SQL Anywhere® Server Spatial Data Support

Version 12.0.1

January 2012

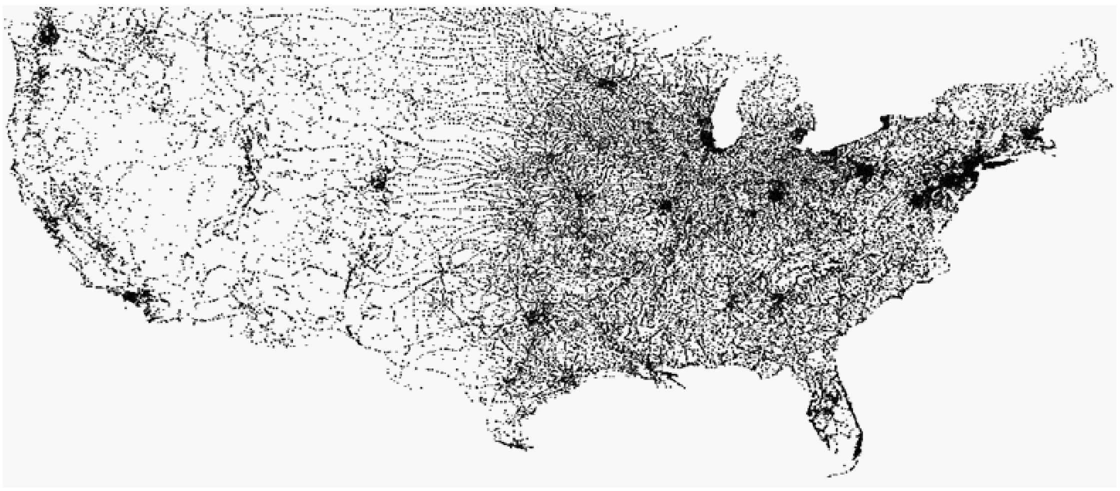
Version 12.0.1
January 2012

Contents

About this book

This book describes the SQL Anywhere spatial data support and how the spatial features can be used to generate and analyze spatial data.

The following image represents the distributions of cities and towns across the United States and is one example of the interesting operations you can perform on spatial data.



Using spatial data

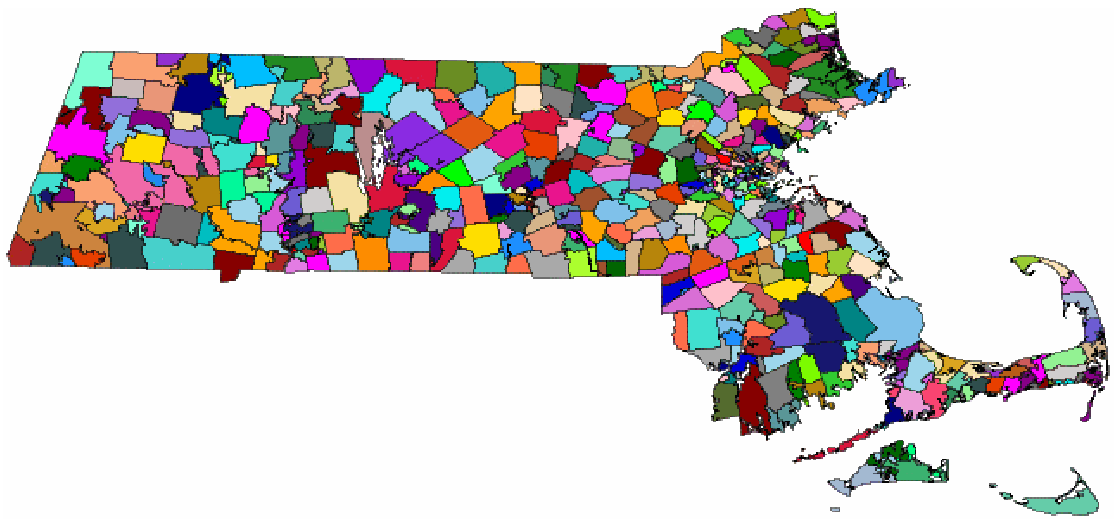
This section introduces SQL Anywhere spatial support and explains its purpose, describes the supported data types, and explains how to generate and analyze spatial data.

The spatial data documentation assumes you already have some familiarity with spatial reference systems and with the spatial data you intend to work with. If you do not, links to additional reading material can be found here: "Recommended reading on spatial topics" on page 16.

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).

Introduction to spatial data

Spatial data is data that describes the position, shape, and orientation of objects in a defined space. Spatial data in SQL Anywhere 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.

Example of how spatial data might be used

Spatial data support in SQL Anywhere 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
```

SQL Anywhere 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. SQL Anywhere also provides a set of SQL spatial functions designed for compatibility with other products.

See also

- "Supported spatial data types and their hierarchy" on page 7
- "Spatial compatibility functions" on page 324
- "Tutorial: Experimenting with the spatial features" on page 55

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:

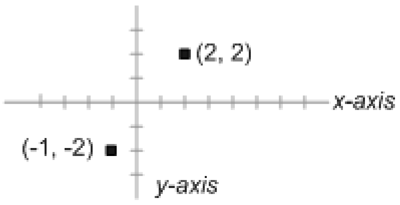
- Units of measure of the underlying coordinate system (degrees, meters, and so on)
- Maximum and minimum coordinates (also referred to as the bounds)
- Default linear unit of measure
- Whether the data is planar or spheroid data
- Projection information for transforming the data to other SRSs

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

By default, SQL Anywhere 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.

  See “Limitations of flat-Earth spatial reference systems” on page 42, and “Supported spatial predicates” on page 10.

- **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

SQL Anywhere 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. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

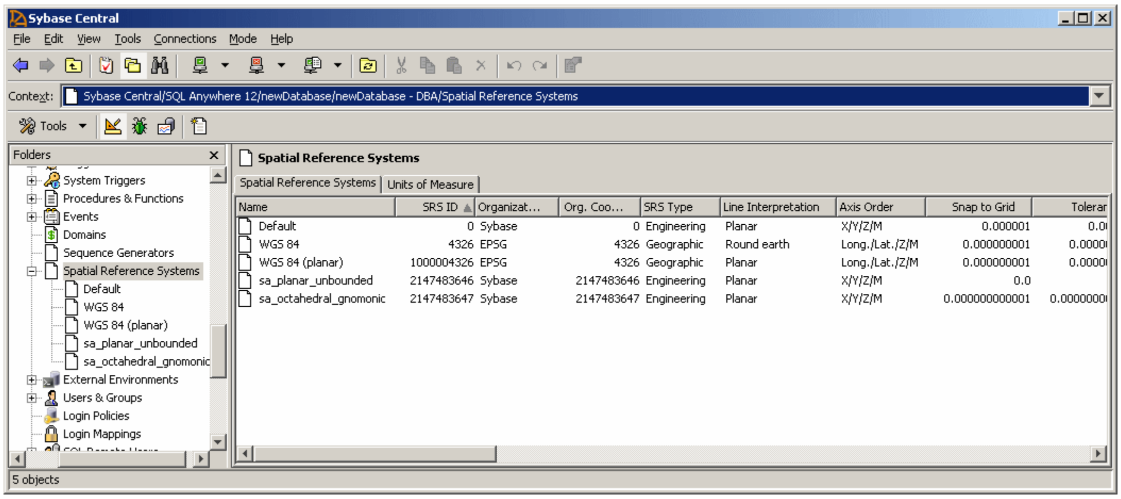
You can find descriptions of these additional spatial reference systems at spatialreference.org and www.epsg-registry.org/.

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. See "ST_SPATIAL_REFERENCE_SYSTEMS consolidated view" [*SQL Anywhere Server - SQL Reference*].

You can also look in the **Spatial Reference Systems** folder in Sybase Central to see the list of spatial reference systems installed in the database:



Name	SRS ID	Organizat...	Org. Coo...	SRS Type	Line Interpretation	Axis Order	Snap to Grid	Tolerar
Default	0	Sybase	0	Engineering	Planar	X/Y/Z/M	0.000001	0.0
WGS 84	4326	EPSG	4326	Geographic	Round earth	Long./Lat./Z/M	0.000000001	0.0000
WGS 84 (planar)	1000004326	EPSG	4326	Geographic	Planar	Long./Lat./Z/M	0.000000001	0.0000
sa_planar_unbounded	2147483646	Sybase	2147483646	Engineering	Planar	X/Y/Z/M	0.0	
sa_octahedral_gnomonic	2147483647	Sybase	2147483647	Engineering	Planar	X/Y/Z/M	0.000000000001	0.0000000

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 SQL Anywhere (including SRID 3857).
2. Perform `dbunload -n` to get the 3857 SRID definition.
3. Use Sybase Central to create a spatial reference system with SRID 900913 using the information from the unloaded SRID definition.

See also

- "Creating a spatial reference system (Sybase Central)" on page 37
- "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*]
- "ST_SPATIAL_REFERENCE_SYSTEMS consolidated view" [*SQL Anywhere Server - SQL Reference*]
- "CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*]
- "How flat-Earth and round-Earth representations work" on page 41

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 SQL Anywhere 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.

See also

- “Installing additional predefined units of measure” on page 6
- “sa_install_feature system procedure” [*SQL Anywhere Server - SQL Reference*]
- “CREATE SPATIAL REFERENCE SYSTEM statement” [*SQL Anywhere Server - SQL Reference*]
- “LINEAR UNIT OF MEASURE clause, CREATE SPATIAL REFERENCE SYSTEM statement” [*SQL Anywhere Server - SQL Reference*]
- “ANGULAR UNIT OF MEASURE clause, CREATE SPATIAL REFERENCE SYSTEM statement” [*SQL Anywhere Server - SQL Reference*]
- “ST_UNITS_OF_MEASURE consolidated view” [*SQL Anywhere Server - SQL Reference*]

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

DBA or be a member of the SYS_SPATIAL_ADMIN_ROLE group

Context and remarks

None.

Install additional predefined units of measure

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

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

Results

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**.

See also

- "Units of measure" on page 5
- "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*]
- "CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*]
- "LINEAR UNIT OF MEASURE clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*]
- "ANGULAR UNIT OF MEASURE clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*]
- "ST_UNITS_OF_MEASURE consolidated view" [*SQL Anywhere Server - SQL Reference*]

SQL Anywhere support for spatial data

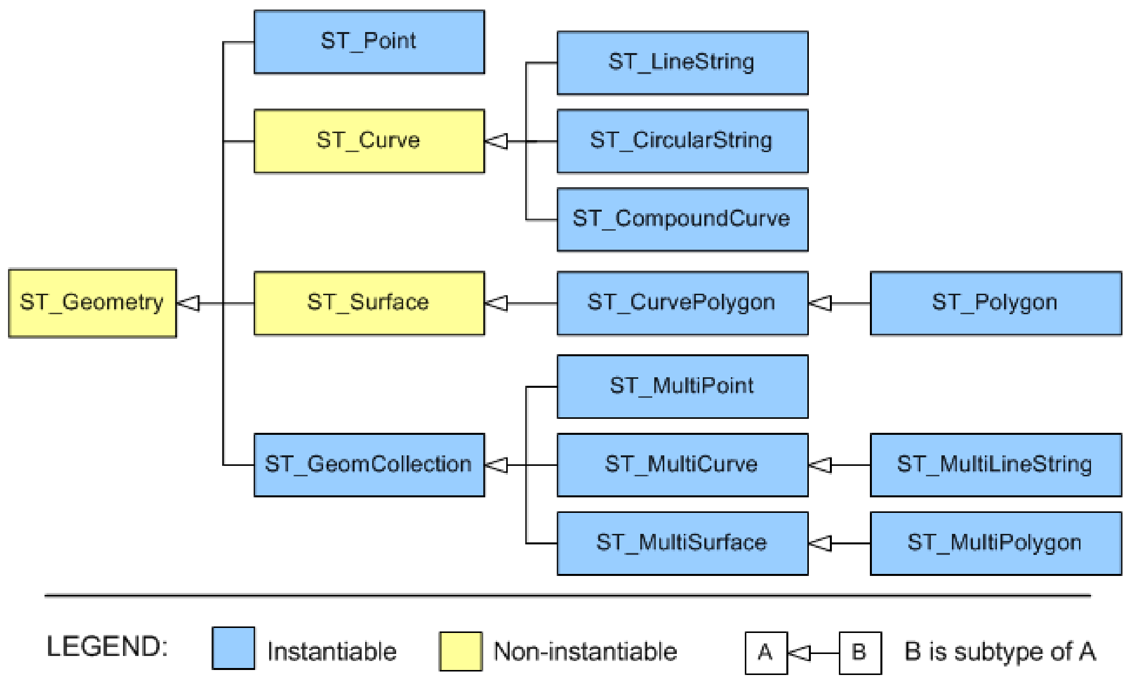
The following sections describe the SQL Anywhere support for spatial data.

Supported spatial data types and their hierarchy

SQL Anywhere 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

SQL Anywhere 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.

  See "ST_Point type" on page 284.

- **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.

  See "ST_LineString type" on page 238.

- **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.

See "How polygon ring orientation works" on page 48 and "ST_Polygon type" on page 301.

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

  See "ST_CircularString type" on page 67.

- **Compound curves** A compound curve is a connected sequence of circularstrings or linestrings.

  See "ST_CompoundCurve type" on page 75.

- **Curve polygons** A curve polygon is a more general polygon that may have circular arc boundary segments.

  See "ST_CurvePolygon type" on page 88.

- **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.

  See "ST_Geometry type" on page 107.

- **Geometry collections** A geometry collection is a collection of one or more geometries (such as points, lines, polygons, and so on).

  See "ST_GeomCollection type" on page 99.

- **Multipoints** A multipoint is a collection of individual points.

  In GIS data, multipoints are typically used to represent a set of locations.

  See "ST_MultiPoint type" on page 260.

- **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.

  See "ST_MultiPolygon type" on page 266.

- **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.

  See "ST_MultiLineString type" on page 254.

- **Multisurfaces** A multisurface is a collection of curve polygons.

See “ST_MultiSurface type” on page 273.

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()`.

See also

- “TREAT function [Data type conversion]” [*SQL Anywhere Server - SQL Reference*]
- “CAST function [Data type conversion]” [*SQL Anywhere Server - SQL Reference*]

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. In order 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.

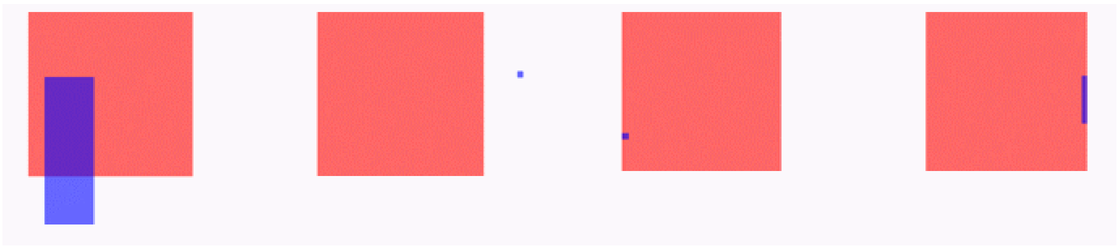
SQL Anywhere supports the following spatial predicates to help answer questions about the spatial relationship between geometries:

- "ST_Contains method for type ST_Geometry"
- "ST_Covers method for type ST_Geometry"
- "ST_CoveredBy method for type ST_Geometry"
- "ST_Crosses method for type ST_Geometry"
- "ST_Disjoint method for type ST_Geometry"
- "ST_IsEmpty method for type ST_Geometry"
- "ST_Equals method for type ST_Geometry"
- "ST_Intersects method for type ST_Geometry"
- "ST_OrderingEquals method for type ST_Geometry"
- "ST_Overlaps method for type ST_Geometry"
- "ST_Relate method for type ST_Geometry"
- "ST_Touches method for type ST_Geometry"
- "ST_IsValid method for type ST_Geometry"
- "ST_Within method for type ST_Geometry"

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.

See also

- "How spatial relationships work" on page 51

Compliance with spatial standards

SQL Anywhere spatial complies with the following standards:

- **International Organization for Standardization (ISO)** SQL Anywhere geometries conform to the ISO standards for defining spatial user-types, routines, schemas, and for processing spatial data. SQL Anywhere 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** SQL Anywhere 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.

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

  To ensure compatibility with SQL Anywhere spatial geometries, it is recommended that you adhere to the standards specified by the OGC.
- **SQL Multimedia (SQL/MM)** SQL Anywhere 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 SQL Anywhere 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 SQL Anywhere, 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 SQL Anywhere 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)'`
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: `'0101000000000000000000F03F000000000000F03F'`
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: `'0101000020040000000000000000000F03F000000000000F03F'`
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>`
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). For more information about ESRI shapefile support, see "Support for ESRI shapefiles" on page 15.

Data format	Import	Export	Description
Geo-JSON	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. SQL Anywhere supports the GeoJSON standard and provides the ST_AsGeoJSON method for converting SQL output to the GeoJSON format. See "ST_AsGeoJSON method for type ST_Geometry" on page 116. 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.
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" stroke-width="1" x="1" y="-1"/>`

Support for ESRI shapefiles

SQL Anywhere 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. SQL Anywhere 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 a working example of loading a shapefile into a SQL Anywhere database, see "Tutorial: Experimenting with the spatial features" on page 55.

For more information about querying within a shapefile, see "Openstring expressions in a FROM clause" [*SQL Anywhere Server - SQL Reference*].

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

See also

- "Import data with the Import Wizard (Interactive SQL)" [*SQL Anywhere Server - SQL Usage*]
- "st_geometry_load_shapefile system procedure" [*SQL Anywhere Server - SQL Reference*]
- "LOAD TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "sa_describe_shapefile system procedure" [*SQL Anywhere Server - SQL Reference*]
- "INPUT statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]

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

Getting started with spatial data

This section provides procedures for creating, accessing, and manipulating spatial data.

Creating a spatial column (Sybase Central)

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

Prerequisites

Must be a user with permissions to alter the table.

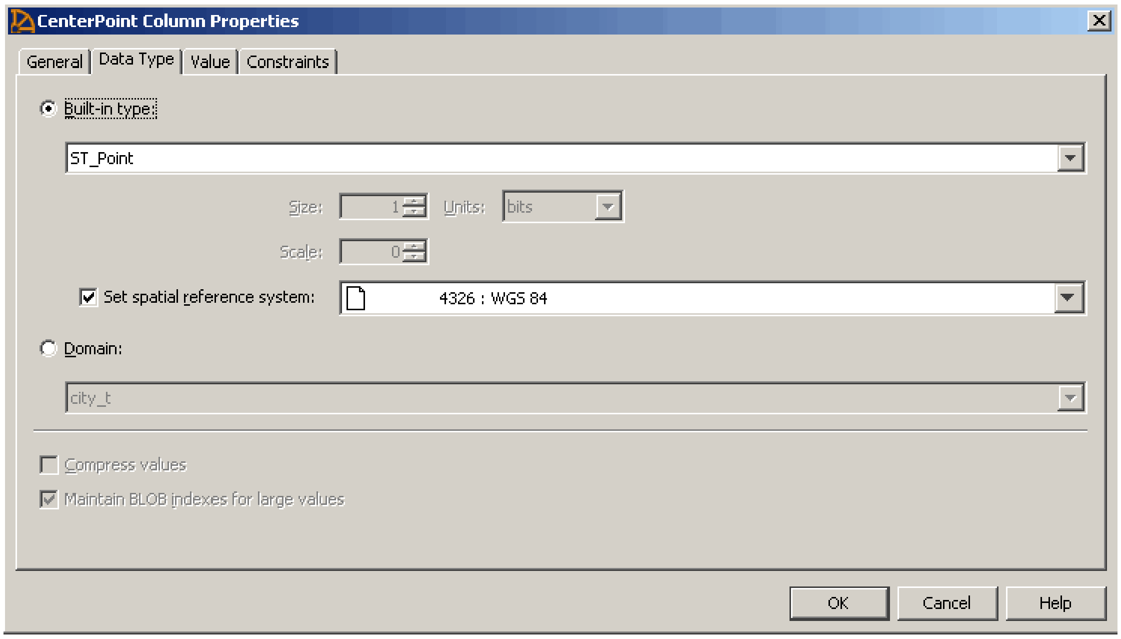
The table has a primary key defined. Update and delete operations are not supported for tables that contain a spatial column unless a primary key is defined.

Context and remarks

Many.

Create a spatial column (Sybase Central)

1. Connect to the database as a user with permissions to alter the table.
2. Create a new column:
   - In the left pane, expand the **Tables** list.
   - Right-click a table, then click **New** » **Column**.
3. Set the spatial data type:
   a. Right-click the column name, then click **Properties**.
   b. Click the **Data Type** tab.
   c. Select **Built-in type** and choose a spatial data type from the dropdown list.
   d. Select **Set spatial reference system** and choose a spatial reference system from the dropdown list.



(●) Built-in type:

ST_Point

Size: 1 Units: bits

Scale: 0

[✓] Set spatial reference system: 4326 : WGS 84

( ) Domain:

city_t

[ ] Compress values

[✓] Maintain BLOB indexes for large values

e. Click **OK**.

4. Click **File** » **Save**.

Results

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 by using the ALTER TABLE statement. You can also add spatial data to the column.

See also

- "Supported spatial data types and their hierarchy" on page 7
- "Spatial reference systems (SRS) and Spatial reference identifiers (SRID)" on page 2
- "ALTER TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "Adding SRID column constraints" on page 18

Adding SRID column constraints

SRID constraints allow you to place restrictions on the values that can be stored in a spatial column. To include a spatial column in an index, the column must have a SRID constraint, which can be added using the CREATE TABLE and ALTER TABLE statements.

Prerequisites

Must be a user with permissions to create or alter a table.

If you add a spatial column to a table, you should make sure that the table has a primary key defined. Update and delete operations are not supported for a table that contains a spatial column unless a primary key is defined.

Context and remarks

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

Add SRID column constraints

- Execute a CREATE TABLE or ALTER TABLE statement that includes the SRID constraint for the spatial column.

  ```
  CREATE TABLE Test (
     ID INTEGER PRIMARY KEY,
     Geometry_1 ST_Geometry,
     Geometry_2 ST_Geometry(SRID=4326),
     );
  ```

Results

The SRID constraint is added to the spatial column in the table.

Next

You can include the spatial column in an index.

Example

For example, execute the following statement to create a table named Test with a SRID constraint (SRID=4326) on the Geometry_2 column:

```
CREATE TABLE Test (
   ID INTEGER PRIMARY KEY,
   Geometry_1 ST_Geometry,
   Geometry_2 ST_Geometry(SRID=4326),
   );
```

This constraint means that only values associated with SRID 4326 can be stored in this column.

The column Geometry_1 is unconstrained and can store values associated with any SRID.

You cannot create an index on the Geometry_1 column. However, you can create an index on the Geometry_2 column.

If you have a table with an existing spatial column, you can use the ALTER TABLE statement to add a SRID constraint to a spatial column. For example, execute a statement similar to the following to add a constraint to the Geometry_1 column in the table named Test:

```
ALTER TABLE Test
   MODIFY Geometry_1 ST_Geometry(SRID=4326);
```

See also

- "CREATE INDEX statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "ALTER TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "Creating a spatial column (Sybase Central)" on page 17
- "Creating a spatial column (Interactive SQL)" on page 20

Creating a spatial column (Interactive SQL)

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

Prerequisites

Must be a user with permissions to alter the table.

Context and remarks

Many.

Create a spatial column (Interactive SQL)

1. Connect to the database as a user with permissions to alter the table.

2. Execute an ALTER TABLE statement.

Results

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.

Example

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

```
ALTER TABLE Customers
ADD Location ST_Point(SRID=1000004326);
```

See also

- “Supported spatial data types and their hierarchy” on page 7
- “Adding SRID column constraints” on page 18
- “Spatial reference systems (SRS) and Spatial reference identifiers (SRID)” on page 2
- “ALTER TABLE statement” [*SQL Anywhere Server - SQL Reference*]

Create an index 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, in order 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.

See also

- “Indexes” [*SQL Anywhere Server - SQL Usage*]
- “Creating an index” [*SQL Anywhere Server - SQL Usage*]
- “CREATE INDEX statement” [*SQL Anywhere Server - SQL Reference*]
- “ALTER INDEX statement” [*SQL Anywhere Server - SQL Reference*]
- “Declaring clustered indexes” [*SQL Anywhere Server - SQL Usage*]
- “Adding SRID column constraints” on page 18

Understand 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 SQL Anywhere does not support user-defined types, the SQL Anywhere 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)
```

SQL Anywhere 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

See:

- "Accessing and manipulating spatial data"
- "ST_Point type"

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;
```

Note that 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.

See:

- "ST_X() method for type ST_Point"

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

See:

- "ST_Point type"
- "ST_GeomFromText method for type ST_Geometry"

Using static aggregate methods (SQL Anywhere extension)

As an extension to ANSI/SQL, SQL Anywhere 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

See:

- "ST_AsSVGAggr method for type ST_Geometry"

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 );
```

See:

- "Search conditions" [*SQL Anywhere Server - SQL Reference*]
- "ST_Point type"
- "ST_GeomFromText method for type ST_Geometry"

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 );
```

See:

- "TREAT function [Data type conversion]" [*SQL Anywhere Server - SQL Reference*]
- "ST_Point type"
- "ST_GeomFromText method for type ST_Geometry"

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. SQL Anywhere performs automatic conversion from these formats to geometry types. For an example of loading from WKT, see "Loading spatial data from a Well Known Text (WKT) file" on page 30.

- **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. See "Support for ESRI shapefiles" on page 15 and "Lesson 3: Load the ESRI shapefile data" on page 57.

- **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 )
  ```

  See "Openstring expressions in a FROM clause" [*SQL Anywhere Server - SQL Reference*].

- **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. See "Instantiating instances of a UDT" on page 22 and "Using static methods" on page 23.

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

Permissions to select from the table containing the spatial data.

Context and remarks

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.

This means that 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 **SQL Anywhere** in the left pane. On the **Results** tab, change **Truncation Length** to a high value, such as 5000. Click **OK** to save your changes, and execute the statement again.

View spatial data as images (Interactive SQL)

1. Connect to your database in Interactive SQL.
2. Execute a query to select spatial data from a table. For example:

   ```
   SELECT * FROM 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).

Results

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.

Example

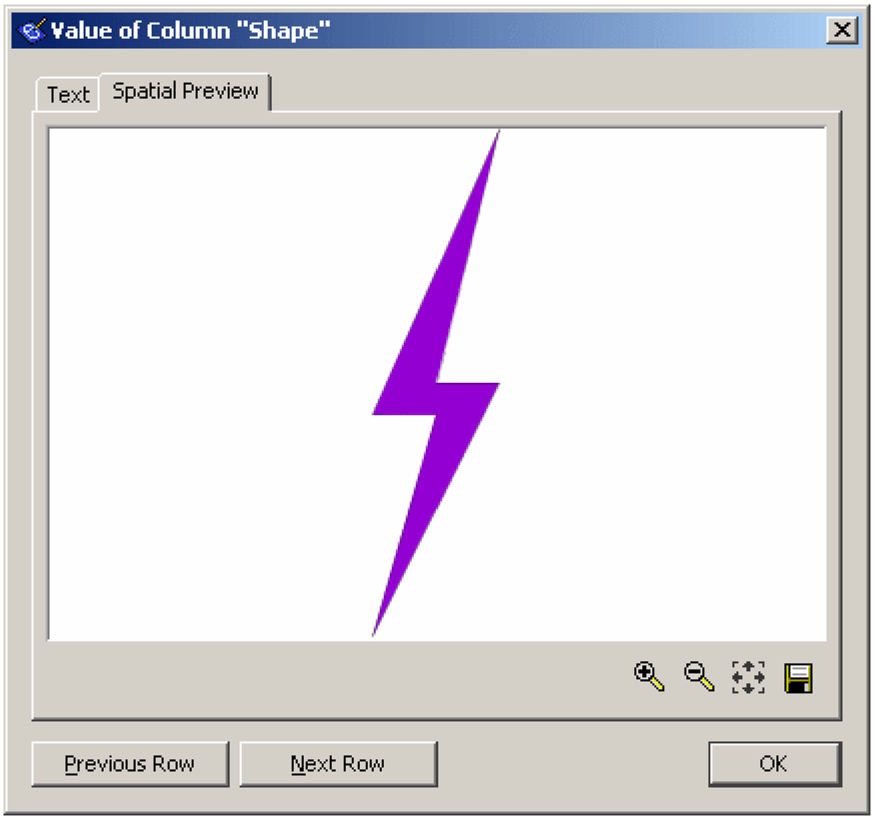
1. Connect to the sample database and execute the following query:

   ```
   SELECT * FROM SpatialShapes;
   ```

2. Double-click the 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.

3. Click the **Spatial Preview** tab to see the geometry as a Scalable Vector Graphic (SVG).



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

Permissions to select from the table containing the spatial data.

Context and remarks

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.

View spatial data as images (Spatial Viewer)

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 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.

Results

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

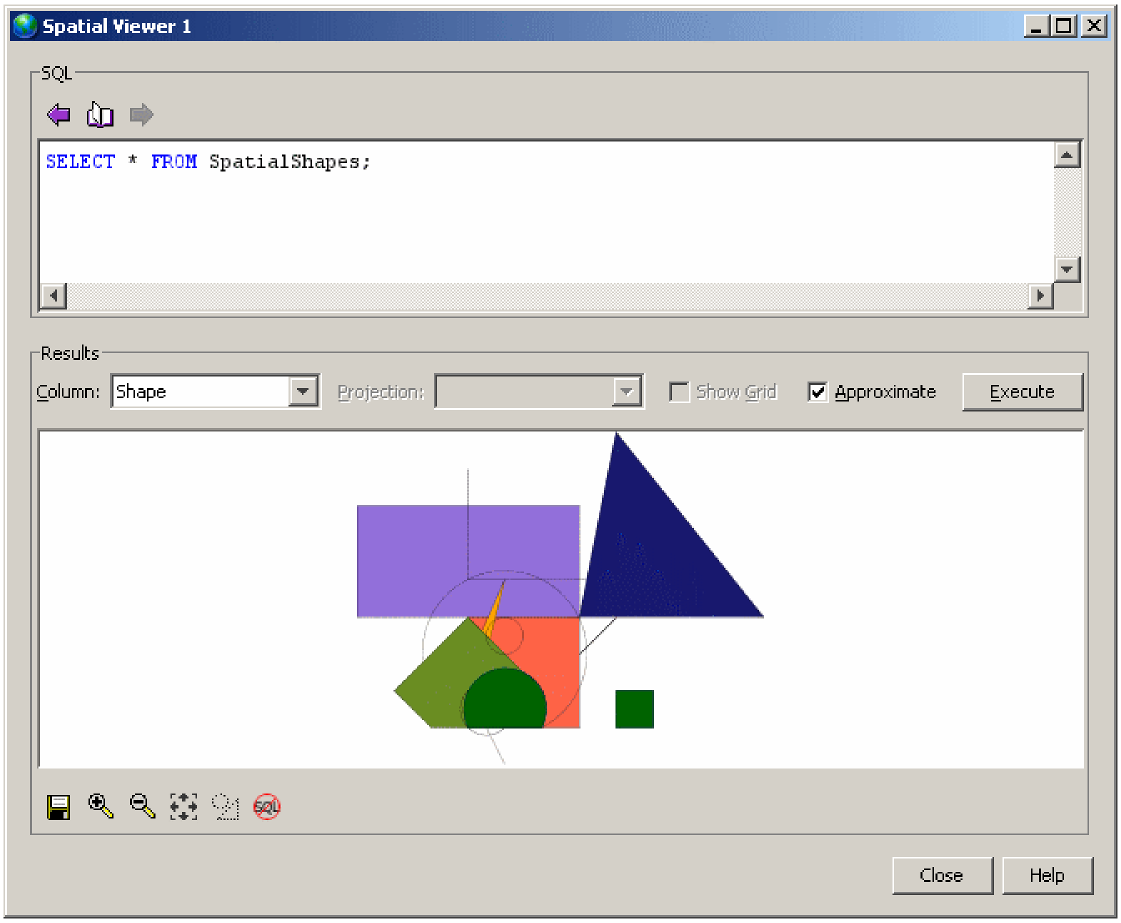
Next

None.

Example

1. Connect to the sample database from Interactive SQL.
2. Click **Tools** » **Spatial Viewer**.
3. In the **Spatial Viewer**, execute the following query in the **SQL** pane:

   ```
   SELECT * FROM SpatialShapes;
   ```



```
SELECT * FROM SpatialShapes;
```

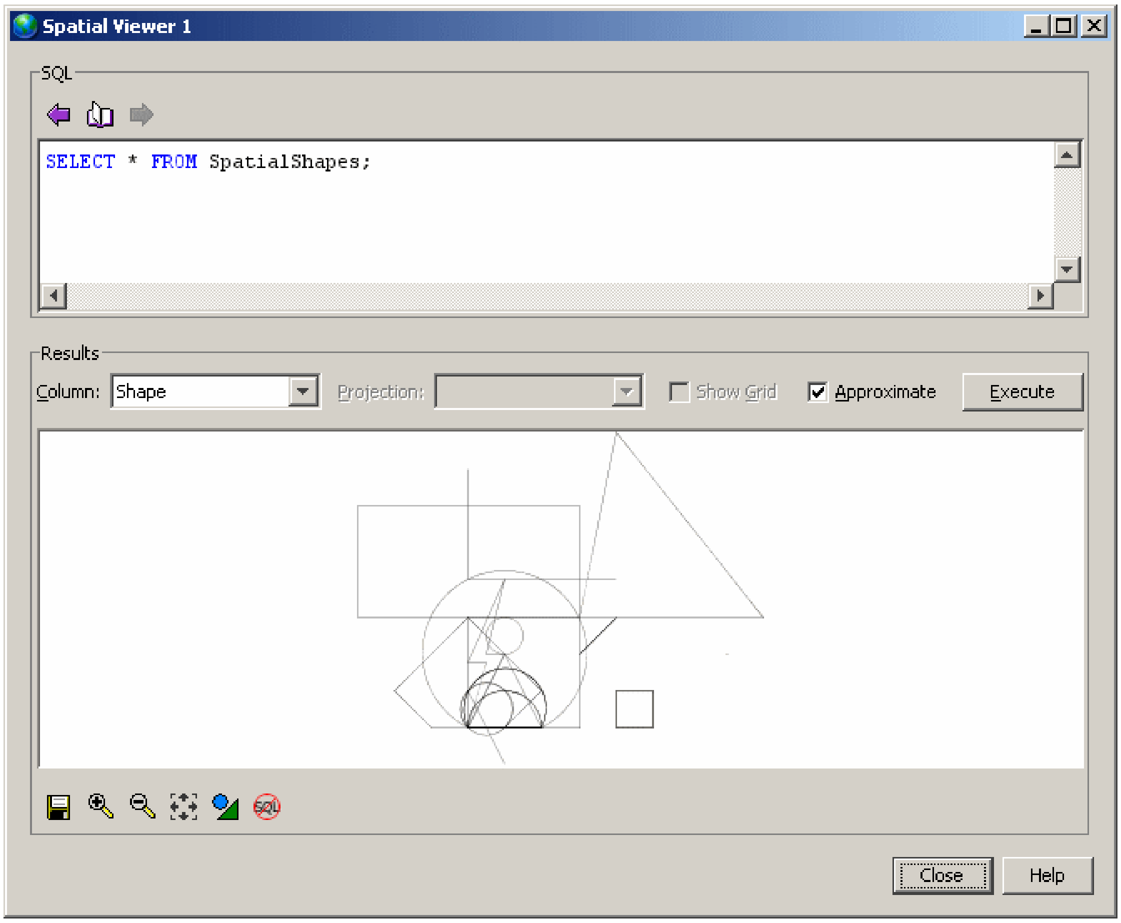
Column: Shape

Projection:

Show Grid [ ]

Approximate [✓]

4. Here is an example of how the image appears as outlines using the **Draw Outlined Polygons** tool:



```
SELECT * FROM SpatialShapes;
```

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

DBA authority or a member of the SYS_SPATIAL_ADMIN_ROLE group.

Context and remarks

Many.

Load spatial data from a WKT file

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 as a user with DBA authority or as a member of the SYS_SPATIAL_ADMIN_ROLE group.

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\
   \SQL Anywhere 12\\Samples\\wktgeometries.csv' DELIMITED BY ',';
   ```

   The data is loaded into the table.

Results

The spatial data is successfully loaded from the WKT file.

Next

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

Example

1. Save the following text in a text file named *wktgeometries.csv*.

   The following text contains a group of geometries, defined in WKT.

   ```
   head,"CircularString(1.1 1.9, 1.1 2.5, 1.1 1.9)"
   left iris,"Point(0.96 2.32)"
   right iris,"Point(1.24 2.32)"
   left eye,"MultiCurve(CircularString(0.9 2.32, 0.95 2.3, 1.0
   2.32),CircularString(0.9 2.32, 0.95 2.34, 1.0 2.32))"
   right eye,"MultiCurve(CircularString(1.2 2.32, 1.25 2.3, 1.3
   2.32),CircularString(1.2 2.32, 1.25 2.34, 1.3 2.32))"
   nose,"CircularString(1.1 2.16, 1.1 2.24, 1.1 2.16)"
   mouth,"CircularString(0.9 2.10, 1.1 2.00, 1.3 2.10)"
   hair,"MultiCurve(CircularString(1.1 2.5, 1.0 2.48, 0.8
   2.4),CircularString(1.1 2.5, 1.0 2.52, 0.7 2.5),CircularString(1.1 2.5,
   1.0 2.56, 0.9 2.6),CircularString(1.1 2.5, 1.05 2.57, 1.0 2.6))"
   neck,"LineString(1.1 1.9, 1.1 1.8)"
   clothes and box,"MultiSurface(((1.6 1.9, 1.9 1.9, 1.9 2.2, 1.6 2.2, 1.6
   1.9)),((1.1 1.8, 0.7 1.2, 1.5 1.2, 1.1 1.8)))"
   holes in box,"MultiPoint((1.65 1.95),(1.75 1.95),(1.85 1.95),(1.65 2.05),
   (1.75 2.05),(1.85 2.05),(1.65 2.15),(1.75 2.15),(1.85 2.15))"
   arms and legs,"MultiLineString((0.9 1.2, 0.9 0.8),(1.3 1.2, 1.3 0.8),
   (0.97 1.6, 1.6 1.9),(1.23 1.6, 1.7 1.9))"
   left cart wheel,"CircularString(2.05 0.8, 2.05 0.9, 2.05 0.8)"
   right cart wheel,"CircularString(2.95 0.8, 2.95 0.9, 2.95 0.8)"
   cart body,"Polygon((1.9 0.9, 1.9 1.0, 3.1 1.0, 3.1 0.9, 1.9 0.9))"
   angular shapes on cart,"MultiPolygon(((2.18 1.0, 2.1 1.2, 2.3 1.4, 2.5
   1.2, 2.35 1.0, 2.18 1.0)),((2.3 1.4, 2.57 1.6, 2.7 1.3, 2.3 1.4)))"
   round shape on cart,"CurvePolygon(CompoundCurve(CircularString(2.6 1.0,
   2.7 1.3, 2.8 1.0),(2.8 1.0, 2.6 1.0)))"
   cart handle,"GeometryCollection(MultiCurve((2.0 1.0, 2.1
   1.0),CircularString(2.0 1.0, 1.98 1.1, 1.9 1.2),CircularString(2.1 1.0,
   2.08 1.1, 2.0 1.2),(1.9 1.2, 1.85 1.3),(2.0 1.2, 1.9 1.35),(1.85 1.3,
   ```

```
1.9 1.35)),CircularString(1.85 1.3, 1.835 1.29, 1.825
1.315),CircularString(1.9 1.35, 1.895 1.38, 1.88 1.365),LineString(1.825
1.315, 1.88 1.365))"
```

2. In Interactive SQL, connect to the sample database (*demo.db*) as a DBA user or as a member of the SYS_SPATIAL_ADMIN_ROLE group.

3. Create a table called SA_WKT and load the data from *wktgeometries.csv* into it. Replace the path to the *.csv* file with the path where you saved the file:

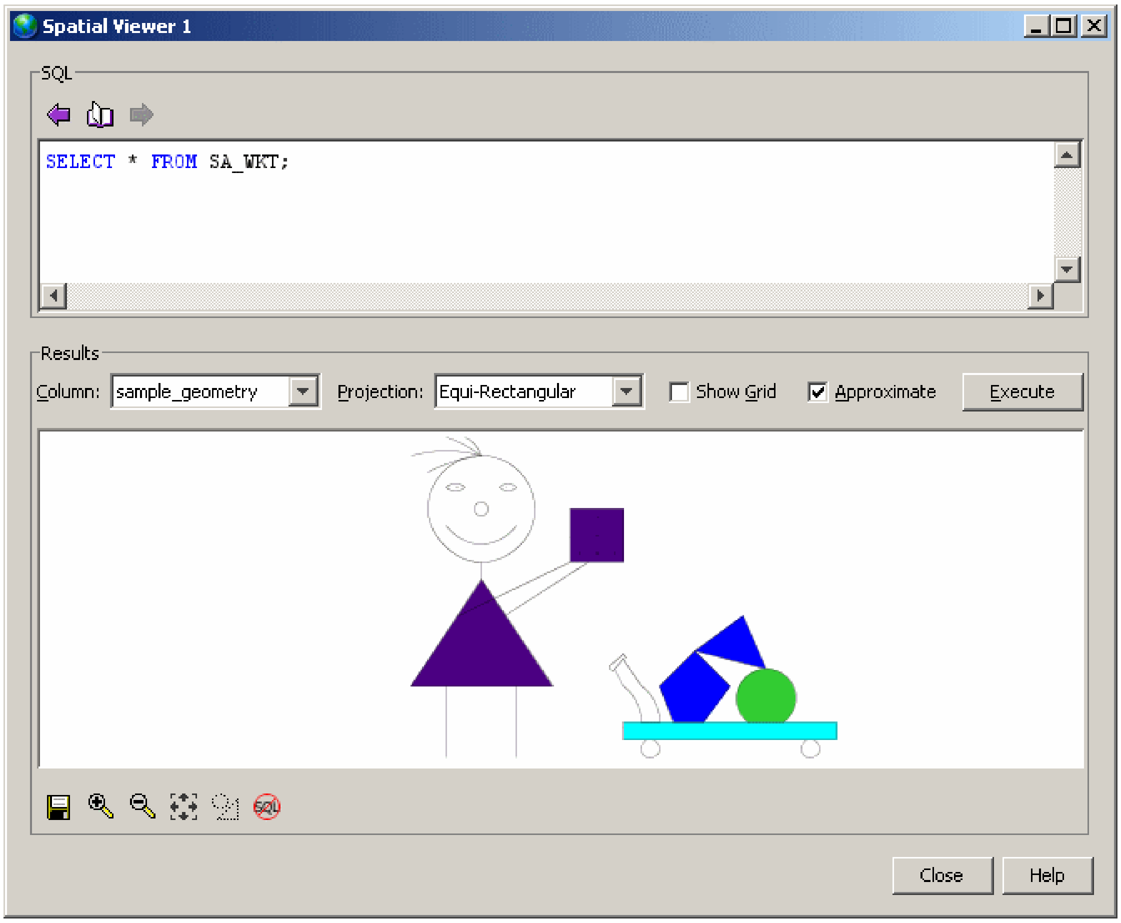
   ```
   DROP TABLE IF EXISTS SA_WKT;
   CREATE TABLE SA_WKT (
       description CHAR(24),
       sample_geometry ST_Geometry(SRID=1000004326)
   );

   LOAD TABLE SA_WKT FROM 'C:\\Documents and Settings\\All Users\\Documents\
   \SQL Anywhere 12\\Samples\\wktgeometries.csv' DELIMITED BY ',';
   ```

   The data is loaded into the table.

4. In Interactive SQL, click **Tools** » **Spatial Viewer**.

5. In the **Spatial Viewer**, execute the following statement to see the geometries:

   ```
   SELECT * FROM SA_WKT;
   ```



```
SELECT * FROM SA_WKT;
```

Column: sample_geometry   Projection: Equi-Rectangular   ☐ Show Grid   ☑ Approximate   Execute

6. Your data may have several columns of spatial data. You can create a file of WKT data containing one of each supported spatial data type, stored in individual columns.

