ST_Transform( 3586 ) ) = 1;
```

6. You can use the ST_UnionAggr method to return a geometry that represents the union of a group of zip code areas. For example, suppose you want a geometry reflecting the union of the zip code areas neighboring, but not including, 01775.

   In Interactive SQL, click **Tools** » **Spatial Viewer** and execute the following query:

```
SELECT ST_Geometry::ST_UnionAggr(proj_geometry)
FROM Massdata
WHERE
proj_geometry.ST_Touches( @Mass_01775.ST_Transform( 3586 ) ) = 1;
```

   Double-click the result to view it.

   If you receive an error saying the full column could not be read from the database, increase the **Truncation Length** setting for Interactive SQL. To do this, in Interactive SQL click

Tools » **Options** » **SAP Sybase IQ**, and set **Truncation Length** to a higher number. Execute your query again and view the geometry.

You have finished the tutorial.

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.

ST_CircularString type

The ST_CircularString type is a subtype of ST_Curve that uses circular line segments between control points.

Syntax

`ST_CircularString` type

Members

All members of the ST_CircularString type, including all inherited members.

Members of ST_CircularString:

- **ST_CircularString(ST_Point , ST_Point , ST_Point , ST_Point) –** Constructs a circularstring value from a list of points in a specified spatial reference system.
- **ST_CircularString() –** Constructs a circularstring representing the empty set.
- **ST_CircularString(LONG BINARY[, INT]) –** Constructs a circularstring from Well Known Binary (WKB).
- **ST_CircularString(LONG VARCHAR[, INT]) –** Constructs a circularstring from a text representation.
- **ST_NumPoints() –** Returns the number of points defining the circularstring.
- **ST_PointN(INT) –** Returns the nth point in the circularstring.

Members of ST_Curve:

- **ST_CurveToLine() –** Returns the ST_LineString interpolation of an ST_Curve value.
- **ST_EndPoint() –** Returns an ST_Point value that is the end point of the ST_Curve value.
- **ST_IsClosed() –** Test if the ST_Curve value is closed. A curve is closed if the start and end points are coincident.
- **ST_IsRing() –** Tests if the ST_Curve value is a ring. A curve is a ring if it is closed and simple (no self intersections).
- **ST_Length(VARCHAR(128)) –** Returns the length measurement of the ST_Curve value. The result is measured in the units specified by the unit-name parameter.
- **ST_StartPoint() –** Returns an ST_Point value that is the start point of the ST_Curve value.

Members of ST_Geometry:

- **ST_Affine(DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a new geometry that is the result of applying the specified 3-D affine transformation.
- **ST_AsBinary(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsBitmap(INT, INT, ST_Point , ST_Point , VARCHAR(128)) –** Returns a LONG VARBIT that is a bitmap representing a geometry value.
- **ST_AsGeoJSON(VARCHAR(128)) –** Returns a string representing a geometry in JSON format.
- **ST_AsGML(VARCHAR(128)) –** Returns the GML representation of an ST_Geometry value.
- **ST_AsKML(VARCHAR(128)) –** Returns the KML representation of an ST_Geometry value.
- **ST_AsSVG(VARCHAR(128)) –** Returns an SVG figure representing a geometry value.
- **ST_AsSVGAggr(ST_Geometry , VARCHAR(128)) –** Returns a complete or partial SVG document which renders the geometries in a group.
- **ST_AsText(VARCHAR(128)) –** Returns the text representation of an ST_Geometry value.
- **ST_AsWKB(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsWKT(VARCHAR(128)) –** Returns the WKT representation of an ST_Geometry value.
- **ST_AsXML(VARCHAR(128)) –** Returns the XML representation of an ST_Geometry value.
- **ST_Boundary() –** Returns the boundary of the geometry value.
- **ST_Buffer(DOUBLE, VARCHAR(128)) –** 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.
- **ST_Contains(ST_Geometry) –** Tests if a geometry value spatially contains another geometry value.
- **ST_ContainsFilter(ST_Geometry) –** An inexpensive test if a geometry might contain another.
- **ST_ConvexHull() –** Returns the convex hull of the geometry value.
- **ST_ConvexHullAggr(ST_Geometry) –** Returns the convex hull for all of the geometries in a group
- **ST_CoordDim() –** Returns the number of coordinate dimensions stored with each point of the ST_Geometry value.
- **ST_CoveredBy(ST_Geometry) –** Tests if a geometry value is spatially covered by another geometry value.
- **ST_CoveredByFilter(ST_Geometry) –** An inexpensive test if a geometry might be covered by another.

- **ST_Covers(ST_Geometry) –** Tests if a geometry value spatially covers another geometry value.
- **ST_CoversFilter(ST_Geometry) –** An inexpensive test if a geometry might cover another.
- **ST_Crosses(ST_Geometry) –** Tests if a geometry value crosses another geometry value.
- **ST_Debug(VARCHAR(128)) –** Returns a LONG BINARY that is debug information for the object.
- **ST_Difference(ST_Geometry) –** Returns the geometry value that represents the point set difference of two geometries.
- **ST_Dimension() –** Returns the dimension of the ST_Geometry value. Points have dimension 0, lines have dimension 1, and surfaces have dimension 2. Any empty geometry has dimension -1.
- **ST_Disjoint(ST_Geometry) –** Test if a geometry value is spatially disjoint from another value.
- **ST_Distance(ST_Geometry , VARCHAR(128)) –** Returns the smallest distance between the geometry-expression and the specified geometry value.
- **ST_Distance_Spheroid(ST_Geometry , VARCHAR(128)) –** Calculates the linear distance between geometries on the surface of the Earth.
- **ST_Envelope() –** Returns the bounding rectangle for the geometry value.
- **ST_EnvelopeAggr(ST_Geometry) –** Returns the bounding rectangle for all of the geometries in a group
- **ST_Equals(ST_Geometry) –** Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.
- **ST_EqualsFilter(ST_Geometry) –** An inexpensive test if a geometry is equal to another.
- **ST_GeometryType() –** Returns the name of the type of the ST_Geometry value.
- **ST_GeometryTypeFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_GeomFromBinary(LONG BINARY, INT) –** Constructs a geometry from a binary string representation.
- **ST_GeomFromShape(LONG BINARY[, INT]) –** Parses a string containing an ESRI shape record and creates a geometry value of the appropriate type.
- **ST_GeomFromText(LONG VARCHAR, INT) –** Constructs a geometry from a character string representation.
- **ST_GeomFromWKB(LONG BINARY, INT) –** Parse a string containing a WKB or EWKB representation of a geometry and creates a geometry value of the appropriate type.
- **ST_GeomFromWKT(LONG VARCHAR, INT) –** Parses a string containing the WKT or EWKT representation of a geometry and create a geometry value of the appropriate type.

- **ST_Intersection(ST_Geometry) –** Returns the geometry value that represents the point set intersection of two geometries.
- **ST_IntersectionAggr(ST_Geometry) –** Returns the spatial intersection of all of the geometries in a group
- **ST_Intersects(ST_Geometry) –** Test if a geometry value spatially intersects another value.
- **ST_IntersectsFilter(ST_Geometry) –** An inexpensive test if the two geometries might intersect.
- **ST_IntersectsRect(ST_Point , ST_Point) –** Test if a geometry intersects a rectangle.
- **ST_Is3D() –** Determines if the geometry value has Z coordinate values.
- **ST_IsEmpty() –** Determines whether the geometry value represents an empty set.
- **ST_IsMeasured() –** Determines if the geometry value has associated measure values.
- **ST_IsSimple() –** Determines whether the geometry value is simple (containing no self intersections or other irregularities).
- **ST_IsValid() –** Determines whether the geometry is a valid spatial object.
- **ST_LatNorth() –** Retrieves the northernmost latitude of a geometry.
- **ST_LatSouth() –** Retrieves the southernmost latitude of a geometry.
- **ST_Length_Spheroid(VARCHAR(128)) –** Calculates the linear length of a curve/multicurve on the surface of the Earth.
- **ST_LinearHash() –** Returns a binary string that is a linear hash of the geometry.
- **ST_LinearUnHash(BINARY(32)[, INT]) –** Returns a geometry representing the index hash.
- **ST_LoadConfigurationData(VARCHAR(128)) –** Returns binary configuration data. For internal use only.
- **ST_LocateAlong(DOUBLE) –** Returns the subset of the geometry value that is associated with the given measure value.
- **ST_LocateBetween(DOUBLE, DOUBLE) –** Returns the subset of the geometry value that is between the specified start measure and end measure.
- **ST_LongEast() –** Retrieves the longitude of the eastern boundary of a geometry.
- **ST_LongWest() –** Retrieves the longitude of the western boundary of a geometry.
- **ST_MMax() –** Retrieves the maximum M coordinate value of a geometry.
- **ST_MMin() –** Retrieves the minimum M coordinate value of a geometry.
- **ST_OrderingEquals(ST_Geometry) –** Tests if a geometry is identical to another geometry.
- **ST_Overlaps(ST_Geometry) –** Tests if a geometry value overlaps another geometry value.
- **ST_Relate(ST_Geometry) –** Determines how a geometry value is spatially related to another geometry value by returning an intersection matrix. The ST_Relate method returns 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.

- **ST_Reverse() –** Returns the geometry with the element order reversed.
- **ST_Segmentize(DOUBLE) –** Add points so that no line segment is longer than a specified distance.
- **ST_Simplify(DOUBLE) –** Remove points from curves so long as the maximum introduced error is less than a specified tolerance.
- **ST_SnapToGrid(ST_Point , DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a copy of the geometry with all points snapped to the specified grid.
- **ST_SRID(INT) –** Changes the spatial reference system associated with the geometry without modifying any of the values.
- **ST_SRIDFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_SymDifference(ST_Geometry) –** Returns the geometry value that represents the point set symmetric difference of two geometries.
- **ST_ToCircular() –** Convert the geometry to a circularstring
- **ST_ToCompound() –** Converts the geometry to a compound curve.
- **ST_ToCurve() –** Converts the geometry to a curve.
- **ST_ToCurvePoly() –** Converts the geometry to a curve polygon.
- **ST_ToGeomColl() –** Converts the geometry to a geometry collection.
- **ST_ToLineString() –** Converts the geometry to a linestring.
- **ST_ToMultiCurve() –** Converts the geometry to a multicurve value.
- **ST_ToMultiLine() –** Converts the geometry to a multilinestring value.
- **ST_ToMultiPoint() –** Converts the geometry to a multi-point value.
- **ST_ToMultiPolygon() –** Converts the geometry to a multi-polygon value.
- **ST_ToMultiSurface() –** Converts the geometry to a multi-surface value.
- **ST_ToPoint() –** Converts the geometry to a point.
- **ST_ToPolygon() –** Converts the geometry to a polygon.
- **ST_ToSurface() –** Converts the geometry to a surface.
- **ST_Touches(ST_Geometry) –** Tests if a geometry value spatially touches another geometry value.
- **ST_Transform(INT) –** Creates a copy of the geometry value transformed into the specified spatial reference system.
- **ST_Union(ST_Geometry) –** Returns the geometry value that represents the point set union of two geometries.
- **ST_UnionAggr(ST_Geometry) –** Returns the spatial union of all of the geometries in a group
- **ST_Within(ST_Geometry) –** Tests if a geometry value is spatially contained within another geometry value.
- **ST_WithinDistance(ST_Geometry , DOUBLE, VARCHAR(128)) –** Test if two geometries are within a specified distance of each other.

- **ST_WithinDistanceFilter(ST_Geometry , DOUBLE, VARCHAR(128)) –** An inexpensive of whether two geometries might be within a specified distance of each other.
- **ST_WithinFilter(ST_Geometry) –** An inexpensive test if a geometry might be within another.
- **ST_XMax() –** Retrieves the maximum X coordinate value of a geometry.
- **ST_XMin() –** Retrieves the minimum X coordinate value of a geometry.
- **ST_YMax() –** Retrieves the maximum Y coordinate value of a geometry.
- **ST_YMin() –** Retrieves the minimum Y coordinate value of a geometry.
- **ST_ZMax() –** Retrieves the maximum Z coordinate value of a geometry.
- **ST_ZMin() –** Retrieves the minimum Z coordinate value of a geometry.

Remarks

The ST_CircularString type is a subtype of ST_Curve 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. For example, attempting to create one for SRID 4326 returns an error.

Standards and compatibility

SQL/MM (ISO/IEC 13249-3: 2006) 7.3

ST_CircularString(ST_Point , ST_Point , ST_Point , ST_Point) constructor

Constructs a circularstring value from a list of points in a specified spatial reference system.

Syntax

NEW `ST_CircularString`(ST_Point `pt1`, ST_Point `pt2`, ST_Point `pt3`, ST_Point `pti`)

Parameters

- **pt1 –** The first point of a segment.
- **pt2 –** Any point on the segment between the first and last point.
- **pt3 –** The last point of a segment.

- **pti –** Additional points defining further segments, each starting with the previous end point, passing through the first additional point and ending with the second additional point.

Returns

`ST_CircularString` Returns a circularstring constructed from the specified points.

Examples

- **Example 1 –** The following returns an error: at least three points must be specified.

```
SELECT NEW ST_CircularString( NEW ST_Point( 0, 0 ), NEW
ST_Point( 1, 1 ) )
```

The following example returns the result `CircularString (0 0, 1 1, 2 0)`.

```
SELECT NEW ST_CircularString( NEW ST_Point( 0, 0 ), NEW
ST_Point( 1, 1 ), NEW ST_Point(2,0) )
```

The following returns an error: the first segment takes three points, and subsequent segments take two points.

```
SELECT NEW ST_CircularString( NEW ST_Point( 0, 0 ), NEW
ST_Point( 1, 1 ), NEW ST_Point(2,0), NEW ST_Point(1,-1) )
```

The following example returns the result `CircularString (0 0, 1 1, 2 0, 1 -1, 0 0)`.

```
SELECT NEW ST_CircularString( NEW ST_Point( 0, 0 ), NEW
ST_Point( 1, 1 ), NEW ST_Point(2,0), NEW ST_Point(1,-1), NEW
ST_Point( 0, 0 ) )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_CircularString() constructor

Constructs a circularstring representing the empty set.

Syntax

NEW `ST_CircularString`()

Returns

`ST_CircularString` Returns an ST_CircularString value representing the empty set.

Examples

- **Example 1 –** The following returns 1, indicating the value is empty.

```
SELECT NEW ST_CircularString().ST_IsEmpty()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Standard feature

ST_CircularString(LONG BINARY[, INT]) constructor

Constructs a circularstring from Well Known Binary (WKB).

Syntax

NEW `ST_CircularString`(LONG BINARY `wkb`[, INT `srid`])

Parameters

- **wkb –** A string containing the binary representation of a circularstring. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_CircularString` Returns an ST_CircularString value constructed from the source string.

Examples

- **Example 1 –** The following returns CircularString (5 10, 10 12, 15 10).

```
SELECT NEW
ST_CircularString(0x01080000000300000000000000000144000000000000
0244000000000000024400000000000028400000000000002e40000000000000
2440)
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 7.3.2

ST_CircularString(LONG VARCHAR[, INT]) constructor

Constructs a circularstring from a text representation.

Syntax

NEW `ST_CircularString`(LONG VARCHAR `text_representation`[, INT `srid`])

Parameters

- **text_representation –** A string containing the text representation of a circularstring. The input can be in any supported text input format, including Well Known Text (WKT) or Extended Well Known Text (EWKT).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_CircularString` Returns an ST_CircularString value constructed from the source string.

Examples

- **Example 1 –** The following returns CircularString (5 10, 10 12, 15 10).

```
SELECT NEW ST_CircularString('CircularString (5 10, 10 12, 15
10)')
```

The following example shows a circularstring with two semi-circle segments.

```
SELECT NEW ST_CircularString('CircularString (0 4, 2.5 6.5, 5 4,
7.5 1.5, 10 4)') CS
```

Example of a circularstring with two semi-circle segments

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 7.3.2

ST_NumPoints() method

Returns the number of points defining the circularstring.

Syntax

`circularstring-expression.ST_NumPoints()`

Returns

`INT` Returns NULL if the circularstring value is empty, otherwise the number of points in the value.

Examples

- **Example 1 –** The following example returns the result `5`.

```
SELECT TREAT( Shape AS ST_CircularString ).ST_NumPoints()
FROM SpatialShapes WHERE ShapeID = 18
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 7.3.4

ST_PointN(INT) method

Returns the nth point in the circularstring.

Syntax

```
circularstring-expression.ST_PointN(INT n)
```

Parameters

- **n –** The position of the element to return, from 1 to circularstring-expression.ST_NumPoints().

Returns

`ST_Point` If the value of circular-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.

Examples

- **Example 1 –** The following example returns the result `Point (2 0)`.

```
SELECT TREAT( Shape AS ST_CircularString ).ST_PointN( 3 )
FROM SpatialShapes WHERE ShapeID = 18
```

The following example returns one row for each point in geom.

```
BEGIN
DECLARE geom ST_CircularString;
SET geom = NEW ST_CircularString( 'CircularString( 0 0, 1 1, 2
0 )' );
SELECT row_num, geom.ST_PointN( row_num )
FROM sa_rowgenerator( 1, geom.ST_NumPoints() )
ORDER BY row_num;
END
```

The example returns the following result set:

row_num	geom.ST_PointN(row_num)
1	Point (0 0)
2	Point (1 1)
3	Point (2 0)

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 7.3.5

ST_CompoundCurve type

A compound curve is a sequence of ST_Curve values such that adjacent curves are joined at their endpoints. The contributing curves are limited to ST_LineString and ST_CircularString. The start point of each curve after the first is coincident with the end point of the previous curve.

Syntax

`ST_CompoundCurve` type

Members

All members of the ST_CompoundCurve type, including all inherited members.

Members of ST_CompoundCurve:

- **ST_CompoundCurve(ST_Curve , ST_Curve) –** Constructs a compound curve from a list of curves.
- **ST_CompoundCurve() –** Constructs a compound curve representing the empty set.
- **ST_CompoundCurve(LONG BINARY[, INT]) –** Constructs a compound curve from Well Known Binary (WKB).
- **ST_CompoundCurve(LONG VARCHAR[, INT]) –** Constructs a compound curve from a text representation.
- **ST_CurveN(INT) –** Returns the nth curve in the compound curve.
- **ST_NumCurves() –** Returns the number of curves defining the compound curve.

Members of ST_Curve:

- **ST_CurveToLine() –** Returns the ST_LineString interpolation of an ST_Curve value.
- **ST_EndPoint() –** Returns an ST_Point value that is the end point of the ST_Curve value.
- **ST_IsClosed() –** Test if the ST_Curve value is closed. A curve is closed if the start and end points are coincident.
- **ST_IsRing() –** Tests if the ST_Curve value is a ring. A curve is a ring if it is closed and simple (no self intersections).
- **ST_Length(VARCHAR(128)) –** Returns the length measurement of the ST_Curve value. The result is measured in the units specified by the unit-name parameter.
- **ST_StartPoint() –** Returns an ST_Point value that is the start point of the ST_Curve value.

Members of ST_Geometry:

- **ST_Affine(DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a new geometry that is the result of applying the specified 3-D affine transformation.

- **ST_AsBinary(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsBitmap(INT, INT, ST_Point , ST_Point , VARCHAR(128)) –** Returns a LONG VARBIT that is a bitmap representing a geometry value.
- **ST_AsGeoJSON(VARCHAR(128)) –** Returns a string representing a geometry in JSON format.
- **ST_AsGML(VARCHAR(128)) –** Returns the GML representation of an ST_Geometry value.
- **ST_AsKML(VARCHAR(128)) –** Returns the KML representation of an ST_Geometry value.
- **ST_AsSVG(VARCHAR(128)) –** Returns an SVG figure representing a geometry value.
- **ST_AsSVGAggr(ST_Geometry , VARCHAR(128)) –** Returns a complete or partial SVG document which renders the geometries in a group.
- **ST_AsText(VARCHAR(128)) –** Returns the text representation of an ST_Geometry value.
- **ST_AsWKB(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsWKT(VARCHAR(128)) –** Returns the WKT representation of an ST_Geometry value.
- **ST_AsXML(VARCHAR(128)) –** Returns the XML representation of an ST_Geometry value.
- **ST_Boundary() –** Returns the boundary of the geometry value.
- **ST_Buffer(DOUBLE, VARCHAR(128)) –** 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.
- **ST_Contains(ST_Geometry) –** Tests if a geometry value spatially contains another geometry value.
- **ST_ContainsFilter(ST_Geometry) –** An inexpensive test if a geometry might contain another.
- **ST_ConvexHull() –** Returns the convex hull of the geometry value.
- **ST_ConvexHullAggr(ST_Geometry) –** Returns the convex hull for all of the geometries in a group
- **ST_CoordDim() –** Returns the number of coordinate dimensions stored with each point of the ST_Geometry value.
- **ST_CoveredBy(ST_Geometry) –** Tests if a geometry value is spatially covered by another geometry value.
- **ST_CoveredByFilter(ST_Geometry) –** An inexpensive test if a geometry might be covered by another.
- **ST_Covers(ST_Geometry) –** Tests if a geometry value spatially covers another geometry value.

- **ST_CoversFilter(ST_Geometry) –** An inexpensive test if a geometry might cover another.
- **ST_Crosses(ST_Geometry) –** Tests if a geometry value crosses another geometry value.
- **ST_Debug(VARCHAR(128)) –** Returns a LONG BINARY that is debug information for the object.
- **ST_Difference(ST_Geometry) –** Returns the geometry value that represents the point set difference of two geometries.
- **ST_Dimension() –** Returns the dimension of the ST_Geometry value. Points have dimension 0, lines have dimension 1, and surfaces have dimension 2. Any empty geometry has dimension -1.
- **ST_Disjoint(ST_Geometry) –** Test if a geometry value is spatially disjoint from another value.
- **ST_Distance(ST_Geometry , VARCHAR(128)) –** Returns the smallest distance between the geometry-expression and the specified geometry value.
- **ST_Distance_Spheroid(ST_Geometry , VARCHAR(128)) –** Calculates the linear distance between geometries on the surface of the Earth.
- **ST_Envelope() –** Returns the bounding rectangle for the geometry value.
- **ST_EnvelopeAggr(ST_Geometry) –** Returns the bounding rectangle for all of the geometries in a group
- **ST_Equals(ST_Geometry) –** Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.
- **ST_EqualsFilter(ST_Geometry) –** An inexpensive test if a geometry is equal to another.
- **ST_GeometryType() –** Returns the name of the type of the ST_Geometry value.
- **ST_GeometryTypeFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_GeomFromBinary(LONG BINARY, INT) –** Constructs a geometry from a binary string representation.
- **ST_GeomFromShape(LONG BINARY[, INT]) –** Parses a string containing an ESRI shape record and creates a geometry value of the appropriate type.
- **ST_GeomFromText(LONG VARCHAR, INT) –** Constructs a geometry from a character string representation.
- **ST_GeomFromWKB(LONG BINARY, INT) –** Parse a string containing a WKB or EWKB representation of a geometry and creates a geometry value of the appropriate type.
- **ST_GeomFromWKT(LONG VARCHAR, INT) –** Parses a string containing the WKT or EWKT representation of a geometry and create a geometry value of the appropriate type.
- **ST_Intersection(ST_Geometry) –** Returns the geometry value that represents the point set intersection of two geometries.

- **ST_IntersectionAggr(ST_Geometry) –** Returns the spatial intersection of all of the geometries in a group
- **ST_Intersects(ST_Geometry) –** Test if a geometry value spatially intersects another value.
- **ST_IntersectsFilter(ST_Geometry) –** An inexpensive test if the two geometries might intersect.
- **ST_IntersectsRect(ST_Point , ST_Point) –** Test if a geometry intersects a rectangle.
- **ST_Is3D() –** Determines if the geometry value has Z coordinate values.
- **ST_IsEmpty() –** Determines whether the geometry value represents an empty set.
- **ST_IsMeasured() –** Determines if the geometry value has associated measure values.
- **ST_IsSimple() –** Determines whether the geometry value is simple (containing no self intersections or other irregularities).
- **ST_IsValid() –** Determines whether the geometry is a valid spatial object.
- **ST_LatNorth() –** Retrieves the northernmost latitude of a geometry.
- **ST_LatSouth() –** Retrieves the southernmost latitude of a geometry.
- **ST_Length_Spheroid(VARCHAR(128)) –** Calculates the linear length of a curve/multicurve on the surface of the Earth.
- **ST_LinearHash() –** Returns a binary string that is a linear hash of the geometry.
- **ST_LinearUnHash(BINARY(32)[, INT]) –** Returns a geometry representing the index hash.
- **ST_LoadConfigurationData(VARCHAR(128)) –** Returns binary configuration data. For internal use only.
- **ST_LocateAlong(DOUBLE) –** Returns the subset of the geometry value that is associated with the given measure value.
- **ST_LocateBetween(DOUBLE, DOUBLE) –** Returns the subset of the geometry value that is between the specified start measure and end measure.
- **ST_LongEast() –** Retrieves the longitude of the eastern boundary of a geometry.
- **ST_LongWest() –** Retrieves the longitude of the western boundary of a geometry.
- **ST_MMax() –** Retrieves the maximum M coordinate value of a geometry.
- **ST_MMin() –** Retrieves the minimum M coordinate value of a geometry.
- **ST_OrderingEquals(ST_Geometry) –** Tests if a geometry is identical to another geometry.
- **ST_Overlaps(ST_Geometry) –** Tests if a geometry value overlaps another geometry value.
- **ST_Relate(ST_Geometry) –** Determines how a geometry value is spatially related to another geometry value by returning an intersection matrix. The ST_Relate method returns 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.
- **ST_Reverse() –** Returns the geometry with the element order reversed.

- **ST_Segmentize(DOUBLE) –** Add points so that no line segment is longer than a specified distance.
- **ST_Simplify(DOUBLE) –** Remove points from curves so long as the maximum introduced error is less than a specified tolerance.
- **ST_SnapToGrid(ST_Point , DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a copy of the geometry with all points snapped to the specified grid.
- **ST_SRID(INT) –** Changes the spatial reference system associated with the geometry without modifying any of the values.
- **ST_SRIDFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_SymDifference(ST_Geometry) –** Returns the geometry value that represents the point set symmetric difference of two geometries.
- **ST_ToCircular() –** Convert the geometry to a circularstring
- **ST_ToCompound() –** Converts the geometry to a compound curve.
- **ST_ToCurve() –** Converts the geometry to a curve.
- **ST_ToCurvePoly() –** Converts the geometry to a curve polygon.
- **ST_ToGeomColl() –** Converts the geometry to a geometry collection.
- **ST_ToLineString() –** Converts the geometry to a linestring.
- **ST_ToMultiCurve() –** Converts the geometry to a multicurve value.
- **ST_ToMultiLine() –** Converts the geometry to a multilinestring value.
- **ST_ToMultiPoint() –** Converts the geometry to a multi-point value.
- **ST_ToMultiPolygon() –** Converts the geometry to a multi-polygon value.
- **ST_ToMultiSurface() –** Converts the geometry to a multi-surface value.
- **ST_ToPoint() –** Converts the geometry to a point.
- **ST_ToPolygon() –** Converts the geometry to a polygon.
- **ST_ToSurface() –** Converts the geometry to a surface.
- **ST_Touches(ST_Geometry) –** Tests if a geometry value spatially touches another geometry value.
- **ST_Transform(INT) –** Creates a copy of the geometry value transformed into the specified spatial reference system.
- **ST_Union(ST_Geometry) –** Returns the geometry value that represents the point set union of two geometries.
- **ST_UnionAggr(ST_Geometry) –** Returns the spatial union of all of the geometries in a group
- **ST_Within(ST_Geometry) –** Tests if a geometry value is spatially contained within another geometry value.
- **ST_WithinDistance(ST_Geometry , DOUBLE, VARCHAR(128)) –** Test if two geometries are within a specified distance of each other.
- **ST_WithinDistanceFilter(ST_Geometry , DOUBLE, VARCHAR(128)) –** An inexpensive of whether two geometries might be within a specified distance of each other.

- **ST_WithinFilter(ST_Geometry) –** An inexpensive test if a geometry might be within another.
- **ST_XMax() –** Retrieves the maximum X coordinate value of a geometry.
- **ST_XMin() –** Retrieves the minimum X coordinate value of a geometry.
- **ST_YMax() –** Retrieves the maximum Y coordinate value of a geometry.
- **ST_YMin() –** Retrieves the minimum Y coordinate value of a geometry.
- **ST_ZMax() –** Retrieves the maximum Z coordinate value of a geometry.
- **ST_ZMin() –** Retrieves the minimum Z coordinate value of a geometry.

Standards and compatibility
SQL/MM (ISO/IEC 13249-3: 2006) 7.4

ST_CompoundCurve(ST_Curve , ST_Curve) constructor

Constructs a compound curve from a list of curves.

Syntax

NEW `ST_CompoundCurve`(ST_Curve `curve1`, ST_Curve `curvei`)

Parameters

- **curve1 –** The first curve to include in the compound curve.
- **curvei –** Additional curves to include in the compound curve.

Returns

`ST_CompoundCurve` Returns a compound curve containing the supplied curves.

Examples

- **Example 1 –** The following returns CompoundCurve ((0 0, 5 10), CircularString (5 10, 10 12, 15 10)).

```
SELECT NEW ST_CompoundCurve(NEW ST_LineString( 'LineString(0 0, 5
10)'),NEW ST_CircularString('CircularString (5 10, 10 12, 15
10)'))
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_CompoundCurve() constructor

Constructs a compound curve representing the empty set.

Syntax

NEW `ST_CompoundCurve`()

Returns

`ST_CompoundCurve` Returns an ST_CompoundCurve value representing the empty set.

Examples

- **Example 1 –** The following returns 1, indicating the value is empty.

```
SELECT NEW ST_CompoundCurve().ST_IsEmpty()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Standard feature

ST_CompoundCurve(LONG BINARY[, INT]) constructor

Constructs a compound curve from Well Known Binary (WKB).

Syntax

NEW `ST_CompoundCurve`(LONG BINARY `wkb`[, INT `srid`])

Parameters

- **wkb –** A string containing the binary representation of a compound curve. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_CompoundCurve` Returns an ST_CompoundCurve value constructed from the source string.

Examples

- **Example 1 –** The following returns CompoundCurve ((0 0, 5 10)).

```
SELECT NEW
ST_CompoundCurve(0x010900000001000000010200000002000000000000000000000000000000000000000000000000000000144000000000000024400)
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 7.4.2

ST_CompoundCurve(LONG VARCHAR[, INT]) constructor

Constructs a compound curve from a text representation.

Syntax

NEW `ST_CompoundCurve`(LONG VARCHAR `text_representation`[, INT `srid`])

Parameters

- **text_representation –** A string containing the text representation of a compound curve. The input can be in any supported text input format, including Well Known Text (WKT) or Extended Well Known Text (EWKT).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_CompoundCurve` Returns an ST_CompoundCurve value constructed from the source string.

Examples

- **Example 1 –** The following returns CompoundCurve ((0 0, 5 10), CircularString (5 10, 10 12, 15 10)).

```
SELECT NEW ST_CompoundCurve('CompoundCurve ((0 0, 5 10),
CircularString (5 10, 10 12, 15 10))')
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 7.4.2

ST_CurveN(INT) method

Returns the nth curve in the compound curve.

Syntax

`compoundcurve-expression.ST_CurveN`(INT `n`)

Parameters

- **n –** The position of the element to return, from 1 to compoundcurve-expression.ST_NumCurves().

Returns

`ST_Curve` Returns the nth curve in the compound curve.

Examples

- **Example 1 –** The following example returns the result `CircularString (0 0, 1 1, 2 0)`.

```
SELECT TREAT( Shape AS ST_CompoundCurve ).ST_CurveN( 1 )
FROM SpatialShapes WHERE ShapeID = 17
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 7.4.5

ST_NumCurves() method

Returns the number of curves defining the compound curve.

Syntax

`compoundcurve-expression.ST_NumCurves()`

Returns

`INT` Returns the number of curves contained in this compound curve.

Examples

- **Example 1 –** The following example returns the result `2`.

```
SELECT TREAT( Shape AS ST_CompoundCurve ).ST_NumCurves()
FROM SpatialShapes WHERE ShapeID = 17
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 7.4.4

ST_Curve type

The ST_Curve type is a supertype for types representing lines using a sequence of points.

Syntax

`ST_Curve` type

Members

All members of the ST_Curve type, including all inherited members.

Members of ST_Curve:

- **ST_CurveToLine() –** Returns the ST_LineString interpolation of an ST_Curve value.
- **ST_EndPoint() –** Returns an ST_Point value that is the end point of the ST_Curve value.

- **ST_IsClosed() –** Test if the ST_Curve value is closed. A curve is closed if the start and end points are coincident.
- **ST_IsRing() –** Tests if the ST_Curve value is a ring. A curve is a ring if it is closed and simple (no self intersections).
- **ST_Length(VARCHAR(128)) –** Returns the length measurement of the ST_Curve value. The result is measured in the units specified by the unit-name parameter.
- **ST_StartPoint() –** Returns an ST_Point value that is the start point of the ST_Curve value.

Members of ST_Geometry:

- **ST_Affine(DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a new geometry that is the result of applying the specified 3-D affine transformation.
- **ST_AsBinary(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsBitmap(INT, INT, ST_Point , ST_Point , VARCHAR(128)) –** Returns a LONG VARBIT that is a bitmap representing a geometry value.
- **ST_AsGeoJSON(VARCHAR(128)) –** Returns a string representing a geometry in JSON format.
- **ST_AsGML(VARCHAR(128)) –** Returns the GML representation of an ST_Geometry value.
- **ST_AsKML(VARCHAR(128)) –** Returns the KML representation of an ST_Geometry value.
- **ST_AsSVG(VARCHAR(128)) –** Returns an SVG figure representing a geometry value.
- **ST_AsSVGAggr(ST_Geometry , VARCHAR(128)) –** Returns a complete or partial SVG document which renders the geometries in a group.
- **ST_AsText(VARCHAR(128)) –** Returns the text representation of an ST_Geometry value.
- **ST_AsWKB(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsWKT(VARCHAR(128)) –** Returns the WKT representation of an ST_Geometry value.
- **ST_AsXML(VARCHAR(128)) –** Returns the XML representation of an ST_Geometry value.
- **ST_Boundary() –** Returns the boundary of the geometry value.
- **ST_Buffer(DOUBLE, VARCHAR(128)) –** 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.
- **ST_Contains(ST_Geometry) –** Tests if a geometry value spatially contains another geometry value.

- **ST_ContainsFilter(ST_Geometry) –** An inexpensive test if a geometry might contain another.
- **ST_ConvexHull() –** Returns the convex hull of the geometry value.
- **ST_ConvexHullAggr(ST_Geometry) –** Returns the convex hull for all of the geometries in a group
- **ST_CoordDim() –** Returns the number of coordinate dimensions stored with each point of the ST_Geometry value.
- **ST_CoveredBy(ST_Geometry) –** Tests if a geometry value is spatially covered by another geometry value.
- **ST_CoveredByFilter(ST_Geometry) –** An inexpensive test if a geometry might be covered by another.
- **ST_Covers(ST_Geometry) –** Tests if a geometry value spatially covers another geometry value.
- **ST_CoversFilter(ST_Geometry) –** An inexpensive test if a geometry might cover another.
- **ST_Crosses(ST_Geometry) –** Tests if a geometry value crosses another geometry value.
- **ST_Debug(VARCHAR(128)) –** Returns a LONG BINARY that is debug information for the object.
- **ST_Difference(ST_Geometry) –** Returns the geometry value that represents the point set difference of two geometries.
- **ST_Dimension() –** Returns the dimension of the ST_Geometry value. Points have dimension 0, lines have dimension 1, and surfaces have dimension 2. Any empty geometry has dimension -1.
- **ST_Disjoint(ST_Geometry) –** Test if a geometry value is spatially disjoint from another value.
- **ST_Distance(ST_Geometry , VARCHAR(128)) –** Returns the smallest distance between the geometry-expression and the specified geometry value.
- **ST_Distance_Spheroid(ST_Geometry , VARCHAR(128)) –** Calculates the linear distance between geometries on the surface of the Earth.
- **ST_Envelope() –** Returns the bounding rectangle for the geometry value.
- **ST_EnvelopeAggr(ST_Geometry) –** Returns the bounding rectangle for all of the geometries in a group
- **ST_Equals(ST_Geometry) –** Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.
- **ST_EqualsFilter(ST_Geometry) –** An inexpensive test if a geometry is equal to another.
- **ST_GeometryType() –** Returns the name of the type of the ST_Geometry value.
- **ST_GeometryTypeFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.

- **ST_GeomFromBinary(LONG BINARY, INT) –** Constructs a geometry from a binary string representation.
- **ST_GeomFromShape(LONG BINARY[, INT]) –** Parses a string containing an ESRI shape record and creates a geometry value of the appropriate type.
- **ST_GeomFromText(LONG VARCHAR, INT) –** Constructs a geometry from a character string representation.
- **ST_GeomFromWKB(LONG BINARY, INT) –** Parse a string containing a WKB or EWKB representation of a geometry and creates a geometry value of the appropriate type.
- **ST_GeomFromWKT(LONG VARCHAR, INT) –** Parses a string containing the WKT or EWKT representation of a geometry and create a geometry value of the appropriate type.
- **ST_Intersection(ST_Geometry) –** Returns the geometry value that represents the point set intersection of two geometries.
- **ST_IntersectionAggr(ST_Geometry) –** Returns the spatial intersection of all of the geometries in a group
- **ST_Intersects(ST_Geometry) –** Test if a geometry value spatially intersects another value.
- **ST_IntersectsFilter(ST_Geometry) –** An inexpensive test if the two geometries might intersect.
- **ST_IntersectsRect(ST_Point , ST_Point) –** Test if a geometry intersects a rectangle.
- **ST_Is3D() –** Determines if the geometry value has Z coordinate values.
- **ST_IsEmpty() –** Determines whether the geometry value represents an empty set.
- **ST_IsMeasured() –** Determines if the geometry value has associated measure values.
- **ST_IsSimple() –** Determines whether the geometry value is simple (containing no self intersections or other irregularities).
- **ST_IsValid() –** Determines whether the geometry is a valid spatial object.
- **ST_LatNorth() –** Retrieves the northernmost latitude of a geometry.
- **ST_LatSouth() –** Retrieves the southernmost latitude of a geometry.
- **ST_Length_Spheroid(VARCHAR(128)) –** Calculates the linear length of a curve/multicurve on the surface of the Earth.
- **ST_LinearHash() –** Returns a binary string that is a linear hash of the geometry.
- **ST_LinearUnHash(BINARY(32)[, INT]) –** Returns a geometry representing the index hash.
- **ST_LoadConfigurationData(VARCHAR(128)) –** Returns binary configuration data. For internal use only.
- **ST_LocateAlong(DOUBLE) –** Returns the subset of the geometry value that is associated with the given measure value.
- **ST_LocateBetween(DOUBLE, DOUBLE) –** Returns the subset of the geometry value that is between the specified start measure and end measure.
- **ST_LongEast() –** Retrieves the longitude of the eastern boundary of a geometry.
- **ST_LongWest() –** Retrieves the longitude of the western boundary of a geometry.

- **ST_MMax() –** Retrieves the maximum M coordinate value of a geometry.
- **ST_MMin() –** Retrieves the minimum M coordinate value of a geometry.
- **ST_OrderingEquals(ST_Geometry) –** Tests if a geometry is identical to another geometry.
- **ST_Overlaps(ST_Geometry) –** Tests if a geometry value overlaps another geometry value.
- **ST_Relate(ST_Geometry) –** Determines how a geometry value is spatially related to another geometry value by returning an intersection matrix. The ST_Relate method returns 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.
- **ST_Reverse() –** Returns the geometry with the element order reversed.
- **ST_Segmentize(DOUBLE) –** Add points so that no line segment is longer than a specified distance.
- **ST_Simplify(DOUBLE) –** Remove points from curves so long as the maximum introduced error is less than a specified tolerance.
- **ST_SnapToGrid(ST_Point , DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a copy of the geometry with all points snapped to the specified grid.
- **ST_SRID(INT) –** Changes the spatial reference system associated with the geometry without modifying any of the values.
- **ST_SRIDFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_SymDifference(ST_Geometry) –** Returns the geometry value that represents the point set symmetric difference of two geometries.
- **ST_ToCircular() –** Convert the geometry to a circularstring
- **ST_ToCompound() –** Converts the geometry to a compound curve.
- **ST_ToCurve() –** Converts the geometry to a curve.
- **ST_ToCurvePoly() –** Converts the geometry to a curve polygon.
- **ST_ToGeomColl() –** Converts the geometry to a geometry collection.
- **ST_ToLineString() –** Converts the geometry to a linestring.
- **ST_ToMultiCurve() –** Converts the geometry to a multicurve value.
- **ST_ToMultiLine() –** Converts the geometry to a multilinestring value.
- **ST_ToMultiPoint() –** Converts the geometry to a multi-point value.
- **ST_ToMultiPolygon() –** Converts the geometry to a multi-polygon value.
- **ST_ToMultiSurface() –** Converts the geometry to a multi-surface value.
- **ST_ToPoint() –** Converts the geometry to a point.
- **ST_ToPolygon() –** Converts the geometry to a polygon.
- **ST_ToSurface() –** Converts the geometry to a surface.
- **ST_Touches(ST_Geometry) –** Tests if a geometry value spatially touches another geometry value.

- **ST_Transform(INT) –** Creates a copy of the geometry value transformed into the specified spatial reference system.
- **ST_Union(ST_Geometry) –** Returns the geometry value that represents the point set union of two geometries.
- **ST_UnionAggr(ST_Geometry) –** Returns the spatial union of all of the geometries in a group
- **ST_Within(ST_Geometry) –** Tests if a geometry value is spatially contained within another geometry value.
- **ST_WithinDistance(ST_Geometry , DOUBLE, VARCHAR(128)) –** Test if two geometries are within a specified distance of each other.
- **ST_WithinDistanceFilter(ST_Geometry , DOUBLE, VARCHAR(128)) –** An inexpensive of whether two geometries might be within a specified distance of each other.
- **ST_WithinFilter(ST_Geometry) –** An inexpensive test if a geometry might be within another.
- **ST_XMax() –** Retrieves the maximum X coordinate value of a geometry.
- **ST_XMin() –** Retrieves the minimum X coordinate value of a geometry.
- **ST_YMax() –** Retrieves the maximum Y coordinate value of a geometry.
- **ST_YMin() –** Retrieves the minimum Y coordinate value of a geometry.
- **ST_ZMax() –** Retrieves the maximum Z coordinate value of a geometry.
- **ST_ZMin() –** Retrieves the minimum Z coordinate value of a geometry.

Remarks

The ST_Curve type is a supertype for types representing lines using a sequence of points. Subtypes specify whether the control points are joined using straight segments (ST_LineString), circular segments (ST_CircularString) or a combination (ST_CompoundCurve).The ST_Curve type is not instantiable.An ST_Curve value is simple if it does not intersect itself (except possibly at the end points). If an ST_Curve value does intersect at its endpoints, it is closed. An ST_Curve value that is both simple and closed is called a ring.

Standards and compatibility

SQL/MM (ISO/IEC 13249-3: 2006) 7.1

ST_CurveToLine() method

Returns the ST_LineString interpolation of an ST_Curve value.

Syntax

```
curve-expression.ST_CurveToLine()
```

Returns

`ST_LineString` Returns the ST_LineString interpolation of curve-expression.

Examples

- **Example 1 –** The following example returns the result `LineString (0 7, 0 4, 4 4)` (a copy of the original linestring).

```
SELECT TREAT( Shape AS ST_Curve ).ST_CurveToLine()
FROM SpatialShapes WHERE ShapeID = 5
```

  The following example returns the result `LineString (0 0, 5 10)` (the compound curve converted to an equivalent linestring).

```
SELECT NEW ST_CompoundCurve( 'CompoundCurve((0 0, 5
10))' ).ST_CurveToLine()
```

  The following returns an interpolated linestring which approximates the original circularstring.

```
SELECT TREAT( Shape AS ST_Curve ).ST_CurveToLine()
FROM SpatialShapes WHERE ShapeID = 19
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 7.1.7

ST_EndPoint() method

Returns an ST_Point value that is the end point of the ST_Curve value.

Syntax

`curve-expression.ST_EndPoint()`

Returns

`ST_Point` If the curve is an empty set, returns NULL. Otherwise, returns the end point of the curve.

Examples

- **Example 1 –** The following example returns the result `Point (5 10)`.

```
SELECT NEW ST_LineString( 'LineString(0 0, 5 5, 5
10)' ).ST_EndPoint()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 7.1.4

ST_IsClosed() method

Test if the ST_Curve value is closed. A curve is closed if the start and end points are coincident.

Syntax

```
curve-expression.ST_IsClosed()
```

Returns

`BIT` Returns 1 if the curve is closed (and non empty). Otherwise, returns 0.

Examples

- **Example 1 –** The following returns all rows in SpatialShapes containing closed curves. The IF expression is required to ensure the TREAT function is not executed if the Shape is not a subtype of ST_Curve. Without the IF expression the server may reorder the conditions in the WHERE clause, leading to an error.

  ```
  SELECT * FROM SpatialShapes
  WHERE IF Shape IS OF ( ST_Curve )
  AND TREAT( Shape AS ST_Curve ).ST_IsClosed() = 1 THEN 1 ENDIF = 1
  ```

  The following returns all rows in curve_table that have closed geometries. This example assumes the geometry column has type ST_Curve, ST_LineString, ST_CircularString or ST_CompoundCurve.

  ```
  SELECT * FROM curve_table WHERE geometry.ST_IsClosed() = 1
  ```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 7.1.5

ST_IsRing() method

Tests if the ST_Curve value is a ring. A curve is a ring if it is closed and simple (no self intersections).

Syntax

```
curve-expression.ST_IsRing()
```

Returns

`BIT` Returns 1 if the curve is a ring (and non empty). Otherwise, returns 0.

Examples

- **Example 1 –** The following returns all rows in SpatialShapes containing rings. The IF expression is required to ensure the TREAT function is not executed if the Shape is not a

subtype of ST_Curve. Without the IF expression the server may reorder the conditions in the WHERE clause, leading to an error.

```
SELECT * FROM SpatialShapes
WHERE IF Shape IS OF ( ST_Curve )
AND TREAT( Shape AS ST_Curve ).ST_IsRing() = 1 THEN 1 ENDIF = 1
```

The following returns all rows in curve_table that have geometries that are rings. This example assumes the geometry column has type ST_Curve, ST_LineString, ST_CircularString or ST_CompoundCurve.

```
SELECT * FROM curve_table WHERE geometry.ST_IsRing() = 1
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 7.1.6

ST_Length(VARCHAR(128)) method

Returns the length measurement of the ST_Curve value. The result is measured in the units specified by the unit-name parameter.

