IBM Spatial Support for Db2 11 for z/OS
Version

User's Guide and Reference



GC19-4069

Notes

Before using this information and the product it supports, be sure to read the general information under "Notices" at the end of this information.

Subsequent editions of this PDF will not be delivered in IBM Publications Center. Always download the latest edition from .

2020-06-17 edition

This edition applies to IBM Spatial Support for DB2 11 for z/OS, part of Version 3.1 of DB2 Accessories Suite for z/OS, product number 5697-Q04, and to any subsequent releases to IBM Spatial Support for DB2 for z/OS until otherwise indicated in new editions. Make sure that you are using the correct edition for the level of the IBM Spatial Support for DB2 for z/OS offering.

© Copyright International Business Machines Corporation 2007, 2013.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this information..... ix
Who should read this information..... ix
Db2 Utilities Suite for z/OS..... ix
Terminology and citations..... ix
Accessibility features for Db2 11 for z/OS..... x
How to send your comments..... x
How to read syntax diagrams..... xi

Chapter 1. IBM Spatial Support for Db2 for z/OS..... 1
The purpose of IBM Spatial Support for Db2 for z/OS..... 1
How data represents geographic features..... 2
The nature of spatial data..... 3
Where spatial data comes from..... 3
How features, spatial information, spatial data, and geometries fit together..... 4

Chapter 2. About geometries..... 5
Geometries..... 5
Properties of geometries..... 6
Types..... 6
Geometry coordinates..... 6
X and Y coordinates..... 6
Z coordinates..... 6
M coordinates..... 6
Interior, boundary, and exterior..... 7
Simple or non-simple..... 7
Closed..... 7
Empty or not empty..... 7
Minimum bounding rectangle (MBR)..... 7
Dimension..... 7
Spatial reference system identifier..... 8

Chapter 3. Getting started with IBM Spatial Support for Db2 for z/OS..... 9
System requirements for installing IBM Spatial Support for Db2 for z/OS..... 9
Setting up and installing spatial support..... 9
Verifying the installation of spatial support..... 10
Inventory of resources supplied for your database..... 10
Enabling spatial support for the first time..... 11
Enabling spatial support for migration to Db2 11..... 12

Chapter 4. Setting up spatial resources..... 13
How to use coordinate systems..... 13
Coordinate systems..... 13
Geographic coordinate system..... 13
Projected coordinate systems..... 17
Selecting or creating coordinate systems..... 18
How to set up spatial reference systems..... 18
Spatial reference systems..... 18
Deciding whether to use a default spatial reference system or create a new system..... 19
Spatial reference systems supplied with IBM Spatial Support for Db2 for z/OS..... 20
Creating a spatial reference system..... 22

Conversion factors that transform coordinate data into integers....23
Calculating offset values....24
Calculating scale factors....25
Determining minimum and maximum coordinates and measures....25

Chapter 5. Setting up spatial columns....27
Spatial columns....27
Spatial columns with viewable content....27
Spatial data types....27
Creating spatial columns....28
Creating inline spatial columns....29
Registering spatial columns....30

Chapter 6. Populating spatial columns....33
About importing spatial data....33
Importing spatial data....33
Importing shape data to a new or existing table....33

Chapter 7. Using indexes to access spatial data....35
Spatial indexes....35
Spatial grid indexes....35
Generation of spatial grid indexes....35
Use of spatial functions in a query....36
How a query uses a spatial grid index....36
Considerations for the number of grid levels and grid sizes....37
Number of grid levels....37
Grid cell sizes....37
Creating spatial grid indexes....39

Chapter 8. Analyzing and generating spatial information....41
Environments for performing spatial analysis....41
Examples of how spatial functions operate....41
Functions that use indexes to optimize queries....42

Chapter 9. Stored procedures....45
ST_alter_coordsys....45
ST_alter_srs....47
ST_create_coordsys....50
ST_create_index....52
ST_create_srs....54
ST_create_srs_2....57
ST_drop_coordsys....61
ST_drop_index....62
ST_drop_srs....63
ST_export_shape....64
ST_import_shape....67
ST_register_spatial_column....73
ST_unregister_spatial_column....75