   Copy the following code into your text editor and save the file as *wktgeometries2.csv*:

```
"Point(0 0)",,,,,,,,,,,,,,
,"LineString(0 0, 1 1)",,,,,,,,,,,,,
,,"CircularString(0 0, 1 1, 0 0)",,,,,,,,,,,,
,,,"CompoundCurve(CircularString(0 0, 1 1, 1 0),(1 0, 0 1))",,,,,,,,,,,
,,,,"CompoundCurve(CircularString(0 0, 1 1, 1 0),(1 0, 0 1),(0 1, 0
0))",,,,,,,,,,
,,,,,"Polygon((-1 0, 1 0, 2 1, 0 3, -2 1, -1 0))",,,,,,,,,
,,,,,,"CurvePolygon(CompoundCurve(CircularString(0 0, 1 1, 1 0),(1 0, 0
0)))",,,,,,,,
,,,,,,,"CurvePolygon(CompoundCurve(CircularString(0 0, 2 1, 2 0),(2 0, 0
0)))",,,,,,,
,,,,,,,,"MultiPoint((2 0),(0 0),(3 0),(1 0))",,,,,,
,,,,,,,,,"MultiPolygon(((4 0, 4 1, 5 1, 5 0, 4 0)),((-1 0, 1 0, 2 1, 0
3, -2 1, -1 0)))",,,,,
,,,,,,,,,,"MultiSurface(((4 0, 4 1, 5 1, 5 0, 4
0)),CurvePolygon(CompoundCurve(CircularString(0 0, 2 1, 2 0),(2 0, 0
0))))",,,,
,,,,,,,,,,,"MultiLineString((2 0, 0 0),(3 0, 1 0),(-2 1, 0 4))",,,
,,,,,,,,,,,,"MultiCurve((3 2, 4 3),CircularString(0 0, 1 1, 0 0))",,
,,,,,,,,,,,,,"GeometryCollection(MultiPoint((2 0),(0 0),(3 0),(1
0)),MultiSurface(((4 0, 4 1, 5 1, 5 0, 4
0)),CurvePolygon(CompoundCurve(CircularString(0 0, 2 1, 2 0),(2 0, 0
0)))),MultiCurve((3 2, 4 3),CircularString(0 0, 1 1, 0 0)))",
,,,,,,,,,,,,,,"GeometryCollection(Point(0
0),CompoundCurve(CircularString(0 0, 1 1, 1 0),(1 0, 0 1),(0 1, 0
0)),CurvePolygon(CompoundCurve(CircularString(0 0, 2 1, 2 0),(2 0, 0
0))),MultiPoint((2 0),(0 0),(3 0),(1 0)),MultiSurface(((4 0, 4 1, 5 1, 5
0, 4 0)),CurvePolygon(CompoundCurve(CircularString(0 0, 2 1, 2 0),(2 0,
0 0)))),MultiCurve((3 2, 4 3),CircularString(0 0, 1 1, 0 0)))"
```

7. Create a table called SA_WKT2 and load the data from *wktgeometries2.csv* into it by executing the following statements. Be sure to replace the path to the *.csv* file with the path where you saved the file:

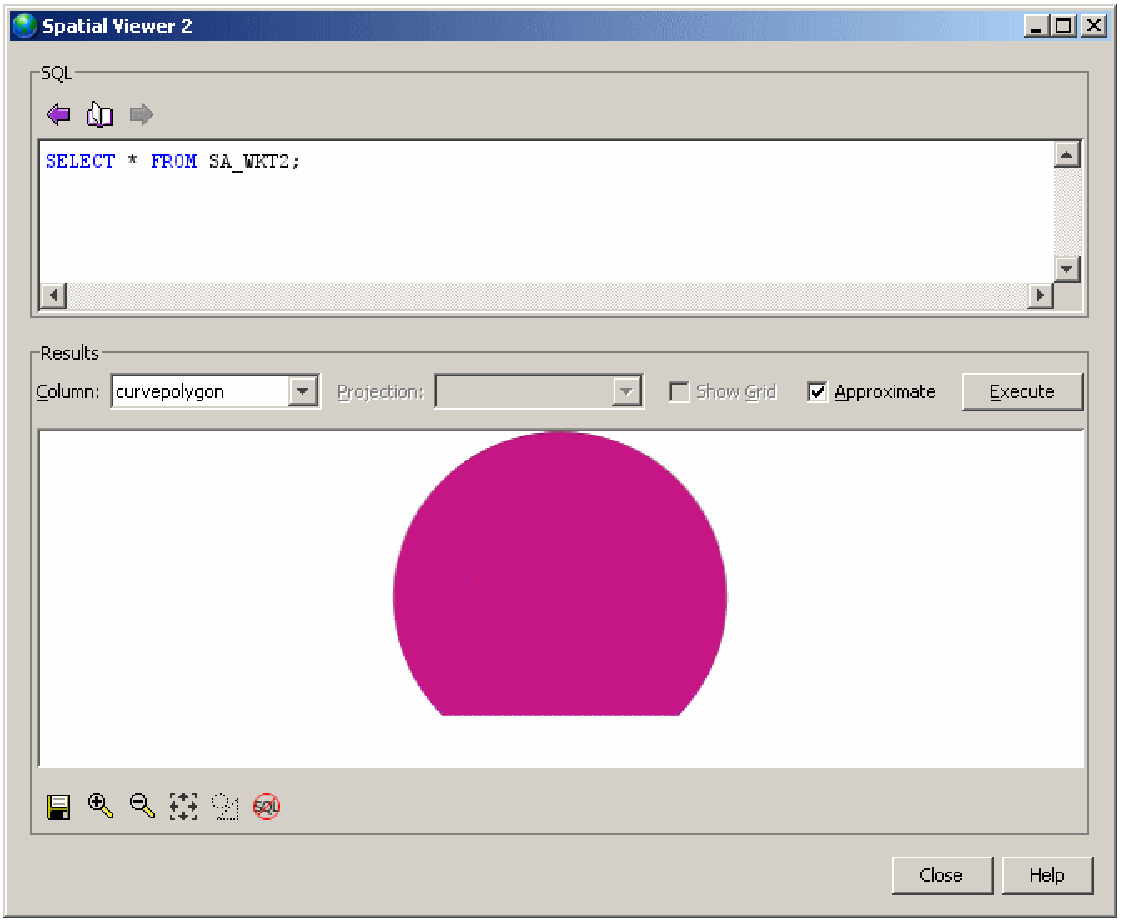
```
DROP TABLE IF EXISTS SA_WKT2;
CREATE TABLE SA_WKT2 (
    point          ST_Point,
    line           ST_LineString,
    circle         ST_CircularString,
    compoundcurve  ST_CompoundCurve,
    curve          ST_Curve,
    polygon1       ST_Polygon,
    curvepolygon   ST_CurvePolygon,
    surface        ST_Surface,
    multipoint     ST_MultiPoint,
    multipolygon   ST_MultiPolygon,
    multisurface   ST_MultiSurface,
    multiline      ST_MultiLineString,
    multicurve     ST_MultiCurve,
    geomcollection ST_GeomCollection,
    geometry       ST_Geometry
);

LOAD TABLE SA_WKT2 FROM 'C:\\Documents and Settings\\All Users\\Documents
\\SQL Anywhere 12\\Samples\\wktgeometries2.csv' DELIMITED BY ',';
```

The data is loaded into the table.

8. In the **Spatial Viewer**, execute the following statement to see the geometries:

```
SELECT * FROM SA_WKT2;
```

You can only see one column of data at a time; you must use the **Column** dropdown in the **Results** area to view the geometries for the other columns. For example, this is the view of the geometry in the curvepolygon column:



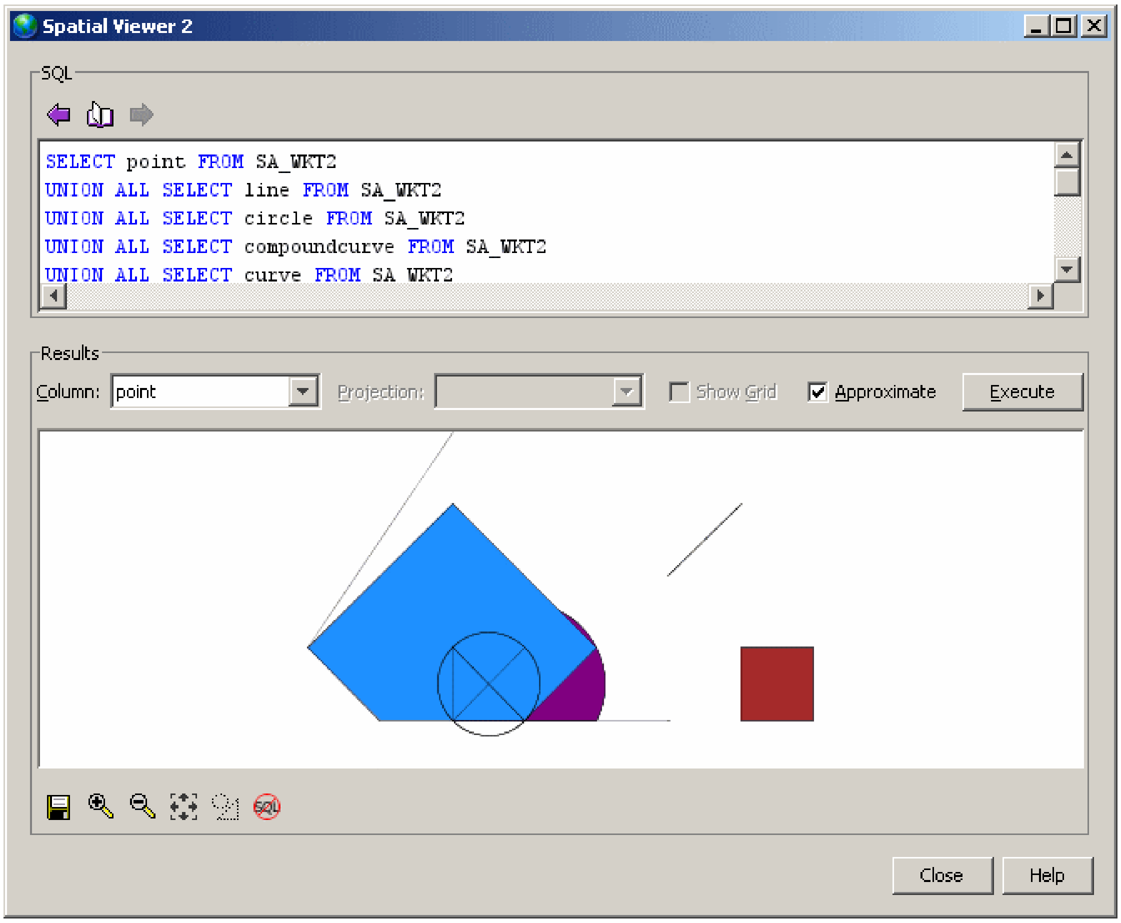
```
SELECT * FROM SA_WKT2;
```

Column: curvepolygon

Approximate: ✓

9. To view the geometries from all of the columns at once, execute a SELECT statement for each column and UNION ALL the results, as follows:

```
SELECT point FROM SA_WKT2
UNION ALL SELECT line FROM SA_WKT2
UNION ALL SELECT circle FROM SA_WKT2
UNION ALL SELECT compoundcurve FROM SA_WKT2
UNION ALL SELECT curve FROM SA_WKT2
UNION ALL SELECT polygon1 FROM SA_WKT2
UNION ALL SELECT curvepolygon FROM SA_WKT2
UNION ALL SELECT surface FROM SA_WKT2
UNION ALL SELECT multipoint FROM SA_WKT2
UNION ALL SELECT multipolygon FROM SA_WKT2
UNION ALL SELECT multisurface FROM SA_WKT2
UNION ALL SELECT multiline FROM SA_WKT2
UNION ALL SELECT multicurve FROM SA_WKT2
UNION ALL SELECT geomcollection FROM SA_WKT2
UNION ALL SELECT geometry FROM SA_WKT2
```



```
SELECT point FROM SA_WKT2
UNION ALL SELECT line FROM SA_WKT2
UNION ALL SELECT circle FROM SA_WKT2
UNION ALL SELECT compoundcurve FROM SA_WKT2
UNION ALL SELECT curve FROM SA_WKT2
```

See also

- "Viewing spatial data as images (Spatial Viewer)" on page 27
- "LOAD TABLE statement" [*SQL Anywhere Server - SQL Reference*]

Creating a unit of measure (Sybase Central)

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.

Prerequisites

DBA authority, or a member of the SYS_SPATIAL_ADMIN_ROLE group.

Context and remarks

Many.

Create a unit of measure (Sybase Central)

1. Connect to the database as a user with DBA authority, or as a member of the SYS_SPATIAL_ADMIN_ROLE group.

2. In the left pane, click **Spatial Reference Systems**.

3. In the right pane, click the **Units of Measure** tab.

4. Right-click the tab and click **New** » **Unit Of Measure**.

5. Click **Create A Custom Unit of Measure**, and then click **Next**.

6. Specify a name in the **What Do You Want To Name The New Unit Of Measure?** field, and then click **Next**.

7. Select **Linear** in the **Which Type of Unit Of Measure Do You Want To Create?** field.

8. Follow the instructions in the **Create Unit Of Measure Wizard**.

9. Click **Finish**.

Results

A unit of measure is created.

Next

You can grant permissions to users in your database to create, alter, or drop spatial reference systems and units of measure.

See also

- “CREATE SPATIAL UNIT OF MEASURE statement” [*SQL Anywhere Server - SQL Reference*]
- “Units of measure” on page 5

Creating a spatial reference system (Sybase Central)

You can create a spatial reference system (SRS) that uses an existing one as a template on which to base your settings using the **Create Spatial Reference System Wizard**.

Prerequisites

DBA authority or a member of the SYS_SPATIAL_ADMIN_ROLE group.

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

Context and remarks

When you create a spatial reference system, you use an existing one as a template on which to base your settings. Therefore you should choose a spatial reference system that is similar to the one you want to create. Later, you can edit the settings.

Create a spatial reference system (Sybase Central)

1. In Sybase Central, connect to the database as a user with DBA authority, or as a member of the SYS_SPATIAL_ADMIN_ROLE group.
2. In the left pane, right-click **Spatial Reference Systems** » **New** » **Spatial Reference System**.
3. Select **Let Me Choose From The List Of All Predefined Spatial Reference Systems**, and then click **Next**.
4. Follow the instructions in the wizard.
5. When you create a spatial reference system based on an existing spatial reference system, you set the SRID value to be 1000000000 plus the Well Known value.

   Note
   When assigning a SRID, review the recommendations provided for ranges of numbers to avoid. These recommendations are found in the IDENTIFIED clause of the CREATE SPATIAL REFERENCE SYSTEM statement.

6. The definition for the spatial reference system appears.
7. If you are satisfied with the definition for the spatial reference system, click **Finish**.

Results

The new spatial reference system is added to the database.

Next

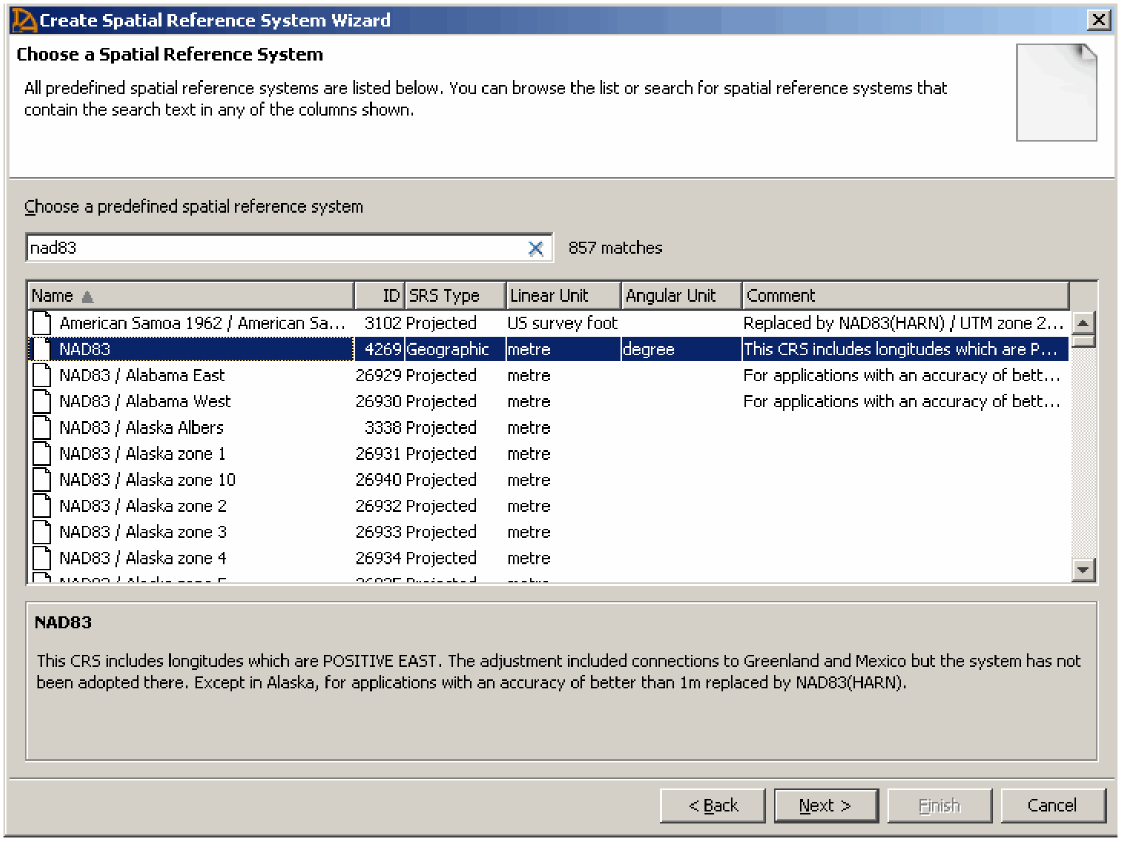
You can define the tables that use the spatial reference system.

Example

This example creates a spatial reference system based on an existing one.

1. Connect to the database as a user with DBA authority, or as a member of the SYS_SPATIAL_ADMIN_ROLE group.
2. In the left pane, right-click **Spatial Reference Systems** » **New** » **Spatial Reference System**.
3. Select **Let Me Choose From The List Of All Predefined Spatial Reference Systems**, and then click **Next**.

The **Choose A Spatial Reference System** window appears.



Create Spatial Reference System Wizard

Choose a Spatial Reference System

All predefined spatial reference systems are listed below. You can browse the list or search for spatial reference systems that contain the search text in any of the columns shown.

Choose a predefined spatial reference system

nad83 — 857 matches

Name	ID	SRS Type	Linear Unit	Angular Unit	Comment
American Samoa 1962 / American Sa...	3102	Projected	US survey foot		Replaced by NAD83(HARN) / UTM zone 2...
NAD83	4269	Geographic	metre	degree	This CRS includes longitudes which are P...
NAD83 / Alabama East	26929	Projected	metre		For applications with an accuracy of bett...
NAD83 / Alabama West	26930	Projected	metre		For applications with an accuracy of bett...
NAD83 / Alaska Albers	3338	Projected	metre		
NAD83 / Alaska zone 1	26931	Projected	metre		
NAD83 / Alaska zone 10	26940	Projected	metre		
NAD83 / Alaska zone 2	26932	Projected	metre		
NAD83 / Alaska zone 3	26933	Projected	metre		
NAD83 / Alaska zone 4	26934	Projected	metre		

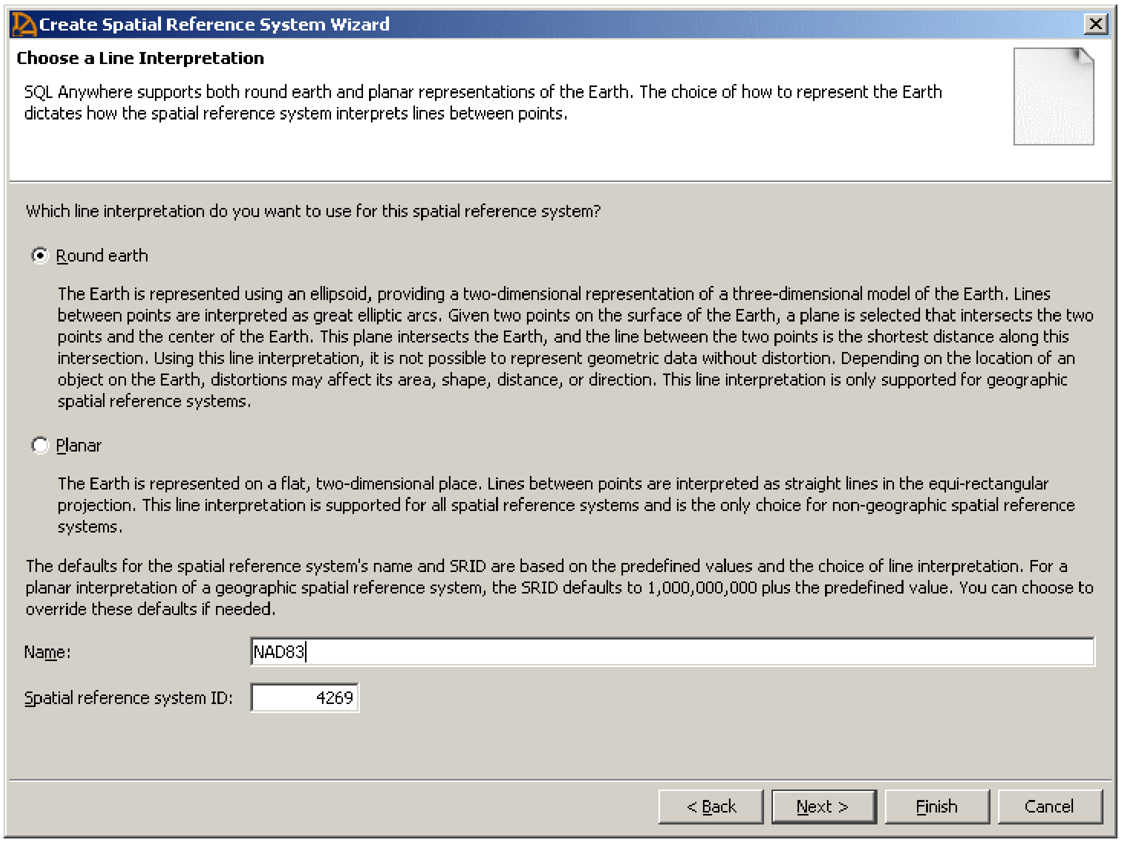
NAD83

This CRS includes longitudes which are POSITIVE EAST. The adjustment included connections to Greenland and Mexico but the system has not been adopted there. Except in Alaska, for applications with an accuracy of better than 1m replaced by NAD83(HARN).

4. To create a spatial reference system based on the NAD83 spatial reference system, type **NAD83**. As you type a name or ID in the **Choose A Predefined Spatial Reference System** field, the list of spatial reference systems moves to display the spatial reference system you want to use as a template.

5. Click **NAD83** and then click **Next**.

   The **Choose a Line Interpretation** window appears.



Choose a Line Interpretation

SQL Anywhere supports both round earth and planar representations of the Earth. The choice of how to represent the Earth dictates how the spatial reference system interprets lines between points.

Which line interpretation do you want to use for this spatial reference system?

[✓ Round earth]

The Earth is represented using an ellipsoid, providing a two-dimensional representation of a three-dimensional model of the Earth. Lines between points are interpreted 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. Using this line interpretation, it is not possible to represent geometric data without distortion. Depending on the location of an object on the Earth, distortions may affect its area, shape, distance, or direction. This line interpretation is only supported for geographic spatial reference systems.

Planar

The Earth is represented on a flat, two-dimensional place. Lines between points are interpreted as straight lines in the equi-rectangular projection. This line interpretation is supported for all spatial reference systems and is the only choice for non-geographic spatial reference systems.

The defaults for the spatial reference system's name and SRID are based on the predefined values and the choice of line interpretation. For a planar interpretation of a geographic spatial reference system, the SRID defaults to 1,000,000,000 plus the predefined value. You can choose to override these defaults if needed.

Name: NAD83

Spatial reference system ID: 4269

6. Click **Round Earth** to select it as the line interpretation.

7. Specify **NAD83custom** in the **Name** field.

8. When you create a spatial reference system based on an existing spatial reference system, you set the SRID value to be 1000000000 plus the Well Known value. For example, change the value in the **Spatial Reference System ID** field from 4269 to **1000004269**.

Note
When assigning a SRID, review the recommendations provided for ranges of numbers to avoid. These recommendations are found in the IDENTIFIED clause of the CREATE SPATIAL REFERENCE SYSTEM statement.

9. Click **Next**.

   The **Specify a Comment** window appears.

10. Optionally, specify a description for the spatial reference system, and then click **Next**.

11. Click **Finish**.

    The definition for the spatial reference system appears.

12. Click **Finish**.

See also

- "Units of measure" on page 5
- "CREATE SPATIAL UNIT OF MEASURE statement" [*SQL Anywhere Server - SQL Reference*]
- "IDENTIFIED BY clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*]
- "CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*]
- "Spatial reference systems (SRS) and Spatial reference identifiers (SRID)" on page 2

Grant spatial permissions

To create, alter, or drop spatial reference systems and units of measure, you must be a user with DBA permissions or belong to the SYS_SPATIAL_ADMIN_ROLE group.

See also

- "Granting group membership to an existing user or group" [*SQL Anywhere Server - Database Administration*]

Advanced spatial topics

This section contains advanced spatial topics.

How flat-Earth and round-Earth representations work

SQL Anywhere 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.

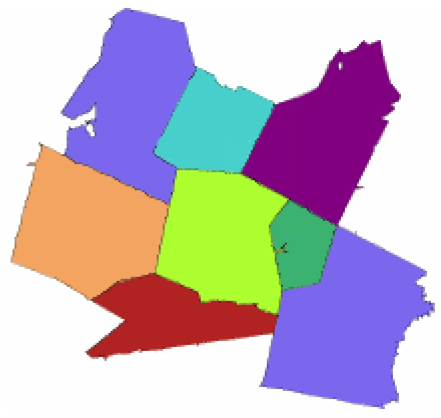
Some predicates and set operations are also not available.

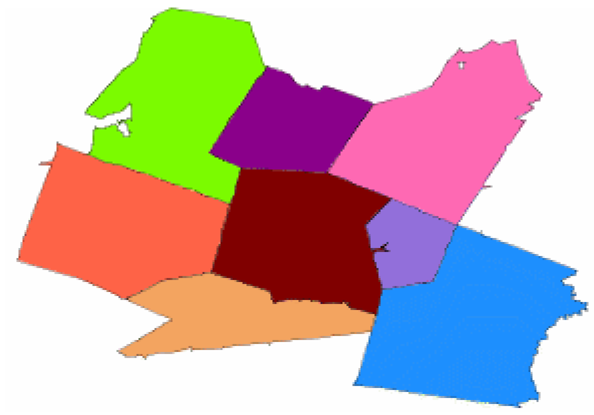
Circularstrings are not allowed in round-Earth spatial reference systems.

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.

See also

- "ST_Transform method for type ST_Geometry" on page 222

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. SQL Anywhere 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, SQL Anywhere 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.

Note that 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.

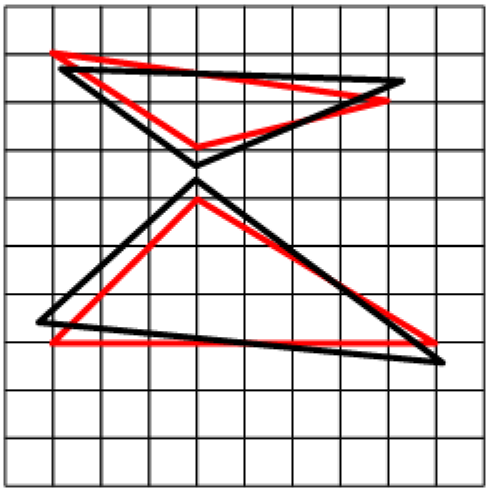
Snap-to-grid and tolerance are set on the spatial reference system and are always specified in same units as the X and Y (or Longitude and Latitude) coordinates. Snap-to-grid and tolerance work together to overcome issues with inexact arithmetic and imprecise data. However, you should be aware of how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

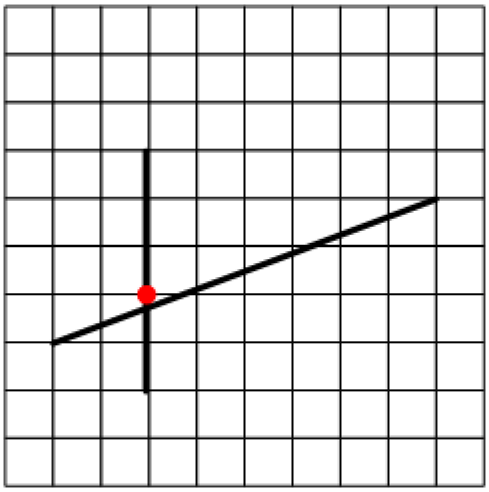


Example 2: Tolerance impacts intersection results

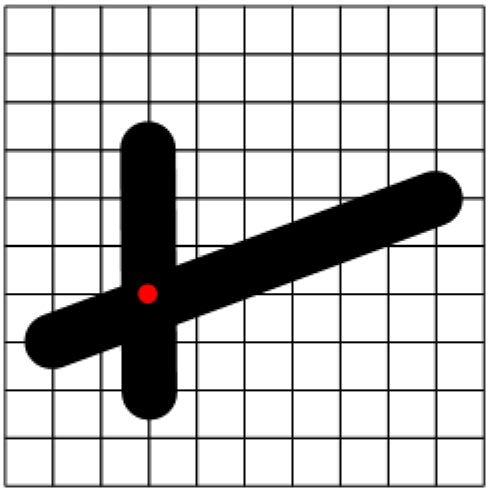
In the following example, two lines lie in a spatial reference system where tolerance is set to 0. The intersection point of the two lines is snapped to the nearest vertex in the grid. Since tolerance is set to 0, a test to determine if the intersection point of the two lines intersects the diagonal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

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

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

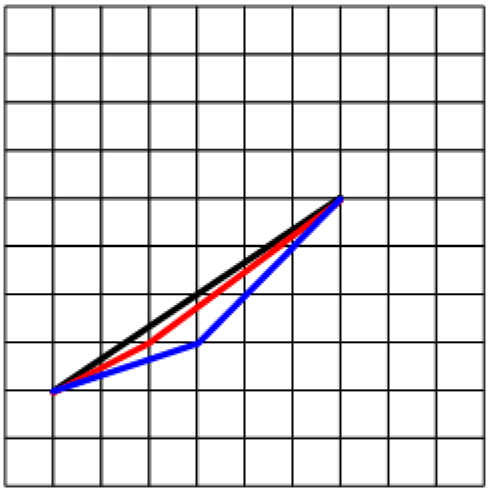


Setting the tolerance to grid size (the default), however, causes the intersection point to be inside the thick diagonal line. So a test of whether the intersection point intersects the diagonal line within tolerance would pass:



Example 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.

Note that 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 SQL Anywhere 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.

See also

- “SNAP TO GRID clause, CREATE SPATIAL REFERENCE SYSTEM statement” [*SQL Anywhere Server - SQL Reference*]
- “TOLERANCE clause, CREATE SPATIAL REFERENCE SYSTEM statement” [*SQL Anywhere Server - SQL Reference*]
- “ST_Equals method for type ST_Geometry” on page 167
- “ST_SnapToGrid method for type ST_Geometry” on page 201
- “ST_OrderingEquals method for type ST_Geometry” on page 192
- “Supported spatial predicates” on page 10
- “CREATE SPATIAL REFERENCE SYSTEM statement” [*SQL Anywhere Server - SQL Reference*]
- “ALTER SPATIAL REFERENCE SYSTEM statement” [*SQL Anywhere Server - SQL Reference*]

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 )' );
   ```

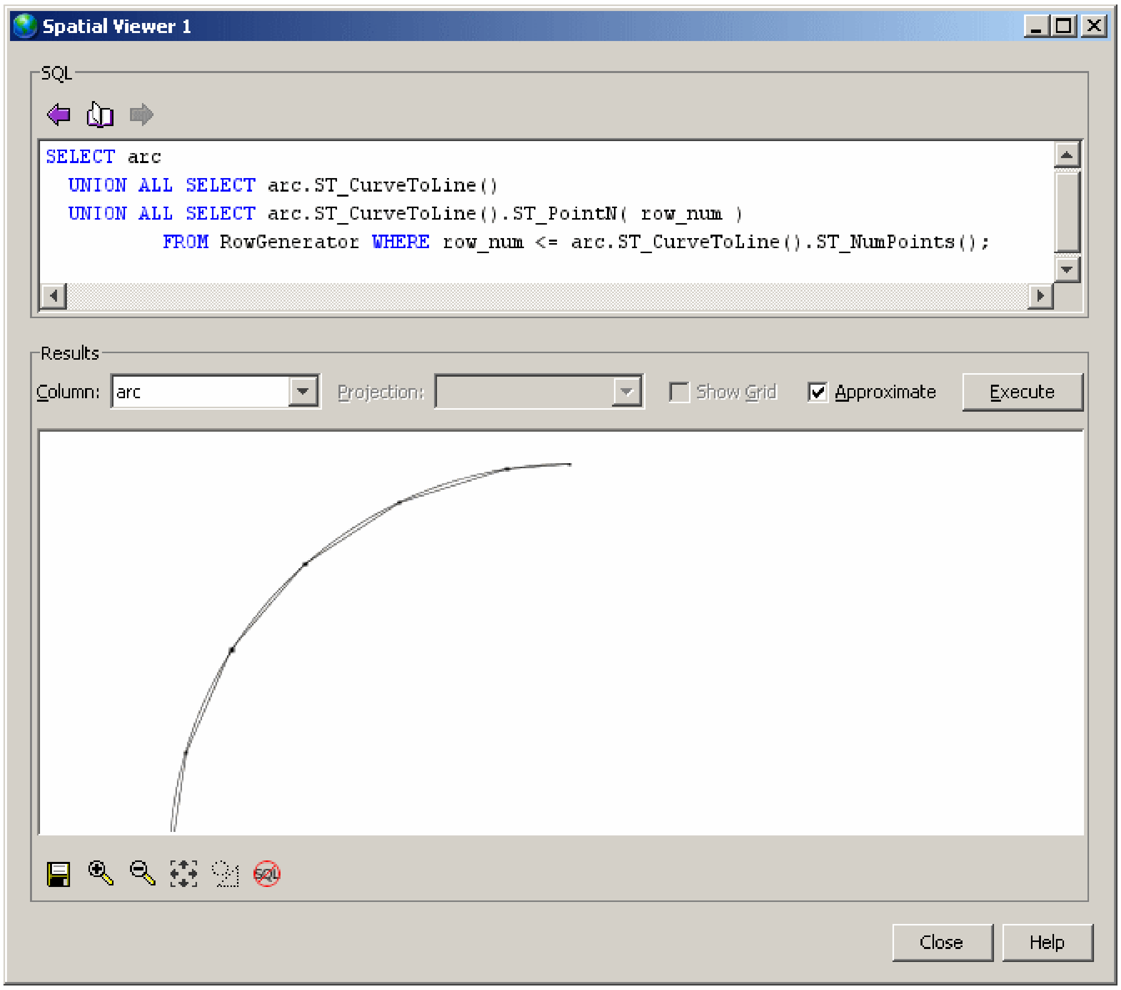
3. Execute the following statement to temporarily set the relative tolerance to 1% using the st_geometry_interpolation option.

   ```
   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. For more information about relative tolerance and how to set it, see “st_geometry_interpolation option” [*SQL Anywhere Server - Database Administration*].

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();
   ```



```
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();
```

Column: arc

[✓] Approximate

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.

See also

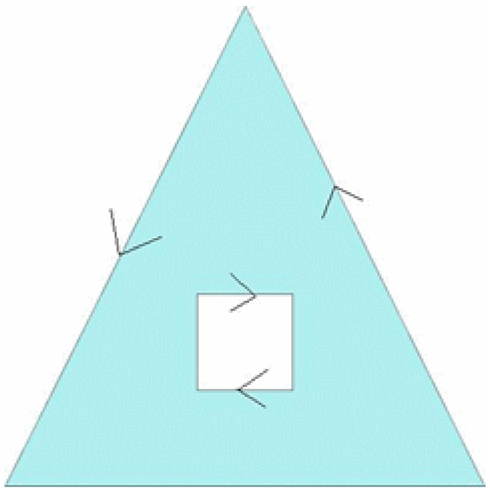
- "st_geometry_interpolation option" [*SQL Anywhere Server - Database Administration*]

How polygon ring orientation works

Internally, SQL Anywhere 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 SQL Anywhere 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.

See also

- "POLYGON FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

How geometry interiors, exteriors, and boundaries work

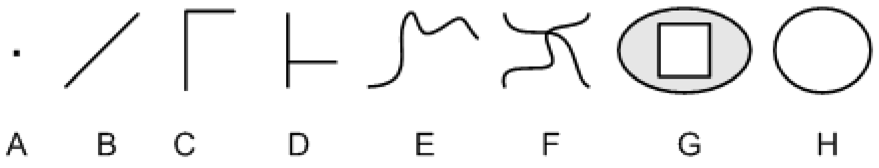
The **interior** of a geometry is all points that are part of the geometry except the boundary.

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.

See also

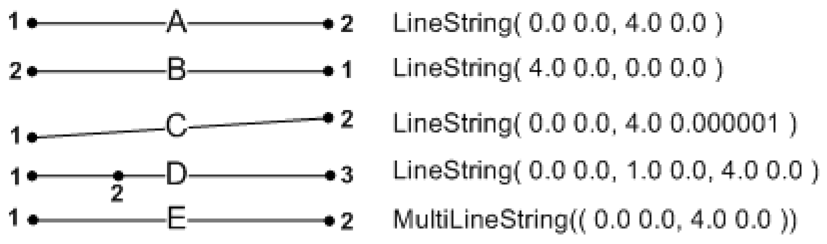
- "ST_Boundary method for type ST_Geometry" on page 147

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.


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

How SQL Anywhere 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 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, you should always 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) ∩ Interior(g2)`	`Interior(g1) ∩ Boundary(g2)`	`Interior(g1) ∩ Exterior(g2)`
g1 boundary	`Boundary(g1) ∩ Interior(g2)`	`Boundary(g1) ∩ Boundary(g2)`	`Boundary(g1) ∩ Exterior(g2)`
g1 exterior	`Exterior(g1) ∩ Interior(g2)`	`Exterior(g1) ∩ Boundary(g2)`	`Exterior(g1) ∩ Exterior(g2)`

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

When you specify:	**The intersection test result must return:**
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***' );
```

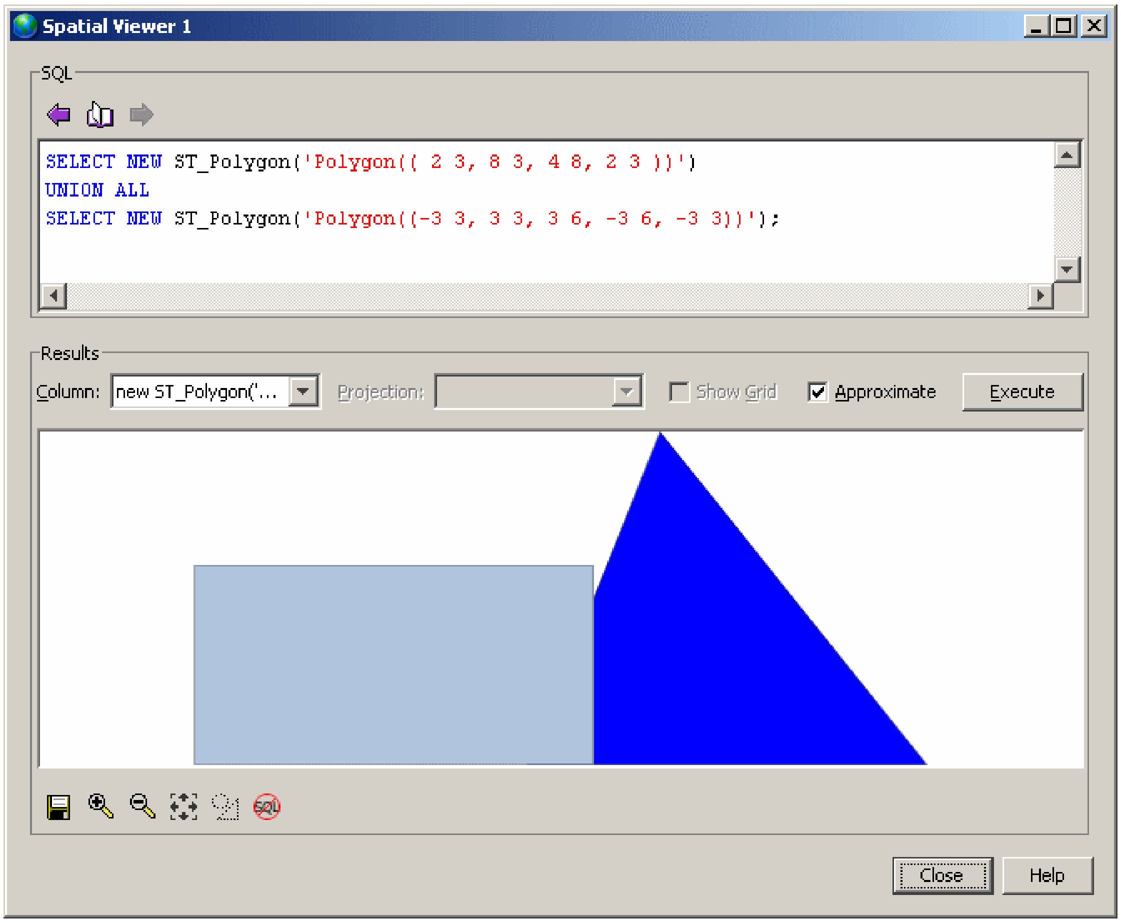
This is equivalent to asking ST_Relate to look for the following conditions when performing the intersection tests:

	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))');
```



```
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))');
```

See also

- "ST_Relate(ST_Geometry,CHAR(9)) method for type ST_Geometry" on page 195

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

See also

- "ST_Intersects method for type ST_Geometry" on page 178
- "ST_Overlaps method for type ST_Geometry" on page 193
- "ST_Within method for type ST_Geometry" on page 225
- "ST_Disjoint method for type ST_Geometry" on page 163
- "ST_Touches method for type ST_Geometry" on page 221
- "ST_Crosses method for type ST_Geometry" on page 159
- "ST_Contains method for type ST_Geometry" on page 148
- "ST_Relate method for type ST_Geometry" on page 195
- "ST_Relate(ST_Geometry) method for type ST_Geometry" on page 197

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

See also

- "ST_CoordDim method for type ST_Geometry" on page 153
- "ST_Dimension method for type ST_Geometry" on page 162

Tutorial: Experimenting with the spatial features

This tutorial allows you to experiment with some of the spatial features in SQL Anywhere. To do so, you will first load an ESRI shapefile into your sample database (demo.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" on page 55
- "Lesson 2: Download the ESRI shapefile data" on page 56
- "Lesson 3: Load the ESRI shapefile data" on page 57
- "Lesson 4: Query spatial data" on page 59
- "Lesson 5: Output spatial data to SVG" on page 61
- "Lesson 6: Project spatial data" on page 63

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.

Install the predefined units of measure and spatial reference systems

1. Using Interactive SQL, start and connect to the sample database (demo.db) as user DBA, or as a member of the SYS_SPATIAL_ADMIN_ROLE group.

   The sample database is located in *%SQLANYSAMP12%*.

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.

See “sa_install_feature system procedure” [*SQL Anywhere Server - SQL Reference*] and “CREATE SPATIAL REFERENCE SYSTEM statement” [*SQL Anywhere Server - SQL Reference*].

3. To determine the units of measure installed in your database, execute the following query in Interactive SQL:

   ```
   SELECT * FROM ST_UNITS_OF_MEASURE;
   ```

   See “ST_UNITS_OF_MEASURE consolidated view” [*SQL Anywhere Server - SQL Reference*].

4. To determine the spatial reference systems installed in your database, you can look in the **Spatial Reference Systems** folder in Sybase Central, or execute the following query in Interactive SQL:

   ```
   SELECT * FROM ST_SPATIAL_REFERENCE_SYSTEMS;
   ```

   See “ST_SPATIAL_REFERENCE_SYSTEMS consolidated view” [*SQL Anywhere Server - SQL Reference*].

5. Proceed to “Lesson 2: Download the ESRI shapefile data” on page 56.

Lesson 2: Download the ESRI shapefile data

In this lesson, you will download an ESRI shapefile from the US Census website (www2.census.gov). The shapefile you download represents the Massachusetts 5-digit code zip code information used during the 2002 census tabulation. Each zip code area is treated as either a polygon or multipolygon.

Download sample spatial data

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 unpacks five files, including an ESRI shapefile (*.shp*) you will use to load the spatial data into the database.

7. Proceed to “Lesson 3: Load the ESRI shapefile data” on page 57.

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.

Load the spatial data from the ESRI shapefile into the database

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",0.0174532925199433]]
   ```

   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. Note that the load may take a minute.

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

   See "st_geometry_load_shapefile system procedure" [*SQL Anywhere Server - SQL Reference*]

   Note
   The **Import Wizard** also supports loading data from shapefiles. See "Import data with the Import Wizard (Interactive SQL)" [*SQL Anywhere Server - SQL Usage*].

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.

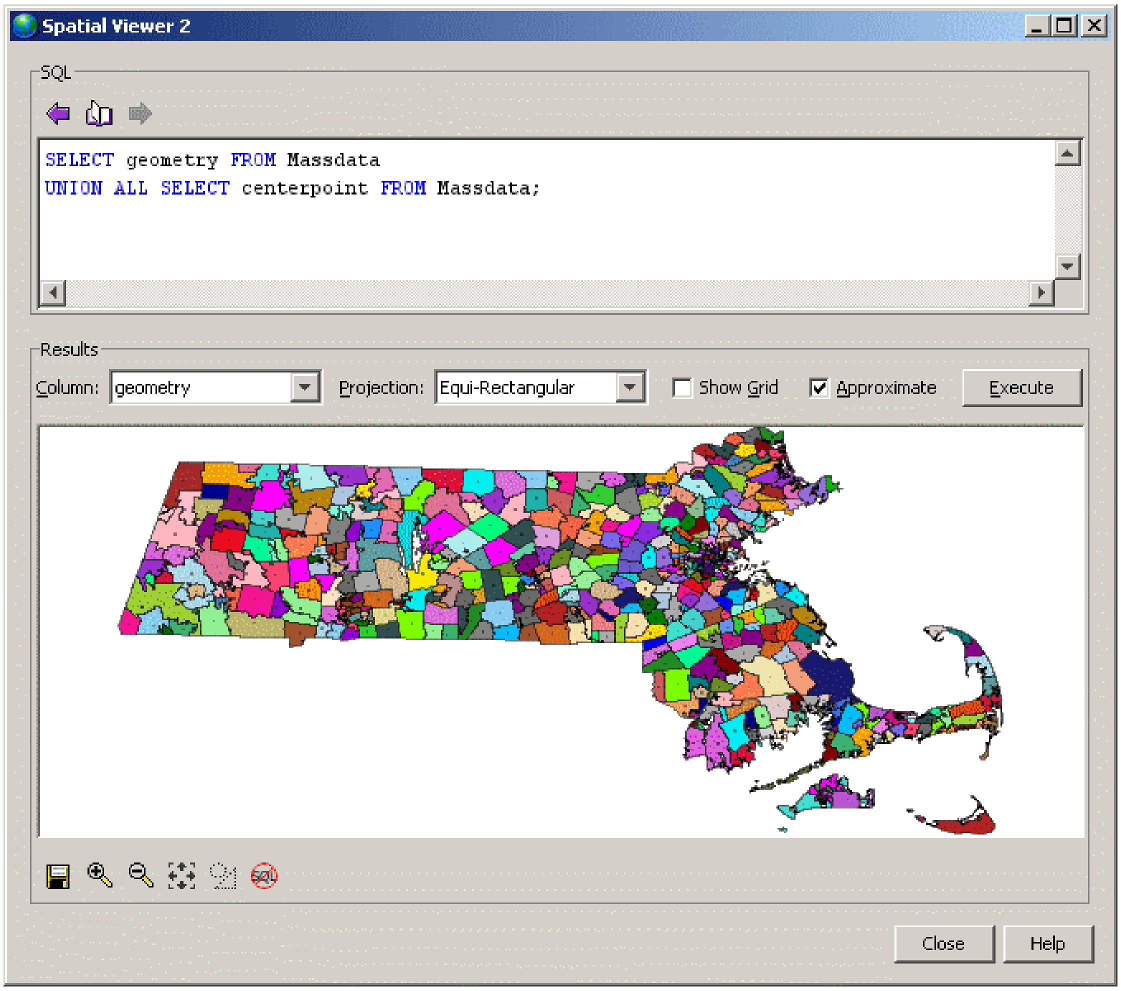
7. To view an individual geometry (a zip code region) as a shape, double-click any value except the first one in Massdata.geometry and then click the **Spatial Preview** tab of the **Value Of Column** window.