Syntax

`curve-expression.ST_Length`(VARCHAR(128) `unit_name`)

Parameters

- **unit_name –** The units in which the length 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` If the curve is an empty set, returns NULL. Otherwise, returns the length of the curve in the specified units.

Examples

- **Example 1 –** The following example returns the result `2`.

```
SELECT NEW ST_LineString( 'LineString(1 0, 1 1, 2
1)' ).ST_Length()
```

The following example creates a circularstring representing a half-circle and uses ST_Length to find the length of the geometry, returning the value PI.

```
SELECT NEW ST_CircularString( 'CircularString( 0 0, 1 1, 2
0 )' ).ST_Length()
```

The following example creates a linestring representing a path from Halifax, NS to Waterloo, ON, Canada and uses ST_Length to find the length of the path in metres, returning the result `1361967.76789`.

```
SELECT NEW ST_LineString( 'LineString( -63.573566 44.646244,
-80.522372 43.465187 )', 4326 )
.ST_Length()
```

The following returns the lengths of the curves in the SpatialShapes table. The lengths are returned in Cartesian units.

```
SELECT ShapeID, TREAT( Shape AS ST_Curve ).ST_Length()
FROM SpatialShapes WHERE Shape IS OF ( ST_Curve )
```

The following example creates a linestring and an example unit of measure (example_unit_halfmetre). The ST_Length method finds the length of the geometry in this unit of measure, returning the value 4.0.

```
BEGIN
DECLARE @curve ST_Curve;
CREATE SPATIAL UNIT OF MEASURE IF NOT EXISTS
"example_unit_halfmetre" TYPE LINEAR CONVERT USING .5;
SET @curve = NEW ST_LineString( 'LineString(1 0, 1 1, 2 1)' ) ;
SELECT @curve.ST_Length('example_unit_halfmetre');
END
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 7.1.2

ST_StartPoint() method

Returns an ST_Point value that is the start point of the ST_Curve value.

Syntax

`curve-expression.ST_StartPoint()`

Returns

`ST_Point` If the curve is an empty set, returns NULL. Otherwise, returns the start point of the curve.

Examples

- **Example 1 –** The following example returns the result `Point (0 0)`.

```
SELECT NEW ST_LineString( 'LineString(0 0, 5 5, 5
10)' ).ST_StartPoint()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 7.1.3

ST_CurvePolygon type

An ST_CurvePolygon represents a planar surface defined by one exterior ring and zero or more interior rings

Syntax

`ST_CurvePolygon` type

Members

All members of the ST_CurvePolygon type, including all inherited members.

Members of ST_CurvePolygon:

- **ST_CurvePolygon(ST_Curve , ST_Curve) –** Creates a curve polygon from a curve representing the exterior ring and a list of curves representing interior rings, all in a specified spatial reference system.
- **ST_CurvePolygon(ST_MultiCurve , VARCHAR(128)) –** Creates a curve polygon from a multi curve containing an exterior ring and an optional list of interior rings.
- **ST_CurvePolygon() –** Constructs a curve polygon representing the empty set.
- **ST_CurvePolygon(LONG BINARY[, INT]) –** Constructs a curve polygon from Well Known Binary (WKB).
- **ST_CurvePolygon(LONG VARCHAR[, INT]) –** Constructs a curve polygon from a text representation.
- **ST_CurvePolyToPoly() –** Returns the interpolation of the curve polygon as a polygon.
- **ST_ExteriorRing(ST_Curve) –** Changes the exterior ring of the curve polygon.
- **ST_InteriorRingN(INT) –** Returns the nth interior ring in the curve polygon.
- **ST_NumInteriorRing() –** Returns the number of interior rings in the curve polygon.

Members of ST_Geometry:

- **ST_Affine(DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a new geometry that is the result of applying the specified 3-D affine transformation.
- **ST_AsBinary(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsBitmap(INT, INT, ST_Point , ST_Point , VARCHAR(128)) –** Returns a LONG VARBIT that is a bitmap representing a geometry value.
- **ST_AsGeoJSON(VARCHAR(128)) –** Returns a string representing a geometry in JSON format.
- **ST_AsGML(VARCHAR(128)) –** Returns the GML representation of an ST_Geometry value.

- **ST_AsKML(VARCHAR(128)) –** Returns the KML representation of an ST_Geometry value.
- **ST_AsSVG(VARCHAR(128)) –** Returns an SVG figure representing a geometry value.
- **ST_AsSVGAggr(ST_Geometry , VARCHAR(128)) –** Returns a complete or partial SVG document which renders the geometries in a group.
- **ST_AsText(VARCHAR(128)) –** Returns the text representation of an ST_Geometry value.
- **ST_AsWKB(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsWKT(VARCHAR(128)) –** Returns the WKT representation of an ST_Geometry value.
- **ST_AsXML(VARCHAR(128)) –** Returns the XML representation of an ST_Geometry value.
- **ST_Boundary() –** Returns the boundary of the geometry value.
- **ST_Buffer(DOUBLE, VARCHAR(128)) –** 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.
- **ST_Contains(ST_Geometry) –** Tests if a geometry value spatially contains another geometry value.
- **ST_ContainsFilter(ST_Geometry) –** An inexpensive test if a geometry might contain another.
- **ST_ConvexHull() –** Returns the convex hull of the geometry value.
- **ST_ConvexHullAggr(ST_Geometry) –** Returns the convex hull for all of the geometries in a group
- **ST_CoordDim() –** Returns the number of coordinate dimensions stored with each point of the ST_Geometry value.
- **ST_CoveredBy(ST_Geometry) –** Tests if a geometry value is spatially covered by another geometry value.
- **ST_CoveredByFilter(ST_Geometry) –** An inexpensive test if a geometry might be covered by another.
- **ST_Covers(ST_Geometry) –** Tests if a geometry value spatially covers another geometry value.
- **ST_CoversFilter(ST_Geometry) –** An inexpensive test if a geometry might cover another.
- **ST_Crosses(ST_Geometry) –** Tests if a geometry value crosses another geometry value.
- **ST_Debug(VARCHAR(128)) –** Returns a LONG BINARY that is debug information for the object.
- **ST_Difference(ST_Geometry) –** Returns the geometry value that represents the point set difference of two geometries.

- **ST_Dimension() –** Returns the dimension of the ST_Geometry value. Points have dimension 0, lines have dimension 1, and surfaces have dimension 2. Any empty geometry has dimension -1.
- **ST_Disjoint(ST_Geometry) –** Test if a geometry value is spatially disjoint from another value.
- **ST_Distance(ST_Geometry , VARCHAR(128)) –** Returns the smallest distance between the geometry-expression and the specified geometry value.
- **ST_Distance_Spheroid(ST_Geometry , VARCHAR(128)) –** Calculates the linear distance between geometries on the surface of the Earth.
- **ST_Envelope() –** Returns the bounding rectangle for the geometry value.
- **ST_EnvelopeAggr(ST_Geometry) –** Returns the bounding rectangle for all of the geometries in a group
- **ST_Equals(ST_Geometry) –** Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.
- **ST_EqualsFilter(ST_Geometry) –** An inexpensive test if a geometry is equal to another.
- **ST_GeometryType() –** Returns the name of the type of the ST_Geometry value.
- **ST_GeometryTypeFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_GeomFromBinary(LONG BINARY, INT) –** Constructs a geometry from a binary string representation.
- **ST_GeomFromShape(LONG BINARY[, INT]) –** Parses a string containing an ESRI shape record and creates a geometry value of the appropriate type.
- **ST_GeomFromText(LONG VARCHAR, INT) –** Constructs a geometry from a character string representation.
- **ST_GeomFromWKB(LONG BINARY, INT) –** Parse a string containing a WKB or EWKB representation of a geometry and creates a geometry value of the appropriate type.
- **ST_GeomFromWKT(LONG VARCHAR, INT) –** Parses a string containing the WKT or EWKT representation of a geometry and create a geometry value of the appropriate type.
- **ST_Intersection(ST_Geometry) –** Returns the geometry value that represents the point set intersection of two geometries.
- **ST_IntersectionAggr(ST_Geometry) –** Returns the spatial intersection of all of the geometries in a group
- **ST_Intersects(ST_Geometry) –** Test if a geometry value spatially intersects another value.
- **ST_IntersectsFilter(ST_Geometry) –** An inexpensive test if the two geometries might intersect.
- **ST_IntersectsRect(ST_Point , ST_Point) –** Test if a geometry intersects a rectangle.
- **ST_Is3D() –** Determines if the geometry value has Z coordinate values.
- **ST_IsEmpty() –** Determines whether the geometry value represents an empty set.

- **ST_IsMeasured() –** Determines if the geometry value has associated measure values.
- **ST_IsSimple() –** Determines whether the geometry value is simple (containing no self intersections or other irregularities).
- **ST_IsValid() –** Determines whether the geometry is a valid spatial object.
- **ST_LatNorth() –** Retrieves the northernmost latitude of a geometry.
- **ST_LatSouth() –** Retrieves the southernmost latitude of a geometry.
- **ST_Length_Spheroid(VARCHAR(128)) –** Calculates the linear length of a curve/multicurve on the surface of the Earth.
- **ST_LinearHash() –** Returns a binary string that is a linear hash of the geometry.
- **ST_LinearUnHash(BINARY(32)[, INT]) –** Returns a geometry representing the index hash.
- **ST_LoadConfigurationData(VARCHAR(128)) –** Returns binary configuration data. For internal use only.
- **ST_LocateAlong(DOUBLE) –** Returns the subset of the geometry value that is associated with the given measure value.
- **ST_LocateBetween(DOUBLE, DOUBLE) –** Returns the subset of the geometry value that is between the specified start measure and end measure.
- **ST_LongEast() –** Retrieves the longitude of the eastern boundary of a geometry.
- **ST_LongWest() –** Retrieves the longitude of the western boundary of a geometry.
- **ST_MMax() –** Retrieves the maximum M coordinate value of a geometry.
- **ST_MMin() –** Retrieves the minimum M coordinate value of a geometry.
- **ST_OrderingEquals(ST_Geometry) –** Tests if a geometry is identical to another geometry.
- **ST_Overlaps(ST_Geometry) –** Tests if a geometry value overlaps another geometry value.
- **ST_Relate(ST_Geometry) –** Determines how a geometry value is spatially related to another geometry value by returning an intersection matrix. The ST_Relate method returns 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.
- **ST_Reverse() –** Returns the geometry with the element order reversed.
- **ST_Segmentize(DOUBLE) –** Add points so that no line segment is longer than a specified distance.
- **ST_Simplify(DOUBLE) –** Remove points from curves so long as the maximum introduced error is less than a specified tolerance.
- **ST_SnapToGrid(ST_Point , DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a copy of the geometry with all points snapped to the specified grid.
- **ST_SRID(INT) –** Changes the spatial reference system associated with the geometry without modifying any of the values.
- **ST_SRIDFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.

- **ST_SymDifference(ST_Geometry) –** Returns the geometry value that represents the point set symmetric difference of two geometries.
- **ST_ToCircular() –** Convert the geometry to a circularstring
- **ST_ToCompound() –** Converts the geometry to a compound curve.
- **ST_ToCurve() –** Converts the geometry to a curve.
- **ST_ToCurvePoly() –** Converts the geometry to a curve polygon.
- **ST_ToGeomColl() –** Converts the geometry to a geometry collection.
- **ST_ToLineString() –** Converts the geometry to a linestring.
- **ST_ToMultiCurve() –** Converts the geometry to a multicurve value.
- **ST_ToMultiLine() –** Converts the geometry to a multilinestring value.
- **ST_ToMultiPoint() –** Converts the geometry to a multi-point value.
- **ST_ToMultiPolygon() –** Converts the geometry to a multi-polygon value.
- **ST_ToMultiSurface() –** Converts the geometry to a multi-surface value.
- **ST_ToPoint() –** Converts the geometry to a point.
- **ST_ToPolygon() –** Converts the geometry to a polygon.
- **ST_ToSurface() –** Converts the geometry to a surface.
- **ST_Touches(ST_Geometry) –** Tests if a geometry value spatially touches another geometry value.
- **ST_Transform(INT) –** Creates a copy of the geometry value transformed into the specified spatial reference system.
- **ST_Union(ST_Geometry) –** Returns the geometry value that represents the point set union of two geometries.
- **ST_UnionAggr(ST_Geometry) –** Returns the spatial union of all of the geometries in a group
- **ST_Within(ST_Geometry) –** Tests if a geometry value is spatially contained within another geometry value.
- **ST_WithinDistance(ST_Geometry , DOUBLE, VARCHAR(128)) –** Test if two geometries are within a specified distance of each other.
- **ST_WithinDistanceFilter(ST_Geometry , DOUBLE, VARCHAR(128)) –** An inexpensive of whether two geometries might be within a specified distance of each other.
- **ST_WithinFilter(ST_Geometry) –** An inexpensive test if a geometry might be within another.
- **ST_XMax() –** Retrieves the maximum X coordinate value of a geometry.
- **ST_XMin() –** Retrieves the minimum X coordinate value of a geometry.
- **ST_YMax() –** Retrieves the maximum Y coordinate value of a geometry.
- **ST_YMin() –** Retrieves the minimum Y coordinate value of a geometry.
- **ST_ZMax() –** Retrieves the maximum Z coordinate value of a geometry.
- **ST_ZMin() –** Retrieves the minimum Z coordinate value of a geometry.

Members of ST_Surface:

- **ST_Area(VARCHAR(128)) –** Calculates the area of a surface in the specified units.
- **ST_Centroid() –** Returns the ST_Point value that is the mathematical centroid of the surface value.
- **ST_IsWorld() –** Test if the ST_Surface covers the entire space.
- **ST_Perimeter(VARCHAR(128)) –** Calculates the perimeter of a surface in the specified units.
- **ST_PointOnSurface() –** Returns an ST_Point value that is guaranteed to spatially intersect the ST_Surface value.

Remarks

An ST_CurvePolygon represents a planar surface defined by one exterior ring and zero or more interior rings that represent holes in the surface. The exterior and interior rings of an ST_CurvePolygon can be any ST_Curve value. For example, a circle is an ST_CurvePolygon with an ST_CircularString exterior ring representing the boundary. No two rings in an ST_CurvePolygon can intersect except possibly at a single point. Further, an ST_CurvePolygon cannot have cut lines, spikes, or punctures.The interior of every ST_CurvePolygon is a connected point set.

Standards and compatibility

SQL/MM (ISO/IEC 13249-3: 2006) 8.2

ST_CurvePolygon(ST_Curve , ST_Curve) constructor

Creates a curve polygon from a curve representing the exterior ring and a list of curves representing interior rings, all in a specified spatial reference system.

Syntax

NEW `ST_CurvePolygon`(ST_Curve `exterior_ring`, ST_Curve `interior_ringi`)

Parameters

- **exterior_ring –** The exterior ring of the curve polygon
- **interior_ringi –** Interior rings of the curve polygon

Returns

`ST_CurvePolygon` Returns a polygon from the specified exterior ring and interior rings.

Examples

- **Example 1 –** The following returns CurvePolygon ((-5 -1, 5 -1, 0 9, -5 -1), CircularString (-2 2, -2 4, 2 4, 2 2, -2 2)) (a triangle with a circular hole).

```
SELECT NEW ST_CurvePolygon(
NEW ST_LineString ('LineString (-5 -1, 5 -1, 0 9, -5 -1)'),
```

```
NEW ST_CircularString ('CircularString (-2 2, -2 4, 2 4, 2 2, -2
2)'))
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 8.2.2

ST_CurvePolygon(ST_MultiCurve , VARCHAR(128)) constructor

Creates a curve polygon from a multi curve containing an exterior ring and an optional list of interior rings.

Syntax

NEW `ST_CurvePolygon`(ST_MultiCurve `multi_curve`, VARCHAR(128) `polygon_format`)

Parameters

- **multi_curve –** A multicurve value containing an exterior ring and (optionally) a set of interior rings.
- **polygon_format –** A string with the polygon format to use when interpreting the provided curves. Valid formats are 'CounterClockwise', 'Clockwise', and 'EvenOdd'

Returns

`ST_CurvePolygon` Returns a curve polygon created from the rings in a multilinestrings.

Examples

- **Example 1 –** The following returns the result `CurvePolygon (CircularString (-2 0, 1 -3, 4 0, 1 3, -2 0), (0 0, 1 1, 2 0, 0 0))` (a circular curve polygon with a triangular hole).

```
SELECT NEW ST_CurvePolygon( NEW ST_MultiCurve(
'MultiCurve(CircularString( -2 0, 4 0, -2 0 ),(0 0, 2 0, 1 1, 0
0 ))' ) )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_CurvePolygon() constructor

Constructs a curve polygon representing the empty set.

Syntax

NEW `ST_CurvePolygon`()

Returns

`ST_CurvePolygon` Returns an ST_CurvePolygon value representing the empty set.

Examples

- **Example 1 –** The following returns 1, indicating the value is empty.

```
SELECT NEW ST_CurvePolygon().ST_IsEmpty()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Standard feature

ST_CurvePolygon(LONG BINARY[, INT]) constructor

Constructs a curve polygon from Well Known Binary (WKB).

Syntax

NEW `ST_CurvePolygon`(LONG BINARY `wkb`[, INT `srid`])

Parameters

- **wkb –** A string containing the binary representation of a curve polygon. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_CurvePolygon` Returns an ST_CurvePolygon value constructed from the source string.

Examples

- **Example 1 –** The following returns CurvePolygon (CircularString (0 0, 10 0, 10 10, 0 10, 0 0)).

```
SELECT NEW
ST_CurvePolygon(0x010a0000001000000010800000005000000000000000000
0000000000000000000000000000000024400000000000000000000000000000000
2440000000000000244000000000000000000000000000000244000000000000000
0000000000000000000)
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 8.2.2

ST_CurvePolygon(LONG VARCHAR[, INT]) constructor

Constructs a curve polygon from a text representation.

Syntax

NEW ST_CurvePolygon(LONG VARCHAR text_representation[, INT srid])

Parameters

- **text_representation –** A string containing the text representation of a curve polygon. The input can be in any supported text input format, including Well Known Text (WKT) or Extended Well Known Text (EWKT).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

ST_CurvePolygon Returns an ST_CurvePolygon value constructed from the source string.

Examples

- **Example 1 –** The following returns CurvePolygon (CompoundCurve (CircularString (-5 -5, 0 -5, 5 -5), (5 -5, 0 5, -5 -5))).

```
SELECT NEW ST_CurvePolygon('CurvePolygon (CompoundCurve
(CircularString (-5 -5, 0 -5, 5 -5), (5 -5, 0 5, -5 -5)))')
```

  The following example shows a curvepolygon with a circle as an outer ring and a triangle inner ring.

```
SELECT NEW ST_CurvePolygon('CurvePolygon ( CircularString (2 0, 5
3, 2 0), (3 1, 4 2, 5 1, 3 1)  )') cpoly
```

  Example of a curvepolygon with a circle for an exterior ring and a triangular interior ring

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 8.2.2

ST_CurvePolyToPoly() method

Returns the interpolation of the curve polygon as a polygon.

Syntax

curvepolygon-expression.ST_CurvePolyToPoly()

Returns

ST_Polygon Returns the interpolation of the curvepolygon-expression as a polygon.

Examples

- **Example 1 –** The following example returns the result `Polygon ((0 0, 2 0, 1 2, 0 0))` (a copy of the original polygon).

```
SELECT TREAT( Shape AS ST_Polygon ).ST_CurvePolyToPoly()
FROM SpatialShapes WHERE ShapeID = 16
```

  The following example returns the result `Polygon ((0 0, 5 0, 5 10, 0 0))` (the curve polygon converted to an equivalent polygon).

```
SELECT NEW ST_CurvePolygon( 'CurvePolygon(CompoundCurve((0 0, 5
10, 5 0, 0 0)))' )
.ST_CurvePolyToPoly()
```

  The following returns an interpolated polygon which approximates the original curve polygon.

```
SELECT TREAT( Shape AS ST_CurvePolygon ).ST_CurvePolyToPoly()
FROM SpatialShapes WHERE ShapeId = 24
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 8.2.7

ST_ExteriorRing(ST_Curve) method

Changes the exterior ring of the curve polygon.

Syntax

`curvepolygon-expression.ST_ExteriorRing`(ST_Curve `exterior_ring`)

Parameters

- **exterior_ring –** The new exterior ring value.

Returns

`ST_CurvePolygon` Returns a copy of the curve polygon value with the exterior ring modified to be the specified value.

Examples

- **Example 1 –** The following example returns the result `CurvePolygon (CircularString (2 0, 6 1, 5 5, 1 4, 2 0), (3 1, 4 2, 5 1, 3 1))`.

```
SELECT NEW ST_CurvePolygon('CurvePolygon ( CircularString (2 0, 5
3, 2 0), (3 1, 4 2, 5 1, 3 1) )')
.ST_ExteriorRing( NEW ST_CircularString( 'CircularString (2 0, 5
5, 2 0)' ) )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 8.2.3

ST_InteriorRingN(INT) method

Returns the nth interior ring in the curve polygon.

Syntax

```
curvepolygon-expression.ST_InteriorRingN(INT n)
```

Parameters

- **n –** The position of the element to return, from 1 to curvepolygon-expression.ST_NumInteriorRing().

Returns

`ST_Curve` Returns the nth interior ring in the curve polygon.

Examples

- **Example 1 –** The following example returns the result `LineString (3 1, 4 2, 5 1, 3 1)`.

```
SELECT NEW ST_CurvePolygon('CurvePolygon ( CircularString (2 0, 5
3, 2 0), (3 1, 4 2, 5 1, 3 1) )')
.ST_InteriorRingN( 1 )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 8.2.6

ST_NumInteriorRing() method

Returns the number of interior rings in the curve polygon.

Syntax

```
curvepolygon-expression.ST_NumInteriorRing()
```

Returns

`INT` Returns the number of interior rings in the curve polygon.

Examples

- **Example 1 –** The following example returns the result `1`.

```
SELECT NEW ST_CurvePolygon('CurvePolygon ( CircularString (2 0, 5
3, 2 0), (3 1, 4 2, 5 1, 3 1) )')
.ST_NumInteriorRing()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 8.2.5

ST_GeomCollection type

An ST_GeomCollection is a collection of zero or more ST_Geometry values.

Syntax

`ST_GeomCollection` type

Members

All members of the ST_GeomCollection type, including all inherited members.

Members of ST_GeomCollection:

- **ST_GeomCollection(ST_Geometry , ST_Geometry) –** Constructs a geometry collection from a list of geometry values.
- **ST_GeomCollection() –** Constructs a geometry collection representing the empty set.
- **ST_GeomCollection(LONG BINARY[, INT]) –** Constructs a geometry collection from Well Known Binary (WKB).
- **ST_GeomCollection(LONG VARCHAR[, INT]) –** Constructs a geometry collection from a text representation.
- **ST_GeomCollectionAggr(ST_Geometry) –** Returns a geometry collection containing all of the geometries in a group.
- **ST_GeometryN(INT) –** Returns the nth geometry in the geometry collection.
- **ST_NumGeometries() –** Returns the number of geometries contained in the geometry collection.

Members of ST_Geometry:

- **ST_Affine(DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a new geometry that is the result of applying the specified 3-D affine transformation.
- **ST_AsBinary(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsBitmap(INT, INT, ST_Point , ST_Point , VARCHAR(128)) –** Returns a LONG VARBIT that is a bitmap representing a geometry value.
- **ST_AsGeoJSON(VARCHAR(128)) –** Returns a string representing a geometry in JSON format.

- **ST_AsGML(VARCHAR(128)) –** Returns the GML representation of an ST_Geometry value.
- **ST_AsKML(VARCHAR(128)) –** Returns the KML representation of an ST_Geometry value.
- **ST_AsSVG(VARCHAR(128)) –** Returns an SVG figure representing a geometry value.
- **ST_AsSVGAggr(ST_Geometry , VARCHAR(128)) –** Returns a complete or partial SVG document which renders the geometries in a group.
- **ST_AsText(VARCHAR(128)) –** Returns the text representation of an ST_Geometry value.
- **ST_AsWKB(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsWKT(VARCHAR(128)) –** Returns the WKT representation of an ST_Geometry value.
- **ST_AsXML(VARCHAR(128)) –** Returns the XML representation of an ST_Geometry value.
- **ST_Boundary() –** Returns the boundary of the geometry value.
- **ST_Buffer(DOUBLE, VARCHAR(128)) –** 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.
- **ST_Contains(ST_Geometry) –** Tests if a geometry value spatially contains another geometry value.
- **ST_ContainsFilter(ST_Geometry) –** An inexpensive test if a geometry might contain another.
- **ST_ConvexHull() –** Returns the convex hull of the geometry value.
- **ST_ConvexHullAggr(ST_Geometry) –** Returns the convex hull for all of the geometries in a group
- **ST_CoordDim() –** Returns the number of coordinate dimensions stored with each point of the ST_Geometry value.
- **ST_CoveredBy(ST_Geometry) –** Tests if a geometry value is spatially covered by another geometry value.
- **ST_CoveredByFilter(ST_Geometry) –** An inexpensive test if a geometry might be covered by another.
- **ST_Covers(ST_Geometry) –** Tests if a geometry value spatially covers another geometry value.
- **ST_CoversFilter(ST_Geometry) –** An inexpensive test if a geometry might cover another.
- **ST_Crosses(ST_Geometry) –** Tests if a geometry value crosses another geometry value.
- **ST_Debug(VARCHAR(128)) –** Returns a LONG BINARY that is debug information for the object.

- **ST_Difference(ST_Geometry) –** Returns the geometry value that represents the point set difference of two geometries.
- **ST_Dimension() –** Returns the dimension of the ST_Geometry value. Points have dimension 0, lines have dimension 1, and surfaces have dimension 2. Any empty geometry has dimension -1.
- **ST_Disjoint(ST_Geometry) –** Test if a geometry value is spatially disjoint from another value.
- **ST_Distance(ST_Geometry , VARCHAR(128)) –** Returns the smallest distance between the geometry-expression and the specified geometry value.
- **ST_Distance_Spheroid(ST_Geometry , VARCHAR(128)) –** Calculates the linear distance between geometries on the surface of the Earth.
- **ST_Envelope() –** Returns the bounding rectangle for the geometry value.
- **ST_EnvelopeAggr(ST_Geometry) –** Returns the bounding rectangle for all of the geometries in a group
- **ST_Equals(ST_Geometry) –** Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.
- **ST_EqualsFilter(ST_Geometry) –** An inexpensive test if a geometry is equal to another.
- **ST_GeometryType() –** Returns the name of the type of the ST_Geometry value.
- **ST_GeometryTypeFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_GeomFromBinary(LONG BINARY, INT) –** Constructs a geometry from a binary string representation.
- **ST_GeomFromShape(LONG BINARY[, INT]) –** Parses a string containing an ESRI shape record and creates a geometry value of the appropriate type.
- **ST_GeomFromText(LONG VARCHAR, INT) –** Constructs a geometry from a character string representation.
- **ST_GeomFromWKB(LONG BINARY, INT) –** Parse a string containing a WKB or EWKB representation of a geometry and creates a geometry value of the appropriate type.
- **ST_GeomFromWKT(LONG VARCHAR, INT) –** Parses a string containing the WKT or EWKT representation of a geometry and create a geometry value of the appropriate type.
- **ST_Intersection(ST_Geometry) –** Returns the geometry value that represents the point set intersection of two geometries.
- **ST_IntersectionAggr(ST_Geometry) –** Returns the spatial intersection of all of the geometries in a group
- **ST_Intersects(ST_Geometry) –** Test if a geometry value spatially intersects another value.
- **ST_IntersectsFilter(ST_Geometry) –** An inexpensive test if the two geometries might intersect.
- **ST_IntersectsRect(ST_Point , ST_Point) –** Test if a geometry intersects a rectangle.

- **ST_Is3D() –** Determines if the geometry value has Z coordinate values.
- **ST_IsEmpty() –** Determines whether the geometry value represents an empty set.
- **ST_IsMeasured() –** Determines if the geometry value has associated measure values.
- **ST_IsSimple() –** Determines whether the geometry value is simple (containing no self intersections or other irregularities).
- **ST_IsValid() –** Determines whether the geometry is a valid spatial object.
- **ST_LatNorth() –** Retrieves the northernmost latitude of a geometry.
- **ST_LatSouth() –** Retrieves the southernmost latitude of a geometry.
- **ST_Length_Spheroid(VARCHAR(128)) –** Calculates the linear length of a curve/multicurve on the surface of the Earth.
- **ST_LinearHash() –** Returns a binary string that is a linear hash of the geometry.
- **ST_LinearUnHash(BINARY(32)[, INT]) –** Returns a geometry representing the index hash.
- **ST_LoadConfigurationData(VARCHAR(128)) –** Returns binary configuration data. For internal use only.
- **ST_LocateAlong(DOUBLE) –** Returns the subset of the geometry value that is associated with the given measure value.
- **ST_LocateBetween(DOUBLE, DOUBLE) –** Returns the subset of the geometry value that is between the specified start measure and end measure.
- **ST_LongEast() –** Retrieves the longitude of the eastern boundary of a geometry.
- **ST_LongWest() –** Retrieves the longitude of the western boundary of a geometry.
- **ST_MMax() –** Retrieves the maximum M coordinate value of a geometry.
- **ST_MMin() –** Retrieves the minimum M coordinate value of a geometry.
- **ST_OrderingEquals(ST_Geometry) –** Tests if a geometry is identical to another geometry.
- **ST_Overlaps(ST_Geometry) –** Tests if a geometry value overlaps another geometry value.
- **ST_Relate(ST_Geometry) –** Determines how a geometry value is spatially related to another geometry value by returning an intersection matrix. The ST_Relate method returns 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.
- **ST_Reverse() –** Returns the geometry with the element order reversed.
- **ST_Segmentize(DOUBLE) –** Add points so that no line segment is longer than a specified distance.
- **ST_Simplify(DOUBLE) –** Remove points from curves so long as the maximum introduced error is less than a specified tolerance.
- **ST_SnapToGrid(ST_Point , DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a copy of the geometry with all points snapped to the specified grid.

- **ST_SRID(INT) –** Changes the spatial reference system associated with the geometry without modifying any of the values.
- **ST_SRIDFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_SymDifference(ST_Geometry) –** Returns the geometry value that represents the point set symmetric difference of two geometries.
- **ST_ToCircular() –** Convert the geometry to a circularstring
- **ST_ToCompound() –** Converts the geometry to a compound curve.
- **ST_ToCurve() –** Converts the geometry to a curve.
- **ST_ToCurvePoly() –** Converts the geometry to a curve polygon.
- **ST_ToGeomColl() –** Converts the geometry to a geometry collection.
- **ST_ToLineString() –** Converts the geometry to a linestring.
- **ST_ToMultiCurve() –** Converts the geometry to a multicurve value.
- **ST_ToMultiLine() –** Converts the geometry to a multilinestring value.
- **ST_ToMultiPoint() –** Converts the geometry to a multi-point value.
- **ST_ToMultiPolygon() –** Converts the geometry to a multi-polygon value.
- **ST_ToMultiSurface() –** Converts the geometry to a multi-surface value.
- **ST_ToPoint() –** Converts the geometry to a point.
- **ST_ToPolygon() –** Converts the geometry to a polygon.
- **ST_ToSurface() –** Converts the geometry to a surface.
- **ST_Touches(ST_Geometry) –** Tests if a geometry value spatially touches another geometry value.
- **ST_Transform(INT) –** Creates a copy of the geometry value transformed into the specified spatial reference system.
- **ST_Union(ST_Geometry) –** Returns the geometry value that represents the point set union of two geometries.
- **ST_UnionAggr(ST_Geometry) –** Returns the spatial union of all of the geometries in a group
- **ST_Within(ST_Geometry) –** Tests if a geometry value is spatially contained within another geometry value.
- **ST_WithinDistance(ST_Geometry , DOUBLE, VARCHAR(128)) –** Test if two geometries are within a specified distance of each other.
- **ST_WithinDistanceFilter(ST_Geometry , DOUBLE, VARCHAR(128)) –** An inexpensive of whether two geometries might be within a specified distance of each other.
- **ST_WithinFilter(ST_Geometry) –** An inexpensive test if a geometry might be within another.
- **ST_XMax() –** Retrieves the maximum X coordinate value of a geometry.
- **ST_XMin() –** Retrieves the minimum X coordinate value of a geometry.
- **ST_YMax() –** Retrieves the maximum Y coordinate value of a geometry.
- **ST_YMin() –** Retrieves the minimum Y coordinate value of a geometry.
- **ST_ZMax() –** Retrieves the maximum Z coordinate value of a geometry.

- **ST_ZMin() –** Retrieves the minimum Z coordinate value of a geometry.

Remarks

An ST_GeomCollection 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_GeomCollection type can contain a heterogeneous collection of objects (for example, points, lines, and polygons). Sub-types of ST_GeomCollection 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.

Standards and compatibility

SQL/MM (ISO/IEC 13249-3: 2006) 9.1

ST_GeomCollection(ST_Geometry , ST_Geometry) constructor

Constructs a geometry collection from a list of geometry values.

Syntax

NEW `ST_GeomCollection`(ST_Geometry `geo1`, ST_Geometry `geoi`)

Parameters

- **geo1 –** The first geometry value of the geometry collection.
- **geoi –** Additional geometry values of the geometry collection.

Returns

`ST_GeomCollection` A geometry collection containing the provided geometry values.

Examples

- **Example 1 –** The following returns a geometry collection containing the single point 'Point (1 2)'

```
SELECT NEW ST_GeomCollection( NEW ST_Point( 1.0, 2.0 ) )
```

The following returns a geometry collection containing two points 'Point (1 2)' and 'Point (3 4)'

```
SELECT NEW ST_GeomCollection( NEW ST_Point( 1.0, 2.0 ), NEW
ST_Point( 3.0, 4.0 ) )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_GeomCollection() constructor

Constructs a geometry collection representing the empty set.

Syntax

NEW `ST_GeomCollection`()

Returns

`ST_GeomCollection` Returns an ST_GeomCollection value representing the empty set.

Examples

- **Example 1 –** The following returns 1, indicating the value is empty.

```
SELECT NEW ST_GeomCollection().ST_IsEmpty()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Standard feature

ST_GeomCollection(LONG BINARY[, INT]) constructor

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

Syntax

NEW `ST_GeomCollection`(LONG BINARY `wkb`[, INT `srid`])

Parameters

- **wkb –** A string containing the binary representation of a geometry collection. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_GeomCollection` Returns an ST_GeomCollection value constructed from the source string.

Examples

- **Example 1 –** The following returns GeometryCollection (Point (10 20)).

```
SELECT NEW
ST_GeomCollection(0x01070000000100000001010000000000000000244000000000000003440)
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 9.1.2

ST_GeomCollection(LONG VARCHAR[, INT]) constructor

Constructs a geometry collection from a text representation.

Syntax

NEW `ST_GeomCollection`(LONG VARCHAR `text_representation`[, INT `srid`])

Parameters

- **text_representation –** A string containing the text representation of a geometry collection. The input can be in any supported text input format, including Well Known Text (WKT) or Extended Well Known Text (EWKT).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_GeomCollection` Returns an ST_GeomCollection value constructed from the source string.

Examples

- **Example 1 –** The following returns GeometryCollection (CircularString (5 10, 10 12, 15 10), Polygon ((10 -5, 15 5, 5 5, 10 -5))).

```
SELECT NEW ST_GeomCollection('GeometryCollection (CircularString
(5 10, 10 12, 15 10), Polygon ((10 -5, 15 5, 5 5, 10 -5)))')
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 9.1.2

ST_GeomCollectionAggr(ST_Geometry) method

Returns a geometry collection containing all of the geometries in a group.

Syntax

`ST_GeomCollection`::`ST_GeomCollectionAggr`(ST_Geometry `geometry_column`)

Parameters

- **geometry_column –** The geometry values to generate the collection. Typically this is a column.

Returns

`ST_GeomCollection` Returns a geometry collection that contains all of the geometries in a group.

Examples

- **Example 1 –** The following example returns a single value which combines all geometries from the SpatialShapes table into a single collection.

```
SELECT ST_GeomCollection::ST_GeomCollectionAggr( Shape ) FROM
SpatialShapes
WHERE Shape.ST_Is3D() = 0
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_GeometryN(INT) method

Returns the nth geometry in the geometry collection.

Syntax

`geomcollection-expression.ST_GeometryN`(INT `n`)

Parameters

- **n –** The position of the element to return, from 1 to geomcollection-expression.ST_NumGeometries().

Returns

`ST_Geometry` Returns the nth geometry in the geometry collection.

Examples

- **Example 1 –** The following example returns the result `Polygon ((10 -5, 15 5, 5 5, 10 -5))`.

```
SELECT NEW ST_GeomCollection('GeometryCollection (CircularString
(5 10, 10 12, 15 10), Polygon ((10 -5, 15 5, 5 5, 10 -5)))')
.ST_GeometryN( 2 )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 9.1.5

ST_NumGeometries() method

Returns the number of geometries contained in the geometry collection.

Syntax

```
geomcollection-expression.ST_NumGeometries()
```

Returns

`INT` Returns the number of geometries stored in this collection.

Examples

- **Example 1 –** The following example returns the result `3`.

```
SELECT NEW ST_MultiPoint('MultiPoint ((10 10), (12 12), (14 10))')
.ST_NumGeometries()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 9.1.4

ST_Geometry type

The ST_Geometry type is the maximal supertype of the geometry type hierarchy.

Syntax

`ST_Geometry` type

Members

All members of the ST_Geometry type, including all inherited members.

Members of ST_Geometry:

- **ST_Affine(DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a new geometry that is the result of applying the specified 3-D affine transformation.
- **ST_AsBinary(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsBitmap(INT, INT, ST_Point , ST_Point , VARCHAR(128)) –** Returns a LONG VARBIT that is a bitmap representing a geometry value.
- **ST_AsGeoJSON(VARCHAR(128)) –** Returns a string representing a geometry in JSON format.
- **ST_AsGML(VARCHAR(128)) –** Returns the GML representation of an ST_Geometry value.

- **ST_AsKML(VARCHAR(128)) –** Returns the KML representation of an ST_Geometry value.
- **ST_AsSVG(VARCHAR(128)) –** Returns an SVG figure representing a geometry value.
- **ST_AsSVGAggr(ST_Geometry , VARCHAR(128)) –** Returns a complete or partial SVG document which renders the geometries in a group.
- **ST_AsText(VARCHAR(128)) –** Returns the text representation of an ST_Geometry value.
- **ST_AsWKB(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsWKT(VARCHAR(128)) –** Returns the WKT representation of an ST_Geometry value.
- **ST_AsXML(VARCHAR(128)) –** Returns the XML representation of an ST_Geometry value.
- **ST_Boundary() –** Returns the boundary of the geometry value.
- **ST_Buffer(DOUBLE, VARCHAR(128)) –** 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.
- **ST_Contains(ST_Geometry) –** Tests if a geometry value spatially contains another geometry value.
- **ST_ContainsFilter(ST_Geometry) –** An inexpensive test if a geometry might contain another.
- **ST_ConvexHull() –** Returns the convex hull of the geometry value.
- **ST_ConvexHullAggr(ST_Geometry) –** Returns the convex hull for all of the geometries in a group
- **ST_CoordDim() –** Returns the number of coordinate dimensions stored with each point of the ST_Geometry value.
- **ST_CoveredBy(ST_Geometry) –** Tests if a geometry value is spatially covered by another geometry value.
- **ST_CoveredByFilter(ST_Geometry) –** An inexpensive test if a geometry might be covered by another.
- **ST_Covers(ST_Geometry) –** Tests if a geometry value spatially covers another geometry value.
- **ST_CoversFilter(ST_Geometry) –** An inexpensive test if a geometry might cover another.
- **ST_Crosses(ST_Geometry) –** Tests if a geometry value crosses another geometry value.
- **ST_Debug(VARCHAR(128)) –** Returns a LONG BINARY that is debug information for the object.
- **ST_Difference(ST_Geometry) –** Returns the geometry value that represents the point set difference of two geometries.

- **ST_Dimension() –** Returns the dimension of the ST_Geometry value. Points have dimension 0, lines have dimension 1, and surfaces have dimension 2. Any empty geometry has dimension -1.
- **ST_Disjoint(ST_Geometry) –** Test if a geometry value is spatially disjoint from another value.
- **ST_Distance(ST_Geometry , VARCHAR(128)) –** Returns the smallest distance between the geometry-expression and the specified geometry value.
- **ST_Distance_Spheroid(ST_Geometry , VARCHAR(128)) –** Calculates the linear distance between geometries on the surface of the Earth.
- **ST_Envelope() –** Returns the bounding rectangle for the geometry value.
- **ST_EnvelopeAggr(ST_Geometry) –** Returns the bounding rectangle for all of the geometries in a group
- **ST_Equals(ST_Geometry) –** Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.
- **ST_EqualsFilter(ST_Geometry) –** An inexpensive test if a geometry is equal to another.
- **ST_GeometryType() –** Returns the name of the type of the ST_Geometry value.
- **ST_GeometryTypeFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_GeomFromBinary(LONG BINARY, INT) –** Constructs a geometry from a binary string representation.
- **ST_GeomFromShape(LONG BINARY[, INT]) –** Parses a string containing an ESRI shape record and creates a geometry value of the appropriate type.
- **ST_GeomFromText(LONG VARCHAR, INT) –** Constructs a geometry from a character string representation.
- **ST_GeomFromWKB(LONG BINARY, INT) –** Parse a string containing a WKB or EWKB representation of a geometry and creates a geometry value of the appropriate type.
- **ST_GeomFromWKT(LONG VARCHAR, INT) –** Parses a string containing the WKT or EWKT representation of a geometry and create a geometry value of the appropriate type.
- **ST_Intersection(ST_Geometry) –** Returns the geometry value that represents the point set intersection of two geometries.
- **ST_IntersectionAggr(ST_Geometry) –** Returns the spatial intersection of all of the geometries in a group
- **ST_Intersects(ST_Geometry) –** Test if a geometry value spatially intersects another value.
- **ST_IntersectsFilter(ST_Geometry) –** An inexpensive test if the two geometries might intersect.
- **ST_IntersectsRect(ST_Point , ST_Point) –** Test if a geometry intersects a rectangle.
- **ST_Is3D() –** Determines if the geometry value has Z coordinate values.
- **ST_IsEmpty() –** Determines whether the geometry value represents an empty set.

- **ST_IsMeasured() –** Determines if the geometry value has associated measure values.
- **ST_IsSimple() –** Determines whether the geometry value is simple (containing no self intersections or other irregularities).
- **ST_IsValid() –** Determines whether the geometry is a valid spatial object.
- **ST_LatNorth() –** Retrieves the northernmost latitude of a geometry.
- **ST_LatSouth() –** Retrieves the southernmost latitude of a geometry.
- **ST_Length_Spheroid(VARCHAR(128)) –** Calculates the linear length of a curve/multicurve on the surface of the Earth.
- **ST_LinearHash() –** Returns a binary string that is a linear hash of the geometry.
- **ST_LinearUnHash(BINARY(32)[, INT]) –** Returns a geometry representing the index hash.
- **ST_LoadConfigurationData(VARCHAR(128)) –** Returns binary configuration data. For internal use only.
- **ST_LocateAlong(DOUBLE) –** Returns the subset of the geometry value that is associated with the given measure value.
- **ST_LocateBetween(DOUBLE, DOUBLE) –** Returns the subset of the geometry value that is between the specified start measure and end measure.
- **ST_LongEast() –** Retrieves the longitude of the eastern boundary of a geometry.
- **ST_LongWest() –** Retrieves the longitude of the western boundary of a geometry.
- **ST_MMax() –** Retrieves the maximum M coordinate value of a geometry.
- **ST_MMin() –** Retrieves the minimum M coordinate value of a geometry.
- **ST_OrderingEquals(ST_Geometry) –** Tests if a geometry is identical to another geometry.
- **ST_Overlaps(ST_Geometry) –** Tests if a geometry value overlaps another geometry value.
- **ST_Relate(ST_Geometry) –** Determines how a geometry value is spatially related to another geometry value by returning an intersection matrix. The ST_Relate method returns 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.
- **ST_Reverse() –** Returns the geometry with the element order reversed.
- **ST_Segmentize(DOUBLE) –** Add points so that no line segment is longer than a specified distance.
- **ST_Simplify(DOUBLE) –** Remove points from curves so long as the maximum introduced error is less than a specified tolerance.
- **ST_SnapToGrid(ST_Point , DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a copy of the geometry with all points snapped to the specified grid.
- **ST_SRID(INT) –** Changes the spatial reference system associated with the geometry without modifying any of the values.
- **ST_SRIDFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.

- **ST_SymDifference(ST_Geometry) –** Returns the geometry value that represents the point set symmetric difference of two geometries.
- **ST_ToCircular() –** Convert the geometry to a circularstring
- **ST_ToCompound() –** Converts the geometry to a compound curve.
- **ST_ToCurve() –** Converts the geometry to a curve.
- **ST_ToCurvePoly() –** Converts the geometry to a curve polygon.
- **ST_ToGeomColl() –** Converts the geometry to a geometry collection.
- **ST_ToLineString() –** Converts the geometry to a linestring.
- **ST_ToMultiCurve() –** Converts the geometry to a multicurve value.
- **ST_ToMultiLine() –** Converts the geometry to a multilinestring value.
- **ST_ToMultiPoint() –** Converts the geometry to a multi-point value.
- **ST_ToMultiPolygon() –** Converts the geometry to a multi-polygon value.
- **ST_ToMultiSurface() –** Converts the geometry to a multi-surface value.
- **ST_ToPoint() –** Converts the geometry to a point.
- **ST_ToPolygon() –** Converts the geometry to a polygon.
- **ST_ToSurface() –** Converts the geometry to a surface.
- **ST_Touches(ST_Geometry) –** Tests if a geometry value spatially touches another geometry value.
- **ST_Transform(INT) –** Creates a copy of the geometry value transformed into the specified spatial reference system.
- **ST_Union(ST_Geometry) –** Returns the geometry value that represents the point set union of two geometries.
- **ST_UnionAggr(ST_Geometry) –** Returns the spatial union of all of the geometries in a group
- **ST_Within(ST_Geometry) –** Tests if a geometry value is spatially contained within another geometry value.
- **ST_WithinDistance(ST_Geometry , DOUBLE, VARCHAR(128)) –** Test if two geometries are within a specified distance of each other.
- **ST_WithinDistanceFilter(ST_Geometry , DOUBLE, VARCHAR(128)) –** An inexpensive of whether two geometries might be within a specified distance of each other.
- **ST_WithinFilter(ST_Geometry) –** An inexpensive test if a geometry might be within another.
- **ST_XMax() –** Retrieves the maximum X coordinate value of a geometry.
- **ST_XMin() –** Retrieves the minimum X coordinate value of a geometry.
- **ST_YMax() –** Retrieves the maximum Y coordinate value of a geometry.
- **ST_YMin() –** Retrieves the minimum Y coordinate value of a geometry.
- **ST_ZMax() –** Retrieves the maximum Z coordinate value of a geometry.
- **ST_ZMin() –** Retrieves the minimum Z coordinate value of a geometry.

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 or change 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.

Standards and compatibility
SQL/MM (ISO/IEC 13249-3: 2006) 5.1

ST_Affine(DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE) method

Returns a new geometry that is the result of applying the specified 3-D affine transformation.

Syntax

`geometry-expression.ST_Affine`(DOUBLE `a00`, DOUBLE `a01`, DOUBLE `a02`, DOUBLE `a10`, DOUBLE `a11`, DOUBLE `a12`, DOUBLE `a20`, DOUBLE `a21`, DOUBLE `a22`, DOUBLE `xoff`, DOUBLE `yoff`, DOUBLE `zoff`)

Parameters

- **a00 –** The affine matrix element in row 0, column 0
- **a01 –** The affine matrix element in row 0, column 1
- **a02 –** The affine matrix element in row 0, column 2
- **a10 –** The affine matrix element in row 1, column 0
- **a11 –** The affine matrix element in row 1, column 1
- **a12 –** The affine matrix element in row 1, column 2
- **a20 –** The affine matrix element in row 2, column 0
- **a21 –** The affine matrix element in row 2, column 1
- **a22 –** The affine matrix element in row 2, column 2
- **xoff –** The x offset for translation
- **yoff –** The y offset for translation
- **zoff –** The z offset for translation

Returns

`ST_Geometry` Returns a new geometry that is the result of the specified transformation.

Examples

- **Example 1 –** The following returns the result `LineString (5 6, 5 3, 9 3)`. The X values are translated by 5 and the Y values are translated by -1.