Chapter 10. Catalog views....77
The DB2GSE.GEOMETRY_COLUMNS catalog view....77
The DB2GSE.SPATIAL_REF_SYS catalog view....77
The DB2GSE.ST_COORDINATE_SYSTEMS catalog view....78
The DB2GSE.ST_GEOMETRY_COLUMNS catalog view....79
The DB2GSE.ST_SIZINGS catalog view....80
The DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS catalog view....81

The DB2GSE.ST_UNITS_OF_MEASURE catalog view ... 84

Chapter 11. Spatial functions: categories and uses ... 85

Constructor functions ... 85
Functions that operate on data exchange formats ... 85
A function that creates geometries from coordinates ... 86
Examples ... 87
Conversion to well-known text (WKT) representation ... 88
Conversion to well-known binary (WKB) representation ... 89
Conversion to ESRI shape representation ... 89
Conversion to Geography Markup Language (GML) representation ... 89
Comparison functions ... 90
Spatial comparison functions ... 91
Functions that compare geographic features ... 91
Functions that check whether one geometry contains another ... 92
ST_Contains ... 92
ST_Within ... 93
Functions that check intersections between geometries ... 95
EnvelopesIntersect ... 95
ST_Intersects ... 95
ST_Crosses ... 96
ST_Overlaps ... 97
ST_Touches ... 98
Function that checks whether two geometries are identical ... 100
ST_Equals ... 100
Functions that return coordinate and measure information ... 100
ST_Is3D ... 101
ST_IsMeasured ... 101
ST_IsValid ... 101
ST_M ... 102
ST_MaxM ... 102
ST_MaxX ... 102
ST_MaxY ... 102
ST_MaxZ ... 102
ST_MinM ... 102
ST_MinX ... 102
ST_MinY ... 102
ST_MinZ ... 102
ST_X ... 102
ST_Y ... 102
ST_Z ... 102
Functions that return information about geometries within a geometry ... 102
ST_Centroid ... 102
ST_EndPoint ... 103
ST_GeometryN ... 103
ST_NumGeometries ... 103
ST_NumPoints ... 103
ST_PointN ... 103
ST_StartPoint ... 103
Functions that show information about boundaries, envelopes, and rings ... 103
Functions that return information about a geometry's dimensions ... 104
ST_Area ... 104
ST_Length ... 104
Functions that reveal whether a geometry is closed, empty, or simple ... 104
ST_IsClosed ... 104
ST_IsEmpty ... 104
ST_IsSimple ... 104

Function that identifies a geometry's spatial reference system..... 105
ST_SRID.....105
Functions that generate new geometries from existing geometries..... 105
Function that converts one geometry to another.....105
Functions that create new geometries with different space configurations.....105
Function that derives one geometry from many..... 108
Function that returns distance information..... 108
Function that returns index information..... 109

Chapter 12. Spatial functions: syntax and parameters..... 111
Considerations for spatial functions..... 111
EnvelopesIntersect..... 111
ST_Area.....113
ST_AsBinary.....115
ST_AsGML.....116
ST_AsShape.....117
ST_AsText.....118
ST_Boundary..... 119
ST_Buffer..... 121
ST_Centroid..... 123
ST_Contains..... 124
ST_ConvexHull.....125
ST_CoordDim..... 127
ST_Crosses.....128
ST_Difference..... 129
ST_Dimension.....131
ST_Disjoint.....132
ST_Distance.....133
ST_Endpoint..... 136
ST_Envelope..... 137
ST_Equals..... 138
ST_ExteriorRing.....140
ST_Geometry..... 141
ST_GeometryN.....142
ST_GeometryType..... 143
ST_GeomFromText.....143
ST_GeomFromWKB..... 144
ST_GetIndexParms.....146
ST_InteriorRingN.....146
ST_Intersection.....147
ST_Intersects..... 149
ST_Is3D..... 150
ST_IsClosed.....151
ST_IsEmpty..... 153
ST_IsMeasured.....154
ST_IsRing.....155
ST_IsSimple.....156
ST_IsValid.....157
ST_Length.....158
ST_LineFromWKB.....159
ST_LineString.....160
ST_LocateAlong.....161
ST_LocateBetween.....163
ST_M..... 164
ST_MaxM.....165
ST_MaxX..... 166
ST_MaxY..... 167