   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** » **SQL Anywhere**, 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 SQL Anywhere **Spatial Viewer** and execute the following query in Interactive SQL:

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



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

Column: geometry
Projection: Equi-Rectangular
Show Grid
[✓ Approximate]
Execute

9. Proceed to "Lesson 4: Query spatial data" on page 59.

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.

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.

You will also learn how to calculate distances, which requires you to add units of measurement to your database.

Query the spatial data

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, SpatialContacts c
   WHERE
   c.PostalCode = z.ZIP
   AND z.geometry.ST_Intersects( @Mass_01775 ) = 1;
   ```

   See "ST_Intersects method for type ST_Geometry" on page 178.

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;
   ```

   See "ST_SRID method for type ST_Geometry" on page 198

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. For information about how to project data to another spatial reference system, see "Lesson 6: Project spatial data" on page 63.

See "ST_CoveredBy method for type ST_Geometry" on page 155.

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, SpatialContacts c
WHERE c.PostalCode = z.ZIP
  AND dist <= 100
ORDER BY dist;
```

See "ST_Distance method for type ST_Geometry" on page 164.

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, 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;
```

See "ST_WithinDistance method for type ST_Geometry" on page 226.

7. Proceed to "Lesson 5: Output spatial data to SVG" on page 61.

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.

Output a geometry as SVG for viewing

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.

   If you receive an error saying that the full value could not be read from the database, you need to change the **Truncation Length** setting for Interactive SQL. To do this, in Interactive SQL click **Tools** » **Options** » **SQL Anywhere**, and set **Truncation Length** to a high number such as 100000. Execute your query again and view the geometry.

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:\\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.

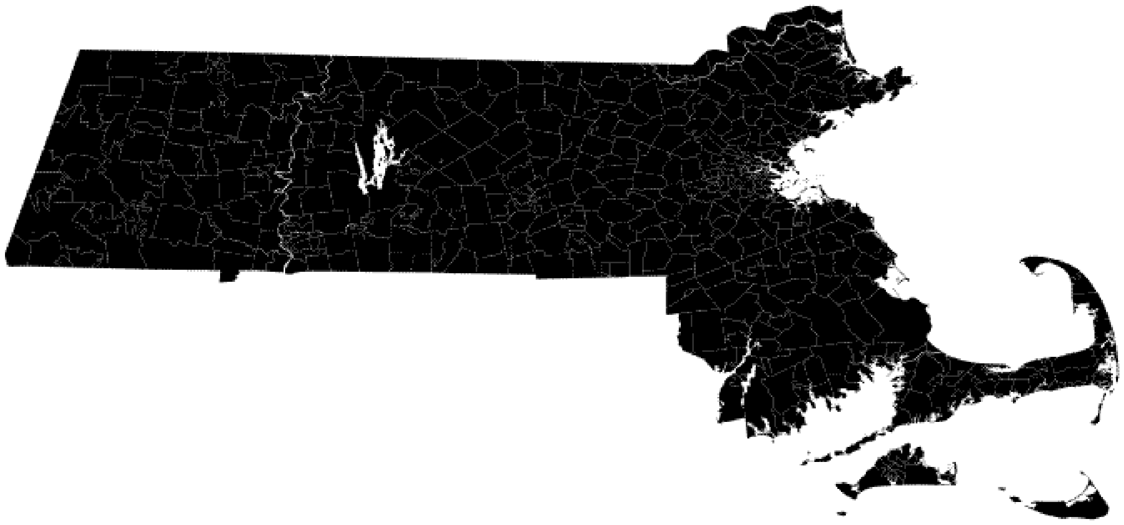
5. The ST_AsSVG method generates an SVG image from a single geometry. In some cases, you want to generate an SVG image including all of the shapes in a group. The ST_AsSVGAggr method is an aggregate function that combines multiple geometries into a single SVG image. First, using Interactive SQL, create 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 permissions are enabled. If you open the SVG image in an application that supports SVG data, you should see an image like the following:



You will notice that 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.

See:

- "OUTPUT statement [Interactive SQL]" [*SQL Anywhere Server - SQL Reference*]
- "ST_AsSVG method for type ST_Geometry" on page 120
- "ST_AsSVGAggr method for type ST_Geometry" on page 124
- "xp_write_file system procedure" [*SQL Anywhere Server - SQL Reference*]
- "WRITE_CLIENT_FILE function [String]" [*SQL Anywhere Server - SQL Reference*]

6. Proceed to "Lesson 6: Project spatial data" on page 63.

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.

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.

Project data

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 ) );
   ```

   See “ST_Transform method for type ST_Geometry” on page 222.

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.ST_Transfo
rm( 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. Note that the results for ST_Touches do not include the row for zip code 01775 (unlike the ST_Intersects method).

   ```
   DROP VARIABLE @Mass_01775;
   CREATE 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;
   ```

   See "ST_Touches method for type ST_Geometry" on page 221.

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** » **SQL Anywhere**, and set **Truncation Length** to a higher number. Execute your query again and view the geometry.

   See "ST_UnionAggr method for type ST_Geometry" on page 224.

7. You have now finished the **Experimenting with the spatial features** tutorial.

8. To restore the sample database (demo.db) to its original state, follow the steps in "Recreate the sample database (demo.db)" [*SQL Anywhere 12 - Introduction*].

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.

For a high level description of how the spatial data types are related, see "Supported spatial data types and their hierarchy" on page 7.

For compatibility with other products, SQL Anywhere also supports some SQL functions for working with spatial data. In almost all cases, these compatibility functions use one of the spatial methods to perform the operation, so a link to the underlying method is provided. See "Spatial compatibility functions" on page 324.

ST_CircularString type

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

Direct supertype

- "ST_Curve type"

Constructor

- "ST_CircularString constructor"

Methods

- Methods of ST_CircularString:

ST_NumPoints	ST_PointN

- All methods of "ST_Curve type" can also be called on a ST_CircularString type:

ST_CurveToLine	ST_EndPoint	ST_IsClosed	ST_IsRing
ST_Length	ST_StartPoint		

- All methods of "ST_Geometry type" can also be called on a ST_CircularString type:

ST_Affine	ST_AsBinary	ST_AsGML	ST_AsGeoJSON
ST_AsKML	ST_AsSVG	ST_AsSVGAggr	ST_AsText
ST_AsWKB	ST_AsWKT	ST_AsXML	ST_Boundary

ST_Contains	ST_ContainsFilter	ST_ConvexHull	ST_ConvexHullAggr
ST_CoordDim	ST_CoveredBy	ST_CoveredByFilter	ST_Covers
ST_CoversFilter	ST_Crosses	ST_Difference	ST_Dimension
ST_Disjoint	ST_Distance	ST_Envelope	ST_EnvelopeAggr
ST_Equals	ST_EqualsFilter	ST_GeomFromBinary	ST_GeomFromShape
ST_GeomFromText	ST_GeomFromWKB	ST_GeomFromWKT	ST_GeometryType
ST_GeometryType-FromBaseType	ST_Intersection	ST_IntersectionAggr	ST_Intersects
ST_IntersectsFilter	ST_IntersectsRect	ST_Is3D	ST_IsEmpty
ST_IsMeasured	ST_IsSimple	ST_IsValid	ST_LatNorth
ST_LatSouth	ST_LinearHash	ST_LinearUnHash	ST_LoadConfiguration-Data
ST_LongEast	ST_LongWest	ST_MMax	ST_MMin
ST_OrderingEquals	ST_Overlaps	ST_Relate	ST_Reverse
ST_SRID	ST_SRIDFromBase-Type	ST_SnapToGrid	ST_SymDifference
ST_ToCircular	ST_ToCompound	ST_ToCurve	ST_ToCurvePoly
ST_ToGeomColl	ST_ToLineString	ST_ToMultiCurve	ST_ToMultiLine
ST_ToMultiPoint	ST_ToMultiPolygon	ST_ToMultiSurface	ST_ToPoint
ST_ToPolygon	ST_ToSurface	ST_Touches	ST_Transform
ST_Union	ST_UnionAggr	ST_Within	ST_WithinDistance
ST_WithinDistanceFil-ter	ST_WithinFilter	ST_XMax	ST_XMin
ST_YMax	ST_YMin	ST_ZMax	ST_ZMin

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 constructor

Constructs a circularstring.

Overload list

Name	Description
"ST_CircularString() constructor"	Constructs a circularstring representing the empty set.
"ST_CircularString(LONG VARCHAR[, INT]) constructor"	Constructs a circularstring from a text representation.
"ST_CircularString(LONG BINARY[, INT]) constructor"	Constructs a circularstring from Well Known Binary (WKB).
"ST_CircularString(ST_Point,ST_Point,ST_Point,...) constructor"	Constructs a circularstring value from a list of points in a specified spatial reference system.

ST_CircularString() constructor

Constructs a circularstring representing the empty set.

Syntax

NEW ST_CircularString()

Standards and compatibility

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

Example

The following returns 1, indicating the value is empty.

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

ST_CircularString(LONG VARCHAR[, INT]) constructor

Constructs a circularstring from a text representation.

Syntax

NEW ST_CircularString(*text-representation*[, *srid*])

Parameters

Name	Type	Description
text-representation	LONG VARCHAR	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	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a circularstring from a character string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

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

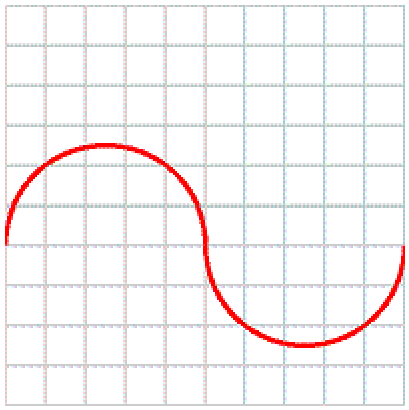
Example

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
```



ST_CircularString(LONG BINARY[, INT]) constructor

Constructs a circularstring from Well Known Binary (WKB).

Syntax

NEW ST_CircularString(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BI-NARY	A string containing the binary representation of an circularstring. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
srid	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a circularstring from a binary string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

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

Example

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

```
SELECT NEW
ST_CircularString(0x010800000003000000000000000000001440000000000000002440000000000
000002440000000000000002840000000000000002e400000000000002440)
```

ST_CircularString(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(*pt1*,*pt2*,*pt3*[,*pt4*,...,*ptN*])

Parameters

Name	Type	Description
pt1	ST_Point	The first point of a segment.
pt2	ST_Point	Any point on the segment between the first and last point.
pt3	ST_Point	The last point of a segment.
pt4,...,ptN	ST_Point	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.

Remarks

Constructs a circularstring value from a list of points. At least three points must be provided. The first of these three is the start of a segment, the third point is the end of the segment, and the second point is any point on the segment between the first and third point. Additional points can be specified in pairs to add more segments to the circularstring. All of the specified points must have the same SRID. The circularstring is constructed with this common SRID. All of the supplied points must be non-empty and have the same answer for Is3D and IsMeasured. The circularstring is 3D if all of the points are 3D, and the circularstring is measured if all of the points are measured.

Note
By default, ST_CircularString uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

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 ) )
```

ST_NumPoints method

Returns the number of points defining the circularstring.

Note
By default, ST_NumPoints uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

circularstring-expression.**ST_NumPoints**()

Returns

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

See also

- "ST_PointN method for type ST_CircularString"
- "ST_NumPoints method for type ST_LineString"

Standards and compatibility

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

Example

The following example returns the result 5.

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

ST_PointN method

Returns the *n*th point in the circularstring.

Note
By default, ST_PointN uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

circularstring-expression.**ST_PointN**(*n*)

Parameters

Name	Type	Description
n	INT	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.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *circularstring-expression*.

See also

- "ST_NumPoints method for type ST_CircularString"
- "ST_PointN method for type ST_LineString"

Standards and compatibility

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

Example

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)

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.

Direct supertype

- "ST_Curve type"

Constructor

- "ST_CompoundCurve constructor"

Methods

- Methods of ST_CompoundCurve:

ST_CurveN	ST_NumCurves

- All methods of "ST_Curve type" can also be called on a ST_CompoundCurve type:

ST_CurveToLine	ST_EndPoint	ST_IsClosed	ST_IsRing
ST_Length	ST_StartPoint		

- All methods of "ST_Geometry type" can also be called on a ST_CompoundCurve type:

ST_Affine	ST_AsBinary	ST_AsGML	ST_AsGeoJSON
ST_AsKML	ST_AsSVG	ST_AsSVGAggr	ST_AsText
ST_AsWKB	ST_AsWKT	ST_AsXML	ST_Boundary
ST_Contains	ST_ContainsFilter	ST_ConvexHull	ST_ConvexHullAggr
ST_CoordDim	ST_CoveredBy	ST_CoveredByFilter	ST_Covers
ST_CoversFilter	ST_Crosses	ST_Difference	ST_Dimension

ST_Disjoint	ST_Distance	ST_Envelope	ST_EnvelopeAggr
ST_Equals	ST_EqualsFilter	ST_GeomFromBinary	ST_GeomFromShape
ST_GeomFromText	ST_GeomFromWKB	ST_GeomFromWKT	ST_GeometryType
ST_GeometryType-FromBaseType	ST_Intersection	ST_IntersectionAggr	ST_Intersects
ST_IntersectsFilter	ST_IntersectsRect	ST_Is3D	ST_IsEmpty
ST_IsMeasured	ST_IsSimple	ST_IsValid	ST_LatNorth
ST_LatSouth	ST_LinearHash	ST_LinearUnHash	ST_LoadConfigura-tionData
ST_LongEast	ST_LongWest	ST_MMax	ST_MMin
ST_OrderingEquals	ST_Overlaps	ST_Relate	ST_Reverse
ST_SRID	ST_SRIDFromBase-Type	ST_SnapToGrid	ST_SymDifference
ST_ToCircular	ST_ToCompound	ST_ToCurve	ST_ToCurvePoly
ST_ToGeomColl	ST_ToLineString	ST_ToMultiCurve	ST_ToMultiLine
ST_ToMultiPoint	ST_ToMultiPolygon	ST_ToMultiSurface	ST_ToPoint
ST_ToPolygon	ST_ToSurface	ST_Touches	ST_Transform
ST_Union	ST_UnionAggr	ST_Within	ST_WithinDistance
ST_WithinDistanceFil-ter	ST_WithinFilter	ST_XMax	ST_XMin
ST_YMax	ST_YMin	ST_ZMax	ST_ZMin

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 7.4

ST_CompoundCurve constructor

Constructs a compound curve.

Overload list

Name	Description
"ST_CompoundCurve() constructor"	Constructs a compound curve representing the empty set.
"ST_CompoundCurve(LONG VARCHAR[, INT]) constructor"	Constructs a compound curve from a text representation.
"ST_CompoundCurve(LONG BINARY[, INT]) constructor"	Constructs a compound curve from Well Known Binary (WKB).
"ST_CompoundCurve(ST_Curve,...) constructor"	Constructs a compound curve from a list of curves.

ST_CompoundCurve() constructor

Constructs a compound curve representing the empty set.

Syntax

NEW ST_CompoundCurve()

Standards and compatibility

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

Example

The following returns 1, indicating the value is empty.

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

ST_CompoundCurve(LONG VARCHAR[, INT]) constructor

Constructs a compound curve from a text representation.

Syntax

NEW ST_CompoundCurve(*text-representation*[, *srid*])

Parameters

Name	Type	Description
text-representation	LONG VARCHAR	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	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a compound curve from a character string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

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

Example

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))')
```

ST_CompoundCurve(LONG BINARY[, INT]) constructor

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

Syntax

NEW ST_CompoundCurve(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BI-NARY	A string containing the binary representation of an compound curve. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
srid	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a compound curve from a binary string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

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

Example

The following returns CompoundCurve ((0 0, 5 10)).

```
SELECT NEW
ST_CompoundCurve(0x0109000000010000000102000000020000000000000000000000000000
00000000000000000000001440000000000000002440)
```

ST_CompoundCurve(ST_Curve,...) constructor

Constructs a compound curve from a list of curves.

Syntax

NEW ST_CompoundCurve(*curve1*[,*curve2*,...,*curveN*])

Parameters

Name	Type	Description
curve1	ST_Curve	The first curve to include in the compound curve.
curve2,...,curveN	ST_Curve	Additional curves to include in the compound curve.

Remarks

Constructs a compound curve from a list of constituent curves. The start point of each curve after the first must be coincident with the end point of the previous curve. All of the supplied curves must have the same SRID. The compound curve is constructed with this common SRID. All of the supplied curves must be non-empty and have the same answer for Is3D and IsMeasured. The compound curve is 3D if all of the points are 3D, and the compound curve is measured if all of the points are measured.

Note
By default, ST_CompoundCurve uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

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)'))
```

ST_CurveN method

Returns the *n*th curve in the compound curve.

Note
By default, ST_CurveN uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

compoundcurve-expression.**ST_CurveN**(*n*)

Parameters

Name	Type	Description
n	INT	The position of the element to return, from 1 to *compoundcurve-expression*.ST_NumCurves().

Returns

- **ST_Curve** Returns the *n*th curve in the compound curve.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *compoundcurve-expression*.

Standards and compatibility

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

Example

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
```

ST_NumCurves method

Returns the number of curves defining the compound curve.

Note
By default, ST_NumCurves uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

compoundcurve-expression.**ST_NumCurves**()

Returns

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

Standards and compatibility

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

Example

The following example returns the result `2`.

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

ST_Curve type

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

Direct supertype

- "ST_Geometry type"

Direct subtypes

- "ST_CircularString type"
- "ST_CompoundCurve type"
- "ST_LineString type"

Methods

- Methods of ST_Curve:

ST_CurveToLine	ST_EndPoint	ST_IsClosed	ST_IsRing
ST_Length	ST_StartPoint		

- All methods of "ST_Geometry type" can also be called on a ST_Curve type:

ST_Affine	ST_AsBinary	ST_AsGML	ST_AsGeoJSON
ST_AsKML	ST_AsSVG	ST_AsSVGAggr	ST_AsText
ST_AsWKB	ST_AsWKT	ST_AsXML	ST_Boundary
ST_Contains	ST_ContainsFilter	ST_ConvexHull	ST_ConvexHullAggr
ST_CoordDim	ST_CoveredBy	ST_CoveredByFilter	ST_Covers
ST_CoversFilter	ST_Crosses	ST_Difference	ST_Dimension
ST_Disjoint	ST_Distance	ST_Envelope	ST_EnvelopeAggr
ST_Equals	ST_EqualsFilter	ST_GeomFromBinary	ST_GeomFromShape
ST_GeomFromText	ST_GeomFromWKB	ST_GeomFromWKT	ST_GeometryType
ST_GeometryType-FromBaseType	ST_Intersection	ST_IntersectionAggr	ST_Intersects
ST_IntersectsFilter	ST_IntersectsRect	ST_Is3D	ST_IsEmpty

ST_IsMeasured	ST_IsSimple	ST_IsValid	ST_LatNorth
ST_LatSouth	ST_LinearHash	ST_LinearUnHash	ST_LoadConfigura-tionData
ST_LongEast	ST_LongWest	ST_MMax	ST_MMin
ST_OrderingEquals	ST_Overlaps	ST_Relate	ST_Reverse
ST_SRID	ST_SRIDFromBase-Type	ST_SnapToGrid	ST_SymDifference
ST_ToCircular	ST_ToCompound	ST_ToCurve	ST_ToCurvePoly
ST_ToGeomColl	ST_ToLineString	ST_ToMultiCurve	ST_ToMultiLine
ST_ToMultiPoint	ST_ToMultiPolygon	ST_ToMultiSurface	ST_ToPoint
ST_ToPolygon	ST_ToSurface	ST_Touches	ST_Transform
ST_Union	ST_UnionAggr	ST_Within	ST_WithinDistance
ST_WithinDistanceFil-ter	ST_WithinFilter	ST_XMax	ST_XMin
ST_YMax	ST_YMin	ST_ZMax	ST_ZMin

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*.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *curve-expression*.

Remarks

If *curve-expression* is empty, the ST_CurveToLine method returns an empty set of type ST_LineString. Otherwise, ST_CurveToLine returns a linestring containing any linestrings in *curve-expression* combined with the interpolation of any circularstrings in *curve-expression*.

Note
By default, ST_CurveToLine uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_ToLineString method for type ST_Geometry"
- "How interpolation impacts spatial calculations"

Standards and compatibility

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

Example

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
```

ST_EndPoint method

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

Note
By default, ST_EndPoint uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

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.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *curve-expression*.

See also

- "ST_StartPoint method for type ST_Curve"

Standards and compatibility

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

Example

The following example returns the result `Point (5 10)`.

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

ST_IsClosed method

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

Note
By default, ST_IsClosed uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

curve-expression.**ST_IsClosed**()

Returns

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

See also

- “ST_IsClosed method for type ST_MultiCurve”
- “ST_IsRing method for type ST_Curve”
- “ST_Boundary method for type ST_Geometry”

Standards and compatibility

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

Example

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
```

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.

See also

- “ST_IsClosed method for type ST_Curve”
- “ST_IsSimple method for type ST_Geometry”

Standards and compatibility

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

Example

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
```

ST_Length 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**([*unit-name*])

Parameters

Name	Type	Description
unit-name	VAR-CHAR(128)	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.

Remarks

The ST_Length method returns the length of a curve in the units identified by the *unit-name* parameter. If the curve is empty, then NULL is returned.

If the curve contains Z values, these are not considered when computing the length of the geometry.

Note
If the *curve-expression* is an empty geometry (ST_IsEmpty()=1), then this method returns NULL.

Note
By default, ST_Length uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_Length method for type ST_MultiCurve"
- "ST_Area method for type ST_Surface"

Standards and compatibility

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

Example

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
```

ST_StartPoint method

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

Note
By default, ST_StartPoint uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

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.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *curve-expression*.

See also

- "ST_EndPoint method for type ST_Curve"

Standards and compatibility

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

Example

The following example returns the result `Point (0 0)`.

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

ST_CurvePolygon type

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

Direct supertype

- "ST_Surface type"

Direct subtypes

- "ST_Polygon type"

Constructor

- "ST_CurvePolygon constructor"

Methods

- Methods of ST_CurvePolygon:

ST_CurvePolyToPoly	ST_ExteriorRing	ST_InteriorRingN	ST_NumInteriorRing

- All methods of "ST_Surface type" can also be called on a ST_CurvePolygon type:

ST_Area	ST_Centroid	ST_IsWorld	ST_Perimeter
ST_PointOnSurface			

- All methods of "ST_Geometry type" can also be called on a ST_CurvePolygon type:

ST_Affine	ST_AsBinary	ST_AsGML	ST_AsGeoJSON
ST_AsKML	ST_AsSVG	ST_AsSVGAggr	ST_AsText
ST_AsWKB	ST_AsWKT	ST_AsXML	ST_Boundary
ST_Contains	ST_ContainsFilter	ST_ConvexHull	ST_ConvexHullAggr
ST_CoordDim	ST_CoveredBy	ST_CoveredByFilter	ST_Covers
ST_CoversFilter	ST_Crosses	ST_Difference	ST_Dimension
ST_Disjoint	ST_Distance	ST_Envelope	ST_EnvelopeAggr
ST_Equals	ST_EqualsFilter	ST_GeomFromBinary	ST_GeomFromShape
ST_GeomFromText	ST_GeomFromWKB	ST_GeomFromWKT	ST_GeometryType
ST_GeometryType-FromBaseType	ST_Intersection	ST_IntersectionAggr	ST_Intersects
ST_IntersectsFilter	ST_IntersectsRect	ST_Is3D	ST_IsEmpty
ST_IsMeasured	ST_IsSimple	ST_IsValid	ST_LatNorth
ST_LatSouth	ST_LinearHash	ST_LinearUnHash	ST_LoadConfigura-tionData
ST_LongEast	ST_LongWest	ST_MMax	ST_MMin
ST_OrderingEquals	ST_Overlaps	ST_Relate	ST_Reverse
ST_SRID	ST_SRIDFromBase-Type	ST_SnapToGrid	ST_SymDifference
ST_ToCircular	ST_ToCompound	ST_ToCurve	ST_ToCurvePoly
ST_ToGeomColl	ST_ToLineString	ST_ToMultiCurve	ST_ToMultiLine
ST_ToMultiPoint	ST_ToMultiPolygon	ST_ToMultiSurface	ST_ToPoint
ST_ToPolygon	ST_ToSurface	ST_Touches	ST_Transform
ST_Union	ST_UnionAggr	ST_Within	ST_WithinDistance
ST_WithinDistanceFil-ter	ST_WithinFilter	ST_XMax	ST_XMin
ST_YMax	ST_YMin	ST_ZMax	ST_ZMin

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 constructor

Constructs a curve polygon.

Overload list

Name	Description
"ST_CurvePolygon() constructor"	Constructs a curve polygon representing the empty set.
"ST_CurvePolygon(LONG VARCHAR[, INT]) constructor"	Constructs a curve polygon from a text representation.
"ST_CurvePolygon(LONG BINARY[, INT]) constructor"	Constructs a curve polygon from Well Known Binary (WKB).
"ST_CurvePolygon(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.
"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.

ST_CurvePolygon() constructor

Constructs a curve polygon representing the empty set.

Syntax

NEW ST_CurvePolygon()

Standards and compatibility

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

Example

The following returns 1, indicating the value is empty.

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

ST_CurvePolygon(LONG VARCHAR[, INT]) constructor

Constructs a curve polygon from a text representation.

Syntax

NEW ST_CurvePolygon(*text-representation*[, *srid*])

Parameters

Name	Type	Description
text-representation	LONG VARCHAR	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	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a curve polygon from a character string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

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

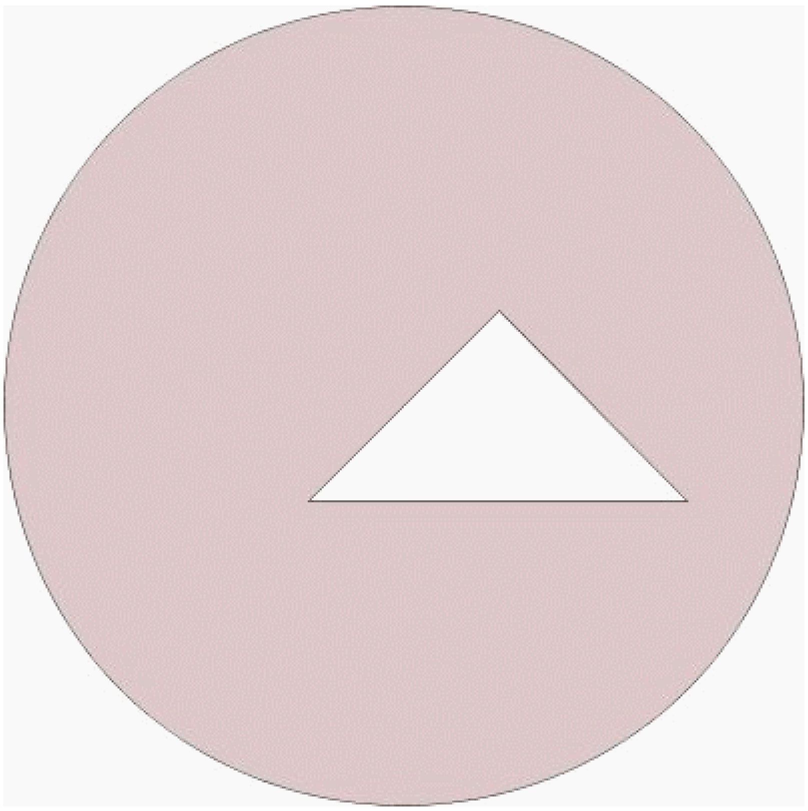
Example

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
```



ST_CurvePolygon(LONG BINARY[, INT]) constructor

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

Syntax

NEW ST_CurvePolygon(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BI-NARY	A string containing the binary representation of an curve polygon. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
srid	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a curve polygon from a binary string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

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

Example

The following returns CurvePolygon (CircularString (0 0, 10 0, 10 10, 0 10, 0 0)).

```
SELECT NEW
ST_CurvePolygon(0x010a000000010000000108000000050000000000000000000000000
000000000000000000000244000000000000000000000000000024400000000000000024400000
000000000000000000000024400000000000000000000000000000000000)
```

ST_CurvePolygon(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(*exterior-ring*[,*interior-ring1*,...,*interior-ringN*])

Parameters

Name	**Type**	**Description**
exterior-ring	ST_Curve	The exterior ring of the curve polygon
interior-ring1,...,interior-ringN	ST_Curve	Interior rings of the curve polygon

Remarks

Creates a curve polygon from a curve representing the exterior ring and a list (possibly empty) of curves representing interior rings. All of the specified rings must have the same SRID. The polygon is created with this common SRID. All of the supplied rings must be non-empty and have the same answer for Is3D and IsMeasured. The polygon is 3D if all of the points are 3D, and the polygon is measured if all of the points are measured.

Note
By default, ST_CurvePolygon uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

The ability to specify a varying length list of interior rings is a vendor extension.

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

Example

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)'))
```

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(*multi-curve*[, *polygon-format*])

Parameters

Name	Type	Description
multi-curve	ST_MultiCurve	A multicurve value containing an exterior ring and (optionally) a set of interior rings.
polygon-format	VARCHAR(128)	A string with the polygon format to use when interpreting the provided curves. Valid formats are 'CounterClockwise', 'Clockwise', and 'EvenOdd'

Remarks

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

If specified, the *polygon-format* parameter selects the algorithm the server uses to determine whether a ring is an exterior or interior ring. If not specified, the polygon format of the spatial reference system is used.

For additional information on *polygon-format*, see "POLYGON FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Note
By default, ST_CurvePolygon uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

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 ))' ) )
```

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.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *curvepolygon-expression*.

Remarks

If *curvepolygon-expression* is empty, the ST_CurvePolyToPoly method returns an empty set of type ST_Polygon. Otherwise, ST_CurvePolyToPoly returns a polygon containing any linear rings in *curvepolygon-expression* combined with the interpolation of any circularstring or compound curve rings in *curvepolygon-expression*.

Note
By default, ST_CurvePolyToPoly uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_ToPolygon method for type ST_Geometry"
- "How interpolation impacts spatial calculations"

Standards and compatibility

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

Example

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
```

ST_ExteriorRing method

Retrieves or modifies the exterior ring.

Overload list

Name	Description
"ST_ExteriorRing() method for type ST_CurvePolygon"	Returns the exterior ring of the curve polygon.
"ST_ExteriorRing(ST_Curve) method for type ST_CurvePolygon"	Changes the exterior ring of the curve polygon.

ST_ExteriorRing() method for type ST_CurvePolygon

Returns the exterior ring of the curve polygon.

Note
By default, ST_ExteriorRing uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

curvepolygon-expression.**ST_ExteriorRing**()

Returns

- **ST_Curve** Returns the exterior ring.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *curvepolygon-expression*.

See also

- "ST_InteriorRingN method for type ST_CurvePolygon"
- "ST_ExteriorRing method for type ST_Polygon"
- "ST_Boundary method for type ST_Geometry"

Standards and compatibility

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

Example

The following example returns the result `CircularString (2 0, 5 0, 5 3, 2 3, 2 0)`.

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

ST_ExteriorRing(ST_Curve) method for type ST_CurvePolygon

Changes the exterior ring of the curve polygon.

Note
By default, ST_ExteriorRing uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

curvepolygon-expression.**ST_ExteriorRing**(*exterior-ring*)

Parameters

Name	Type	Description
exterior-ring	ST_Curve	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.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *curvepolygon-expression*.

Standards and compatibility

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

Example

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)' ) )
```

ST_InteriorRingN method

Returns the *n*th interior ring in the curve polygon.

Note
By default, ST_InteriorRingN uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

curvepolygon-expression.**ST_InteriorRingN**(*n*)

Parameters

Name	Type	Description
n	INT	The position of the element to return, from 1 to *curvepolygon-expression*.ST_NumInteriorRing().

Returns

- **ST_Curve**   Returns the *n*th interior ring in the curve polygon.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *curvepolygon-expression*.

See also

- "ST_NumInteriorRing method for type ST_CurvePolygon"
- "ST_ExteriorRing method for type ST_CurvePolygon"
- "ST_InteriorRingN method for type ST_Polygon"
- "ST_Boundary method for type ST_Geometry"

Standards and compatibility

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

Example

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 )
```

ST_NumInteriorRing method

Returns the number of interior rings in the curve polygon.

Note
By default, ST_NumInteriorRing uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

curvepolygon-expression.**ST_NumInteriorRing**()

Returns

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

See also

- "ST_InteriorRingN method for type ST_CurvePolygon"
- "ST_InteriorRingN method for type ST_Polygon"

Standards and compatibility

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

Example

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()
```

ST_GeomCollection type

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

Direct supertype

- "ST_Geometry type"

Direct subtypes

- "ST_MultiCurve type"
- "ST_MultiPoint type"
- "ST_MultiSurface type"

Constructor

- "ST_GeomCollection constructor"

Methods

- Methods of ST_GeomCollection:

ST_GeomCollectionAggr	ST_GeometryN	ST_NumGeometries

- All methods of "ST_Geometry type" can also be called on a ST_GeomCollection type:

ST_Affine	ST_AsBinary	ST_AsGML	ST_AsGeoJSON
ST_AsKML	ST_AsSVG	ST_AsSVGAggr	ST_AsText
ST_AsWKB	ST_AsWKT	ST_AsXML	ST_Boundary
ST_Contains	ST_ContainsFilter	ST_ConvexHull	ST_ConvexHullAggr
ST_CoordDim	ST_CoveredBy	ST_CoveredByFilter	ST_Covers
ST_CoversFilter	ST_Crosses	ST_Difference	ST_Dimension
ST_Disjoint	ST_Distance	ST_Envelope	ST_EnvelopeAggr
ST_Equals	ST_EqualsFilter	ST_GeomFromBinary	ST_GeomFromShape
ST_GeomFromText	ST_GeomFromWKB	ST_GeomFromWKT	ST_GeometryType
ST_GeometryType-FromBaseType	ST_Intersection	ST_IntersectionAggr	ST_Intersects
ST_IntersectsFilter	ST_IntersectsRect	ST_Is3D	ST_IsEmpty
ST_IsMeasured	ST_IsSimple	ST_IsValid	ST_LatNorth
ST_LatSouth	ST_LinearHash	ST_LinearUnHash	ST_LoadConfigura-tionData
ST_LongEast	ST_LongWest	ST_MMax	ST_MMin
ST_OrderingEquals	ST_Overlaps	ST_Relate	ST_Reverse

ST_SRID	ST_SRIDFromBase-Type	ST_SnapToGrid	ST_SymDifference
ST_ToCircular	ST_ToCompound	ST_ToCurve	ST_ToCurvePoly
ST_ToGeomColl	ST_ToLineString	ST_ToMultiCurve	ST_ToMultiLine
ST_ToMultiPoint	ST_ToMultiPolygon	ST_ToMultiSurface	ST_ToPoint
ST_ToPolygon	ST_ToSurface	ST_Touches	ST_Transform
ST_Union	ST_UnionAggr	ST_Within	ST_WithinDistance
ST_WithinDistanceFil-ter	ST_WithinFilter	ST_XMax	ST_XMin
ST_YMax	ST_YMin	ST_ZMax	ST_ZMin

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 constructor

Constructs a geometry collection.

Overload list

Name	Description
"ST_GeomCollection() constructor"	Constructs a geometry collection representing the empty set.
"ST_GeomCollection(LONG VAR-CHAR[, INT]) constructor"	Constructs a geometry collection from a text representa-tion.

Name	Description
"ST_GeomCollection(LONG BINARY[, INT]) constructor"	Constructs a geometry collection from Well Known Binary (WKB).
"ST_GeomCollection(ST_Geometry,...) constructor"	Constructs a geometry collection from a list of geometry values.

ST_GeomCollection() constructor

Constructs a geometry collection representing the empty set.

Syntax

NEW ST_GeomCollection()

Standards and compatibility

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

Example

The following returns 1, indicating the value is empty.

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

ST_GeomCollection(LONG VARCHAR[, INT]) constructor

Constructs a geometry collection from a text representation.

Syntax

NEW ST_GeomCollection(*text-representation*[, *srid*])

Parameters

Name	Type	Description
text-representation	LONG VARCHAR	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	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a geometry collection from a character string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

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

Example

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)))')
```

ST_GeomCollection(LONG BINARY[, INT]) constructor

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

Syntax

NEW ST_GeomCollection(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	A string containing the binary representation of an geometry collection. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
srid	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a geometry collection from a binary string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

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

Example

The following returns GeometryCollection (Point (10 20)).

```
SELECT NEW
ST_GeomCollection(0x010700000001000000010100000000000000000000244000000000000000003
440)
```

ST_GeomCollection(ST_Geometry,...) constructor

Constructs a geometry collection from a list of geometry values.

Syntax

NEW ST_GeomCollection(*geo1*[,*geo2*,...,*geoN*])

Parameters

Name	Type	Description
geo1	ST_Geometry	The first geometry value of the geometry collection.
geo2,...,geoN	ST_Geometry	Additional geometry values of the geometry collection.

Remarks

Constructs a geometry collection from a list of geometry values. All of the supplied geometry values must have the same SRID, and the geometry collection is constructed with this common SRID.

All of the supplied geometry values must have the same answer for Is3D and IsMeasured. The geometry collection is 3D if all of the geometry values are 3D, and the geometry collection is measured if all of the geometry values are measured.

Note
By default, ST_GeomCollection uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

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 ) )
```

ST_GeomCollectionAggr method

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

Syntax

ST_GeomCollection::**ST_GeomCollectionAggr**(*geometry-column*[**ORDER BY** *order-by-expression* [**ASC** | **DESC**], ...])

Parameters

Name	Type	Description
geometry-column	ST_Geometry	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.

  The spatial reference system identifier of the result is the same as that for the first parameter.

Remarks

The ST_GeomCollectionAggr aggregate function can be used to combine a group of geometries into a single collection. All of the geometries to be combined must have both the same SRID and the same coordinate dimension.

Rows where the argument is NULL are not included.

Returns NULL for an empty group or a group containing no non-NULL values.

The resulting ST_GeomCollection has the same coordinate dimension as each geometries.

The optional ORDER BY clause can be used to arrange the elements in a particular order so that ST_GeometryN returns them in the desired order. If this ordering is not relevant, it is more efficient to not specify an ordering. In that case, the ordering of elements depends on the access plan selected by the query optimizer.

ST_GeomCollectionAggr is more efficient than ST_UnionAggr, but ST_GeomCollectionAggr can return a collection with duplicate or overlapping geometries if they exist in the group of geometries. In particular, returned collections containing overlapping surfaces may case unexpected results if they are used as input to other spatial methods. ST_UnionAggr handles duplicate and overlapping geometries.

Note
By default, ST_GeomCollectionAggr uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_UnionAggr method for type ST_Geometry"

Standards and compatibility

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

Example

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
```

ST_GeometryN method

Returns the *n*th geometry in the geometry collection.

Note
By default, ST_GeometryN uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

geomcollection-expression.**ST_GeometryN**(*n*)

Parameters

Name	Type	Description
n	INT	The position of the element to return, from 1 to *geomcollection-expression*.ST_NumGeometries().

Returns

- **ST_Geometry** Returns the *n*th geometry in the geometry collection.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *geomcollection-expression*.

See also

- "ST_NumGeometries method for type ST_GeomCollection"

Standards and compatibility

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

Example

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 )
```

ST_NumGeometries method

Returns the number of geometries contained in the geometry collection.

Note
By default, ST_NumGeometries uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see “STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement” [*SQL Anywhere Server - SQL Reference*].

Syntax

geomcollection-expression.**ST_NumGeometries**()

Returns

- **INT** Returns the number of geometries stored in this collection.

See also

- “ST_GeometryN method for type ST_GeomCollection”

Standards and compatibility

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

Example

The following example returns the result 3.

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

ST_Geometry type

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

Direct subtypes

- “ST_Curve type”
- “ST_GeomCollection type”
- “ST_Point type”
- “ST_Surface type”

Methods

- Methods of ST_Geometry:

ST_Affine	ST_AsBinary	ST_AsGML	ST_AsGeoJSON
ST_AsKML	ST_AsSVG	ST_AsSVGAggr	ST_AsText
ST_AsWKB	ST_AsWKT	ST_AsXML	ST_Boundary
ST_Contains	ST_ContainsFilter	ST_ConvexHull	ST_ConvexHullAggr

ST_CoordDim	ST_CoveredBy	ST_CoveredByFilter	ST_Covers
ST_CoversFilter	ST_Crosses	ST_Difference	ST_Dimension
ST_Disjoint	ST_Distance	ST_Envelope	ST_EnvelopeAggr
ST_Equals	ST_EqualsFilter	ST_GeomFromBinary	ST_GeomFromShape
ST_GeomFromText	ST_GeomFromWKB	ST_GeomFromWKT	ST_GeometryType
ST_GeometryType-FromBaseType	ST_Intersection	ST_IntersectionAggr	ST_Intersects
ST_IntersectsFilter	ST_IntersectsRect	ST_Is3D	ST_IsEmpty
ST_IsMeasured	ST_IsSimple	ST_IsValid	ST_LatNorth
ST_LatSouth	ST_LinearHash	ST_LinearUnHash	ST_LoadConfigura-tionData
ST_LongEast	ST_LongWest	ST_MMax	ST_MMin
ST_OrderingEquals	ST_Overlaps	ST_Relate	ST_Reverse
ST_SRID	ST_SRIDFromBase-Type	ST_SnapToGrid	ST_SymDifference
ST_ToCircular	ST_ToCompound	ST_ToCurve	ST_ToCurvePoly
ST_ToGeomColl	ST_ToLineString	ST_ToMultiCurve	ST_ToMultiLine
ST_ToMultiPoint	ST_ToMultiPolygon	ST_ToMultiSurface	ST_ToPoint
ST_ToPolygon	ST_ToSurface	ST_Touches	ST_Transform
ST_Union	ST_UnionAggr	ST_Within	ST_WithinDistance
ST_WithinDistanceFil-ter	ST_WithinFilter	ST_XMax	ST_XMin
ST_YMax	ST_YMin	ST_ZMax	ST_ZMin

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 method

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

Syntax

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

Parameters

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

Returns

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

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

Remarks

An affine transformation combines rotation, translation and scaling into a single method call. The affine transform is defined using matrix multiplication.

For a point (x,y,z), the result (x',y',z') is computed as follows:

```
 / x' \ / a00 a01 a02 xoff \ / x \ | y' | = | a10 a11 a12 yoff | * | y | |
z' | | a20 a21 a22 zoff | | z | \ w' / \ 0 0 0 1 / \ 1 /
```

Note
This method can not be used with geometries in round-Earth spatial reference system.

Standards and compatibility

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

Example

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
```

ST_AsBinary method

Returns the WKB representation of an ST_Geometry value.

Syntax

geometry-expression.**ST_AsBinary**([*format*])

Parameters

Name	Type	Description
format	VARCHAR(128)	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" [*SQL Anywhere Server - Database Administration*].

Returns

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

Remarks

The ST_AsBinary method returns a binary string representing the geometry. A number of different binary formats are supported (with associated options) and the desired format is selected using the optional *format* parameter. If the *format* parameter is not specified, the st_geometry_asbinary_format option is used to select the output format to use. See "st_geometry_asbinary_format option" [*SQL Anywhere Server - Database Administration*].

The format string defines an output format and parameters to the format. The format string has one of the following formats:

```
format-name

format-name(parameter1=value1;parameter2=value2;...)

parameter1=value1;parameter2=value2;...
```

The first format specifies the format name and no parameters. All format parameters use their default values. The second format specifies the format name and a list of named parameter values. Parameters that are not supplied use their default values. The last format specifies only parameter values, and the format name defaults to 'WKB'.

The following format names may be used:

- **WKB** The Well-Known Binary format defined by SQL/MM and the OGC.
- **EWKB** The Extended Well-Known Binary format defined by PostGIS. This format includes the geometry's SRID and it differs from WKB in the way it represents Z and M values.

The following format parameters can be specified:

Format Name	Parameter Name	Default Value	Allowed Values	Description
WKB	Version	1.2	• **1.1** The WKB defined by OGC SFS 1.1. This format does not contain Z and M values. If the geometry contains Z or M values, they are removed in the output. • **1.2** The WKB defined by OGC SFS 1.2. This matches version 1.1 on 2D data and extends the format to support Z and M values.	The version parameter controls the version of the WKB specification used.

Note
When converting a geometry value to BINARY, the server uses the ST_AsBinary method. The st_geometry_asbinary_format option defines the format that is used for the conversion. See "st_geometry_asbinary_format option" [*SQL Anywhere Server - Database Administration*].