```
SELECT Shape.ST_Affine( 1,0,0, 0,1,0, 0,0,1, 5,-1,0 )
FROM SpatialShapes WHERE ShapeID = 5
```

  The following returns the result `LineString (.698833 6.965029, .399334 3.980017, 4.379351 3.580683)`. The Shape is rotated around the Z axis by 0.1 radians (about 5.7 degrees).

```
SELECT Shape.ST_Affine( cos(0.1),sin(0.1),0, -sin(0.1),cos(0.1),
0, 0,0,1, 0,0,0 )
FROM SpatialShapes WHERE ShapeID = 5
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_AsBinary(VARCHAR(128)) method

Returns the WKB representation of an ST_Geometry value.

Syntax

`geometry-expression.ST_AsBinary`(VARCHAR(128) `format`)

Parameters

- **format –** A string defining the output binary format to use when converting the geometry-expression to a binary representation. If not specified, the value of the st_geometry_asbinary_format option is used to choose the binary representation. See st_geometry_asbinary_format option.

Returns

`LONG BINARY` Returns the WKB representation of the geometry-expression.

Examples

- **Example 1 –** If the st_geometry_asbinary_format option has its default value of 'WKB', the following returns the result `0x01b90b0000000000000000f03f000000000000000400000000000000084000000000000001040`.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0, 4326 ).ST_AsBinary()
```

  If the st_geometry_asbinary_format option has its default value of 'WKB', the following returns the result `0x01b90b0000000000000000f03f000000000000000400000000000000008`

`400000000000001040`. The server implicitly invokes the ST_AsBinary method when converting geometries to BINARY.

```
SELECT CAST( NEW ST_Point( 1.0, 2.0, 3.0, 4.0, 4326 ) AS LONG
BINARY)
```

The following returns the result `0x0101000000000000000000f03f0000000000000040`. The Z and M values are omitted because version 1.1 of the OGC specification for WKB does not support these.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0,
4326 ).ST_AsBinary('WKB(Version=1.1;endian=little)')
```

The following returns the result `0x01010000e0e6100000000000000000f03f000000000000004000000000000008400000000000001040`. The extended WKB contains the SRID.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0,
4326 ).ST_AsBinary('EWKB(endian=little)')
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.37

ST_AsBitmap(INT, INT, ST_Point , ST_Point , VARCHAR(128)) method

Returns a LONG VARBIT that is a bitmap representing a geometry value.

Syntax

`geometry-expression.ST_AsBitmap`(INT `x_pixels`, INT `y_pixels`, ST_Point `pt_ll`, ST_Point `pt_ur`, VARCHAR(128) `format`)

Parameters

- **x_pixels –** The number of horizontal pixels to use
- **y_pixels –** The number of vertical pixels to use
- **pt_ll –** The lower left point of the bitmap
- **pt_ur –** The upper right point of the bitmap
- **format –** A string defining the parameters to use when converting the geometry-expression to a bitmap.

Returns

`LONG VARBIT` Returns a LONG VARBIT encoding a bitmap of the geometry.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_AsGeoJSON(VARCHAR(128)) method

Returns a string representing a geometry in JSON format.

Syntax

`geometry-expression.ST_AsGeoJSON(VARCHAR(128) format)`

Parameters

- **format –** A string defining parameters controlling how the GeoJSON result is generated. If not specified, the default is 'GeoJSON'.

Returns

`LONG VARCHAR` Returns the GeoJSON representation of the geometry-expression.

Examples

- **Example 1 –** The following example returns the result `{"type":"Point", "coordinates":[1,2]}`.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0, 4326 ).ST_AsGeoJSON()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_AsGML(VARCHAR(128)) method

Returns the GML representation of an ST_Geometry value.

Syntax

`geometry-expression.ST_AsGML(VARCHAR(128) format)`

Parameters

- **format –** A string defining the parameters to use when converting the geometry-expression to a GML representation. If not specified, the default is 'GML'.

Returns

`LONG VARCHAR` Returns the GML representation of the geometry-expression.

Examples

- **Example 1 –** The following example returns the result `<Point srsName="EPSG:4326"><pos>1 2 3 4</pos></Point>`.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0, 4326 ).ST_AsGML()
```

The following example returns the result `<Point srsName="EPSG:4326"><coordinates>1,2</coordinates></Point>`.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0,
4326 ).ST_AsGML('GML(Version=2)')
```

The following returns the result `<gml:Point srsName="EPSG:4326"><gml:coordinates>1,2</gml:coordinates></gml:Point>`. The Namespace=global parameter provides a dedicated ("gml") prefix for the given element and its sub elements. This is useful when the query is used within an aggregate operation, such that, some top level element defines the namespace for the "gml" prefix.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0,
4326 ).ST_AsGML('GML(Version=2;Namespace=global)')
```

The following returns the result `<Point srsName="EPSG:4326"><coordinates>1,2</coordinates></Point>`. No namespace information is included in the output.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0,
4326 ).ST_AsGML('GML(Version=2;Namespace=none)')
```

The following returns the result `<Point srsName="http://www.opengis.net/gml/srs/epsg.xml#4326"><coordinates>1,2</coordinates></Point>`. The long format of the srsName attribute is used.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0,
4326 ).ST_AsGML('GML(Version=2;Namespace=none;SRSNameFormat=long)
')
```

The following returns the result `<Point srsName="urn:x-ogc:def:crs:EPSG:4326"><pos>1 2 3 4</pos></Point>`. The long format of the srsName attribute is used and the format differs in version 3 from the version 2 format.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0,
4326 ).ST_AsGML('GML(Version=3;Namespace=none;SRSNameFormat=long)
')
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.39

ST_AsKML(VARCHAR(128)) method

Returns the KML representation of an ST_Geometry value.

Syntax

`geometry-expression.ST_AsKML`(VARCHAR(128) `format`)

Parameters

- **format –** A string defining the parameters to use when converting the geometry-expression to a KML representation. If not specified, the default is 'KML'.

Returns

`LONG VARCHAR` Returns the KML representation of the geometry-expression.

Examples

- **Example 1 –** The following example returns the result `<Point><coordinates>1,2,3,4</coordinates></Point>`.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0, 4326 ).ST_AsKML()
```

  The following example returns the result `<Point><coordinates>1,2,3,4</coordinates></Point>`.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0,
4326 ).ST_AsKML('KML(Version=2)')
```

  The following returns the result `<kml:Point><kml:coordinates>1,2,3,4</kml:coordinates></kml:Point>`. The Namespace=global parameter provides a dedicated ("kml") prefix for the given element and its sub elements. This is useful when the query is used within an aggregate operation, such that, some top level element defines the namespace for the "kml" prefix.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0,
4326 ).ST_AsKML('KML(Version=2;Namespace=global)')
```

  The following returns the result `<Point><coordinates>1,2,3,4</coordinates></Point>`. No namespace information is included in the output.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0,
4326 ).ST_AsKML('KML(Version=2;Namespace=none)')
```

  The following returns the result `<Point xmlns="http://www.opengis.net/kml/2.2"><coordinates>1,2,3,4</coordinates></Point>`. The default xml namespace is used.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0,
4326 ).ST_AsKML('KML(Version=2;Namespace=default)')
```

  The following returns the result `<Point><altitudeMode>absolute</altitudeMode><coordinates>1,2,3,4</coordinates></Point>`. An AltitudeMode sub element is included in the output.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0,
4326 ).ST_AsKML('SubElement=<altitudeMode>absolute</
altitudeMode>')
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.39

ST_AsSVG(VARCHAR(128)) method

Returns an SVG figure representing a geometry value.

Syntax

`geometry-expression.ST_AsSVG`(VARCHAR(128) `format`)

Parameters

- **format –** A string defining the parameters to use when converting the geometry-expression to a SVG representation. If not specified, the default is 'SVG'.

Returns

`LONG VARCHAR` Returns a complete or partial SVG document which renders the geometry-expression.

Examples

- **Example 1 –** 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()
```

The following returns a complete SVG document with outlined polygons and limits coordinates to 3 digits after the decimal place.

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

The following returns a complete SVG documents with polygons filled with blue and coordinates with maximum precision.

```
SELECT Shape.ST_AsSVG( 'Attribute=fill="blue";DecimalDigits=-1' )
FROM SpatialShapes
```

The following returns a complete SVG document from SVG path data with relative coordinates limited to 5 digits after the decimal place.

```
SELECT '<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg viewBox="-180 -90 360 180" xmlns="http://www.w3.org/2000/
svg"
version="1.1">
<path fill="lightblue" stroke="black" stroke-width="0.1%" d="' ||
NEW ST_Polygon( 'Polygon(( 0 0, 0 20, 60 10, 0 0 ))' )
.ST_AsSVG( 'PathDataOnly=Yes' ) ||
'"/></svg>'
```

The following returns SVG path data using absolute coordinates limited to 7 digits after the decimal place.

```
SELECT NEW ST_Polygon( 'Polygon(( 0 0, 0 20, 60 10, 0 0 ))' )
.ST_AsSVG( 'PathDataOnly=Yes;Relative=No;DecimalDigits=7' )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_AsSVGAggr(ST_Geometry , VARCHAR(128)) method

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

Syntax

`ST_Geometry::ST_AsSVGAggr(` ST_Geometry `geometry_column`, VARCHAR(128) `format`)

Parameters

- **geometry_column –** The geometry value to contribute to the SVG figure. Typically this is a column.
- **format –** A string defining the parameters to use when converting each geometry value to a SVG representation. If not specified, the default is 'SVG'.

Returns

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

Examples

- **Example 1 –** The following returns a complete SVG document with polygons filled with random colors.

```
SELECT ST_Geometry::ST_AsSVGAggr( Shape ) FROM SpatialShapes
```

The following returns a complete SVG document from SVG path data with relative coordinates limited to 5 digits after the decimal place.

```
SELECT '<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg viewBox="-10 -10 20 12" xmlns="http://www.w3.org/2000/svg"
version="1.1">
<path fill="lightblue" stroke="black" stroke-width="0.1%" d="' ||
ST_Geometry::ST_AsSVGAggr( Shape, 'PathDataOnly=Yes' ) ||
'"/></svg>'
FROM SpatialShapes
```

The following statements create a web service that returns a complete SVG document that renders all geometries in the SpatialShapes table. If the database server is started with the -xs http option, you can use a browser that supports SVG to display the SVG. To do this, browse to the address http://localhost/demo/svg_shapes This works assuming that the

browser and the database server are on the same computer, and that the database is named demo).

```
CREATE SERVICE svg_shapes TYPE 'RAW' USER DBA AUTHORIZATION OFF
AS CALL svg_shapes();CREATE PROCEDURE svg_shapes()
RESULT( svg LONG VARCHAR )
BEGIN
CALL sa_set_http_header( 'Content-type', 'image/svg+xml');
SELECT ST_Geometry::ST_AsSVGAggr( Shape ) FROM SpatialShapes;
END;
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_AsText(VARCHAR(128)) method

Returns the text representation of an ST_Geometry value.

Syntax

`geometry-expression.ST_AsText`(VARCHAR(128) `format`)

Parameters

- **format –** A string defining the output text format to use when converting the geometry-expression to a text representation. If not specified, the st_geometry_astext_format option is used to choose the text representation. Seest_geometry_astext_format option.

Returns

`LONG VARCHAR` Returns the text representation of the geometry-expression.

Examples

- **Example 1 –** Assuming that the st_geometry_astext_format option has the value 'WKT', the following returns the result `Point ZM (1 2 3 4)`.See st_geometry_astext_format option.

  ```
  SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0, 4326 ).ST_AsText()
  ```

  Assuming that the st_geometry_astext_format option has the value 'WKT', the following returns the result `Point ZM (1 2 3 4)`. The ST_AsText method is implicitly invoked when converting geometries to VARCHAR or NVARCHAR types.See st_geometry_astext_format option.

  ```
  SELECT CAST( NEW ST_Point( 1.0, 2.0, 3.0, 4.0, 4326 ) as long
  varchar)
  ```

  The following returns the result `Point (1 2)`. The Z and M values are not output because they are not supported in version 1.1.0 of the OGC specification for WKT.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0,
4326 ).ST_AsText('WKT(Version=1.1)')
```

The following returns the result `SRID=4326;Point ZM (1 2 3 4)`. The SRID is included in the result as a prefix.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0, 4326 ).ST_AsText('EWKT')
```

The following example returns the result `<Point srsName="EPSG:4326"><pos>1 2 3 4</pos></Point>`.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0, 4326 ).ST_AsText('GML')
```

The following returns `'{"type":"Point", "coordinates":[1,2]} '`.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0,
4326 ).ST_AsText('GeoJSON')
```

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_AsText( 'SVG' )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.35

ST_AsWKB(VARCHAR(128)) method

Returns the WKB representation of an ST_Geometry value.

Syntax

`geometry-expression.ST_AsWKB`(VARCHAR(128) `format`)

Parameters

- **format –** A string defining the WKB format to use when converting the geometry-expression to binary. If not specified, the default is 'WKB'.

Returns

`LONG BINARY` Returns the WKB representation of the geometry-expression.

Examples

- **Example 1 –** The following example returns the result `0x01b90b0000000000000000f03f0000000000000040000000000000084000000000000001040`.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0,
4326 ).ST_AsWKB('endian=little')
```

The following returns the result `0x0101000000000000000000f03f0000000000000040`. The Z and M values are omitted because version 1.1 of the OGC specification for WKB does not support these.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0,
4326 ).ST_AsWKB('WKB(Version=1.1;endian=little)')
```

The following returns the result `0x01010000e0e6100000000000000000f03f000000000000004000000000000008400000000000001040`.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0,
4326 ).ST_AsWKB('EWKB(endian=little)')
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_AsWKT(VARCHAR(128)) method

Returns the WKT representation of an ST_Geometry value.

Syntax

`geometry-expression.ST_AsWKT`(VARCHAR(128) `format`)

Parameters

- **format –** A string defining the output text format to use when converting the geometry-expression to WKT. If not specified, the format string defaults to 'WKT'.

Returns

`LONG VARCHAR` Returns the WKT representation of the geometry-expression.

Examples

- **Example 1 –** The following example returns the result `SRID=0;Polygon ((3 3, 8 3, 4 8, 3 3))`.

```
SELECT Shape.ST_AsWKT( 'EWKT' ) FROM SpatialShapes WHERE ShapeID =
22
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_AsXML(VARCHAR(128)) method

Returns the XML representation of an ST_Geometry value.

Syntax

`geometry-expression.ST_AsXML(VARCHAR(128) format)`

Parameters

- **format –** A string defining the output text format to use when converting the geometry-expression to an XML representation. If not specified, the st_geometry_asxml_format option is used to choose the XML representation. Seest_geometry_asxml_format option.

Returns

`LONG VARCHAR` Returns the XML representation of the geometry-expression.

Examples

- **Example 1 –** If the st_geometry_asxml_format option has its default value of 'GML', then the following returns the result `<Point srsName="EPSG:4326"><pos>1 2 3 4</pos></Point>`.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0, 4326 ).ST_AsXML()
```

  If the st_geometry_asxml_format option has its default value of 'GML', then the following returns the result `<Point srsName="EPSG:4326"><pos>1 2 3 4</pos></Point>`.

```
SELECT CAST( NEW ST_Point( 1.0, 2.0, 3.0, 4.0, 4326 ) AS XML)
```

  The following example returns the result `<Point srsName="EPSG:4326"><coordinates>1,2</coordinates></Point>`.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0,
4326 ).ST_AsXML('GML(Version=2)')
```

  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_AsXML( 'SVG' )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

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.

Examples

- **Example 1 –** The following example construct a geometry collection containing a polygon and a linestring and returns the boundary for the collection. The returned boundary is a collection containing the exterior ring of the polygon and the two end points of the linestring. It is equivalent to the following collection: `'GeometryCollection (LineString (0 0, 3 0, 3 3, 0 3, 0 0), MultiPoint ((0 7), (4 4)))'`

```
SELECT NEW ST_GeomCollection('GeometryCollection (Polygon ((0 0, 3 0, 3 3, 0 3, 0 0)), LineString (0 7, 0 4, 4 4))').ST_Boundary()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.14

ST_Buffer(DOUBLE, VARCHAR(128)) 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(DOUBLE distance, VARCHAR(128) unit_name)
```

Parameters

- **distance –** The distance the buffer should be from the geometry value.
- **unit_name –** 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.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.17

ST_Contains(ST_Geometry) method

Tests if a geometry value spatially contains another geometry value.

Syntax

`geometry-expression.ST_Contains(` ST_Geometry `geo2)`

Parameters

- **geo2 –** The other geometry value that is to be compared to the geometry-expression.

Returns

`BIT` Returns 1 if the geometry-expression contains geo2, otherwise 0.

Examples

- **Example 1 –** 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 ) )
```

  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 )' ) )
```

  The following example lists the ShapeIDs where the given polygon contains each Shape geometry. This example returns the result `16,17,19`. Note that ShapeID 1 is not listed because the polygon intersects that row's Shape point at the polygon's boundary.

```
SELECT LIST( ShapeID ORDER BY ShapeID )
FROM SpatialShapes
WHERE NEW ST_Polygon( NEW ST_Point( 0, 0 ),
NEW ST_Point( 8, 2 ) ).ST_Contains( Shape ) = 1
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.31

ST_ContainsFilter(ST_Geometry) method

An inexpensive test if a geometry might contain another.

Syntax

`geometry-expression.ST_ContainsFilter( ST_Geometry geo2)`

Parameters

- **geo2 –** The other geometry value that is to be compared to the geometry-expression.

Returns

`BIT` Returns 1 if the geometry-expression might contain geo2, otherwise 0.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

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.

Examples

- **Example 1 –** 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))`. Convex hull of a set of points

  ```
  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()
  ```

  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()
  ```

  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()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.16

ST_ConvexHullAggr(ST_Geometry) method

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

Syntax

`ST_Geometry::ST_ConvexHullAggr(` ST_Geometry `geometry_column)`

Parameters

- **geometry_column –** The geometry values to generate the convex hull. Typically this is a column.

Returns

`ST_Geometry` Returns the convex hull for all the geometries in a group.

Examples

- **Example 1 –** The following example returns the result `Polygon ((3 0, 7 2, 3 6, 0 7, -3 6, -3 3, 0 0, 3 0))`.

```
SELECT ST_Geometry::ST_ConvexHullAggr( Shape )
FROM SpatialShapes WHERE ShapeID <= 16
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

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.

Examples

- **Example 1 –** The following example returns the result 2.

```
SELECT NEW ST_Point(1.0, 1.0).ST_CoordDim()
```

The following example returns the result 3.

```
SELECT NEW ST_Point(1.0, 1.0, 1.0, 0).ST_CoordDim()
```

The following example returns the result 3.

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

The following example returns the result 4.

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

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.3

ST_CoveredBy(ST_Geometry) method

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

Syntax

geometry-expression.ST_CoveredBy(ST_Geometry geo2)

Parameters

- **geo2 –** The other geometry value that is to be compared to the geometry-expression.

Returns

BIT Returns 1 if the geometry-expression covers geo2, otherwise 0.

Examples

- **Example 1 –** 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 ))' ) )
```

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 ))' ) )
```

The following example lists the ShapeIDs where the given point is within the Shape geometry. This example returns the result `3,5,6`. Note that ShapeID 6 is listed even though the point intersects that row's Shape polygon only at the polygon's boundary.

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

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_CoveredByFilter(ST_Geometry) method

An inexpensive test if a geometry might be covered by another.

Syntax

`geometry-expression.ST_CoveredByFilter(` ST_Geometry `geo2)`

Parameters

- **geo2 –** The other geometry value that is to be compared to the geometry-expression.

Returns

`BIT` Returns 1 if the geometry-expression might be covered by geo2, otherwise 0.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_Covers(ST_Geometry) method

Tests if a geometry value spatially covers another geometry value.

Syntax

`geometry-expression.ST_Covers(` ST_Geometry `geo2)`

Parameters

- **geo2 –** The other geometry value that is to be compared to the geometry-expression.

Returns

`BIT` Returns 1 if the geometry-expression covers geo2, otherwise 0.

Examples

- **Example 1 –** 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 ) )
```

  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 )' ) )
```

  The following example lists the ShapeIDs where the given polygon covers each Shape geometry. This example returns the result `1,16,17,19,26`. Note that ShapeID 1 is listed even though the polygon intersects that row's Shape point only at the polygon's boundary.

```
SELECT LIST( ShapeID ORDER BY ShapeID )
FROM SpatialShapes
WHERE NEW ST_Polygon( NEW ST_Point( 0, 0 ),
NEW ST_Point( 8, 2 ) ).ST_Covers( Shape ) = 1
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_CoversFilter(ST_Geometry) method

An inexpensive test if a geometry might cover another.

Syntax

`geometry-expression.ST_CoversFilter( ST_Geometry geo2)`

Parameters

- **geo2 –** The other geometry value that is to be compared to the geometry-expression.

Returns

`BIT` Returns 1 if the geometry-expression might cover geo2, otherwise 0.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_Crosses(ST_Geometry) method

Tests if a geometry value crosses another geometry value.

Syntax

```
geometry-expression.ST_Crosses( ST_Geometry geo2)
```

Parameters

- **geo2 –** The other geometry value that is to be compared to the geometry-expression.

Returns

`BIT` Returns 1 if the geometry-expression crosses geo2, otherwise 0. Returns NULL if geometry-expression is a surface or multisurface, or if geo2 is a point or multipoint.

Examples

- **Example 1 –** 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 )' ) )
```

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 )' ) )
```

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

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

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.29

ST_Debug(VARCHAR(128)) method

Returns a LONG BINARY that is debug information for the object.

Syntax

```
geometry-expression.ST_Debug(VARCHAR(128) format)
```

Parameters

- **format –** The type of debug information and parameters as key=value pairs.

Returns

`LONG BINARY` Returns a LONG BINARY encoding debug information about the geometry.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_Difference(ST_Geometry) method

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

Syntax

`geometry-expression.ST_Difference(` ST_Geometry `geo2)`

Parameters

- **geo2 –** 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.

Examples

- **Example 1 –** The following example shows the difference (C) of a square (A) with a circle (B) removed and the difference (D) of a circle (B) with a square (A) removed.

```
SELECT NEW ST_Polygon( 'Polygon( (-1 -0.25, 1 -0.25, 1 2.25, -1
2.25, -1 -0.25) )' ) AS A
, NEW ST_CurvePolygon( 'CurvePolygon( CircularString( 0 1, 1 2, 2
1, 1 0, 0 1 ) )' )  AS B
, A.ST_Difference( B ) AS C
, B.ST_Difference( A ) AS D
```

  The following picture shows the difference C=A-B and D=B-A as the shaded portion of the picture. Each difference is a single surface that contains all of the points that are in the geometry on the left hand side of the difference and not in the geometry on the right hand side. Union of a square and a circle

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.20

ST_Dimension() method

Returns the dimension of the ST_Geometry value. Points have dimension 0, lines have dimension 1, and surfaces 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.

Examples

- **Example 1 –** The following example returns the result `0`.

```
SELECT NEW ST_Point(1.0,1.0).ST_Dimension()
```

The following example returns the result `1`.

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

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.2

ST_Disjoint(ST_Geometry) method

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

Syntax

```
geometry-expression.ST_Disjoint( ST_Geometry geo2)
```

Parameters

- **geo2 –** The other geometry value that is to be compared to the geometry-expression.

Returns

`BIT` Returns 1 if the geometry-expression is spatially disjoint from geo2, otherwise 0.

Examples

- **Example 1 –** The following example returns a result with one row for each shape that has no points in common with the specified triangle.

```
SELECT ShapeID, "Description"
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 result set:

ShapeID	Description
1	Point
22	Triangle

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.26

ST_Distance(ST_Geometry , VARCHAR(128)) method

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

Syntax

geometry-expression.ST_Distance(ST_Geometry geo2, VARCHAR(128) unit_name)

Parameters

- **geo2 –** The other geometry value whose distance is to be measured from the geometry-expression.
- **unit_name –** The units in which the distance 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 smallest distance between the geometry-expression and geo2 in the specified linear units of measure. If either geometry-expression or geo2 is empty, then NULL is returned.

Examples

- **Example 1 –** The following example returns an ordered result set with one row for each shape and the corresponding distance from the point (2,3).

```
SELECT ShapeID, ROUND( Shape.ST_Distance( NEW ST_Point( 2, 3 ) ),
2 ) AS dist
FROM SpatialShapes
WHERE ShapeID < 17
ORDER BY dist
```

The example returns the following result set:

ShapeID	dist
2	0.0
3	0.0
5	1.0
6	1.21
16	1.41
1	5.1

The following example creates points representing Halifax, NS and Waterloo, ON, Canada and uses ST_Distance to find the distance between the two points in miles, returning the result 846. This example assumes that the 'st_geometry_predefined_uom' feature has been installed by the sa_install_feature system procedure. Seesa_install_feature system procedure.

```
SELECT ROUND( NEW ST_Point( -63.573566, 44.646244, 4326 )
.ST_Distance( NEW ST_Point( -80.522372, 43.465187, 4326 )
, 'Statute mile' ), 0 )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.23

ST_Distance_Spheroid(ST_Geometry , VARCHAR(128)) method

Calculates the linear distance between geometries on the surface of the Earth.

Syntax

geometry-expression.ST_Distance_Spheroid(ST_Geometry geo2, VARCHAR(128) unit_name)

Parameters

- **geo2 –** The other geometry value whose distance is to be measured from the geometry-expression.
- **unit_name –** The linear unit of measure. Defaults to the unit of the spatial reference system.

Returns

DOUBLE Returns the linear distance between geometries on the surface of the Earth calculated in the specified linear units.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

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.

Examples

- **Example 1 –** The following example returns the result `Polygon ((0 0, 1 0, 1 4, 0 4, 0 0))`.

```
SELECT Shape.ST_Envelope()
FROM SpatialShapes WHERE ShapeID = 6
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.15

ST_EnvelopeAggr(ST_Geometry) method

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

Syntax

`ST_Geometry::ST_EnvelopeAggr(` ST_Geometry `geometry_column)`

Parameters

- **geometry_column –** The geometry values to generate the bounding rectangle. Typically this is a column.

Returns

`ST_Polygon` Returns a polygon that is the bounding rectangle for all the geometries in a group.

Examples

- **Example 1 –** The following example returns the result `Polygon ((-3 -1, 8 -1, 8 8, -3 8, -3 -1))`.

```
SELECT ST_Geometry::ST_EnvelopeAggr( Shape ) FROM SpatialShapes
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_Equals(ST_Geometry) method

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

Syntax

`geometry-expression.ST_Equals( ST_Geometry geo2)`

Parameters

- **geo2 –** The other geometry value that is to be compared to the geometry-expression.

Returns

`BIT` Returns 1 if the two geometry values are spatially equal, otherwise 0.

Examples

- **Example 1 –** The following example returns the result `16`. The Shape corresponding to ShapeID the result `16` contains the same points but in a different order as the specified polygon.

  ```
  SELECT ShapeID FROM SpatialShapes
  WHERE Shape.ST_Equals( NEW ST_Polygon( 'Polygon ((2 0, 1 2, 0 0, 2
  0))' ) ) = 1
  ```

  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 )' ) )
  ```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.24

ST_EqualsFilter(ST_Geometry) method

An inexpensive test if a geometry is equal to another.

Syntax

`geometry-expression.ST_EqualsFilter( ST_Geometry geo2)`

Parameters

- **geo2 –** The other geometry value that is to be compared to geometry-expression.

Returns

`BIT` Returns 1 if the bounding box for geometry-expression is equal, within tolerance, to the bounding box for geo2, otherwise 0.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_GeometryType() method

Returns the name of the type of the ST_Geometry value.

Syntax

`geometry-expression.ST_GeometryType()`

Returns

`VARCHAR_0123128321_` 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.

Examples

- **Example 1 –** The following returns the result `2,3,6,16,22,24,25`, which is the list of ShapeIDs whose corresponding Shape is one of the specified types.

```
SELECT LIST( ShapeID ORDER BY ShapeID )
FROM SpatialShapes
WHERE Shape.ST_GeometryType() IN( 'ST_Polygon',
'ST_CurvePolygon' )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.4

ST_GeometryTypeFromBaseType(VARCHAR(128)) method

Parses a string defining the type string.

Syntax

`ST_Geometry::ST_GeometryTypeFromBaseType(VARCHAR(128) base_type_str)`

Parameters

- **base_type_str –** A string containing the base type string

Returns

VARCHAR_0123128321_ Returns the geometry type from a base type string (which may include a SRID definition). If the type string is not a valid geometry type string, an error is returned.

Examples

- **Example 1 –** The following example returns the result ST_Geometry.

```
SELECT ST_Geometry::ST_GeometryTypeFromBaseType('ST_Geometry')
```

The following example returns the result ST_Point.

```
SELECT
ST_Geometry::ST_GeometryTypeFromBaseType('ST_Point(SRID=4326)')
```

The following example finds the geometry type (ST_Point) accepted by a stored procedure parameter.

```
CREATE PROCEDURE myprocedure( parm1 ST_Point(SRID=0) )
BEGIN
-- ...
END;SELECT    parm_name nm, base_type_str,
ST_Geometry::ST_GeometryTypeFromBaseType(base_type_str) geom_type
FROM    sysprocedure KEY JOIN sysprocparm
WHERE    proc_name='myprocedure' and parm_name='parm1'
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_GeomFromBinary(LONG BINARY, INT) method

Constructs a geometry from a binary string representation.

Syntax

ST_Geometry::ST_GeomFromBinary(LONG BINARY binary_string, INT srid)

Parameters

- **binary_string –** A string containing the binary representation of a geometry. The input can be in any supported binary format, including WKB or EWKB.
- **srid –** The SRID of the result. If not specified and the input string does not provide a SRID, the default is 0.

Returns

`ST_Geometry` Returns a geometry value of the appropriate type based on the source string.

Examples

- **Example 1 –** The following example returns the result `Point (10 20)`.

```
SELECT
ST_Geometry::ST_GeomFromBinary( 0x01010000000000000000002440000000
00000003440 )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_GeomFromShape(LONG BINARY[, INT]) method

Parses a string containing an ESRI shape record and creates a geometry value of the appropriate type.

Syntax

`ST_Geometry`::`ST_GeomFromShape`(LONG BINARY `shape`[, INT `srid`])

Parameters

- **shape –** A string containing a geometry in the ESRI shape format.
- **srid –** 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.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_GeomFromText(LONG VARCHAR, INT) method

Constructs a geometry from a character string representation.

Syntax

`ST_Geometry`::`ST_GeomFromText`(LONG VARCHAR `character_string`, INT `srid`)

Parameters

- **character_string –** A string containing the text representation of a geometry. The input can be in any supported text input format, including Well Known Text (WKT) or Extended Well Known Text (EWKT).
- **srid –** 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.

Examples

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

```
SELECT ST_Geometry::ST_GeomFromText( 'LineString( 1 2, 5 7 )',
4326 )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.40

ST_GeomFromWKB(LONG BINARY, INT) method

Parse a string containing a WKB or EWKB representation of a geometry and creates a geometry value of the appropriate type.

Syntax

`ST_Geometry`::`ST_GeomFromWKB`(LONG BINARY `wkb`, INT `srid`)

Parameters

- **wkb –** A string containing the WKB or EWKB representation of a geometry value.
- **srid –** 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.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.41

ST_GeomFromWKT(LONG VARCHAR, INT) method

Parses a string containing the WKT or EWKT representation of a geometry and create a geometry value of the appropriate type.

Syntax

`ST_Geometry::ST_GeomFromWKT`(LONG VARCHAR `wkt`, INT `srid`)

Parameters

- **wkt –** A string containing the WKT or EWKT representation of a geometry value.
- **srid –** 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.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_Intersection(ST_Geometry) method

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

Syntax

`geometry-expression.ST_Intersection`(ST_Geometry `geo2`)

Parameters

- **geo2 –** 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.

Examples

- **Example 1 –** The following example shows the intersection (C) of a square (A) and a circle (B).

```
SELECT NEW ST_Polygon( 'Polygon( (-1 -0.25, 1 -0.25, 1 2.25, -1
2.25, -1 -0.25) )' ) AS A
, NEW ST_CurvePolygon( 'CurvePolygon( CircularString( 0 1, 1 2, 2
1, 1 0, 0 1 ) )' )  AS B
, A.ST_Intersection( B ) AS C
```

The intersection is shaded in the following picture. It is a single surface that includes all of the points that are in the square and also in the circle. Union of a square and a circle

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.18

ST_IntersectionAggr(ST_Geometry) method

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

Syntax

`ST_Geometry::ST_IntersectionAggr(` ST_Geometry `geometry_column)`

Parameters

- **geometry_column –** The geometry values to generate the spatial intersection. Typically this is a column.

Returns

`ST_Geometry` Returns a geometry that is the spatial intersection for all the geometries in a group.

Examples

- **Example 1 –** The following example returns the result `Polygon ((0 0, 1 2, .5 2, .75 3, .555555 3, 0 1.75, .5 1.75, 0 0))`.

```
SELECT ST_Geometry::ST_IntersectionAggr( Shape )
FROM SpatialShapes WHERE ShapeID IN ( 2, 6 )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_Intersects(ST_Geometry) method

Test if a geometry value spatially intersects another value.

Syntax

`geometry-expression.ST_Intersects(` ST_Geometry `geo2)`

Parameters

- **geo2 –** The other geometry value that is to be compared to the geometry-expression.

Returns

`BIT` Returns 1 if the geometry-expression spatially intersects with geo2, otherwise 0.

Examples

- **Example 1 –** The following example returns a result with one row for each shape that intersects the specified line.

```
SELECT ShapeID, "Description"
FROM SpatialShapes
WHERE NEW ST_LineString( 'LineString( 2 2, 4
4 )' ).ST_Intersects( Shape ) = 1
ORDER BY ShapeID
```

The example returns the following result set:

ShapeID	Description
2	Square
3	Rectangle
5	L shape line
18	CircularString
22	Triangle

To visualize how the geometries in the SpatialShapes table intersect the line in the above example, execute the following query in the Interactive SQL Spatial Viewer.

```
SELECT Shape
FROM SpatialShapes
WHERE NEW ST_LineString( 'LineString( 2 2, 4
4 )' ).ST_Intersects( Shape ) = 1
UNION ALL SELECT NEW ST_LineString( 'LineString( 2 2, 4 4 )' )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.27

ST_IntersectsFilter(ST_Geometry) method

An inexpensive test if the two geometries might intersect.

Syntax

`geometry-expression.ST_IntersectsFilter(` ST_Geometry `geo2)`

Parameters

- **geo2 –** The other geometry value that is to be compared to the geometry-expression.

Returns

`BIT` Returns 1 if the geometry-expression might intersect with geo2, otherwise 0.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_IntersectsRect(ST_Point , ST_Point) method

Test if a geometry intersects a rectangle.

Syntax

`geometry-expression.ST_IntersectsRect(` ST_Point `pmin`, ST_Point `pmax`)

Parameters

- **pmin –** The minimum point value that is to be compared to the geometry-expression.
- **pmax –** The maximum point value that is to be compared to the geometry-expression.

Returns

`BIT` Returns 1 if the geometry-expression intersects with the specified rectangle, otherwise 0.

Examples

- **Example 1 –** The following example lists the ShapeIDs where the rectangle specified by the envelope of the two points intersects the corresponding Shape geometry. This example returns the result `3,5,6,18`.

```
SELECT LIST( ShapeID ORDER BY ShapeID )
FROM SpatialShapes
WHERE Shape.ST_IntersectsRect( NEW ST_Point( 0, 4 ), NEW
ST_Point( 2, 5 ) ) = 1
```

  The following example tests if a linestring intersects a rectangle. The provided linestring does not intersect the rectangle identified by the two points (even though the envelope of the linestring does intersect the envelope of the two points).

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

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_Is3D() method

Determines if the geometry value has Z coordinate values.

Syntax

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

Returns

`BIT` Returns 1 if the geometry value has Z coordinate values, otherwise 0.

Examples

- **Example 1 –** The following example returns the result `1`.

```
SELECT ShapeID FROM SpatialShapes WHERE Shape.ST_Is3D() = 1
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.10

ST_IsEmpty() method

Determines whether the geometry value represents an empty set.

Syntax

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

Returns

`BIT` Returns 1 if the geometry value is empty, otherwise 0.

Examples

- **Example 1 –** The following example returns the result `1`.

```
SELECT NEW ST_LineString().ST_IsEmpty()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.7

ST_IsMeasured() method

Determines if the geometry value has associated measure values.

Syntax

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

Returns

`BIT` Returns 1 if the geometry value has measure values, otherwise 0.

Examples

- **Example 1 –** The following example returns the result `1`.

```
SELECT ST_Geometry::ST_GeomFromText( 'LineString M( 1 2 4, 5 7
3 )' ).ST_IsMeasured()
```

  The following example returns the result `0`.

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

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.11

ST_IsSimple() method

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

Syntax

`geometry-expression.ST_IsSimple()`

Returns

`BIT` Returns 1 if the geometry value is simple, otherwise 0.

Examples

- **Example 1 –** The following returns the result `29` because the corresponding multi linestring contains two lines which cross.

```
SELECT ShapeID FROM SpatialShapes WHERE Shape.ST_IsSimple() = 0
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.8

ST_IsValid() method

Determines whether the geometry is a valid spatial object.

Syntax

`geometry-expression.ST_IsValid()`

Returns

`BIT` Returns 1 if the geometry value is valid, otherwise 0.

Examples

- **Example 1 –** The following returns the result `0` because the polygon contains a bow tie (the ring has a self-intersection).

```
SELECT ST_Geometry::ST_GeomFromText( 'Polygon(( 0 0, 4 0, 4 5, 0
-1, 0 0 ))' )
.ST_IsValid()
```

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

```
SELECT ST_Geometry::ST_GeomFromText(
'MultiPolygon((( 0 0, 2 0, 1 2, 0 0 )),((0 2, 1 0, 2 2, 0 2)))' )
.ST_IsValid()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.9

ST_LatNorth() method

Retrieves the northernmost latitude of a geometry.

Syntax

`geometry-expression.ST_LatNorth()`

Returns

`DOUBLE` Returns the northernmost latitude of the geometry-expression.

Examples

- **Example 1 –** The following example returns the result `49.74`.

```
SELECT ROUND( NEW ST_LineString( 'LineString( -122 49, -96 49 )',
4326 )
.ST_LatNorth(), 2 )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_LatSouth() method

Retrieves the southernmost latitude of a geometry.

Syntax

`geometry-expression.ST_LatSouth()`

Returns

DOUBLE Returns the southernmost latitude of the geometry-expression.

Examples

- **Example 1 –** The following example returns the result 49.

```
SELECT ROUND( NEW ST_LineString( 'LineString( -122 49, -96 49 )',
4326 )
.ST_LatSouth(), 2 )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_Length_Spheroid(VARCHAR(128)) method

Calculates the linear length of a curve/multicurve on the surface of the Earth.

Syntax

geometry-expression.ST_Length_Spheroid(VARCHAR(128) unit_name)

Parameters

- **unit_name –** The linear unit of measure. Defaults to the unit of the spatial reference system.

Returns

DOUBLE Returns the linear length of the curve/multicurve on the surface of the Earth calculated in the specified linear units.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_LinearHash() method

Returns a binary string that is a linear hash of the geometry.

Syntax

geometry-expression.ST_LinearHash()

Returns

BINARY_012332321_ Returns a binary string that is a linear hash of the geometry.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_LinearUnHash(BINARY(32)[, INT]) method

Returns a geometry representing the index hash.

Syntax

`ST_Geometry::ST_LinearUnHash`(BINARY(32) `index_hash`[, INT `srid`])

Parameters

- **index_hash –** The index hash string.
- **srid –** The SRID of the index hash. If not specified, the default is 0.

Returns

`ST_Geometry` Returns a representative geometry for the given linear hash.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_LoadConfigurationData(VARCHAR(128)) method

Returns binary configuration data. For internal use only.

Syntax

`ST_Geometry::ST_LoadConfigurationData`(VARCHAR(128) `configuration_name`)

Parameters

- **configuration_name –** The name of the configuration data item to load.

Returns

`LONG BINARY` Returns binary configuration data. For internal use only.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_LocateAlong(DOUBLE) method

Returns the subset of the geometry value that is associated with the given measure value.

Syntax

```
geometry-expression.ST_LocateAlong(DOUBLE measure)
```

Parameters

- **measure –** The measure value to look for in the geometry value.

Returns

`ST_Geometry` Returns a geometry value representing all parts of the geometry value that have the given measure value.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.12

ST_LocateBetween(DOUBLE, DOUBLE) method

Returns the subset of the geometry value that is between the specified start measure and end measure.

Syntax

```
geometry-expression.ST_LocateBetween(DOUBLE start_measure,
DOUBLE end_measure)
```

Parameters

- **start_measure –** The minimum measure value to look for in the geometry value.
- **end_measure –** The maximum measure value to look for in the geometry value.

Returns

`ST_Geometry` Returns a geometry value representing all parts of the geometry value that have a measure value between the specified start and end.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.13

ST_LongEast() method

Retrieves the longitude of the eastern boundary of a geometry.

Syntax

```
geometry-expression.ST_LongEast()
```

Returns

`DOUBLE` Retrieves the longitude of the eastern boundary of the geometry-expression.

Examples

- **Example 1 –** The following example returns the result `-157.8`.

```
SELECT NEW ST_LineString( 'LineString( -157.8 21.3, 144.5 13 )',
4326 )
.ST_LongEast()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_LongWest() method

Retrieves the longitude of the western boundary of a geometry.

Syntax

```
geometry-expression.ST_LongWest()
```

Returns

`DOUBLE` Retrieves the longitude of the western boundary of the geometry-expression.

Examples

- **Example 1 –** The following example returns the result `144.5`.

```
SELECT NEW ST_LineString( 'LineString( -157.8 21.3, 144.5 13 )',
4326 )
.ST_LongWest()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

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.

Examples

- **Example 1 –** The following example returns the result `8`.

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

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

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.

Examples

- **Example 1 –** The following example returns the result `4`.

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

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_OrderingEquals(ST_Geometry) method

Tests if a geometry is identical to another geometry.

Syntax

`geometry-expression.ST_OrderingEquals( ST_Geometry geo2)`

Parameters

- **geo2 –** The other geometry value that is to be compared to the geometry-expression.

Returns

`BIT` Returns 1 if the two geometry values are exactly equal, otherwise 0.

Examples

- **Example 1 –** The following example returns the result `16`. The Shape corresponding to ShapeID the result `16` contains the exact same points in the exact same order as the specified polygon.

```
SELECT ShapeID FROM SpatialShapes
WHERE Shape.ST_OrderingEquals( NEW ST_Polygon( 'Polygon ((0 0, 2
0, 1 2, 0 0))' ) ) = 1
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.43

ST_Overlaps(ST_Geometry) method

Tests if a geometry value overlaps another geometry value.

Syntax

`geometry-expression.ST_Overlaps( ST_Geometry geo2)`

Parameters

- **geo2 –** The other geometry value that is to be compared to the geometry-expression.

Returns

`BIT` Returns 1 if the geometry-expression overlaps geo2, otherwise 0. Returns NULL if geometry-expression and geo2 have different dimensions.

Examples

- **Example 1 –** 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 )' ) )
```

  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 ) )
```

  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 ) )
```

  The following returns the result `24,25,28,31`, which is the list of ShapeIDs that overlap the specified polygon.

```
SELECT LIST( ShapeID ORDER BY ShapeID ) FROM SpatialShapes
WHERE Shape.ST_Overlaps( NEW ST_Polygon( 'Polygon(( -1 0, 0 0, 0
1, -1 1, -1 0 ))' )
) = 1
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.32

ST_Relate(ST_Geometry) method

Determines how a geometry value is spatially related to another geometry value by returning an intersection matrix. The ST_Relate method returns 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(` ST_Geometry `geo2)`

Parameters

- **geo2 –** The second geometry value that is to be compared to the geometry-expression.

Returns

`CHAR_01239321_` 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.

Examples

- **Example 1 –** 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 )' ) )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_Reverse() method

Returns the geometry with the element order reversed.

Syntax

`geometry-expression.ST_Reverse()`

Returns

`ST_Geometry` Returns the geometry with the element order reversed.

Examples

- **Example 1 –** The following example returns the result `LineString (3 4, 1 2)`. It shows how the order of points in a linestring is reversed by ST_Reverse.

```
SELECT NEW ST_LineString( NEW ST_Point(1,2), NEW
ST_Point(3,4) ).ST_Reverse()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_Segmentize(DOUBLE) method

Add points so that no line segment is longer than a specified distance.

Syntax

`geometry-expression.ST_Segmentize(`DOUBLE `max_linesg)`

Parameters

- **max_linesg –** The maximum length of a line segment in the result.

Returns

`ST_Geometry` Returns a geometry with no individual line segment longer than the specified distance.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_Simplify(DOUBLE) method

Remove points from curves so long as the maximum introduced error is less than a specified tolerance.

Syntax

```
geometry-expression.ST_Simplify(DOUBLE tolerance)
```

Parameters

- **tolerance –** The maximum tolerance to use while simplifying.

Returns

`ST_Geometry` Returns a simplified geometry with some points removed.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_SnapToGrid(ST_Point , DOUBLE, DOUBLE, DOUBLE, DOUBLE) method

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

Syntax

```
geometry-expression.ST_SnapToGrid( ST_Point origin, DOUBLE
cell_size_x, DOUBLE cell_size_y, DOUBLE cell_size_z, DOUBLE
cell_size_m)
```

Parameters

- **origin –** The origin of the grid.
- **cell_size_x –** The cell size for the grid in the X dimension.
- **cell_size_y –** The cell size for the grid in the Y dimension.
- **cell_size_z –** The cell size for the grid in the Z dimension.
- **cell_size_m –** The cell size for the grid in the M dimension.

Returns

`ST_Geometry` Returns the geometry with all points snapped to the grid.

Examples

- **Example 1 –** The following example returns the result `LineString (1.010101 20.20202, 1.015625 20.203125, 1.01 20.2)`.