ST_MaxZ ... 169
ST_MinM ... 170
ST_MinX ... 171
ST_MinY ... 172
ST_MinZ ... 173
ST_MLineFromWKB ... 174
ST_MPointFromWKB ... 176
ST_MPolyFromWKB ... 177
ST_MultiLineString ... 178
ST_MultiPoint ... 180
ST_MultiPolygon ... 181
ST_NumGeometries ... 183
ST_NumInteriorRing ... 184
ST_NumPoints ... 185
ST_Overlaps ... 186
ST_Perimeter ... 188
ST_Point ... 189
ST_PointFromWKB ... 191
ST_PointN ... 192
ST_PointOnSurface ... 193
ST_PolyFromWKB ... 194
ST_Polygon ... 195
ST_Relate ... 197
ST_SRID ... 198
ST_StartPoint ... 198
ST_SymDifference ... 199
ST_Touches ... 201
ST_Union ... 202
ST_UnionAggr ... 204
ST_Within ... 206
ST_WKBToSQL ... 207
ST_WKTToSQL ... 208
ST_X ... 209
ST_Y ... 210
ST_Z ... 210

Chapter 13. Supported data formats ... 213
Well-known text (WKT) representation ... 213
Well-known binary (WKB) representation ... 217
Shape representation ... 219
Geography Markup Language (GML) representation ... 219

Chapter 14. Supported coordinate systems ... 221
Coordinate systems syntax ... 221
Supported linear units ... 224
Supported angular units ... 224
Supported spheroids ... 225
Supported prime meridians ... 227
Supported map projections ... 227

Chapter 15. The DSN5SCLP program ... 231
Commands for the DSN5SCLP program ... 231
alter_cs ... 232
alter_srs ... 233
create_cs ... 235
create_idx ... 237
create_srs ... 239

create_srs_2 ... 241
disable_spatial ... 243
drop_cs ... 244
drop_idx ... 245
drop_srs ... 245
enable_spatial ... 246
function_level ... 247
import_shape ... 248
register_spatial_column ... 253
unregister_spatial_column ... 254

Chapter 16. Identifying IBM Spatial Support for Db2 for z/OS problems ... 257
How to interpret spatial support messages ... 257
Output parameters for spatial support stored procedures ... 258
Messages for spatial support stored procedures ... 259
Spatial support function messages ... 259

Chapter 17. GSE Messages ... 261

Information resources for Db2 11 for z/OS and related products ... 299

Notices ... 301
Trademarks ... 302
Privacy policy considerations ... 302

Glossary ... 303

Index ... 305

About this information

This documentation provides usage and reference information about IBM® Spatial Support for Db2® for z/OS®.

Who should read this information

This information is primarily intended for information technology professionals who manage enterprise mainframe systems and need to understand spatial data, spatial information administration, and spatial analysis using Db2 for z/OS.

This information is also a valuable reference guide for line-of-business people and mid-level managers who are looking to validate business plans to leverage their IT infrastructure and move distributed spatial information and analysis to centralized systems.

Db2 Utilities Suite for z/OS

Important: In Db2 11, the Db2 Utilities Suite for z/OS is available as an optional product. You must separately order and purchase a license to such utilities, and discussion of those utility functions in this publication is not intended to otherwise imply that you have a license to them.