Note
By default, ST_AsBinary uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

If the st_geometry_asbinary_format option has its default value of 'WKB', the following returns the result `0x01b90b0000000000000000f03f000000000000004000000000000008400000000000 001040`.

```
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 `0x01b90b0000000000000000f03f000000000000004000000000000008400000000000 001040`. 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)')
```

ST_AsGML method

Returns the GML representation of an ST_Geometry value.

Syntax

geometry-expression.**ST_AsGML**([*format*])

Parameters

Name	Type	Description
format	VARCHAR(128)	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*.

Remarks

The ST_AsGML method returns a GML string representing the geometry. A number of different formats are supported (with associated options) and the desired format is selected using the optional *format* parameter. If the *format* parameter is not specified, the default is 'GML'.

The format string defines an output format and parameters to the format. The format string has one of the following formats:

```
format-name

format-name(parameter1=value1;parameter2=value2;...)

parameter1=value1;parameter2=value2;...
```

The first format specifies the format name and no parameters. All format parameters use their default values. The second format specifies the format name and a list of named parameter values. Parameters that are not supplied use their default values. The last format specifies only parameter values, and the format name defaults to 'GML'.

The following format names may be used:

- **GML** The Geography Markup Language format defined by ISO 19136 and the OGC.

The following format parameters can be specified:

Format Name	Parameter Name	Default Value	Allowed Values	Description
GML	Namespace	none	● **local** Provides a default namespace attribute for the given element (in this case Point) and its sub elements. ● **global** 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. ● **none** Provides no namespace or prefix for the given element (in this case Point) and its sub elements	The namespace parameter specifies the output format convention for namespace.
GML	SRSName-Format	short	● **short** Uses a short format for the spatial reference system name, for example EPSG:4326 ● **long** Uses a long format for the spatial reference system name, for example urn:x-ogc:def:crs:EPSG:4326. ● **none** Spatial reference system name attribute is not included for the geometry.	The SRSNameFormat parameter specifies the format for the srsName attribute.
GML	SRSDimen-sion	No	**Yes** or **No**	The SRSDimension parameter specifies the number of coordinate values for the given geometry. This only applies to GML(version=3).

Format Name	Parameter Name	Default Value	Allowed Values	Description
GML	SRSFillAll	No	**Yes** or **No**	The SRSFillAll parameter specifies whether or not SRS attributes should be propagated to child geometry elements. As an example a MultiGeometry or MultiPolygon would propagate the attributes to its child geometries.
GML	UseDeprecated	No	**Yes** or **No**	The UseDeprecated parameter only applies to GML(version=3). It is used to output older GML representations where possible. As an example a Surface may be output as a Polygon if the geometry contains no CircularStrings.
GML	Attribute	Automatically generated optional attributes	One or more attributes may be specified for the top level geometry element only	Any legal XML attributes may be specified.
GML	SubElement	Automatically generated GML sub elements	One or more sub elements may be specified for the top level geometry element only	Any legal XML elements may be specified.

Note
By default, ST_AsGML uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

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)')
```

ST_AsGeoJSON method

Returns a string representing a geometry in JSON format.

Syntax

geometry-expression.**ST_AsGeoJSON**([*format*])

Parameters

Name	Type	Description
format	VAR-CHAR(128)	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*.

Remarks

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

The ST_AsGeoJSON method returns a text string representing the geometry. A number of different text formats are supported (with associated options) and the desired format is selected using the optional *format* parameter. If the *format* parameter is not specified, the default is 'GeoJSON'.

The format string defines an output format and parameters to the format. The format string has one of the following formats:

```
format-name
```

```
format-name(parameter1=value1;parameter2=value2;...)
```

```
parameter1=value1;parameter2=value2;...
```

The first format specifies the format name and no parameters. All format parameters use their default values. The second format specifies the format name and a list of named parameter values. Parameters that are not supplied use their default values. The last format specifies only parameter values, and the format name defaults to 'GeoJSON'.

The following format names may be used:

- **GeoJSON** The GeoJSON format uses JavaScript Object Notation (JSON) as defined by http://geojson.org/geojson-spec.html.

Format Name	Parameter Name	Default Value	Allowed Values	Description
GeoJSON	Version	1.0	1.0	The version of the GeoJSON specification to follow. At present, only 1.0 is supported.

Note
By default, ST_AsGeoJSON uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

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()
```

ST_AsKML method

Returns the KML representation of an ST_Geometry value.

Syntax

geometry-expression.**ST_AsKML**([*format*])

Parameters

Name	Type	Description
format	VARCHAR(128)	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*.

Remarks

The ST_AsKML method returns a KML string representing the geometry. A number of different formats are supported (with associated options) and the desired format is selected using the optional *format* parameter. If the *format* parameter is not specified, the default is 'KML'.

The format string defines an output format and parameters to the format. The format string has one of the following formats:

```
format-name
```

```
format-name(parameter1=value1;parameter2=value2;...)
```

```
parameter1=value1;parameter2=value2;...
```

The first format specifies the format name and no parameters. All format parameters use their default values. The second format specifies the format name and a list of named parameter values. Parameters that are not supplied use their default values. The last format specifies only parameter values, and the format name defaults to 'KML'.

The following format names may be used:

- **KML** The Keyhole Markup Language format defined by the OGC.

The following format parameters can be specified:

Format Name	Parameter Name	Default Value	Allowed Values	Description
KML	Version	2	2	KML version 2.2 is supported.
KML	Attribute	Automatically generated optional attributes	One or more attributes may be specified for the top level geometry element only	Any legal XML attributes may be specified.
KML	Namespace	none	• **local** Provides the default namespace attribute **http://www.opengis.net/kml/2.2** for the given geometry element (in this case Point) and its sub elements. • **global** 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. • **none** Provides no namespace or prefix for the given element (in this case Point) and its sub elements	The namespace parameter specifies the output format convention for namespace.
KML	SubElement	Automatically generated KML sub elements	One or more sub elements may be specified for the top level geometry element only	Any legal XML elements may be specified. As an example extrude, tessellate and altitudeMode elements may be specified.

Note
By default, ST_AsKML uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

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>')
```

ST_AsSVG method

Returns an SVG figure representing a geometry value.

Syntax

geometry-expression.**ST_AsSVG**([*format*])

Parameters

Name	Type	Description
format	VAR-CHAR(128)	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*.

Remarks

The ST_AsSVG method returns a complete or partial SVG document that can be used to graphically display geometries using an SVG viewer. Most major web browsers with the exception of Microsoft Internet Explorer include built-in SVG viewers.

A number of different options are supported and the desired format is selected using the optional *format* parameter. If the *format* parameter is not specified, the default is 'SVG'.

The format string defines an output format and parameters to the format. The format string has one of the following formats:

```
format-name
```

```
format-name(parameter1=value1;parameter2=value2;...)
```

```
parameter1=value1;parameter2=value2;...
```

The first format specifies the format name and no parameters. All format parameters use their default values. The second format specifies the format name and a list of named parameter values. Parameters that are not supplied use their default values. The last format specifies only parameter values, and the format name defaults to 'SVG'.

The following format names may be used:

- **SVG** The Scalable Vector Graphics (SVG) 1.1 format defined by the World Wide Web Consortium (W3C).

The following format parameters can be specified:

Format Name	Parameter Name	Default Value	Allowed Values	Description
SVG	Approximate	Yes	**Yes** or **No**	The Approximate parameter specifies whether or not to reduce the size of the output SVG document with a slight reduction in visible detail. The SVG data is approximated by not including points which are within the line width of the last point. With multiple megabyte geometries this can result in compression rates of 80% or more.
SVG	Attribute	Automatically generated optional attributes	One or more SVG attributes that can be applied to SVG shape elements	By default, optional SVG shape attributes such as fill, stroke and stroke-width are generated. If the Attributes parameter is specified, then no optional SVG shape attributes are generated, and the Attribute value is used instead. Ignored if PathDataOnly=Yes is specified. The maximum length of the Attribute value is about 1000 bytes.
SVG	DecimalDigits	Based on the number of decimal digits in the spatial reference system's snap to grid gridsize. The maximum default value is 5 and the minimum is 0.	integer	The DecimalDigits parameter limits the number of digits after the decimal place for coordinates generated in the SVG output. Specifying a negative number of digits indicates that the full precision of coordinates should be included in the SVG output.
SVG	PathDataOnly	No (a complete SVG document is generated)	**Yes** or **No**	The PathDataOnly parameter specifies whether or not only data for the SVG Path Element should be generated. The PathDataOnly example below demonstrates how PathDataOnly=Yes can be used to build a complete SVG document that can be displayed. By default a complete SVG document is generated. The path data returned by PathDataOnly=Yes can be used to build more flexible SVG documents containing other elements, such as text.

Format Name	Parameter Name	Default Value	Allowed Values	Description
SVG	RandomFill	Yes	**Yes** or **No**	The RandomFill parameter specifies whether or not polygons should be filled by a randomly generated color. The sequence of colors used does not follow a well-defined sequence, and typically changes each time SVG output is generated. **No** indicates that only an outline of each polygon is drawn. The RandomFill parameter is ignored if the Attribute or PathDataOnly=Yes parameter is specified.
SVG	Relative	Yes	**Yes** or **No**	The Relative parameter specifies if coordinates should be output in relative (offset) or absolute formats. Relative coordinate data is typically more compact than absolute coordinate data.

Note
By default, ST_AsSVG uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see “STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement” [*SQL Anywhere Server - SQL Reference*].

See also

- “ST_AsSVGAggr method for type ST_Geometry”

Standards and compatibility

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

Example

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

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

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' )
```

ST_AsSVGAggr method

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

Syntax

ST_Geometry::**ST_AsSVGAggr**(*geometry-column*[**ORDER BY** *order-by-expression* [**ASC** | **DESC**], ...] [, *format*])

Parameters

Name	Type	Description
geometry-column	ST_Geometry	The geometry value to contribute to the SVG figure. Typically this is a column.
format	VARCHAR(128)	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.

Remarks

The ST_AsSVGAggr method returns a complete or partial SVG document that can be used to graphically display the union of a group of geometries using an SVG viewer. Most major web browsers with the exception of Microsoft Internet Explorer include built-in SVG viewers.

A number of different options are supported and the desired format is selected using the optional *format* parameter. If the *format* parameter is not specified, the default is 'SVG'.

The format string defines an output format and parameters to the format. The format string has one of the following formats:

```
format-name

format-name(parameter1=value1;parameter2=value2;...)

parameter1=value1;parameter2=value2;...
```

The first format specifies the format name and no parameters. All format parameters use their default values. The second format specifies the format name and a list of named parameter values. Parameters that are not supplied use their default values. The last format specifies only parameter values, and the format name defaults to 'SVG'.

The following format names may be used:

- **SVG** The Scalable Vector Graphics (SVG) 1.1 format defined by the World Wide Web Consortium (W3C).

The following format parameters can be specified:

Format Name	Parameter Name	Default Value	Allowed Values	Description
SVG	Approximate	Yes	**Yes** or **No**	The Approximate parameter specifies whether or not to reduce the size of the output SVG document with a slight reduction in visible detail. The SVG data is approximated by not including points which are within the line width of the last point. With multiple megabyte geometries this can result in compression rates of 80% or more.
SVG	Attribute	Automatically generated optional attributes	One or more SVG attributes that can be applied to SVG shape elements	By default, optional SVG shape attributes such as fill, stroke and stroke-width are generated. If the Attributes parameter is specified, then no optional SVG shape attributes are generated, and the Attribute value is used instead. Ignored if PathDataOnly=Yes is specified.

Format Name	Parameter Name	Default Value	Allowed Values	Description
SVG	DecimalDigits	Based on the number of decimal digits in the spatial reference system's snap to grid gridsize. The maximum default value is 5 and the minimum is 0.	integer	The DecimalDigits parameter limits the number of digits after the decimal place for coordinates generated in the SVG output. Specifying a negative number of digits indicates that the full precision of coordinates should be included in the SVG output.
SVG	PathDataOnly	No (a complete SVG document is generated)	**Yes** or **No**	The PathDataOnly parameter specifies whether or not only data for the SVG Path Element should be generated. The PathDataOnly example below demonstrates how PathDataOnly=Yes can be used to build a complete SVG document that can be displayed. By default a complete SVG document is generated. The path data returned by PathDataOnly=Yes can be used to build more flexible SVG documents containing other elements, such as text.
SVG	RandomFill	Yes	**Yes** or **No**	The RandomFill parameter specifies whether or not polygons should be filled by a randomly generated color. The sequence of colors used does not follow a well-defined sequence, and typically changes each time SVG output is generated. **No** indicates that only an outline of each polygon is drawn. The RandomFill parameter is ignored if the Attribute or PathDataOnly=Yes parameter is specified.
SVG	Relative	Yes	**Yes** or **No**	The Relative parameter specifies if coordinates should be output in relative (offset) or absolute formats. Relative coordinate data is typically more compact than absolute coordinate data.

The ORDER BY clause can be specified to control how overlapping geometries are displayed, with geometries displayed in order from back to front. If not specified, the geometries are displayed in an order that depends on the execution plan selected by the query optimizer, and this may vary between executions.

Note
By default, ST_AsSVGAggr uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_AsSVG method for type ST_Geometry"

Standards and compatibility

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

Example

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;
```

ST_AsText method

Returns the text representation of an ST_Geometry value.

Syntax

geometry-expression.**ST_AsText**([*format*])

Parameters

Name	Type	Description
format	VARCHAR(128)	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. See "st_geometry_astext_format option" [*SQL Anywhere Server - Database Administration*].

Returns

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

Remarks

The ST_AsText method returns a text string representing the geometry. A number of different text formats are supported (with associated options) and the desired format is selected using the optional *format* parameter. If the *format* parameter is not specified, the st_geometry_astext_format option is used to select the output format to use. See "st_geometry_astext_format option" [*SQL Anywhere Server - Database Administration*].

The format string defines an output format and parameters to the format. The format string has one of the following formats:

```
format-name
```

```
format-name(parameter1=value1;parameter2=value2;...)
```

```
parameter1=value1;parameter2=value2;...
```

The first format specifies the format name and no parameters. All format parameters use their default values. The second format specifies the format name and a list of named parameter values. Parameters that are not supplied use their default values. The last format specifies only parameter values, and the format name defaults to 'WKT'.

The following format names may be used:

- **WKT** The Well-Known Text format defined by SQL/MM and the OGC.
- **EWKT** The Extended Well Known Text format. This format includes the geometry's SRID as a prefix.
- **GML** The Geography Markup Language format defined by ISO 19136 and the OGC.
- **KML** Keyhole Markup Language format defined by OGC.
- **GeoJSON** The GeoJSON format uses JavaScript Object Notation (JSON) as defined by http://geojson.org/geojson-spec.html.

- **SVG** The Scalable Vector Graphics (SVG) 1.1 format defined by the World Wide Web Consortium (W3C).

The following format parameters can be specified:

Format Name	Parameter Name	Default Value	Allowed Values	Description
WKT	Version	1.2	● **1.1** The WKT defined by OGC SFS 1.1. This format does not contain Z and M values. If the geometry contains Z or M values, they are removed in the output. ● **1.2** The WKT defined by OGC SFS 1.2. This matches version 1.1 on 2D data and extends the format to support Z and M values. ● **PostGIS** The WKT format used by some other vendors; Z and M values are included in a fashion that does not match OGC 1.2.	The version parameter controls the version of the WKT specification used.
GML	Version	3	● **2** Version 2 of the GML specification. ● **3** Version 3.2 of the GML specification	The version parameter controls the version of the GML specification used.

Format Name	Parameter Name	Default Value	Allowed Values	Description
GML	Namespace	none	• **local** Provides a default namespace attribute for the given element (in this case Point) and its sub elements. • **global** Provides a dedica-ted ("gml") prefix for the given element and its sub el-ements. 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. • **none** Provides no name-space or prefix for the given element (in this case Point) and its sub elements	The namespace parameter specifies the output format convention for namespace.
GML	SRSName-Format	short	• **short** Uses a short format for the spatial reference sys-tem name, for example EPSG:4326 • **long** Uses a long format for the spatial reference sys-tem name, for example urn:x-ogc:def:crs:EPSG: 4326. • **none** Spatial reference system name attribute is not included for the geometry.	The SRSNameFormat pa-rameter specifies the format for the srsName attribute.
GML	SRSDimen-sion	No	**Yes** or **No**	The SRSDimension parame-ter specifies the number of coordinate values for the giv-en geometry. This only ap-plies to GML(version=3).

Format Name	Parameter Name	Default Value	Allowed Values	Description
GML	SRSFillAll	No	**Yes** or **No**	The SRSFillAll parameter specifies whether or not SRS attributes should be propagated to child geometry elements. As an example a MultiGeometry or MultiPolygon would propagate the attributes to its child geometries.
GML	UseDeprecated	No	**Yes** or **No**	The UseDeprecated parameter only applies to GML(version=3). It is used to output older GML representations where possible. As an example a Surface may be output as a Polygon if the geometry contains no CircularStrings.
GML	Attribute	Automatically generated optional attributes	One or more attributes may be specified for the top level geometry element only	Any legal XML attributes may be specified.
GML	SubElement	Automatically generated GML sub elements	One or more sub elements may be specified for the top level geometry element only	Any legal XML elements may be specified.
KML	Version	2	2	KML version 2.2 is supported.
KML	Attribute	Automatically generated optional attributes	One or more attributes may be specified for the top level geometry element only	Any legal XML attributes may be specified.

Format Name	Parameter Name	Default Value	Allowed Values	Description
KML	Namespace	none	● **local** Provides the default namespace attribute **http://www.opengis.net/kml/2.2** for the given geometry element (in this case Point) and its sub elements. ● **global** 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. ● **none** Provides no namespace or prefix for the given element (in this case Point) and its sub elements	The namespace parameter specifies the output format convention for namespace.
KML	SubElement	Automatically generated KML sub elements	One or more sub elements may be specified for the top level geometry element only	Any legal XML elements may be specified. As an example extrude, tessellate and altitudeMode elements may be specified.
GeoJSON	Version	1	1	The version of the GeoJSON specification to follow. At present, only 1.0 is supported.

Format Name	Parameter Name	Default Value	Allowed Values	Description
SVG	Approxi-mate	Yes	**Yes** or **No**	The Approximate parameter specifies whether or not to reduce the size of the output SVG document with a slight reduction in visible detail. The SVG data is approximated by not including points which are within the line width of the last point. With multiple megabyte geometries this can result in compression rates of 80% or more.
SVG	Attribute	Automatically generated optional attributes	One or more SVG attributes that can be applied to SVG shape elements	By default, optional SVG shape attributes such as fill, stroke and stroke-width are generated. If the Attributes parameter is specified, then no optional SVG shape attributes are generated, and the Attribute value is used instead. Ignored if PathDataOnly=Yes is specified. The maximum length of the Attribute value is about 1000 bytes.

Format Name	Parameter Name	Default Value	Allowed Values	Description
SVG	DecimalDigits	Based on the number of decimal digits in the spatial reference system's snap to grid gridsize. The maximum default value is 5 and the minimum is 0.	integer	The DecimalDigits parameter limits the number of digits after the decimal place for coordinates generated in the SVG output. Specifying a negative number of digits indicates that the full precision of coordinates should be included in the SVG output.
SVG	PathDataOnly	No (a complete SVG document is generated)	**Yes** or **No**	The PathDataOnly parameter specifies whether or not only data for the SVG Path Element should be generated. The PathDataOnly example below demonstrates how PathDataOnly=Yes can be used to build a complete SVG document that can be displayed. By default a complete SVG document is generated. The path data returned by PathDataOnly=Yes can be used to build more flexible SVG documents containing other elements, such as text.

Format Name	Parameter Name	Default Value	Allowed Values	Description
SVG	RandomFill	Yes	**Yes** or **No**	The RandomFill parameter specifies whether or not polygons should be filled by a randomly generated color. The sequence of colors used does not follow a well-defined sequence, and typically changes each time SVG output is generated. **No** indicates that only an outline of each polygon is drawn. The RandomFill parameter is ignored if the Attribute or PathDataOnly=Yes parameter is specified.
SVG	Relative	Yes	**Yes** or **No**	The Relative parameter specifies if coordinates should be output in relative (offset) or absolute formats. Relative coordinate data is typically more compact than absolute coordinate data.

Note
When converting a geometry value to VARCHAR or NVARCHAR, the server uses the ST_AsText method. The st_geometry_astext_format option defines the format that is used for the conversion. See "st_geometry_astext_format option" [*SQL Anywhere Server - Database Administration*].

Note
By default, ST_AsText uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_AsGeoJSON method for type ST_Geometry"
- "ST_AsGML method for type ST_Geometry"
- "ST_AsKML method for type ST_Geometry"
- "ST_AsSVG method for type ST_Geometry"
- "ST_AsWKT method for type ST_Geometry"

Standards and compatibility

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

Example

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" [*SQL Anywhere Server - Database Administration*].

```
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" [*SQL Anywhere Server - Database Administration*].

```
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' )
```

ST_AsWKB method

Returns the WKB representation of an ST_Geometry value.

Syntax

geometry-expression.**ST_AsWKB**([*format*])

Parameters

Name	Type	Description
format	VAR-CHAR(128)	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*.

Remarks

The ST_AsWKB method returns a binary string representing the geometry in WKB format. A number of different formats are supported (with associated options) and the desired format is selected using the optional *format* parameter. If the *format* parameter is not specified, the default is 'WKB'.

The format string defines an output format and parameters to the format. The format string has one of the following formats:

```
format-name
format-name(parameter1=value1;parameter2=value2;...)
parameter1=value1;parameter2=value2;...
```

The first format specifies the format name and no parameters. All format parameters use their default values. The second format specifies the format name and a list of named parameter values. Parameters that are not supplied use their default values. The last format specifies only parameter values, and the format name defaults to 'WKB'.

The following format names may be used:

- **WKB** The Well-Known Binary format defined by SQL/MM and the OGC.
- **EWKB** The Extended Well-Known Binary format defined by PostGIS. This format includes the geometry's SRID and it differs from WKB in the way it represents Z and M values.

The following format parameters can be specified:

Format Name	Parameter Name	Default Value	Allowed Values	Description
WKB	Version	1.2	• **1.1** The WKB defined by OGC SFS 1.1. This format does not contain Z and M values. If the geometry contains Z or M values, they are removed in the output. • **1.2** The WKB defined by OGC SFS 1.2. This matches version 1.1 on 2D data and extends the format to support Z and M values.	The Version parameter controls the version of the WKB specification used.

Note
By default, ST_AsWKB uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

The following example returns the result `0x01b90b0000000000000000f03f00000000000000400000000000000084000000000000001040`.

```
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 `0x01010000e0e6100000000000000000f03f0000000000000040000000000000084000000000000001040`.

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

ST_AsWKT method

Returns the WKT representation of an ST_Geometry value.

Syntax

geometry-expression.**ST_AsWKT**([*format*])

Parameters

Name	Type	Description
format	VAR-CHAR(128)	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*.

Remarks

The ST_AsWKT method returns a text string representing the geometry. A number of different text formats are supported (with associated options) and the desired format is selected using the optional *format* parameter.

The format string defines an output format and parameters to the format. The format string has one of the following formats:

```
format-name

format-name(parameter1=value1;parameter2=value2;...)

parameter1=value1;parameter2=value2;...
```

The first format specifies the format name and no parameters. All format parameters use their default values. The second format specifies the format name and a list of named parameter values. Parameters that are not supplied use their default values. The last format specifies only parameter values, and the format name defaults to 'WKT'.

The following format names may be used:

- **WKT** The Well Known Text format defined by SQL/MM and the OGC.
- **EWKT** The Extended Well Known Text format defined by PostGIS. This format includes the geometry's SRID and it differs from WKT in the way it represents Z and M values.

The following format parameters can be specified:

Format Name	Parameter Name	Default Value	Allowed Values	Description
WKT	Version	1.2	• **1.1** The WKT defined by OGC SFS 1.1. This format does not contain Z and M values. If the geometry contains Z or M values, they are removed in the output. • **1.2** The WKT defined by OGC SFS 1.2. This matches version 1.1 on 2D data and extends the format to support Z and M values. • **PostGIS** The WKT format used by some other vendors; Z and M values are included in a fashion that does not match OGC 1.2.	The Version parameter controls the version of the WKT specification used.

Note
By default, ST_AsWKT uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

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
```

ST_AsXML method

Returns the XML representation of an ST_Geometry value.

Syntax

geometry-expression.**ST_AsXML**([*format*])

Parameters

Name	Type	Description
format	VARCHAR(128)	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. See "st_geometry_asxml_format option" [*SQL Anywhere Server - Database Administration*].

Returns

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

Remarks

The ST_AsXML method returns an XML string representing the geometry. GML, KML and SVG are the supported XML formats. The *format* parameter specifies parameters that control the conversion to XML. If *format* is not specified, the value of the st_geometry_asxml_format option is used to select the output format. See "st_geometry_asxml_format option" [*SQL Anywhere Server - Database Administration*].

The format string defines an output format and parameters to the format. The format string has one of the following formats:

```
format-name
format-name(parameter1=value1;parameter2=value2;...)
parameter1=value1;parameter2=value2;...
```

The first format specifies the format name and no parameters. All format parameters use their default values. The second format specifies the format name and a list of named parameter values. Parameters that are not supplied use their default values. The last format specifies only parameter values, and the format name defaults to 'GML'.

The following format names may be used:

- **GML** The Geography Markup Language format defined by ISO 19136 and the OGC.
- **KML** The Keyhole Markup Language format defined by the OGC.
- **SVG** The Scalable Vector Graphics (SVG) 1.1 format defined by the World Wide Web Consortium (W3C).

The following format parameters can be specified:

Format Name	Parameter Name	Default Value	Allowed Values	Description
GML	Version	3	• **2** Version 2 of the GML specification. • **3** Version 3.2 of the GML specification	The version parameter controls the version of the GML specification used.
GML	Namespace	none	• **local** Provides a default namespace attribute for the given element (in this case Point) and its sub elements. • **global** 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. • **none** Provides no namespace or prefix for the given element (in this case Point) and its sub elements	The namespace parameter specifies the output format convention for namespace.
GML	SRSName-Format	short	• **short** Uses a short format for the spatial reference system name, for example EPSG:4326 • **long** Uses a long format for the spatial reference system name, for example urn:x-ogc:def:crs:EPSG:4326. • **none** Spatial reference system name attribute is not included for the geometry.	The SRSNameFormat parameter specifies the format for the srsName attribute.

Format Name	Parameter Name	Default Value	Allowed Values	Description
GML	SRSDimen-sion	No	**Yes** or **No**	The SRSDimension parame-ter specifies the number of coordinate values for the giv-en geometry. This only ap-plies to GML(version=3).
GML	SRSFillAll	No	**Yes** or **No**	The SRSFillAll parameter specifies whether or not SRS attributes should be propaga-ted to child geometry ele-ments. As an example a Mul-tiGeometry or MultiPolygon would propagate the attributes to its child geometries.
GML	UseDepre-cated	No	**Yes** or **No**	The UseDeprecated parameter only applies to GML(ver-sion=3). It is used to output older GML representations where possible. As an exam-ple a Surface may be output as a Polygon if the geometry contains no CircularStrings.
GML	Attribute	Automati-cally gen-erated op-tional at-tributes	One or more attributes may be specified for the top level ge-ometry element only	Any legal XML attributes may be specified.
GML	SubElement	Automati-cally gen-erated GML sub elements	One or more sub elements may be specified for the top level geometry element only	Any legal XML elements may be specified.
KML	Version	2	2	KML version 2.2 is suppor-ted.
KML	Attribute	Automati-cally gen-erated op-tional at-tributes	One or more attributes may be specified for the top level ge-ometry element only	Any legal XML attributes may be specified.

Format Name	**Parameter Name**	**Default Value**	**Allowed Values**	**Description**
KML	Namespace	none	• **local** Provides the default namespace attribute **http://www.opengis.net/kml/2.2** for the given geometry element (in this case Point) and its sub elements. • **global** 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. • **none** Provides no namespace or prefix for the given element (in this case Point) and its sub elements	The namespace parameter specifies the output format convention for namespace.
KML	SubElement	Automatically generated KML sub elements	One or more sub elements may be specified for the top level geometry element only	Any legal XML elements may be specified. As an example extrude, tessellate and altitudeMode elements may be specified.
SVG	Approximate	Yes	**Yes** or **No**	The Approximate parameter specifies whether or not to reduce the size of the output SVG document with a slight reduction in visible detail. The SVG data is approximated by not including points which are within the line width of the last point. With multiple megabyte geometries this can result in compression rates of 80% or more.

Format Name	Parameter Name	Default Value	Allowed Values	Description
SVG	Attribute	Automatically generated optional attributes	One or more SVG attributes that can be applied to SVG shape elements	By default, optional SVG shape attributes such as fill, stroke and stroke-width are generated. If the Attributes parameter is specified, then no optional SVG shape attributes are generated, and the Attribute value is used instead. Ignored if PathDataOnly=Yes is specified. The maximum length of the Attribute value is about 1000 bytes.
SVG	DecimalDigits	Based on the number of decimal digits in the spatial reference system's snap to grid gridsize. The maximum default value is 5 and the minimum is 0.	integer	The DecimalDigits parameter limits the number of digits after the decimal place for coordinates generated in the SVG output. Specifying a negative number of digits indicates that the full precision of coordinates should be included in the SVG output.

Format Name	Parameter Name	Default Value	Allowed Values	Description
SVG	PathDataOnly	No (a complete SVG document is generated)	**Yes** or **No**	The PathDataOnly parameter specifies whether or not only data for the SVG Path Element should be generated. The PathDataOnly example below demonstrates how PathDataOnly=Yes can be used to build a complete SVG document that can be displayed. By default a complete SVG document is generated. The path data returned by PathDataOnly=Yes can be used to build more flexible SVG documents containing other elements, such as text.
SVG	RandomFill	Yes	**Yes** or **No**	The RandomFill parameter specifies whether or not polygons should be filled by a randomly generated color. The sequence of colors used does not follow a well-defined sequence, and typically changes each time SVG output is generated. **No** indicates that only an outline of each polygon is drawn. The RandomFill parameter is ignored if the Attribute or PathDataOnly=Yes parameter is specified.
SVG	Relative	Yes	**Yes** or **No**	The Relative parameter specifies if coordinates should be output in relative (offset) or absolute formats. Relative coordinate data is typically more compact than absolute coordinate data.

Note
When converting a geometry value to XML, the server uses the ST_AsXML method. The st_geometry_asxml_format option defines the format that is used for the conversion. See "st_geometry_asxml_format option" [*SQL Anywhere Server - Database Administration*].

> **Note**
> By default, ST_AsXML uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_AsGML method for type ST_Geometry"
- "ST_AsKML method for type ST_Geometry"
- "ST_AsSVG method for type ST_Geometry"

Standards and compatibility

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

Example

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' )
```

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*.

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

Remarks

The ST_Boundary method returns the spatial boundary of the *geometry-expression*. Geometries are characterized by their interior, boundary, and exterior. All geometry values are defined to be topologically closed, that is the boundary is considered to be part of the geometry.

Point geometries have an empty boundary. Curve geometries may be closed, in which case they have an empty boundary. If a curve is not closed, the start and end point of the curve form the boundary. For a surface geometry, the boundary is the set of curves that delineate the edge of the surface. For example, for a polygon the boundary of the geometry consists of the exterior ring and any interior rings.

See also: "How geometry interiors, exteriors, and boundaries work" on page 49.

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

Note
This method can not be used with geometries in round-Earth spatial reference system.

Standards and compatibility

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

Example

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()
```

ST_Contains method

Tests if a geometry value spatially contains another geometry value.

Syntax

geometry-expression.**ST_Contains**(*geo2*)

Parameters

Name	Type	Description
geo2	ST_Geometry	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.

Remarks

The ST_Contains method tests if the *geometry-expression* completely contains *geo2* and there is one or more interior points of *geo2* that lies in the interior of the *geometry-expression*.

geometry-expression.ST_Contains(*geo2*) is equivalent to *geo2*.ST_Within(*geometry-expression*).

The ST_Contains and ST_Covers methods are similar. The difference is that ST_Covers does not require intersecting interior points.

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

Note
This method can not be used with geometries in round-Earth spatial reference system.

See also

- "ST_Within method for type ST_Geometry"
- "ST_Covers method for type ST_Geometry"
- "ST_Intersects method for type ST_Geometry"
- "ST_ContainsFilter method for type ST_Geometry"

Standards and compatibility

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

Example

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

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

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
```

ST_ContainsFilter method

An inexpensive test if a geometry might contain another.

Syntax

geometry-expression.**ST_ContainsFilter**(*geo2*)

Parameters

Name	Type	Description
geo2	ST_Geometry	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.

Remarks

The ST_ContainsFilter method provides an efficient test to determine if one geometry might contain the other. Returns 1 if the *geometry-expression* might contain *geo2*, otherwise 0.

This test is less expensive than ST_Contains, but may return 1 in some cases where the *geometry-expression* does not actually contain *geo2*.

Therefore, this method can be useful as a primary filter when further processing will determine whether geometries interact in the desired way.

The implementation of ST_ContainsFilter relies upon metadata associated with the stored geometries. Because the available metadata may change between server versions, depending upon how the data is loaded, or where ST_ContainsFilter is used within a query, the expression *geometry-expression*.ST_ContainsFilter(*geo2*) can return different results when *geometry-expression* does not contain *geo2*. Whenever *geometry-expression* contains *geo2*, ST_ContainsFilter always returns 1.

Note
This method can not be used with geometries in round-Earth spatial reference system.

See also

- "ST_Contains method for type ST_Geometry"

Standards and compatibility

- **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.

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

Remarks

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

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

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

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

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

Note
ST_ConvexHull is not supported on geometries which contain circularstrings.

Note
This method can not be used with geometries in round-Earth spatial reference system.

See also

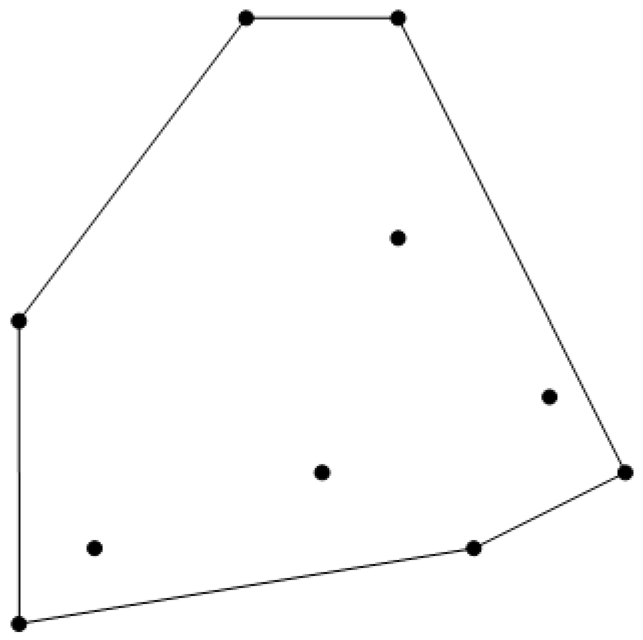
- “ST_ConvexHullAggr method for type ST_Geometry”

Standards and compatibility

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

Example

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



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

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()
```

ST_ConvexHullAggr method

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

Syntax

ST_Geometry::**ST_ConvexHullAggr**(*geometry-column*)

Parameters

Name	Type	Description
geometry-column	ST_Geometry	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.

The spatial reference system identifier of the result is the same as that for the first parameter.

Remarks

The ST_ConvexHullAggr method considers all of the points in the group of geometries it is computed over and returns the convex hull of all these points. The convex hull of a geometry is the smallest convex geometry that contains all of the points in the geometry.

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

If the geometries in the group consist of a single point, the point is returned. If all of the points of the group of geometries lie on a single straight line segment, a linestring is returned. Otherwise, a convex polygon is returned.

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

> **Note**
> ST_ConvexHullAggr is not supported on geometries which contain circularstrings.

> **Note**
> This method can not be used with geometries in round-Earth spatial reference system.

See also

- "ST_ConvexHull method for type ST_Geometry"

Standards and compatibility

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

Example

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
```

ST_CoordDim method

Returns the number of coordinate dimensions stored with each point of the ST_Geometry value.

Syntax

geometry-expression.**ST_CoordDim**()

Returns

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

Remarks

The ST_CoordDim method returns the number of coordinates stored within each point in the geometry. All geometries have at least two coordinate dimensions. For geographic spatial reference systems, these are the latitude and longitude of the point. For other spatial reference system, these coordinates are the X and Y positions of the point.

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

The following values may be returned by the ST_CoordDim method:

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

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

Note
By default, ST_CoordDim uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_Is3D method for type ST_Geometry"
- "ST_IsMeasured method for type ST_Geometry"

Standards and compatibility

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

Example

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()
```

ST_CoveredBy method

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

Syntax

geometry-expression.**ST_CoveredBy**(*geo2*)

Parameters

Name	Type	Description
geo2	ST_Geometry	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.

Remarks

The ST_CoveredBy method tests if the *geometry-expression* is completely covered by *geo2*.

geometry-expression.ST_CoveredBy(*geo2*) is equivalent to *geo2*.ST_Covers(*geometry-expression*).

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

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

See also

- "ST_Covers method for type ST_Geometry"
- "ST_Within method for type ST_Geometry"
- "ST_Intersects method for type ST_Geometry"
- "ST_CoveredByFilter method for type ST_Geometry"

Standards and compatibility

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

Example

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

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

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
```

ST_CoveredByFilter method

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

Syntax

geometry-expression.**ST_CoveredByFilter**(*geo2*)

Parameters

Name	Type	Description
geo2	ST_Geometry	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.

Remarks

The ST_CoveredByFilter method provides an efficient test to determine if one geometry might be covered by the other.

This test is less expensive than ST_CoveredBy, but may return 1 in some cases where the *geometry-expression* is not actually covered by *geo2*.

Therefore, this method can be useful as a primary filter when further processing will determine whether geometries interact in the desired way.

The implementation of ST_CoveredByFilter relies upon metadata associated with the stored geometries. Because the available metadata may change between server versions, depending upon how the data is loaded, or where ST_CoveredByFilter is used within a query, the expression *geometry-expression*.ST_CoveredByFilter(*geo2*) can return different results when *geometry-expression* is not covered by *geo2*. Whenever *geometry-expression* is covered by *geo2*, ST_CoveredByFilter always returns 1.

See also

- "ST_CoveredBy method for type ST_Geometry"

Standards and compatibility

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

ST_Covers method

Tests if a geometry value spatially covers another geometry value.

Syntax

geometry-expression.**ST_Covers**(*geo2*)

Parameters

Name	Type	Description
geo2	ST_Geometry	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.

Remarks

The ST_Covers method tests if the *geometry-expression* completely covers *geo2*. *geometry-expression*.ST_Covers(*geo2*) is equivalent to *geo2*.ST_CoveredBy(*geometry-expression*).

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

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

See also

- "ST_CoveredBy method for type ST_Geometry"
- "ST_Contains method for type ST_Geometry"
- "ST_Intersects method for type ST_Geometry"
- "ST_CoversFilter method for type ST_Geometry"

Standards and compatibility

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

Example

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

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

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
```

ST_CoversFilter method

An inexpensive test if a geometry might cover another.

Syntax

geometry-expression.**ST_CoversFilter**(*geo2*)

Parameters

Name	Type	Description
geo2	ST_Geometry	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.

Remarks

The ST_CoversFilter method provides an efficient test to determine if one geometry might cover the other. Returns 1 if the *geometry-expression* might cover *geo2*, otherwise 0.

This test is less expensive than ST_Covers, but may return 1 in some cases where the *geometry-expression* does not actually cover *geo2*.

Therefore, this method can be useful as a primary filter when further processing will determine whether geometries interact in the desired way.

The implementation of ST_CoversFilter relies upon metadata associated with the stored geometries. Because the available metadata may change between server versions, depending upon how the data is loaded, or where ST_CoversFilter is used within a query, the expression *geometry-expression*.ST_CoversFilter(*geo2*) can return different results when *geometry-expression* does not cover *geo2*. Whenever *geometry-expression* covers *geo2*, ST_CoversFilter always returns 1.

See also

- "ST_Covers method for type ST_Geometry"

Standards and compatibility

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

ST_Crosses method

Tests if a geometry value crosses another geometry value.

Syntax

geometry-expression.**ST_Crosses**(*geo2*)

Parameters

Name	Type	Description
geo2	ST_Geometry	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.

Remarks

Tests if a geometry value crosses another geometry value.

When both *geometry-expression* and *geo2* are curves or multicurves, they cross each other if their interiors intersect at one or more points. If the intersection results in a curve or multicurve, the geometries do not cross. If all of the intersecting points are boundary points, the geometries do not cross.

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

More precisely, *geometry-expression*.ST_Crosses(*geo2*) returns 1 when the following is TRUE:

```
( geometry-expression.ST_Dimension() = 1 AND geo2.ST_Dimension() = 1 AND
geometry-expression.ST_Relate( geo2, '0********' ) = 1 ) OR( geometry-
expression.ST_Dimension() < geo2.ST_Dimension() AND geometry-
expression.ST_Relate( geo2, 'T*T******' ) = 1 )
```

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

Note
This method can not be used with geometries in round-Earth spatial reference system.

See also

- "ST_Intersects method for type ST_Geometry"
- "ST_Dimension method for type ST_Geometry"
- "ST_Relate method for type ST_Geometry"
- "ST_Overlaps method for type ST_Geometry"

Standards and compatibility

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

Example

The following example returns the result `1`.

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

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 )' ) )
```

ST_Difference method

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

Syntax

geometry-expression.**ST_Difference**(*geo2*)

Parameters

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

Returns

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

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

Remarks

The ST_Difference method finds the spatial difference of two geometries. A point is included in the result if it is present in the *geometry-expression* but not present in *geo2*.

Unlike other spatial set operations (ST_Union, ST_Intersection, ST_SymDifference), the ST_Difference method is not symmetric: the method can give a different answer for `A.ST_Difference( B )` and `B.ST_Difference( A )`.

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

See also

- "ST_Intersection method for type ST_Geometry"
- "ST_SymDifference method for type ST_Geometry"
- "ST_Union method for type ST_Geometry"

Standards and compatibility

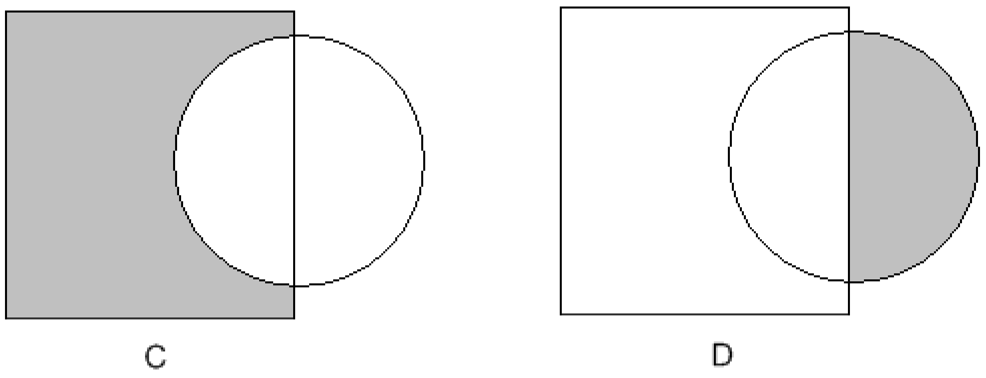
- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.20

Example

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.



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.

Remarks

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

- **-1** The geometry corresponds to the empty set.
- **0** The geometry consists only of individual points (for example, an ST_Point or ST_MultiPoint).
- **1** The geometry contains at least one curve and no surfaces (for example, an ST_LineString or ST_MultiCurve).
- **2** The geometry consists of at least one surface (for example, an ST_Polygon or ST_MultiPolygon).

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

See "How spatial dimensions work" on page 54.

Note
By default, ST_Dimension uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_CoordDim method for type ST_Geometry"
- "ST_Relate method for type ST_Geometry"
- "ST_Relate method for type ST_Geometry"

Standards and compatibility

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

Example

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()
```

ST_Disjoint method

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

Syntax

geometry-expression.**ST_Disjoint**(*geo2*)

Parameters

Name	Type	Description
geo2	ST_Geometry	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.

Remarks

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

geometry-expression.ST_Disjoint(*geo2*) = 1 is equivalent to *geometry-expression*.ST_Intersects(*geo2*) = 0.

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

See also

- "ST_Intersects method for type ST_Geometry"

Standards and compatibility

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

Example

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

```
SELECT ShapeID, "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

ST_Distance method

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

Syntax

geometry-expression.**ST_Distance**(*geo2*[, *unit-name*])

Parameters

Name	Type	Description
geo2	ST_Geometry	The other geometry value whose distance is to be measured from the *geometry-expression*.
unit-name	VARCHAR(128)	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.

Remarks

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

Note
For round-Earth spatial reference systems, the ST_Distance method is only supported if *geometry-expression* and *geo2* contain only points.

Note
By default, ST_Distance uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_Area method for type ST_Surface"
- "ST_Length method for type ST_Curve"
- "ST_Perimeter method for type ST_Surface"
- "ST_WithinDistance method for type ST_Geometry"
- "ST_WithinDistanceFilter method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.23

Example

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

ShapeID	dist
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. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

```
SELECT ROUND( NEW ST_Point( -63.573566, 44.646244, 4326 )
    .ST_Distance( NEW ST_Point( -80.522372, 43.465187, 4326 )
                    , 'Statute mile' ), 0 )
```

ST_Envelope method

Returns the bounding rectangle for the geometry value.

Syntax

geometry-expression.**ST_Envelope**()

Returns

- **ST_Polygon** Returns a polygon that is the bounding rectangle for the *geometry-expression*.

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

Remarks

The ST_Envelope method constructs a polygon that is an axis-aligned bounding rectangle for the *geometry-expression*. The envelope covers the entire geometry, and it can be used as a simple approximation for the geometry.

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

Note
This method can not be used with geometries in round-Earth spatial reference system.

See also

- "ST_EnvelopeAggr method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.15

Example

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
```

ST_EnvelopeAggr method

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

Note
This method can not be used with geometries in round-Earth spatial reference system.