```
SELECT NEW ST_LineString(
NEW ST_Point( 1.010101, 20.202020 ),
TREAT( NEW ST_Point( 1.010101, 20.202020 ).ST_SnapToGrid( NEW
ST_Point( 0.0, 0.0 ), POWER( 2, -6 ), POWER( 2, -7 ), 0.0, 0.0 ) AS
ST_Point ),
TREAT( NEW ST_Point( 1.010101, 20.202020 ).ST_SnapToGrid( NEW
ST_Point( 1.01, 20.2 ), POWER( 2, -6 ), POWER( 2, -7 ), 0.0, 0.0 )
AS ST_Point ) )
```

  The first point of the linestring is the point ST_Point(1.010101, 20.202020), snapped to the grid defined for SRID 0. The second point of the linestring is the same point snapped to a grid defined with its origin at point (0.0 0.0), where cell size x is POWER(2, -6) and cell size y is POWER(2, -7). The third point of the linestring is the same point snapped to a grid defined with its origin at point (1.01, 20.2), where cell size x is POWER(2, -6) and cell size y is POWER(2, -7).

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_SRID(INT) method

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

Syntax

`geometry-expression.ST_SRID(`INT `srid`)

Parameters

- **srid –** The SRID to use for the result.

Returns

`ST_Geometry` Returns a copy of the geometry value with the specified spatial reference system.

Examples

- **Example 1 –** The following example returns the result `SRID=1000004326;Point (-118 34)`.

```
SELECT NEW ST_Point( -118, 34,
4326 ).ST_SRID( 1000004326 ).ST_AsText( 'EWKT' )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.5

ST_SRIDFromBaseType(VARCHAR(128)) method

Parses a string defining the type string.

Syntax

```
ST_Geometry::ST_SRIDFromBaseType(VARCHAR(128) base_type_str)
```

Parameters

- **base_type_str –** A string containing the base type string

Returns

`INT` Returns the SRID from a type string. If no SRID is specified by the string, returns NULL. If the type string is not a valid geometry type string, an error is returned.

Examples

- **Example 1 –** The following example returns the result `NULL`.

```
SELECT ST_Geometry::ST_SRIDFromBaseType('ST_Geometry')
```

The following example returns the result `4326`.

```
SELECT ST_Geometry::ST_SRIDFromBaseType('ST_Geometry(SRID=4326)')
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_SymDifference(ST_Geometry) method

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

Syntax

```
geometry-expression.ST_SymDifference( ST_Geometry geo2)
```

Parameters

- **geo2 –** The other geometry value that is to be subtracted from the geometry-expression to find the symmetric difference.

Returns

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

Examples

- **Example 1 –** The following example shows the symmetric difference (C) of a square (A) and a circle (B).

```
SELECT NEW ST_Polygon( 'Polygon( (-1 -0.25, 1 -0.25, 1 2.25, -1
2.25, -1 -0.25) )' ) AS A
, NEW ST_CurvePolygon( 'CurvePolygon( CircularString( 0 1, 1 2, 2
1, 1 0, 0 1 ) )' )  AS B
, A.ST_SymDifference( B ) AS C
```

  The following picture shows the result of the symmetric difference as the shaded portion of the picture. The symmetric difference is a multisurface that includes two surfaces: one surface contains all of the points from the square that are not in the circle, and the other surface contains all of the points of the circle that are not in the square. Union of a square and a circle

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.21

ST_ToCircular() method

Convert the geometry to a circularstring

Syntax

`geometry-expression.ST_ToCircular()`

Returns

`ST_CircularString` If the geometry-expression is of type ST_CircularString, return the geometry-expression. If the geometry-expression is of type ST_CompoundCurve with a single element which is of type ST_CircularString, return that element. If the geometry-expression is a geometry collection with a single element of type ST_CircularString, return that element. If the geometry-expression is the empty set, return an empty set of type ST_CircularString. Otherwise, raise an exception condition.

Examples

- **Example 1 –** The following example returns the result `CircularString (0 0, 1 1, 2 0)`.

```
SELECT NEW ST_CompoundCurve( 'CompoundCurve(CircularString( 0 0, 1
1, 2 0 ))' ).ST_ToCircular()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.33

ST_ToCompound() method

Converts the geometry to a compound curve.

Syntax

```
geometry-expression.ST_ToCompound()
```

Returns

`ST_CompoundCurve` If the geometry-expression is of type ST_CompoundCurve, return the geometry-expression. If the geometry-expression is of type ST_LineString or ST_CircularString, return a compound curve containing one element, the geometry-expression. If the geometry-expression is a geometry collection with a single element of type ST_Curve, return that element cast as ST_CompoundCurve. If the geometry-expression is the empty set, return an empty set of type ST_CompoundCurve. Otherwise, raise an exception condition.

Examples

- **Example 1 –** The following example returns the result `CompoundCurve ((0 0, 2 1))`.

```
SELECT NEW ST_LineString( 'LineString( 0 0, 2
1 )' ).ST_ToCompound()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.33

ST_ToCurve() method

Converts the geometry to a curve.

Syntax

```
geometry-expression.ST_ToCurve()
```

Returns

`ST_Curve` If the geometry-expression is of type ST_Curve, return the geometry-expression. If the geometry-expression is a geometry collection with a single element of type ST_Curve, return that element. If the geometry-expression is the empty set, return an empty set of type ST_LineString. Otherwise, raise an exception condition.

Examples

- **Example 1 –** The following example returns the result `LineString (0 0, 1 1, 2 0)`.

```
SELECT NEW ST_GeomCollection( 'GeometryCollection(LineString(0 0,
1 1, 2 0))' ).ST_ToCurve()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_ToCurvePoly() method

Converts the geometry to a curve polygon.

Syntax

`geometry-expression.ST_ToCurvePoly()`

Returns

`ST_CurvePolygon` If the geometry-expression is of type ST_CurvePolygon, return the geometry-expression. If the geometry-expression is a geometry collection with a single element of type ST_CurvePolygon, return that element. If the geometry-expression is the empty set, return an empty set of type ST_CurvePolygon. Otherwise, raise an exception condition.

Examples

- **Example 1 –** The following example returns the result `Polygon ((0 0, 2 0, 1 2, 0 0))`.

```
SELECT NEW ST_MultiPolygon('MultiPolygon(((0 0, 2 0, 1 2, 0
0)))' ).ST_ToCurvePoly()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.33

ST_ToGeomColl() method

Converts the geometry to a geometry collection.

Syntax

`geometry-expression.ST_ToGeomColl()`

Returns

`ST_GeomCollection` If the geometry-expression is of type ST_GeomCollection, returns the geometry-expression. If the geometry-expression is of type ST_Point, ST_Curve, or

ST_Surface, then return a geometry collection containing one element, the geometry-expression. If the geometry-expression is the empty set, returns an empty set of type ST_GeomCollection. Otherwise, raises an exception condition.

Examples

- **Example 1 –** The following example returns the result `GeometryCollection (Point (0 1))`.

```
SELECT NEW ST_Point( 0, 1 ).ST_ToGeomColl()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.33

ST_ToLineString() method

Converts the geometry to a linestring.

Syntax

`geometry-expression.ST_ToLineString()`

Returns

`ST_LineString` If the geometry-expression is of type ST_LineString, return the geometry-expression. If the geometry-expression is of type ST_CircularString or ST_CompoundCurve, return geometry-expression.ST_CurveToLine(). If the geometry-expression is a geometry collection with a single element of type ST_Curve, return that element cast as ST_LineString. If the geometry-expression is the empty set, return an empty set of type ST_LineString. Otherwise, raise an exception condition.

Examples

- **Example 1 –** The following returns an error because the Shape column is of type ST_Geometry and ST_Geometry does not support the ST_Length method.

```
SELECT Shape.ST_Length()
FROM SpatialShapes WHERE ShapeID = 5
```

The following uses ST_ToLineString to change the type of the Shape column expression to ST_LineString. ST_Length returns the result `7`.

```
SELECT Shape.ST_ToLineString().ST_Length()
FROM SpatialShapes WHERE ShapeID = 5
```

In this case, the value of the Shape column is known be of type ST_LineString, so TREAT can be used to efficiently change the type of the expression. ST_Length returns the result `7`.

```
SELECT TREAT( Shape AS ST_LineString ).ST_Length()
FROM SpatialShapes WHERE ShapeID = 5
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.33

ST_ToMultiCurve() method

Converts the geometry to a multicurve value.

Syntax

`geometry-expression.ST_ToMultiCurve()`

Returns

`ST_MultiCurve` If the geometry-expression is of type ST_MultiCurve, returns the geometry-expression. If the geometry-expression is a geometry collection containing only curves, returns a multicurve object containing the elements of the geometry-expression. If the geometry-expression is of type ST_Curve then return a multicurve value containing one element, the geometry-expression. If the geometry-expression is the empty set, returns an empty set of type ST_MultiCurve. Otherwise, raises an exception condition.

Examples

- **Example 1 –** The following example returns the result `MultiCurve ((0 7, 0 4, 4 4))`.

```
SELECT Shape.ST_ToMultiCurve()
FROM SpatialShapes WHERE ShapeID = 5
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.33

ST_ToMultiLine() method

Converts the geometry to a multilinestring value.

Syntax

`geometry-expression.ST_ToMultiLine()`

Returns

`ST_MultiLineString` If the geometry-expression is of type ST_MultiLineString, returns the geometry-expression. If the geometry-expression is a geometry collection containing only lines, returns a multilinestring object containing the elements of the geometry-expression. If the geometry-expression is of type ST_LineString then return a multilinestring value containing one element, the geometry-expression. If the geometry-

expression is the empty set, returns an empty set of type ST_MultiCurve. Otherwise, raises an exception condition.

Examples

- **Example 1 –** The following returns an error because the Shape column is of type ST_Geometry and ST_Geometry does not support the ST_Length method.

```
SELECT Shape.ST_Length()
FROM SpatialShapes WHERE ShapeID = 29
```

The following uses ST_ToMultiLine to change the type of the Shape column expression to ST_MultiLineString. This example would also work with ShapeID 5, where the Shape value is of type ST_LineString. ST_Length returns the result `4.236068`.

```
SELECT Shape.ST_ToMultiLine().ST_Length()
FROM SpatialShapes WHERE ShapeID = 29
```

In this case, the value of the Shape column is known be of type ST_MultiLineString, so TREAT can be used to efficiently change the type of the expression. This example would *not*work with ShapeID 5, where the Shape value is of type ST_LineString. ST_Length returns the result `4.236068`.

```
SELECT TREAT( Shape AS ST_MultiLineString ).ST_Length()
FROM SpatialShapes WHERE ShapeID = 29
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.33

ST_ToMultiPoint() method

Converts the geometry to a multi-point value.

Syntax

`geometry-expression.ST_ToMultiPoint()`

Returns

`ST_MultiPoint` If the geometry-expression is of type ST_MultiPoint, returns the geometry-expression. If the geometry-expression is a geometry collection containing only points, returns a multipoint object containing the elements of the geometry-expression. If the geometry-expression is of type ST_Point then return a multi-point value containing one element, the geometry-expression. If the geometry-expression is the empty set, returns an empty set of type ST_MultiPoint. Otherwise, raises an exception condition.

Examples

- **Example 1 –** The following example returns the result `MultiPoint EMPTY`.

```
SELECT NEW ST_GeomCollection().ST_ToMultiPoint()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.33

ST_ToMultiPolygon() method

Converts the geometry to a multi-polygon value.

Syntax

`geometry-expression.ST_ToMultiPolygon()`

Returns

`ST_MultiPolygon` If the geometry-expression is of type ST_MultiPolygon, returns the geometry-expression. If the geometry-expression is a geometry collection containing only polygons, returns a multi-polygon object containing the elements of the geometry-expression. If the geometry-expression is of type ST_Polygon then return a multi-polygon value containing one element, the geometry-expression. If the geometry-expression is the empty set, returns an empty set of type ST_MultiSurface. Otherwise, raises an exception condition.

Examples

- **Example 1 –** The following example returns the result `MultiPolygon EMPTY`.

  ```
  SELECT NEW ST_GeomCollection().ST_ToMultiPolygon()
  ```

  The following returns an error because the Shape column is of type ST_Geometry and ST_Geometry does not support the ST_Area method.

  ```
  SELECT Shape.ST_Area()
  FROM SpatialShapes WHERE ShapeID = 27
  ```

  The following uses ST_ToMultiPolygon to change the type of the Shape column expression to ST_MultiPolygon. This example would also work with ShapeID 22, where the Shape value is of type ST_LineString. ST_Area returns the result `8`.

  ```
  SELECT Shape.ST_ToMultiPolygon().ST_Area()
  FROM SpatialShapes WHERE ShapeID = 27
  ```

  In this case, the value of the Shape column is known be of type ST_MultiPolygon, so TREAT can be used to efficiently change the type of the expression. This example would *not* work with ShapeID 22, where the Shape value is of type ST_Polygon. ST_Area returns the result `8`.

  ```
  SELECT TREAT( Shape AS ST_MultiPolygon ).ST_Area()
  FROM SpatialShapes WHERE ShapeID = 27
  ```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.33

ST_ToMultiSurface() method

Converts the geometry to a multi-surface value.

Syntax

```
geometry-expression.ST_ToMultiSurface()
```

Returns

`ST_MultiSurface` If the geometry-expression is of type ST_MultiSurface, returns the geometry-expression. If the geometry-expression is a geometry collection containing only surfaces, returns a multi-surface object containing the elements of the geometry-expression. If the geometry-expression is of type ST_Surface then return a multi-surface value containing one element, the geometry-expression. If the geometry-expression is the empty set, returns an empty set of type ST_MultiSurface. Otherwise, raises an exception condition.

Examples

- **Example 1 –** The following example returns the result `MultiSurface EMPTY`.

```
SELECT NEW ST_GeomCollection().ST_ToMultiSurface()
```

The following example returns the result `MultiSurface (((3 3, 8 3, 4 8, 3 3)))`.

```
SELECT Shape.ST_ToMultiSurface()
FROM SpatialShapes WHERE ShapeID = 22
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.33

ST_ToPoint() method

Converts the geometry to a point.

Syntax

```
geometry-expression.ST_ToPoint()
```

Returns

`ST_Point` If the geometry-expression is of type ST_Point, return the geometry-expression. If the geometry-expression is a geometry collection with a single element of type ST_Point, return that element. If the geometry-expression is the empty set, return an empty set of type ST_Point. Otherwise, raise an exception condition.

Examples

- **Example 1 –** The following example returns the result `Point (1 2)`.

```
SELECT NEW ST_GeomCollection( NEW ST_Point(1,2) ).ST_ToPoint()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.33

ST_ToPolygon() method

Converts the geometry to a polygon.

Syntax

`geometry-expression.ST_ToPolygon()`

Returns

`ST_Polygon` If the geometry-expression is of type ST_Polygon, returns the geometry-expression. If the geometry-expression is of type ST_CurvePolygon, returns geometry-expression.ST_CurvePolyToPoly(). If the geometry-expression is a geometry collection with a single element of type ST_CurvePolygon, returns that element. If the geometry-expression is the empty set, returns an empty set of type ST_Polygon. Otherwise, raises an exception condition.

Examples

- **Example 1 –** The following example returns the result `Polygon EMPTY`.

```
SELECT NEW ST_GeomCollection().ST_ToPolygon()
```

The following returns an error because the Shape column is of type ST_Geometry and ST_Geometry does not support the ST_Area method.

```
SELECT Shape.ST_Area()
FROM SpatialShapes WHERE ShapeID = 22
```

The following uses ST_ToPolygon to change the type of the Shape column expression to ST_Polygon. ST_Area returns the result `12.5`.

```
SELECT Shape.ST_ToPolygon().ST_Area()
FROM SpatialShapes WHERE ShapeID = 22
```

In this case, the value of the Shape column is known be of type ST_Polygon, so TREAT can be used to efficiently change the type of the expression. ST_Area returns the result `12.5`.

```
SELECT TREAT( Shape AS ST_Polygon ).ST_Area()
FROM SpatialShapes WHERE ShapeID = 22
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.33

ST_ToSurface() method

Converts the geometry to a surface.

Syntax

```
geometry-expression.ST_ToSurface()
```

Returns

`ST_Surface` If the geometry-expression is of type ST_Surface, return the geometry-expression. If the geometry-expression is a geometry collection with a single element of type ST_Surface, return that element. If the geometry-expression is the empty set, return an empty set of type ST_Polygon. Otherwise, raise an exception condition.

Examples

- **Example 1 –** The following example returns the result `Polygon EMPTY`.

```
SELECT NEW ST_GeomCollection().ST_ToSurface()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_Touches(ST_Geometry) method

Tests if a geometry value spatially touches another geometry value.

Syntax

```
geometry-expression.ST_Touches( ST_Geometry geo2)
```

Parameters

- **geo2 –** The other geometry value that is to be compared to the geometry-expression.

Returns

`BIT` Returns 1 if the geometry-expression touches geo2, otherwise 0. Returns NULL if both geometry-expression and geo2 have dimension 0.

Examples

- **Example 1 –** The following example returns NULL because both inputs are points and have no boundary.

```
SELECT NEW ST_Point(1,1).ST_Touches( NEW ST_Point( 1,1 ) )
```

The following example lists the ShapeIDs of the geometries that touch the "Lighting Bolt" shape, which has ShapeID 6. This example returns the result `5,16,26`. Each of the three touching geometries intersect the Lighting Bolt only at its boundary.

```
SELECT List( ShapeID ORDER BY ShapeID )
FROM SpatialShapes
WHERE Shape.ST_Touches( ( SELECT Shape FROM SpatialShapes WHERE
ShapeID = 6 ) ) = 1
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.28

ST_Transform(INT) method

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

Syntax

`geometry-expression.ST_Transform`(INT `srid`)

Parameters

- **srid –** The SRID of the result.

Returns

`ST_Geometry` Returns a copy of the geometry value transformed into the specified spatial reference system.

Examples

- **Example 1 –** The following example returns the result `Point (184755.86861 -444218.175691)`. It transforms a point in Los Angeles which is specified in longitude and latitude to the projected planar SRID 3310 ("NAD83 / California Albers"). This example assumes that the 'st_geometry_predefined_srs' feature has been installed by the sa_install_feature system procedure. See sa_install_feature system procedure.

  ```
  SELECT NEW ST_Point( -118, 34, 4326 ).ST_Transform( 3310 )
  ```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.6

ST_Union(ST_Geometry) method

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

Syntax

`geometry-expression.ST_Union`(ST_Geometry `geo2`)

Parameters

- **geo2 –** 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.

Examples

- **Example 1 –** The following example shows the union (C) of a square (A) and a circle (B).

```
SELECT NEW ST_Polygon( 'Polygon( (-1 -0.25, 1 -0.25, 1 2.25, -1
2.25, -1 -0.25) )' ) AS A
, NEW ST_CurvePolygon( 'CurvePolygon( CircularString( 0 1, 1 2, 2
1, 1 0, 0 1 ) )' )  AS B
, A.ST_Union( B ) AS C
```

  The union is shaded in the following picture. The union is a single surface that includes all of the points that are in A or in B. Union of a square and a circle

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.19

ST_UnionAggr(ST_Geometry) method

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

Syntax

`ST_Geometry::ST_UnionAggr`(ST_Geometry `geometry_column`)

Parameters

- **geometry_column –** 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.

Examples

- **Example 1 –** The following example returns the result `Polygon ((.555555 3, 0 3, 0 1.75, 0 0, 3 0, 3 3, .75 3, 1 4, .555555 3))`.

```
SELECT ST_Geometry::ST_UnionAggr( Shape )
FROM SpatialShapes WHERE ShapeID IN ( 2, 6 )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_Within(ST_Geometry) method

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

Syntax

`geometry-expression.ST_Within( ST_Geometry geo2)`

Parameters

- **geo2 –** The other geometry value that is to be compared to the geometry-expression.

Returns

`BIT` Returns 1 if the geometry-expression is within geo2, otherwise 0.

Examples

- **Example 1 –** 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 ))' ) )
  ```

  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 ))' ) )
  ```

  The following example lists the ShapeIDs where the given point is within the Shape geometry. This example returns the result `3,5`. Note that ShapeID 6 is not listed because the point intersects that row's Shape polygon at the polygon's boundary.

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

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 5.1.30

ST_WithinDistance(ST_Geometry , DOUBLE, VARCHAR(128)) method

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

Syntax

`geometry-expression.ST_WithinDistance( ST_Geometry geo2, DOUBLE distance, VARCHAR(128) unit_name)`

Parameters

- **geo2 –** The other geometry value whose distance is to be measured from the geometry-expression.
- **distance –** The distance the two geometries should be within.
- **unit_name –** 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

`BIT` Returns 1 if geometry-expression and geo2 are within the specified distance of each other, otherwise 0.

Examples

- **Example 1 –** The following example returns an ordered result set with one row for each shape that is within distance 1.4 of the point (2,3).

```
SELECT ShapeID, ROUND( Shape.ST_Distance( NEW ST_Point( 2, 3 ) ),
2 ) AS dist
FROM SpatialShapes
WHERE ShapeID < 17
AND Shape.ST_WithinDistance( NEW ST_Point( 2, 3 ), 1.4 ) = 1
ORDER BY dist
```

The example returns the following result set:

ShapeID	dist
2	0.0
3	0.0
5	1.0
6	1.21

The following example creates points representing Halifax, NS and Waterloo, ON, Canada and uses ST_WithinDistance to demonstrate that the distance between the two

points is within 850 miles, but not within 840 miles. This example assumes that the 'st_geometry_predefined_uom' feature has been installed by the sa_install_feature system procedure. Seesa_install_feature system procedure.

```
SELECT NEW ST_Point( -63.573566, 44.646244, 4326 )
.ST_WithinDistance( NEW ST_Point( -80.522372, 43.465187, 4326 )
, 850, 'Statute mile' ) within850,
NEW ST_Point( -63.573566, 44.646244, 4326 )
.ST_WithinDistance( NEW ST_Point( -80.522372, 43.465187, 4326 )
, 840, 'Statute mile' ) within840
```

The example returns the following result set:

within850	within840
1	0

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_WithinDistanceFilter(ST_Geometry , DOUBLE, VARCHAR(128)) method

An inexpensive of whether two geometries might be within a specified distance of each other.

Syntax

`geometry-expression.ST_WithinDistanceFilter`(ST_Geometry `geo2`, DOUBLE `distance`, VARCHAR(128) `unit_name`)

Parameters

- **geo2 –** The other geometry value whose distance is to be measured from the geometry-expression.
- **distance –** The distance the two geometries should be within.
- **unit_name –** The units in which the distance parameter should be interpreted. The default is 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

`BIT` Returns 1 if geometry-expression and geo2 might be within the specified distance of each other, otherwise 0.

Examples

- **Example 1 –** The following example returns an ordered result set with one row for each shape that might be within distance 1.4 of the point (2,3). The result contains a shape that is not actually within the specified distance.

```
SELECT ShapeID, ROUND( Shape.ST_Distance( NEW ST_Point( 2, 3 ) ),
2 ) AS dist
FROM SpatialShapes
WHERE ShapeID < 17
AND Shape.ST_WithinDistanceFilter( NEW ST_Point( 2, 3 ), 1.4 ) = 1
ORDER BY dist
```

The example returns the following result set:

ShapeID	dist
2	0.0
3	0.0
5	1.0
6	1.21
16	1.41

The following example creates points representing Halifax, NS and Waterloo, ON, Canada, and uses ST_WithinDistanceFilter to demonstrate that the distance between the two points might be within 850 miles, but definitely is not within 750 miles. This example assumes that the st_geometry_predefined_uom feature has been installed by the sa_install_feature system procedure. Seesa_install_feature system procedure.

```
SELECT NEW ST_Point( -63.573566, 44.646244, 4326 )
.ST_WithinDistanceFilter( NEW ST_Point( -80.522372, 43.465187,
4326 )
, 850, 'Statute mile' ) within850,
NEW ST_Point( -63.573566, 44.646244, 4326 )
.ST_WithinDistanceFilter( NEW ST_Point( -80.522372, 43.465187,
4326 )
, 750, 'Statute mile' ) within750
```

The example returns the following result set:

within850	within750
1	0

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_WithinFilter(ST_Geometry) method

An inexpensive test if a geometry might be within another.

Syntax

```
geometry-expression.ST_WithinFilter( ST_Geometry geo2)
```

Parameters

- **geo2 –** The other geometry value that is to be compared to the geometry-expression.

Returns

`BIT` Returns 1 if the geometry-expression might be within geo2, otherwise 0.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

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.

Examples

- **Example 1 –** The following example returns the result `5`.

```
SELECT NEW ST_LineString( 'LineString ZM( 1 2 3 4, 5 6 7
8 )' ).ST_XMax()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

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.

Examples

- **Example 1 –** The following example returns the result 1.

```
SELECT NEW ST_LineString( 'LineString ZM( 1 2 3 4, 5 6 7
8 )' ).ST_XMin()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

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.

Examples

- **Example 1 –** The following example returns the result 6.

```
SELECT NEW ST_LineString( 'LineString ZM( 1 2 3 4, 5 6 7
8 )' ).ST_YMax()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

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.

Examples

- **Example 1 –** The following example returns the result `2`.

```
SELECT NEW ST_LineString( 'LineString ZM( 1 2 3 4, 5 6 7
8 )' ).ST_YMin()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

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.

Examples

- **Example 1 –** The following example returns the result `7`.

```
SELECT NEW ST_LineString( 'LineString ZM( 1 2 3 4, 5 6 7
8 )' ).ST_ZMax()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

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.

Examples

- **Example 1 –** The following example returns the result `3`.

```
SELECT NEW ST_LineString( 'LineString ZM( 1 2 3 4, 5 6 7
8 )' ).ST_ZMin()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_LineString type

The ST_LineString type is a subtype of ST_Curve that uses straight line segments between control points.

Syntax

`ST_LineString` type

Members

All members of the ST_LineString type, including all inherited members.

Members of ST_LineString:

- **ST_LineString(ST_Point , ST_Point , ST_Point) –** Constructs a linestring value from a list of points in a specified spatial reference system.
- **ST_LineString() –** Constructs a linestring representing the empty set.
- **ST_LineString(LONG BINARY[, INT]) –** Constructs a linestring from Well Known Binary (WKB).
- **ST_LineString(LONG VARCHAR[, INT]) –** Constructs a linestring from a text representation.
- **ST_LineStringAggr(ST_Point) –** Returns a linestring built from the ordered points in a group.
- **ST_NumPoints() –** Returns the number of points defining the linestring.
- **ST_PointN(INT) –** Returns the nth point in the linestring.

Members of ST_Curve:

- **ST_CurveToLine() –** Returns the ST_LineString interpolation of an ST_Curve value.
- **ST_EndPoint() –** Returns an ST_Point value that is the end point of the ST_Curve value.
- **ST_IsClosed() –** Test if the ST_Curve value is closed. A curve is closed if the start and end points are coincident.
- **ST_IsRing() –** Tests if the ST_Curve value is a ring. A curve is a ring if it is closed and simple (no self intersections).
- **ST_Length(VARCHAR(128)) –** Returns the length measurement of the ST_Curve value. The result is measured in the units specified by the unit-name parameter.
- **ST_StartPoint() –** Returns an ST_Point value that is the start point of the ST_Curve value.

Members of ST_Geometry:

- **ST_Affine(DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a new geometry that is the result of applying the specified 3-D affine transformation.
- **ST_AsBinary(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsBitmap(INT, INT, ST_Point , ST_Point , VARCHAR(128)) –** Returns a LONG VARBIT that is a bitmap representing a geometry value.
- **ST_AsGeoJSON(VARCHAR(128)) –** Returns a string representing a geometry in JSON format.
- **ST_AsGML(VARCHAR(128)) –** Returns the GML representation of an ST_Geometry value.
- **ST_AsKML(VARCHAR(128)) –** Returns the KML representation of an ST_Geometry value.
- **ST_AsSVG(VARCHAR(128)) –** Returns an SVG figure representing a geometry value.
- **ST_AsSVGAggr(ST_Geometry , VARCHAR(128)) –** Returns a complete or partial SVG document which renders the geometries in a group.
- **ST_AsText(VARCHAR(128)) –** Returns the text representation of an ST_Geometry value.
- **ST_AsWKB(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsWKT(VARCHAR(128)) –** Returns the WKT representation of an ST_Geometry value.
- **ST_AsXML(VARCHAR(128)) –** Returns the XML representation of an ST_Geometry value.
- **ST_Boundary() –** Returns the boundary of the geometry value.
- **ST_Buffer(DOUBLE, VARCHAR(128)) –** 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.
- **ST_Contains(ST_Geometry) –** Tests if a geometry value spatially contains another geometry value.
- **ST_ContainsFilter(ST_Geometry) –** An inexpensive test if a geometry might contain another.
- **ST_ConvexHull() –** Returns the convex hull of the geometry value.
- **ST_ConvexHullAggr(ST_Geometry) –** Returns the convex hull for all of the geometries in a group
- **ST_CoordDim() –** Returns the number of coordinate dimensions stored with each point of the ST_Geometry value.
- **ST_CoveredBy(ST_Geometry) –** Tests if a geometry value is spatially covered by another geometry value.
- **ST_CoveredByFilter(ST_Geometry) –** An inexpensive test if a geometry might be covered by another.

- **ST_Covers(ST_Geometry) –** Tests if a geometry value spatially covers another geometry value.
- **ST_CoversFilter(ST_Geometry) –** An inexpensive test if a geometry might cover another.
- **ST_Crosses(ST_Geometry) –** Tests if a geometry value crosses another geometry value.
- **ST_Debug(VARCHAR(128)) –** Returns a LONG BINARY that is debug information for the object.
- **ST_Difference(ST_Geometry) –** Returns the geometry value that represents the point set difference of two geometries.
- **ST_Dimension() –** Returns the dimension of the ST_Geometry value. Points have dimension 0, lines have dimension 1, and surfaces have dimension 2. Any empty geometry has dimension -1.
- **ST_Disjoint(ST_Geometry) –** Test if a geometry value is spatially disjoint from another value.
- **ST_Distance(ST_Geometry , VARCHAR(128)) –** Returns the smallest distance between the geometry-expression and the specified geometry value.
- **ST_Distance_Spheroid(ST_Geometry , VARCHAR(128)) –** Calculates the linear distance between geometries on the surface of the Earth.
- **ST_Envelope() –** Returns the bounding rectangle for the geometry value.
- **ST_EnvelopeAggr(ST_Geometry) –** Returns the bounding rectangle for all of the geometries in a group
- **ST_Equals(ST_Geometry) –** Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.
- **ST_EqualsFilter(ST_Geometry) –** An inexpensive test if a geometry is equal to another.
- **ST_GeometryType() –** Returns the name of the type of the ST_Geometry value.
- **ST_GeometryTypeFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_GeomFromBinary(LONG BINARY, INT) –** Constructs a geometry from a binary string representation.
- **ST_GeomFromShape(LONG BINARY[, INT]) –** Parses a string containing an ESRI shape record and creates a geometry value of the appropriate type.
- **ST_GeomFromText(LONG VARCHAR, INT) –** Constructs a geometry from a character string representation.
- **ST_GeomFromWKB(LONG BINARY, INT) –** Parse a string containing a WKB or EWKB representation of a geometry and creates a geometry value of the appropriate type.
- **ST_GeomFromWKT(LONG VARCHAR, INT) –** Parses a string containing the WKT or EWKT representation of a geometry and create a geometry value of the appropriate type.

- **ST_Intersection(ST_Geometry) –** Returns the geometry value that represents the point set intersection of two geometries.
- **ST_IntersectionAggr(ST_Geometry) –** Returns the spatial intersection of all of the geometries in a group
- **ST_Intersects(ST_Geometry) –** Test if a geometry value spatially intersects another value.
- **ST_IntersectsFilter(ST_Geometry) –** An inexpensive test if the two geometries might intersect.
- **ST_IntersectsRect(ST_Point , ST_Point) –** Test if a geometry intersects a rectangle.
- **ST_Is3D() –** Determines if the geometry value has Z coordinate values.
- **ST_IsEmpty() –** Determines whether the geometry value represents an empty set.
- **ST_IsMeasured() –** Determines if the geometry value has associated measure values.
- **ST_IsSimple() –** Determines whether the geometry value is simple (containing no self intersections or other irregularities).
- **ST_IsValid() –** Determines whether the geometry is a valid spatial object.
- **ST_LatNorth() –** Retrieves the northernmost latitude of a geometry.
- **ST_LatSouth() –** Retrieves the southernmost latitude of a geometry.
- **ST_Length_Spheroid(VARCHAR(128)) –** Calculates the linear length of a curve/multicurve on the surface of the Earth.
- **ST_LinearHash() –** Returns a binary string that is a linear hash of the geometry.
- **ST_LinearUnHash(BINARY(32)[, INT]) –** Returns a geometry representing the index hash.
- **ST_LoadConfigurationData(VARCHAR(128)) –** Returns binary configuration data. For internal use only.
- **ST_LocateAlong(DOUBLE) –** Returns the subset of the geometry value that is associated with the given measure value.
- **ST_LocateBetween(DOUBLE, DOUBLE) –** Returns the subset of the geometry value that is between the specified start measure and end measure.
- **ST_LongEast() –** Retrieves the longitude of the eastern boundary of a geometry.
- **ST_LongWest() –** Retrieves the longitude of the western boundary of a geometry.
- **ST_MMax() –** Retrieves the maximum M coordinate value of a geometry.
- **ST_MMin() –** Retrieves the minimum M coordinate value of a geometry.
- **ST_OrderingEquals(ST_Geometry) –** Tests if a geometry is identical to another geometry.
- **ST_Overlaps(ST_Geometry) –** Tests if a geometry value overlaps another geometry value.
- **ST_Relate(ST_Geometry) –** Determines how a geometry value is spatially related to another geometry value by returning an intersection matrix. The ST_Relate method returns 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.

- **ST_Reverse() –** Returns the geometry with the element order reversed.
- **ST_Segmentize(DOUBLE) –** Add points so that no line segment is longer than a specified distance.
- **ST_Simplify(DOUBLE) –** Remove points from curves so long as the maximum introduced error is less than a specified tolerance.
- **ST_SnapToGrid(ST_Point , DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a copy of the geometry with all points snapped to the specified grid.
- **ST_SRID(INT) –** Changes the spatial reference system associated with the geometry without modifying any of the values.
- **ST_SRIDFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_SymDifference(ST_Geometry) –** Returns the geometry value that represents the point set symmetric difference of two geometries.
- **ST_ToCircular() –** Convert the geometry to a circularstring
- **ST_ToCompound() –** Converts the geometry to a compound curve.
- **ST_ToCurve() –** Converts the geometry to a curve.
- **ST_ToCurvePoly() –** Converts the geometry to a curve polygon.
- **ST_ToGeomColl() –** Converts the geometry to a geometry collection.
- **ST_ToLineString() –** Converts the geometry to a linestring.
- **ST_ToMultiCurve() –** Converts the geometry to a multicurve value.
- **ST_ToMultiLine() –** Converts the geometry to a multilinestring value.
- **ST_ToMultiPoint() –** Converts the geometry to a multi-point value.
- **ST_ToMultiPolygon() –** Converts the geometry to a multi-polygon value.
- **ST_ToMultiSurface() –** Converts the geometry to a multi-surface value.
- **ST_ToPoint() –** Converts the geometry to a point.
- **ST_ToPolygon() –** Converts the geometry to a polygon.
- **ST_ToSurface() –** Converts the geometry to a surface.
- **ST_Touches(ST_Geometry) –** Tests if a geometry value spatially touches another geometry value.
- **ST_Transform(INT) –** Creates a copy of the geometry value transformed into the specified spatial reference system.
- **ST_Union(ST_Geometry) –** Returns the geometry value that represents the point set union of two geometries.
- **ST_UnionAggr(ST_Geometry) –** Returns the spatial union of all of the geometries in a group
- **ST_Within(ST_Geometry) –** Tests if a geometry value is spatially contained within another geometry value.
- **ST_WithinDistance(ST_Geometry , DOUBLE, VARCHAR(128)) –** Test if two geometries are within a specified distance of each other.

- **ST_WithinDistanceFilter(ST_Geometry , DOUBLE, VARCHAR(128)) –** An inexpensive of whether two geometries might be within a specified distance of each other.
- **ST_WithinFilter(ST_Geometry) –** An inexpensive test if a geometry might be within another.
- **ST_XMax() –** Retrieves the maximum X coordinate value of a geometry.
- **ST_XMin() –** Retrieves the minimum X coordinate value of a geometry.
- **ST_YMax() –** Retrieves the maximum Y coordinate value of a geometry.
- **ST_YMin() –** Retrieves the minimum Y coordinate value of a geometry.
- **ST_ZMax() –** Retrieves the maximum Z coordinate value of a geometry.
- **ST_ZMin() –** Retrieves the minimum Z coordinate value of a geometry.

Remarks

The ST_LineString type is a subtype of ST_Curve that uses 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.

Standards and compatibility

SQL/MM (ISO/IEC 13249-3: 2006) 7.2

ST_LineString(ST_Point , ST_Point , ST_Point) constructor

Constructs a linestring value from a list of points in a specified spatial reference system.

Syntax

NEW `ST_LineString`(ST_Point `pt1`, ST_Point `pt2`, ST_Point `pti`)

Parameters

- **pt1 –** The first point of the linestring.
- **pt2 –** The second point of the linestring.
- **pti –** Additional points of the linestring.

Returns

`ST_LineString` Returns a linestring constructed from the specified points.

Examples

- **Example 1 –** The following example returns the result `LineString (0 0, 1 1)`.

```
SELECT NEW ST_LineString( NEW ST_Point( 0, 0 ), NEW ST_Point( 1,
1 ) )
```

The following example returns the result `LineString (0 0, 1 1, 2 0)`.

```
SELECT NEW ST_LineString( NEW ST_Point( 0, 0 ), NEW ST_Point( 1,
1 ), NEW ST_Point(2,0) )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_LineString() constructor

Constructs a linestring representing the empty set.

Syntax

NEW `ST_LineString`()

Returns

`ST_LineString` Returns an ST_LineString value representing the empty set.

Examples

- **Example 1 –** The following returns 1, indicating the value is empty.

```
SELECT NEW ST_LineString().ST_IsEmpty()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Standard feature

ST_LineString(LONG BINARY[, INT]) constructor

Constructs a linestring from Well Known Binary (WKB).

Syntax

NEW `ST_LineString`(LONG BINARY `wkb`[, INT `srid`])

Parameters

- **wkb –** A string containing the binary representation of a linestring. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_LineString` Returns an ST_LineString value constructed from the source string.

Examples

- **Example 1 –** The following returns LineString (0 0, 5 10).

```
SELECT NEW
ST_LineString(0x0102000000020000000000000000000000000000000000000000
00000000000014400000000000002440)
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 7.2.2

ST_LineString(LONG VARCHAR[, INT]) constructor

Constructs a linestring from a text representation.

Syntax

NEW `ST_LineString`(LONG VARCHAR `text_representation`[, INT `srid`])

Parameters

- **text_representation –** A string containing the text representation of a linestring. The input can be in any supported text input format, including Well Known Text (WKT) or Extended Well Known Text (EWKT).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_LineString` Returns an ST_LineString value constructed from the source string.

Examples

- **Example 1 –** The following returns LineString (0 0, 5 10).

```
SELECT NEW ST_LineString('LineString (0 0, 5 10)')
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 7.2.2

ST_LineStringAggr(ST_Point) method

Returns a linestring built from the ordered points in a group.

Syntax

`ST_LineString`::`ST_LineStringAggr`(ST_Point `point`)

Parameters

- **point –** The points to generate the linestring. Typically this is a column.

Returns

`ST_LineString` Returns a linestring built from the points in a group.

Examples

- **Example 1 –** The following example returns the result `LineString (0 0, 2 0, 1 1)`.

```
BEGIN
DECLARE LOCAL TEMPORARY TABLE t_points( pk INT PRIMARY KEY,
p ST_Point );
INSERT INTO t_points VALUES( 1, 'Point( 0 0 )' );
INSERT INTO t_points VALUES( 2, 'Point( 2 0 )' );
INSERT INTO t_points VALUES( 3, 'Point( 1 1 )' );SELECT
ST_LineString::ST_LineStringAggr( p ORDER BY pk )
FROM t_points;
END
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

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.

Examples

- **Example 1 –** The following example returns the result `3`.

```
SELECT TREAT( Shape AS ST_LineString ).ST_NumPoints()
FROM SpatialShapes WHERE ShapeID = 5
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 7.2.4

ST_PointN(INT) method

Returns the nth point in the linestring.

Syntax

`linestring-expression.ST_PointN(INT n)`

Parameters

- **n –** The position of the element to return, from 1 to linestring-expression.ST_NumPoints().

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.

Examples

- **Example 1 –** The following example returns the result `Point (0 4)`.

```
SELECT TREAT( Shape AS ST_LineString ).ST_PointN( 2 )
FROM SpatialShapes WHERE ShapeID = 5
```

The following example returns one row for each point in geom.

```
BEGIN
DECLARE geom ST_LineString;
SET geom = NEW ST_LineString( 'LineString( 0 0, 1 0 )' );
SELECT row_num, geom.ST_PointN( row_num )
FROM sa_rowgenerator( 1, geom.ST_NumPoints() )
ORDER BY row_num;
END
```

The example returns the following result set:

row_num	geom.ST_PointN(row_num)
1	Point (0 0)
2	Point (1 0)

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 7.2.5

ST_MultiCurve type

An ST_MultiCurve is a collection of zero or more ST_Curve values, and all of the curves are within the spatial reference system.

Syntax

`ST_MultiCurve` type

Members

All members of the ST_MultiCurve type, including all inherited members.

Members of ST_MultiCurve:

- **ST_MultiCurve(ST_Curve , ST_Curve) –** Constructs a multi-curve from a list of curve values.
- **ST_MultiCurve() –** Constructs a multi curve representing the empty set.
- **ST_MultiCurve(LONG BINARY[, INT]) –** Constructs a multi curve from Well Known Binary (WKB).
- **ST_MultiCurve(LONG VARCHAR[, INT]) –** Constructs a multi curve from a text representation.
- **ST_IsClosed() –** Tests if the ST_MultiCurve value is closed. A curve is closed if the start and end points are coincident. A multicurve is closed if it is non-empty and has an empty boundary.
- **ST_Length(VARCHAR(128)) –** Returns the length measurement of the ST_MultiCurve value. The result is measured in the units specified by the parameter.
- **ST_MultiCurveAggr(ST_Curve) –** Returns a multicurve containing all of the curves in a group.

Members of ST_GeomCollection:

- **ST_GeomCollection(ST_Geometry , ST_Geometry) –** Constructs a geometry collection from a list of geometry values.
- **ST_GeomCollection() –** Constructs a geometry collection representing the empty set.
- **ST_GeomCollection(LONG BINARY[, INT]) –** Constructs a geometry collection from Well Known Binary (WKB).
- **ST_GeomCollection(LONG VARCHAR[, INT]) –** Constructs a geometry collection from a text representation.
- **ST_GeomCollectionAggr(ST_Geometry) –** Returns a geometry collection containing all of the geometries in a group.
- **ST_GeometryN(INT) –** Returns the nth geometry in the geometry collection.
- **ST_NumGeometries() –** Returns the number of geometries contained in the geometry collection.

Members of ST_Geometry:

- **ST_Affine(DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a new geometry that is the result of applying the specified 3-D affine transformation.
- **ST_AsBinary(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsBitmap(INT, INT, ST_Point , ST_Point , VARCHAR(128)) –** Returns a LONG VARBIT that is a bitmap representing a geometry value.
- **ST_AsGeoJSON(VARCHAR(128)) –** Returns a string representing a geometry in JSON format.
- **ST_AsGML(VARCHAR(128)) –** Returns the GML representation of an ST_Geometry value.
- **ST_AsKML(VARCHAR(128)) –** Returns the KML representation of an ST_Geometry value.
- **ST_AsSVG(VARCHAR(128)) –** Returns an SVG figure representing a geometry value.
- **ST_AsSVGAggr(ST_Geometry , VARCHAR(128)) –** Returns a complete or partial SVG document which renders the geometries in a group.
- **ST_AsText(VARCHAR(128)) –** Returns the text representation of an ST_Geometry value.
- **ST_AsWKB(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsWKT(VARCHAR(128)) –** Returns the WKT representation of an ST_Geometry value.
- **ST_AsXML(VARCHAR(128)) –** Returns the XML representation of an ST_Geometry value.
- **ST_Boundary() –** Returns the boundary of the geometry value.
- **ST_Buffer(DOUBLE, VARCHAR(128)) –** 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.
- **ST_Contains(ST_Geometry) –** Tests if a geometry value spatially contains another geometry value.
- **ST_ContainsFilter(ST_Geometry) –** An inexpensive test if a geometry might contain another.
- **ST_ConvexHull() –** Returns the convex hull of the geometry value.
- **ST_ConvexHullAggr(ST_Geometry) –** Returns the convex hull for all of the geometries in a group
- **ST_CoordDim() –** Returns the number of coordinate dimensions stored with each point of the ST_Geometry value.
- **ST_CoveredBy(ST_Geometry) –** Tests if a geometry value is spatially covered by another geometry value.
- **ST_CoveredByFilter(ST_Geometry) –** An inexpensive test if a geometry might be covered by another.