Db2 11 utilities can use the DFSORT program regardless of whether you purchased a license for DFSORT on your system. For more information, see the following informational APARs:

- II14047
- II14213
- II13495

Db2 utilities can use IBM Db2 Sort for z/OS (5655-W42) as an alternative to DFSORT for utility SORT and MERGE functions. Use of Db2 Sort for z/OS requires the purchase of a Db2 Sort for z/OS license. For more information about Db2 Sort for z/OS, see Db2 Sort for z/OS.

Related information

Terminology and citations

When referring to a Db2 product other than Db2 for z/OS, this information uses the product's full name to avoid ambiguity.

About the Db2 brand change: IBM has re-branded DB2® to Db2, and Db2 for z/OS is now the name of the offering previously know as "DB2 for z/OS". During and after implementation of this change, you might sometimes still see references to the original names, such as "DB2 for z/OS" and "DB2", in different IBM web pages and documents. However, assume that these refer to the same "Db2 for z/OS" product when the PID, Entitlement Entity, version, modification, and release information match. For more information, see Revised naming for IBM Db2 family products on IBM z/OS platform.

The following terms are used as indicated:

Db2
Represents either the Db2 licensed program or a particular Db2 subsystem.

Tivoli® OMEGAMON® XE for Db2 Performance Expert on z/OS
Refers to any of the following products:

- IBM Tivoli OMEGAMON XE for Db2 Performance Expert on z/OS
- IBM Db2 Performance Monitor on z/OS
- IBM Db2 Performance Expert for Multiplatforms and Workgroups

© Copyright IBM Corp. 2007, 2013

- IBM Db2 Buffer Pool Analyzer for z/OS

C, C++, and C language
Represent the C or C++ programming language.

CICS®
Represents CICS Transaction Server for z/OS.

IMS
Represents the IMS Database Manager or IMS Transaction Manager.

MVS™
Represents the MVS element of the z/OS operating system, which is equivalent to the Base Control Program (BCP) component of the z/OS operating system.

RACF®
Represents the functions that are provided by the RACF component of the z/OS Security Server.

Accessibility features for Db2 11 for z/OS

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility features

The following list includes the major accessibility features in z/OS products, including Db2 11 for z/OS. These features support:

- Keyboard-only operation.
- Interfaces that are commonly used by screen readers and screen magnifiers.
- Customization of display attributes such as color, contrast, and font size

Tip: The IBM Knowledge Center (which includes information for Db2 for z/OS) and its related publications are accessibility-enabled for the IBM Home Page Reader. You can operate all features using the keyboard instead of the mouse.

Keyboard navigation

For information about navigating the Db2 for z/OS ISPF panels using TSO/E or ISPF, refer to the *z/OS TSO/E Primer*, the *z/OS TSO/E User's Guide*, and the *z/OS ISPF User's Guide*. These guides describe how to navigate each interface, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Related accessibility information

IBM and accessibility

See the *IBM Accessibility Center* at http://www.ibm.com/able for more information about the commitment that IBM has to accessibility.

How to send your comments

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other Db2 for z/OS documentation.

Send your comments by email to db2zinfo@us.ibm.com and include the name of the product, the version number of the product, and the number of the book. If you are commenting on specific text, please list the location of the text (for example, a chapter and section title or a help topic title).

How to read syntax diagrams

Certain conventions apply to the syntax diagrams that are used in IBM documentation.

Apply the following rules when reading the syntax diagrams that are used in Db2 for z/OS documentation:

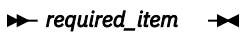
- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The ►►—— symbol indicates the beginning of a statement.

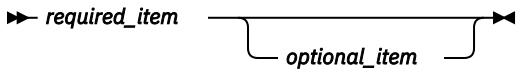
  The ——► symbol indicates that the statement syntax is continued on the next line.

  The ►—— symbol indicates that a statement is continued from the previous line.

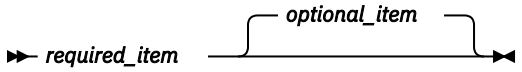
  The ——►◄ symbol indicates the end of a statement.
- Required items appear on the horizontal line (the main path).