Syntax

ST_Geometry::**ST_EnvelopeAggr**(*geometry-column*)

Parameters

Name	Type	Description
geometry-column	ST_Geometry	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.

  The spatial reference system identifier of the result is the same as that for the first parameter.

See also

- "ST_Envelope method for type ST_Geometry"

Standards and compatibility

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

Example

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
```

ST_Equals method

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

Syntax

geometry-expression.**ST_Equals**(*geo2*)

Parameters

Name	Type	Description
geo2	ST_Geometry	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.

Remarks

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

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

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

Two geometry values can be considered equal even though they have different representations. For example, two linestrings may have opposite orientations but contain the same set of points in space. These two linestrings are considered equal by ST_Equals but not by ST_OrderingEquals. See “How spatial comparisons work” on page 50.

ST_Equals may be limited by the resolution of the spatial reference system or the accuracy of the data.

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

See also

- “ST_OrderingEquals method for type ST_Geometry”
- “ST_EqualsFilter method for type ST_Geometry”

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.24

Example

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 )' ) )
```

ST_EqualsFilter method

An inexpensive test if a geometry is equal to another.

Syntax

geometry-expression.**ST_EqualsFilter**(*geo2*)

Parameters

Name	Type	Description
geo2	ST_Geometry	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.

Remarks

The ST_EqualsFilter method provides an efficient test to determine if a geometry might be equal to another. ST_EqualsFilter returns 1 if *geometry-expression* might be equal to *geo2*; otherwise ST_EqualsFilter returns 0.

This test is less expensive than ST_Equals, but can return 1 in some cases where the *geometry-expression* is not actually equal to *geo2*.

Therefore, This method can be useful as a primary filter when further processing will determine whether geometries interact in the desired way.

The implementation of ST_EqualsFilter relies upon metadata associated with the stored geometries. Because the available metadata may change between server versions, depending upon how the data is loaded, or where ST_EqualsFilter is used within a query, the expression *geometry-expression*.ST_EqualsFilter(*geo2*) can return different results when *geometry-expression* does not equal *geo2*. Whenever *geometry-expression* equals *geo2*, ST_EqualsFilter always returns 1.

See also

- "ST_Equals method for type ST_Geometry"
- "ST_OrderingEquals method for type ST_Geometry"

Standards and compatibility

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

ST_GeomFromBinary method

Constructs a geometry from a binary string representation.

Syntax

ST_Geometry::**ST_GeomFromBinary**(*binary-string*[, *srid*])

Parameters

Name	Type	Description
binary-string	LONG BINARY	A string containing the binary representation of a geometry. The input can be in any supported binary format, including WKB or EWKB.
srid	INT	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.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

Parses a string containing one of the supported formats and creates a geometry value of the appropriate type. This method is used by the server when evaluating a cast from a binary string to a geometry type.

Some input formats contain a SRID definition. If provided, the *srid* parameter must match any value taken from the input string.

See also

- "ST_GeomFromWKB method for type ST_Geometry"
- "ST_GeomFromText method for type ST_Geometry"

Standards and compatibility

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

Example

The following example returns the result `Point (10 20)`.

```
SELECT
ST_Geometry::ST_GeomFromBinary( 0x010100000000000000000024400000000000003440
)
```

ST_GeomFromShape method

Parses a string containing an ESRI shape record and creates a geometry value of the appropriate type.

Syntax

ST_Geometry::**ST_GeomFromShape**(*shape*[, *srid*])

Parameters

Name	**Type**	**Description**
shape	LONG BINARY	A string containing a geometry in the ESRI shape format.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_Geometry** Returns a geometry value of the appropriate type based on the source string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

Parses a string containing a single ESRI shape and creates a geometry value of the appropriate type. The record is a single record from the *.shp* file of an ESRI shapefile or it could be a single string value from a geodatabase.

The Shape representation is widely used to represent spatial data. For a full description of the shape definition, see the ESRI website, http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf.

In most cases it is easier to load an ESRI shapefile using the SHAPEFILE format with the FORMAT clause of the LOAD TABLE statement, or an OPENSTRING expression in a FROM clause instead of using the ST_GeomFromShape method. See "LOAD TABLE statement" [*SQL Anywhere Server - SQL Reference*], and "FROM clause" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

ST_GeomFromText method

Constructs a geometry from a character string representation.

Syntax

ST_Geometry::**ST_GeomFromText**(*character-string*[, *srid*])

Parameters

Name	Type	Description
character-string	LONG VAR-CHAR	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	INT	The SRID of the result. If not specified and the input string does not contain a SRID, the default is 0.

Returns

- **ST_Geometry** Returns a geometry value of the appropriate type based on the source string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

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

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

See also

- "ST_GeomFromBinary method for type ST_Geometry"
- "ST_GeomFromWKT method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.40

Example

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

```
SELECT ST_Geometry::ST_GeomFromText( 'LineString( 1 2, 5 7 )', 4326 )
```

ST_GeomFromWKB 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**(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINA-RY	A string containing the WKB or EWKB representation of a geometry value.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_Geometry** Returns a geometry value of the appropriate type based on the source string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

Parses a string containing the WKB or EWKB representation of a geometry value and creates a geometry value of the appropriate type.

See also

- "ST_GeomFromBinary method for type ST_Geometry"
- "ST_GeomFromWKT method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.41

ST_GeomFromWKT 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**(*wkt*[, *srid*])

Parameters

Name	Type	Description
wkt	LONG VAR-CHAR	A string containing the WKT or EWKT representation of a geometry value.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_Geometry** Returns a geometry value of the appropriate type based on the source string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

Parses a string containing the WKT or EWKT representation of a geometry value and creates a geometry value of the appropriate type.

See also

- "ST_GeomFromText method for type ST_Geometry"
- "ST_GeomFromWKB method for type ST_Geometry"

Standards and compatibility

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

Remarks

The ST_GeometryType method returns a string containing the specific type name of *geometry-expression*.

The value IS OF(type) syntax can also be used to determined the specific type of a value.

Note
By default, ST_GeometryType uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)**   5.1.4

Example

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' )
```

ST_GeometryTypeFromBaseType method

Parses a string defining the type string.

Syntax

ST_Geometry::**ST_GeometryTypeFromBaseType**(*base-type-str*)

Parameters

Name	Type	Description
base-type-str	VARCHAR(128)	A string containing the base type string

Returns

- **VARCHAR(128)** 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.

Remarks

The *ST_Geometry::ST_GeometryTypeFromBaseType* method can be used to parse the geometry type name out of a type string definition.

See also

- "ST_SRIDFromBaseType method for type ST_Geometry"

Standards and compatibility

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

Example

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'
```

ST_Intersection method

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

Syntax

geometry-expression.**ST_Intersection**(*geo2*)

Parameters

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

Returns

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

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

Remarks

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

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

See also

- "ST_Difference method for type ST_Geometry"
- "ST_IntersectionAggr method for type ST_Geometry"
- "ST_SymDifference method for type ST_Geometry"
- "ST_Union method for type ST_Geometry"

Standards and compatibility

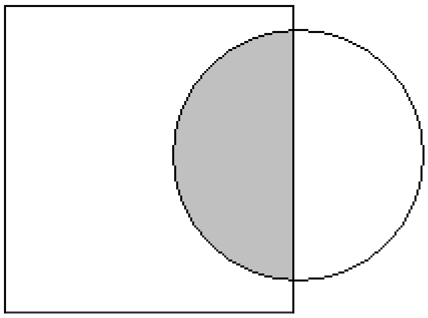
- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.18

Example

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.



ST_IntersectionAggr method

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

Syntax

ST_Geometry::**ST_IntersectionAggr**(*geometry-column*)

Parameters

Name	Type	Description
geometry-column	ST_Geometry	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.

  The spatial reference system identifier of the result is the same as that for the first parameter.

Remarks

Rows where the argument is NULL are not included.

Returns NULL for an empty group or a group containing no non-NULL values.

If the group contains a single non-NULL geometry, it is returned. Otherwise, the intersection is logically computed by repeatedly applying the ST_Intersection method to combine two geometries at a time. See "ST_Intersection method for type ST_Geometry" on page 176.

See also

- "ST_Intersection method for type ST_Geometry"

Standards and compatibility

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

Example

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 )
```

ST_Intersects method

Test if a geometry value spatially intersects another value.

Syntax

geometry-expression.**ST_Intersects**(*geo2*)

Parameters

Name	Type	Description
geo2	ST_Geometry	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.

Remarks

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

geometry-expression.ST_Intersects(*geo2*) = 1 is equivalent to *geometry-expression*.ST_Disjoint(*geo2*) = 0.

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

See also

- "ST_IntersectsRect method for type ST_Geometry"
- "ST_Disjoint method for type ST_Geometry"
- "ST_IntersectsFilter method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.27

Example

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 )' )
```

ST_IntersectsFilter method

An inexpensive test if the two geometries might intersect.

Syntax

geometry-expression.**ST_IntersectsFilter**(*geo2*)

Parameters

Name	Type	Description
geo2	ST_Geometry	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.

Remarks

The ST_IntersectsFilter method provides an efficient test to determine if two geometries might intersect. Returns 1 if the *geometry-expression* might intersect with *geo2*, otherwise 0.

This test is less expensive than ST_Intersects, but may return 1 in some cases where the geometries do not actually intersect. Therefore, this method can be useful as a primary filter when further processing will determine if geometries truly intersect.

The implementation of ST_IntersectsFilter relies upon metadata associated with the stored geometries. Because the available metadata may change between server versions, depending upon how the data is loaded, or where ST_IntersectsFilter is used within a query, the expression *geometry-expression*.ST_IntersectsFilter(*geo2*) can return different results when *geometry-expression* does not intersect *geo2*. Whenever *geometry-expression* intersects *geo2*, ST_IntersectsFilter always returns 1.

See also

- "ST_Intersects method for type ST_Geometry"
- "ST_IntersectsRect method for type ST_Geometry"

Standards and compatibility

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

ST_IntersectsRect method

Test if a geometry intersects a rectangle.

Syntax

geometry-expression.**ST_IntersectsRect**(*pmin*,*pmax*)

Parameters

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

Returns

- **BIT** Returns 1 if the *geometry-expression* intersects with the specified rectangle, otherwise 0.

Remarks

The ST_IntersectsRect method tests if a geometry intersects with a specified axis-aligned bounding rectangle.

The method is equivalent to the following: *geometry-expression*.ST_Intersects(NEW ST_Polygon(pmin, pmax))

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

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

See also

- "ST_Intersects method for type ST_Geometry"
- "ST_IntersectsFilter method for type ST_Geometry"

Standards and compatibility

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

Example

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 ) )
```

ST_Is3D method

Determines if the geometry value has Z coordinate values.

Note
By default, ST_Is3D uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

geometry-expression.**ST_Is3D**()

Returns

- **BIT** Returns 1 if the geometry value has Z coordinate values, otherwise 0.

See also

- "ST_CoordDim method for type ST_Geometry"
- "ST_IsMeasured method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.10

Example

The following example returns the result `1`.

```
SELECT ShapeID FROM SpatialShapes WHERE Shape.ST_Is3D() = 1
```

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.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.7

Example

The following example returns the result `1`.

```
SELECT NEW ST_LineString().ST_IsEmpty()
```

ST_IsMeasured method

Determines if the geometry value has associated measure values.

Note
By default, ST_IsMeasured uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

geometry-expression.**ST_IsMeasured**()

Returns

- **BIT** Returns 1 if the geometry value has measure values, otherwise 0.

See also

- "ST_CoordDim method for type ST_Geometry"
- "ST_Is3D method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.11

Example

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
```

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.

See also

- "ST_IsValid method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.8

Example

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
```

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.

Remarks

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

See also

- "ST_IsSimple method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.9

Example

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()
```

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*.

Remarks

Returns the northernmost latitude value of the *geometry-expression*. Note that in the round-Earth model, the northernmost latitude may not correspond to the latitude of any of the points defining the geometry.

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

Note
By default, ST_LatNorth uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see “STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement” [*SQL Anywhere Server - SQL Reference*].

See also

- “ST_LatSouth method for type ST_Geometry”
- “ST_LongEast method for type ST_Geometry”
- “ST_LongWest method for type ST_Geometry”
- “ST_YMax method for type ST_Geometry”

Standards and compatibility

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

Example

The following example returns the result `49.74`.

```
SELECT ROUND( NEW ST_LineString( 'LineString( -122 49, -96 49 )', 4326 )
                .ST_LatNorth(), 2 )
```

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*.

Remarks

Returns the southernmost latitude value of the *geometry-expression*. Note that in the round-Earth model, the southernmost latitude may not correspond to the latitude of any of the points defining the geometry.

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

Note
By default, ST_LatSouth uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_LatNorth method for type ST_Geometry"
- "ST_LongEast method for type ST_Geometry"
- "ST_LongWest method for type ST_Geometry"
- "ST_YMin method for type ST_Geometry"

Standards and compatibility

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

Example

The following example returns the result 49.

```
SELECT ROUND( NEW ST_LineString( 'LineString( -122 49, -96 49 )', 4326 )
                .ST_LatSouth(), 2 )
```

ST_LinearHash method

Returns a binary string that is a linear hash of the geometry.

Syntax

geometry-expression.**ST_LinearHash**()

Returns

- **BINARY(32)** Returns a binary string that is a linear hash of the geometry.

Remarks

The spatial index support uses a linear hash for geometries that maps the geometries in a table into a linear order in a B-Tree index. The ST_LinearHash method exposes this mapping by returning a binary string that gives the ordering of the rows in the B-Tree index. The hash string provides the following property: if geometry *A* covers geometry *B*, then `A.ST_LinearHash() >= B.ST_LinearHash()`.

The linear hash can be used in an ORDER BY clause. For example, when unloading data from a SELECT statement, ST_LinearHash can be used to generate a data file that matches the clustering of a spatial index.

See also

- "ST_LinearUnHash method for type ST_Geometry"

Standards and compatibility

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

ST_LinearUnHash method

Returns a geometry representing the index hash.

Syntax

ST_Geometry::**ST_LinearUnHash**(*index-hash*[, *srid*])

Parameters

Name	Type	Description
index-hash	BINARY(32)	The index hash string.
srid	INT	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.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_LinearUnHash method generates a representative geometry for a linear hash string generated by ST_LinearHash(). The server maps geometries to a linear order for spatial indexes, and the ST_LinearHash method gives a binary string that defines this linear ordering. The ST_LinearUnHash method reverses this operation to give a geometry that represents a particular hash string. The hash operation is lossy in the sense that multiple distinct geometries may hash to the same binary string. The ST_LinearUnHash method returns a geometry that covers any geometry that maps to the given linear hash.

The graphical plan for a query that uses a spatial index shows the linear hash values used to probe the spatial index. The ST_LinearUnHash method can be used to generate a geometry that represents these hashes.

See also

- "ST_LinearHash method for type ST_Geometry"

Standards and compatibility

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

ST_LoadConfigurationData method

Returns binary configuration data. For internal use only.

Syntax

ST_Geometry::**ST_LoadConfigurationData**(*configuration-name*)

Parameters

Name	Type	Description
configuration-name	VARCHAR(128)	The name of the configuration data item to load.

Returns

- **LONG BINARY** Returns binary configuration data. For internal use only.

Remarks

This method is used by the server to load configuration data from installed files. If the configuration files are not installed with the server, NULL is returned. For internal use only.

Standards and compatibility

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

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*.

Remarks

Returns the longitude of the eastern boundary of the *geometry-expression*. If the geometry crosses the date line in the round-Earth model, ST_LongWest will be greater than the ST_LongEast value.

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

Note
By default, ST_LongEast uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_LongWest method for type ST_Geometry"
- "ST_LatNorth method for type ST_Geometry"
- "ST_LatSouth method for type ST_Geometry"
- "ST_XMax method for type ST_Geometry"

Standards and compatibility

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

Example

The following example returns the result `-157.8`.

```
SELECT NEW ST_LineString( 'LineString( -157.8 21.3, 144.5 13 )', 4326 )
                .ST_LongEast()
```

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*.

Remarks

Returns the longitude of the western boundary of the *geometry-expression*. If the geometry crosses the date line in the round-Earth model, ST_LongWest will be greater than the ST_LongEast value.

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

Note
By default, ST_LongWest uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_LongEast method for type ST_Geometry"
- "ST_LatNorth method for type ST_Geometry"
- "ST_LatSouth method for type ST_Geometry"
- "ST_XMin method for type ST_Geometry"

Standards and compatibility

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

Example

The following example returns the result `144.5`.

```
SELECT NEW ST_LineString( 'LineString( -157.8 21.3, 144.5 13 )', 4326 )
                .ST_LongWest()
```

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*.

Remarks

Returns the maximum M coordinate value of the *geometry-expression*. This is computed by comparing the M attribute of all points in the geometry.

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

Note
By default, ST_MMax uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- “ST_XMin method for type ST_Geometry”
- “ST_XMax method for type ST_Geometry”
- “ST_YMin method for type ST_Geometry”
- “ST_YMax method for type ST_Geometry”
- “ST_ZMin method for type ST_Geometry”
- “ST_ZMax method for type ST_Geometry”
- “ST_MMin method for type ST_Geometry”

Standards and compatibility

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

Example

The following example returns the result 8.

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

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*.

Remarks

Returns the minimum M coordinate value of the *geometry-expression*. This is computed by comparing the M attribute of all points in the geometry.

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

Note
By default, ST_MMin uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see “STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement” [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_XMin method for type ST_Geometry"
- "ST_XMax method for type ST_Geometry"
- "ST_YMin method for type ST_Geometry"
- "ST_YMax method for type ST_Geometry"
- "ST_ZMin method for type ST_Geometry"
- "ST_ZMax method for type ST_Geometry"
- "ST_MMax method for type ST_Geometry"

Standards and compatibility

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

Example

The following example returns the result 4.

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

ST_OrderingEquals method

Tests if a geometry is identical to another geometry.

Syntax

geometry-expression.**ST_OrderingEquals**(*geo2*)

Parameters

Name	Type	Description
geo2	ST_Geometry	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.

Remarks

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

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

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

For more information, see "How spatial comparisons work" on page 50.

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

See also

- "ST_Equals method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.43

Example

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
```

ST_Overlaps method

Tests if a geometry value overlaps another geometry value.

Syntax

geometry-expression.**ST_Overlaps**(*geo2*)

Parameters

Name	Type	Description
geo2	ST_Geometry	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.

Remarks

Two geometries overlap if the following conditions are all true:

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

More precisely, *geometry-expression*.ST_Overlaps(*geo2*) returns 1 when the following is TRUE:

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

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

Note
This method can not be used with geometries in round-Earth spatial reference system.

See also

- "ST_Dimension method for type ST_Geometry"
- "ST_Intersects method for type ST_Geometry"
- "ST_Covers method for type ST_Geometry"
- "ST_Crosses method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.32

Example

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

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

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
```

ST_Relate method

Tests if a geometry value is spatially related to another geometry value as specified by the intersection matrix. The ST_Relate method uses a 9-character string from the Dimensionally Extended 9 Intersection Model (DE-9IM) to describe the pair-wise relationship between two spatial data items. For example, the ST_Relate method determines if an intersection occurs between the geometries, and the geometry of the resulting intersection, if it exists. See also: "How spatial relationships work" on page 51.

Overload list

Name	Description
"ST_Relate(ST_Geometry,CHAR(9)) method for type ST_Geometry"	Tests if a geometry value is spatially related to another geometry value as specified by the intersection matrix. The ST_Relate method uses a 9-character string from the Dimensionally Extended 9 Intersection Model (DE-9IM) to describe the pair-wise relationship between two spatial data items. For example, the ST_Relate method determines if an intersection occurs between the geometries, and the geometry of the resulting intersection, if it exists.
"ST_Relate(ST_Geometry) method for type 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_Relate(ST_Geometry,CHAR(9)) method for type ST_Geometry

Tests if a geometry value is spatially related to another geometry value as specified by the intersection matrix. The ST_Relate method uses a 9-character string from the Dimensionally Extended 9 Intersection Model (DE-9IM) to describe the pair-wise relationship between two spatial data items. For example, the ST_Relate method determines if an intersection occurs between the geometries, and the geometry of the resulting intersection, if it exists.

Syntax

geometry-expression.**ST_Relate**(*geo2*,*relate-matrix*)

Parameters

Name	Type	Description
geo2	ST_Geometry	The second geometry value that is to be compared to the *geometry-expression*.
relate-matrix	CHAR(9)	A 9-character string representing a matrix in the dimensionally-extended 9 intersection model. Each character defined in the 9-character string represents the type of intersection allowed at one of the nine possible intersections between the interior, boundary, and exterior of the two geometries.

Returns

- **BIT** Returns 1 if the two geometries have the specified relationship; otherwise 0.

Remarks

Tests if a geometry value is spatially related to another geometry value by testing for intersection between the interior, boundary, and exterior of two geometries, as specified by the intersection matrix. See also: "How spatial relationships work" on page 51.

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

Note
This method can not be used with geometries in round-Earth spatial reference system.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.25

Example

The following example returns a result with one row for each shape that has the '0F***T***' relationship with the specified line. The '0' means the interiors of both geometries must intersect and result in a point or multipoint. The 'F' means the interior of the line and the boundary of Shape must not intersect. The 'T' means the exterior of the line and the interior of the Shape must intersect.

```
SELECT ShapeID, "Description" From SpatialShapes
WHERE NEW ST_LineString( 'LineString( 0 0, 10 0 )' )
    .ST_Relate( Shape, '0F****T**' ) = 1
ORDER BY ShapeID
```

The example returns the following result set:

ShapeID	Description
18	CircularString
30	Multicurve

ST_Relate(ST_Geometry) method for type 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.

Syntax

geometry-expression.**ST_Relate**(*geo2*)

Parameters

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

Returns

- **CHAR(9)** 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.

Remarks

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

The intersection matrix is returned as a string. While it is possible to use this method variant to test a spatial relationship by examining the returned string, it is more efficient to test relationships by passing a pattern string as second parameter or by using specific spatial predicates such as ST_Contains or ST_Intersects. See also: “How spatial relationships work” on page 51.

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

Note
This method can not be used with geometries in round-Earth spatial reference system.

Standards and compatibility

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

Example

The following example returns the result `1F2001102`.

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

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.

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

Remarks

Returns the geometry with the order of its elements reversed. For a curve, a curve is returned with the vertices in the opposite order. For a collection, a collection is returned with the child geometries in the reversed order.

Note
By default, ST_Reverse uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

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()
```

ST_SRID method

Retrieves or modifies the spatial reference system associated with the geometry value.

Overload list

Name	Description
"ST_SRID() method for type ST_Geometry"	Returns the SRID of the geometry.

Name	Description
“ST_SRID(INT) method for type ST_Geometry”	Changes the spatial reference system associated with the geometry without modifying any of the values.

ST_SRID() method for type ST_Geometry

Returns the SRID of the geometry.

Syntax

geometry-expression.**ST_SRID**()

Returns

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

Remarks

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

Note
By default, ST_SRID uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see “STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement” [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.5

Example

The following returns the result `0`, indicating all Shapes in the table have the SRID 0, corresponding to the 'Default' spatial reference system.

```
SELECT DISTINCT Shape.ST_SRID() FROM SpatialShapes
```

ST_SRID(INT) method for type ST_Geometry

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

Syntax

geometry-expression.**ST_SRID**(*srid*)

Parameters

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

Returns

- **ST_Geometry** Returns a copy of the geometry value with the specified spatial reference system.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

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

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

Note
By default, ST_SRID uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_Transform method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.5

Example

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

```
SELECT NEW ST_Point( -118, 34,
4326 ).ST_SRID( 1000004326 ).ST_AsText( 'EWKT' )
```

ST_SRIDFromBaseType method

Parses a string defining the type string.

Syntax

ST_Geometry::**ST_SRIDFromBaseType**(*base-type-str*)

Parameters

Name	Type	Description
base-type-str	VARCHAR(128)	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.

Remarks

The *ST_Geometry::ST_SRIDFromBaseType* method can be used to parse the SRID out of a type string definition.

See also

- "ST_GeometryTypeFromBaseType method for type ST_Geometry"

Standards and compatibility

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

Example

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)')
```

ST_SnapToGrid method

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

Overload list

Name	Description
"ST_SnapToGrid(DOUBLE) method for type ST_Geometry"	Returns a copy of the geometry with all points snapped to the specified grid.
"ST_SnapToGrid(ST_Point,DOUBLE,DOUBLE,DOUBLE,DOUBLE) method for type ST_Geometry"	Returns a copy of the geometry with all points snapped to the specified grid.

ST_SnapToGrid(DOUBLE) method for type ST_Geometry

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

Syntax

geometry-expression.**ST_SnapToGrid**(*cell-size*)

Parameters

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

Returns

- **ST_Geometry** Returns the geometry with all points snapped to the grid.

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

Remarks

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

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

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

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

Note
By default, ST_SnapToGrid uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see “STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement” [*SQL Anywhere Server - SQL Reference*].

See also

- “How snap-to-grid and tolerance impact spatial calculations”

Standards and compatibility

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

Example

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

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

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

ST_SnapToGrid(ST_Point,DOUBLE,DOUBLE,DOUBLE,DOUBLE) method for type ST_Geometry

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

Syntax

geometry-expression.**ST_SnapToGrid**(*origin*,*cell-size-x*,*cell-size-y*,*cell-size-z*,*cell-size-m*)

Parameters

Name	Type	Description
origin	ST_Point	The origin of the grid.
cell-size-x	DOUBLE	The cell size for the grid in the X dimension.
cell-size-y	DOUBLE	The cell size for the grid in the Y dimension.
cell-size-z	DOUBLE	The cell size for the grid in the Z dimension.
cell-size-m	DOUBLE	The cell size for the grid in the M dimension.

Returns

- **ST_Geometry** Returns the geometry with all points snapped to the grid.

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

Remarks

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

You can specify a different cell size for each dimension. If you do not wish to snap the points of one dimension, you can use a cell size of zero.

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

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

Note
By default, ST_SnapToGrid uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "How snap-to-grid and tolerance impact spatial calculations"

Standards and compatibility

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

Example

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).

ST_SymDifference method

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

Syntax

geometry-expression.**ST_SymDifference**(*geo2*)

Parameters

Name	Type	Description
geo2	ST_Geometry	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.

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

Remarks

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

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

See also

- "ST_Difference method for type ST_Geometry"
- "ST_Intersection method for type ST_Geometry"
- "ST_Union method for type ST_Geometry"

Standards and compatibility

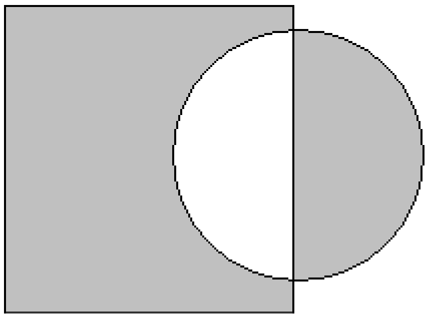
- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.21

Example

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.



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.

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

Remarks

Convert this geometry to a circularstring. The logic is equivalent to that used for CAST(*geometry-expression* AS ST_CircularString).

If *geometry-expression* is already known to be an ST_CircularString value, it is more efficient to use TREAT(*geometry-expression* AS ST_CircularString) than the ST_ToCircular method.

Note
By default, ST_ToCircular uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_ToCompound method for type ST_Geometry"
- "ST_ToCurve method for type ST_Geometry"
- "ST_ToLineString method for type ST_Geometry"
- "ST_ToMultiCurve method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.33

Example

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()
```

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.

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

Remarks

Converts the geometry to a circularstring. The logic is equivalent to that used for CAST(*geometry-expression* AS ST_CompoundCurve).

If *geometry-expression* is already known to be an ST_CompoundCurve value, it is more efficient to use TREAT(*geometry-expression* AS ST_CompoundCurve) than the ST_ToCompound method.

Note
By default, ST_ToCompound uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_ToCircular method for type ST_Geometry"
- "ST_ToCurve method for type ST_Geometry"
- "ST_ToLineString method for type ST_Geometry"
- "ST_ToMultiCurve method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.33

Example

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

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

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.

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

Remarks

Converts the geometry to a curve. The logic is equivalent to that used for **CAST(** *geometry-expression* **AS ST_Curve)**.

If *geometry-expression* is already known to be an ST_Curve value, it is more efficient to use TREAT(*geometry-expression* AS ST_Curve) than the ST_ToCurve method.

Note
By default, ST_ToCurve uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_ToCircular method for type ST_Geometry"
- "ST_ToCompound method for type ST_Geometry"
- "ST_ToLineString method for type ST_Geometry"
- "ST_ToMultiCurve method for type ST_Geometry"

Standards and compatibility

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

Example

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()
```

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.

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

Remarks

If *geometry-expression* is already known to be an ST_CurvePolygon value, it is more efficient to use TREAT(*geometry-expression* AS ST_CurvePolygon) than the ST_ToCurvePoly method.

Note
By default, ST_ToCurvePoly uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_ToPolygon method for type ST_Geometry"
- "ST_ToSurface method for type ST_Geometry"
- "ST_ToMultiSurface method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.33

Example

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()
```

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.

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

Remarks

If *geometry-expression* is already known to be an ST_GeomCollection value, it is more efficient to use TREAT(*geometry-expression* AS ST_GeomCollection) than the ST_ToGeomColl method.

Note
By default, ST_ToGeomColl uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_ToMultiCurve method for type ST_Geometry"
- "ST_ToMultiPoint method for type ST_Geometry"
- "ST_ToMultiSurface method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.33

Example

The following example returns the result `GeometryCollection (Point (0 1))`.

```
SELECT NEW ST_Point( 0, 1 ).ST_ToGeomColl()
```

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.

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

Remarks

Converts the geometry to a linestring. The logic is equivalent to that used for CAST(*geometry-expression* AS ST_LineString). If the *geometry-expression* is a circularstring or compound curve, it is interpolated using ST_CurveToLine().

If *geometry-expression* is already known to be an ST_LineString value, it is more efficient to use TREAT(*geometry-expression* AS ST_LineString) than the ST_ToLineString method.

Note
By default, ST_ToLineString uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_ToMultiLine method for type ST_Geometry"
- "ST_ToCircular method for type ST_Geometry"
- "ST_ToCompound method for type ST_Geometry"
- "ST_ToCurve method for type ST_Geometry"
- "ST_CurveToLine method for type ST_Curve"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.33

Example

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
```

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.

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

Remarks

If *geometry-expression* is already known to be an ST_MultiCurve value, it is more efficient to use TREAT(*geometry-expression* AS ST_MultiCurve) than the ST_ToMultiCurve method.

> **Note**
> By default, ST_ToMultiCurve uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_ToMultiLine method for type ST_Geometry"
- "ST_ToGeomColl method for type ST_Geometry"
- "ST_ToCurve method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.33

Example

The following example returns the result `MultiCurve ((0 7, 0 4, 4 4))`.

```
SELECT Shape.ST_ToMultiCurve()
FROM SpatialShapes WHERE ShapeID = 5
```

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.

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

Remarks

If *geometry-expression* is already known to be an ST_MultiLineString value, it is more efficient to use TREAT(*geometry-expression* AS ST_MultiLineString) than the ST_ToMultiLine method.

Note
By default, ST_ToMultiLine uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_ToMultiCurve method for type ST_Geometry"
- "ST_ToGeomColl method for type ST_Geometry"
- "ST_ToLineString method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.33

Example

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
```

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.

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

Remarks

If *geometry-expression* is already known to be an ST_MultiPoint value, it is more efficient to use TREAT(*geometry-expression* AS ST_MultiPoint) than the ST_ToMultiPoint method.

Note
By default, ST_ToMultiPoint uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_ToGeomColl method for type ST_Geometry"
- "ST_ToPoint method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.33

Example

The following example returns the result `MultiPoint EMPTY`.

```
SELECT NEW ST_GeomCollection().ST_ToMultiPoint()
```

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.

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

Remarks

If *geometry-expression* is already known to be an ST_MultiPolygon value, it is more efficient to use TREAT(*geometry-expression* AS ST_MultiPolygon) than the ST_ToMultiPolygon method.

Note
By default, ST_ToMultiPolygon uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_ToMultiSurface method for type ST_Geometry"
- "ST_ToGeomColl method for type ST_Geometry"
- "ST_ToPolygon method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.33

Example

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
```

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.

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

Remarks

If *geometry-expression* is already known to be an ST_MultiSurface value, it is more efficient to use TREAT(*geometry-expression* AS ST_MultiSurface) than the ST_ToMultiSurface method.

Note
By default, ST_ToMultiSurface uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_ToMultiPolygon method for type ST_Geometry"
- "ST_ToGeomColl method for type ST_Geometry"
- "ST_ToSurface method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.33

Example

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
```

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.

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

Remarks

Converts the geometry to a point. The logic is equivalent to that used for **CAST(** *geometry-expression* **AS ST_Point)**.

If *geometry-expression* is already known to be an ST_Point value, it is more efficient to use TREAT(*geometry-expression* AS ST_Point) than the ST_ToPoint method.

Note
By default, ST_ToPoint uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_ToMultiPoint method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.33

Example

The following example returns the result `Point (1 2)`.

```
SELECT NEW ST_GeomCollection( NEW ST_Point(1,2) ).ST_ToPoint()
```

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.

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

Remarks

Convert the geometry to a polygon. The logic is equivalent to that used for **CAST(** *geometry-expression* **AS ST_Polygon)**. If the *geometry-expression* is a curve polygon, it is interpolated using ST_CurvePolyToPoly().

If *geometry-expression* is already known to be an ST_Polygon value, it is more efficient to use TREAT(*geometry-expression* AS ST_Polygon) than the ST_ToPolygon method.

Note
By default, ST_ToPolygon uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_ToCurvePoly method for type ST_Geometry"
- "ST_ToSurface method for type ST_Geometry"
- "ST_ToMultiPolygon method for type ST_Geometry"
- "ST_CurvePolyToPoly method for type ST_CurvePolygon"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.33

Example

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
```

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.

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

Remarks

Converts the geometry to a surface. The logic is equivalent to that used for **CAST(** *geometry-expression* **AS ST_Surface)**.

If *geometry-expression* is already known to be an ST_Surface value, it is more efficient to use TREAT(*geometry-expression* AS ST_Surface) than the ST_ToSurface method.

Note
By default, ST_ToSurface uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_ToCurvePoly method for type ST_Geometry"
- "ST_ToPolygon method for type ST_Geometry"
- "ST_ToMultiSurface method for type ST_Geometry"

Standards and compatibility

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

Example

The following example returns the result `Polygon EMPTY`.

```
SELECT NEW ST_GeomCollection().ST_ToSurface()
```

ST_Touches method

Tests if a geometry value spatially touches another geometry value.

Syntax

geometry-expression.**ST_Touches**(*geo2*)

Parameters

Name	Type	Description
geo2	ST_Geometry	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.

Remarks

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

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

Note
This method can not be used with geometries in round-Earth spatial reference system.

See also

- "ST_Intersects method for type ST_Geometry"
- "ST_Boundary method for type ST_Geometry"
- "ST_Dimension method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.28

Example

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
```

ST_Transform method

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

Syntax

geometry-expression.**ST_Transform**(*srid*)

Parameters

Name	Type	Description
srid	INT	The SRID of the result.

Returns

- **ST_Geometry** Returns a copy of the geometry value transformed into the specified spatial reference system.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_Transform method transforms *geometry-expression* from its spatial reference system to the specified spatial reference system using the transform definition of both spatial reference systems. The transformation is performed using the PROJ.4 library.

ST_Transform is required to move between different coordinate systems. For example, use can use ST_Transform to transform a geometry which uses latitude and longitude to a geometry with the SRID 3310 "NAD83 / California Albers". The "NAD83 / California Albers" spatial reference system is a planar projection for California data which uses the Albers projection algorithm and metres for its linear unit of measure.

Transformations from a lat/long system to a Cartesian system can be problematic for polar points. If the database server is unable to transform a point close to the North or South pole, the latitude value of the point is shifted a small distance (slightly more than 1e-10 radians) away from the pole, and along the same longitude, so that the transformation can succeed.

If you are transforming a geometry between two spatial reference systems that have the same coordinate system, you can use the ST_SRID method instead of ST_Transform.

The spatial tutorial includes steps showing you how to transforming data between spatial reference systems. See "Tutorial: Experimenting with the spatial features" on page 55.

See also

- "ST_SRID method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.6

Example

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" [*SQL Anywhere Server - SQL Reference*].

```
SELECT NEW ST_Point( -118, 34, 4326 ).ST_Transform( 3310 )
```

ST_Union method

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

Syntax

geometry-expression.**ST_Union**(*geo2*)

Parameters

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

Returns

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

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

Remarks

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

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

See also

- "ST_Difference method for type ST_Geometry"
- "ST_Intersection method for type ST_Geometry"
- "ST_SymDifference method for type ST_Geometry"
- "ST_UnionAggr method for type ST_Geometry"

Standards and compatibility

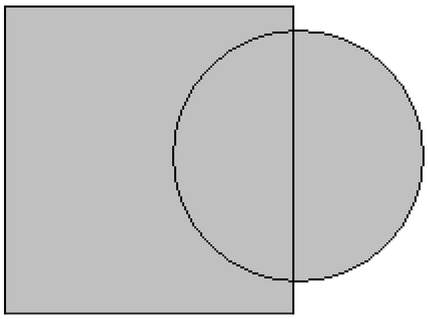
- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.19

Example

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.



ST_UnionAggr method

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

Syntax

ST_Geometry::**ST_UnionAggr**(*geometry-column*)

Parameters

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

Returns

- **ST_Geometry** Returns a geometry that is the spatial union for all the geometries in a group.

  The spatial reference system identifier of the result is the same as that for the first parameter.

Remarks

Rows where the argument is NULL are not included.

Returns NULL for an empty group or a group containing no non-NULL values.

If the group contains a single non-NULL geometry, it is returned. Otherwise, the union is logically computed by repeatedly applying the ST_Union method to combine two geometries at a time. See "ST_Union method for type ST_Geometry" on page 223.

See also

- "ST_Union method for type ST_Geometry"

Standards and compatibility

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

Example

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 )
```

ST_Within method

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

Syntax

geometry-expression.**ST_Within**(*geo2*)

Parameters

Name	Type	Description
geo2	ST_Geometry	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.

Remarks

The ST_Within method tests if the *geometry-expression* is completely within *geo2* and there is one or more interior points of *geo2* that lies in the interior of the *geometry-expression*.

geometry-expression.ST_Within(*geo2*) is equivalent to *geo2*.ST_Contains(*geometry-expression*).

The ST_Within and ST_CoveredBy methods are similar. The difference is that ST_CoveredBy does not require intersecting interior points.

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

Note
This method can not be used with geometries in round-Earth spatial reference system.

See also

- "ST_Contains method for type ST_Geometry"
- "ST_CoveredBy method for type ST_Geometry"
- "ST_Intersects method for type ST_Geometry"
- "ST_WithinFilter method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.30

Example

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

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

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
```

ST_WithinDistance method

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

Syntax

geometry-expression.**ST_WithinDistance**(*geo2*,*distance*[, *unit-name*])

Parameters

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

Returns

- **BIT** Returns 1 if *geometry-expression* and *geo2* are within the specified distance of each other, otherwise 0.

Remarks

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

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

For planar spatial reference systems, the distance is calculated as the Cartesian distance within the plane, computed in the linear units of measure for the associated spatial reference system. For round-Earth spatial reference systems, the distance is computed taking the curvature of the Earth's surface into account using the ellipsoid parameters in the spatial reference system definition.

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

Note
For round-Earth spatial reference systems, the ST_WithinDistance method is only supported if *geometry-expression* and *geo2* contain only points.

See also

- "ST_Distance method for type ST_Geometry"
- "ST_WithinDistanceFilter method for type ST_Geometry"
- "ST_Intersects method for type ST_Geometry"

Standards and compatibility

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

Example

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. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

```
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

ST_WithinDistanceFilter method

An inexpensive of whether two geometries might be within a specified distance of each other.

Syntax

geometry-expression.**ST_WithinDistanceFilter**(*geo2*,*distance*[, *unit-name*])

Parameters

Name	Type	Description
geo2	ST_Geometry	The other geometry value whose distance is to be measured from the *geometry-expression*.
distance	DOUBLE	The distance the two geometries should be within.
unit-name	VARCHAR(128)	The units in which the distance parameter should be interpreted. 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.

Remarks

The ST_WithinDistanceFilter method provides an efficient test to determine if two geometries might be within a specified distance of each other (as determined by the ST_WithinDistance method). Returns 1 if the *geometry-expression* might be within the given distance of *geo2*, otherwise it returns 0.

This test is less expensive than ST_WithinDistance, but may return 1 in some cases where the smallest distance between the two geometries is actually larger than the specified distance. Therefore, this method can be useful as a primary filter when further processing will determine the true distance between the geometries.

The implementation of ST_WithinDistanceFilter relies upon metadata associated with the stored geometries. Because the available metadata may change between server versions, depending upon how the data is loaded, or where ST_WithinDistanceFilter is used within a query, the expression *geometry-expression*.ST_WithinDistanceFilter(*geo2*, *distance* [, *unit_name*]) can return different results when *geometry-expression* is not within the specified distance of *geo2*. Whenever *geometry-expression* is within the specified distance of *geo2*, ST_WithinDistanceFilter always returns 1.

Note
By default, ST_WithinDistanceFilter uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_Distance method for type ST_Geometry"
- "ST_WithinDistance method for type ST_Geometry"
- "ST_IntersectsFilter method for type ST_Geometry"

Standards and compatibility

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

Example

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. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

```
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

ST_WithinFilter method

An inexpensive test if a geometry might be within another.

Syntax

geometry-expression.**ST_WithinFilter**(*geo2*)

Parameters

Name	Type	Description
geo2	ST_Geometry	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.

Remarks

The ST_WithinFilter method provides an efficient test to determine if one geometry might be within the other. Returns 1 if the *geometry-expression* might be within *geo2*, otherwise 0.

This test is less expensive than ST_Within, but may return 1 in some cases where the *geometry-expression* is not actually spatially within *geo2*.

Therefore, this method can be useful as a primary filter when further processing will determine if geometries interact in the desired way.

The implementation of ST_WithinFilter relies upon metadata associated with the stored geometries. Because the available metadata may change between server versions, depending upon how the data is loaded, or where ST_WithinFilter is used within a query, the expression *geometry-expression*.ST_WithinFilter(*geo2*) can return different results when *geometry-expression* is not within *geo2*. Whenever *geometry-expression* is within *geo2*, ST_WithinFilter always returns 1.

Note
This method can not be used with geometries in round-Earth spatial reference system.

See also

- "ST_Within method for type ST_Geometry"

Standards and compatibility

- **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*.

Remarks

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

Note that for geographic spatial reference systems, the returned value corresponds to the first coordinate in the axis order. If the axis order is lat/lon/a/m, the minimum value corresponds to the western boundary of the *geometry-expression* as returned by ST_LongWest and the maximum value corresponds to the eastern boundary of *geometry-expression* as returned by ST_LongEast. In round-Earth model, this means that if *geometry-expression* crosses the date line, the minimum value will be greater than the maximum value. If the axis order is lon/lat/z/m, the minimum value corresponds to southernmost point of *geometry-expression* as returned by ST_LatSouth and the maximum value corresponds to the northernmost point of *geometry-expression* as returned by ST_LatNorth.

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

Note
By default, ST_XMax uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_XMin method for type ST_Geometry"
- "ST_YMin method for type ST_Geometry"
- "ST_YMax method for type ST_Geometry"
- "ST_ZMin method for type ST_Geometry"
- "ST_ZMax method for type ST_Geometry"
- "ST_MMin method for type ST_Geometry"
- "ST_MMax method for type ST_Geometry"
- "ST_LongEast method for type ST_Geometry"

Standards and compatibility

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

Example

The following example returns the result 5.

```
SELECT NEW ST_LineString( 'LineString ZM( 1 2 3 4, 5 6 7 8 )' ).ST_XMax()
```

ST_XMin method

Retrieves the minimum X coordinate value of a geometry.

Syntax

geometry-expression.**ST_XMin**()

Returns

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

Remarks

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

Note that for geographic spatial reference systems, the returned value corresponds to the first coordinate in the axis order. If the axis order is lat/lon/a/m, the minimum value corresponds to the western boundary of the *geometry-expression* as returned by ST_LongWest and the maximum value corresponds to the eastern boundary of *geometry-expression* as returned by ST_LongEast. In round-Earth model, this means that if *geometry-expression* crosses the date line, the minimum value will be greater than the maximum value. If the axis order is lon/lat/z/m, the minimum value corresponds to southernmost point of *geometry-expression* as returned by ST_LatSouth and the maximum value corresponds to the northernmost point of *geometry-expression* as returned by ST_LatNorth.

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

Note
By default, ST_XMin uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_XMax method for type ST_Geometry"
- "ST_YMin method for type ST_Geometry"
- "ST_YMax method for type ST_Geometry"
- "ST_ZMin method for type ST_Geometry"
- "ST_ZMax method for type ST_Geometry"
- "ST_MMin method for type ST_Geometry"
- "ST_MMax method for type ST_Geometry"
- "ST_LongWest method for type ST_Geometry"

Standards and compatibility

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

Example

The following example returns the result `1`.