- **ST_Covers(ST_Geometry) –** Tests if a geometry value spatially covers another geometry value.
- **ST_CoversFilter(ST_Geometry) –** An inexpensive test if a geometry might cover another.
- **ST_Crosses(ST_Geometry) –** Tests if a geometry value crosses another geometry value.
- **ST_Debug(VARCHAR(128)) –** Returns a LONG BINARY that is debug information for the object.
- **ST_Difference(ST_Geometry) –** Returns the geometry value that represents the point set difference of two geometries.
- **ST_Dimension() –** Returns the dimension of the ST_Geometry value. Points have dimension 0, lines have dimension 1, and surfaces have dimension 2. Any empty geometry has dimension -1.
- **ST_Disjoint(ST_Geometry) –** Test if a geometry value is spatially disjoint from another value.
- **ST_Distance(ST_Geometry , VARCHAR(128)) –** Returns the smallest distance between the geometry-expression and the specified geometry value.
- **ST_Distance_Spheroid(ST_Geometry , VARCHAR(128)) –** Calculates the linear distance between geometries on the surface of the Earth.
- **ST_Envelope() –** Returns the bounding rectangle for the geometry value.
- **ST_EnvelopeAggr(ST_Geometry) –** Returns the bounding rectangle for all of the geometries in a group
- **ST_Equals(ST_Geometry) –** Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.
- **ST_EqualsFilter(ST_Geometry) –** An inexpensive test if a geometry is equal to another.
- **ST_GeometryType() –** Returns the name of the type of the ST_Geometry value.
- **ST_GeometryTypeFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_GeomFromBinary(LONG BINARY, INT) –** Constructs a geometry from a binary string representation.
- **ST_GeomFromShape(LONG BINARY[, INT]) –** Parses a string containing an ESRI shape record and creates a geometry value of the appropriate type.
- **ST_GeomFromText(LONG VARCHAR, INT) –** Constructs a geometry from a character string representation.
- **ST_GeomFromWKB(LONG BINARY, INT) –** Parse a string containing a WKB or EWKB representation of a geometry and creates a geometry value of the appropriate type.
- **ST_GeomFromWKT(LONG VARCHAR, INT) –** Parses a string containing the WKT or EWKT representation of a geometry and create a geometry value of the appropriate type.

- **ST_Intersection(ST_Geometry) –** Returns the geometry value that represents the point set intersection of two geometries.
- **ST_IntersectionAggr(ST_Geometry) –** Returns the spatial intersection of all of the geometries in a group
- **ST_Intersects(ST_Geometry) –** Test if a geometry value spatially intersects another value.
- **ST_IntersectsFilter(ST_Geometry) –** An inexpensive test if the two geometries might intersect.
- **ST_IntersectsRect(ST_Point , ST_Point) –** Test if a geometry intersects a rectangle.
- **ST_Is3D() –** Determines if the geometry value has Z coordinate values.
- **ST_IsEmpty() –** Determines whether the geometry value represents an empty set.
- **ST_IsMeasured() –** Determines if the geometry value has associated measure values.
- **ST_IsSimple() –** Determines whether the geometry value is simple (containing no self intersections or other irregularities).
- **ST_IsValid() –** Determines whether the geometry is a valid spatial object.
- **ST_LatNorth() –** Retrieves the northernmost latitude of a geometry.
- **ST_LatSouth() –** Retrieves the southernmost latitude of a geometry.
- **ST_Length_Spheroid(VARCHAR(128)) –** Calculates the linear length of a curve/multicurve on the surface of the Earth.
- **ST_LinearHash() –** Returns a binary string that is a linear hash of the geometry.
- **ST_LinearUnHash(BINARY(32)[, INT]) –** Returns a geometry representing the index hash.
- **ST_LoadConfigurationData(VARCHAR(128)) –** Returns binary configuration data. For internal use only.
- **ST_LocateAlong(DOUBLE) –** Returns the subset of the geometry value that is associated with the given measure value.
- **ST_LocateBetween(DOUBLE, DOUBLE) –** Returns the subset of the geometry value that is between the specified start measure and end measure.
- **ST_LongEast() –** Retrieves the longitude of the eastern boundary of a geometry.
- **ST_LongWest() –** Retrieves the longitude of the western boundary of a geometry.
- **ST_MMax() –** Retrieves the maximum M coordinate value of a geometry.
- **ST_MMin() –** Retrieves the minimum M coordinate value of a geometry.
- **ST_OrderingEquals(ST_Geometry) –** Tests if a geometry is identical to another geometry.
- **ST_Overlaps(ST_Geometry) –** Tests if a geometry value overlaps another geometry value.
- **ST_Relate(ST_Geometry) –** Determines how a geometry value is spatially related to another geometry value by returning an intersection matrix. The ST_Relate method returns 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.

- **ST_Reverse() –** Returns the geometry with the element order reversed.
- **ST_Segmentize(DOUBLE) –** Add points so that no line segment is longer than a specified distance.
- **ST_Simplify(DOUBLE) –** Remove points from curves so long as the maximum introduced error is less than a specified tolerance.
- **ST_SnapToGrid(ST_Point , DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a copy of the geometry with all points snapped to the specified grid.
- **ST_SRID(INT) –** Changes the spatial reference system associated with the geometry without modifying any of the values.
- **ST_SRIDFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_SymDifference(ST_Geometry) –** Returns the geometry value that represents the point set symmetric difference of two geometries.
- **ST_ToCircular() –** Convert the geometry to a circularstring
- **ST_ToCompound() –** Converts the geometry to a compound curve.
- **ST_ToCurve() –** Converts the geometry to a curve.
- **ST_ToCurvePoly() –** Converts the geometry to a curve polygon.
- **ST_ToGeomColl() –** Converts the geometry to a geometry collection.
- **ST_ToLineString() –** Converts the geometry to a linestring.
- **ST_ToMultiCurve() –** Converts the geometry to a multicurve value.
- **ST_ToMultiLine() –** Converts the geometry to a multilinestring value.
- **ST_ToMultiPoint() –** Converts the geometry to a multi-point value.
- **ST_ToMultiPolygon() –** Converts the geometry to a multi-polygon value.
- **ST_ToMultiSurface() –** Converts the geometry to a multi-surface value.
- **ST_ToPoint() –** Converts the geometry to a point.
- **ST_ToPolygon() –** Converts the geometry to a polygon.
- **ST_ToSurface() –** Converts the geometry to a surface.
- **ST_Touches(ST_Geometry) –** Tests if a geometry value spatially touches another geometry value.
- **ST_Transform(INT) –** Creates a copy of the geometry value transformed into the specified spatial reference system.
- **ST_Union(ST_Geometry) –** Returns the geometry value that represents the point set union of two geometries.
- **ST_UnionAggr(ST_Geometry) –** Returns the spatial union of all of the geometries in a group
- **ST_Within(ST_Geometry) –** Tests if a geometry value is spatially contained within another geometry value.
- **ST_WithinDistance(ST_Geometry , DOUBLE, VARCHAR(128)) –** Test if two geometries are within a specified distance of each other.

- **ST_WithinDistanceFilter(ST_Geometry , DOUBLE, VARCHAR(128)) –** An inexpensive of whether two geometries might be within a specified distance of each other.
- **ST_WithinFilter(ST_Geometry) –** An inexpensive test if a geometry might be within another.
- **ST_XMax() –** Retrieves the maximum X coordinate value of a geometry.
- **ST_XMin() –** Retrieves the minimum X coordinate value of a geometry.
- **ST_YMax() –** Retrieves the maximum Y coordinate value of a geometry.
- **ST_YMin() –** Retrieves the minimum Y coordinate value of a geometry.
- **ST_ZMax() –** Retrieves the maximum Z coordinate value of a geometry.
- **ST_ZMin() –** Retrieves the minimum Z coordinate value of a geometry.

Standards and compatibility
SQL/MM (ISO/IEC 13249-3: 2006) 9.3

ST_MultiCurve(ST_Curve , ST_Curve) constructor

Constructs a multi-curve from a list of curve values.

Syntax

NEW `ST_MultiCurve`(ST_Curve `curve1`, ST_Curve `curvei`)

Parameters

- **curve1 –** The first curve value of the multi-curve.
- **curvei –** Additional curve values of the multi-curve.

Returns

`ST_MultiCurve` A multi-curve containing the provided curve values.

Examples

- **Example 1 –** The following example returns the result `MultiCurve ((0 0, 1 1))`.

```
SELECT NEW ST_MultiCurve( NEW ST_LineString('LineString (0 0, 1
1)' ) )
```

The following example returns the result `MultiCurve ((0 0, 1 1), CircularString (0 0, 1 1, 2 0))`.

```
SELECT NEW ST_MultiCurve(
NEW ST_LineString('LineString (0 0, 1 1)' ),
NEW ST_CircularString( 'CircularString( 0 0, 1 1, 2 0)' ) )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_MultiCurve() constructor

Constructs a multi curve representing the empty set.

Syntax

NEW `ST_MultiCurve`()

Returns

`ST_MultiCurve` Returns an ST_MultiCurve value representing the empty set.

Examples

- **Example 1 –** The following returns 1, indicating the value is empty.

```
SELECT NEW ST_MultiCurve().ST_IsEmpty()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Standard feature

ST_MultiCurve(LONG BINARY[, INT]) constructor

Constructs a multi curve from Well Known Binary (WKB).

Syntax

NEW `ST_MultiCurve`(LONG BINARY `wkb`[, INT `srid`])

Parameters

- **wkb –** A string containing the binary representation of a multi curve. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_MultiCurve` Returns an ST_MultiCurve value constructed from the source string.

Examples

- **Example 1 –** The following returns MultiCurve (CircularString (5 10, 10 12, 15 10)).

```
SELECT NEW
ST_MultiCurve(0x010b0000000100000001080000000300000000000000000014400000000000024400000000000024400000000000028400000000000002e40000000000002440)
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 9.3.2

ST_MultiCurve(LONG VARCHAR[, INT]) constructor

Constructs a multi curve from a text representation.

Syntax

NEW `ST_MultiCurve`(LONG VARCHAR `text_representation`[, INT `srid`])

Parameters

- **text_representation –** A string containing the text representation of a multi curve. The input can be in any supported text input format, including Well Known Text (WKT) or Extended Well Known Text (EWKT).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_MultiCurve` Returns an ST_MultiCurve value constructed from the source string.

Examples

- **Example 1 –** The following returns MultiCurve ((10 10, 12 12), CircularString (5 10, 10 12, 15 10)).

```
SELECT NEW ST_MultiCurve('MultiCurve ((10 10, 12 12),
CircularString (5 10, 10 12, 15 10))')
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 9.3.2

ST_IsClosed() method

Tests if the ST_MultiCurve value is closed. A curve is closed if the start and end points are coincident. A multicurve is closed if it is non-empty and has an empty boundary.

Syntax

`multicurve-expression.ST_IsClosed`()

Returns

`BIT` Returns 1 if the multicurve is closed, otherwise 0.

Examples

- **Example 1 –** The following returns the result `0` because the boundary of the multicurve has two points.

```
SELECT NEW ST_MultiCurve( 'MultiCurve((0 0, 1 1))' ).ST_IsClosed()
```

  The following returns all rows in multicurve_table that have closed geometries. This example assumes the geometry column has type ST_MultiCurve or ST_MultiLineString.

```
SELECT * FROM multicurve_table WHERE geometry.ST_IsClosed() = 1
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 9.3.3

ST_Length(VARCHAR(128)) method

Returns the length measurement of the ST_MultiCurve value. The result is measured in the units specified by the parameter.

Syntax

`multicurve-expression.ST_Length`(VARCHAR(128) `unit_name`)

Parameters

- **unit_name –** The units in which the length 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 length measurement of the ST_MultiCurve value.

Examples

- **Example 1 –** The following example creates a multicurve and uses ST_Length to find the length of the geometry, returning the value PI+1.

```
SELECT NEW ST_MultiCurve(
NEW ST_LineString('LineString (0 0, 1 0)' ),
NEW ST_CircularString( 'CircularString( 0 0, 1 1, 2 0)' ) )
.ST_Length()
```

  The following example returns the name and length of all roads longer than 100 miles. This example assumes the road table exists, that the geometry column has type ST_MultiCurve or ST_MultiLineString, and the sa_install_feature system procedure has been used to load the st_geometry_predefined_uom.

```
SELECT name, geometry.ST_Length( 'Statute Mile' ) len
FROM roads WHERE len > 100
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 9.3.4

ST_MultiCurveAggr(ST_Curve) method

Returns a multicurve containing all of the curves in a group.

Syntax

`ST_MultiCurve::ST_MultiCurveAggr( ST_Curve geometry_column)`

Parameters

- **geometry_column –** The geometry values to generate the collection. Typically this is a column.

Returns

`ST_MultiCurve` Returns a multicurve that contains all of the geometries in a group.

Examples

- **Example 1 –** The following example returns a single value which combines all geometries of type ST_Curve from the SpatialShapes table into a single collection of type ST_MultiCurve. If the Shape column was of type ST_Curve then the TREAT function and WHERE clause would not be necessary.

```
SELECT ST_MultiCurve::ST_MultiCurveAggr( TREAT( Shape AS
ST_Curve ) )
FROM SpatialShapes WHERE Shape IS OF( ST_Curve )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

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.

Syntax

`ST_MultiLineString` type

Members

All members of the ST_MultiLineString type, including all inherited members.

Members of ST_MultiLineString:

- **ST_MultiLineString(ST_LineString , ST_LineString) –** Constructs a multi-linestring from a list of linestring values.
- **ST_MultiLineString() –** Constructs a multi linestring representing the empty set.
- **ST_MultiLineString(LONG BINARY[, INT]) –** Constructs a multi linestring from Well Known Binary (WKB).
- **ST_MultiLineString(LONG VARCHAR[, INT]) –** Constructs a multi linestring from a text representation.
- **ST_MultiLineStringAggr(ST_LineString) –** Returns a multilinestring containing all of the linestrings in a group.

Members of ST_GeomCollection:

- **ST_GeomCollection(ST_Geometry , ST_Geometry) –** Constructs a geometry collection from a list of geometry values.
- **ST_GeomCollection() –** Constructs a geometry collection representing the empty set.
- **ST_GeomCollection(LONG BINARY[, INT]) –** Constructs a geometry collection from Well Known Binary (WKB).
- **ST_GeomCollection(LONG VARCHAR[, INT]) –** Constructs a geometry collection from a text representation.
- **ST_GeomCollectionAggr(ST_Geometry) –** Returns a geometry collection containing all of the geometries in a group.
- **ST_GeometryN(INT) –** Returns the nth geometry in the geometry collection.
- **ST_NumGeometries() –** Returns the number of geometries contained in the geometry collection.

Members of ST_Geometry:

- **ST_Affine(DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a new geometry that is the result of applying the specified 3-D affine transformation.
- **ST_AsBinary(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsBitmap(INT, INT, ST_Point , ST_Point , VARCHAR(128)) –** Returns a LONG VARBIT that is a bitmap representing a geometry value.
- **ST_AsGeoJSON(VARCHAR(128)) –** Returns a string representing a geometry in JSON format.
- **ST_AsGML(VARCHAR(128)) –** Returns the GML representation of an ST_Geometry value.
- **ST_AsKML(VARCHAR(128)) –** Returns the KML representation of an ST_Geometry value.
- **ST_AsSVG(VARCHAR(128)) –** Returns an SVG figure representing a geometry value.
- **ST_AsSVGAggr(ST_Geometry , VARCHAR(128)) –** Returns a complete or partial SVG document which renders the geometries in a group.

- **ST_AsText(VARCHAR(128)) –** Returns the text representation of an ST_Geometry value.
- **ST_AsWKB(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsWKT(VARCHAR(128)) –** Returns the WKT representation of an ST_Geometry value.
- **ST_AsXML(VARCHAR(128)) –** Returns the XML representation of an ST_Geometry value.
- **ST_Boundary() –** Returns the boundary of the geometry value.
- **ST_Buffer(DOUBLE, VARCHAR(128)) –** 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.
- **ST_Contains(ST_Geometry) –** Tests if a geometry value spatially contains another geometry value.
- **ST_ContainsFilter(ST_Geometry) –** An inexpensive test if a geometry might contain another.
- **ST_ConvexHull() –** Returns the convex hull of the geometry value.
- **ST_ConvexHullAggr(ST_Geometry) –** Returns the convex hull for all of the geometries in a group
- **ST_CoordDim() –** Returns the number of coordinate dimensions stored with each point of the ST_Geometry value.
- **ST_CoveredBy(ST_Geometry) –** Tests if a geometry value is spatially covered by another geometry value.
- **ST_CoveredByFilter(ST_Geometry) –** An inexpensive test if a geometry might be covered by another.
- **ST_Covers(ST_Geometry) –** Tests if a geometry value spatially covers another geometry value.
- **ST_CoversFilter(ST_Geometry) –** An inexpensive test if a geometry might cover another.
- **ST_Crosses(ST_Geometry) –** Tests if a geometry value crosses another geometry value.
- **ST_Debug(VARCHAR(128)) –** Returns a LONG BINARY that is debug information for the object.
- **ST_Difference(ST_Geometry) –** Returns the geometry value that represents the point set difference of two geometries.
- **ST_Dimension() –** Returns the dimension of the ST_Geometry value. Points have dimension 0, lines have dimension 1, and surfaces have dimension 2. Any empty geometry has dimension -1.
- **ST_Disjoint(ST_Geometry) –** Test if a geometry value is spatially disjoint from another value.

- **ST_Distance(ST_Geometry , VARCHAR(128)) –** Returns the smallest distance between the geometry-expression and the specified geometry value.
- **ST_Distance_Spheroid(ST_Geometry , VARCHAR(128)) –** Calculates the linear distance between geometries on the surface of the Earth.
- **ST_Envelope() –** Returns the bounding rectangle for the geometry value.
- **ST_EnvelopeAggr(ST_Geometry) –** Returns the bounding rectangle for all of the geometries in a group
- **ST_Equals(ST_Geometry) –** Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.
- **ST_EqualsFilter(ST_Geometry) –** An inexpensive test if a geometry is equal to another.
- **ST_GeometryType() –** Returns the name of the type of the ST_Geometry value.
- **ST_GeometryTypeFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_GeomFromBinary(LONG BINARY, INT) –** Constructs a geometry from a binary string representation.
- **ST_GeomFromShape(LONG BINARY[, INT]) –** Parses a string containing an ESRI shape record and creates a geometry value of the appropriate type.
- **ST_GeomFromText(LONG VARCHAR, INT) –** Constructs a geometry from a character string representation.
- **ST_GeomFromWKB(LONG BINARY, INT) –** Parse a string containing a WKB or EWKB representation of a geometry and creates a geometry value of the appropriate type.
- **ST_GeomFromWKT(LONG VARCHAR, INT) –** Parses a string containing the WKT or EWKT representation of a geometry and create a geometry value of the appropriate type.
- **ST_Intersection(ST_Geometry) –** Returns the geometry value that represents the point set intersection of two geometries.
- **ST_IntersectionAggr(ST_Geometry) –** Returns the spatial intersection of all of the geometries in a group
- **ST_Intersects(ST_Geometry) –** Test if a geometry value spatially intersects another value.
- **ST_IntersectsFilter(ST_Geometry) –** An inexpensive test if the two geometries might intersect.
- **ST_IntersectsRect(ST_Point , ST_Point) –** Test if a geometry intersects a rectangle.
- **ST_Is3D() –** Determines if the geometry value has Z coordinate values.
- **ST_IsEmpty() –** Determines whether the geometry value represents an empty set.
- **ST_IsMeasured() –** Determines if the geometry value has associated measure values.
- **ST_IsSimple() –** Determines whether the geometry value is simple (containing no self intersections or other irregularities).
- **ST_IsValid() –** Determines whether the geometry is a valid spatial object.
- **ST_LatNorth() –** Retrieves the northernmost latitude of a geometry.

- **ST_LatSouth() –** Retrieves the southernmost latitude of a geometry.
- **ST_Length_Spheroid(VARCHAR(128)) –** Calculates the linear length of a curve/multicurve on the surface of the Earth.
- **ST_LinearHash() –** Returns a binary string that is a linear hash of the geometry.
- **ST_LinearUnHash(BINARY(32)[, INT]) –** Returns a geometry representing the index hash.
- **ST_LoadConfigurationData(VARCHAR(128)) –** Returns binary configuration data. For internal use only.
- **ST_LocateAlong(DOUBLE) –** Returns the subset of the geometry value that is associated with the given measure value.
- **ST_LocateBetween(DOUBLE, DOUBLE) –** Returns the subset of the geometry value that is between the specified start measure and end measure.
- **ST_LongEast() –** Retrieves the longitude of the eastern boundary of a geometry.
- **ST_LongWest() –** Retrieves the longitude of the western boundary of a geometry.
- **ST_MMax() –** Retrieves the maximum M coordinate value of a geometry.
- **ST_MMin() –** Retrieves the minimum M coordinate value of a geometry.
- **ST_OrderingEquals(ST_Geometry) –** Tests if a geometry is identical to another geometry.
- **ST_Overlaps(ST_Geometry) –** Tests if a geometry value overlaps another geometry value.
- **ST_Relate(ST_Geometry) –** Determines how a geometry value is spatially related to another geometry value by returning an intersection matrix. The ST_Relate method returns 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.
- **ST_Reverse() –** Returns the geometry with the element order reversed.
- **ST_Segmentize(DOUBLE) –** Add points so that no line segment is longer than a specified distance.
- **ST_Simplify(DOUBLE) –** Remove points from curves so long as the maximum introduced error is less than a specified tolerance.
- **ST_SnapToGrid(ST_Point , DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a copy of the geometry with all points snapped to the specified grid.
- **ST_SRID(INT) –** Changes the spatial reference system associated with the geometry without modifying any of the values.
- **ST_SRIDFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_SymDifference(ST_Geometry) –** Returns the geometry value that represents the point set symmetric difference of two geometries.
- **ST_ToCircular() –** Convert the geometry to a circularstring
- **ST_ToCompound() –** Converts the geometry to a compound curve.
- **ST_ToCurve() –** Converts the geometry to a curve.

- **ST_ToCurvePoly() –** Converts the geometry to a curve polygon.
- **ST_ToGeomColl() –** Converts the geometry to a geometry collection.
- **ST_ToLineString() –** Converts the geometry to a linestring.
- **ST_ToMultiCurve() –** Converts the geometry to a multicurve value.
- **ST_ToMultiLine() –** Converts the geometry to a multilinestring value.
- **ST_ToMultiPoint() –** Converts the geometry to a multi-point value.
- **ST_ToMultiPolygon() –** Converts the geometry to a multi-polygon value.
- **ST_ToMultiSurface() –** Converts the geometry to a multi-surface value.
- **ST_ToPoint() –** Converts the geometry to a point.
- **ST_ToPolygon() –** Converts the geometry to a polygon.
- **ST_ToSurface() –** Converts the geometry to a surface.
- **ST_Touches(ST_Geometry) –** Tests if a geometry value spatially touches another geometry value.
- **ST_Transform(INT) –** Creates a copy of the geometry value transformed into the specified spatial reference system.
- **ST_Union(ST_Geometry) –** Returns the geometry value that represents the point set union of two geometries.
- **ST_UnionAggr(ST_Geometry) –** Returns the spatial union of all of the geometries in a group
- **ST_Within(ST_Geometry) –** Tests if a geometry value is spatially contained within another geometry value.
- **ST_WithinDistance(ST_Geometry , DOUBLE, VARCHAR(128)) –** Test if two geometries are within a specified distance of each other.
- **ST_WithinDistanceFilter(ST_Geometry , DOUBLE, VARCHAR(128)) –** An inexpensive of whether two geometries might be within a specified distance of each other.
- **ST_WithinFilter(ST_Geometry) –** An inexpensive test if a geometry might be within another.
- **ST_XMax() –** Retrieves the maximum X coordinate value of a geometry.
- **ST_XMin() –** Retrieves the minimum X coordinate value of a geometry.
- **ST_YMax() –** Retrieves the maximum Y coordinate value of a geometry.
- **ST_YMin() –** Retrieves the minimum Y coordinate value of a geometry.
- **ST_ZMax() –** Retrieves the maximum Z coordinate value of a geometry.
- **ST_ZMin() –** Retrieves the minimum Z coordinate value of a geometry.

Members of ST_MultiCurve:

- **ST_MultiCurve(ST_Curve , ST_Curve) –** Constructs a multi-curve from a list of curve values.
- **ST_MultiCurve() –** Constructs a multi curve representing the empty set.
- **ST_MultiCurve(LONG BINARY[, INT]) –** Constructs a multi curve from Well Known Binary (WKB).

- **ST_MultiCurve(LONG VARCHAR[, INT]) –** Constructs a multi curve from a text representation.
- **ST_IsClosed() –** Tests if the ST_MultiCurve value is closed. A curve is closed if the start and end points are coincident. A multicurve is closed if it is non-empty and has an empty boundary.
- **ST_Length(VARCHAR(128)) –** Returns the length measurement of the ST_MultiCurve value. The result is measured in the units specified by the parameter.
- **ST_MultiCurveAggr(ST_Curve) –** Returns a multicurve containing all of the curves in a group.

Standards and compatibility
SQL/MM (ISO/IEC 13249-3: 2006) 9.4

ST_MultiLineString(ST_LineString , ST_LineString) constructor

Constructs a multi-linestring from a list of linestring values.

Syntax

NEW `ST_MultiLineString`(ST_LineString `linestring1`, ST_LineString `linestringi`)

Parameters

- **linestring1 –** The first linestring value of the multi-linestring.
- **linestringi –** Additional linestring values of the multi-linestring.

Returns

`ST_MultiLineString` A multi-linestring containing the provided linestring values.

Examples

- **Example 1 –** The following returns a multilinestring containing a single linestring and is equivalent to the following WKT:`'MultiLineString ((0 0, 1 1))'`

```
SELECT NEW ST_MultiLineString( NEW ST_LineString('LineString (0 0,
1 1)' ) )
```

The following returns a multilinestring containing two linestrings equivalent to the following WKT:`'MultiLineString ((0 0, 1 1), (0 0, 1 1, 2 0))'`.

```
SELECT NEW ST_MultiLineString(
NEW ST_LineString( 'LineString (0 0, 1 1)' ),
NEW ST_LineString( 'LineString (0 0, 1 1, 2 0)' ) )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_MultiLineString() constructor

Constructs a multi linestring representing the empty set.

Syntax

NEW `ST_MultiLineString`()

Returns

`ST_MultiLineString` Returns an ST_MultiLineString value representing the empty set.

Examples

- **Example 1 –** The following returns 1, indicating the value is empty.

```
SELECT NEW ST_MultiLineString().ST_IsEmpty()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Standard feature

ST_MultiLineString(LONG BINARY[, INT]) constructor

Constructs a multi linestring from Well Known Binary (WKB).

Syntax

NEW `ST_MultiLineString`(LONG BINARY `wkb`[, INT `srid`])

Parameters

- **wkb –** A string containing the binary representation of a multi linestring. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_MultiLineString` Returns an ST_MultiLineString value constructed from the source string.

Examples

- **Example 1 –** The following returns MultiLineString ((10 10, 12 12)).

```
SELECT NEW
ST_MultiLineString(0x01050000000100000001020000000200000000000000
00002440000000000000244000000000000028400000000000002840)
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 9.4.2

ST_MultiLineString(LONG VARCHAR[, INT]) constructor

Constructs a multi linestring from a text representation.

Syntax

NEW `ST_MultiLineString`(LONG VARCHAR `text_representation`[, INT `srid`])

Parameters

- **text_representation –** A string containing the text representation of a multi linestring. The input can be in any supported text input format, including Well Known Text (WKT) or Extended Well Known Text (EWKT).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_MultiLineString` Returns an ST_MultiLineString value constructed from the source string.

Examples

- **Example 1 –** The following returns MultiLineString ((10 10, 12 12), (14 10, 16 12)).

```
SELECT NEW ST_MultiLineString('MultiLineString ((10 10, 12 12),
(14 10, 16 12))')
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 9.4.2

ST_MultiLineStringAggr(ST_LineString) method

Returns a multilinestring containing all of the linestrings in a group.

Syntax

`ST_MultiLineString`::`ST_MultiLineStringAggr`(ST_LineString `geometry_column`)

Parameters

- **geometry_column –** The geometry values to generate the collection. Typically this is a column.

Returns

`ST_MultiLineString` Returns a multilinestring that contains all of the geometries in a group.

Examples

- **Example 1 –** The following example returns a single value which combines all geometries of type ST_LineString from the SpatialShapes table into a single collection of type ST_MultiLineString. If the Shape column was of type ST_LineString then the TREAT function and WHERE clause would not be necessary.

```
SELECT ST_MultiLineString::ST_MultiLineStringAggr( TREAT( Shape
AS ST_LineString ) )
FROM SpatialShapes WHERE Shape IS OF( ST_LineString )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

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.

Syntax

`ST_MultiPoint` type

Members

All members of the ST_MultiPoint type, including all inherited members.

Members of ST_MultiPoint:

- **ST_MultiPoint(ST_Point , ST_Point) –** Constructs a multi-point from a list of point values.
- **ST_MultiPoint() –** Constructs a multi point representing the empty set.
- **ST_MultiPoint(LONG BINARY[, INT]) –** Constructs a multi point from Well Known Binary (WKB).
- **ST_MultiPoint(LONG VARCHAR[, INT]) –** Constructs a multi point from a text representation.
- **ST_MultiPointAggr(ST_Point) –** Returns a multipoint containing all of the points in a group.

Members of ST_GeomCollection:

- **ST_GeomCollection(ST_Geometry , ST_Geometry) –** Constructs a geometry collection from a list of geometry values.

- **ST_GeomCollection() –** Constructs a geometry collection representing the empty set.
- **ST_GeomCollection(LONG BINARY[, INT]) –** Constructs a geometry collection from Well Known Binary (WKB).
- **ST_GeomCollection(LONG VARCHAR[, INT]) –** Constructs a geometry collection from a text representation.
- **ST_GeomCollectionAggr(ST_Geometry) –** Returns a geometry collection containing all of the geometries in a group.
- **ST_GeometryN(INT) –** Returns the nth geometry in the geometry collection.
- **ST_NumGeometries() –** Returns the number of geometries contained in the geometry collection.

Members of ST_Geometry:

- **ST_Affine(DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a new geometry that is the result of applying the specified 3-D affine transformation.
- **ST_AsBinary(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsBitmap(INT, INT, ST_Point , ST_Point , VARCHAR(128)) –** Returns a LONG VARBIT that is a bitmap representing a geometry value.
- **ST_AsGeoJSON(VARCHAR(128)) –** Returns a string representing a geometry in JSON format.
- **ST_AsGML(VARCHAR(128)) –** Returns the GML representation of an ST_Geometry value.
- **ST_AsKML(VARCHAR(128)) –** Returns the KML representation of an ST_Geometry value.
- **ST_AsSVG(VARCHAR(128)) –** Returns an SVG figure representing a geometry value.
- **ST_AsSVGAggr(ST_Geometry , VARCHAR(128)) –** Returns a complete or partial SVG document which renders the geometries in a group.
- **ST_AsText(VARCHAR(128)) –** Returns the text representation of an ST_Geometry value.
- **ST_AsWKB(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsWKT(VARCHAR(128)) –** Returns the WKT representation of an ST_Geometry value.
- **ST_AsXML(VARCHAR(128)) –** Returns the XML representation of an ST_Geometry value.
- **ST_Boundary() –** Returns the boundary of the geometry value.
- **ST_Buffer(DOUBLE, VARCHAR(128)) –** 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.

- **ST_Contains(ST_Geometry) –** Tests if a geometry value spatially contains another geometry value.
- **ST_ContainsFilter(ST_Geometry) –** An inexpensive test if a geometry might contain another.
- **ST_ConvexHull() –** Returns the convex hull of the geometry value.
- **ST_ConvexHullAggr(ST_Geometry) –** Returns the convex hull for all of the geometries in a group
- **ST_CoordDim() –** Returns the number of coordinate dimensions stored with each point of the ST_Geometry value.
- **ST_CoveredBy(ST_Geometry) –** Tests if a geometry value is spatially covered by another geometry value.
- **ST_CoveredByFilter(ST_Geometry) –** An inexpensive test if a geometry might be covered by another.
- **ST_Covers(ST_Geometry) –** Tests if a geometry value spatially covers another geometry value.
- **ST_CoversFilter(ST_Geometry) –** An inexpensive test if a geometry might cover another.
- **ST_Crosses(ST_Geometry) –** Tests if a geometry value crosses another geometry value.
- **ST_Debug(VARCHAR(128)) –** Returns a LONG BINARY that is debug information for the object.
- **ST_Difference(ST_Geometry) –** Returns the geometry value that represents the point set difference of two geometries.
- **ST_Dimension() –** Returns the dimension of the ST_Geometry value. Points have dimension 0, lines have dimension 1, and surfaces have dimension 2. Any empty geometry has dimension -1.
- **ST_Disjoint(ST_Geometry) –** Test if a geometry value is spatially disjoint from another value.
- **ST_Distance(ST_Geometry , VARCHAR(128)) –** Returns the smallest distance between the geometry-expression and the specified geometry value.
- **ST_Distance_Spheroid(ST_Geometry , VARCHAR(128)) –** Calculates the linear distance between geometries on the surface of the Earth.
- **ST_Envelope() –** Returns the bounding rectangle for the geometry value.
- **ST_EnvelopeAggr(ST_Geometry) –** Returns the bounding rectangle for all of the geometries in a group
- **ST_Equals(ST_Geometry) –** Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.
- **ST_EqualsFilter(ST_Geometry) –** An inexpensive test if a geometry is equal to another.
- **ST_GeometryType() –** Returns the name of the type of the ST_Geometry value.

- **ST_GeometryTypeFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_GeomFromBinary(LONG BINARY, INT) –** Constructs a geometry from a binary string representation.
- **ST_GeomFromShape(LONG BINARY[, INT]) –** Parses a string containing an ESRI shape record and creates a geometry value of the appropriate type.
- **ST_GeomFromText(LONG VARCHAR, INT) –** Constructs a geometry from a character string representation.
- **ST_GeomFromWKB(LONG BINARY, INT) –** Parse a string containing a WKB or EWKB representation of a geometry and creates a geometry value of the appropriate type.
- **ST_GeomFromWKT(LONG VARCHAR, INT) –** Parses a string containing the WKT or EWKT representation of a geometry and create a geometry value of the appropriate type.
- **ST_Intersection(ST_Geometry) –** Returns the geometry value that represents the point set intersection of two geometries.
- **ST_IntersectionAggr(ST_Geometry) –** Returns the spatial intersection of all of the geometries in a group
- **ST_Intersects(ST_Geometry) –** Test if a geometry value spatially intersects another value.
- **ST_IntersectsFilter(ST_Geometry) –** An inexpensive test if the two geometries might intersect.
- **ST_IntersectsRect(ST_Point , ST_Point) –** Test if a geometry intersects a rectangle.
- **ST_Is3D() –** Determines if the geometry value has Z coordinate values.
- **ST_IsEmpty() –** Determines whether the geometry value represents an empty set.
- **ST_IsMeasured() –** Determines if the geometry value has associated measure values.
- **ST_IsSimple() –** Determines whether the geometry value is simple (containing no self intersections or other irregularities).
- **ST_IsValid() –** Determines whether the geometry is a valid spatial object.
- **ST_LatNorth() –** Retrieves the northernmost latitude of a geometry.
- **ST_LatSouth() –** Retrieves the southernmost latitude of a geometry.
- **ST_Length_Spheroid(VARCHAR(128)) –** Calculates the linear length of a curve/multicurve on the surface of the Earth.
- **ST_LinearHash() –** Returns a binary string that is a linear hash of the geometry.
- **ST_LinearUnHash(BINARY(32)[, INT]) –** Returns a geometry representing the index hash.
- **ST_LoadConfigurationData(VARCHAR(128)) –** Returns binary configuration data. For internal use only.
- **ST_LocateAlong(DOUBLE) –** Returns the subset of the geometry value that is associated with the given measure value.
- **ST_LocateBetween(DOUBLE, DOUBLE) –** Returns the subset of the geometry value that is between the specified start measure and end measure.

- **ST_LongEast() –** Retrieves the longitude of the eastern boundary of a geometry.
- **ST_LongWest() –** Retrieves the longitude of the western boundary of a geometry.
- **ST_MMax() –** Retrieves the maximum M coordinate value of a geometry.
- **ST_MMin() –** Retrieves the minimum M coordinate value of a geometry.
- **ST_OrderingEquals(ST_Geometry) –** Tests if a geometry is identical to another geometry.
- **ST_Overlaps(ST_Geometry) –** Tests if a geometry value overlaps another geometry value.
- **ST_Relate(ST_Geometry) –** Determines how a geometry value is spatially related to another geometry value by returning an intersection matrix. The ST_Relate method returns 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.
- **ST_Reverse() –** Returns the geometry with the element order reversed.
- **ST_Segmentize(DOUBLE) –** Add points so that no line segment is longer than a specified distance.
- **ST_Simplify(DOUBLE) –** Remove points from curves so long as the maximum introduced error is less than a specified tolerance.
- **ST_SnapToGrid(ST_Point , DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a copy of the geometry with all points snapped to the specified grid.
- **ST_SRID(INT) –** Changes the spatial reference system associated with the geometry without modifying any of the values.
- **ST_SRIDFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_SymDifference(ST_Geometry) –** Returns the geometry value that represents the point set symmetric difference of two geometries.
- **ST_ToCircular() –** Convert the geometry to a circularstring
- **ST_ToCompound() –** Converts the geometry to a compound curve.
- **ST_ToCurve() –** Converts the geometry to a curve.
- **ST_ToCurvePoly() –** Converts the geometry to a curve polygon.
- **ST_ToGeomColl() –** Converts the geometry to a geometry collection.
- **ST_ToLineString() –** Converts the geometry to a linestring.
- **ST_ToMultiCurve() –** Converts the geometry to a multicurve value.
- **ST_ToMultiLine() –** Converts the geometry to a multilinestring value.
- **ST_ToMultiPoint() –** Converts the geometry to a multi-point value.
- **ST_ToMultiPolygon() –** Converts the geometry to a multi-polygon value.
- **ST_ToMultiSurface() –** Converts the geometry to a multi-surface value.
- **ST_ToPoint() –** Converts the geometry to a point.
- **ST_ToPolygon() –** Converts the geometry to a polygon.
- **ST_ToSurface() –** Converts the geometry to a surface.

- **ST_Touches(ST_Geometry) –** Tests if a geometry value spatially touches another geometry value.
- **ST_Transform(INT) –** Creates a copy of the geometry value transformed into the specified spatial reference system.
- **ST_Union(ST_Geometry) –** Returns the geometry value that represents the point set union of two geometries.
- **ST_UnionAggr(ST_Geometry) –** Returns the spatial union of all of the geometries in a group
- **ST_Within(ST_Geometry) –** Tests if a geometry value is spatially contained within another geometry value.
- **ST_WithinDistance(ST_Geometry , DOUBLE, VARCHAR(128)) –** Test if two geometries are within a specified distance of each other.
- **ST_WithinDistanceFilter(ST_Geometry , DOUBLE, VARCHAR(128)) –** An inexpensive of whether two geometries might be within a specified distance of each other.
- **ST_WithinFilter(ST_Geometry) –** An inexpensive test if a geometry might be within another.
- **ST_XMax() –** Retrieves the maximum X coordinate value of a geometry.
- **ST_XMin() –** Retrieves the minimum X coordinate value of a geometry.
- **ST_YMax() –** Retrieves the maximum Y coordinate value of a geometry.
- **ST_YMin() –** Retrieves the minimum Y coordinate value of a geometry.
- **ST_ZMax() –** Retrieves the maximum Z coordinate value of a geometry.
- **ST_ZMin() –** Retrieves the minimum Z coordinate value of a geometry.

Standards and compatibility
SQL/MM (ISO/IEC 13249-3: 2006) 9.2

ST_MultiPoint(ST_Point , ST_Point) constructor

Constructs a multi-point from a list of point values.

Syntax

NEW `ST_MultiPoint`(ST_Point `point1`, ST_Point `pointi`)

Parameters

- **point1 –** The first point value of the multi-point.
- **pointi –** Additional point values of the multi-point.

Returns

`ST_MultiPoint` A multi-point containing the provided point values.

Examples

- **Example 1 –** The following returns a multi-point containing the single point `'Point (1 2)'`.

```
SELECT NEW ST_MultiPoint( NEW ST_Point( 1.0, 2.0 ) )
```

The following returns a multi-point containing two points `'Point (1 2)'` and `'Point (3 4)'`.

```
SELECT NEW ST_MultiPoint( NEW ST_Point( 1.0, 2.0 ), NEW
ST_Point( 3.0, 4.0 ) )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_MultiPoint() constructor

Constructs a multi point representing the empty set.

Syntax

NEW `ST_MultiPoint`()

Returns

`ST_MultiPoint` Returns an ST_MultiPoint value representing the empty set.

Examples

- **Example 1 –** The following returns 1, indicating the value is empty.

```
SELECT NEW ST_MultiPoint().ST_IsEmpty()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Standard feature

ST_MultiPoint(LONG BINARY[, INT]) constructor

Constructs a multi point from Well Known Binary (WKB).

Syntax

NEW `ST_MultiPoint`(LONG BINARY `wkb`[, INT `srid`])

Parameters

- **wkb –** A string containing the binary representation of a multi point. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).

- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_MultiPoint` Returns an ST_MultiPoint value constructed from the source string.

Examples

- **Example 1 –** The following returns MultiPoint ((10 10), (12 12), (14 10)).

```
SELECT NEW
ST_MultiPoint(0x01040000000300000001010000000000000000244000000
000000024400101000000000000000000284000000000000028400101000000000
00000000002c400000000000002440)
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 9.2.2

ST_MultiPoint(LONG VARCHAR[, INT]) constructor

Constructs a multi point from a text representation.

Syntax

NEW `ST_MultiPoint`(LONG VARCHAR `text_representation`[, INT `srid`])

Parameters

- **text_representation –** A string containing the text representation of a multi point. The input can be in any supported text input format, including Well Known Text (WKT) or Extended Well Known Text (EWKT).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_MultiPoint` Returns an ST_MultiPoint value constructed from the source string.

Examples

- **Example 1 –** The following returns MultiPoint ((10 10), (12 12), (14 10)).

```
SELECT NEW ST_MultiPoint('MultiPoint ((10 10), (12 12), (14 10))')
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 9.2.2

ST_MultiPointAggr(ST_Point) method

Returns a multipoint containing all of the points in a group.

Syntax

```
ST_MultiPoint::ST_MultiPointAggr( ST_Point geometry_column)
```

Parameters

- **geometry_column –** The geometry values to generate the collection. Typically this is a column.

Returns

`ST_MultiPoint` Returns a multipoint that contains all of the geometries in a group.