```
SELECT NEW ST_LineString( 'LineString ZM( 1 2 3 4, 5 6 7 8 )' ).ST_XMin()
```

ST_YMax method

Retrieves the maximum Y coordinate value of a geometry.

Syntax

geometry-expression.**ST_YMax**()

Returns

- **DOUBLE** Returns the maximum Y coordinate value of the *geometry-expression*.

Remarks

Returns the maximum Y coordinate value of the *geometry-expression*. This is computed by comparing the Y attribute of all points in the geometry.

Note that for geographic spatial reference systems, the returned value corresponds to the first coordinate in the axis order. If the axis order is lon/lat/z/m, the minimum value corresponds to southernmost point of *geometry-expression* as returned by ST_LatSouth and the maximum value corresponds to the northernmost point of *geometry-expression* as returned by ST_LatNorth. If the axis order is lat/lon/a/m, the minimum value corresponds to the western boundary of *geometry-expression* as returned by ST_LongWest and the maximum value corresponds to the eastern boundary of the *geometry-expression* as returned by ST_LongEast. In round-Earth model, this means that if *geometry-expression* crosses the date line, the minimum value will be greater than the maximum value.

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

Note
By default, ST_YMax uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_XMin method for type ST_Geometry"
- "ST_XMax method for type ST_Geometry"
- "ST_YMin method for type ST_Geometry"
- "ST_ZMin method for type ST_Geometry"
- "ST_ZMax method for type ST_Geometry"
- "ST_MMin method for type ST_Geometry"
- "ST_MMax method for type ST_Geometry"
- "ST_LatNorth method for type ST_Geometry"

Standards and compatibility

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

Example

The following example returns the result 6.

```
SELECT NEW ST_LineString( 'LineString ZM( 1 2 3 4, 5 6 7 8 )' ).ST_YMax()
```

ST_YMin method

Retrieves the minimum Y coordinate value of a geometry.

Syntax

geometry-expression.**ST_YMin**()

Returns

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

Remarks

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

Note that for geographic spatial reference systems, the returned value corresponds to the first coordinate in the axis order. If the axis order is lon/lat/z/m, the minimum value corresponds to southernmost point of *geometry-expression* as returned by ST_LatSouth and the maximum value corresponds to the northernmost point of *geometry-expression* as returned by ST_LatNorth. If the axis order is lat/lon/a/m, the minimum value corresponds to the western boundary of *geometry-expression* as returned by ST_LongWest and the maximum value corresponds to the eastern boundary of the *geometry-expression* as returned by ST_LongEast. In round-Earth model, this means that if *geometry-expression* crosses the date line, the minimum value will be greater than the maximum value.

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

Note
By default, ST_YMin uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_XMin method for type ST_Geometry"
- "ST_XMax method for type ST_Geometry"
- "ST_YMax method for type ST_Geometry"
- "ST_ZMin method for type ST_Geometry"
- "ST_ZMax method for type ST_Geometry"
- "ST_MMin method for type ST_Geometry"
- "ST_MMax method for type ST_Geometry"
- "ST_LatSouth method for type ST_Geometry"

Standards and compatibility

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

Example

The following example returns the result 2.

```
SELECT NEW ST_LineString( 'LineString ZM( 1 2 3 4, 5 6 7 8 )' ).ST_YMin()
```

ST_ZMax method

Retrieves the maximum Z coordinate value of a geometry.

Syntax

geometry-expression.**ST_ZMax**()

Returns

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

Remarks

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

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

Note
By default, ST_ZMax uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_XMin method for type ST_Geometry"
- "ST_XMax method for type ST_Geometry"
- "ST_YMin method for type ST_Geometry"
- "ST_YMax method for type ST_Geometry"
- "ST_ZMin method for type ST_Geometry"
- "ST_MMin method for type ST_Geometry"
- "ST_MMax method for type ST_Geometry"

Standards and compatibility

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

Example

The following example returns the result 7.

```
SELECT NEW ST_LineString( 'LineString ZM( 1 2 3 4, 5 6 7 8 )' ).ST_ZMax()
```

ST_ZMin method

Retrieves the minimum Z coordinate value of a geometry.

Syntax

geometry-expression.**ST_ZMin**()

Returns

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

Remarks

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

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

Note
By default, ST_ZMin uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_XMin method for type ST_Geometry"
- "ST_XMax method for type ST_Geometry"
- "ST_YMin method for type ST_Geometry"
- "ST_YMax method for type ST_Geometry"
- "ST_ZMax method for type ST_Geometry"
- "ST_MMin method for type ST_Geometry"
- "ST_MMax method for type ST_Geometry"

Standards and compatibility

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

Example

The following example returns the result 3.

```
SELECT NEW ST_LineString( 'LineString ZM( 1 2 3 4, 5 6 7 8 )' ).ST_ZMin()
```

ST_LineString type

The ST_LineString type is a subtype of ST_Curve that uses straight line segments between control points.

Direct supertype

- "ST_Curve type"

Constructor

- "ST_LineString constructor"

Methods

- Methods of ST_LineString:

ST_LineStringAggr	ST_NumPoints	ST_PointN

- All methods of "ST_Curve type" can also be called on a ST_LineString type:

ST_CurveToLine	ST_EndPoint	ST_IsClosed	ST_IsRing
ST_Length	ST_StartPoint		

- All methods of "ST_Geometry type" can also be called on a ST_LineString type:

ST_Affine	ST_AsBinary	ST_AsGML	ST_AsGeoJSON
ST_AsKML	ST_AsSVG	ST_AsSVGAggr	ST_AsText
ST_AsWKB	ST_AsWKT	ST_AsXML	ST_Boundary
ST_Contains	ST_ContainsFilter	ST_ConvexHull	ST_ConvexHullAggr
ST_CoordDim	ST_CoveredBy	ST_CoveredByFilter	ST_Covers
ST_CoversFilter	ST_Crosses	ST_Difference	ST_Dimension
ST_Disjoint	ST_Distance	ST_Envelope	ST_EnvelopeAggr
ST_Equals	ST_EqualsFilter	ST_GeomFromBinary	ST_GeomFromShape
ST_GeomFromText	ST_GeomFromWKB	ST_GeomFromWKT	ST_GeometryType
ST_GeometryType-FromBaseType	ST_Intersection	ST_IntersectionAggr	ST_Intersects
ST_IntersectsFilter	ST_IntersectsRect	ST_Is3D	ST_IsEmpty
ST_IsMeasured	ST_IsSimple	ST_IsValid	ST_LatNorth
ST_LatSouth	ST_LinearHash	ST_LinearUnHash	ST_LoadConfigura-tionData
ST_LongEast	ST_LongWest	ST_MMax	ST_MMin
ST_OrderingEquals	ST_Overlaps	ST_Relate	ST_Reverse
ST_SRID	ST_SRIDFromBase-Type	ST_SnapToGrid	ST_SymDifference
ST_ToCircular	ST_ToCompound	ST_ToCurve	ST_ToCurvePoly
ST_ToGeomColl	ST_ToLineString	ST_ToMultiCurve	ST_ToMultiLine
ST_ToMultiPoint	ST_ToMultiPolygon	ST_ToMultiSurface	ST_ToPoint
ST_ToPolygon	ST_ToSurface	ST_Touches	ST_Transform
ST_Union	ST_UnionAggr	ST_Within	ST_WithinDistance
ST_WithinDistanceFil-ter	ST_WithinFilter	ST_XMax	ST_XMin

ST_YMax	ST_YMin	ST_ZMax	ST_ZMin

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 constructor

Constructs a linestring.

Overload list

Name	Description
"ST_LineString() constructor"	Constructs a linestring representing the empty set.
"ST_LineString(LONG VARCHAR[, INT]) constructor"	Constructs a linestring from a text representation.
"ST_LineString(LONG BINARY[, INT]) constructor"	Constructs a linestring from Well Known Binary (WKB).
"ST_LineString(ST_Point,ST_Point,...) constructor"	Constructs a linestring value from a list of points in a specified spatial reference system.

ST_LineString() constructor

Constructs a linestring representing the empty set.

Syntax

NEW ST_LineString()

Standards and compatibility

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

Example

The following returns 1, indicating the value is empty.

```
SELECT NEW ST_LineString().ST_IsEmpty()
```

ST_LineString(LONG VARCHAR[, INT]) constructor

Constructs a linestring from a text representation.

Syntax

NEW ST_LineString(*text-representation*[, *srid*])

Parameters

Name	Type	Description
text-repre-sentation	LONG VAR-CHAR	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	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a linestring from a character string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 7.2.2

Example

The following returns LineString (0 0, 5 10).

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

ST_LineString(LONG BINARY[, INT]) constructor

Constructs a linestring from Well Known Binary (WKB).

Syntax

NEW ST_LineString(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BI-NARY	A string containing the binary representation of an linestring. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
srid	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a linestring from a binary string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 7.2.2

Example

The following returns LineString (0 0, 5 10).

```
SELECT NEW
ST_LineString(0x0102000000020000000000000000000000000000000000000000000000000000
014400000000000002440)
```

ST_LineString(ST_Point,ST_Point,...) constructor

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

Syntax

NEW ST_LineString(*pt1*,*pt2*[,*pt3*,...,*ptN*])

Parameters

Name	Type	Description
pt1	ST_Point	The first point of the linestring.
pt2	ST_Point	The second point of the linestring.
pt3,...,ptN	ST_Point	Additional points of the linestring.

Remarks

Constructs a linestring value from a list of points. All of the points must have the same SRID. The resulting linestring is constructed with this common SRID. All of the supplied points must be non-empty and have the same answer for Is3D and IsMeasured. The linestring is 3D if all of the points are 3D, and the linestring is measured if all of the points are measured.

Note
By default, ST_LineString uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see “STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement” [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

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) )
```

ST_LineStringAggr method

Returns a linestring built from the ordered points in a group.

Syntax

ST_LineString::**ST_LineStringAggr**(*point*[**ORDER BY** *order-by-expression* [**ASC** | **DESC**], ...])

Parameters

Name	Type	Description
point	ST_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.

  The spatial reference system identifier of the result is the same as that for the first parameter.

Remarks

The ST_LineStringAggr aggregate method can be used to build a linestring out of a group of ordered points. All of the geometry columns to be combined must have the same SRID. All of the points to be combined must be non-empty with the same coordinate dimension.

Rows where the point is NULL are not included.

Returns NULL for an empty group or a group containing no non-NULL values.

The resulting linestring has the same coordinate dimension as each point.

Note
The ORDER BY clause should be specified to control the order of points within the linestring. If not present, the order of points in the linestring will vary depending on the access plan selected by the query optimizer.

> **Note**
> By default, ST_LineStringAggr uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

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
```

ST_NumPoints method

Returns the number of points defining the linestring.

> **Note**
> By default, ST_NumPoints uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

linestring-expression.**ST_NumPoints**()

Returns

- **INT** Returns NULL if the linestring value is empty, otherwise the number of points in the value.

See also

- "ST_PointN method for type ST_LineString"
- "ST_NumPoints method for type ST_CircularString"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 7.2.4

Example

The following example returns the result 3.

```
SELECT TREAT( Shape AS ST_LineString ).ST_NumPoints()
FROM SpatialShapes WHERE ShapeID = 5
```

ST_PointN method

Returns the *n*th point in the linestring.

Note
By default, ST_PointN uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

linestring-expression.**ST_PointN**(*n*)

Parameters

Name	Type	Description
n	INT	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.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *linestring-expression*.

See also

- "ST_NumPoints method for type ST_LineString"
- "ST_PointN method for type ST_CircularString"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 7.2.5

Example

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)

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.

Direct supertype

- "ST_GeomCollection type"

Direct subtypes

- "ST_MultiLineString type"

Constructor

- "ST_MultiCurve constructor"

Methods

- Methods of ST_MultiCurve:

ST_IsClosed	ST_Length	ST_MultiCurveAggr

- All methods of "ST_GeomCollection type" can also be called on a ST_MultiCurve type:

ST_GeomCollectionAggr	ST_GeometryN	ST_NumGeometries

- All methods of "ST_Geometry type" can also be called on a ST_MultiCurve type:

ST_Affine	ST_AsBinary	ST_AsGML	ST_AsGeoJSON

ST_AsKML	ST_AsSVG	ST_AsSVGAggr	ST_AsText
ST_AsWKB	ST_AsWKT	ST_AsXML	ST_Boundary
ST_Contains	ST_ContainsFilter	ST_ConvexHull	ST_ConvexHullAggr
ST_CoordDim	ST_CoveredBy	ST_CoveredByFilter	ST_Covers
ST_CoversFilter	ST_Crosses	ST_Difference	ST_Dimension
ST_Disjoint	ST_Distance	ST_Envelope	ST_EnvelopeAggr
ST_Equals	ST_EqualsFilter	ST_GeomFromBinary	ST_GeomFromShape
ST_GeomFromText	ST_GeomFromWKB	ST_GeomFromWKT	ST_GeometryType
ST_GeometryType-FromBaseType	ST_Intersection	ST_IntersectionAggr	ST_Intersects
ST_IntersectsFilter	ST_IntersectsRect	ST_Is3D	ST_IsEmpty
ST_IsMeasured	ST_IsSimple	ST_IsValid	ST_LatNorth
ST_LatSouth	ST_LinearHash	ST_LinearUnHash	ST_LoadConfigura-tionData
ST_LongEast	ST_LongWest	ST_MMax	ST_MMin
ST_OrderingEquals	ST_Overlaps	ST_Relate	ST_Reverse
ST_SRID	ST_SRIDFromBase-Type	ST_SnapToGrid	ST_SymDifference
ST_ToCircular	ST_ToCompound	ST_ToCurve	ST_ToCurvePoly
ST_ToGeomColl	ST_ToLineString	ST_ToMultiCurve	ST_ToMultiLine
ST_ToMultiPoint	ST_ToMultiPolygon	ST_ToMultiSurface	ST_ToPoint
ST_ToPolygon	ST_ToSurface	ST_Touches	ST_Transform
ST_Union	ST_UnionAggr	ST_Within	ST_WithinDistance
ST_WithinDistanceFil-ter	ST_WithinFilter	ST_XMax	ST_XMin
ST_YMax	ST_YMin	ST_ZMax	ST_ZMin

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.3

ST_MultiCurve constructor

Constructs a multi curve.

Overload list

Name	Description
"ST_MultiCurve() constructor"	Constructs a multi curve representing the empty set.
"ST_MultiCurve(LONG VARCHAR[, INT]) constructor"	Constructs a multi curve from a text representation.
"ST_MultiCurve(LONG BINARY[, INT]) constructor"	Constructs a multi curve from Well Known Binary (WKB).
"ST_MultiCurve(ST_Curve,...) constructor"	Constructs a multi-curve from a list of curve values.

ST_MultiCurve() constructor

Constructs a multi curve representing the empty set.

Syntax

NEW ST_MultiCurve()

Standards and compatibility

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

Example

The following returns 1, indicating the value is empty.

```
SELECT NEW ST_MultiCurve().ST_IsEmpty()
```

ST_MultiCurve(LONG VARCHAR[, INT]) constructor

Constructs a multi curve from a text representation.

Syntax

NEW ST_MultiCurve(*text-representation*[, *srid*])

Parameters

Name	Type	Description
text-representation	LONG VARCHAR	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	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a multi curve from a character string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.3.2

Example

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))')
```

ST_MultiCurve(LONG BINARY[, INT]) constructor

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

Syntax

NEW ST_MultiCurve(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	A string containing the binary representation of an multi curve. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
srid	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a multi curve from a binary string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.3.2

Example

The following returns MultiCurve (CircularString (5 10, 10 12, 15 10)).

```
SELECT NEW
ST_MultiCurve(0x010b000000010000000108000000030000000000000000000000144000000000000000
0002440000000000000002440000000000000002840000000000000002e400000000000002440)
```

ST_MultiCurve(ST_Curve,...) constructor

Constructs a multi-curve from a list of curve values.

Syntax

NEW ST_MultiCurve(*curve1*[,*curve2*,...,*curveN*])

Parameters

Name	Type	Description
curve1	ST_Curve	The first curve value of the multi-curve.
curve2,...,curveN	ST_Curve	Additional curve values of the multi-curve.

Remarks

Constructs a multi-curve from a list of curve values. All of the supplied curve values must have the same SRID, and the multi-curve is constructed with this common SRID.

All of the supplied curve values must have the same answer for Is3D and IsMeasured. The multi-curve is 3D if all of the curve values are 3D, and the multi-curve is measured if all of the curve values are measured.

Note
By default, ST_MultiCurve uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

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)' ) )
```

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.

Note
By default, ST_IsClosed uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

multicurve-expression.**ST_IsClosed**()

Returns

- **BIT** Returns 1 if the multicurve is closed, otherwise 0.

See also

- "ST_IsClosed method for type ST_Curve"
- "ST_Boundary method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.3.3

Example

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
```

ST_Length 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**([*unit-name*])

Parameters

Name	Type	Description
unit-name	VARCHAR(128)	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.

Remarks

The ST_Length method returns the length of a multicurve in the units identified by the *unit-name* parameter. The length of a multicurve is the sum of the lengths of the contained curves. If the curve is empty, then NULL is returned.

If the curve contains Z values, these are not considered when computing the length of the geometry.

Note
If the *multicurve-expression* is an empty geometry (ST_IsEmpty()=1), then this method returns NULL.

Note
By default, ST_Length uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_Length method for type ST_Curve"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.3.4

Example

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
```

ST_MultiCurveAggr method

Returns a multicurve containing all of the curves in a group.

Syntax

ST_MultiCurve::**ST_MultiCurveAggr**(*geometry-column*[**ORDER BY** *order-by-expression* [**ASC** | **DESC**], ...])

Parameters

Name	Type	Description
geometry-column	ST_Curve	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.

  The spatial reference system identifier of the result is the same as that for the first parameter.

Remarks

The ST_MultiCurveAggr aggregate function can be used to combine a group of curves into a single collection. All of the geometries to be combined must have both the same SRID and the same coordinate dimension.

Rows where the argument is NULL are not included.

Returns NULL for an empty group or a group containing no non-NULL values.

The resulting ST_MultiCurve has the same coordinate dimension as each curves.

The optional ORDER BY clause can be used to arrange the elements in a particular order so that ST_GeometryN returns them in the desired order. If this ordering is not relevant, it is more efficient to not specify an ordering. In that case, the ordering of elements depends on the access plan selected by the query optimizer.

ST_MultiCurveAggr is more efficient than ST_UnionAggr, but ST_MultiCurveAggr can return a collection with duplicate or overlapping curves if they exist in the group of curves. ST_UnionAggr handles duplicate and overlapping geometries.

Note
By default, ST_MultiCurveAggr uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see “STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement” [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_UnionAggr method for type ST_Geometry"

Standards and compatibility

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

Example

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 )
```

ST_MultiLineString type

An ST_MultiLineString is a collection of zero or more ST_LineString values, and all of the linestrings are within the spatial reference system.

Direct supertype

- "ST_MultiCurve type"

Constructor

- "ST_MultiLineString constructor"

Methods

- Methods of ST_MultiLineString:

ST_MultiLineStringAggr

- All methods of "ST_MultiCurve type" can also be called on a ST_MultiLineString type:

ST_IsClosed	ST_Length	ST_MultiCurveAggr

- All methods of "ST_GeomCollection type" can also be called on a ST_MultiLineString type:

ST_GeomCollectionAggr	ST_GeometryN	ST_NumGeometries

- All methods of "ST_Geometry type" can also be called on a ST_MultiLineString type:

ST_Affine	ST_AsBinary	ST_AsGML	ST_AsGeoJSON
ST_AsKML	ST_AsSVG	ST_AsSVGAggr	ST_AsText

ST_AsWKB	ST_AsWKT	ST_AsXML	ST_Boundary
ST_Contains	ST_ContainsFilter	ST_ConvexHull	ST_ConvexHullAggr
ST_CoordDim	ST_CoveredBy	ST_CoveredByFilter	ST_Covers
ST_CoversFilter	ST_Crosses	ST_Difference	ST_Dimension
ST_Disjoint	ST_Distance	ST_Envelope	ST_EnvelopeAggr
ST_Equals	ST_EqualsFilter	ST_GeomFromBinary	ST_GeomFromShape
ST_GeomFromText	ST_GeomFromWKB	ST_GeomFromWKT	ST_GeometryType
ST_GeometryType-FromBaseType	ST_Intersection	ST_IntersectionAggr	ST_Intersects
ST_IntersectsFilter	ST_IntersectsRect	ST_Is3D	ST_IsEmpty
ST_IsMeasured	ST_IsSimple	ST_IsValid	ST_LatNorth
ST_LatSouth	ST_LinearHash	ST_LinearUnHash	ST_LoadConfigura-tionData
ST_LongEast	ST_LongWest	ST_MMax	ST_MMin
ST_OrderingEquals	ST_Overlaps	ST_Relate	ST_Reverse
ST_SRID	ST_SRIDFromBase-Type	ST_SnapToGrid	ST_SymDifference
ST_ToCircular	ST_ToCompound	ST_ToCurve	ST_ToCurvePoly
ST_ToGeomColl	ST_ToLineString	ST_ToMultiCurve	ST_ToMultiLine
ST_ToMultiPoint	ST_ToMultiPolygon	ST_ToMultiSurface	ST_ToPoint
ST_ToPolygon	ST_ToSurface	ST_Touches	ST_Transform
ST_Union	ST_UnionAggr	ST_Within	ST_WithinDistance
ST_WithinDistanceFil-ter	ST_WithinFilter	ST_XMax	ST_XMin
ST_YMax	ST_YMin	ST_ZMax	ST_ZMin

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.4

ST_MultiLineString constructor

Constructs a multi linestring.

Overload list

Name	Description
"ST_MultiLineString() constructor"	Constructs a multi linestring representing the empty set.
"ST_MultiLineString(LONG VARCHAR[, INT]) constructor"	Constructs a multi linestring from a text representation.
"ST_MultiLineString(LONG BINARY[, INT]) constructor"	Constructs a multi linestring from Well Known Binary (WKB).
"ST_MultiLineString(ST_LineString,...) constructor"	Constructs a multi-linestring from a list of linestring values.

ST_MultiLineString() constructor

Constructs a multi linestring representing the empty set.

Syntax

NEW ST_MultiLineString()

Standards and compatibility

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

Example

The following returns 1, indicating the value is empty.

```
SELECT NEW ST_MultiLineString().ST_IsEmpty()
```

ST_MultiLineString(LONG VARCHAR[, INT]) constructor

Constructs a multi linestring from a text representation.

Syntax

NEW ST_MultiLineString(*text-representation*[, *srid*])

Parameters

Name	Type	Description
text-representation	LONG VARCHAR	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	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a multi linestring from a character string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.4.2

Example

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))')
```

ST_MultiLineString(LONG BINARY[, INT]) constructor

Constructs a multi linestring from Well Known Binary (WKB).

Syntax

NEW ST_MultiLineString(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	A string containing the binary representation of an multi linestring. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
srid	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a multi linestring from a binary string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.4.2

Example

The following returns MultiLineString ((10 10, 12 12)).

```
SELECT NEW
ST_MultiLineString(0x01050000000100000001020000000200000000000000000000244000000
00000000244000000000000028400000000000002840)
```

ST_MultiLineString(ST_LineString,...) constructor

Constructs a multi-linestring from a list of linestring values.

Syntax

NEW ST_MultiLineString(*linestring1*[,*linestring2*,...,*linestringN*])

Parameters

Name	Type	Description
linestring1	ST_LineString	The first linestring value of the multi-linestring.
linestring2,...,linestringN	ST_LineString	Additional linestring values of the multi-linestring.

Remarks

Constructs a multi-linestring from a list of linestring values. All of the supplied linestring values must have the same SRID, and the multi-linestring is constructed with this common SRID.

All of the supplied linestring values must have the same answer for Is3D and IsMeasured. The multi-linestring is 3D if all of the linestring values are 3D, and the multi-linestring is measured if all of the linestring values are measured.

Note
By default, ST_MultiLineString uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

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)' ) )
```

ST_MultiLineStringAggr method

Returns a multilinestring containing all of the linestrings in a group.

Syntax

ST_MultiLineString::**ST_MultiLineStringAggr**(*geometry-column*[**ORDER BY** *order-by-expression* [**ASC** | **DESC**], ...])

Parameters

Name	Type	Description
geometry-column	ST_LineString	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.

  The spatial reference system identifier of the result is the same as that for the first parameter.

Remarks

The ST_MultiLineStringAggr aggregate function can be used to combine a group of linestrings into a single collection. All of the geometries to be combined must have both the same SRID and the same coordinate dimension.

Rows where the argument is NULL are not included.

Returns NULL for an empty group or a group containing no non-NULL values.

The resulting ST_MultiLineString has the same coordinate dimension as each linestrings.

The optional ORDER BY clause can be used to arrange the elements in a particular order so that ST_GeometryN returns them in the desired order. If this ordering is not relevant, it is more efficient to not specify an ordering. In that case, the ordering of elements depends on the access plan selected by the query optimizer.

ST_MultiLineStringAggr is more efficient than ST_UnionAggr, but ST_MultiLineStringAggr can return a collection with duplicate or overlapping linestrings if they exist in the group of linestrings. ST_UnionAggr handles duplicate and overlapping geometries.

> **Note**
> By default, ST_MultiLineStringAggr uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_UnionAggr method for type ST_Geometry"

Standards and compatibility

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

Example

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 )
```

ST_MultiPoint type

An ST_MultiPoint is a collection of zero or more ST_Point values, and all of the points are within the spatial reference system.

Direct supertype

- "ST_GeomCollection type"

Constructor

- "ST_MultiPoint constructor"

Methods

- Methods of ST_MultiPoint:

ST_MultiPointAggr

- All methods of "ST_GeomCollection type" can also be called on a ST_MultiPoint type:

ST_GeomCollectionAggr	ST_GeometryN	ST_NumGeometries

- All methods of "ST_Geometry type" can also be called on a ST_MultiPoint type:

ST_Affine	ST_AsBinary	ST_AsGML	ST_AsGeoJSON
ST_AsKML	ST_AsSVG	ST_AsSVGAggr	ST_AsText
ST_AsWKB	ST_AsWKT	ST_AsXML	ST_Boundary
ST_Contains	ST_ContainsFilter	ST_ConvexHull	ST_ConvexHullAggr
ST_CoordDim	ST_CoveredBy	ST_CoveredByFilter	ST_Covers
ST_CoversFilter	ST_Crosses	ST_Difference	ST_Dimension
ST_Disjoint	ST_Distance	ST_Envelope	ST_EnvelopeAggr
ST_Equals	ST_EqualsFilter	ST_GeomFromBinary	ST_GeomFromShape
ST_GeomFromText	ST_GeomFromWKB	ST_GeomFromWKT	ST_GeometryType
ST_GeometryType-FromBaseType	ST_Intersection	ST_IntersectionAggr	ST_Intersects
ST_IntersectsFilter	ST_IntersectsRect	ST_Is3D	ST_IsEmpty
ST_IsMeasured	ST_IsSimple	ST_IsValid	ST_LatNorth
ST_LatSouth	ST_LinearHash	ST_LinearUnHash	ST_LoadConfiguration-Data
ST_LongEast	ST_LongWest	ST_MMax	ST_MMin
ST_OrderingEquals	ST_Overlaps	ST_Relate	ST_Reverse
ST_SRID	ST_SRIDFromBase-Type	ST_SnapToGrid	ST_SymDifference
ST_ToCircular	ST_ToCompound	ST_ToCurve	ST_ToCurvePoly
ST_ToGeomColl	ST_ToLineString	ST_ToMultiCurve	ST_ToMultiLine
ST_ToMultiPoint	ST_ToMultiPolygon	ST_ToMultiSurface	ST_ToPoint
ST_ToPolygon	ST_ToSurface	ST_Touches	ST_Transform
ST_Union	ST_UnionAggr	ST_Within	ST_WithinDistance
ST_WithinDistanceFil-ter	ST_WithinFilter	ST_XMax	ST_XMin
ST_YMax	ST_YMin	ST_ZMax	ST_ZMin

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.2

ST_MultiPoint constructor

Constructs a multi point.

Overload list

Name	Description
"ST_MultiPoint() constructor"	Constructs a multi point representing the empty set.
"ST_MultiPoint(LONG VARCHAR[, INT]) constructor"	Constructs a multi point from a text representation.
"ST_MultiPoint(LONG BINARY[, INT]) constructor"	Constructs a multi point from Well Known Binary (WKB).
"ST_MultiPoint(ST_Point,...) constructor"	Constructs a multi-point from a list of point values.

ST_MultiPoint() constructor

Constructs a multi point representing the empty set.

Syntax

NEW ST_MultiPoint()

Standards and compatibility

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

Example

The following returns 1, indicating the value is empty.

```
SELECT NEW ST_MultiPoint().ST_IsEmpty()
```

ST_MultiPoint(LONG VARCHAR[, INT]) constructor

Constructs a multi point from a text representation.

Syntax

NEW ST_MultiPoint(*text-representation*[, *srid*])

Parameters

Name	Type	Description
text-representation	LONG VARCHAR	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	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a multi point from a character string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.2.2

Example

The following returns MultiPoint ((10 10), (12 12), (14 10)).

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

ST_MultiPoint(LONG BINARY[, INT]) constructor

Constructs a multi point from Well Known Binary (WKB).

Syntax

NEW ST_MultiPoint(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	A string containing the binary representation of an multi point. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
srid	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a multi point from a binary string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.2.2

Example

The following returns MultiPoint ((10 10), (12 12), (14 10)).

```
SELECT NEW
ST_MultiPoint(0x0104000000030000000101000000000000000000244000000000000024400
1010000000000000000000028400000000000002840010100000000000000000002c400000000000
002440)
```

ST_MultiPoint(ST_Point,...) constructor

Constructs a multi-point from a list of point values.

Syntax

NEW ST_MultiPoint(*point1*[,*point2*,...,*pointN*])

Parameters

Name	Type	Description
point1	ST_Point	The first point value of the multi-point.
point2,...,pointN	ST_Point	Additional point values of the multi-point.

Remarks

Constructs a multi-point from a list of point values. All of the supplied point values must have the same SRID, and the multi-point is constructed with this common SRID.

All of the supplied point values must have the same answer for Is3D and IsMeasured. The multi-point is 3D if all of the point values are 3D, and the multi-point is measured if all of the point values are measured.

Note
By default, ST_MultiPoint uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

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 ) )
```

ST_MultiPointAggr method

Returns a multipoint containing all of the points in a group.

Syntax

ST_MultiPoint::**ST_MultiPointAggr**(*geometry-column*[**ORDER BY** *order-by-expression* [**ASC** | **DESC**], ...])

Parameters

Name	Type	Description
geometry-column	ST_Point	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.

  The spatial reference system identifier of the result is the same as that for the first parameter.

Remarks

The ST_MultiPointAggr aggregate function can be used to combine a group of points into a single collection. All of the geometries to be combined must have both the same SRID and the same coordinate dimension.

Rows where the argument is NULL are not included.

Returns NULL for an empty group or a group containing no non-NULL values.

The resulting ST_MultiPoint has the same coordinate dimension as each points.

The optional ORDER BY clause can be used to arrange the elements in a particular order so that ST_GeometryN returns them in the desired order. If this ordering is not relevant, it is more efficient to not specify an ordering. In that case, the ordering of elements depends on the access plan selected by the query optimizer.

ST_MultiPointAggr is more efficient than ST_UnionAggr, but ST_MultiPointAggr can return a collection with duplicate or overlapping points if they exist in the group of points. ST_UnionAggr handles duplicate and overlapping geometries.

Note
By default, ST_MultiPointAggr uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see “STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement” [*SQL Anywhere Server - SQL Reference*].

See also

- “ST_UnionAggr method for type ST_Geometry”

Standards and compatibility

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

Example

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 )
```

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.

Direct supertype

- "ST_MultiSurface type"

Constructor

- "ST_MultiPolygon constructor"

Methods

- Methods of ST_MultiPolygon:

ST_MultiPolygonAggr

- All methods of "ST_MultiSurface type" can also be called on a ST_MultiPolygon type:

ST_Area	ST_Centroid	ST_MultiSurfaceAggr	ST_Perimeter
ST_PointOnSurface			

- All methods of "ST_GeomCollection type" can also be called on a ST_MultiPolygon type:

ST_GeomCollectionAggr	ST_GeometryN	ST_NumGeometries

- All methods of "ST_Geometry type" can also be called on a ST_MultiPolygon type:

ST_Affine	ST_AsBinary	ST_AsGML	ST_AsGeoJSON
ST_AsKML	ST_AsSVG	ST_AsSVGAggr	ST_AsText
ST_AsWKB	ST_AsWKT	ST_AsXML	ST_Boundary

ST_Contains	ST_ContainsFilter	ST_ConvexHull	ST_ConvexHullAggr
ST_CoordDim	ST_CoveredBy	ST_CoveredByFilter	ST_Covers
ST_CoversFilter	ST_Crosses	ST_Difference	ST_Dimension
ST_Disjoint	ST_Distance	ST_Envelope	ST_EnvelopeAggr
ST_Equals	ST_EqualsFilter	ST_GeomFromBinary	ST_GeomFromShape
ST_GeomFromText	ST_GeomFromWKB	ST_GeomFromWKT	ST_GeometryType
ST_GeometryType-FromBaseType	ST_Intersection	ST_IntersectionAggr	ST_Intersects
ST_IntersectsFilter	ST_IntersectsRect	ST_Is3D	ST_IsEmpty
ST_IsMeasured	ST_IsSimple	ST_IsValid	ST_LatNorth
ST_LatSouth	ST_LinearHash	ST_LinearUnHash	ST_LoadConfigura-tionData
ST_LongEast	ST_LongWest	ST_MMax	ST_MMin
ST_OrderingEquals	ST_Overlaps	ST_Relate	ST_Reverse
ST_SRID	ST_SRIDFromBase-Type	ST_SnapToGrid	ST_SymDifference
ST_ToCircular	ST_ToCompound	ST_ToCurve	ST_ToCurvePoly
ST_ToGeomColl	ST_ToLineString	ST_ToMultiCurve	ST_ToMultiLine
ST_ToMultiPoint	ST_ToMultiPolygon	ST_ToMultiSurface	ST_ToPoint
ST_ToPolygon	ST_ToSurface	ST_Touches	ST_Transform
ST_Union	ST_UnionAggr	ST_Within	ST_WithinDistance
ST_WithinDistanceFil-ter	ST_WithinFilter	ST_XMax	ST_XMin
ST_YMax	ST_YMin	ST_ZMax	ST_ZMin

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.6

ST_MultiPolygon constructor

Constructs a multi polygon.

Overload list

Name	Description
"ST_MultiPolygon() constructor"	Constructs a multi polygon representing the empty set.
"ST_MultiPolygon(LONG VARCHAR[, INT]) constructor"	Constructs a multi polygon from a text representation.
"ST_MultiPolygon(LONG BINARY[, INT]) constructor"	Constructs a multi polygon from Well Known Binary (WKB).
"ST_MultiPolygon(ST_Polygon,...) constructor"	Constructs a multi-polygon from a list of polygon values.
"ST_MultiPolygon(ST_MultiLineString[, VARCHAR(128)]) constructor"	Creates a multi-polygon from a multilinestring containing exterior rings and an optional list of interior rings.

ST_MultiPolygon() constructor

Constructs a multi polygon representing the empty set.

Syntax

NEW ST_MultiPolygon()

Standards and compatibility

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

Example

The following returns 1, indicating the value is empty.

```
SELECT NEW ST_MultiPolygon().ST_IsEmpty()
```

ST_MultiPolygon(LONG VARCHAR[, INT]) constructor

Constructs a multi polygon from a text representation.

Syntax

NEW ST_MultiPolygon(*text-representation*[, *srid*])

Parameters

Name	Type	Description
text-representation	LONG VARCHAR	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	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a multi polygon from a character string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.6.2

Example

The following returns MultiPolygon (((-5 -5, 5 -5, 0 5, -5 -5), (-2 -2, -2 0, 2 0, 2 -2, -2 -2)), ((10 -5, 15 5, 5 5, 10 -5))).

```
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)))')
```

ST_MultiPolygon(LONG BINARY[, INT]) constructor

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

Syntax

NEW ST_MultiPolygon(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	A string containing the binary representation of an multi polygon. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
srid	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a multi polygon from a binary string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.6.2

Example

The following returns MultiPolygon (((10 -5, 15 5, 5 5, 10 -5))).

```
SELECT NEW
ST_MultiPolygon(0x0106000000010000000103000000010000000400000000000000000244
000000000000014c000000000000002e4000000000000001440000000000000144000000000000
1440000000000000244000000000000014c0)
```

ST_MultiPolygon(ST_Polygon,...) constructor

Constructs a multi-polygon from a list of polygon values.

Syntax

NEW ST_MultiPolygon(*polygon1*[,*polygon2*,...,*polygonN*])

Parameters

Name	Type	Description
polygon1	ST_Polygon	The first polygon value of the multi-polygon.
polygon2,...,polygonN	ST_Polygon	Additional polygon values of the multi-polygon.

Remarks

Constructs a multi-polygon from a list of polygon values. All of the supplied polygon values must have the same SRID, and the multi-polygon is constructed with this common SRID.

All of the supplied polygon values must have the same answer for Is3D and IsMeasured. The multi-polygon is 3D if all of the polygon values are 3D, and the multi-polygon is measured if all of the polygon values are measured.

Note
By default, ST_MultiPolygon uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

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))' )  )
```

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(*multi-linestring*[, *polygon-format*])

Parameters

Name	Type	Description
multi-line-string	ST_MultiLine-String	A multilinestring value containing exterior rings and (optionally) a set of interior rings.
polygon-for-mat	VARCHAR(128)	A string with the polygon format to use when interpreting the provided linestrings. Valid formats are 'CounterClockwise', 'Clockwise', and 'EvenOdd'

Remarks

Creates a multi-polygon from a multilinestring containing exterior rings and an optional list of interior rings. The multilinestring must contain only linear rings.

If specified, the *polygon-format* parameter selects the algorithm the server uses to determine whether a ring is an exterior or interior ring. If not specified, the polygon format of the spatial reference system is used.

For additional information on *polygon-format*, see "POLYGON FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Note
By default, ST_MultiPolygon uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

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))'))
```

ST_MultiPolygonAggr method

Returns a multipolygon containing all of the polygons in a group.

Syntax

ST_MultiPolygon::**ST_MultiPolygonAggr**(*geometry-column*[**ORDER BY** *order-by-expression* [**ASC** | **DESC**], ...])

Parameters

Name	Type	Description
geometry-column	ST_Polygon	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.

  The spatial reference system identifier of the result is the same as that for the first parameter.

Remarks

The ST_MultiPolygonAggr aggregate function can be used to combine a group of polygons into a single collection. All of the geometries to be combined must have both the same SRID and the same coordinate dimension.

Rows where the argument is NULL are not included.

Returns NULL for an empty group or a group containing no non-NULL values.

The resulting ST_MultiPolygon has the same coordinate dimension as each polygons.

The optional ORDER BY clause can be used to arrange the elements in a particular order so that ST_GeometryN returns them in the desired order. If this ordering is not relevant, it is more efficient to not specify an ordering. In that case, the ordering of elements depends on the access plan selected by the query optimizer.

ST_MultiPolygonAggr is more efficient than ST_UnionAggr, but ST_MultiPolygonAggr can return a collection with duplicate or overlapping polygons if they exist in the group of polygons. In particular, returned collections containing overlapping surfaces may case unexpected results if they are used as input to other spatial methods. ST_UnionAggr handles duplicate and overlapping geometries.

Note
By default, ST_MultiPolygonAggr uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_UnionAggr method for type ST_Geometry"

Standards and compatibility

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

Example

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 )
```

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.

Direct supertype

- "ST_GeomCollection type"

Direct subtypes

- "ST_MultiPolygon type"

Constructor

- "ST_MultiSurface constructor"

Methods

- Methods of ST_MultiSurface:

ST_Area	ST_Centroid	ST_MultiSurfaceAggr	ST_Perimeter
ST_PointOnSurface			

- All methods of "ST_GeomCollection type" can also be called on a ST_MultiSurface type:

ST_GeomCollectionAggr	ST_GeometryN	ST_NumGeometries

- All methods of "ST_Geometry type" can also be called on a ST_MultiSurface type:

ST_Affine	ST_AsBinary	ST_AsGML	ST_AsGeoJSON
ST_AsKML	ST_AsSVG	ST_AsSVGAggr	ST_AsText
ST_AsWKB	ST_AsWKT	ST_AsXML	ST_Boundary
ST_Contains	ST_ContainsFilter	ST_ConvexHull	ST_ConvexHullAggr
ST_CoordDim	ST_CoveredBy	ST_CoveredByFilter	ST_Covers
ST_CoversFilter	ST_Crosses	ST_Difference	ST_Dimension
ST_Disjoint	ST_Distance	ST_Envelope	ST_EnvelopeAggr
ST_Equals	ST_EqualsFilter	ST_GeomFromBinary	ST_GeomFromShape
ST_GeomFromText	ST_GeomFromWKB	ST_GeomFromWKT	ST_GeometryType
ST_GeometryType-FromBaseType	ST_Intersection	ST_IntersectionAggr	ST_Intersects
ST_IntersectsFilter	ST_IntersectsRect	ST_Is3D	ST_IsEmpty
ST_IsMeasured	ST_IsSimple	ST_IsValid	ST_LatNorth
ST_LatSouth	ST_LinearHash	ST_LinearUnHash	ST_LoadConfigura-tionData
ST_LongEast	ST_LongWest	ST_MMax	ST_MMin
ST_OrderingEquals	ST_Overlaps	ST_Relate	ST_Reverse
ST_SRID	ST_SRIDFromBase-Type	ST_SnapToGrid	ST_SymDifference
ST_ToCircular	ST_ToCompound	ST_ToCurve	ST_ToCurvePoly
ST_ToGeomColl	ST_ToLineString	ST_ToMultiCurve	ST_ToMultiLine
ST_ToMultiPoint	ST_ToMultiPolygon	ST_ToMultiSurface	ST_ToPoint
ST_ToPolygon	ST_ToSurface	ST_Touches	ST_Transform
ST_Union	ST_UnionAggr	ST_Within	ST_WithinDistance

ST_WithinDistanceFilter	ST_WithinFilter	ST_XMax	ST_XMin
ST_YMax	ST_YMin	ST_ZMax	ST_ZMin

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.5

ST_MultiSurface constructor

Constructs a multi surface.

Overload list

Name	**Description**
"ST_MultiSurface() constructor"	Constructs a multi surface representing the empty set.
"ST_MultiSurface(LONG VARCHAR[, INT]) constructor"	Constructs a multi surface from a text representation.
"ST_MultiSurface(LONG BINARY[, INT]) constructor"	Constructs a multi surface from Well Known Binary (WKB).
"ST_MultiSurface(ST_Surface,...) constructor"	Constructs a multi-surface from a list of surface values.
"ST_MultiSurface(ST_MultiCurve[, VARCHAR(128)]) constructor"	Creates a multi-surface from a multicurve containing exterior rings and an optional list of interior rings.

ST_MultiSurface() constructor

Constructs a multi surface representing the empty set.

Syntax

NEW ST_MultiSurface()

Standards and compatibility

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

Example

The following returns 1, indicating the value is empty.

```
SELECT NEW ST_MultiSurface().ST_IsEmpty()
```

ST_MultiSurface(LONG VARCHAR[, INT]) constructor

Constructs a multi surface from a text representation.

Syntax

NEW ST_MultiSurface(*text-representation*[, *srid*])

Parameters

Name	Type	Description
text-representation	LONG VARCHAR	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	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a multi surface from a character string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.5.2

Example

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)))')
```

ST_MultiSurface(LONG BINARY[, INT]) constructor

Constructs a multi surface from Well Known Binary (WKB).

Syntax

NEW ST_MultiSurface(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	A string containing the binary representation of an multi surface. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
srid	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a multi surface from a binary string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.5.2

Example

The following returns MultiSurface (CurvePolygon (CircularString (0 0, 10 0, 10 10, 0 10, 0 0))).

```
SELECT NEW
ST_MultiSurface(0x010c00000001000000010a0000000100000001080000000500000000000
000000000000000000000000000000000000000244000000000000000000000000000000244000
00000000002440000000000000000000000000000002440000000000000000000000000000000
0)
```

ST_MultiSurface(ST_Surface,...) constructor

Constructs a multi-surface from a list of surface values.

Syntax

NEW ST_MultiSurface(*surface1*[,*surface2*,...,*surfaceN*])

Parameters

Name	Type	Description
surface1	ST_Surface	The first surface value of the multi-surface.
surface2,...,surfaceN	ST_Surface	Additional surface values of the multi-surface.

Remarks

Constructs a multi-surface from a list of surface values. All of the supplied surface values must have the same SRID, and the multi-surface is constructed with this common SRID.

All of the supplied surface values must have the same answer for Is3D and IsMeasured. The multi-surface is 3D if all of the surface values are 3D, and the multi-surface is measured if all of the surface values are measured.

Note
By default, ST_MultiSurface uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

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))' )  )
```

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(*multi-curve*[, *polygon-format*])

Parameters

Name	Type	Description
multi-curve	ST_MultiCurve	A multicurve value containing exterior rings and (optionally) a set of interior rings.
polygon-format	VARCHAR(128)	A string with the polygon format to use when interpreting the provided curves. Valid formats are 'CounterClockwise', 'Clockwise', and 'EvenOdd'

Remarks

Creates a multi-surface from a multicurve containing exterior rings and an optional list of interior rings. The multicurve may contain any curve type.

If specified, the *polygon-format* parameter selects the algorithm the server uses to determine whether a ring is an exterior or interior ring. If not specified, the polygon format of the spatial reference system is used.

For additional information on *polygon-format*, see "POLYGON FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Note
By default, ST_MultiSurface uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

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))'))
```

ST_Area method

Computes the area of the multi-surface in the specified units.

Syntax

multisurface-expression.**ST_Area**([*unit-name*])

Parameters

Name	Type	Description
unit-name	VARCHAR(128)	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.

Remarks

Computes the area of the multi-surface in the specified units. The area of the multi-surface is the sum of the areas of the contained surfaces.

Note
This method can not be used with geometries in round-Earth spatial reference system.

See also

- "ST_Perimeter method for type ST_MultiSurface"
- "ST_Area method for type ST_Surface"
- "ST_Length method for type ST_MultiCurve"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.5.3

Example

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
```

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.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *multisurface-expression*.

Remarks

Computes the ST_Point that is the mathematical centroid of the multi-surface. Note that this point will not necessarily be a point on the surface.

Note
This method can not be used with geometries in round-Earth spatial reference system.