Examples

- **Example 1 –** The following example returns a single value which combines all geometries of type ST_Point from the SpatialShapes table into a single collection of type ST_MultiPoint. If the Shape column was of type ST_Point then the TREAT function and WHERE clause would not be necessary.

```
SELECT ST_MultiPoint::ST_MultiPointAggr( TREAT( Shape AS
ST_Point ) )
FROM SpatialShapes WHERE Shape IS OF( ST_Point )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_MultiPolygon type

An ST_MultiPolygon is a collection of zero or more ST_Polygon values, and all of the polygons are within the spatial reference system.

Syntax

`ST_MultiPolygon` type

Members

All members of the ST_MultiPolygon type, including all inherited members.

Members of ST_MultiPolygon:

- **ST_MultiPolygon(ST_MultiLineString , VARCHAR(128)) –** Creates a multi-polygon from a multilinestring containing exterior rings and an optional list of interior rings.
- **ST_MultiPolygon(ST_Polygon , ST_Polygon) –** Constructs a multi-polygon from a list of polygon values.
- **ST_MultiPolygon() –** Constructs a multi polygon representing the empty set.
- **ST_MultiPolygon(LONG BINARY[, INT]) –** Constructs a multi polygon from Well Known Binary (WKB).
- **ST_MultiPolygon(LONG VARCHAR[, INT]) –** Constructs a multi polygon from a text representation.
- **ST_MultiPolygonAggr(ST_Polygon) –** Returns a multipolygon containing all of the polygons in a group.

Members of ST_GeomCollection:

- **ST_GeomCollection(ST_Geometry , ST_Geometry) –** Constructs a geometry collection from a list of geometry values.
- **ST_GeomCollection() –** Constructs a geometry collection representing the empty set.
- **ST_GeomCollection(LONG BINARY[, INT]) –** Constructs a geometry collection from Well Known Binary (WKB).
- **ST_GeomCollection(LONG VARCHAR[, INT]) –** Constructs a geometry collection from a text representation.
- **ST_GeomCollectionAggr(ST_Geometry) –** Returns a geometry collection containing all of the geometries in a group.
- **ST_GeometryN(INT) –** Returns the nth geometry in the geometry collection.
- **ST_NumGeometries() –** Returns the number of geometries contained in the geometry collection.

Members of ST_Geometry:

- **ST_Affine(DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a new geometry that is the result of applying the specified 3-D affine transformation.
- **ST_AsBinary(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsBitmap(INT, INT, ST_Point , ST_Point , VARCHAR(128)) –** Returns a LONG VARBIT that is a bitmap representing a geometry value.
- **ST_AsGeoJSON(VARCHAR(128)) –** Returns a string representing a geometry in JSON format.
- **ST_AsGML(VARCHAR(128)) –** Returns the GML representation of an ST_Geometry value.
- **ST_AsKML(VARCHAR(128)) –** Returns the KML representation of an ST_Geometry value.

- **ST_AsSVG(VARCHAR(128)) –** Returns an SVG figure representing a geometry value.
- **ST_AsSVGAggr(ST_Geometry , VARCHAR(128)) –** Returns a complete or partial SVG document which renders the geometries in a group.
- **ST_AsText(VARCHAR(128)) –** Returns the text representation of an ST_Geometry value.
- **ST_AsWKB(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsWKT(VARCHAR(128)) –** Returns the WKT representation of an ST_Geometry value.
- **ST_AsXML(VARCHAR(128)) –** Returns the XML representation of an ST_Geometry value.
- **ST_Boundary() –** Returns the boundary of the geometry value.
- **ST_Buffer(DOUBLE, VARCHAR(128)) –** 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.
- **ST_Contains(ST_Geometry) –** Tests if a geometry value spatially contains another geometry value.
- **ST_ContainsFilter(ST_Geometry) –** An inexpensive test if a geometry might contain another.
- **ST_ConvexHull() –** Returns the convex hull of the geometry value.
- **ST_ConvexHullAggr(ST_Geometry) –** Returns the convex hull for all of the geometries in a group
- **ST_CoordDim() –** Returns the number of coordinate dimensions stored with each point of the ST_Geometry value.
- **ST_CoveredBy(ST_Geometry) –** Tests if a geometry value is spatially covered by another geometry value.
- **ST_CoveredByFilter(ST_Geometry) –** An inexpensive test if a geometry might be covered by another.
- **ST_Covers(ST_Geometry) –** Tests if a geometry value spatially covers another geometry value.
- **ST_CoversFilter(ST_Geometry) –** An inexpensive test if a geometry might cover another.
- **ST_Crosses(ST_Geometry) –** Tests if a geometry value crosses another geometry value.
- **ST_Debug(VARCHAR(128)) –** Returns a LONG BINARY that is debug information for the object.
- **ST_Difference(ST_Geometry) –** Returns the geometry value that represents the point set difference of two geometries.
- **ST_Dimension() –** Returns the dimension of the ST_Geometry value. Points have dimension 0, lines have dimension 1, and surfaces have dimension 2. Any empty geometry has dimension -1.

- **ST_Disjoint(ST_Geometry) –** Test if a geometry value is spatially disjoint from another value.
- **ST_Distance(ST_Geometry , VARCHAR(128)) –** Returns the smallest distance between the geometry-expression and the specified geometry value.
- **ST_Distance_Spheroid(ST_Geometry , VARCHAR(128)) –** Calculates the linear distance between geometries on the surface of the Earth.
- **ST_Envelope() –** Returns the bounding rectangle for the geometry value.
- **ST_EnvelopeAggr(ST_Geometry) –** Returns the bounding rectangle for all of the geometries in a group
- **ST_Equals(ST_Geometry) –** Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.
- **ST_EqualsFilter(ST_Geometry) –** An inexpensive test if a geometry is equal to another.
- **ST_GeometryType() –** Returns the name of the type of the ST_Geometry value.
- **ST_GeometryTypeFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_GeomFromBinary(LONG BINARY, INT) –** Constructs a geometry from a binary string representation.
- **ST_GeomFromShape(LONG BINARY[, INT]) –** Parses a string containing an ESRI shape record and creates a geometry value of the appropriate type.
- **ST_GeomFromText(LONG VARCHAR, INT) –** Constructs a geometry from a character string representation.
- **ST_GeomFromWKB(LONG BINARY, INT) –** Parse a string containing a WKB or EWKB representation of a geometry and creates a geometry value of the appropriate type.
- **ST_GeomFromWKT(LONG VARCHAR, INT) –** Parses a string containing the WKT or EWKT representation of a geometry and create a geometry value of the appropriate type.
- **ST_Intersection(ST_Geometry) –** Returns the geometry value that represents the point set intersection of two geometries.
- **ST_IntersectionAggr(ST_Geometry) –** Returns the spatial intersection of all of the geometries in a group
- **ST_Intersects(ST_Geometry) –** Test if a geometry value spatially intersects another value.
- **ST_IntersectsFilter(ST_Geometry) –** An inexpensive test if the two geometries might intersect.
- **ST_IntersectsRect(ST_Point , ST_Point) –** Test if a geometry intersects a rectangle.
- **ST_Is3D() –** Determines if the geometry value has Z coordinate values.
- **ST_IsEmpty() –** Determines whether the geometry value represents an empty set.
- **ST_IsMeasured() –** Determines if the geometry value has associated measure values.
- **ST_IsSimple() –** Determines whether the geometry value is simple (containing no self intersections or other irregularities).

- **ST_IsValid() –** Determines whether the geometry is a valid spatial object.
- **ST_LatNorth() –** Retrieves the northernmost latitude of a geometry.
- **ST_LatSouth() –** Retrieves the southernmost latitude of a geometry.
- **ST_Length_Spheroid(VARCHAR(128)) –** Calculates the linear length of a curve/multicurve on the surface of the Earth.
- **ST_LinearHash() –** Returns a binary string that is a linear hash of the geometry.
- **ST_LinearUnHash(BINARY(32)[, INT]) –** Returns a geometry representing the index hash.
- **ST_LoadConfigurationData(VARCHAR(128)) –** Returns binary configuration data. For internal use only.
- **ST_LocateAlong(DOUBLE) –** Returns the subset of the geometry value that is associated with the given measure value.
- **ST_LocateBetween(DOUBLE, DOUBLE) –** Returns the subset of the geometry value that is between the specified start measure and end measure.
- **ST_LongEast() –** Retrieves the longitude of the eastern boundary of a geometry.
- **ST_LongWest() –** Retrieves the longitude of the western boundary of a geometry.
- **ST_MMax() –** Retrieves the maximum M coordinate value of a geometry.
- **ST_MMin() –** Retrieves the minimum M coordinate value of a geometry.
- **ST_OrderingEquals(ST_Geometry) –** Tests if a geometry is identical to another geometry.
- **ST_Overlaps(ST_Geometry) –** Tests if a geometry value overlaps another geometry value.
- **ST_Relate(ST_Geometry) –** Determines how a geometry value is spatially related to another geometry value by returning an intersection matrix. The ST_Relate method returns 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.
- **ST_Reverse() –** Returns the geometry with the element order reversed.
- **ST_Segmentize(DOUBLE) –** Add points so that no line segment is longer than a specified distance.
- **ST_Simplify(DOUBLE) –** Remove points from curves so long as the maximum introduced error is less than a specified tolerance.
- **ST_SnapToGrid(ST_Point , DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a copy of the geometry with all points snapped to the specified grid.
- **ST_SRID(INT) –** Changes the spatial reference system associated with the geometry without modifying any of the values.
- **ST_SRIDFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_SymDifference(ST_Geometry) –** Returns the geometry value that represents the point set symmetric difference of two geometries.
- **ST_ToCircular() –** Convert the geometry to a circularstring

- **ST_ToCompound() –** Converts the geometry to a compound curve.
- **ST_ToCurve() –** Converts the geometry to a curve.
- **ST_ToCurvePoly() –** Converts the geometry to a curve polygon.
- **ST_ToGeomColl() –** Converts the geometry to a geometry collection.
- **ST_ToLineString() –** Converts the geometry to a linestring.
- **ST_ToMultiCurve() –** Converts the geometry to a multicurve value.
- **ST_ToMultiLine() –** Converts the geometry to a multilinestring value.
- **ST_ToMultiPoint() –** Converts the geometry to a multi-point value.
- **ST_ToMultiPolygon() –** Converts the geometry to a multi-polygon value.
- **ST_ToMultiSurface() –** Converts the geometry to a multi-surface value.
- **ST_ToPoint() –** Converts the geometry to a point.
- **ST_ToPolygon() –** Converts the geometry to a polygon.
- **ST_ToSurface() –** Converts the geometry to a surface.
- **ST_Touches(ST_Geometry) –** Tests if a geometry value spatially touches another geometry value.
- **ST_Transform(INT) –** Creates a copy of the geometry value transformed into the specified spatial reference system.
- **ST_Union(ST_Geometry) –** Returns the geometry value that represents the point set union of two geometries.
- **ST_UnionAggr(ST_Geometry) –** Returns the spatial union of all of the geometries in a group
- **ST_Within(ST_Geometry) –** Tests if a geometry value is spatially contained within another geometry value.
- **ST_WithinDistance(ST_Geometry , DOUBLE, VARCHAR(128)) –** Test if two geometries are within a specified distance of each other.
- **ST_WithinDistanceFilter(ST_Geometry , DOUBLE, VARCHAR(128)) –** An inexpensive of whether two geometries might be within a specified distance of each other.
- **ST_WithinFilter(ST_Geometry) –** An inexpensive test if a geometry might be within another.
- **ST_XMax() –** Retrieves the maximum X coordinate value of a geometry.
- **ST_XMin() –** Retrieves the minimum X coordinate value of a geometry.
- **ST_YMax() –** Retrieves the maximum Y coordinate value of a geometry.
- **ST_YMin() –** Retrieves the minimum Y coordinate value of a geometry.
- **ST_ZMax() –** Retrieves the maximum Z coordinate value of a geometry.
- **ST_ZMin() –** Retrieves the minimum Z coordinate value of a geometry.

Members of ST_MultiSurface:

- **ST_MultiSurface(ST_MultiCurve , VARCHAR(128)) –** Creates a multi-surface from a multicurve containing exterior rings and an optional list of interior rings.

- **ST_MultiSurface(ST_Surface , ST_Surface) –** Constructs a multi-surface from a list of surface values.
- **ST_MultiSurface() –** Constructs a multi surface representing the empty set.
- **ST_MultiSurface(LONG BINARY[, INT]) –** Constructs a multi surface from Well Known Binary (WKB).
- **ST_MultiSurface(LONG VARCHAR[, INT]) –** Constructs a multi surface from a text representation.
- **ST_Area(VARCHAR(128)) –** Computes the area of the multi-surface in the specified units.
- **ST_Centroid() –** Computes the ST_Point that is the mathematical centroid of the multi-surface.
- **ST_MultiSurfaceAggr(ST_Surface) –** Returns a multisurface containing all of the surfaces in a group.
- **ST_Perimeter(VARCHAR(128)) –** Computes the perimeter of the multi-surface in the specified units.
- **ST_PointOnSurface() –** Returns a point that is guaranteed to be on a surface in the multi-surface

Standards and compatibility
SQL/MM (ISO/IEC 13249-3: 2006) 9.6

ST_MultiPolygon(ST_MultiLineString , VARCHAR(128)) constructor

Creates a multi-polygon from a multilinestring containing exterior rings and an optional list of interior rings.

Syntax

NEW `ST_MultiPolygon`(ST_MultiLineString `multi_linestring`, VARCHAR(128) `polygon_format`)

Parameters

- **multi_linestring –** A multilinestring value containing exterior rings and (optionally) a set of interior rings.
- **polygon_format –** A string with the polygon format to use when interpreting the provided linestrings. Valid formats are 'CounterClockwise', 'Clockwise', and 'EvenOdd'

Returns

`ST_MultiPolygon` Returns a multi-polygon from the specified multilinestring.

Examples

- **Example 1 –** The following returns MultiPolygon (((-4 -4, 4 -4, 4 4, -4 4, -4 -4), (-2 1, -3 3, -1 3, -2 1)), ((6 -4, 14 -4, 14 4, 6 4, 6 -4), (8 1, 7 3, 9 3, 8 1))) (two square polygons each with a triangular hole).

```
SELECT NEW ST_MultiPolygon(
NEW ST_MultiLineString ('MultiLineString ((-4 -4, 4 -4, 4 4, -4 4,
-4 -4), (-2 1, -3 3, -1 3, -2 1), (6 -4, 14 -4, 14 4, 6 4, 6 -4), (8
1, 7 3, 9 3, 8 1))'))
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_MultiPolygon(ST_Polygon , ST_Polygon) constructor

Constructs a multi-polygon from a list of polygon values.

Syntax

NEW `ST_MultiPolygon`(ST_Polygon `polygon1`, ST_Polygon `polygoni`)

Parameters

- **polygon1 –** The first polygon value of the multi-polygon.
- **polygoni –** Additional polygon values of the multi-polygon.

Returns

`ST_MultiPolygon` A multi-polygon containing the provided polygon values.

Examples

- **Example 1 –** The following example returns the result `MultiPolygon (((0 0, 1 0, 1 1, 0 1, 0 0)))`.

```
SELECT NEW ST_MultiPolygon( NEW ST_Polygon('Polygon ((0 0, 0 1, 1
1, 1 0, 0 0))' )  )
```

  The following example returns the result `MultiPolygon (((0 0, 1 0, 1 1, 0 1, 0 0)), ((5 5, 10 5, 10 10, 5 10, 5 5)))`.

```
SELECT NEW ST_MultiPolygon(
NEW ST_Polygon('Polygon ((0 0, 0 1, 1 1, 1 0, 0 0))' ),
NEW ST_Polygon('Polygon ((5 5, 5 10, 10 10, 10 5, 5 5))' )  )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_MultiPolygon() constructor

Constructs a multi polygon representing the empty set.

Syntax

NEW `ST_MultiPolygon`()

Returns

`ST_MultiPolygon` Returns an ST_MultiPolygon value representing the empty set.

Examples

- **Example 1 –** The following returns 1, indicating the value is empty.

```
SELECT NEW ST_MultiPolygon().ST_IsEmpty()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Standard feature

ST_MultiPolygon(LONG BINARY[, INT]) constructor

Constructs a multi polygon from Well Known Binary (WKB).

Syntax

NEW `ST_MultiPolygon`(LONG BINARY `wkb`[, INT `srid`])

Parameters

- **wkb –** A string containing the binary representation of a multi polygon. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_MultiPolygon` Returns an ST_MultiPolygon value constructed from the source string.

Examples

- **Example 1 –** The following returns MultiPolygon (((10 -5, 15 5, 5 5, 10 -5))).

```
SELECT NEW
ST_MultiPolygon(0x010600000001000000010300000001000000040000000000
0000000024400000000000000014c0000000000000002e40000000000000014400000
0000000014400000000000000144000000000000002440000000000000014c0)
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 9.6.2

ST_MultiPolygon(LONG VARCHAR[, INT]) constructor

Constructs a multi polygon from a text representation.

Syntax

NEW `ST_MultiPolygon`(LONG VARCHAR `text_representation`[, INT `srid`])

Parameters

- **text_representation –** A string containing the text representation of a multi polygon. The input can be in any supported text input format, including Well Known Text (WKT) or Extended Well Known Text (EWKT).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_MultiPolygon` Returns an ST_MultiPolygon value constructed from the source string.

Examples

- **Example 1 –** 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)))')
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 9.6.2

ST_MultiPolygonAggr(ST_Polygon) method

Returns a multipolygon containing all of the polygons in a group.

Syntax

`ST_MultiPolygon`::`ST_MultiPolygonAggr`(ST_Polygon `geometry_column`)

Parameters

- **geometry_column –** The geometry values to generate the collection. Typically this is a column.

Returns

`ST_MultiPolygon` Returns a multipolygon that contains all of the geometries in a group.

Examples

- **Example 1 –** The following example returns a single value which combines all geometries of type ST_Polygon from the SpatialShapes table into a single collection of type ST_MultiPolygon. If the Shape column was of type ST_Polygon then the TREAT function and WHERE clause would not be necessary.

```
SELECT ST_MultiPolygon::ST_MultiPolygonAggr( TREAT( Shape AS
ST_Polygon ) )
FROM SpatialShapes WHERE Shape IS OF( ST_Polygon )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_MultiSurface type

An ST_MultiSurface is a collection of zero or more ST_Surface values, and all of the surfaces are within the spatial reference system.

Syntax

`ST_MultiSurface` type

Members

All members of the ST_MultiSurface type, including all inherited members.

Members of ST_MultiSurface:

- **ST_MultiSurface(ST_MultiCurve , VARCHAR(128)) –** Creates a multi-surface from a multicurve containing exterior rings and an optional list of interior rings.
- **ST_MultiSurface(ST_Surface , ST_Surface) –** Constructs a multi-surface from a list of surface values.
- **ST_MultiSurface() –** Constructs a multi surface representing the empty set.
- **ST_MultiSurface(LONG BINARY[, INT]) –** Constructs a multi surface from Well Known Binary (WKB).
- **ST_MultiSurface(LONG VARCHAR[, INT]) –** Constructs a multi surface from a text representation.
- **ST_Area(VARCHAR(128)) –** Computes the area of the multi-surface in the specified units.
- **ST_Centroid() –** Computes the ST_Point that is the mathematical centroid of the multi-surface.

- **ST_MultiSurfaceAggr(ST_Surface) –** Returns a multisurface containing all of the surfaces in a group.
- **ST_Perimeter(VARCHAR(128)) –** Computes the perimeter of the multi-surface in the specified units.
- **ST_PointOnSurface() –** Returns a point that is guaranteed to be on a surface in the multi-surface

Members of ST_GeomCollection:

- **ST_GeomCollection(ST_Geometry , ST_Geometry) –** Constructs a geometry collection from a list of geometry values.
- **ST_GeomCollection() –** Constructs a geometry collection representing the empty set.
- **ST_GeomCollection(LONG BINARY[, INT]) –** Constructs a geometry collection from Well Known Binary (WKB).
- **ST_GeomCollection(LONG VARCHAR[, INT]) –** Constructs a geometry collection from a text representation.
- **ST_GeomCollectionAggr(ST_Geometry) –** Returns a geometry collection containing all of the geometries in a group.
- **ST_GeometryN(INT) –** Returns the nth geometry in the geometry collection.
- **ST_NumGeometries() –** Returns the number of geometries contained in the geometry collection.

Members of ST_Geometry:

- **ST_Affine(DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a new geometry that is the result of applying the specified 3-D affine transformation.
- **ST_AsBinary(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsBitmap(INT, INT, ST_Point , ST_Point , VARCHAR(128)) –** Returns a LONG VARBIT that is a bitmap representing a geometry value.
- **ST_AsGeoJSON(VARCHAR(128)) –** Returns a string representing a geometry in JSON format.
- **ST_AsGML(VARCHAR(128)) –** Returns the GML representation of an ST_Geometry value.
- **ST_AsKML(VARCHAR(128)) –** Returns the KML representation of an ST_Geometry value.
- **ST_AsSVG(VARCHAR(128)) –** Returns an SVG figure representing a geometry value.
- **ST_AsSVGAggr(ST_Geometry , VARCHAR(128)) –** Returns a complete or partial SVG document which renders the geometries in a group.
- **ST_AsText(VARCHAR(128)) –** Returns the text representation of an ST_Geometry value.

- **ST_AsWKB(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsWKT(VARCHAR(128)) –** Returns the WKT representation of an ST_Geometry value.
- **ST_AsXML(VARCHAR(128)) –** Returns the XML representation of an ST_Geometry value.
- **ST_Boundary() –** Returns the boundary of the geometry value.
- **ST_Buffer(DOUBLE, VARCHAR(128)) –** 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.
- **ST_Contains(ST_Geometry) –** Tests if a geometry value spatially contains another geometry value.
- **ST_ContainsFilter(ST_Geometry) –** An inexpensive test if a geometry might contain another.
- **ST_ConvexHull() –** Returns the convex hull of the geometry value.
- **ST_ConvexHullAggr(ST_Geometry) –** Returns the convex hull for all of the geometries in a group
- **ST_CoordDim() –** Returns the number of coordinate dimensions stored with each point of the ST_Geometry value.
- **ST_CoveredBy(ST_Geometry) –** Tests if a geometry value is spatially covered by another geometry value.
- **ST_CoveredByFilter(ST_Geometry) –** An inexpensive test if a geometry might be covered by another.
- **ST_Covers(ST_Geometry) –** Tests if a geometry value spatially covers another geometry value.
- **ST_CoversFilter(ST_Geometry) –** An inexpensive test if a geometry might cover another.
- **ST_Crosses(ST_Geometry) –** Tests if a geometry value crosses another geometry value.
- **ST_Debug(VARCHAR(128)) –** Returns a LONG BINARY that is debug information for the object.
- **ST_Difference(ST_Geometry) –** Returns the geometry value that represents the point set difference of two geometries.
- **ST_Dimension() –** Returns the dimension of the ST_Geometry value. Points have dimension 0, lines have dimension 1, and surfaces have dimension 2. Any empty geometry has dimension -1.
- **ST_Disjoint(ST_Geometry) –** Test if a geometry value is spatially disjoint from another value.
- **ST_Distance(ST_Geometry , VARCHAR(128)) –** Returns the smallest distance between the geometry-expression and the specified geometry value.

- **ST_Distance_Spheroid(ST_Geometry , VARCHAR(128)) –** Calculates the linear distance between geometries on the surface of the Earth.
- **ST_Envelope() –** Returns the bounding rectangle for the geometry value.
- **ST_EnvelopeAggr(ST_Geometry) –** Returns the bounding rectangle for all of the geometries in a group
- **ST_Equals(ST_Geometry) –** Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.
- **ST_EqualsFilter(ST_Geometry) –** An inexpensive test if a geometry is equal to another.
- **ST_GeometryType() –** Returns the name of the type of the ST_Geometry value.
- **ST_GeometryTypeFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_GeomFromBinary(LONG BINARY, INT) –** Constructs a geometry from a binary string representation.
- **ST_GeomFromShape(LONG BINARY[, INT]) –** Parses a string containing an ESRI shape record and creates a geometry value of the appropriate type.
- **ST_GeomFromText(LONG VARCHAR, INT) –** Constructs a geometry from a character string representation.
- **ST_GeomFromWKB(LONG BINARY, INT) –** Parse a string containing a WKB or EWKB representation of a geometry and creates a geometry value of the appropriate type.
- **ST_GeomFromWKT(LONG VARCHAR, INT) –** Parses a string containing the WKT or EWKT representation of a geometry and create a geometry value of the appropriate type.
- **ST_Intersection(ST_Geometry) –** Returns the geometry value that represents the point set intersection of two geometries.
- **ST_IntersectionAggr(ST_Geometry) –** Returns the spatial intersection of all of the geometries in a group
- **ST_Intersects(ST_Geometry) –** Test if a geometry value spatially intersects another value.
- **ST_IntersectsFilter(ST_Geometry) –** An inexpensive test if the two geometries might intersect.
- **ST_IntersectsRect(ST_Point , ST_Point) –** Test if a geometry intersects a rectangle.
- **ST_Is3D() –** Determines if the geometry value has Z coordinate values.
- **ST_IsEmpty() –** Determines whether the geometry value represents an empty set.
- **ST_IsMeasured() –** Determines if the geometry value has associated measure values.
- **ST_IsSimple() –** Determines whether the geometry value is simple (containing no self intersections or other irregularities).
- **ST_IsValid() –** Determines whether the geometry is a valid spatial object.
- **ST_LatNorth() –** Retrieves the northernmost latitude of a geometry.
- **ST_LatSouth() –** Retrieves the southernmost latitude of a geometry.

- **ST_Length_Spheroid(VARCHAR(128)) –** Calculates the linear length of a curve/multicurve on the surface of the Earth.
- **ST_LinearHash() –** Returns a binary string that is a linear hash of the geometry.
- **ST_LinearUnHash(BINARY(32)[, INT]) –** Returns a geometry representing the index hash.
- **ST_LoadConfigurationData(VARCHAR(128)) –** Returns binary configuration data. For internal use only.
- **ST_LocateAlong(DOUBLE) –** Returns the subset of the geometry value that is associated with the given measure value.
- **ST_LocateBetween(DOUBLE, DOUBLE) –** Returns the subset of the geometry value that is between the specified start measure and end measure.
- **ST_LongEast() –** Retrieves the longitude of the eastern boundary of a geometry.
- **ST_LongWest() –** Retrieves the longitude of the western boundary of a geometry.
- **ST_MMax() –** Retrieves the maximum M coordinate value of a geometry.
- **ST_MMin() –** Retrieves the minimum M coordinate value of a geometry.
- **ST_OrderingEquals(ST_Geometry) –** Tests if a geometry is identical to another geometry.
- **ST_Overlaps(ST_Geometry) –** Tests if a geometry value overlaps another geometry value.
- **ST_Relate(ST_Geometry) –** Determines how a geometry value is spatially related to another geometry value by returning an intersection matrix. The ST_Relate method returns 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.
- **ST_Reverse() –** Returns the geometry with the element order reversed.
- **ST_Segmentize(DOUBLE) –** Add points so that no line segment is longer than a specified distance.
- **ST_Simplify(DOUBLE) –** Remove points from curves so long as the maximum introduced error is less than a specified tolerance.
- **ST_SnapToGrid(ST_Point , DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a copy of the geometry with all points snapped to the specified grid.
- **ST_SRID(INT) –** Changes the spatial reference system associated with the geometry without modifying any of the values.
- **ST_SRIDFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_SymDifference(ST_Geometry) –** Returns the geometry value that represents the point set symmetric difference of two geometries.
- **ST_ToCircular() –** Convert the geometry to a circularstring
- **ST_ToCompound() –** Converts the geometry to a compound curve.
- **ST_ToCurve() –** Converts the geometry to a curve.
- **ST_ToCurvePoly() –** Converts the geometry to a curve polygon.

- **ST_ToGeomColl() –** Converts the geometry to a geometry collection.
- **ST_ToLineString() –** Converts the geometry to a linestring.
- **ST_ToMultiCurve() –** Converts the geometry to a multicurve value.
- **ST_ToMultiLine() –** Converts the geometry to a multilinestring value.
- **ST_ToMultiPoint() –** Converts the geometry to a multi-point value.
- **ST_ToMultiPolygon() –** Converts the geometry to a multi-polygon value.
- **ST_ToMultiSurface() –** Converts the geometry to a multi-surface value.
- **ST_ToPoint() –** Converts the geometry to a point.
- **ST_ToPolygon() –** Converts the geometry to a polygon.
- **ST_ToSurface() –** Converts the geometry to a surface.
- **ST_Touches(ST_Geometry) –** Tests if a geometry value spatially touches another geometry value.
- **ST_Transform(INT) –** Creates a copy of the geometry value transformed into the specified spatial reference system.
- **ST_Union(ST_Geometry) –** Returns the geometry value that represents the point set union of two geometries.
- **ST_UnionAggr(ST_Geometry) –** Returns the spatial union of all of the geometries in a group
- **ST_Within(ST_Geometry) –** Tests if a geometry value is spatially contained within another geometry value.
- **ST_WithinDistance(ST_Geometry , DOUBLE, VARCHAR(128)) –** Test if two geometries are within a specified distance of each other.
- **ST_WithinDistanceFilter(ST_Geometry , DOUBLE, VARCHAR(128)) –** An inexpensive of whether two geometries might be within a specified distance of each other.
- **ST_WithinFilter(ST_Geometry) –** An inexpensive test if a geometry might be within another.
- **ST_XMax() –** Retrieves the maximum X coordinate value of a geometry.
- **ST_XMin() –** Retrieves the minimum X coordinate value of a geometry.
- **ST_YMax() –** Retrieves the maximum Y coordinate value of a geometry.
- **ST_YMin() –** Retrieves the minimum Y coordinate value of a geometry.
- **ST_ZMax() –** Retrieves the maximum Z coordinate value of a geometry.
- **ST_ZMin() –** Retrieves the minimum Z coordinate value of a geometry.

Standards and compatibility

SQL/MM (ISO/IEC 13249-3: 2006) 9.5

ST_MultiSurface(ST_MultiCurve , VARCHAR(128)) constructor

Creates a multi-surface from a multicurve containing exterior rings and an optional list of interior rings.

Syntax

NEW `ST_MultiSurface`(ST_MultiCurve `multi_curve`, VARCHAR(128) `polygon_format`)

Parameters

- **multi_curve –** A multicurve value containing exterior rings and (optionally) a set of interior rings.
- **polygon_format –** A string with the polygon format to use when interpreting the provided curves. Valid formats are 'CounterClockwise', 'Clockwise', and 'EvenOdd'

Returns

`ST_MultiSurface` Returns a multi-surface from the specified multilinestring.

Examples

- **Example 1 –** The following returns MultiSurface (CurvePolygon ((-4 -4, 4 -4, 4 4, -4 4, -4 -4), (-2 1, -3 3, -1 3, -2 1)), CurvePolygon ((6 -4, 14 -4, 14 4, 6 4, 6 -4), CircularString (9 -1, 9 1, 11 1, 11 -1, 9 -1))).

```
SELECT NEW ST_MultiSurface(NEW ST_MultiCurve ('MultiCurve ((-4 -4,
4 -4, 4 4, -4 4, -4 -4), (-2 1, -3 3, -1 3, -2 1), (6 -4, 14 -4, 14
4, 6 4, 6 -4), CircularString (9 -1, 9 1, 11 1, 11 -1, 9 -1))'))
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_MultiSurface(ST_Surface , ST_Surface) constructor

Constructs a multi-surface from a list of surface values.

Syntax

NEW `ST_MultiSurface`(ST_Surface `surface1`, ST_Surface `surfacei`)

Parameters

- **surface1 –** The first surface value of the multi-surface.
- **surfacei –** Additional surface values of the multi-surface.

Returns

`ST_MultiSurface` A multi-surface containing the provided surface values.

Examples

- **Example 1 –** The following example returns the result `MultiSurface (((0 0, 1 0, 1 1, 0 1, 0 0)))`.

```
SELECT NEW ST_MultiSurface( NEW ST_Polygon('Polygon ((0 0, 0 1, 1 1, 1 0, 0 0))' )  )
```

  The following example returns the result `MultiSurface (((0 0, 1 0, 1 1, 0 1, 0 0)), ((5 5, 10 5, 10 10, 5 10, 5 5)))`.

```
SELECT NEW ST_MultiSurface(
NEW ST_Polygon('Polygon ((0 0, 0 1, 1 1, 1 0, 0 0))' ),
NEW ST_Polygon('Polygon ((5 5, 5 10, 10 10, 10 5, 5 5))' )  )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_MultiSurface() constructor

Constructs a multi surface representing the empty set.

Syntax

NEW `ST_MultiSurface`()

Returns

`ST_MultiSurface` Returns an ST_MultiSurface value representing the empty set.

Examples

- **Example 1 –** The following returns 1, indicating the value is empty.

```
SELECT NEW ST_MultiSurface().ST_IsEmpty()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Standard feature

ST_MultiSurface(LONG BINARY[, INT]) constructor

Constructs a multi surface from Well Known Binary (WKB).

Syntax

NEW `ST_MultiSurface`(LONG BINARY `wkb`[, INT `srid`])

Parameters

- **wkb –** A string containing the binary representation of a multi surface. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_MultiSurface` Returns an ST_MultiSurface value constructed from the source string.

Examples

- **Example 1 –** The following returns MultiSurface (CurvePolygon (CircularString (0 0, 10 0, 10 10, 0 10, 0 0))).

```
SELECT NEW
ST_MultiSurface(0x010c00000001000000010a0000000100000001080000000
500000000000000000000000000000000000000000000000000024400000000000
000000000000000000244000000000000024400000000000000000000000000000
0244000000000000000000000000000000000)
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 9.5.2

ST_MultiSurface(LONG VARCHAR[, INT]) constructor

Constructs a multi surface from a text representation.

Syntax

NEW `ST_MultiSurface`(LONG VARCHAR `text_representation`[, INT `srid`])

Parameters

- **text_representation –** A string containing the text representation of a multi surface. The input can be in any supported text input format, including Well Known Text (WKT) or Extended Well Known Text (EWKT).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_MultiSurface` Returns an ST_MultiSurface value constructed from the source string.

Examples

- **Example 1 –** The following returns MultiSurface (((-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_MultiSurface('MultiSurface (((-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)))')
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 9.5.2

ST_Area(VARCHAR(128)) method

Computes the area of the multi-surface in the specified units.

Syntax

`multisurface-expression.ST_Area`(VARCHAR(128) `unit_name`)

Parameters

- **unit_name –** The units in which the area 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 area of the multi-surface.

Examples

- **Example 1 –** The following example returns the result `8`.

  ```
  SELECT TREAT( Shape AS ST_MultiSurface ).ST_Area()
  FROM SpatialShapes WHERE ShapeID = 27
  ```

  The following returns the area of the multipoly_geometry column in square miles from the fictional region table.

  ```
  SELECT name, multipoly_geometry.ST_Area( 'Statute Mile' )
  FROM region
  ```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 9.5.3

ST_Centroid() method

Computes the ST_Point that is the mathematical centroid of the multi-surface.

Syntax

`multisurface-expression.ST_Centroid`()

Returns

`ST_Point` If the multi-surface is the empty set, returns NULL. Otherwise, returns the mathematical centroid of the surface.

Examples

- **Example 1 –** The following example returns the result `Point (1.865682 . 664892)`.

```
SELECT TREAT( Shape AS ST_MultiSurface ).ST_Centroid()
FROM SpatialShapes WHERE ShapeID = 28
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 9.5.5

ST_MultiSurfaceAggr(ST_Surface) method

Returns a multisurface containing all of the surfaces in a group.

Syntax

`ST_MultiSurface::ST_MultiSurfaceAggr`(ST_Surface `geometry_column`)

Parameters

- **geometry_column –** The geometry values to generate the collection. Typically this is a column.

Returns

`ST_MultiSurface` Returns a multisurface that contains all of the geometries in a group.

Examples

- **Example 1 –** The following example returns a single value which combines all geometries of type ST_Surface from the SpatialShapes table into a single collection of type ST_MultiSurface. If the Shape column was of type ST_Surface then the TREAT function and WHERE clause would not be necessary.

```
SELECT ST_MultiSurface::ST_MultiSurfaceAggr( TREAT( Shape AS
ST_Surface ) )
FROM SpatialShapes WHERE Shape IS OF( ST_Surface )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_Perimeter(VARCHAR(128)) method

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

Syntax

```
multisurface-expression.ST_Perimeter(VARCHAR(128) unit_name)
```

Parameters

- **unit_name –** The units 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.

Examples

- **Example 1 –** 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_MultiSurface( NEW ST_Polygon('Polygon((0 0, 1 0, 1
1,0 1, 0 0))')
, NEW ST_Polygon('Polygon((10 10, 20 10, 20 20,10 20, 10 10))') )
.ST_Perimeter()
```

The following example creates a multi-surface containing two polygons and an example unit of measure (example_unit_halfmetre). The ST_Perimeter method finds the length of the perimeter, returning the value 88.0.

```
CREATE SPATIAL UNIT OF MEASURE IF NOT EXISTS
"example_unit_halfmetre" TYPE LINEAR CONVERT USING .5;
SELECT NEW ST_MultiSurface( NEW ST_Polygon('Polygon((0 0, 1 0, 1
1,0 1, 0 0))')
, NEW ST_Polygon('Polygon((10 10, 20 10, 20 20,10 20, 10 10))') )
.ST_Perimeter('example_unit_halfmetre');
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 9.5.4

ST_PointOnSurface() method

Returns a point that is guaranteed to be on a surface in the multi-surface

Syntax

```
multisurface-expression.ST_PointOnSurface()
```

Returns

`ST_Point` If the multi-surface is the empty set, returns NULL. Otherwise, returns an ST_Point value guaranteed to spatially intersect the ST_MultiSurface value.

Examples

- **Example 1 –** The following returns a point that intersects the multi surface.

```
SELECT TREAT( Shape AS ST_MultiSurface ).ST_PointOnSurface()
FROM SpatialShapes WHERE ShapeID = 27
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 9.5.6

ST_Point type

The ST_Point type is a 0-dimensional geometry and represents a single location.

Syntax

`ST_Point` type

Members

All members of the ST_Point type, including all inherited members.

Members of ST_Point:

- **ST_Point() –** Constructs a point representing the empty set.
- **ST_Point(DOUBLE, DOUBLE, DOUBLE, DOUBLE[, INT]) –** Constructs a 3D, measured point from X,Y,Z coordinates and a measure value
- **ST_Point(DOUBLE, DOUBLE, DOUBLE[, INT]) –** Constructs a 3D point from X,Y,Z coordinates.
- **ST_Point(DOUBLE, DOUBLE[, INT]) –** Constructs a 2D point from X,Y coordinates.
- **ST_Point(LONG BINARY[, INT]) –** Constructs a point from Well Known Binary (WKB).
- **ST_Point(LONG VARCHAR[, INT]) –** Constructs a point from a text representation.
- **ST_Lat(DOUBLE) –** Returns a copy of the point with the latitude coordinate set to the specified latitude value.
- **ST_Long(DOUBLE) –** Returns a copy of the point with the longitude coordinate set to the specified longitude value.
- **ST_M(DOUBLE) –** Returns a copy of the point with the measure value set to the specified mcoord value.
- **ST_X(DOUBLE) –** Returns a copy of the point with the X coordinate set to the specified xcoord value.

- **ST_Y(DOUBLE) –** Returns a copy of the point with the Y coordinate set to the specified ycoord value.
- **ST_Z(DOUBLE) –** Returns a copy of the point with the Z coordinate set to the specified zcoord value.

Members of ST_Geometry:

- **ST_Affine(DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a new geometry that is the result of applying the specified 3-D affine transformation.
- **ST_AsBinary(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsBitmap(INT, INT, ST_Point , ST_Point , VARCHAR(128)) –** Returns a LONG VARBIT that is a bitmap representing a geometry value.
- **ST_AsGeoJSON(VARCHAR(128)) –** Returns a string representing a geometry in JSON format.
- **ST_AsGML(VARCHAR(128)) –** Returns the GML representation of an ST_Geometry value.
- **ST_AsKML(VARCHAR(128)) –** Returns the KML representation of an ST_Geometry value.
- **ST_AsSVG(VARCHAR(128)) –** Returns an SVG figure representing a geometry value.
- **ST_AsSVGAggr(ST_Geometry , VARCHAR(128)) –** Returns a complete or partial SVG document which renders the geometries in a group.
- **ST_AsText(VARCHAR(128)) –** Returns the text representation of an ST_Geometry value.
- **ST_AsWKB(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsWKT(VARCHAR(128)) –** Returns the WKT representation of an ST_Geometry value.
- **ST_AsXML(VARCHAR(128)) –** Returns the XML representation of an ST_Geometry value.
- **ST_Boundary() –** Returns the boundary of the geometry value.
- **ST_Buffer(DOUBLE, VARCHAR(128)) –** 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.
- **ST_Contains(ST_Geometry) –** Tests if a geometry value spatially contains another geometry value.
- **ST_ContainsFilter(ST_Geometry) –** An inexpensive test if a geometry might contain another.
- **ST_ConvexHull() –** Returns the convex hull of the geometry value.
- **ST_ConvexHullAggr(ST_Geometry) –** Returns the convex hull for all of the geometries in a group

- **ST_CoordDim() –** Returns the number of coordinate dimensions stored with each point of the ST_Geometry value.
- **ST_CoveredBy(ST_Geometry) –** Tests if a geometry value is spatially covered by another geometry value.
- **ST_CoveredByFilter(ST_Geometry) –** An inexpensive test if a geometry might be covered by another.
- **ST_Covers(ST_Geometry) –** Tests if a geometry value spatially covers another geometry value.
- **ST_CoversFilter(ST_Geometry) –** An inexpensive test if a geometry might cover another.
- **ST_Crosses(ST_Geometry) –** Tests if a geometry value crosses another geometry value.
- **ST_Debug(VARCHAR(128)) –** Returns a LONG BINARY that is debug information for the object.
- **ST_Difference(ST_Geometry) –** Returns the geometry value that represents the point set difference of two geometries.
- **ST_Dimension() –** Returns the dimension of the ST_Geometry value. Points have dimension 0, lines have dimension 1, and surfaces have dimension 2. Any empty geometry has dimension -1.
- **ST_Disjoint(ST_Geometry) –** Test if a geometry value is spatially disjoint from another value.
- **ST_Distance(ST_Geometry , VARCHAR(128)) –** Returns the smallest distance between the geometry-expression and the specified geometry value.
- **ST_Distance_Spheroid(ST_Geometry , VARCHAR(128)) –** Calculates the linear distance between geometries on the surface of the Earth.
- **ST_Envelope() –** Returns the bounding rectangle for the geometry value.
- **ST_EnvelopeAggr(ST_Geometry) –** Returns the bounding rectangle for all of the geometries in a group
- **ST_Equals(ST_Geometry) –** Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.
- **ST_EqualsFilter(ST_Geometry) –** An inexpensive test if a geometry is equal to another.
- **ST_GeometryType() –** Returns the name of the type of the ST_Geometry value.
- **ST_GeometryTypeFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_GeomFromBinary(LONG BINARY, INT) –** Constructs a geometry from a binary string representation.
- **ST_GeomFromShape(LONG BINARY[, INT]) –** Parses a string containing an ESRI shape record and creates a geometry value of the appropriate type.
- **ST_GeomFromText(LONG VARCHAR, INT) –** Constructs a geometry from a character string representation.

- **ST_GeomFromWKB(LONG BINARY, INT) –** Parse a string containing a WKB or EWKB representation of a geometry and creates a geometry value of the appropriate type.
- **ST_GeomFromWKT(LONG VARCHAR, INT) –** Parses a string containing the WKT or EWKT representation of a geometry and create a geometry value of the appropriate type.
- **ST_Intersection(ST_Geometry) –** Returns the geometry value that represents the point set intersection of two geometries.
- **ST_IntersectionAggr(ST_Geometry) –** Returns the spatial intersection of all of the geometries in a group
- **ST_Intersects(ST_Geometry) –** Test if a geometry value spatially intersects another value.
- **ST_IntersectsFilter(ST_Geometry) –** An inexpensive test if the two geometries might intersect.
- **ST_IntersectsRect(ST_Point , ST_Point) –** Test if a geometry intersects a rectangle.
- **ST_Is3D() –** Determines if the geometry value has Z coordinate values.
- **ST_IsEmpty() –** Determines whether the geometry value represents an empty set.
- **ST_IsMeasured() –** Determines if the geometry value has associated measure values.
- **ST_IsSimple() –** Determines whether the geometry value is simple (containing no self intersections or other irregularities).
- **ST_IsValid() –** Determines whether the geometry is a valid spatial object.
- **ST_LatNorth() –** Retrieves the northernmost latitude of a geometry.
- **ST_LatSouth() –** Retrieves the southernmost latitude of a geometry.
- **ST_Length_Spheroid(VARCHAR(128)) –** Calculates the linear length of a curve/multicurve on the surface of the Earth.
- **ST_LinearHash() –** Returns a binary string that is a linear hash of the geometry.
- **ST_LinearUnHash(BINARY(32)[, INT]) –** Returns a geometry representing the index hash.
- **ST_LoadConfigurationData(VARCHAR(128)) –** Returns binary configuration data. For internal use only.
- **ST_LocateAlong(DOUBLE) –** Returns the subset of the geometry value that is associated with the given measure value.
- **ST_LocateBetween(DOUBLE, DOUBLE) –** Returns the subset of the geometry value that is between the specified start measure and end measure.
- **ST_LongEast() –** Retrieves the longitude of the eastern boundary of a geometry.
- **ST_LongWest() –** Retrieves the longitude of the western boundary of a geometry.
- **ST_MMax() –** Retrieves the maximum M coordinate value of a geometry.
- **ST_MMin() –** Retrieves the minimum M coordinate value of a geometry.
- **ST_OrderingEquals(ST_Geometry) –** Tests if a geometry is identical to another geometry.
- **ST_Overlaps(ST_Geometry) –** Tests if a geometry value overlaps another geometry value.

- **ST_Relate(ST_Geometry) –** Determines how a geometry value is spatially related to another geometry value by returning an intersection matrix. The ST_Relate method returns 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.
- **ST_Reverse() –** Returns the geometry with the element order reversed.
- **ST_Segmentize(DOUBLE) –** Add points so that no line segment is longer than a specified distance.
- **ST_Simplify(DOUBLE) –** Remove points from curves so long as the maximum introduced error is less than a specified tolerance.
- **ST_SnapToGrid(ST_Point , DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a copy of the geometry with all points snapped to the specified grid.
- **ST_SRID(INT) –** Changes the spatial reference system associated with the geometry without modifying any of the values.
- **ST_SRIDFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_SymDifference(ST_Geometry) –** Returns the geometry value that represents the point set symmetric difference of two geometries.
- **ST_ToCircular() –** Convert the geometry to a circularstring
- **ST_ToCompound() –** Converts the geometry to a compound curve.
- **ST_ToCurve() –** Converts the geometry to a curve.
- **ST_ToCurvePoly() –** Converts the geometry to a curve polygon.
- **ST_ToGeomColl() –** Converts the geometry to a geometry collection.
- **ST_ToLineString() –** Converts the geometry to a linestring.
- **ST_ToMultiCurve() –** Converts the geometry to a multicurve value.
- **ST_ToMultiLine() –** Converts the geometry to a multilinestring value.
- **ST_ToMultiPoint() –** Converts the geometry to a multi-point value.
- **ST_ToMultiPolygon() –** Converts the geometry to a multi-polygon value.
- **ST_ToMultiSurface() –** Converts the geometry to a multi-surface value.
- **ST_ToPoint() –** Converts the geometry to a point.
- **ST_ToPolygon() –** Converts the geometry to a polygon.
- **ST_ToSurface() –** Converts the geometry to a surface.
- **ST_Touches(ST_Geometry) –** Tests if a geometry value spatially touches another geometry value.
- **ST_Transform(INT) –** Creates a copy of the geometry value transformed into the specified spatial reference system.
- **ST_Union(ST_Geometry) –** Returns the geometry value that represents the point set union of two geometries.
- **ST_UnionAggr(ST_Geometry) –** Returns the spatial union of all of the geometries in a group

- **ST_Within(ST_Geometry) –** Tests if a geometry value is spatially contained within another geometry value.
- **ST_WithinDistance(ST_Geometry , DOUBLE, VARCHAR(128)) –** Test if two geometries are within a specified distance of each other.
- **ST_WithinDistanceFilter(ST_Geometry , DOUBLE, VARCHAR(128)) –** An inexpensive of whether two geometries might be within a specified distance of each other.
- **ST_WithinFilter(ST_Geometry) –** An inexpensive test if a geometry might be within another.
- **ST_XMax() –** Retrieves the maximum X coordinate value of a geometry.
- **ST_XMin() –** Retrieves the minimum X coordinate value of a geometry.
- **ST_YMax() –** Retrieves the maximum Y coordinate value of a geometry.
- **ST_YMin() –** Retrieves the minimum Y coordinate value of a geometry.
- **ST_ZMax() –** Retrieves the maximum Z coordinate value of a geometry.
- **ST_ZMin() –** Retrieves the minimum Z coordinate value of a geometry.

Standards and compatibility
SQL/MM (ISO/IEC 13249-3: 2006) 6.1

ST_Point() constructor

Constructs a point representing the empty set.

Syntax

NEW `ST_Point`()

Returns

`ST_Point` Returns an ST_Point value representing the empty set.

Examples

- **Example 1 –** The following returns 1, indicating the value is empty.

```
SELECT NEW ST_Point().ST_IsEmpty()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Standard feature

ST_Point(DOUBLE, DOUBLE, DOUBLE, DOUBLE[, INT]) constructor

Constructs a 3D, measured point from X,Y,Z coordinates and a measure value

Syntax

NEW `ST_Point`(DOUBLE `x`, DOUBLE `y`, DOUBLE `z`, DOUBLE `m`[, INT `srid`])

Parameters

- **x –** The X coordinate value.
- **y –** The Y coordinate value.
- **z –** The Z coordinate value.
- **m –** The measure value.
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_Point` Returns a 3D, measured point with the specified X,Y,Z coordinates and a measure value

Examples

- **Example 1 –** The following returns Point ZM (10 20 100 1224).

```
SELECT NEW ST_Point(10.0,20.0,100.0,1224.0,0)
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 6.1.2

ST_Point(DOUBLE, DOUBLE, DOUBLE[, INT]) constructor

Constructs a 3D point from X,Y,Z coordinates.

Syntax

NEW `ST_Point`(DOUBLE `x`, DOUBLE `y`, DOUBLE `z`[, INT `srid`])

Parameters

- **x –** The X coordinate value.
- **y –** The Y coordinate value.
- **z –** The Z coordinate value.
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_Point` Returns a 3D point with the specified X,Y,Z coordinates.

Examples

- **Example 1 –** The following returns Point Z (10 20 100).

```
SELECT NEW ST_Point(10.0,20.0,100.0,0)
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 6.1.2

ST_Point(DOUBLE, DOUBLE[, INT]) constructor

Constructs a 2D point from X,Y coordinates.

Syntax

NEW `ST_Point`(DOUBLE `x`, DOUBLE `y`[, INT `srid`])

Parameters

- **x –** The X coordinate value.
- **y –** The Y coordinate value.
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_Point` Returns a 2D point with the specified X,Y coordinates.

Examples

- **Example 1 –** The following returns Point (10 20).

```
SELECT NEW ST_Point(10.0,20.0,0)
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 6.1.2

ST_Point(LONG BINARY[, INT]) constructor

Constructs a point from Well Known Binary (WKB).

Syntax

NEW `ST_Point`(LONG BINARY `wkb`[, INT `srid`])

Parameters

- **wkb –** A string containing the binary representation of a point. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_Point` Returns an ST_Point value constructed from the source string.

Examples

- **Example 1 –** The following returns Point (10 20).
```
SELECT NEW ST_Point(0x01010000000000000000002440000000000000003440)
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 6.1.2

ST_Point(LONG VARCHAR[, INT]) constructor

Constructs a point from a text representation.