See also

- "ST_Centroid method for type ST_Surface"
- "ST_PointOnSurface method for type ST_MultiSurface"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.5.5

Example

The following example returns the result `Point (1.865682 .664892)`.

```
SELECT TREAT( Shape AS ST_MultiSurface ).ST_Centroid()
FROM SpatialShapes WHERE ShapeID = 28
```

ST_MultiSurfaceAggr method

Returns a multisurface containing all of the surfaces in a group.

Syntax

ST_MultiSurface::**ST_MultiSurfaceAggr**(*geometry-column*[**ORDER BY** *order-by-expression* [**ASC** | **DESC**], ...])

Parameters

Name	Type	Description
geometry-column	ST_Surface	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.

  The spatial reference system identifier of the result is the same as that for the first parameter.

Remarks

The ST_MultiSurfaceAggr aggregate function can be used to combine a group of surfaces into a single collection. All of the geometries to be combined must have both the same SRID and the same coordinate dimension.

Rows where the argument is NULL are not included.

Returns NULL for an empty group or a group containing no non-NULL values.

The resulting ST_MultiSurface has the same coordinate dimension as each surfaces.

The optional ORDER BY clause can be used to arrange the elements in a particular order so that ST_GeometryN returns them in the desired order. If this ordering is not relevant, it is more efficient to not specify an ordering. In that case, the ordering of elements depends on the access plan selected by the query optimizer.

ST_MultiSurfaceAggr is more efficient than ST_UnionAggr, but ST_MultiSurfaceAggr can return a collection with duplicate or overlapping surfaces if they exist in the group of surfaces. In particular, returned collections containing overlapping surfaces may case unexpected results if they are used as input to other spatial methods. ST_UnionAggr handles duplicate and overlapping geometries.

Note
By default, ST_MultiSurfaceAggr uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_UnionAggr method for type ST_Geometry"

Standards and compatibility

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

Example

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 )
```

ST_Perimeter method

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

Syntax

multisurface-expression.**ST_Perimeter**([*unit-name*])

Parameters

Name	Type	Description
unit-name	VARCHAR(128)	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.

Remarks

The ST_Perimeter method returns the length of the perimeter of a multi-surface in the units identified by the *unit-name* parameter. If the multi-surface is empty, then NULL is returned.

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

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

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

Note
By default, ST_Perimeter uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_Perimeter method for type ST_Surface"
- "ST_Boundary method for type ST_Geometry"
- "ST_Length method for type ST_MultiCurve"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.5.4

Example

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

```
SELECT NEW ST_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');
```

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.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *multisurface-expression*.

Remarks

Returns a point that is in the interior of one of the surfaces of a multi-surface.

> **Note**
> If the *multisurface-expression* contains circularstrings, then these are interpolated to line strings.

See also

- "ST_PointOnSurface method for type ST_Surface"
- "ST_Centroid method for type ST_MultiSurface"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.5.6

Example

The following returns a point that intersects the multi surface.

```
SELECT TREAT( Shape AS ST_MultiSurface ).ST_PointOnSurface()
FROM SpatialShapes WHERE ShapeID = 27
```

ST_Point type

The ST_Point type is a 0-dimensional geometry and represents a single location.

Direct supertype

- "ST_Geometry type"

Constructor

- "ST_Point constructor"

Methods

- Methods of ST_Point:

ST_Lat	ST_Long	ST_M	ST_X
ST_Y	ST_Z		

- All methods of "ST_Geometry type" can also be called on a ST_Point type:

ST_Affine	ST_AsBinary	ST_AsGML	ST_AsGeoJSON
ST_AsKML	ST_AsSVG	ST_AsSVGAggr	ST_AsText
ST_AsWKB	ST_AsWKT	ST_AsXML	ST_Boundary

ST_Contains	ST_ContainsFilter	ST_ConvexHull	ST_ConvexHullAggr
ST_CoordDim	ST_CoveredBy	ST_CoveredByFilter	ST_Covers
ST_CoversFilter	ST_Crosses	ST_Difference	ST_Dimension
ST_Disjoint	ST_Distance	ST_Envelope	ST_EnvelopeAggr
ST_Equals	ST_EqualsFilter	ST_GeomFromBinary	ST_GeomFromShape
ST_GeomFromText	ST_GeomFromWKB	ST_GeomFromWKT	ST_GeometryType
ST_GeometryType-FromBaseType	ST_Intersection	ST_IntersectionAggr	ST_Intersects
ST_IntersectsFilter	ST_IntersectsRect	ST_Is3D	ST_IsEmpty
ST_IsMeasured	ST_IsSimple	ST_IsValid	ST_LatNorth
ST_LatSouth	ST_LinearHash	ST_LinearUnHash	ST_LoadConfigura-tionData
ST_LongEast	ST_LongWest	ST_MMax	ST_MMin
ST_OrderingEquals	ST_Overlaps	ST_Relate	ST_Reverse
ST_SRID	ST_SRIDFromBase-Type	ST_SnapToGrid	ST_SymDifference
ST_ToCircular	ST_ToCompound	ST_ToCurve	ST_ToCurvePoly
ST_ToGeomColl	ST_ToLineString	ST_ToMultiCurve	ST_ToMultiLine
ST_ToMultiPoint	ST_ToMultiPolygon	ST_ToMultiSurface	ST_ToPoint
ST_ToPolygon	ST_ToSurface	ST_Touches	ST_Transform
ST_Union	ST_UnionAggr	ST_Within	ST_WithinDistance
ST_WithinDistanceFil-ter	ST_WithinFilter	ST_XMax	ST_XMin
ST_YMax	ST_YMin	ST_ZMax	ST_ZMin

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1

ST_Point constructor

Constructs a point.

Note
When creating an ST_Point value from coordinates, the overload that is picked is not always predictable. For example, the expression "NEW ST_Point(1,2,3)" creates a 2D point with X=1, Y=2 and SRID=3. The expression "NEW ST_Point(1,2,3.0)" creates a 3D point with Z=3.0.

Overload list

Name	Description
"ST_Point() constructor"	Constructs a point representing the empty set.
"ST_Point(LONG VARCHAR[, INT]) constructor"	Constructs a point from a text representation.
"ST_Point(LONG BINARY[, INT]) constructor"	Constructs a point from Well Known Binary (WKB).
"ST_Point(DOUBLE,DOUBLE[, INT]) constructor"	Constructs a 2D point from X,Y coordinates.
"ST_Point(DOUBLE,DOUBLE,DOUBLE[, INT]) constructor"	Constructs a 3D point from X,Y,Z coordinates.
"ST_Point(DOUBLE,DOUBLE,DOUBLE,DOUBLE[, INT]) constructor"	Constructs a 3D, measured point from X,Y,Z coordinates and a measure value

ST_Point() constructor

Constructs a point representing the empty set.

Syntax

NEW ST_Point()

Standards and compatibility

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

Example

The following returns 1, indicating the value is empty.

```
SELECT NEW ST_Point().ST_IsEmpty()
```

ST_Point(LONG VARCHAR[, INT]) constructor

Constructs a point from a text representation.

Syntax

NEW ST_Point(*text-representation*[, *srid*])

Parameters

Name	Type	Description
text-repre-sentation	LONG VAR-CHAR	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	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a point from a character string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.2

Example

The following returns Point (10 20).

```
SELECT NEW ST_Point('Point (10 20)')
```

ST_Point(LONG BINARY[, INT]) constructor

Constructs a point from Well Known Binary (WKB).

Syntax

NEW ST_Point(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BI-NARY	A string containing the binary representation of an point. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
srid	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a point from a binary string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.2

Example

The following returns Point (10 20).

```
SELECT NEW ST_Point(0x010100000000000000000024400000000000003440)
```

ST_Point(DOUBLE,DOUBLE[, INT]) constructor

Constructs a 2D point from X,Y coordinates.

Syntax

NEW ST_Point(*x*,*y*[, *srid*])

Parameters

Name	Type	Description
x	DOUBLE	The X coordinate value.
y	DOUBLE	The Y coordinate value.
srid	INT	The SRID of the result. If not specified, the default is 0.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.2

Example

The following returns Point (10 20).

```
SELECT NEW ST_Point(10.0,20.0,0)
```

ST_Point(DOUBLE,DOUBLE,DOUBLE[, INT]) constructor

Constructs a 3D point from X,Y,Z coordinates.

Syntax

NEW ST_Point(*x*,*y*,*z*[, *srid*])

Parameters

Name	Type	Description
x	DOUBLE	The X coordinate value.
y	DOUBLE	The Y coordinate value.
z	DOUBLE	The Z coordinate value.
srid	INT	The SRID of the result. If not specified, the default is 0.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.2

Example

The following returns Point Z (10 20 100).

```
SELECT NEW ST_Point(10.0,20.0,100.0,0)
```

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(*x*,*y*,*z*,*m*[, *srid*])

Parameters

Name	Type	Description
x	DOUBLE	The X coordinate value.
y	DOUBLE	The Y coordinate value.
z	DOUBLE	The Z coordinate value.
m	DOUBLE	The measure value.
srid	INT	The SRID of the result. If not specified, the default is 0.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.2

Example

The following returns Point ZM (10 20 100 1224).

```
SELECT NEW ST_Point(10.0,20.0,100.0,1224.0,0)
```

ST_Lat method

Returns the latitude coordinate of the ST_Point value.

Overload list

Name	Description
"ST_Lat() method for type ST_Point"	Returns the latitude coordinate of the ST_Point value.
"ST_Lat(DOUBLE) method for type ST_Point"	Returns a copy of the point with the latitude coordinate set to the specified latitude value.

ST_Lat() method for type ST_Point

Returns the latitude coordinate of the ST_Point value.

Note
If the *point-expression* is an empty geometry (ST_IsEmpty()=1), then this method returns NULL.

Note
By default, ST_Lat uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

point-expression.**ST_Lat**()

Returns

- **DOUBLE** Returns the latitude coordinate of the ST_Point value.

See also

- "ST_Long method for type ST_Point"
- "ST_Y method for type ST_Point"
- "ST_LatNorth method for type ST_Geometry"
- "ST_LatSouth method for type ST_Geometry"

Standards and compatibility

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

Example

The following example gives an error because the spatial reference system identified by 0 is not a geographic spatial reference system.

```
SELECT NEW ST_Point( 10.0, 20.0, 0 ).ST_Lat()
```

The following example returns the result `20.0`.

```
SELECT NEW ST_Point( 10.0, 20.0, 4326 ).ST_Lat()
```

ST_Lat(DOUBLE) method for type ST_Point

Returns a copy of the point with the latitude coordinate set to the specified latitude value.

Note
If the *point-expression* is an empty geometry (ST_IsEmpty()=1), then this method returns NULL.

Note
By default, ST_Lat uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

point-expression.**ST_Lat**(*latitude-val*)

Parameters

Name	Type	Description
latitude-val	DOUBLE	The new latitude value.

Returns

- **ST_Point** Returns a copy of the point with the latitude set to the specified value.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *point-expression*.

Standards and compatibility

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

ST_Long method

Returns the longitude coordinate of the ST_Point value.

Overload list

Name	Description
"ST_Long() method for type ST_Point"	Returns the longitude coordinate of the ST_Point value.
"ST_Long(DOUBLE) method for type ST_Point"	Returns a copy of the point with the longitude coordinate set to the specified longitude value.

ST_Long() method for type ST_Point

Returns the longitude coordinate of the ST_Point value.

Note
If the *point-expression* is an empty geometry (ST_IsEmpty()=1), then this method returns NULL.

Note
By default, ST_Long uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

point-expression.**ST_Long**()

Returns

- **DOUBLE** Returns the longitude coordinate of the ST_Point value.

See also

- "ST_Lat method for type ST_Point"
- "ST_X method for type ST_Point"
- "ST_LongEast method for type ST_Geometry"
- "ST_LongWest method for type ST_Geometry"

Standards and compatibility

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

Example

The following example gives an error because the spatial reference system identified by 0 is not a geographic spatial reference system.

```
SELECT NEW ST_Point( 10.0, 20.0, 0 ).ST_Long()
```

The following example returns the result 10.0.

```
SELECT NEW ST_Point( 10.0, 20.0, 4326 ).ST_Long()
```

ST_Long(DOUBLE) method for type ST_Point

Returns a copy of the point with the longitude coordinate set to the specified longitude value.

Note
If the *point-expression* is an empty geometry (ST_IsEmpty()=1), then this method returns NULL.

Note
By default, ST_Long uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

point-expression.**ST_Long**(*longitude-val*)

Parameters

Name	Type	Description
longitude-val	DOUBLE	The new longitude value.

Returns

- **ST_Point** Returns a copy of the point with the longitude set to the specified value.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *point-expression*.

Standards and compatibility

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

ST_M method

Retrieves or modifies the measure value of a point.

Overload list

Name	Description
"ST_M() method for type ST_Point"	Returns the measure value of the ST_Point value.
"ST_M(DOUBLE) method for type ST_Point"	Returns a copy of the point with the measure value set to the specified mcoord value.

ST_M() method for type ST_Point

Returns the measure value of the ST_Point value.

Note
If the *point-expression* is an empty geometry (ST_IsEmpty()=1), then this method returns NULL.

Note
By default, ST_M uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

point-expression.**ST_M**()

Returns

- **DOUBLE** Returns the measure value of the ST_Point value.

See also

- "ST_X method for type ST_Point"
- "ST_Y method for type ST_Point"
- "ST_MMin method for type ST_Geometry"
- "ST_MMax method for type ST_Geometry"
- "ST_IsMeasured method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.6

Example

The following example returns the result `40.0`.

```
SELECT NEW ST_Point( 10.0, 20.0, 30.0, 40.0, 0 ).ST_M()
```

ST_M(DOUBLE) method for type ST_Point

Returns a copy of the point with the measure value set to the specified mcoord value.

Note
If the *point-expression* is an empty geometry (ST_IsEmpty()=1), then this method returns NULL.

Note
By default, ST_M uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

point-expression.**ST_M**(*mcoord*)

Parameters

Name	Type	Description
mcoord	DOUBLE	The new measure value.

Returns

- **ST_Point** Returns a copy of the point with the measure value set to the specified mcoord value.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *point-expression*.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.6

Example

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 )
```

ST_X method

Retrieves or modifies the X coordinate value of a point.

Overload list

Name	Description
"ST_X() method for type ST_Point"	Returns the X coordinate of the ST_Point value.
"ST_X(DOUBLE) method for type ST_Point"	Returns a copy of the point with the X coordinate set to the specified xcoord value.

ST_X() method for type ST_Point

Returns the X coordinate of the ST_Point value.

Note
If the *point-expression* is an empty geometry (ST_IsEmpty()=1), then this method returns NULL.

> **Note**
> By default, ST_X uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

point-expression.**ST_X**()

Returns

- **DOUBLE** Returns the X coordinate of the ST_Point value.

See also

- "ST_Y method for type ST_Point"
- "ST_Lat method for type ST_Point"
- "ST_XMin method for type ST_Geometry"
- "ST_XMax method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.3

Example

The following example returns the result `10.0`.

```
SELECT NEW ST_Point( 10.0, 20.0, 30.0, 40.0, 0 ).ST_X()
```

ST_X(DOUBLE) method for type ST_Point

Returns a copy of the point with the X coordinate set to the specified xcoord value.

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

> **Note**
> By default, ST_X uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

point-expression.**ST_X**(*xcoord*)

Parameters

Name	Type	Description
xcoord	DOUBLE	The new X coordinate value.

Returns

- **ST_Point** Returns a copy of the point with the X coordinate set to the specified xcoord value.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *point-expression*.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.3

ST_Y method

Retrieves or modifies the Y coordinate value of a point.

Overload list

Name	Description
"ST_Y() method for type ST_Point"	Returns the Y coordinate of the ST_Point value.
"ST_Y(DOUBLE) method for type ST_Point"	Returns a copy of the point with the Y coordinate set to the specified ycoord value.

ST_Y() method for type ST_Point

Returns the Y coordinate of the ST_Point value.

Note
If the *point-expression* is an empty geometry (ST_IsEmpty()=1), then this method returns NULL.

Note
By default, ST_Y uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

point-expression.**ST_Y**()

Returns

- **DOUBLE** Returns the Y coordinate of the ST_Point value.

See also

- "ST_X method for type ST_Point"
- "ST_Long method for type ST_Point"
- "ST_YMin method for type ST_Geometry"
- "ST_YMax method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.4

Example

The following example returns the result `20.0`.

```
SELECT NEW ST_Point( 10.0, 20.0, 30.0, 40.0, 0 ).ST_Y()
```

ST_Y(DOUBLE) method for type ST_Point

Returns a copy of the point with the Y coordinate set to the specified ycoord value.

Note
If the *point-expression* is an empty geometry (ST_IsEmpty()=1), then this method returns NULL.

Note
By default, ST_Y uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

point-expression.**ST_Y**(*ycoord*)

Parameters

Name	Type	Description
ycoord	DOUBLE	The new Y coordinate value.

Returns

- **ST_Point** Returns a copy of the point with the Y coordinate set to the specified ycoord value.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *point-expression*.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.4

Example

The following example returns the result `Point (1 3)`.

```
SELECT NEW ST_Point( 1, 2 ).ST_Y( 3 )
```

ST_Z method

Retrieves or modifies the Z coordinate value of a point.

Overload list

Name	Description
"ST_Z() method for type ST_Point"	Returns the Z coordinate of the ST_Point value.
"ST_Z(DOUBLE) method for type ST_Point"	Returns a copy of the point with the Z coordinate set to the specified zcoord value.

ST_Z() method for type ST_Point

Returns the Z coordinate of the ST_Point value.

Note
If the *point-expression* is an empty geometry (ST_IsEmpty()=1), then this method returns NULL.

Note
By default, ST_Z uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

point-expression.**ST_Z**()

Returns

- **DOUBLE** Returns the Z coordinate of the ST_Point value.

See also

- "ST_X method for type ST_Point"
- "ST_Y method for type ST_Point"
- "ST_ZMin method for type ST_Geometry"
- "ST_ZMax method for type ST_Geometry"
- "ST_Is3D method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.4

Example

The following example returns the result `30.0`.

```
SELECT NEW ST_Point( 10.0, 20.0, 30.0, 40.0, 0 ).ST_Z()
```

ST_Z(DOUBLE) method for type ST_Point

Returns a copy of the point with the Z coordinate set to the specified zcoord value.

Note
If the *point-expression* is an empty geometry (ST_IsEmpty()=1), then this method returns NULL.

Note
By default, ST_Z uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

point-expression.**ST_Z**(*zcoord*)

Parameters

Name	Type	Description
zcoord	DOUBLE	The new Z coordinate value.

Returns

- **ST_Point** Returns a copy of the point with the Z coordinate set to the specified zcoord value.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *point-expression*.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.5

Example

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 )
```

ST_Polygon type

An ST_Polygon is an ST_CurvePolygon that is formed with interior and exterior rings that are linear rings.

Direct supertype

- "ST_CurvePolygon type"

Constructor

- "ST_Polygon constructor"

Methods

- Methods of ST_Polygon:

ST_ExteriorRing	ST_InteriorRingN

- All methods of "ST_CurvePolygon type" can also be called on a ST_Polygon type:

ST_CurvePolyToPoly	ST_ExteriorRing	ST_InteriorRingN	ST_NumInteriorRing

- All methods of "ST_Surface type" can also be called on a ST_Polygon type:

ST_Area	ST_Centroid	ST_IsWorld	ST_Perimeter
ST_PointOnSurface			

- All methods of "ST_Geometry type" can also be called on a ST_Polygon type:

ST_Affine	ST_AsBinary	ST_AsGML	ST_AsGeoJSON
ST_AsKML	ST_AsSVG	ST_AsSVGAggr	ST_AsText
ST_AsWKB	ST_AsWKT	ST_AsXML	ST_Boundary
ST_Contains	ST_ContainsFilter	ST_ConvexHull	ST_ConvexHullAggr
ST_CoordDim	ST_CoveredBy	ST_CoveredByFilter	ST_Covers
ST_CoversFilter	ST_Crosses	ST_Difference	ST_Dimension

ST_Disjoint	ST_Distance	ST_Envelope	ST_EnvelopeAggr
ST_Equals	ST_EqualsFilter	ST_GeomFromBinary	ST_GeomFromShape
ST_GeomFromText	ST_GeomFromWKB	ST_GeomFromWKT	ST_GeometryType
ST_GeometryType-FromBaseType	ST_Intersection	ST_IntersectionAggr	ST_Intersects
ST_IntersectsFilter	ST_IntersectsRect	ST_Is3D	ST_IsEmpty
ST_IsMeasured	ST_IsSimple	ST_IsValid	ST_LatNorth
ST_LatSouth	ST_LinearHash	ST_LinearUnHash	ST_LoadConfigura-tionData
ST_LongEast	ST_LongWest	ST_MMax	ST_MMin
ST_OrderingEquals	ST_Overlaps	ST_Relate	ST_Reverse
ST_SRID	ST_SRIDFromBase-Type	ST_SnapToGrid	ST_SymDifference
ST_ToCircular	ST_ToCompound	ST_ToCurve	ST_ToCurvePoly
ST_ToGeomColl	ST_ToLineString	ST_ToMultiCurve	ST_ToMultiLine
ST_ToMultiPoint	ST_ToMultiPolygon	ST_ToMultiSurface	ST_ToPoint
ST_ToPolygon	ST_ToSurface	ST_Touches	ST_Transform
ST_Union	ST_UnionAggr	ST_Within	ST_WithinDistance
ST_WithinDistanceFil-ter	ST_WithinFilter	ST_XMax	ST_XMin
ST_YMax	ST_YMin	ST_ZMax	ST_ZMin

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 8.3

ST_Polygon constructor

Constructs a polygon.

Overload list

Name	Description
"ST_Polygon() constructor"	Constructs a polygon representing the empty set.
"ST_Polygon(LONG VARCHAR[, INT]) constructor"	Constructs a polygon from a text representation.
"ST_Polygon(LONG BINARY[, INT]) constructor"	Constructs a polygon from Well Known Binary (WKB).
"ST_Polygon(ST_Point,ST_Point) constructor"	Creates an axis-aligned rectangle from two points representing the lower-left and upper-right corners.
"ST_Polygon(ST_MultiLineString[, VARCHAR(128)]) constructor"	Creates a polygon from a multilinestring containing an exterior ring and an optional list of interior rings.
"ST_Polygon(ST_LineString,...) constructor"	Creates a polygon from a linestring representing the exterior ring and an optional list of linestrings representing interior rings.

ST_Polygon() constructor

Constructs a polygon representing the empty set.

Syntax

NEW ST_Polygon()

Standards and compatibility

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

Example

The following returns 1, indicating the value is empty.

```
SELECT NEW ST_Polygon().ST_IsEmpty()
```

ST_Polygon(LONG VARCHAR[, INT]) constructor

Constructs a polygon from a text representation.

Syntax

NEW ST_Polygon(*text-representation*[, *srid*])

Parameters

Name	Type	Description
text-representation	LONG VARCHAR	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	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a polygon from a character string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 8.3.2

Example

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))')
```

ST_Polygon(LONG BINARY[, INT]) constructor

Constructs a polygon from Well Known Binary (WKB).

Syntax

NEW ST_Polygon(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	A string containing the binary representation of an polygon. The input can be in any supported binary input format, including Well Known Binary (WKB) or Extended Well Known Binary (EWKB).
srid	INT	The SRID of the result. If not specified, the default is 0.

Remarks

Constructs a polygon from a binary string representation. The database server determines the input format by inspecting the provided string.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 8.3.2

Example

The following returns Polygon ((10 -5, 15 5, 5 5, 10 -5)).

```
SELECT NEW
ST_Polygon(0x0103000000010000000400000000000000000024400000000000000014c0000000
0000002e400000000000001440000000000000144000000000000014400000000000002440000
00000000014c0)
```

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(*pmin*,*pmax*)

Parameters

Name	Type	Description
pmin	ST_Point	A point that is the lower-left corner of the rectangle.
pmax	ST_Point	A point that is the upper-right corner of the rectangle.

Remarks

Returns a rectangle defined as the envelope of two points.

The constructor is equivalent to the following: `NEW ST_MultiPoint( pmin, pmax ).ST_Envelope()`

If the input points do not specify the corresponding corners of a rectangle, the server will determine the correct containing rectangle for the specified points.

Note
If input points are in a round-Earth spatial reference system, the produced polygon will not be an axis-aligned rectangle. The eastern and western boundaries will coincide with the meridians between the eastern and western point pairs, respectively, while the northern and southern edges will not coincide with the parallels and will rather follow great elliptical arches containing the edge end points. Additionally, the server will always generate a valid polygon, if possible. If such a valid polygon crosses the date line, longitude of the western points will be higher than that of the eastern points, counter to the relationship for polygons not crossing the date line.

Note
By default, ST_Polygon uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

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

Example

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))
```

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(*multi-linestring*[, *polygon-format*])

Parameters

Name	Type	Description
multi-line-string	ST_MultiLine-String	A multilinestring value containing an exterior ring and (optional-ly) a set of interior rings.
polygon-for-mat	VARCHAR(128)	A string with the polygon format to use when interpreting the pro-vided linestrings. Valid formats are 'CounterClockwise', 'Clock-wise', and 'EvenOdd'

Remarks

Creates a polygon from a multilinestring containing an exterior ring and an optional list of interior rings. The multilinestring must contain only linear rings.

If specified, the *polygon-format* parameter selects the algorithm the server uses to determine whether a ring is an exterior or interior ring. If not specified, the polygon format of the spatial reference system is used.

For additional information on *polygon-format*, see "POLYGON FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Note
By default, ST_Polygon uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 8.3.2

Example

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))'))
```

ST_Polygon(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(*exterior-ring*[,*interior-ring1*,...,*interior-ringN*])

Parameters

Name	Type	Description
exterior-ring	ST_LineString	The exterior ring of the polygon
interior-ring1,...,interior-ringN	ST_LineString	Interior rings of the polygon

Remarks

Creates a polygon from a linestring representing the exterior ring and a list (possibly empty) of linestrings representing interior rings. All of the specified linestring values must have the same SRID. The resulting polygon is constructed with this common SRID.

All of the supplied linestrings must be non-empty and have the same answer for Is3D and IsMeasured. The polygon is 3D if all of the linestrings are 3D, and the polygon is measured if all of the linestrings are measured.

Note
By default, ST_Polygon uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Standards and compatibility

The ability to specify a varying length list of interior rings is a vendor extension.

- **SQL/MM (ISO/IEC 13249-3: 2006)** 8.3.2

Example

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)'))
```

ST_ExteriorRing method

Retrieve or modify the exterior ring.

Overload list

Name	Description
"ST_ExteriorRing() method for type ST_Polygon"	Returns the exterior ring of the polygon.
"ST_ExteriorRing(ST_Curve) method for type ST_Polygon"	Changes the exterior ring of the polygon.

ST_ExteriorRing() method for type ST_Polygon

Returns the exterior ring of the polygon.

Note
By default, ST_ExteriorRing uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

polygon-expression.**ST_ExteriorRing**()

Returns

- **ST_LineString** Returns the exterior ring of the polygon.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *polygon-expression*.

See also

- "ST_InteriorRingN method for type ST_Polygon"
- "ST_ExteriorRing method for type ST_CurvePolygon"
- "ST_Boundary method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 8.3.3

Example

The following example returns the result `LineString (0 0, 10 0, 5 10, 0 0)`.

```
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()
```

ST_ExteriorRing(ST_Curve) method for type ST_Polygon

Changes the exterior ring of the polygon.

> **Note**
> By default, ST_ExteriorRing uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

polygon-expression.**ST_ExteriorRing**(*curve*)

Parameters

Name	Type	Description
curve	ST_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.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *polygon-expression*.

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 8.3.3

Example

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)' ) )
```

ST_InteriorRingN method

Returns the *n*th interior ring in the polygon.

Note
By default, ST_InteriorRingN uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

Syntax

polygon-expression.**ST_InteriorRingN**(*n*)

Parameters

Name	Type	Description
n	INT	The position of the element to return, from 1 to *polygon-expression*.ST_NumInteriorRing().

Returns

- **ST_LineString** Returns the *n*th interior ring in the polygon.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *polygon-expression*.

See also

- "ST_NumInteriorRing method for type ST_CurvePolygon"
- "ST_ExteriorRing method for type ST_Polygon"
- "ST_InteriorRingN method for type ST_CurvePolygon"
- "ST_Boundary method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 8.3.5

Example

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 )
```

ST_SpatialRefSys type

The ST_SpatialRefSys type defines routines for working with spatial reference systems.

Methods

- Methods of ST_SpatialRefSys:

ST_CompareWKT	ST_FormatTransformDefinition	ST_FormatWKT	ST_GetUnProjectedTransformDefinition
ST_ParseWKT	ST_TransformGeom	ST_World	

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 13.1

ST_CompareWKT method

Compares two spatial reference system definitions.

Syntax

ST_SpatialRefSys::**ST_CompareWKT**(*transform-definition-1*,*transform-definition-2*)

Parameters

Name	**Type**	**Description**
transform-definition-1	LONG VARCHAR	The first spatial reference system definition text
transform-definition-2	LONG VARCHAR	The second spatial reference system definition text

Returns

- **BIT** Returns 1 if the two spatial reference systems are logically equivalent, otherwise 0.

Remarks

Determines if two spatial reference systems (defined by WKT) are logically equivalent. The systems are considered logically equal if they are defined by the same authority with the same identifier or if the strings are exactly equal.

Standards and compatibility

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

Example

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"]],PRI
MEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",
0.01745329251994328,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4326"]]'
,    'GEOGCS["WGS 84 alternate name",DATUM["WGS_1984",SPHEROID["WGS 84",
6378137,298.257223563,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRI
MEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",
0.01745329251994328,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4326"]]'
) 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"]],PRI
MEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",
0.01745329251994328,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4326"]]'
,   'GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",
6378137,298.257223563,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRI
MEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",
0.01745329251994328,AUTHORITY["EPSG","9122"]],AUTHORITY["AnotherAuthority","4
326"]]'
) Considered_NotEqual
```

ST_FormatTransformDefinition method

Returns a formatted copy of the transform definition.

Syntax

ST_SpatialRefSys::**ST_FormatTransformDefinition**(*transform-definition*)

Parameters

Name	Type	Description
transform-definition	LONG VARCHAR	The spatial reference system transform definition text

Returns

- **LONG VARCHAR** Returns a text string defining the transform definition

Remarks

Returns a formatted copy of the transform definition.

Standards and compatibility

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

Example

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')
```

ST_FormatWKT method

Returns a formatted copy of the Well Known Text (WKT) definition.

Syntax

ST_SpatialRefSys::**ST_FormatWKT**(*definition*)

Parameters

Name	Type	Description
definition	LONG VARCHAR	The spatial reference system definition text

Returns

- **LONG VARCHAR** Returns a text string defining the spatial reference system in WKT.

Remarks

Returns a formatted copy of the WKT spatial reference system definition.

The Well Known Text which describes spatial reference systems is a different format from the Well Known Text which describes geometries.

Standards and compatibility

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

Example

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"]],PRI
MEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",
0.01745329251994328,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4326"]]')
```

ST_GetUnProjectedTransformDefinition method

Returns the transform definition of the spatial reference system that is the source of the projection.

Syntax

ST_SpatialRefSys::**ST_GetUnProjectedTransformDefinition**(*transform-definition*)

Parameters

Name	Type	Description
transform-definition	LONG VARCHAR	The spatial reference system transform definition text

Returns

- **LONG VARCHAR** Returns a text string defining the transform definition of the unprojected spatial reference system.

Remarks

If the *transform-definition* parameter defines a projected spatial reference system, returns the definition of the source spatial reference system.

Standards and compatibility

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

Example

The following example returns the result `+proj=latlong +a=6371000 +b=6371000 +no_defs`.

```
SELECT ST_SpatialRefSys::ST_GetUnProjectedTransformDefinition( '+proj=robin
+lon_0=0 +x_0=0 +y_0=0 +a=6371000 +b=6371000 +units=m  no_defs' )
```

ST_ParseWKT method

Retrieves a named element from the Well Known Text (WKT) definition of a spatial reference system.

Syntax

ST_SpatialRefSys::**ST_ParseWKT**(*element*,*srs-text*)

Parameters

Name	Type	Description
element	VAR-CHAR(128)	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	LONG VAR-CHAR	The spatial reference system definition text

Returns

- **LONG VARCHAR** Retrieves a named element from the WKT definition of a spatial reference system.

Remarks

Retrieves a named element from the WKT definition of a spatial reference system. If the WKT does not define the named element, NULL is returned.

The Well Known Text which describes spatial reference systems is a different format from the Well Known Text which describes geometries.

Standards and compatibility

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

Example

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"]],PRI
MEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",
0.01745329251994328,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4326"]]' as
srs_text
    FROM
sa_split_list('srs_name,srs_type,organization,organization_id,linear_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`	`.017453292519943282`

ST_TransformGeom method

Returns the geometry transformed using the given transform definition.

Syntax

ST_SpatialRefSys::**ST_TransformGeom**(*geom*,*target-transform-definition*[, *source-transform-definition*])

Parameters

Name	Type	Description
geom	ST_Geometry	The geometry to be transformed
target-transform-definition	LONG VARCHAR	The target spatial reference system transform definition text

Name	Type	Description
source-transform-definition	LONG VARCHAR	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.

  The spatial reference system identifier of the result is sa_planar_unbounded (with SRID 2147483646).

Remarks

The ST_TransformGeom method transforms a single geometry given the transform definition of the destination. The transformation is performed using the PROJ.4 library. This method can be used in select situations when the appropriate spatial reference systems have not yet been created in the database. If the appropriate spatial reference systems are available, the ST_Transform method is often more appropriate.

Transformations from a lat/long system to a Cartesian system can be problematic for polar points. If the database server is unable to transform a point close to the North or South pole, the latitude value of the point is shifted a small distance (slightly more than 1e-10 radians) away from the pole, and along the same longitude, so that the transformation can succeed.

See also

- "ST_Transform method for type ST_Geometry"

Standards and compatibility

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

Example

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')
```

ST_World method

Returns a geometry that represents all of the points in the spatial reference system.

Note
This method can not be used with geometries in round-Earth spatial reference system.

Syntax

ST_SpatialRefSys::**ST_World**(*srid*)

Parameters

Name	Type	Description
srid	INT	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.

  The spatial reference system identifier of the result is the given by parameter *srid*.

See also

- "ST_IsWorld method for type ST_Surface"

Standards and compatibility

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

Example

The following example returns the result `Polygon ((-1000000 -1000000, 1000000 -1000000, 1000000 1000000, -1000000 1000000, -1000000 -1000000))`.

```
SELECT ST_SpatialRefSys::ST_World(0)
```

ST_Surface type

The ST_Surface type is a supertype for 2-dimensional geometry types. The ST_Surface type is not instantiable.

Direct supertype

- "ST_Geometry type"

Direct subtypes

- "ST_CurvePolygon type"

Methods

- Methods of ST_Surface:

ST_Area	ST_Centroid	ST_IsWorld	ST_Perimeter
ST_PointOnSurface			

- All methods of "ST_Geometry type" can also be called on a ST_Surface type:

ST_Affine	ST_AsBinary	ST_AsGML	ST_AsGeoJSON
ST_AsKML	ST_AsSVG	ST_AsSVGAggr	ST_AsText
ST_AsWKB	ST_AsWKT	ST_AsXML	ST_Boundary
ST_Contains	ST_ContainsFilter	ST_ConvexHull	ST_ConvexHullAggr
ST_CoordDim	ST_CoveredBy	ST_CoveredByFilter	ST_Covers
ST_CoversFilter	ST_Crosses	ST_Difference	ST_Dimension
ST_Disjoint	ST_Distance	ST_Envelope	ST_EnvelopeAggr
ST_Equals	ST_EqualsFilter	ST_GeomFromBinary	ST_GeomFromShape
ST_GeomFromText	ST_GeomFromWKB	ST_GeomFromWKT	ST_GeometryType
ST_GeometryType-FromBaseType	ST_Intersection	ST_IntersectionAggr	ST_Intersects
ST_IntersectsFilter	ST_IntersectsRect	ST_Is3D	ST_IsEmpty
ST_IsMeasured	ST_IsSimple	ST_IsValid	ST_LatNorth
ST_LatSouth	ST_LinearHash	ST_LinearUnHash	ST_LoadConfigura-tionData
ST_LongEast	ST_LongWest	ST_MMax	ST_MMin
ST_OrderingEquals	ST_Overlaps	ST_Relate	ST_Reverse
ST_SRID	ST_SRIDFromBase-Type	ST_SnapToGrid	ST_SymDifference
ST_ToCircular	ST_ToCompound	ST_ToCurve	ST_ToCurvePoly
ST_ToGeomColl	ST_ToLineString	ST_ToMultiCurve	ST_ToMultiLine
ST_ToMultiPoint	ST_ToMultiPolygon	ST_ToMultiSurface	ST_ToPoint
ST_ToPolygon	ST_ToSurface	ST_Touches	ST_Transform
ST_Union	ST_UnionAggr	ST_Within	ST_WithinDistance
ST_WithinDistanceFil-ter	ST_WithinFilter	ST_XMax	ST_XMin
ST_YMax	ST_YMin	ST_ZMax	ST_ZMin

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 8.1

ST_Area method

Calculates the area of a surface in the specified units.

Syntax

surface-expression.**ST_Area**([*unit-name*])

Parameters

Name	Type	Description
unit-name	VARCHAR(128)	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.

Remarks

The ST_Area method computes the area of a surface. The units used to represent the area are based on the specified linear unit of measure. For example, if the specified linear unit of measure is feet, the unit used for area is square feet.

Note
This method can not be used with geometries in round-Earth spatial reference system.

See also

- “ST_Perimeter method for type ST_Surface”
- “ST_Area method for type ST_MultiSurface”
- “ST_Length method for type ST_Curve”

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 8.1.2

Example

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
```

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.

  The spatial reference system identifier of the result is the same as the spatial reference system of the *surface-expression*.

Remarks

Returns the ST_Point value that is the mathematical centroid of the surface value. Note that this point will not necessarily be a point on the surface.

Note
This method can not be used with geometries in round-Earth spatial reference system.

See also

- "ST_Centroid method for type ST_MultiSurface"
- "ST_PointOnSurface method for type ST_Surface"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 8.1.4

Example

The following example returns the result `Point (5 4.666667)`.

```
SELECT TREAT( Shape as ST_Surface ).ST_Centroid()
FROM SpatialShapes WHERE ShapeID = 22
```

ST_IsWorld method

Test if the ST_Surface covers the entire space.

Note
This method can not be used with geometries in round-Earth spatial reference system.

Syntax

surface-expression.**ST_IsWorld**()

Returns

- **BIT** Returns 1 if the surface covers the entire space, otherwise 0.

See also

- "ST_World method for type ST_SpatialRefSys"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 8.1.6

Example

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()
```

ST_Perimeter method

Calculates the perimeter of a surface in the specified units.

Syntax

surface-expression.**ST_Perimeter**([*unit-name*])

Parameters

Name	Type	Description
unit-name	VAR-CHAR(128)	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.

Remarks

The ST_Perimeter method returns the length of the perimeter of a surface in the units identified by the *unit-name* parameter. If the surface is empty, then NULL is returned.

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

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

Note
If the *surface-expression* is an empty geometry (ST_IsEmpty()=1), then this method returns NULL.

Note
By default, ST_Perimeter uses the original format for a geometry, if it is available. Otherwise, the internal format is used. For more information about internal and original formats, see "STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_Perimeter method for type ST_MultiSurface"
- "ST_Boundary method for type ST_Geometry"
- "ST_Length method for type ST_Curve"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 8.1.3

Example

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
```

ST_PointOnSurface method

Returns an ST_Point value that is guaranteed to spatially intersect the ST_Surface value.

Note
If the *surface-expression* contains circularstrings, then these are interpolated to line strings.

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.

The spatial reference system identifier of the result is the same as the spatial reference system of the *surface-expression*.

See also

- "ST_PointOnSurface method for type ST_MultiSurface"
- "ST_Centroid method for type ST_Surface"
- "ST_Intersects method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 8.1.5

Example

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()
```

Spatial compatibility functions

The SQL/MM standard defines a number of functions that can be used to perform spatial operations. In most cases, these functions duplicate functionality of methods or constructors of spatial data types.

Functions

Name	Description
"ST_BdMPolyFromText function [Spatial]"	Returns an ST_MultiPolygon value built from the Well Known Text (WKT) representation of a multilinestring.
"ST_BdMPolyFromWKB function [Spatial]"	Returns an ST_MultiPolygon value built from the Well Known Binary (WKB) representation of a multilinestring.
"ST_BdPolyFromText function [Spatial]"	Returns an ST_Polygon value built from the Well Known Text (WKT) representation of a multilinestring.
"ST_BdPolyFromWKB function [Spatial]"	Returns an ST_Polygon value built from the Well Known Binary (WKB) representation of a multilinestring.
"ST_CPolyFromText function [Spatial]"	Returns an ST_CurvePolygon value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_CurvePolygon
"ST_CPolyFromWKB function [Spatial]"	Returns an ST_CurvePolygon value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_CurvePolygon

Name	Description
"ST_CircularFromTxt function [Spatial]"	Returns an ST_CircularString value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_CircularString
"ST_CircularFromWKB function [Spatial]"	Returns an ST_CircularString value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_CircularString
"ST_CompoundFromTxt function [Spatial]"	Returns an ST_CompoundCurve value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_CompoundCurve
"ST_CompoundFromWKB function [Spatial]"	Returns an ST_CompoundCurve value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_CompoundCurve
"ST_GeomCollFromTxt function [Spatial]"	Returns an ST_GeomCollection value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_GeomCollection
"ST_GeomCollFromWKB function [Spatial]"	Returns an ST_GeomCollection value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_GeomCollection
"ST_GeomFromText function [Spatial]"	Returns an ST_Geometry value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_Geometry
"ST_GeomFromWKB function [Spatial]"	Returns an ST_Geometry value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_Geometry
"ST_LineFromText function [Spatial]"	Returns an ST_LineString value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_LineString
"ST_LineFromWKB function [Spatial]"	Returns an ST_LineString value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_LineString
"ST_MCurveFromText function [Spatial]"	Returns an ST_MultiCurve value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_MultiCurve

Name	**Description**
“ST_MCurveFromWKB function [Spatial]”	Returns an ST_MultiCurve value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_MultiCurve
“ST_MLineFromText function [Spatial]”	Returns an ST_MultiLineString value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_MultiLineString
“ST_MLineFromWKB function [Spatial]”	Returns an ST_MultiLineString value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_MultiLineString
“ST_MPointFromText function [Spatial]”	Returns an ST_MultiPoint value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_MultiPoint
“ST_MPointFromWKB function [Spatial]”	Returns an ST_MultiPoint value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_MultiPoint
“ST_MPolyFromText function [Spatial]”	Returns an ST_MultiPolygon value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_MultiPolygon
“ST_MPolyFromWKB function [Spatial]”	Returns an ST_MultiPolygon value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_MultiPolygon
“ST_MSurfaceFromTxt function [Spatial]”	Returns an ST_MultiSurface value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_MultiSurface
“ST_MSurfaceFromWKB function [Spatial]”	Returns an ST_MultiSurface value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_MultiSurface
“ST_OrderingEquals function [Spatial]”	Tests if a geometry is identical to another geometry.
“ST_PointFromText function [Spatial]”	Returns an ST_Point value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_Point
“ST_PointFromWKB function [Spatial]”	Returns an ST_Point value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_Point

Name	Description
"ST_PolyFromText function [Spatial]"	Returns an ST_Polygon value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_Polygon
"ST_PolyFromWKB function [Spatial]"	Returns an ST_Polygon value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_Polygon

ST_BdMPolyFromText function [Spatial]

Returns an ST_MultiPolygon value built from the Well Known Text (WKT) representation of a multilinestring.

Syntax

[DBO.]ST_BdMPolyFromText(*wkt*[, *srid*])

Parameters

Name	Type	Description
wkt	LONG VARCHAR	The WKT representation of a multilinestring value.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_MultiPolygon** Returns an ST_MultiPolygon value built from the WKT representation of a multilinestring.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_BdMPolyFromText function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_BdMPolyFromText( awkt LONG VARCHAR, srid INT DEFAULT
0 )
RETURNS ST_MultiPolygon
BEGIN
    DECLARE mls ST_MultiLineString;
    SET mls = NEW ST_MultiLineString( awkt, srid );
    RETURN NEW ST_MultiPolygon( mls );
END
```

Note
The ST_BdMPolyFromText function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_Polygon constructor"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.6.7

ST_BdMPolyFromWKB function [Spatial]

Returns an ST_MultiPolygon value built from the Well Known Binary (WKB) representation of a multilinestring.

Syntax

[**DBO.**]**ST_BdMPolyFromWKB**(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	The WKB representation of a multilinestring value.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_MultiPolygon** Returns an ST_MultiPolygon value built from the WKB representation of a multilinestring.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_BdMPolyFromWKB function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_BdMPolyFromWKB( awkb LONG BINARY, srid INT DEFAULT 0 )
RETURNS ST_MultiPolygon
BEGIN
    DECLARE mls ST_MultiLineString;
    SET mls = NEW ST_MultiLineString( awkb, srid );
    RETURN NEW ST_MultiPolygon( mls );
END
```

Note
The ST_BdMPolyFromWKB function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_Polygon constructor"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.6.8

ST_BdPolyFromText function [Spatial]

Returns an ST_Polygon value built from the Well Known Text (WKT) representation of a multilinestring.

Syntax

[**DBO.**]**ST_BdPolyFromText**(*wkt*[, *srid*])

Parameters

Name	Type	Description
wkt	LONG VARCHAR	The WKT representation of a multilinestring value.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_Polygon** Returns an ST_Polygon value built from the WKT representation of a multilinestring.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_BdPolyFromText function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_BdPolyFromText( awkt LONG VARCHAR, srid INT DEFAULT
0 )
RETURNS ST_Polygon
BEGIN
    DECLARE mls ST_MultiLineString;
    SET mls = NEW ST_MultiLineString( awkt, srid );
    RETURN NEW ST_Polygon( mls );
END
```

Note
The ST_BdPolyFromText function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_Polygon constructor"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 8.3.9

ST_BdPolyFromWKB function [Spatial]

Returns an ST_Polygon value built from the Well Known Binary (WKB) representation of a multilinestring.