Syntax

NEW `ST_Point`(LONG VARCHAR `text_representation`[, INT `srid`])

Parameters

- **text_representation –** A string containing the text representation of a point. The input can be in any supported text input format, including Well Known Text (WKT) or Extended Well Known Text (EWKT).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_Point` Returns an ST_Point value constructed from the source string.

Examples

- **Example 1 –** The following returns Point (10 20).
```
SELECT NEW ST_Point('Point (10 20)')
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 6.1.2

ST_Lat(DOUBLE) method

Returns a copy of the point with the latitude coordinate set to the specified latitude value.

Syntax

`point-expression.ST_Lat`(DOUBLE `latitude_val`)

Parameters

- **latitude_val –** The new latitude value.

Returns

`ST_Point` Returns a copy of the point with the latitude set to the specified value.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_Long(DOUBLE) method

Returns a copy of the point with the longitude coordinate set to the specified longitude value.

Syntax

`point-expression.ST_Long`(DOUBLE `longitude_val`)

Parameters

- **longitude_val –** The new longitude value.

Returns

`ST_Point` Returns a copy of the point with the longitude set to the specified value.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_M(DOUBLE) method

Returns a copy of the point with the measure value set to the specified mcoord value.

Syntax

`point-expression.ST_M`(DOUBLE `mcoord`)

Parameters

- **mcoord –** The new measure value.

Returns

`ST_Point` Returns a copy of the point with the measure value set to the specified mcoord value.

Examples

- **Example 1 –** The following example returns the result `Point ZM (1 2 3 5)`.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0, 4.0 ).ST_M( 5.0 )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 6.1.6

ST_X(DOUBLE) method

Returns a copy of the point with the X coordinate set to the specified xcoord value.

Syntax

```
point-expression.ST_X(DOUBLE xcoord)
```

Parameters

- **xcoord –** The new X coordinate value.

Returns

`ST_Point` Returns a copy of the point with the X coordinate set to the specified xcoord value.

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 6.1.3

ST_Y(DOUBLE) method

Returns a copy of the point with the Y coordinate set to the specified ycoord value.

Syntax

```
point-expression.ST_Y(DOUBLE ycoord)
```

Parameters

- **ycoord –** The new Y coordinate value.

Returns

`ST_Point` Returns a copy of the point with the Y coordinate set to the specified ycoord value.

Examples

- **Example 1 –** The following example returns the result `Point (1 3)`.

```
SELECT NEW ST_Point( 1, 2 ).ST_Y( 3 )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 6.1.4

ST_Z(DOUBLE) method

Returns a copy of the point with the Z coordinate set to the specified zcoord value.

Syntax

`point-expression.ST_Z(DOUBLE zcoord)`

Parameters

- **zcoord –** The new Z coordinate value.

Returns

`ST_Point` Returns a copy of the point with the Z coordinate set to the specified zcoord value.

Examples

- **Example 1 –** The following example returns the result `Point Z (1 2 5)`.

```
SELECT NEW ST_Point( 1.0, 2.0, 3.0 ).ST_Z( 5.0 )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 6.1.5

ST_Polygon type

An ST_Polygon is an ST_CurvePolygon that is formed with interior and exterior rings that are linear rings.

Syntax

`ST_Polygon` type

Members

All members of the ST_Polygon type, including all inherited members.

Members of ST_Polygon:

- **ST_Polygon(ST_LineString , ST_LineString) –** Creates a polygon from a linestring representing the exterior ring and an optional list of linestrings representing interior rings.
- **ST_Polygon(ST_MultiLineString , VARCHAR(128)) –** Creates a polygon from a multilinestring containing an exterior ring and an optional list of interior rings.
- **ST_Polygon(ST_Point , ST_Point) –** Creates an axis-aligned rectangle from two points representing the lower-left and upper-right corners.

- **ST_Polygon() –** Constructs a polygon representing the empty set.
- **ST_Polygon(LONG BINARY[, INT]) –** Constructs a polygon from Well Known Binary (WKB).
- **ST_Polygon(LONG VARCHAR[, INT]) –** Constructs a polygon from a text representation.
- **ST_ExteriorRing(ST_Curve) –** Changes the exterior ring of the polygon.
- **ST_InteriorRingN(INT) –** Returns the nth interior ring in the polygon.

Members of ST_CurvePolygon:

- **ST_CurvePolygon(ST_Curve , ST_Curve) –** Creates a curve polygon from a curve representing the exterior ring and a list of curves representing interior rings, all in a specified spatial reference system.
- **ST_CurvePolygon(ST_MultiCurve , VARCHAR(128)) –** Creates a curve polygon from a multi curve containing an exterior ring and an optional list of interior rings.
- **ST_CurvePolygon() –** Constructs a curve polygon representing the empty set.
- **ST_CurvePolygon(LONG BINARY[, INT]) –** Constructs a curve polygon from Well Known Binary (WKB).
- **ST_CurvePolygon(LONG VARCHAR[, INT]) –** Constructs a curve polygon from a text representation.
- **ST_CurvePolyToPoly() –** Returns the interpolation of the curve polygon as a polygon.
- **ST_ExteriorRing(ST_Curve) –** Changes the exterior ring of the curve polygon.
- **ST_InteriorRingN(INT) –** Returns the nth interior ring in the curve polygon.
- **ST_NumInteriorRing() –** Returns the number of interior rings in the curve polygon.

Members of ST_Geometry:

- **ST_Affine(DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a new geometry that is the result of applying the specified 3-D affine transformation.
- **ST_AsBinary(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsBitmap(INT, INT, ST_Point , ST_Point , VARCHAR(128)) –** Returns a LONG VARBIT that is a bitmap representing a geometry value.
- **ST_AsGeoJSON(VARCHAR(128)) –** Returns a string representing a geometry in JSON format.
- **ST_AsGML(VARCHAR(128)) –** Returns the GML representation of an ST_Geometry value.
- **ST_AsKML(VARCHAR(128)) –** Returns the KML representation of an ST_Geometry value.
- **ST_AsSVG(VARCHAR(128)) –** Returns an SVG figure representing a geometry value.
- **ST_AsSVGAggr(ST_Geometry , VARCHAR(128)) –** Returns a complete or partial SVG document which renders the geometries in a group.

- **ST_AsText(VARCHAR(128)) –** Returns the text representation of an ST_Geometry value.
- **ST_AsWKB(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsWKT(VARCHAR(128)) –** Returns the WKT representation of an ST_Geometry value.
- **ST_AsXML(VARCHAR(128)) –** Returns the XML representation of an ST_Geometry value.
- **ST_Boundary() –** Returns the boundary of the geometry value.
- **ST_Buffer(DOUBLE, VARCHAR(128)) –** 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.
- **ST_Contains(ST_Geometry) –** Tests if a geometry value spatially contains another geometry value.
- **ST_ContainsFilter(ST_Geometry) –** An inexpensive test if a geometry might contain another.
- **ST_ConvexHull() –** Returns the convex hull of the geometry value.
- **ST_ConvexHullAggr(ST_Geometry) –** Returns the convex hull for all of the geometries in a group
- **ST_CoordDim() –** Returns the number of coordinate dimensions stored with each point of the ST_Geometry value.
- **ST_CoveredBy(ST_Geometry) –** Tests if a geometry value is spatially covered by another geometry value.
- **ST_CoveredByFilter(ST_Geometry) –** An inexpensive test if a geometry might be covered by another.
- **ST_Covers(ST_Geometry) –** Tests if a geometry value spatially covers another geometry value.
- **ST_CoversFilter(ST_Geometry) –** An inexpensive test if a geometry might cover another.
- **ST_Crosses(ST_Geometry) –** Tests if a geometry value crosses another geometry value.
- **ST_Debug(VARCHAR(128)) –** Returns a LONG BINARY that is debug information for the object.
- **ST_Difference(ST_Geometry) –** Returns the geometry value that represents the point set difference of two geometries.
- **ST_Dimension() –** Returns the dimension of the ST_Geometry value. Points have dimension 0, lines have dimension 1, and surfaces have dimension 2. Any empty geometry has dimension -1.
- **ST_Disjoint(ST_Geometry) –** Test if a geometry value is spatially disjoint from another value.

- **ST_Distance(ST_Geometry , VARCHAR(128)) –** Returns the smallest distance between the geometry-expression and the specified geometry value.
- **ST_Distance_Spheroid(ST_Geometry , VARCHAR(128)) –** Calculates the linear distance between geometries on the surface of the Earth.
- **ST_Envelope() –** Returns the bounding rectangle for the geometry value.
- **ST_EnvelopeAggr(ST_Geometry) –** Returns the bounding rectangle for all of the geometries in a group
- **ST_Equals(ST_Geometry) –** Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.
- **ST_EqualsFilter(ST_Geometry) –** An inexpensive test if a geometry is equal to another.
- **ST_GeometryType() –** Returns the name of the type of the ST_Geometry value.
- **ST_GeometryTypeFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_GeomFromBinary(LONG BINARY, INT) –** Constructs a geometry from a binary string representation.
- **ST_GeomFromShape(LONG BINARY[, INT]) –** Parses a string containing an ESRI shape record and creates a geometry value of the appropriate type.
- **ST_GeomFromText(LONG VARCHAR, INT) –** Constructs a geometry from a character string representation.
- **ST_GeomFromWKB(LONG BINARY, INT) –** Parse a string containing a WKB or EWKB representation of a geometry and creates a geometry value of the appropriate type.
- **ST_GeomFromWKT(LONG VARCHAR, INT) –** Parses a string containing the WKT or EWKT representation of a geometry and create a geometry value of the appropriate type.
- **ST_Intersection(ST_Geometry) –** Returns the geometry value that represents the point set intersection of two geometries.
- **ST_IntersectionAggr(ST_Geometry) –** Returns the spatial intersection of all of the geometries in a group
- **ST_Intersects(ST_Geometry) –** Test if a geometry value spatially intersects another value.
- **ST_IntersectsFilter(ST_Geometry) –** An inexpensive test if the two geometries might intersect.
- **ST_IntersectsRect(ST_Point , ST_Point) –** Test if a geometry intersects a rectangle.
- **ST_Is3D() –** Determines if the geometry value has Z coordinate values.
- **ST_IsEmpty() –** Determines whether the geometry value represents an empty set.
- **ST_IsMeasured() –** Determines if the geometry value has associated measure values.
- **ST_IsSimple() –** Determines whether the geometry value is simple (containing no self intersections or other irregularities).
- **ST_IsValid() –** Determines whether the geometry is a valid spatial object.
- **ST_LatNorth() –** Retrieves the northernmost latitude of a geometry.

- **ST_LatSouth() –** Retrieves the southernmost latitude of a geometry.
- **ST_Length_Spheroid(VARCHAR(128)) –** Calculates the linear length of a curve/multicurve on the surface of the Earth.
- **ST_LinearHash() –** Returns a binary string that is a linear hash of the geometry.
- **ST_LinearUnHash(BINARY(32)[, INT]) –** Returns a geometry representing the index hash.
- **ST_LoadConfigurationData(VARCHAR(128)) –** Returns binary configuration data. For internal use only.
- **ST_LocateAlong(DOUBLE) –** Returns the subset of the geometry value that is associated with the given measure value.
- **ST_LocateBetween(DOUBLE, DOUBLE) –** Returns the subset of the geometry value that is between the specified start measure and end measure.
- **ST_LongEast() –** Retrieves the longitude of the eastern boundary of a geometry.
- **ST_LongWest() –** Retrieves the longitude of the western boundary of a geometry.
- **ST_MMax() –** Retrieves the maximum M coordinate value of a geometry.
- **ST_MMin() –** Retrieves the minimum M coordinate value of a geometry.
- **ST_OrderingEquals(ST_Geometry) –** Tests if a geometry is identical to another geometry.
- **ST_Overlaps(ST_Geometry) –** Tests if a geometry value overlaps another geometry value.
- **ST_Relate(ST_Geometry) –** Determines how a geometry value is spatially related to another geometry value by returning an intersection matrix. The ST_Relate method returns 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.
- **ST_Reverse() –** Returns the geometry with the element order reversed.
- **ST_Segmentize(DOUBLE) –** Add points so that no line segment is longer than a specified distance.
- **ST_Simplify(DOUBLE) –** Remove points from curves so long as the maximum introduced error is less than a specified tolerance.
- **ST_SnapToGrid(ST_Point , DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a copy of the geometry with all points snapped to the specified grid.
- **ST_SRID(INT) –** Changes the spatial reference system associated with the geometry without modifying any of the values.
- **ST_SRIDFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_SymDifference(ST_Geometry) –** Returns the geometry value that represents the point set symmetric difference of two geometries.
- **ST_ToCircular() –** Convert the geometry to a circularstring
- **ST_ToCompound() –** Converts the geometry to a compound curve.
- **ST_ToCurve() –** Converts the geometry to a curve.

- **ST_ToCurvePoly() –** Converts the geometry to a curve polygon.
- **ST_ToGeomColl() –** Converts the geometry to a geometry collection.
- **ST_ToLineString() –** Converts the geometry to a linestring.
- **ST_ToMultiCurve() –** Converts the geometry to a multicurve value.
- **ST_ToMultiLine() –** Converts the geometry to a multilinestring value.
- **ST_ToMultiPoint() –** Converts the geometry to a multi-point value.
- **ST_ToMultiPolygon() –** Converts the geometry to a multi-polygon value.
- **ST_ToMultiSurface() –** Converts the geometry to a multi-surface value.
- **ST_ToPoint() –** Converts the geometry to a point.
- **ST_ToPolygon() –** Converts the geometry to a polygon.
- **ST_ToSurface() –** Converts the geometry to a surface.
- **ST_Touches(ST_Geometry) –** Tests if a geometry value spatially touches another geometry value.
- **ST_Transform(INT) –** Creates a copy of the geometry value transformed into the specified spatial reference system.
- **ST_Union(ST_Geometry) –** Returns the geometry value that represents the point set union of two geometries.
- **ST_UnionAggr(ST_Geometry) –** Returns the spatial union of all of the geometries in a group
- **ST_Within(ST_Geometry) –** Tests if a geometry value is spatially contained within another geometry value.
- **ST_WithinDistance(ST_Geometry , DOUBLE, VARCHAR(128)) –** Test if two geometries are within a specified distance of each other.
- **ST_WithinDistanceFilter(ST_Geometry , DOUBLE, VARCHAR(128)) –** An inexpensive of whether two geometries might be within a specified distance of each other.
- **ST_WithinFilter(ST_Geometry) –** An inexpensive test if a geometry might be within another.
- **ST_XMax() –** Retrieves the maximum X coordinate value of a geometry.
- **ST_XMin() –** Retrieves the minimum X coordinate value of a geometry.
- **ST_YMax() –** Retrieves the maximum Y coordinate value of a geometry.
- **ST_YMin() –** Retrieves the minimum Y coordinate value of a geometry.
- **ST_ZMax() –** Retrieves the maximum Z coordinate value of a geometry.
- **ST_ZMin() –** Retrieves the minimum Z coordinate value of a geometry.

Members of ST_Surface:

- **ST_Area(VARCHAR(128)) –** Calculates the area of a surface in the specified units.
- **ST_Centroid() –** Returns the ST_Point value that is the mathematical centroid of the surface value.
- **ST_IsWorld() –** Test if the ST_Surface covers the entire space.

- **ST_Perimeter(VARCHAR(128)) –** Calculates the perimeter of a surface in the specified units.
- **ST_PointOnSurface() –** Returns an ST_Point value that is guaranteed to spatially intersect the ST_Surface value.

Standards and compatibility
SQL/MM (ISO/IEC 13249-3: 2006) 8.3

ST_Polygon(ST_LineString , ST_LineString) constructor

Creates a polygon from a linestring representing the exterior ring and an optional list of linestrings representing interior rings.

Syntax

NEW `ST_Polygon`(ST_LineString `exterior_ring`, ST_LineString `interior_ringi`)

Parameters

- **exterior_ring –** The exterior ring of the polygon
- **interior_ringi –** Interior rings of the polygon

Returns

`ST_Polygon` Returns a polygon created from the specified rings.

Examples

- **Example 1 –** The following returns Polygon ((-5 -1, 5 -1, 0 9, -5 -1), (-2 0, 0 4, 2 0, -2 0)) (a triangle with a triangular hole).

```
SELECT NEW ST_Polygon(
NEW ST_LineString ('LineString (-5 -1, 5 -1, 0 9, -5 -1)'),
NEW ST_LineString ('LineString (-2 0, 0 4, 2 0, -2 0)'))
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 8.3.2

ST_Polygon(ST_MultiLineString , VARCHAR(128)) constructor

Creates a polygon from a multilinestring containing an exterior ring and an optional list of interior rings.

Syntax

NEW `ST_Polygon`(ST_MultiLineString `multi_linestring`, VARCHAR(128) `polygon_format`)

Parameters

- **multi_linestring –** A multilinestring value containing an exterior ring and (optionally) a set of interior rings.
- **polygon_format –** A string with the polygon format to use when interpreting the provided linestrings. Valid formats are 'CounterClockwise', 'Clockwise', and 'EvenOdd'

Returns

`ST_Polygon` Returns a polygon created from the rings in a multilinestrings.

Examples

- **Example 1 –** The following returns Polygon ((-5 -1, 5 -1, 0 9, -5 -1), (-2 0, 0 4, 2 0, -2 0)) (a triangle with a triangular hole).

```
SELECT NEW ST_Polygon(
NEW ST_MultiLineString ('MultiLineString ((-5 -1, 5 -1, 0 9, -5
-1), (-2 0, 0 4, 2 0, -2 0))'))
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 8.3.2

ST_Polygon(ST_Point , ST_Point) constructor

Creates an axis-aligned rectangle from two points representing the lower-left and upper-right corners.

Syntax

NEW `ST_Polygon`(ST_Point `pmin`, ST_Point `pmax`)

Parameters

- **pmin –** A point that is the lower-left corner of the rectangle.
- **pmax –** A point that is the upper-right corner of the rectangle.

Returns

`ST_Polygon` Returns an axis-aligned rectangle.

Examples

- **Example 1 –** The following returns Polygon ((0 0, 4 0, 4 10, 0 10, 0 0)).

```
SELECT NEW ST_Polygon(NEW ST_Point(0.0, 0.0), NEW ST_Point(4.0,
10.0))
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_Polygon() constructor

Constructs a polygon representing the empty set.

Syntax

NEW `ST_Polygon`()

Returns

`ST_Polygon` Returns an ST_Polygon value representing the empty set.

Examples

- **Example 1 –** The following returns 1, indicating the value is empty.

```
SELECT NEW ST_Polygon().ST_IsEmpty()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Standard feature

ST_Polygon(LONG BINARY[, INT]) constructor

Constructs a polygon from Well Known Binary (WKB).

Syntax

NEW `ST_Polygon`(LONG BINARY `wkb`[, INT `srid`])

Parameters

- **wkb –** A string containing the binary representation of a polygon. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_Polygon` Returns an ST_Polygon value constructed from the source string.

Examples

- **Example 1 –** The following returns Polygon ((10 -5, 15 5, 5 5, 10 -5)).

```
SELECT NEW
ST_Polygon(0x010300000001000000040000000000000000002440000000000000
```

```
0014c00000000000002e4000000000000014400000000000001440000000000000
01440000000000002440000000000014c0)
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 8.3.2

ST_Polygon(LONG VARCHAR[, INT]) constructor

Constructs a polygon from a text representation.

Syntax

NEW `ST_Polygon`(LONG VARCHAR `text_representation`[, INT `srid`])

Parameters

- **text_representation –** A string containing the text representation of a polygon. The input can be in any supported text input format, including Well Known Text (WKT) or Extended Well Known Text (EWKT).
- **srid –** The SRID of the result. If not specified, the default is 0.

Returns

`ST_Polygon` Returns an ST_Polygon value constructed from the source string.

Examples

- **Example 1 –** The following returns Polygon ((-5 -5, 5 -5, 0 5, -5 -5), (-2 -2, -2 0, 2 0, 2 -2, -2 -2)).

```
SELECT NEW ST_Polygon('Polygon ((-5 -5, 5 -5, 0 5, -5 -5), (-2 -2,
-2 0, 2 0, 2 -2, -2 -2))')
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 8.3.2

ST_ExteriorRing(ST_Curve) method

Changes the exterior ring of the polygon.

Syntax

`polygon-expression.ST_ExteriorRing`(ST_Curve `curve`)

Parameters

- **curve –** The new exterior ring of the polygon. This must be a linear ring value.

Returns

`ST_Polygon` Returns a copy of the polygon with specified exterior ring.

Examples

- **Example 1 –** The following example returns the result `Polygon ((0 1, 10 1, 5 10, 0 1), (3 3, 3 5, 7 5, 7 3, 3 3))`.

```
SELECT NEW ST_Polygon('Polygon ((0 0, 10 0, 5 10, 0 0), (3 3, 3 5,
7 5, 7 3, 3 3))')
.ST_ExteriorRing( NEW ST_LineString( 'LineString(0 1, 10 1, 5 10,
0 1)' ) )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 8.3.3

ST_InteriorRingN(INT) method

Returns the nth interior ring in the polygon.

Syntax

`polygon-expression.ST_InteriorRingN`(INT `n`)

Parameters

- **n –** The position of the element to return, from 1 to polygon-expression.ST_NumInteriorRing().

Returns

`ST_LineString` Returns the nth interior ring in the polygon.

Examples

- **Example 1 –** The following example returns the result `LineString (3 3, 3 5, 7 5, 7 3, 3 3)`.

```
SELECT NEW ST_Polygon('Polygon ((0 0, 10 0, 5 10, 0 0), (3 3, 3 5,
7 5, 7 3, 3 3))')
.ST_InteriorRingN( 1 )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 8.3.5

ST_SpatialRefSys type

The ST_SpatialRefSys type defines routines for working with spatial reference systems.

Syntax
`ST_SpatialRefSys` type

Members
All members of the ST_SpatialRefSys type, including all inherited members.

Members of ST_SpatialRefSys:

- **ST_CompareWKT(LONG VARCHAR, LONG VARCHAR) –** Compares two spatial reference system definitions.
- **ST_FormatTransformDefinition(LONG VARCHAR) –** Returns a formatted copy of the transform definition.
- **ST_FormatWKT(LONG VARCHAR) –** Returns a formatted copy of the Well Known Text (WKT) definition.
- **ST_GetUnProjectedTransformDefinition(LONG VARCHAR) –** Returns the transform definition of the spatial reference system that is the source of the projection.
- **ST_ParseWKT(VARCHAR(128), LONG VARCHAR) –** Retrieves a named element from the Well Known Text (WKT) definition of a spatial reference system.
- **ST_TransformGeom(ST_Geometry , LONG VARCHAR, LONG VARCHAR) –** Returns the geometry transformed using the given transform definition.
- **ST_World(INT) –** Returns a geometry that represents all of the points in the spatial reference system.

Standards and compatibility
SQL/MM (ISO/IEC 13249-3: 2006) 13.1

ST_CompareWKT(LONG VARCHAR, LONG VARCHAR) method

Compares two spatial reference system definitions.

Syntax

`ST_SpatialRefSys`::`ST_CompareWKT`(LONG VARCHAR `transform_definition_1`, LONG VARCHAR `transform_definition_2`)

Parameters

- **transform_definition_1 –** The first spatial reference system definition text
- **transform_definition_2 –** The second spatial reference system definition text

Returns

`BIT` Returns 1 if the two spatial reference systems are logically equivalent, otherwise 0.

Examples

- **Example 1 –** The following example shows that two spatial reference systems are considered equal even though they have different names:

```
SELECT  ST_SpatialRefSys::ST_CompareWKT(
'GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",
6378137,298.257223563,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG",
"6326"]],PRIMEM["Greenwich",
0,AUTHORITY["EPSG","8901"]],UNIT["degree",
0.01745329251994328,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4
326"]]'
,   'GEOGCS["WGS 84 alternate name",DATUM["WGS_1984",SPHEROID["WGS
84",
6378137,298.257223563,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG",
"6326"]],PRIMEM["Greenwich",
0,AUTHORITY["EPSG","8901"]],UNIT["degree",
0.01745329251994328,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4
326"]]'
) Considered_Equal
```

  The following example shows two spatial reference systems that are considered non-equal because they are defined by different authorities:

```
SELECT  ST_SpatialRefSys::ST_CompareWKT(
'GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",
6378137,298.257223563,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG",
"6326"]],PRIMEM["Greenwich",
0,AUTHORITY["EPSG","8901"]],UNIT["degree",
0.01745329251994328,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4
326"]]'
,   'GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",
6378137,298.257223563,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG",
"6326"]],PRIMEM["Greenwich",
0,AUTHORITY["EPSG","8901"]],UNIT["degree",
0.01745329251994328,AUTHORITY["EPSG","9122"]],AUTHORITY["AnotherA
uthority","4326"]]'
) Considered_NotEqual
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_FormatTransformDefinition(LONG VARCHAR) method

Returns a formatted copy of the transform definition.

Syntax

`ST_SpatialRefSys::ST_FormatTransformDefinition`(LONG VARCHAR `transform_definition`)

Parameters

- **transform_definition –** The spatial reference system transform definition text

Returns

`LONG VARCHAR` Returns a text string defining the transform definition

Examples

- **Example 1 –** The following example returns the result `+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs +towgs84=0,0,0 +no_defs`.

```
SELECT
ST_SpatialRefSys::ST_FormatTransformDefinition('+proj=longlat
+ellps=WGS84 +datum=WGS84 +no_defs')
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_FormatWKT(LONG VARCHAR) method

Returns a formatted copy of the Well Known Text (WKT) definition.

Syntax

`ST_SpatialRefSys::ST_FormatWKT`(LONG VARCHAR `definition`)

Parameters

- **definition –** The spatial reference system definition text

Returns

`LONG VARCHAR` Returns a text string defining the spatial reference system in WKT.

Examples

- **Example 1 –** The following example returns the result `GEOGCS["WGS 84", DATUM["WGS_1984", SPHEROID["WGS 84", 6378137,298.257223563,AUTHORITY["EPSG","7030"]], AUTHORITY["EPSG","6326"]], PRIMEM["Greenwich", 0,AUTHORITY["EPSG","8901"]], UNIT["degree", 0.01745329251994328,AUTHORITY["EPSG","9122"]], AUTHORITY["EPSG","4326"]]`.

```
SELECT  ST_SpatialRefSys::ST_FormatWKT('GEOGCS["WGS
84",DATUM["WGS_1984",SPHEROID["WGS 84",
6378137,298.257223563,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG",
"6326"]],PRIMEM["Greenwich",
```

```
0,AUTHORITY["EPSG","8901"]],UNIT["degree",
0.01745329251994328,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4
326"]]')
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_GetUnProjectedTransformDefinition(LONG VARCHAR) method

Returns the transform definition of the spatial reference system that is the source of the projection.

Syntax

`ST_SpatialRefSys::ST_GetUnProjectedTransformDefinition`(LONG VARCHAR `transform_definition`)

Parameters

- **transform_definition –** The spatial reference system transform definition text

Returns

`LONG VARCHAR` Returns a text string defining the transform definition of the unprojected spatial reference system.

Examples

- **Example 1 –** The following example returns the result `+proj=latlong +a=6371000 +b=6371000 +no_defs`.

```
SELECT
ST_SpatialRefSys::ST_GetUnProjectedTransformDefinition( '+proj=ro
bin +lon_0=0 +x_0=0 +y_0=0 +a=6371000 +b=6371000 +units=m
no_defs' )
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_ParseWKT(VARCHAR(128), LONG VARCHAR) method

Retrieves a named element from the Well Known Text (WKT) definition of a spatial reference system.

Syntax

`ST_SpatialRefSys::ST_ParseWKT`(VARCHAR(128) `element`, LONG VARCHAR `srs_text`)

Parameters

- **element –** The element to retrieve from the WKT. The following named elements may be retrieved:
  - `srs_name`
    The name of the spatial reference system
  - `srs_type`
    The coordinate system type.
  - `organization`
    The name of the organization that defined the spatial reference system.
  - `organization_id`
    The integer identifier assigned by the organization that defined the spatial reference system.
  - `linear_unit_of_measure`
    The name of the linear unit of measure.
  - `linear_unit_of_measure_factor`
    The conversion factor for the linear unit of measure.
  - `angular_unit_of_measure`
    The name of the angular unit of measure.
  - `angular_unit_of_measure_factor`
    The conversion factor for the angular unit of measure.
- **srs_text –** The spatial reference system definition text

Returns

`LONG VARCHAR` Retrieves a named element from the WKT definition of a spatial reference system.

Examples

- **Example 1 –** The following example returns a result with one row for each of the named elements.

```
with V(element,srs_text) as (
SELECT row_value as element, 'GEOGCS["WGS
84",DATUM["WGS_1984",SPHEROID["WGS 84",
6378137,298.257223563,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG",
"6326"]],PRIMEM["Greenwich",
0,AUTHORITY["EPSG","8901"]],UNIT["degree",
0.01745329251994328,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4
326"]]' as srs_text
FROM
sa_split_list('srs_name,srs_type,organization,organization_id,lin
ear_unit_of_measure,linear_unit_of_measure_factor,angular_unit_of
_measure,angular_unit_of_measure_factor') D
)
SELECT element, ST_SpatialRefSys::ST_ParseWKT( element,
```

```
srs_text ) parsed
FROM V
```

The example returns the following result set:

element	parsed
srs_name	WGS 84
srs_type	GEOGRAPHIC
organization	EPSG
organization_id	4326
linear_unit_of_measure	NULL
linear_unit_of_measure_factor	NULL
angular_unit_of_measure	degree
angular_unit_of_measure_factor	.0174532925199433282

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_TransformGeom(ST_Geometry , LONG VARCHAR, LONG VARCHAR) method

Returns the geometry transformed using the given transform definition.

Syntax

ST_SpatialRefSys::ST_TransformGeom(ST_Geometry geom, LONG VARCHAR target_transform_definition, LONG VARCHAR source_transform_definition)

Parameters

- **geom –** The geometry to be transformed
- **target_transform_definition –** The target spatial reference system transform definition text
- **source_transform_definition –** The source spatial reference system transform definition text. If not specified, the transform definition from the spatial reference system of the geom parameter is used.

Returns

`ST_Geometry` Returns the input geometry transformed using the given transform definition.

Examples

- **Example 1 –** The following example returns the result `Point (-5387692.968586 4763459.253243)`.

```
SELECT  ST_SpatialRefSys::ST_TransformGeom( NEW
ST_Point(-63.57,44.65,4326), '+proj=robin +lon_0=0 +x_0=0 +y_0=0
+a=6371000 +b=6371000 +units=m
no_defs' ).ST_AsText('DecimalDigits=6')
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_World(INT) method

Returns a geometry that represents all of the points in the spatial reference system.

Syntax

`ST_SpatialRefSys::ST_World`(INT `srid`)

Parameters

- **srid –** The SRID to use for the result.

Returns

`ST_Surface` Returns a geometry that represents all of the points in the spatial reference system identified by the srid parameter.

Examples

- **Example 1 –** The following example returns the result `Polygon ((-1000000 -1000000, 1000000 -1000000, 1000000 1000000, -1000000 1000000, -1000000 -1000000))`.

```
SELECT ST_SpatialRefSys::ST_World(0)
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) Vendor extension

ST_Surface type

The ST_Surface type is a supertype for 2-dimensional geometry types. The ST_Surface type is not instantiable.

Syntax
`ST_Surface` type

Members
All members of the ST_Surface type, including all inherited members.

Members of ST_Surface:

- **ST_Area(VARCHAR(128)) –** Calculates the area of a surface in the specified units.
- **ST_Centroid() –** Returns the ST_Point value that is the mathematical centroid of the surface value.
- **ST_IsWorld() –** Test if the ST_Surface covers the entire space.
- **ST_Perimeter(VARCHAR(128)) –** Calculates the perimeter of a surface in the specified units.
- **ST_PointOnSurface() –** Returns an ST_Point value that is guaranteed to spatially intersect the ST_Surface value.

Members of ST_Geometry:

- **ST_Affine(DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a new geometry that is the result of applying the specified 3-D affine transformation.
- **ST_AsBinary(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsBitmap(INT, INT, ST_Point , ST_Point , VARCHAR(128)) –** Returns a LONG VARBIT that is a bitmap representing a geometry value.
- **ST_AsGeoJSON(VARCHAR(128)) –** Returns a string representing a geometry in JSON format.
- **ST_AsGML(VARCHAR(128)) –** Returns the GML representation of an ST_Geometry value.
- **ST_AsKML(VARCHAR(128)) –** Returns the KML representation of an ST_Geometry value.
- **ST_AsSVG(VARCHAR(128)) –** Returns an SVG figure representing a geometry value.
- **ST_AsSVGAggr(ST_Geometry , VARCHAR(128)) –** Returns a complete or partial SVG document which renders the geometries in a group.
- **ST_AsText(VARCHAR(128)) –** Returns the text representation of an ST_Geometry value.

- **ST_AsWKB(VARCHAR(128)) –** Returns the WKB representation of an ST_Geometry value.
- **ST_AsWKT(VARCHAR(128)) –** Returns the WKT representation of an ST_Geometry value.
- **ST_AsXML(VARCHAR(128)) –** Returns the XML representation of an ST_Geometry value.
- **ST_Boundary() –** Returns the boundary of the geometry value.
- **ST_Buffer(DOUBLE, VARCHAR(128)) –** 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.
- **ST_Contains(ST_Geometry) –** Tests if a geometry value spatially contains another geometry value.
- **ST_ContainsFilter(ST_Geometry) –** An inexpensive test if a geometry might contain another.
- **ST_ConvexHull() –** Returns the convex hull of the geometry value.
- **ST_ConvexHullAggr(ST_Geometry) –** Returns the convex hull for all of the geometries in a group
- **ST_CoordDim() –** Returns the number of coordinate dimensions stored with each point of the ST_Geometry value.
- **ST_CoveredBy(ST_Geometry) –** Tests if a geometry value is spatially covered by another geometry value.
- **ST_CoveredByFilter(ST_Geometry) –** An inexpensive test if a geometry might be covered by another.
- **ST_Covers(ST_Geometry) –** Tests if a geometry value spatially covers another geometry value.
- **ST_CoversFilter(ST_Geometry) –** An inexpensive test if a geometry might cover another.
- **ST_Crosses(ST_Geometry) –** Tests if a geometry value crosses another geometry value.
- **ST_Debug(VARCHAR(128)) –** Returns a LONG BINARY that is debug information for the object.
- **ST_Difference(ST_Geometry) –** Returns the geometry value that represents the point set difference of two geometries.
- **ST_Dimension() –** Returns the dimension of the ST_Geometry value. Points have dimension 0, lines have dimension 1, and surfaces have dimension 2. Any empty geometry has dimension -1.
- **ST_Disjoint(ST_Geometry) –** Test if a geometry value is spatially disjoint from another value.
- **ST_Distance(ST_Geometry , VARCHAR(128)) –** Returns the smallest distance between the geometry-expression and the specified geometry value.

- **ST_Distance_Spheroid(ST_Geometry , VARCHAR(128)) –** Calculates the linear distance between geometries on the surface of the Earth.
- **ST_Envelope() –** Returns the bounding rectangle for the geometry value.
- **ST_EnvelopeAggr(ST_Geometry) –** Returns the bounding rectangle for all of the geometries in a group
- **ST_Equals(ST_Geometry) –** Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.
- **ST_EqualsFilter(ST_Geometry) –** An inexpensive test if a geometry is equal to another.
- **ST_GeometryType() –** Returns the name of the type of the ST_Geometry value.
- **ST_GeometryTypeFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_GeomFromBinary(LONG BINARY, INT) –** Constructs a geometry from a binary string representation.
- **ST_GeomFromShape(LONG BINARY[, INT]) –** Parses a string containing an ESRI shape record and creates a geometry value of the appropriate type.
- **ST_GeomFromText(LONG VARCHAR, INT) –** Constructs a geometry from a character string representation.
- **ST_GeomFromWKB(LONG BINARY, INT) –** Parse a string containing a WKB or EWKB representation of a geometry and creates a geometry value of the appropriate type.
- **ST_GeomFromWKT(LONG VARCHAR, INT) –** Parses a string containing the WKT or EWKT representation of a geometry and create a geometry value of the appropriate type.
- **ST_Intersection(ST_Geometry) –** Returns the geometry value that represents the point set intersection of two geometries.
- **ST_IntersectionAggr(ST_Geometry) –** Returns the spatial intersection of all of the geometries in a group
- **ST_Intersects(ST_Geometry) –** Test if a geometry value spatially intersects another value.
- **ST_IntersectsFilter(ST_Geometry) –** An inexpensive test if the two geometries might intersect.
- **ST_IntersectsRect(ST_Point , ST_Point) –** Test if a geometry intersects a rectangle.
- **ST_Is3D() –** Determines if the geometry value has Z coordinate values.
- **ST_IsEmpty() –** Determines whether the geometry value represents an empty set.
- **ST_IsMeasured() –** Determines if the geometry value has associated measure values.
- **ST_IsSimple() –** Determines whether the geometry value is simple (containing no self intersections or other irregularities).
- **ST_IsValid() –** Determines whether the geometry is a valid spatial object.
- **ST_LatNorth() –** Retrieves the northernmost latitude of a geometry.
- **ST_LatSouth() –** Retrieves the southernmost latitude of a geometry.

- **ST_Length_Spheroid(VARCHAR(128)) –** Calculates the linear length of a curve/multicurve on the surface of the Earth.
- **ST_LinearHash() –** Returns a binary string that is a linear hash of the geometry.
- **ST_LinearUnHash(BINARY(32)[, INT]) –** Returns a geometry representing the index hash.
- **ST_LoadConfigurationData(VARCHAR(128)) –** Returns binary configuration data. For internal use only.
- **ST_LocateAlong(DOUBLE) –** Returns the subset of the geometry value that is associated with the given measure value.
- **ST_LocateBetween(DOUBLE, DOUBLE) –** Returns the subset of the geometry value that is between the specified start measure and end measure.
- **ST_LongEast() –** Retrieves the longitude of the eastern boundary of a geometry.
- **ST_LongWest() –** Retrieves the longitude of the western boundary of a geometry.
- **ST_MMax() –** Retrieves the maximum M coordinate value of a geometry.
- **ST_MMin() –** Retrieves the minimum M coordinate value of a geometry.
- **ST_OrderingEquals(ST_Geometry) –** Tests if a geometry is identical to another geometry.
- **ST_Overlaps(ST_Geometry) –** Tests if a geometry value overlaps another geometry value.
- **ST_Relate(ST_Geometry) –** Determines how a geometry value is spatially related to another geometry value by returning an intersection matrix. The ST_Relate method returns 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.
- **ST_Reverse() –** Returns the geometry with the element order reversed.
- **ST_Segmentize(DOUBLE) –** Add points so that no line segment is longer than a specified distance.
- **ST_Simplify(DOUBLE) –** Remove points from curves so long as the maximum introduced error is less than a specified tolerance.
- **ST_SnapToGrid(ST_Point , DOUBLE, DOUBLE, DOUBLE, DOUBLE) –** Returns a copy of the geometry with all points snapped to the specified grid.
- **ST_SRID(INT) –** Changes the spatial reference system associated with the geometry without modifying any of the values.
- **ST_SRIDFromBaseType(VARCHAR(128)) –** Parses a string defining the type string.
- **ST_SymDifference(ST_Geometry) –** Returns the geometry value that represents the point set symmetric difference of two geometries.
- **ST_ToCircular() –** Convert the geometry to a circularstring
- **ST_ToCompound() –** Converts the geometry to a compound curve.
- **ST_ToCurve() –** Converts the geometry to a curve.
- **ST_ToCurvePoly() –** Converts the geometry to a curve polygon.

- **ST_ToGeomColl() –** Converts the geometry to a geometry collection.
- **ST_ToLineString() –** Converts the geometry to a linestring.
- **ST_ToMultiCurve() –** Converts the geometry to a multicurve value.
- **ST_ToMultiLine() –** Converts the geometry to a multilinestring value.
- **ST_ToMultiPoint() –** Converts the geometry to a multi-point value.
- **ST_ToMultiPolygon() –** Converts the geometry to a multi-polygon value.
- **ST_ToMultiSurface() –** Converts the geometry to a multi-surface value.
- **ST_ToPoint() –** Converts the geometry to a point.
- **ST_ToPolygon() –** Converts the geometry to a polygon.
- **ST_ToSurface() –** Converts the geometry to a surface.
- **ST_Touches(ST_Geometry) –** Tests if a geometry value spatially touches another geometry value.
- **ST_Transform(INT) –** Creates a copy of the geometry value transformed into the specified spatial reference system.
- **ST_Union(ST_Geometry) –** Returns the geometry value that represents the point set union of two geometries.
- **ST_UnionAggr(ST_Geometry) –** Returns the spatial union of all of the geometries in a group
- **ST_Within(ST_Geometry) –** Tests if a geometry value is spatially contained within another geometry value.
- **ST_WithinDistance(ST_Geometry , DOUBLE, VARCHAR(128)) –** Test if two geometries are within a specified distance of each other.
- **ST_WithinDistanceFilter(ST_Geometry , DOUBLE, VARCHAR(128)) –** An inexpensive of whether two geometries might be within a specified distance of each other.
- **ST_WithinFilter(ST_Geometry) –** An inexpensive test if a geometry might be within another.
- **ST_XMax() –** Retrieves the maximum X coordinate value of a geometry.
- **ST_XMin() –** Retrieves the minimum X coordinate value of a geometry.
- **ST_YMax() –** Retrieves the maximum Y coordinate value of a geometry.
- **ST_YMin() –** Retrieves the minimum Y coordinate value of a geometry.
- **ST_ZMax() –** Retrieves the maximum Z coordinate value of a geometry.
- **ST_ZMin() –** Retrieves the minimum Z coordinate value of a geometry.

Standards and compatibility
SQL/MM (ISO/IEC 13249-3: 2006) 8.1

ST_Area(VARCHAR(128)) method

Calculates the area of a surface in the specified units.

Syntax

```
surface-expression.ST_Area(VARCHAR(128) unit_name)
```

Parameters

- **unit_name –** The units in which the length 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 area of the surface.

Examples

- **Example 1 –** The following example returns the result `12.5`.

```
SELECT TREAT( Shape AS ST_Polygon ).ST_Area()
FROM SpatialShapes WHERE ShapeID = 22
```

  The following returns the area of the poly_geometry column in square miles from the fictional region table.

```
SELECT name, poly_geometry.ST_Area( 'Statute Mile' )
FROM region
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 8.1.2

ST_Centroid() method

Returns the ST_Point value that is the mathematical centroid of the surface value.

Syntax

`surface-expression.ST_Centroid()`

Returns

`ST_Point` If the surface is the empty set, returns NULL. Otherwise, returns the mathematical centroid of the surface.

Examples

- **Example 1 –** The following example returns the result `Point (5 4.666667)`.

```
SELECT TREAT( Shape as ST_Surface ).ST_Centroid()
FROM SpatialShapes WHERE ShapeID = 22
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 8.1.4

ST_IsWorld() method

Test if the ST_Surface covers the entire space.

Syntax

```
surface-expression.ST_IsWorld()
```

Returns

`BIT` Returns 1 if the surface covers the entire space, otherwise 0.

Examples

- **Example 1 –** The following example returns the result `1`.

```
SELECT NEW ST_Polygon( NEW ST_Point( -180, -90, 1000004326 ),
NEW ST_Point( 180, 90, 1000004326 ) ).ST_IsWorld()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 8.1.6

ST_Perimeter(VARCHAR(128)) method

Calculates the perimeter of a surface in the specified units.

Syntax

```
surface-expression.ST_Perimeter(VARCHAR(128) unit_name)
```

Parameters

- **unit_name –** The units in which the length 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 surface in the specified unit of measure.

Examples

- **Example 1 –** The following example returns the result `18`.

```
SELECT TREAT( Shape as ST_Surface ).ST_Perimeter()
FROM SpatialShapes WHERE ShapeID = 3
```

  The following returns the perimeter of the poly_geometry column in miles from the fictional region table.

```
SELECT name, poly_geometry.ST_Perimeter( 'Statute Mile' )
FROM region
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 8.1.3

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 ST_Surface value.

Examples

- **Example 1 –** The following returns a point that intersects the polygon.

```
SELECT NEW ST_Polygon( 'Polygon(( 1 0, 0 10, 1 1, 2 10, 1 0 ))' )
.ST_PointOnSurface()
```

Standards

SQL/MM (ISO/IEC 13249-3: 2006) 8.1.5

Appendix – SQL Statements

Reference material for SQL statements mentioned in this document.

CREATE SPATIAL REFERENCE SYSTEM Statement

Creates or replaces a spatial reference system.

Quick Links:

Go to Parameters on page 270

Go to Examples on page 275

Go to Usage on page 275

Go to Standards on page 276

Go to Permissions on page 276

Syntax

```
{ CREATE [ OR REPLACE ] SPATIAL REFERENCE SYSTEM
     | CREATE SPATIAL REFERENCE SYSTEM IF NOT EXISTS }
    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

(back to top) on page 269

- **OR REPLACE –** Specifying OR REPLACE creates the spatial reference system if it does not already exist in the database, and replaces it if it does exist. An error is returned if you attempt to replace a spatial reference system while it is in use. An error is also returned if you attempt to replace a spatial reference system that already xists in the database without specifying the OR REPLACE clause.
- **IF NOT EXISTS –** Specifying CREATE SPATIAL REFERENCE IF NOT EXISTS checks to see if a spatial reference system by that name already exists. If it does not exist, the database server creates the spatial reference system. If it does exist, no further action is performed and no error is returned.
- **IDENTIFIED BY –** the SRID (*srs-id*) for the spatial reference system. If the spatial reference system is defined by an organization with an *organization-srs-id*, then *srs-id* should be set to that value.

  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.

  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.

  With the exception of SRID 0, spatial reference systems provided by SAP Sybase IQ 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 Sybase IQ and you should not create SRIDs in this range.

  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.

  Also, since the SRID is stored as a signed 32-bit integer, the number cannot exceed 231-1 or 2147483647.

- **DEFINITION –** set, or override, default coordinate system settings. If any attribute is set in a clause other than the DEFINITION clause, it takes the value specified in the other clause regardless of what is specified in the DEFINITION clause.

  definition-string is a string in the Spatial Reference System Well Known Text syntax as defined by SQL/MM and OGC. For example, the following query returns the definition for WGS 84.

  ```
  SELECT ST_SpatialRefSys::ST_FormatWKT( definition )
      FROM ST_SPATIAL_REFERENCE_SYSTEMS
      WHERE srs_id=4326;
  ```

  In Interactive SQL, if you double-click the value returned, an easier to read version of the value appears.

  When the DEFINITION clause is specified, definition-string is parsed and used to choose default values for attributes. For example, definition-string may contain an AUTHORITY element that defines the organization-name and *organization-srs-id*.

  Parameter values in definition-string are overridden by values explicitly set using the SQL statement clauses. For example, if the ORGANIZATION clause is specified, it overrides the value for ORGANIZATION in definition-string.

- **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.

- **LINEAR UNIT OF MEASURE –** the linear unit of measure for the spatial reference system. The value you specify must match a linear unit of measure defined in the `ST_UNITS_OF_MEASURE` system view.

  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.

  To add custom units of measure to the database, use the CREATE SPATIAL UNIT OF MEASURE statement.

  Note: While both METRE and METER are accepted spellings, METRE is preferred as it conforms to the SQL/MM standard.

- **ANGULAR UNIT OF MEASURE –** the angular unit of measure for the spatial reference system. The value you specify must match an angular unit of measure defined in the `ST_UNITS_OF_MEASURE` system table.

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.

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.

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. To add predefined units of measure to the database, use the sa_install_feature system procedure.

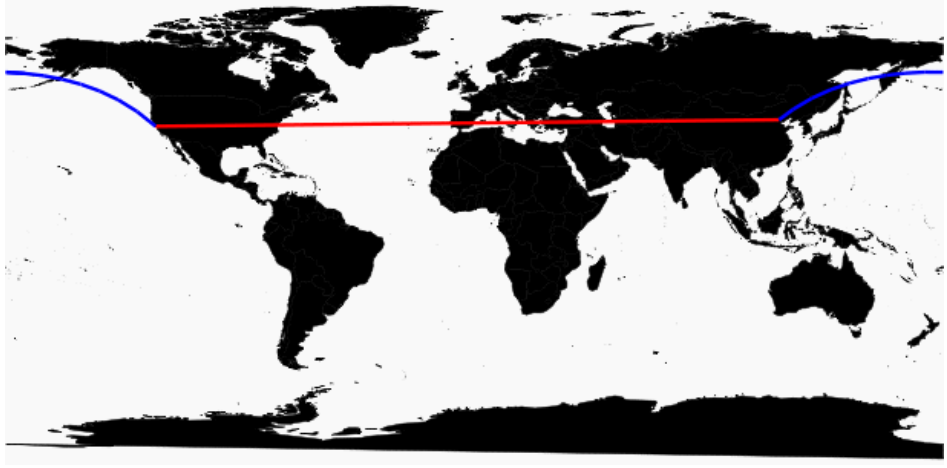
To add custom units of measure to the database, use the CREATE SPATIAL UNIT OF MEASURE statement.

- **TYPE –** control how the SRS interprets lines between points. 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.

  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 anti-podal points are not valid and give an error in the ROUND EARTH model.

  The ROUND EARTH 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 x=long, y=lat.

  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.



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.

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).

- **COORDINATE –** the bounds on the spatial reference system's dimensions. coordinate-name is the name of the coordinate system used by the spatial reference system. For non-geographic coordinate systems, coordinate-name can be x, y, or m. For geographic coordinate systems, coordinate-name can be LATITUDE, LONGITUDE, z, or m.

  Specify UNBOUNDED to place no bounds on the dimensions. Use the BETWEEN clause to set low and high bounds.

  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 the unless COORDINATE clause overrides these settings. For non-geographic spatial reference systems, the CREATE statement must specify bounds for both X and Y coordinates.

  LATITUDE and LONGITUDE are used for geographic coordinate systems. The bounds for LATITUDE and LONGITUDE default to the entire Earth, if not specified.

- **ELLIPSOID –** 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. If the ELLIPSOID clause is specified, it overrides this default ellipsoid.

  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 (1/f) is a ratio:

  ```
  1/f = (semi-major-axis) / (semi-major-axis - semi-minor-axis)
  ```

  product-name uses the ellipsoid definition when computing distance in geographic spatial reference systems.

- **SNAP TO GRID –** flat-Earth (planar) spatial reference systems, use the SNAP TO GRID clause to define the size of the grid SAP Sybase IQ uses when performing calculations. By

default, SAP Sybase IQ 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.

- **TOLERANCE –** flat-Earth (planar) spatial reference systems, use the TOLERANCE clause to specify the precision to use when comparing points. 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.

  When set to 0, two points must be exactly equal to be considered equal.

  For round-Earth spatial reference systems, TOLERANCE must be set to 0.

- **POLYGON FORMAT –** internally, SAP Sybase IQ 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 Sybase IQ 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 POLYGON FORMAT clause can be used to select a polygon interpretation that matches the input data, as needed. The following values are supported:

  - **CounterClockwise –** input follows SAP Sybase IQ's internal interpretation: the inside of the polygon is on the left side while following ring orientation.
  - **Clockwise –** input follows the opposite of SAP Sybase IQ's approach: the inside of the polygon is on the right side while following ring orientation.
  - **EvenOdd –** (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.

- **STORAGE FORMAT –** control what is stored when spatial data is loaded into the database. Possible values are:

  - **Internal –** SAP Sybase IQ 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).
  - **Original –** SAP Sybase IQ 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.
  - **Mixed –** SAP Sybase IQ stores the internal version and, if it is different from the original version, SAP Sybase SQL Anywhere® stores the original version as well. By storing both versions, the original representation characteristics can be reproduced and

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).

Examples

(back to top) on page 269

- **Example 1 –** creates a spatial reference system named mySpatialRS:

```
CREATE SPATIAL REFERENCE SYSTEM "mySpatialRS"
IDENTIFIED BY 1000026980
LINEAR UNIT OF MEASURE "metre"
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",0.01745329251994328,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",36.33333333333334],
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=36.3333333
3333334+lon_0=-85.75+x_0=500000+y_0=500000+ellps=GRS80+datum=NAD8
3+units=m+no_defs';
```

Usage

(back to top) on page 269

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.

Standards

(back to top) on page 269

ANSI SQL–Compliance level: Transact-SQL® extension.

Permissions

(back to top) on page 269

Requires one of:

- MANAGE ANY SPATIAL OBJECT system privilege.
- CREATE ANY OBJECT system privilege.

CREATE SPATIAL UNIT OF MEASURE Statement

Creates or replaces a spatial unit of measurement.

Quick Links:

Go to Parameters on page 276

Go to Examples on page 277

Go to Usage on page 277

Go to Standards on page 277

Go to Permissions on page 278

Syntax

```
CREATE [ OR REPLACE ] SPATIAL UNIT OF MEASURE identifier
    TYPE { LINEAR | ANGULAR }
    [ CONVERT USING number ]
```

Parameters

(back to top) on page 276

- **OR REPLACE –** including the OR REPLACE creates a new spatial unit of measure, or replaces an existing spatial unit of measure with the same name. This clause preserves existing privileges. An error is returned if you attempt to replace a spatial unit that is already in use.

- **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 RADIAN.

Examples

(back to top) on page 276

- **Example 1 –** creates a spatial unit of measure named `Test`:

```
CREATE SPATIAL UNIT OF MEASURE Test
TYPE LINEAR
CONVERT USING 15;
```

Usage

(back to top) on page 276

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.

Standards

(back to top) on page 276

ANSI SQL–Compliance level: Transact-SQL extension.

Permissions

(back to top) on page 276

Requires one of:

- MANAGE ANY SPATIAL OBJECT system privilege.
- CREATE ANY OBJECT system privilege.

DROP SPATIAL UNIT OF MEASURE Statement

Drops a spatial unit of measurement.

Quick Links:

Go to Parameters on page 278

Go to Examples on page 278

Go to Standards on page 278

Go to Permissions on page 279

Syntax

DROP SPATIAL UNIT OF MEASURE [IF EXISTS] *identifier*

Parameters

(back to top) on page 278

- **IF EXISTS –** prevents an error from being returned when the **DROP SPATIAL UNIT OF MEASURE** statement attempts to remove a spatial unit of measure that does not exist.

Examples

(back to top) on page 278

- **Example –** the following example drops a fictitious spatial unit of measure named `Test`:

```
DROP SPATIAL UNIT OF MEASURE Test;
```

Standards

(back to top) on page 278

ANSI SQL–Compliance level: Transact-SQL extension.

Permissions

(back to top) on page 278

Requires one of:

- MANAGE ANY SPATIAL OBJECT system privilege.
- DROP ANY OBJECT system privilege.
- You own the spatial unit of measure.

DROP SPATIAL REFERENCE SYSTEM Statement

Drops a spatial reference system.

Quick Links:

Go to Parameters on page 279

Go to Standards on page 279

Go to Permissions on page 279

Syntax

```
DROP SPATIAL REFERENCE SYSTEM [ IF EXISTS ] name
```

Parameters

(back to top) on page 279

- **IF EXISTS –** prevents an error from being returned when the **DROP SPATIAL REFERENCE SYSTEM** statement attempts to remove a spatial reference system that does not exist.

Standards

(back to top) on page 279

ANSI SQL–Compliance level: Transact-SQL extension.

Permissions

(back to top) on page 279

Requires one of:

- MANAGE ANY SPATIAL OBJECT system privilege.
- DROP ANY OBJECT system privilege.
- You own the spatial references system.

ALTER SPATIAL REFERENCE SYSTEM Statement

Changes the settings of an existing spatial reference system.

Quick Links:

Go to Parameters on page 281

Go to Examples on page 285

Go to Usage on page 285

Go to Standards on page 285

Go to Permissions on page 286

Syntax

```
ALTER 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

(back to top) on page 280

- **IDENTIFIED BY –** the SRID number for the spatial reference system.
- **DEFINITION –** set, or override, default coordinate system settings. If any attribute is set in a clause other than the DEFINITION clause, it takes the value specified in the other clause regardless of what is specified in the DEFINITION clause.

  definition-string is a string in the Spatial Reference System Well Known Text syntax as defined by SQL/MM and OGC. For example, the following query returns the definition for WGS 84.

  ```
  SELECT ST_SpatialRefSys::ST_FormatWKT( definition )
      FROM ST_SPATIAL_REFERENCE_SYSTEMS
      WHERE srs_id=4326;
  ```

  In Interactive SQL, if you double-click the value returned, an easier to read version of the value appears.

  When the DEFINITION clause is specified, definition-string is parsed and used to choose default values for attributes. For example, definition-string may contain an AUTHORITY element that defines the organization-name and *organization-srs-id*.

  Parameter values in definition-string are overridden by values explicitly set using the SQL statement clauses. For example, if the ORGANIZATION clause is specified, it overrides the value for ORGANIZATION in definition-string.
- **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.
- **LINEAR UNIT OF MEASURE –** the linear unit of measure for the spatial reference system. The value you specify must match a linear unit of measure defined in the `ST_UNITS_OF_MEASURE` system view.

  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.

  To add custom units of measure to the database, use the CREATE SPATIAL UNIT OF MEASURE statement.

  Note: While both METRE and METER are accepted spellings, METRE is preferred as it conforms to the SQL/MM standard.

- **ANGULAR UNIT OF MEASURE –** the angular unit of measure for the spatial reference system. The value you specify must match an angular unit of measure defined in the `ST_UNITS_OF_MEASURE` system table.

  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.

  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.

  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. To add predefined units of measure to the database, use the sa_install_feature system procedure.

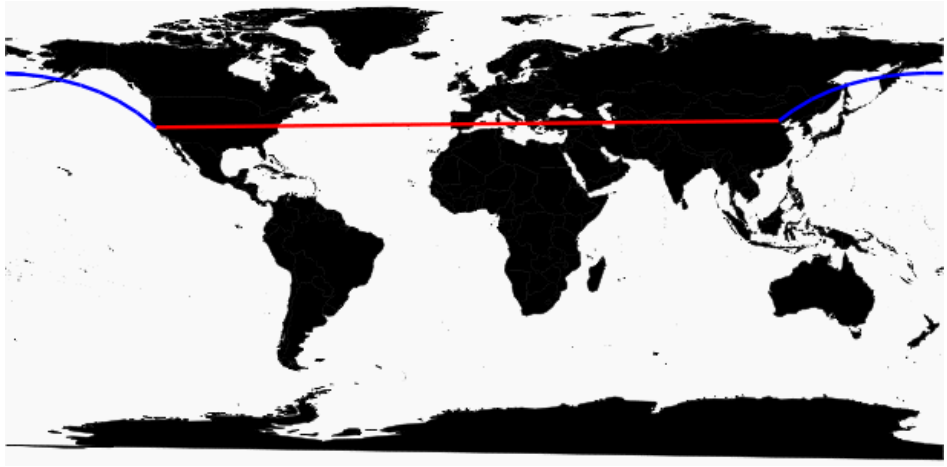
  To add custom units of measure to the database, use the CREATE SPATIAL UNIT OF MEASURE statement.

- **TYPE –** control how the SRS interprets lines between points. 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.

  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 anti-podal points are not valid and give an error in the ROUND EARTH model.

  The ROUND EARTH 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 x=long, y=lat.

  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.



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.

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).

- **COORDINATE –** the bounds on the spatial reference system's dimensions. coordinate-name is the name of the coordinate system used by the spatial reference system. For non-geographic coordinate systems, coordinate-name can be x, y, or m. For geographic coordinate systems, coordinate-name can be LATITUDE, LONGITUDE, z, or m.

  Specify UNBOUNDED to place no bounds on the dimensions. Use the BETWEEN clause to set low and high bounds.

  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 the unless COORDINATE clause overrides these settings. For non-geographic spatial reference systems, the CREATE statement must specify bounds for both X and Y coordinates.

  LATITUDE and LONGITUDE are used for geographic coordinate systems. The bounds for LATITUDE and LONGITUDE default to the entire Earth, if not specified.

- **ELLIPSOID –** 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. If the ELLIPSOID clause is specified, it overrides this default ellipsoid.

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 (1/f) is a ratio:

```
1/f = (semi-major-axis) / (semi-major-axis - semi-minor-axis)
```

product-name uses the ellipsoid definition when computing distance in geographic spatial reference systems.

- **SNAP TO GRID –** flat-Earth (planar) spatial reference systems, use the SNAP TO GRID clause to define the size of the grid SAP Sybase IQ uses when performing calculations. By default, SAP Sybase IQ 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.
- **TOLERANCE –** flat-Earth (planar) spatial reference systems, use the TOLERANCE clause to specify the precision to use when comparing points. 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.

  When set to 0, two points must be exactly equal to be considered equal.

  For round-Earth spatial reference systems, TOLERANCE must be set to 0.
- **POLYGON FORMAT –** internally, SAP Sybase IQ 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 Sybase IQ 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 POLYGON FORMAT clause can be used to select a polygon interpretation that matches the input data, as needed. The following values are supported:

  - **CounterClockwise –** input follows SAP Sybase IQ's internal interpretation: the inside of the polygon is on the left side while following ring orientation.
  - **Clockwise –** input follows the opposite of SAP Sybase IQ's approach: the inside of the polygon is on the right side while following ring orientation.
  - **EvenOdd –** (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.

- **STORAGE FORMAT –** control what is stored when spatial data is loaded into the database. Possible values are:
  - **Internal –** SAP Sybase IQ 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).
  - **Original –** SAP Sybase IQ 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.
  - **Mixed –** SAP Sybase IQ stores the internal version and, if it is different from the original version, SAP Sybase SQL Anywhere® stores the original version as well. By storing both versions, the original representation characteristics can be reproduced and 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).

Examples

(back to top) on page 280

- **Example –** changes the polygon format of a fictitious spatial reference system named mySpatialRef to EvenOdd:

```
ALTER SPATIAL REFERENCE SYSTEM mySpatialRef
POLYGON FORMAT 'EvenOdd';
```

Usage

(back to top) on page 280

You cannot alter a spatial reference system if there is existing data that references it. For example, if you have a column declared as ST_Point(SRID=8743), you cannot alter the spatial reference system with SRID 8743. This is because many spatial reference system attributes, such as storage format, impact the storage format of the data. If you have data that references the SRID, create a new spatial reference system and transform the data to the new SRID.

Standards

(back to top) on page 280

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

(back to top) on page 280

Requires one of:

- You are the owner of the spatial reference system.
- ALTER privilege on the spatial reference system.
- MANAGE ANY SPATIAL OBJECT system privilege.
- ALTER ANY OBJECT system privilege.

ALTER TABLE Statement

Modifies a table definition.

Quick Links:

Go to Parameters on page 289

Go to Examples on page 296

Go to Usage on page 298

Go to Standards on page 299

Go to Permissions on page 299

Syntax

Syntax 1 - Alter Owner

```
ALTER TABLE table_name ALTER OWNER TO new_owner
    [ { PRESERVE | DROP } PERMISSIONS ]
    [ { PRESERVE | DROP } FOREIGN KEYS ]
```

Syntax 2

```
ALTER TABLE [ owner.]table-name
    |{ ENABLE | DISABLE } RLV STORE
    { alter-clause, ... }

alter-clause – (back to Syntax 2)
    ADD create-clause
        | ALTER column-name column-alteration
        | ALTER [ CONSTRAINT constraint-name ] CHECK ( condition )
        | DROP drop-object
        | RENAME rename-object
        | move-clause
        | SPLIT PARTITION range-partition-name
              INTO ( range-partition-decl, range-partition-decl )
        | MERGE PARTITION partition-name-1 INTO partition-name-2
        | UNPARTITION
        | PARTITION BY
                        range-partitioning-scheme
                    | hash-partitioning-scheme
```

```
                    | composite-partitioning-schemecomposite-partitioning-scheme

create-clause – (back to alter-clause)
    column-name column-definition [ column-constraint ]
    | table-constraint
    | [ PARTITION BY ] range-partitioning-scheme

column definition – (back to create-clause)
     column-name data-type [ NOT NULL | NULL ]
     [ IN dbspace-name ]
     [ DEFAULT default-value | IDENTITY ]

column-constraint – (back to create-clause)
    [ CONSTRAINT constraint-name ]
    { UNIQUE
      | PRIMARY KEY
      | REFERENCES table-name [ (column-name ) ] [ actions ]
      | CHECK ( condition )
      | IQ UNIQUE ( integer )
    }

table-constraint – (back to create-clause)
    [ CONSTRAINT constraint-name ]
     { UNIQUE ( column-name [ , … ] )
       | PRIMARY KEY ( column-name [ , … ] )
       | foreign-key-constraint
       | CHECK ( condition )
    }

foreign-key-constraint – (back to table-constraint)
    FOREIGN KEY [ role-name ] [ ( column-name [ , … ] ) ]
    ... REFERENCES table-name [ ( column-name [ , … ] ) ]
    ... [ actions ]

actions – (back to foreign-key-constraint)
    [ ON { UPDATE | DELETE } { RESTRICT } ]

column-alteration – (back to alter-clause)
   { column-data-type | alterable-column-attribute } [ alterable-column-attribute … ]

     | ADD [ constraint-name ] CHECK ( condition )
     | DROP { DEFAULT | CHECK | CONSTRAINT constraint-name }

alterable-column-attribute – (back to column-alteration)
    [ NOT ] NULL
    | DEFAULT default-value
    | [ CONSTRAINT constraint-name ] CHECK { NULL |( condition )
      }

default-value – (back to alterable-column-attribute)
   CURRENT { DATABASE |DATE |REMOTE USER |TIME |TIMESTAMP | USER |
PUBLISHER )
    | string
    | global variable
    | [ - ] number
    | ( constant-expression )
```

```
    | built-in-function ( constant-expression )
    | AUTOINCREMENT
    | NULL
    | TIMESTAMP
    | LAST USER
    | USER

drop-object – (back to alter-clause)
    { column-name
    | CHECK constraint-name
    | CONSTRAINT
    | UNIQUE ( index-columns-list )
    | PRIMARY KEY
    | FOREIGN KEY fkey-name
    | [ PARTITION ] range-partition-name
   }

rename-object – (back to alter-clause)
    new-table-name
     | column-name TO new-column-name
     | CONSTRAINT constraint-name TO new-constraint-name
     | [ PARTITION ] range-partition-name TO new-range-partition-name

move-clause – (back to alter-clause)
    { ALTER column-name
          MOVE
          { PARTITION ( range-partition-name TO new-dbspace-name)
             | TO new-dbspace-name }
           }
          | MOVE PARTITION range-partition-name TO new-dbspace-name
          | MOVE TO new-dbspace-name
          | MOVE TABLE METADATA TO new-dbspace-name
    }
```

```
range-partitioning-scheme – (back to alter-clause)
    RANGE ( partition-key )
      ( range-partition-decl [,range-partition-decl ...] )

partition-key – (back to range-partitioning-scheme)
    column-name

range-partition-decl – (back to alter-clause) or (back to range-
partitioning-scheme)
   range-partition-name VALUES <= ( {constant | MAX } ) [ IN dbspace-
name ]

hash-partitioning-scheme – (back to alter-clause) or (back to composite-
partitioning-scheme)
    HASH ( partition-key, … ] )

composite-partitioning-scheme – (back to alter-clause)
    hash-partitioning-scheme SUBPARTITION range-partitioning-scheme
```

Parameters

(back to top) on page 286

- **{ ENABLE | DISABLE } RLV STORE –** registers this table with the RLV store for real-time in-memory updates. Not supported for IQ temporary tables or in multiplex environments. This value overrides the value of the database option **BASE_TABLES_IN_RLV**.
- **ADD column-definition [column-constraint] –** add a new column to the table.

  The table must be empty to specify `NOT NULL`. The table might contain data when you add an `IDENTITY` or `DEFAULT AUTOINCREMENT` column. If the column has a default `IDENTITY` value, all rows of the new column are populated with sequential values. You can also add `FOREIGN` constraint as a column constraint for a single column key. The value of the `IDENTITY/DEFAULT AUTOINCREMENT` column uniquely identifies every row in a table.

  The `IDENTITY/DEFAULT AUTOINCREMENT` column stores sequential numbers that are automatically generated during inserts and updates. `DEFAULT AUTOINCREMENT` columns are also known as `IDENTITY` columns. When using `IDENTITY/DEFAULT AUTOINCREMENT`, the column must be one of the integer data types, or an exact numeric type, with scale 0. See *CREATE TABLE Statement* for more about column constraints and `IDENTITY/DEFAULT AUTOINCREMENT` columns.

  IQ UNIQUE constraint – Defines the expected cardinality of a column and determines whether the column loads as Flat FP or NBit FP. An IQ UNIQUE(*n*) value explicitly set to 0 loads the column as Flat FP. Columns without an IQ UNIQUE constraint implicitly load as NBit up to the limits defined by the FP_NBIT_AUTOSIZE_LIMIT, FP_NBIT_LOOKUP_MB, and FP_NBIT_ROLLOVER_MAX_MB options.

  Using IQ UNIQUE with an *n* value less than the FP_NBIT_AUTOSIZE_LIMIT is not necessary. Auto-size functionality automatically sizes all low or medium cardinality columns as NBit. Use IQ UNIQUE in cases where you want to load the column as Flat FP or when you want to load a column as NBit when the number of distinct values exceeds the FP_NBIT_AUTOSIZE_LIMIT.

  Note:

  - Consider memory usage when specifying high IQ UNIQUE values. If machine resources are limited, avoid loads with FP_NBIT_ENFORCE_LIMITS='OFF' (default).

    Prior to SAP Sybase IQ 16.0, an IQ UNIQUE *n* value > 16777216 would rollover to Flat FP. In 16.0, larger IQ UNIQUE values are supported for tokenization, but may require significant memory resource requirements depending on cardinality and column width.
  - BIT, BLOB,and CLOB data types do not support NBit dictionary compression. If FP_NBIT_IQ15_COMPATIBILITY='OFF', a non-zero IQ UNIQUE column

specification in a CREATE TABLE or ALTER TABLE statement that includes these data types returns an error.

- **ALTER *column-name* column-alteration –** change the column definition:
  - **SET DEFAULT** *default-value* – Change the default value of an existing column in a table. You can also use the MODIFY clause for this task, but ALTER is `ISO/ANSI SQL` compliant, and MODIFY is not. Modifying a default value does not change any existing values in the table.
  - **DROP DEFAULT** – Remove the default value of an existing column in a table. You can also use the MODIFY clause for this task, but ALTER is `ISO/ANSI SQL` compliant, and MODIFY is not. Dropping a default does not change any existing values in the table.
  - **ADD** – Add a named constraint or a CHECK condition to the column. The new constraint or condition applies only to operations on the table after its definition. The existing values in the table are not validated to confirm that they satisfy the new constraint or condition.
  - **CONSTRAINT** *column-constraint-name* – The optional column constraint name lets you modify or drop individual constraints at a later time, rather than having to modify the entire column constraint.
  - [**CONSTRAINT** *constraint-name*] **CHECK** (*condition*) – Use this clause to add a CHECK constraint on the column.
  - **SET COMPUTE** (*expression*) – Change the expression associated with a computed column. The values in the column are recalculated when the statement is executed, and the statement fails if the new expression is invalid.
  - **DROP COMPUTE** – Change a column from being a computed column to being a non-computed column. This statement does not change any existing values in the table.
- **ADD table-constraint –** add a constraint to the table.

  You can also add a foreign key constraint as a table constraint for a single-column or multicolumn key. If PRIMARY KEY is specified, the table must not already have a primary key created by the CREATE TABLE statement or another ALTER TABLE statement. See *CREATE TABLE Statement* for a full explanation of table constraints.

  Note: You cannot MODIFY a table or column constraint. To change a constraint, DELETE the old constraint and ADD the new constraint.
- **DROP *drop-object* –** drops a table object:
  - **DROP** *column-name* – Drop the column from the table. If the column is contained in any multicolumn index, uniqueness constraint, foreign key, or primary key, then the index, constraint, or key must be deleted before the column can be deleted. This does not delete CHECK constraints that refer to the column. An `IDENTITY/DEFAULT`

AUTOINCREMENT column can only be deleted if IDENTITY_INSERT is turned off and the table is not a local temporary table.

- **DROP CHECK** – Drop all check constraints for the table. This includes both table check constraints and column check constraints.
- **DROP CONSTRAINT** *constraint-name* – Drop the named constraint for the table or specified column.
- **DROP UNIQUE** (*column-name, ...*) – Drop the unique constraints on the specified column(s). Any foreign keys referencing the unique constraint (rather than the primary key) are also deleted. Reports an error if there are associated foreign-key constraints. Use ALTER TABLE to delete all foreign keys that reference the primary key before you delete the primary key constraint.
- **DROP PRIMARY KEY** – Drop the primary key. All foreign keys referencing the primary key for this table are also deleted. Reports an error if there are associated foreign key constraints. If the primary key is unenforced, DELETE returns an error if associated unenforced foreign key constraints exist.
- **DROP FOREIGN KEY** *role-name* – Drop the foreign key constraint for this table with the given role name. Retains the implicitly created non-unique HG index for the foreign key constraint. Users can explicitly remove the HG index with the DROP INDEX statement.
- **DROP** [**PARTITION**] – Drop the specified partition. The rows in partition P1 are deleted and the partition definition is dropped. You cannot drop the last partition because dropping the last partition would transform a partitioned table to a non-partitioned table. (To merge a partitioned table, use an UNPARTITION clause instead.) For example:

```
CREATE TABLE foo (c1 INT, c2 INT)
    PARTITION BY RANGE (c1)
    (P1 VALUES <= (100) IN dbsp1,
     P2 VALUES <= (200) IN dbsp2,
     P3 VALUES <= (MAX) IN dbsp3
    ) IN dbsp4);
LOAD TABLE ….
ALTER TABLE DROP PARTITION P1;
```

- **RENAME *rename-object* –** renames an object in the table:
  - **RENAME** *new-table-name* – Change the name of the table to the *new-table-name*. Any applications using the old table name must be modified. Also, any foreign keys that were automatically assigned the same name as the old table name do not change names.
  - **RENAME** *column-name* **TO** *new-column-name* – Change the name of the column to *new-column-name*. Any applications using the old column name must be modified.
  - **RENAME** [**PARTITION**] – Rename an existing partition.
  - **RENAME** *constraint-name* **TO** *new-constraint-name* – Change the name of the constraint to *new-constraint-name*. Any applications using the old constraint name must be modified.

- **MOVE clause –** moves a table object. A table object can only reside in one dbspace. Any type of ALTER MOVE blocks any modification to the table for the entire duration of the move.

  Note: You cannot move objects to a cache dbspace.

  - **MOVE TO** – Move all table objects including columns, indexes, unique constraints, primary key, foreign keys, and metadata resided in the same dbspace as the table is mapped to the new dbspace. The ALTER Column MOVE TO clause cannot be requested on a partitioned table.

    A BIT data type column cannot be explicitly placed in a dbspace. The following is not supported for BIT data types:

    ```
    ALTER TABLE t2 alter c1_bit MOVE TO iq_main;
    ```

  - **MOVE TABLE METADATA** – Move the metadata of the table to a new dbspace. For a partitioned table, MOVE TABLE METADATA also moves metadata that is shared among partitions.
  - **MOVE PARTITION** – Move the specified partition to the new dbspace.

- **PARTITION BY –** divides large tables into smaller, more manageable storage objects. Partitions share the same logical attributes of the parent table, but can be placed in separate dbspaces and managed individually. SAP Sybase IQ supports several table partitioning schemes:

  - hash-partitions
  - range-partitions
  - composite-partitions

  A partition-key is the column or columns that contain the table partitioning keys. Partition keys can contain NULL and DEFAULT values, but cannot contain:

  - LOB (BLOB or CLOB) columns
  - BINARY, or VARBINARY columns
  - CHAR or VARCHAR columns whose length is over 255 bytes
  - BIT columns
  - FLOAT/DOUBLE/REAL columns

- **PARTITION BY RANGE –** partitions rows by a range of values in the partitioning column. Range partitioning is restricted to a single partition key column and a maximum of 1024 partitions. In a range-partitioning-scheme, the partition-key is the column that contains the table partitioning keys:

  ```
  range-partition-decl:
    partition-name VALUES <= ( {constant-expr |  MAX } [ ,
  { constant-expr |  MAX }]... )
      [ IN dbspace-name ]
  ```

The partition-name is the name of a new partition on which table rows are stored. Partition names must be unique within the set of partitions on a table. The partition-name is required.

- **VALUE –** specifies the inclusive upper bound for each partition (in ascending order). The user must specify the partitioning criteria for each range partition to guarantee that each row is distributed to only one partition. NULLs are allowed for the partition column and rows with NULL as partition key value belong to the first table partition. However, NULL cannot be the bound value.

  There is no lower bound (MIN value) for the first partition. Rows of NULL cells in the first column of the partition key will go to the first partition. For the last partition, you can either specify an inclusive upper bound or MAX. If the upper bound value for the last partition is not `MAX`, loading or inserting any row with partition key value larger than the upper bound value of the last partition generates an error.
- **Max –** denotes the infinite upper bound and can only be specified for the last partition.
- **IN –** specifies the dbspace in the partition-decl on which rows of the partition should reside.

These restrictions affect partitions keys and bound values for range partitioned tables:

- You can only range partition a non-partitioned table if all existing rows belong to the first partition.
- Partition bounds must be constants, not constant expressions.
- Partition bounds must be in ascending order according to the order in which the partitions were created. That is, the upper bound for the second partition must be higher than for the first partition, and so on.

  In addition, partition bound values must be compatible with the corresponding partition-key column data type. For example, VARCHAR is compatible with CHAR.
- If a bound value has a different data type than that of its corresponding partition key column, SAP Sybase IQ converts the bound value to the data type of the partition key column, with these exceptions:
- Explicit conversions are not allowed. This example attempts an explicit conversion from INT to VARCHAR and generates an error:

```
CREATE TABLE Employees(emp_name VARCHAR(20))
PARTITION BY RANGE(emp_name)
(p1 VALUES <=(CAST (1 AS VARCHAR(20))),
p2 VALUES <= (CAST (10 AS VARCHAR(20)))
```

- Implicit conversions that result in data loss are not allowed. In this example, the partition bounds are not compatible with the partition key type. Rounding assumptions may lead to data loss and an error is generated:

```
CREATE TABLE emp_id (id INT) PARTITION BY RANGE(id) (p1 VALUES
<= (10.5), p2 VALUES <= (100.5))
```

- In this example, the partition bounds and the partition key data type are compatible. The bound values are directly converted to float values. No rounding is required, and conversion is supported:

```
CREATE TABLE id_emp (id FLOAT)
PARTITION BY RANGE(id) (p1 VALUES <= (10),
p2 VALUES <= (100))
```

- Conversions from non-binary data types to binary data types are not allowed. For example, this conversion is not allowed and returns an error:

```
CREATE TABLE newemp (name BINARY)
PARTITION BY RANGE(name)
(p1 VALUES <= ("Maarten"),
p2 VALUES <= ("Zymmerman")
```

- NULL cannot be used as a boundary in a range-partitioned table.
- The row will be in the first partition if the cell value of the 1st column of the partition key evaluated to be NULL. SAP Sybase IQ supports only single column partition keys, so any NULL in the partition key distributes the row to the first partition.

- **PARTITION BY HASH –** maps data to partitions based on partition-key values processed by an internal hashing function. Hash partition keys are restricted to a maximum of eight columns with a combined declared column width of 5300 bytes or less. For hash partitions, the table creator determines only the partition key columns; the number and location of the partitions are determined internally.

  In a hash-partitioning declaration, the partition-key is a column or group of columns, whose composite value determines the partition where each row of data is stored:

```
hash-partitioning-scheme:
  HASH  ( partition-key [ , partition-key, … ] )
```

  - **Restrictions –**
    - You can only hash partition a base table. Attempting to partitioning a global temporary table or a local temporary table raises an error.
    - You can only hash partition a non-partitioned table that is empty.
    - You cannot add, drop, merge, or split a hash partition.
    - You cannot add or drop a column from a hash partition key.
- **PARTITION BY HASH RANGE –** subpartitions a hash-partitioned table by range. In a hash-range-partitioning-scheme declaration, a SUBPARTITION BY RANGE clause adds a new range subpartition to an existing hash-range partitioned table:

```
hash-range-partitioning-scheme:
PARTITION BY HASH  ( partition-key [ , partition-key, … ] )
    [ SUBPARTITION BY RANGE  ( range-partition-decl [ , range-
partition-decl ... ] ) ]
```

  The hash partition specifies how the data is logically distributed and colocated; the range subpartition specifies how the data is physically placed. The new range subpartition is logically partitioned by hash with the same hash partition keys as the existing hash-range partitioned table. The range subpartition key is restricted to one column.

  - **Restrictions –**

- You can only hash partition a base table. Attempting to partitioning a global temporary table or a local temporary table raises an error.
- You can only subpartition a hash-partitioned table by range if the the table is empty.
- You cannot add, drop, merge, or split a hash partition.
- You cannot add or drop a column from a hash partition key.

Note: Range-partitions and composite partitioning schemes, like hash-range partitions, require the separately licensed VLDB Management option.

- **MERGE PARTITION –** merge *partition-name-1* into *partition-name-2*. Two partitions can be merged if they are adjacent partitions and the data resides on the same dbspace. You can only merge a partition with a lower partition value into the adjacent partition with a higher partition value. Note that the server does not check CREATE privilege on the dbspace into which the partition is merged. For an example of how to create adjacent partitions, see CREATE TABLE Statement examples.
- **RENAME PARTITION –** rename an existing `PARTITION`.
- **UNPARTITION –** remove partitions from a partitioned table. Each column is placed in a single dbspace. Note that the server does not check `CREATE` privilege on the dbspace to which data of all partitions is moved. ALTER TABLE UNPARTITION blocks all database activities.
- **ALTER OWNER –** change the owner of a table. The **ALTER OWNER** clause may not be used in conjunction with any other [alter-clause] clauses of the ALTER TABLE statement.
  - [**PRESERVE** | **DROP] PERMISSIONS** – If you do not want the new owner to have the same privileges as the old owner, use the DROP privileges clause (default) to drop all explicitly-granted privileges that allow a user access to the table. Implicitly-granted privileges given to the owner of the table are given to the new owner and dropped from the old owner.
  - [**PRESERVE** | **DROP] FOREIGN KEYS** – If you want to prevent the new owner from accessing data in referenced tables, use the DROP FOREIGN KEYS clause (default) to drop all foreign keys within the table, as well as all foreign keys referring to the table. Use of the PRESERVE FOREIGN KEYS clause with the DROP PERMISSIONS clause fails unless all referencing tables are owned by the new owner.

  The **ALTER TABLE ALTER OWNER** statement fails if:

  - Another table with the same name as the original table exists and is owned by the new user.
  - The PRESERVE FOREIGN KEYS and PRESERVE PERMISSIONS clauses are both specified and there is a foreign key owned by a user other than the new table owner referencing the table that relies on implicitly-granted privileges (such as those given to the owner of a table). To avoid this failure, explicitly grant SELECT privileges to the referring table's original owner, or drop the foreign keys.

- The PRESERVE FOREIGN KEYS clause is specified, but the PRESERVE PERMISSIONS clause is NOT, and there is a foreign key owned by a user other than the new table owner referencing the table. To avoid this failure, drop the foreign keys.
- The PRESERVE FOREIGN KEYS clause is specified and the table contains a foreign key that relies on implicitly-granted privileges (such as those given to the owner of a table). To avoid this failure, explicitly GRANT SELECT privileges to the new owner on the referenced table, or drop the foreign keys.
- The table contains a column with a default value that refers to a sequence, and the USAGE privilege of the sequence generator relies on implicitly-granted privileges (such as those given to the owner of a sequence). To avoid this failure, explicitly grant USAGE privilege on the sequence generator to the new owner of the table.
- Enabled materialized views that depend on the original table exist.

Examples

(back to top) on page 286

- **Example 1 –** adds a new column to the `Employees` table showing which office they work in:

```
ALTER TABLE Employees
ADD office CHAR(20)
```

- **Example 2 –** drops the `office` column from the `Employees` table:

```
ALTER TABLE Employees
DROP office
```

- **Example 3 –** Adds a column to the `Customers` table assigning each customer a sales contact:

```
ALTER TABLE Customers
ADD SalesContact INTEGER
REFERENCES Employees (EmployeeID)
```

- **Example 4 –** adds a new column `CustomerNum` to the `Customers` table and assigns a default value of 88:

```
ALTER TABLE Customers
ADD CustomerNum INTEGER DEFAULT 88
```

- **Example 5 –** moves **FP** indexes for `c2`, `c4`, and `c5`, from dbspace `Dsp3` to `Dsp6`. **FP** index for `c1` remains in `Dsp1`. **FP** index for `c3` remains in `Dsp2`. The primary key for `c5` remains in `Dsp4`. **DATE** index `c4_date` remains in `Dsp5`.

```
CREATE TABLE foo (
        c1 INT IN Dsp1,
        c2 VARCHAR(20),
        c3 CLOB IN Dsp2,
        c4 DATE,
        c5 BIGINT,
        PRIMARY KEY (c5) IN Dsp4) IN Dsp3);
```

```
    CREATE DATE INDEX c4_date ON foo(c4) IN Dsp5;
    ALTER TABLE foo
        MOVE TO Dsp6;
```

- **Example 6 –** moves only **FP** index `c1` from dbspace `Dsp1` to `Dsp7`:

```
ALTER TABLE foo ALTER c1 MOVE TO Dsp7
```

- **Example 7 –** uses many **ALTER TABLE** clauses to move, split, rename, and merge partitions.

  Create a partitioned table:

```
CREATE TABLE bar (
       c1 INT,
       c2 DATE,
       c3 VARCHAR(10))
   PARTITION BY RANGE(c2)
      (p1 VALUES <= ('2005-12-31') IN dbsp1,
      p2 VALUES <= ('2006-12-31') IN dbsp2,
      P3 VALUES <= ('2007-12-31') IN dbsp3,
      P4 VALUES <= ('2008-12-31') IN dbsp4);
INSERT INTO bar VALUES(3, '2007-01-01', 'banana nut');
INSERT INTO BAR VALUES(4, '2007-09-09', 'grape jam');
INSERT INTO BAR VALUES(5, '2008-05-05', 'apple cake');
```

  Move partition `p2` to `dbsp5`:

```
ALTER TABLE bar MOVE PARTITION p2 TO DBSP5;
```

  Split partition `p4` into 2 partitions:

```
ALTER TABLE bar SPLIT PARTITION p4 INTO
   (P41 VALUES <= ('2008-06-30') IN dbsp4,
    P42 VALUES <= ('2008-12-31') IN dbsp4);
```

  This **SPLIT PARTITION** reports an error, as it requires data movement. Not all existing rows are in the same partition after split.

```
ALTER TABLE bar SPLIT PARTITION p3 INTO
   (P31 VALUES <= ('2007-06-30') IN dbsp3,
   P32 VALUES <= ('2007-12-31') IN dbsp3);
```

  This error is reported:

```
No data move is allowed, cannot split partition p3.
```

  This **SPLIT PARTITION** reports an error, because it changes the partition boundary value:

```
ALTER TABLE bar SPLIT PARTITION p2 INTO
   (p21 VALUES <= ('2006-06-30') IN dbsp2,
    P22 VALUES <= ('2006-12-01') IN dbsp2);
```

  This error is reported:

```
Boundary value for the partition p2 cannot be changed.
```

  Merge partition `p3` into `p2`. An error is reported as a merge from a higher boundary value partition into a lower boundary value partition is not allowed.

```
ALTER TABLE bar MERGE PARTITION p3 into p2;
```

This error is reported:

```
Partition 'p2' is not adjacent to or before partition 'p3'.
```

Merge partition `p2` into `p3`:

```
ALTER TABLE bar MERGE PARTITION p2 INTO P3;
```

Rename partition `p1` to `p1_new`:

```
ALTER TABLE bar RENAME PARTITION p1 TO p1_new;
```

Unpartition table `bar`:

```
ALTER TABLE bar UNPARTITION;
```

Partition table `bar`. This command reports an error, because all rows must be in the first partition.

```
ALTER TABLE bar PARTITION BY RANGE(c2)
   (p1 VALUES <= ('2005-12-31') IN dbsp1,
    P2 VALUES <= ('2006-12-31') IN DBSP2,
    P3 VALUES <= ('2007-12-31') IN dbsp3,
    P4 VALUES <= ('2008-12-31') IN dbsp4);
```

This error is reported:

```
All rows must be in the first partition.
```

Partition table `bar`:

```
ALTER TABLE bar PARTITION BY RANGE(c2)
   (p1 VALUES <= ('2008-12-31') IN dbsp1,
   P2 VALUES <= ('2009-12-31') IN dbsp2,
   P3 VALUES <= ('2010-12-31') IN dbsp3,
   P4 VALUES <= ('2011-12-31') IN dbsp4);
```

- **Example 8 –** changes a table `tab1` so that it is no longer registered for in-memory real-time updates in the RLV store.

```
ALTER TABLE tab1 DISABLE RLV STORE
```

Usage

(back to top) on page 286

The ALTER TABLE statement changes table attributes (column definitions and constraints) in a table that was previously created. The syntax allows a list of alter clauses; however, only one table constraint or column constraint can be added, modified, or deleted in each ALTER TABLE statement. ALTER TABLE is prevented whenever the statement affects a table that is currently being used by another connection. ALTER TABLE can be time consuming, and the server does not process requests referencing the same table while the statement is being processed.

Note: You cannot alter local temporary tables, but you can alter global temporary tables when they are in use by only one connection.

SAP Sybase IQ enforces REFERENCES and CHECK constraints. Table and/or column check constraints added in an ALTER TABLE statement are evaluated, only if they are defined on one of the new columns added, as part of that alter table operation. For details about CHECK constraints, see *CREATE TABLE Statement*.

If **SELECT** * is used in a view definition and you alter a table referenced by the **SELECT** *, then you must run **ALTER VIEW** *<viewname>* **RECOMPILE** to ensure that the view definition is correct and to prevent unexpected results when querying the view.

Side effects:

- Automatic commit. The ALTER and DROP options close all cursors for the current connection. The Interactive SQL data window is also cleared.
- A checkpoint is carried out at the beginning of the ALTER TABLE operation.
- Once you alter a column or table, any stored procedures, views or other items that refer to the altered column no longer work.

Standards

(back to top) on page 286

- SQL–Vendor extension to ISO/ANSI SQL grammar.
- SAP Sybase Database product–Some clauses are supported by SAP Adaptive Server® Enterprise.

Permissions

(back to top) on page 286

Syntax 1

Requires one of:

- ALTER ANY TABLE system privilege
- ALTER ANY OBJECT system privilege
- ALTER privilege on the table
- You own the table

Syntax 2

The system privileges required for syntax 1 varies depending upon the clause used.

Clause	Privilege Required
Add	Requires one of: • ALTER ANY TABLE system privilege • ALTER ANY OBJECT system privilege • ALTER privilege on the underlying table • You own the underlying table UNIQUE, PRIMARY KEY, FOREIGN KEY, or IQ UNIQUE column constraint – Requires above along with REFERENCES privilege on the underlying table. FOREIGN KEY table constraint requires above along with one of: • CREATE ANY INDEX system privilege • CREATE ANY OBJECT system privilege • REFERENCES privilege on the base table PARTITION BY RANGE requires above along with one of: • CREATE ANY OBJECT system privilege • CREATE privilege on the dbspaces where the partitions are being created
Alter	Requires one of: • ALTER ANY TABLE system privilege • ALTER ANY OBJECT system privilege • ALTER privilege on the table • You own the table. To alter a primary key or unique constraint, also requires REFERENCES privilege on the table.
Drop	Drop a column with no constraints – Requires one of: • ALTER ANY OBJECT system privilege • ALTER ANY TABLE system privilege • ALTER privilege on the underlying table • You own the underlying table Drop a column or table with a constraint requires above along with REFERENCES privilege if using ALTER privilege. Drop a partition on table owned by self – None required. Drop a partition on table owned by other users – Requires one of: • ALTER ANY TABLE system privilege • ALTER ANY OBJECT system privilege • ALTER privilege on the table

Clause	Privilege Required
RENAME	Requires one of: • ALTER ANY TABLE system privilege • ALTER ANY OBJECT system privilege • ALTER privilege on the table • You own the table
Move	Requires one of: • ALTER ANY TABLE system privilege • ALTER ANY OBJECT system privilege • system privilege • ALTER privilege on the underlying table • You own the underlying table Also requires one of the following: • CREATE ANY OBJECT system privilege • CREATE privilege on the dbspace to which the partition is being moved
Split Partition	Partition on table owned by self – None required. Partition on table owned by other users – Requires one of: • SELECT ANY TABLE system privilege • SELECT privilege on table Also requires one of: • ALTER ANY TABLE system privilege • ALTER ANY OBJECT system privilege • ALTER privilege on the table
Merge Partition, Unpartition	Table owned by self – None required. Table owned by other users – Requires one of: • ALTER ANY TABLE system privilege • ALTER ANY OBJECT system privilege • ALTER privilege on the table

Clause	Privilege Required
Partition By	Requires one of: • CREATE ANY OBJECT system privilege • CREATE privilege on the dbspaces where the partitions are being created Also requires one of: • ALTER ANY TABLE system privilege • ALTER ANY OBJECT system privilege • ALTER privilege on the table • You own the table
or disable RLV store	Requires one of: • ALTER ANY TABLE system privilege • ALTER ANY OBJECT system privilege

Index

T

V