Syntax

[**DBO**.]**ST_BdPolyFromWKB**(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	The WKB representation of a multilinestring value.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_Polygon** Returns an ST_Polygon value built from the WKB representation of a multilinestring.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_BdPolyFromWKB function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_BdPolyFromWKB( awkb LONG BINARY, srid INT DEFAULT 0 )
RETURNS ST_Polygon
BEGIN
    DECLARE mls ST_MultiLineString;
    SET mls = NEW ST_MultiLineString( awkb, srid );
    RETURN NEW ST_Polygon( mls );
END
```

Note
The ST_BdPolyFromWKB function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_Polygon constructor"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 8.3.10

ST_CPolyFromText function [Spatial]

Returns an ST_CurvePolygon value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_CurvePolygon

Syntax

[**DBO.**]**ST_CPolyFromText**(*wkt*[, *srid*])

Parameters

Name	Type	Description
wkt	LONG VARCHAR	The WKT representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_CurvePolygon** Returns an ST_CurvePolygon value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_CPolyFromText function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_CPolyFromText( wkt LONG VARCHAR, srid INT DEFAULT 0 )
RETURNS ST_CurvePolygon
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromText( wkt, srid );
    RETURN CAST( geo AS ST_CurvePolygon);
END
```

Note
The ST_CPolyFromText function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_CurvePolygon constructor"
- "ST_GeomFromText method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 8.2.8

ST_CPolyFromWKB function [Spatial]

Returns an ST_CurvePolygon value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_CurvePolygon

Syntax

[**DBO.**]**ST_CPolyFromWKB**(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	The WKB representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_CurvePolygon** Returns an ST_CurvePolygon value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_CPolyFromWKB function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_CPolyFromWKB( wkb LONG BINARY, srid INT DEFAULT 0 )
RETURNS ST_CurvePolygon
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromWKB( wkb, srid );
    RETURN CAST( geo AS ST_CurvePolygon);
END
```

Note
The ST_CPolyFromWKB function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_CurvePolygon constructor"
- "ST_GeomFromWKB method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 8.2.9

ST_CircularFromTxt function [Spatial]

Returns an ST_CircularString value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_CircularString

Syntax

[**DBO.**]**ST_CircularFromTxt**(*wkt*[, *srid*])

Parameters

Name	Type	Description
wkt	LONG VARCHAR	The WKT representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_CircularString** Returns an ST_CircularString value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_CircularFromTxt function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_CircularFromTxt( wkt LONG VARCHAR, srid INT DEFAULT
0 )
RETURNS ST_CircularString
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromText( wkt, srid );
    RETURN CAST( geo AS ST_CircularString);
END
```

Note
The ST_CircularFromTxt function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_CircularString constructor"
- "ST_GeomFromText method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 7.3.9

ST_CircularFromWKB function [Spatial]

Returns an ST_CircularString value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_CircularString

Syntax

[**DBO.**]**ST_CircularFromWKB**(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	The WKB representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_CircularString** Returns an ST_CircularString value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_CircularFromWKB function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_CircularFromWKB( wkb LONG BINARY, srid INT DEFAULT 0 )
RETURNS ST_CircularString
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromWKB( wkb, srid );
    RETURN CAST( geo AS ST_CircularString);
END
```

Note
The ST_CircularFromWKB function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_CircularString constructor"
- "ST_GeomFromWKB method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 7.3.10

ST_CompoundFromTxt function [Spatial]

Returns an ST_CompoundCurve value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_CompoundCurve

Syntax

[**DBO.**]**ST_CompoundFromTxt**(*wkt*[, *srid*])

Parameters

Name	Type	Description
wkt	LONG VARCHAR	The WKT representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_CompoundCurve** Returns an ST_CompoundCurve value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_CompoundFromTxt function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_CompoundFromTxt( wkt LONG VARCHAR, srid INT DEFAULT
0 )
RETURNS ST_CompoundCurve
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromText( wkt, srid );
    RETURN CAST( geo AS ST_CompoundCurve);
END
```

Note
The ST_CompoundFromTxt function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_CompoundCurve constructor"
- "ST_GeomFromText method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 7.4.8

ST_CompoundFromWKB function [Spatial]

Returns an ST_CompoundCurve value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_CompoundCurve

Syntax

[**DBO.**]**ST_CompoundFromWKB**(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	The WKB representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_CompoundCurve** Returns an ST_CompoundCurve value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_CompoundFromWKB function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_CompoundFromWKB( wkb LONG BINARY, srid INT DEFAULT 0 )
RETURNS ST_CompoundCurve
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromWKB( wkb, srid );
    RETURN CAST( geo AS ST_CompoundCurve);
END
```

Note
The ST_CompoundFromWKB function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_CompoundCurve constructor"
- "ST_GeomFromWKB method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 7.4.9

ST_GeomCollFromTxt function [Spatial]

Returns an ST_GeomCollection value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_GeomCollection

Syntax

[**DBO.**]**ST_GeomCollFromTxt**(*wkt*[, *srid*])

Parameters

Name	Type	Description
wkt	LONG VARCHAR	The WKT representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_GeomCollection** Returns an ST_GeomCollection value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_GeomCollFromTxt function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_GeomCollFromTxt( wkt LONG VARCHAR, srid INT DEFAULT
0 )
RETURNS ST_GeomCollection
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromText( wkt, srid );
    RETURN CAST( geo AS ST_GeomCollection);
END
```

Note
The ST_GeomCollFromTxt function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_GeomCollection constructor"
- "ST_GeomFromText method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.1.6

ST_GeomCollFromWKB function [Spatial]

Returns an ST_GeomCollection value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_GeomCollection

Syntax

[**DBO.**]**ST_GeomCollFromWKB**(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	The WKB representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_GeomCollection** Returns an ST_GeomCollection value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_GeomCollFromWKB function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_GeomCollFromWKB( wkb LONG BINARY, srid INT DEFAULT 0 )
RETURNS ST_GeomCollection
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromWKB( wkb, srid );
    RETURN CAST( geo AS ST_GeomCollection);
END
```

Note
The ST_GeomCollFromWKB function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_GeomCollection constructor"
- "ST_GeomFromWKB method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.1.7

ST_GeomFromText function [Spatial]

Returns an ST_Geometry value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_Geometry

Syntax

[**DBO.**]**ST_GeomFromText**(*wkt*[, *srid*])

Parameters

Name	Type	Description
wkt	LONG VARCHAR	The WKT representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_Geometry** Returns an ST_Geometry value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_GeomFromText function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_GeomFromText( wkt LONG VARCHAR, srid INT DEFAULT 0 )
RETURNS ST_Geometry
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromText( wkt, srid );
    RETURN CAST( geo AS ST_Geometry);
END
```

Note
The ST_GeomFromText function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_GeomFromText method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.40

ST_GeomFromWKB function [Spatial]

Returns an ST_Geometry value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_Geometry

Syntax

[**DBO.**]**ST_GeomFromWKB**(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	The WKB representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_Geometry** Returns an ST_Geometry value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_GeomFromWKB function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_GeomFromWKB( wkb LONG BINARY, srid INT DEFAULT 0 )
RETURNS ST_Geometry
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromWKB( wkb, srid );
    RETURN CAST( geo AS ST_Geometry);
END
```

Note
The ST_GeomFromWKB function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_GeomFromWKB method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.41

ST_LineFromText function [Spatial]

Returns an ST_LineString value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_LineString

Syntax

[**DBO.**]**ST_LineFromText**(*wkt*[, *srid*])

Parameters

Name	Type	Description
wkt	LONG VARCHAR	The WKT representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_LineString** Returns an ST_LineString value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_LineFromText function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_LineFromText( wkt LONG VARCHAR, srid INT DEFAULT 0 )
RETURNS ST_LineString
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromText( wkt, srid );
    RETURN CAST( geo AS ST_LineString);
END
```

Note
The ST_LineFromText function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_LineString constructor"
- "ST_GeomFromText method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 7.2.8

ST_LineFromWKB function [Spatial]

Returns an ST_LineString value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_LineString

Syntax

[**DBO.**]**ST_LineFromWKB**(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	The WKB representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_LineString** Returns an ST_LineString value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_LineFromWKB function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_LineFromWKB( wkb LONG BINARY, srid INT DEFAULT 0 )
RETURNS ST_LineString
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromWKB( wkb, srid );
    RETURN CAST( geo AS ST_LineString);
END
```

Note
The ST_LineFromWKB function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_LineString constructor"
- "ST_GeomFromWKB method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 7.2.9

ST_MCurveFromText function [Spatial]

Returns an ST_MultiCurve value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_MultiCurve

Syntax

[**DBO.**]**ST_MCurveFromText**(*wkt*[, *srid*])

Parameters

Name	Type	Description
wkt	LONG VARCHAR	The WKT representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_MultiCurve** Returns an ST_MultiCurve value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_MCurveFromText function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_MCurveFromText( wkt LONG VARCHAR, srid INT DEFAULT 0 )
RETURNS ST_MultiCurve
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromText( wkt, srid );
    RETURN CAST( geo AS ST_MultiCurve);
END
```

Note
The ST_MCurveFromText function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_MultiCurve constructor"
- "ST_GeomFromText method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.3.6

ST_MCurveFromWKB function [Spatial]

Returns an ST_MultiCurve value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_MultiCurve

Syntax

[**DBO.**]**ST_MCurveFromWKB**(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	The WKB representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_MultiCurve** Returns an ST_MultiCurve value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_MCurveFromWKB function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_MCurveFromWKB( wkb LONG BINARY, srid INT DEFAULT 0 )
RETURNS ST_MultiCurve
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromWKB( wkb, srid );
    RETURN CAST( geo AS ST_MultiCurve);
END
```

Note
The ST_MCurveFromWKB function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_MultiCurve constructor"
- "ST_GeomFromWKB method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.3.7

ST_MLineFromText function [Spatial]

Returns an ST_MultiLineString value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_MultiLineString

Syntax

[**DBO.**]**ST_MLineFromText**(*wkt*[, *srid*])

Parameters

Name	Type	Description
wkt	LONG VARCHAR	The WKT representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_MultiLineString** Returns an ST_MultiLineString value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_MLineFromText function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_MLineFromText( wkt LONG VARCHAR, srid INT DEFAULT 0 )
RETURNS ST_MultiLineString
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromText( wkt, srid );
    RETURN CAST( geo AS ST_MultiLineString);
END
```

Note
The ST_MLineFromText function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_MultiLineString constructor"
- "ST_GeomFromText method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.4.4

ST_MLineFromWKB function [Spatial]

Returns an ST_MultiLineString value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_MultiLineString

Syntax

[**DBO.**]**ST_MLineFromWKB**(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	The WKB representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_MultiLineString** Returns an ST_MultiLineString value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_MLineFromWKB function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_MLineFromWKB( wkb LONG BINARY, srid INT DEFAULT 0 )
RETURNS ST_MultiLineString
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromWKB( wkb, srid );
    RETURN CAST( geo AS ST_MultiLineString);
END
```

Note
The ST_MLineFromWKB function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_MultiLineString constructor"
- "ST_GeomFromWKB method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.4.5

ST_MPointFromText function [Spatial]

Returns an ST_MultiPoint value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_MultiPoint

Syntax

[**DBO.**]**ST_MPointFromText**(*wkt*[, *srid*])

Parameters

Name	Type	Description
wkt	LONG VARCHAR	The WKT representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_MultiPoint** Returns an ST_MultiPoint value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_MPointFromText function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_MPointFromText( wkt LONG VARCHAR, srid INT DEFAULT 0 )
RETURNS ST_MultiPoint
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromText( wkt, srid );
    RETURN CAST( geo AS ST_MultiPoint);
END
```

Note
The ST_MPointFromText function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_MultiPoint constructor"
- "ST_GeomFromText method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.2.4

ST_MPointFromWKB function [Spatial]

Returns an ST_MultiPoint value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_MultiPoint

Syntax

[**DBO.**]**ST_MPointFromWKB**(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	The WKB representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_MultiPoint**   Returns an ST_MultiPoint value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_MPointFromWKB function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_MPointFromWKB( wkb LONG BINARY, srid INT DEFAULT 0 )
RETURNS ST_MultiPoint
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromWKB( wkb, srid );
    RETURN CAST( geo AS ST_MultiPoint);
END
```

Note
The ST_MPointFromWKB function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_MultiPoint constructor"
- "ST_GeomFromWKB method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)**   9.2.5

ST_MPolyFromText function [Spatial]

Returns an ST_MultiPolygon value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_MultiPolygon

Syntax

[**DBO.**]**ST_MPolyFromText**(*wkt*[, *srid*])

Parameters

Name	Type	Description
wkt	LONG VARCHAR	The WKT representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_MultiPolygon** Returns an ST_MultiPolygon value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_MPolyFromText function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_MPolyFromText( wkt LONG VARCHAR, srid INT DEFAULT 0 )
RETURNS ST_MultiPolygon
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromText( wkt, srid );
    RETURN CAST( geo AS ST_MultiPolygon);
END
```

Note
The ST_MPolyFromText function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_MultiPolygon constructor"
- "ST_GeomFromText method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.6.4

ST_MPolyFromWKB function [Spatial]

Returns an ST_MultiPolygon value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_MultiPolygon

Syntax

[**DBO.**]**ST_MPolyFromWKB**(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	The WKB representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_MultiPolygon** Returns an ST_MultiPolygon value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_MPolyFromWKB function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_MPolyFromWKB( wkb LONG BINARY, srid INT DEFAULT 0 )
RETURNS ST_MultiPolygon
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromWKB( wkb, srid );
    RETURN CAST( geo AS ST_MultiPolygon);
END
```

Note
The ST_MPolyFromWKB function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_MultiPolygon constructor"
- "ST_GeomFromWKB method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.6.5

ST_MSurfaceFromTxt function [Spatial]

Returns an ST_MultiSurface value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_MultiSurface

Syntax

[DBO.]ST_MSurfaceFromTxt(*wkt*[, *srid*])

Parameters

Name	Type	Description
wkt	LONG VARCHAR	The WKT representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_MultiSurface** Returns an ST_MultiSurface value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_MSurfaceFromTxt function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_MSurfaceFromTxt( wkt LONG VARCHAR, srid INT DEFAULT
0 )
RETURNS ST_MultiSurface
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromText( wkt, srid );
    RETURN CAST( geo AS ST_MultiSurface);
END
```

Note
The ST_MSurfaceFromTxt function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_MultiSurface constructor"
- "ST_GeomFromText method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.5.8

ST_MSurfaceFromWKB function [Spatial]

Returns an ST_MultiSurface value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_MultiSurface

Syntax

[**DBO.**]**ST_MSurfaceFromWKB**(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	The WKB representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_MultiSurface** Returns an ST_MultiSurface value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_MSurfaceFromWKB function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_MSurfaceFromWKB( wkb LONG BINARY, srid INT DEFAULT 0 )
RETURNS ST_MultiSurface
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromWKB( wkb, srid );
    RETURN CAST( geo AS ST_MultiSurface);
END
```

Note
The ST_MSurfaceFromWKB function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_MultiSurface constructor"
- "ST_GeomFromWKB method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 9.5.9

ST_OrderingEquals function [Spatial]

Tests if a geometry is identical to another geometry.

Syntax

[**DBO**.]**ST_OrderingEquals**(*geo1*,*geo2*)

Parameters

Name	Type	Description
geo1	ST_Geometry	The first geometry value that is to be ordered.
geo2	ST_Geometry	The second geometry value that is to be ordered.

Returns

- **INT** Returns 1 if *geo1* is exactly equal to *geo2*, otherwise 0.

Remarks

The ST_OrderingEquals function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_OrderingEquals( geo1 ST_Geometry, geo2 ST_Geometry )
RETURNS INT
BEGIN
    RETURN geo1.ST_OrderingEquals( geo2 );
END
```

Note
The ST_OrderingEquals function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_OrderingEquals method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 5.1.43

ST_PointFromText function [Spatial]

Returns an ST_Point value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_Point

Syntax

[**DBO**.]**ST_PointFromText**(*wkt*[, *srid*])

Parameters

Name	Type	Description
wkt	LONG VARCHAR	The WKT representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_Point** Returns an ST_Point value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_PointFromText function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_PointFromText( wkt LONG VARCHAR, srid INT DEFAULT 0 )
RETURNS ST_Point
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromText( wkt, srid );
    RETURN CAST( geo AS ST_Point);
END
```

Note
The ST_PointFromText function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_Point constructor"
- "ST_GeomFromText method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.8

ST_PointFromWKB function [Spatial]

Returns an ST_Point value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_Point

Syntax

[DBO.]ST_PointFromWKB(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	The WKB representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_Point** Returns an ST_Point value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_PointFromWKB function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_PointFromWKB( wkb LONG BINARY, srid INT DEFAULT 0 )
RETURNS ST_Point
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromWKB( wkb, srid );
    RETURN CAST( geo AS ST_Point);
END
```

Note
The ST_PointFromWKB function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_Point constructor"
- "ST_GeomFromWKB method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 6.1.9

ST_PolyFromText function [Spatial]

Returns an ST_Polygon value, which is transformed from a LONG VARCHAR value containing the Well Known Text (WKT) representation of an ST_Polygon

Syntax

[**DBO**.]**ST_PolyFromText**(*wkt*[, *srid*])

Parameters

Name	Type	Description
wkt	LONG VARCHAR	The WKT representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_Polygon** Returns an ST_Polygon value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_PolyFromText function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_PolyFromText( wkt LONG VARCHAR, srid INT DEFAULT 0 )
RETURNS ST_Polygon
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromText( wkt, srid );
    RETURN CAST( geo AS ST_Polygon);
END
```

Note
The ST_PolyFromText function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_Polygon constructor"
- "ST_GeomFromText method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 8.3.6

ST_PolyFromWKB function [Spatial]

Returns an ST_Polygon value, which is transformed from a LONG BINARY value containing the Well Known Binary (WKB) representation of an ST_Polygon

Syntax

[DBO.]ST_PolyFromWKB(*wkb*[, *srid*])

Parameters

Name	Type	Description
wkb	LONG BINARY	The WKB representation.
srid	INT	The SRID of the result. If not specified, the default is 0.

Returns

- **ST_Polygon** Returns an ST_Polygon value created from the input string.

  The spatial reference system identifier of the result is the given by parameter *srid*.

Remarks

The ST_PolyFromWKB function is equivalent to the following:

```
CREATE FUNCTION DBO.ST_PolyFromWKB( wkb LONG BINARY, srid INT DEFAULT 0 )
RETURNS ST_Polygon
BEGIN
    DECLARE geo ST_Geometry;

    set geo = ST_Geometry::ST_GeomFromWKB( wkb, srid );
    RETURN CAST( geo AS ST_Polygon);
END
```

Note
The ST_PolyFromWKB function is not present by default in newly created databases. Use the sa_install_feature system procedure to install the spatial SQL compatibility functions. See "sa_install_feature system procedure" [*SQL Anywhere Server - SQL Reference*].

See also

- "ST_Polygon constructor"
- "ST_GeomFromWKB method for type ST_Geometry"

Standards and compatibility

- **SQL/MM (ISO/IEC 13249-3: 2006)** 8.3.7

List of all supported methods

The following is a list of all supported spatial methods. An X in the Round-Earth column indicates that the method is also supported on round-Earth spatial reference systems. The SQL/MM column reflects compliance with the SQL/MM standard (ISO/IEC 13249-3: 2006).

Method	Type	Description	Round-Earth	SQL/MM
ST_Affine method	"ST_Geometry type"	Perform an affine transformation that performs rotation, translation and scaling in one call.		Vendor extension
ST_Area method	"ST_MultiSurface type"	Computes the area of the multi-surface in the specified units.		9.5.3
ST_Area method	"ST_Surface type"	Calculates the area of a surface in the specified units.		8.1.2
ST_AsBinary method	"ST_Geometry type"	Returns the WKB representation of an ST_Geometry value.	X	5.1.37
ST_AsGML method	"ST_Geometry type"	Returns the GML representation of an ST_Geometry value.	X	5.1.39
ST_AsGeoJSON method	"ST_Geometry type"	Returns a string representing a geometry in JSON format.	X	Vendor extension
ST_AsKML method	"ST_Geometry type"	Returns the KML representation of an ST_Geometry value.	X	5.1.39
ST_AsSVG method	"ST_Geometry type"	Returns an SVG figure representing a geometry value.	X	Vendor extension
ST_AsText method	"ST_Geometry type"	Returns the text representation of an ST_Geometry value.	X	5.1.35
ST_AsWKB method	"ST_Geometry type"	Returns the WKB representation of an ST_Geometry value.	X	Vendor extension
ST_AsWKT method	"ST_Geometry type"	Returns the WKT representation of an ST_Geometry value.	X	Vendor extension
ST_AsXML method	"ST_Geometry type"	Returns the XML representation of an ST_Geometry value.	X	Vendor extension
ST_Boundary method	"ST_Geometry type"	Returns the boundary of the geometry value.		5.1.14
ST_Centroid method	"ST_MultiSurface type"	Computes the ST_Point that is the mathematical centroid of the multi-surface.		9.5.5

Method	Type	Description	Round-Earth	SQL/MM
ST_Centroid method	"ST_Surface type"	Returns the ST_Point value that is the mathematical centroid of the surface value.		8.1.4
ST_Contains method	"ST_Geometry type"	Tests if a geometry value spatially contains another geometry value.		5.1.31
ST_ContainsFilter method	"ST_Geometry type"	An inexpensive test if a geometry might contain another.		Vendor extension
ST_ConvexHull method	"ST_Geometry type"	Returns the convex hull of the geometry value.		5.1.16
ST_CoordDim method	"ST_Geometry type"	Returns the number of coordinate dimensions stored with each point of the ST_Geometry value.	X	5.1.3
ST_CoveredBy method	"ST_Geometry type"	Tests if a geometry value is spatially covered by another geometry value.	X	Vendor extension
ST_CoveredByFilter method	"ST_Geometry type"	An inexpensive test if a geometry might be covered by another.	X	Vendor extension
ST_Covers method	"ST_Geometry type"	Tests if a geometry value spatially covers another geometry value.	X	Vendor extension
ST_CoversFilter method	"ST_Geometry type"	An inexpensive test if a geometry might cover another.	X	Vendor extension
ST_Crosses method	"ST_Geometry type"	Tests if a geometry value crosses another geometry value.		5.1.29
ST_CurveN method	"ST_CompoundCurve type"	Returns the *n*th curve in the compound curve.	X	7.4.5
ST_CurvePolyToPoly method	"ST_CurvePolygon type"	Returns the interpolation of the curve polygon as a polygon.	X	8.2.7
ST_CurveToLine method	"ST_Curve type"	Returns the ST_LineString interpolation of an ST_Curve value.	X	7.1.7
ST_Difference method	"ST_Geometry type"	Returns the geometry value that represents the point set difference of two geometries.	X	5.1.20

Method	Type	Description	Round-Earth	SQL/MM
ST_Dimension method	"ST_Geometry type"	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.	X	5.1.2
ST_Disjoint method	"ST_Geometry type"	Test if a geometry value is spatially disjoint from another value.	X	5.1.26
ST_Distance method	"ST_Geometry type"	Returns the smallest distance between the ${selfexpr} and the specified geometry value.	X	5.1.23
ST_EndPoint method	"ST_Curve type"	Returns an ST_Point value that is the end point of the ST_Curve value.	X	7.1.4
ST_Envelope method	"ST_Geometry type"	Returns the bounding rectangle for the geometry value.		5.1.15
ST_Equals method	"ST_Geometry type"	Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.	X	5.1.24
ST_EqualsFilter method	"ST_Geometry type"	An inexpensive test if a geometry is equal to another.	X	Vendor extension
ST_ExteriorRing method	"ST_CurvePolygon type"	Retrieves or modifies the exterior ring.	X	8.2.3
ST_ExteriorRing method	"ST_Polygon type"	Retrieve or modify the exterior ring.	X	8.3.3
ST_GeometryN method	"ST_GeomCollection type"	Returns the *n*th geometry in the geometry collection.	X	9.1.5
ST_GeometryType method	"ST_Geometry type"	Returns the name of the type of the ST_Geometry value.	X	5.1.4
ST_InteriorRingN method	"ST_CurvePolygon type"	Returns the *n*th interior ring in the curve polygon.	X	8.2.6
ST_InteriorRingN method	"ST_Polygon type"	Returns the *n*th interior ring in the polygon.	X	8.3.5

Method	Type	Description	Round-Earth	SQL/MM
ST_Intersection method	“ST_Geometry type”	Returns the geometry value that represents the point set intersection of two geometries.	X	5.1.18
ST_Intersects method	“ST_Geometry type”	Test if a geometry value spatially intersects another value.	X	5.1.27
ST_IntersectsFilter method	“ST_Geometry type”	An inexpensive test if the two geometries might intersect.	X	Vendor extension
ST_IntersectsRect method	“ST_Geometry type”	Test if a geometry intersects a rectangle.	X	Vendor extension
ST_Is3D method	“ST_Geometry type”	Determines if the geometry value has Z coordinate values.	X	5.1.10
ST_IsClosed method	“ST_Curve type”	Test if the ST_Curve value is closed. A curve is closed if the start and end points are coincident.	X	7.1.5
ST_IsClosed method	“ST_MultiCurve type”	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.	X	9.3.3
ST_IsEmpty method	“ST_Geometry type”	Determines whether the geometry value represents an empty set.	X	5.1.7
ST_IsMeasured method	“ST_Geometry type”	Determines if the geometry value has associated measure values.	X	5.1.11
ST_IsRing method	“ST_Curve type”	Tests if the ST_Curve value is a ring. A curve is a ring if it is closed and simple (no self intersections).	X	7.1.6
ST_IsSimple method	“ST_Geometry type”	Determines whether the geometry value is simple (containing no self intersections or other irregularities).	X	5.1.8
ST_IsValid method	“ST_Geometry type”	Determines whether the geometry is a valid spatial object.	X	5.1.9
ST_IsWorld method	“ST_Surface type”	Test if the ST_Surface covers the entire space.		8.1.6

Method	Type	Description	Round-Earth	SQL/MM
ST_Lat method	"ST_Point type"	Returns the latitude coordinate of the ST_Point value.	X	Vendor extension
ST_LatNorth method	"ST_Geometry type"	Retrieves the northernmost latitude of a geometry.	X	Vendor extension
ST_LatSouth method	"ST_Geometry type"	Retrieves the southernmost latitude of a geometry.	X	Vendor extension
ST_Length method	"ST_Curve type"	Retrieves the length measurement of the curve value.	X	7.1.2
ST_Length method	"ST_Multi-Curve type"	Returns the length measurement of the ST_MultiCurve value. The result is measured in the units specified by the parameter.	X	9.3.4
ST_LinearHash method	"ST_Geometry type"	Returns a binary string that is a linear hash of the geometry.	X	Vendor extension
ST_Long method	"ST_Point type"	Returns the longitude coordinate of the ST_Point value.	X	Vendor extension
ST_LongEast method	"ST_Geometry type"	Retrieves the longitude of the eastern boundary of a geometry.	X	Vendor extension
ST_LongWest method	"ST_Geometry type"	Retrieves the longitude of the western boundary of a geometry.	X	Vendor extension
ST_M method	"ST_Point type"	Retrieves or modifies the measure value of a point.	X	6.1.6
ST_MMax method	"ST_Geometry type"	Retrieves the maximum M coordinate value of a geometry.	X	Vendor extension
ST_MMin method	"ST_Geometry type"	Retrieves the minimum M coordinate value of a geometry.	X	Vendor extension
ST_NumCurves method	"ST_Com-poundCurve type"	Returns the number of curves defin-ing the compound curve.	X	7.4.4
ST_NumGeome-tries method	"ST_GeomCol-lection type"	Returns the number of geometries contained in the geometry collection.	X	9.1.4

Method	Type	Description	Round-Earth	SQL/MM
ST_NumInteriorRing method	"ST_CurvePolygon type"	Returns the number of interior rings in the curve polygon.	X	8.2.5
ST_NumPoints method	"ST_CircularString type"	Returns the number of points defining the circularstring.	X	7.3.4
ST_NumPoints method	"ST_LineString type"	Returns the number of points defining the linestring.	X	7.2.4
ST_OrderingEquals method	"ST_Geometry type"	Tests if a geometry is identical to another geometry.	X	5.1.43
ST_Overlaps method	"ST_Geometry type"	Tests if a geometry value overlaps another geometry value.		5.1.32
ST_Perimeter method	"ST_MultiSurface type"	Computes the perimeter of the multi-surface in the specified units.	X	9.5.4
ST_Perimeter method	"ST_Surface type"	Calculates the perimeter of a surface in the specified units.	X	8.1.3
ST_PointN method	"ST_CircularString type"	Returns the *n*th point in the circularstring.	X	7.3.5
ST_PointN method	"ST_LineString type"	Returns the *n*th point in the linestring.	X	7.2.5
ST_PointOnSurface method	"ST_MultiSurface type"	Returns a point that is guaranteed to be on a surface in the multi-surface	X	9.5.6
ST_PointOnSurface method	"ST_Surface type"	Returns an ST_Point value that is guaranteed to spatially intersect the ST_Surface value.	X	8.1.5

Method	Type	Description	Round-Earth	SQL/MM
ST_Relate method	"ST_Geometry type"	Tests if a geometry value is spatially related to another geometry value as specified by the intersection matrix. The ST_Relate method uses a 9-character string from the Dimensionally Extended 9 Intersection Model (DE-9IM) to describe the pair-wise relationship between two spatial data items. For example, the ST_Relate method determines if an intersection occurs between the geometries, and the geometry of the resulting intersection, if it exists. See also: "How spatial relationships work" on page 51.		5.1.25, Vendor extension
ST_Reverse method	"ST_Geometry type"	Returns the geometry with the element order reversed.	X	Vendor extension
ST_SRID method	"ST_Geometry type"	Retrieves or modifies the spatial reference system associated with the geometry value.	X	5.1.5
ST_SnapToGrid method	"ST_Geometry type"	Returns a copy of the geometry with all points snapped to the specified grid.	X	Vendor extension
ST_StartPoint method	"ST_Curve type"	Returns an ST_Point value that is the start point of the ST_Curve value.	X	7.1.3
ST_SymDifference method	"ST_Geometry type"	Returns the geometry value that represents the point set symmetric difference of two geometries.	X	5.1.21
ST_ToCircular method	"ST_Geometry type"	Convert the geometry to a circularstring	X	5.1.33
ST_ToCompound method	"ST_Geometry type"	Converts the geometry to a compound curve.	X	5.1.33
ST_ToCurve method	"ST_Geometry type"	Converts the geometry to a curve.	X	Vendor extension
ST_ToCurvePoly method	"ST_Geometry type"	Converts the geometry to a curve polygon.	X	5.1.33

Method	Type	Description	Round-Earth	SQL/MM
ST_ToGeomColl method	“ST_Geometry type”	Converts the geometry to a geometry collection.	X	5.1.33
ST_ToLineString method	“ST_Geometry type”	Converts the geometry to a linestring.	X	5.1.33
ST_ToMultiCurve method	“ST_Geometry type”	Converts the geometry to a multi-curve value.	X	5.1.33
ST_ToMultiLine method	“ST_Geometry type”	Converts the geometry to a multiline-string value.	X	5.1.33
ST_ToMultiPoint method	“ST_Geometry type”	Converts the geometry to a multi-point value.	X	5.1.33
ST_ToMultiPoly-gon method	“ST_Geometry type”	Converts the geometry to a multi-pol-ygon value.	X	5.1.33
ST_ToMultiSur-face method	“ST_Geometry type”	Converts the geometry to a multi-sur-face value.	X	5.1.33
ST_ToPoint meth-od	“ST_Geometry type”	Converts the geometry to a point.	X	5.1.33
ST_ToPolygon method	“ST_Geometry type”	Converts the geometry to a polygon.	X	5.1.33
ST_ToSurface method	“ST_Geometry type”	Converts the geometry to a surface.	X	Vendor extension
ST_Touches meth-od	“ST_Geometry type”	Tests if a geometry value spatially touches another geometry value.		5.1.28
ST_Transform method	“ST_Geometry type”	Creates a copy of the geometry value transformed into the specified spatial reference system.	X	5.1.6
ST_Union method	“ST_Geometry type”	Returns the geometry value that rep-resents the point set union of two ge-ometries.	X	5.1.19
ST_Within method	“ST_Geometry type”	Tests if a geometry value is spatially contained within another geometry value.		5.1.30

Method	Type	Description	Round-Earth	SQL/MM
ST_WithinDistance method	"ST_Geometry type"	Test if two geometries are within a specified distance of each other.	X	Vendor extension
ST_WithinDistanceFilter method	"ST_Geometry type"	An inexpensive of whether two geometries might be within a specified distance of each other.	X	Vendor extension
ST_WithinFilter method	"ST_Geometry type"	An inexpensive test if a geometry might be within another.		Vendor extension
ST_X method	"ST_Point type"	Retrieves or modifies the X coordinate value of a point.	X	6.1.3
ST_XMax method	"ST_Geometry type"	Retrieves the maximum X coordinate value of a geometry.	X	Vendor extension
ST_XMin method	"ST_Geometry type"	Retrieves the minimum X coordinate value of a geometry.	X	Vendor extension
ST_Y method	"ST_Point type"	Retrieves or modifies the Y coordinate value of a point.	X	6.1.4
ST_YMax method	"ST_Geometry type"	Retrieves the maximum Y coordinate value of a geometry.	X	Vendor extension
ST_YMin method	"ST_Geometry type"	Retrieves the minimum Y coordinate value of a geometry.	X	Vendor extension
ST_Z method	"ST_Point type"	Retrieves or modifies the Z coordinate value of a point.	X	6.1.4, 6.1.5
ST_ZMax method	"ST_Geometry type"	Retrieves the maximum Z coordinate value of a geometry.	X	Vendor extension
ST_ZMin method	"ST_Geometry type"	Retrieves the minimum Z coordinate value of a geometry.	X	Vendor extension

List of all supported constructors

The following is a list of all supported spatial constructors. An X in the Round-Earth column indicates that the method is also supported on round-Earth spatial reference systems. The SQL/MM column reflects compliance with the SQL/MM standard (ISO/IEC 13249-3: 2006).

Constructor	Description	Round-Earth	SQL/MM
“ST_CircularString constructor”	Constructs a circularstring.	X	7.3.2, Vendor extension
“ST_CompoundCurve constructor”	Constructs a compound curve.	X	7.4.2, Vendor extension
“ST_CurvePolygon constructor”	Constructs a curve polygon.	X	8.2.2, Vendor extension
“ST_GeomCollection constructor”	Constructs a geometry collection.	X	9.1.2, Vendor extension
“ST_LineString constructor”	Constructs a linestring.	X	7.2.2, Vendor extension
“ST_MultiCurve constructor”	Constructs a multi curve.	X	9.3.2, Vendor extension
“ST_MultiLineString constructor”	Constructs a multi linestring.	X	9.4.2, Vendor extension
“ST_MultiPoint constructor”	Constructs a multi point.	X	9.2.2, Vendor extension
“ST_MultiPolygon constructor”	Constructs a multi polygon.	X	9.6.2, Vendor extension
“ST_MultiSurface constructor”	Constructs a multi surface.	X	9.5.2, Vendor extension
“ST_Point constructor”	Constructs a point.	X	6.1.2
“ST_Polygon constructor”	Constructs a polygon.	X	8.3.2, Vendor extension

List of static methods

The following is a list of all static methods available for use with spatial data. An X in the Round-Earth column indicates that the method is also supported on round-Earth spatial reference systems. The SQL/MM column reflects compliance with the SQL/MM standard (ISO/IEC 13249-3: 2006).

Method	Type	Description	Round-Earth	SQL/MM
ST_AsSVGAggr method	"ST_Geometry type"	Returns a complete or partial SVG document which renders the geometries in a group.	X	Vendor extension
ST_CompareWKT method	"ST_SpatialRefSys type"	Compares two spatial reference system definitions.	X	Vendor extension
ST_ConvexHullAggr method	"ST_Geometry type"	Returns the convex hull for all of the geometries in a group		Vendor extension
ST_EnvelopeAggr method	"ST_Geometry type"	Returns the bounding rectangle for all of the geometries in a group		Vendor extension
ST_FormatTransformDefinition method	"ST_SpatialRefSys type"	Returns a formatted copy of the transform definition.	X	Vendor extension
ST_FormatWKT method	"ST_SpatialRefSys type"	Returns a formatted copy of the Well Known Text (WKT) definition.	X	Vendor extension
ST_GeomCollectionAggr method	"ST_GeomCollection type"	Returns a geometry collection containing all of the geometries in a group.	X	Vendor extension
ST_GeomFromBinary method	"ST_Geometry type"	Constructs a geometry from a binary string representation.	X	Vendor extension
ST_GeomFromShape method	"ST_Geometry type"	Parses a string containing an ESRI shape record and creates a geometry value of the appropriate type.	X	Vendor extension
ST_GeomFromText method	"ST_Geometry type"	Constructs a geometry from a character string representation.	X	5.1.40
ST_GeomFromWKB method	"ST_Geometry type"	Parse a string containing a WKB or EWKB representation of a geometry and creates a geometry value of the appropriate type.	X	5.1.41
ST_GeomFromWKT method	"ST_Geometry type"	Parses a string containing the WKT or EWKT representation of a geometry and create a geometry value of the appropriate type.	X	Vendor extension

Method	Type	Description	Round-Earth	SQL/MM
ST_GeometryTypeFromBaseType method	"ST_Geometry type"	Parses a string defining the type string.	X	Vendor extension
ST_GetUnProjectedTransformDefinition method	"ST_SpatialRefSys type"	Returns the transform definition of the spatial reference system that is the source of the projection.	X	Vendor extension
ST_IntersectionAggr method	"ST_Geometry type"	Returns the spatial intersection of all of the geometries in a group	X	Vendor extension
ST_LineStringAggr method	"ST_LineString type"	Returns a linestring built from the ordered points in a group.	X	Vendor extension
ST_LinearUnHash method	"ST_Geometry type"	Returns a geometry representing the index hash.	X	Vendor extension
ST_LoadConfigurationData method	"ST_Geometry type"	Returns binary configuration data. For internal use only.	X	Vendor extension
ST_MultiCurveAggr method	"ST_MultiCurve type"	Returns a multicurve containing all of the curves in a group.	X	Vendor extension
ST_MultiLineStringAggr method	"ST_MultiLineString type"	Returns a multilinestring containing all of the linestrings in a group.	X	Vendor extension
ST_MultiPointAggr method	"ST_MultiPoint type"	Returns a multipoint containing all of the points in a group.	X	Vendor extension
ST_MultiPolygonAggr method	"ST_MultiPolygon type"	Returns a multipolygon containing all of the polygons in a group.	X	Vendor extension
ST_MultiSurfaceAggr method	"ST_MultiSurface type"	Returns a multisurface containing all of the surfaces in a group.	X	Vendor extension
ST_ParseWKT method	"ST_SpatialRefSys type"	Retrieves a named element from the Well Known Text (WKT) definition of a spatial reference system.	X	Vendor extension
ST_SRIDFromBaseType method	"ST_Geometry type"	Parses a string defining the type string.	X	Vendor extension
ST_TransformGeom method	"ST_SpatialRefSys type"	Returns the geometry transformed using the given transform definition.	X	Vendor extension

Method	Type	Description	Round-Earth	SQL/MM
ST_UnionAggr method	“ST_Geometry type”	Returns the spatial union of all of the geometries in a group	X	Vendor extension
ST_World method	“ST_SpatialRefSys type”	Returns a geometry that represents all of the points in the spatial reference system.		Vendor extension

List of aggregate methods

The following is a list of aggregate methods available for use with spatial data. An X in the Round-Earth column indicates that the method is also supported on round-Earth spatial reference systems. The SQL/MM column reflects compliance with the SQL/MM standard (ISO/IEC 13249-3: 2006).

Method	Type	Description	Round-Earth	SQL/MM
ST_AsSVGAggr method	“ST_Geometry type”	Returns a complete or partial SVG document which renders the geometries in a group.	X	Vendor extension
ST_ConvexHullAggr method	“ST_Geometry type”	Returns the convex hull for all of the geometries in a group		Vendor extension
ST_EnvelopeAggr method	“ST_Geometry type”	Returns the bounding rectangle for all of the geometries in a group		Vendor extension
ST_GeomCollectionAggr method	“ST_GeomCollection type”	Returns a geometry collection containing all of the geometries in a group.	X	Vendor extension
ST_IntersectionAggr method	“ST_Geometry type”	Returns the spatial intersection of all of the geometries in a group	X	Vendor extension
ST_LineStringAggr method	“ST_LineString type”	Returns a linestring built from the ordered points in a group.	X	Vendor extension
ST_MultiCurveAggr method	“ST_MultiCurve type”	Returns a multicurve containing all of the curves in a group.	X	Vendor extension
ST_MultiLineStringAggr method	“ST_MultiLineString type”	Returns a multilinestring containing all of the linestrings in a group.	X	Vendor extension

Method	Type	Description	Round-Earth	SQL/MM
ST_MultiPointAggr method	"ST_MultiPoint type"	Returns a multipoint containing all of the points in a group.	X	Vendor extension
ST_MultiPolygonAggr method	"ST_MultiPolygon type"	Returns a multipolygon containing all of the polygons in a group.	X	Vendor extension
ST_MultiSurfaceAggr method	"ST_MultiSurface type"	Returns a multisurface containing all of the surfaces in a group.	X	Vendor extension
ST_UnionAggr method	"ST_Geometry type"	Returns the spatial union of all of the geometries in a group	X	Vendor extension

List of set operation methods

The following is a list of set operation methods available for use with spatial data. An X in the Round-Earth column indicates that the method is also supported on round-Earth spatial reference systems. The SQL/MM column reflects compliance with the SQL/MM standard (ISO/IEC 13249-3: 2006).

Method	Type	Description	Round-Earth	SQL/MM
ST_Difference method	"ST_Geometry type"	Returns the geometry value that represents the point set difference of two geometries.	X	5.1.20
ST_Intersection method	"ST_Geometry type"	Returns the geometry value that represents the point set intersection of two geometries.	X	5.1.18
ST_IntersectionAggr method	"ST_Geometry type"	Returns the spatial intersection of all of the geometries in a group	X	Vendor extension
ST_SymDifference method	"ST_Geometry type"	Returns the geometry value that represents the point set symmetric difference of two geometries.	X	5.1.21
ST_Union method	"ST_Geometry type"	Returns the geometry value that represents the point set union of two geometries.	X	5.1.19
ST_UnionAggr method	"ST_Geometry method"	Returns the spatial union of all of the geometries in a group	X	Vendor extension

List of spatial predicates

The following is a list of predicate methods available for use with spatial data. An X in the Round-Earth column indicates that the method is also supported on round-Earth spatial reference systems. The SQL/MM column reflects compliance with the SQL/MM standard (ISO/IEC 13249-3: 2006).

Method	Type	Description	Round-Earth	SQL/MM
ST_Contains method	"ST_Geometry type"	Tests if a geometry value spatially contains another geometry value.		5.1.31
ST_ContainsFilter method	"ST_Geometry type"	An inexpensive test if a geometry might contain another.		Vendor extension
ST_CoveredBy method	"ST_Geometry type"	Tests if a geometry value is spatially covered by another geometry value.	X	Vendor extension
ST_CoveredByFilter method	"ST_Geometry type"	An inexpensive test if a geometry might be covered by another.	X	Vendor extension
ST_Covers method	"ST_Geometry type"	Tests if a geometry value spatially covers another geometry value.	X	Vendor extension
ST_CoversFilter method	"ST_Geometry type"	An inexpensive test if a geometry might cover another.	X	Vendor extension
ST_Crosses method	"ST_Geometry type"	Tests if a geometry value crosses another geometry value.		5.1.29
ST_Disjoint method	"ST_Geometry type"	Test if a geometry value is spatially disjoint from another value.	X	5.1.26
ST_Equals method	"ST_Geometry type"	Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.	X	5.1.24
ST_EqualsFilter method	"ST_Geometry type"	An inexpensive test if a geometry is equal to another.	X	Vendor extension
ST_Intersects method	"ST_Geometry type"	Test if a geometry value spatially intersects another value.	X	5.1.27
ST_IntersectsFilter method	"ST_Geometry type"	An inexpensive test if the two geometries might intersect.	X	Vendor extension
ST_IntersectsRect method	"ST_Geometry type"	Test if a geometry intersects a rectangle.	X	Vendor extension

Method	Type	Description	Round-Earth	SQL/MM
ST_OrderingEquals method	"ST_Geometry type"	Tests if a geometry is identical to another geometry.	X	5.1.43
ST_Overlaps method	"ST_Geometry type"	Tests if a geometry value overlaps another geometry value.		5.1.32
ST_Relate method	"ST_Geometry type"	Tests if a geometry value is spatially related to another geometry value as specified by the intersection matrix. The ST_Relate method uses a 9-character string from the Dimensionally Extended 9 Intersection Model (DE-9IM) to describe the pair-wise relationship between two spatial data items. For example, the ST_Relate method determines if an intersection occurs between the geometries, and the geometry of the resulting intersection, if it exists. See also: "How spatial relationships work" on page 51.		5.1.25, Vendor extension
ST_Touches method	"ST_Geometry type"	Tests if a geometry value spatially touches another geometry value.		5.1.28
ST_Within method	"ST_Geometry type"	Tests if a geometry value is spatially contained within another geometry value.		5.1.30
ST_WithinDistance method	"ST_Geometry type"	Test if two geometries are within a specified distance of each other.	X	Vendor extension
ST_WithinDistanceFilter method	"ST_Geometry type"	An inexpensive of whether two geometries might be within a specified distance of each other.	X	Vendor extension
ST_WithinFilter method	"ST_Geometry type"	An inexpensive test if a geometry might be within another.		Vendor extension

Index