- Optional items appear below the main path.

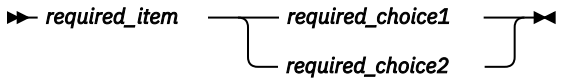


  If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

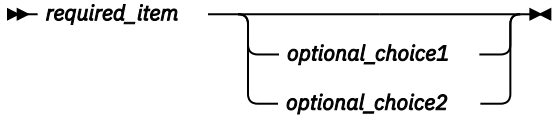


- If you can choose from two or more items, they appear vertically, in a stack.

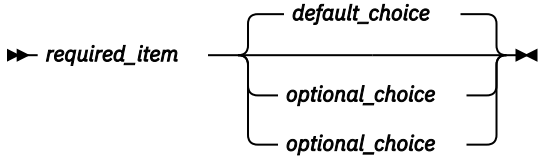
  If you *must* choose one of the items, one item of the stack appears on the main path.



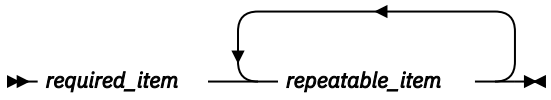
  If choosing one of the items is optional, the entire stack appears below the main path.



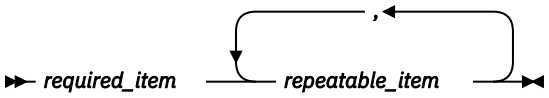
  If one of the items is the default, it appears above the main path and the remaining choices are shown below.



- An arrow returning to the left, above the main line, indicates an item that can be repeated.



  If the repeat arrow contains a comma, you must separate repeated items with a comma.

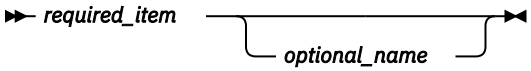


A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Sometimes a diagram must be split into fragments. The syntax fragment is shown separately from the main syntax diagram, but the contents of the fragment should be read as if they are on the main path of the diagram.



fragment-name



- With the exception of XPath keywords, keywords appear in uppercase (for example, FROM). Keywords must be spelled exactly as shown. XPath keywords are defined as lowercase names, and must be spelled exactly as shown. Variables appear in all lowercase letters (for example, *column-name*). They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Related information

Chapter 1. IBM Spatial Support for Db2 for z/OS

For many years, spatial support offerings from IBM have provided the basis for building robust spatial applications using international and industry standard interfaces. For example, you might have experience using Db2 Spatial Extender for Linux®, UNIX, and Windows. Now, IBM Spatial Support for Db2 for z/OS provides Db2 for z/OS customers with the ability to enable a Db2 for z/OS subsystem for spatial support.

IBM Spatial Support for Db2 for z/OS provides a set of spatial data types that you can use to model real-world entities, such as the locations of customers, the boundaries of parks, and the path of cable lines. You can manipulate spatial data by using spatial functions, which you can invoke from within an SQL statement. Also, you can create indexes on spatial data, which can be used by Db2 to optimize spatial query performance.

The following information introduces IBM Spatial Support for Db2 for z/OS in more detail by explaining its purpose, describing the data that it supports, and explaining how its underlying concepts fit together.

The purpose of IBM Spatial Support for Db2 for z/OS

You can use IBM Spatial Support for Db2 for z/OS to generate and analyze spatial information about geographic features, and to store and manage the data on which this information is based.

A geographic feature (sometimes called feature in this discussion, for short) is anything in the real world that has an identifiable location, or anything that could be imagined as existing at an identifiable location. A feature can be:

- An object (that is, a concrete entity of any sort); for example, a river, forest, or range of mountains.
- A space; for example, a safety zone around a hazardous site, or the marketing area serviced by a particular business.
- An event that occurs at a definable location; for example, an auto accident that occurred at a particular intersection, or a sales transaction at a specific store.

Features exist in multiple environments. For example, the objects mentioned in the preceding list—river, forest, mountain range—belong to the natural environment. Other objects, such as cities, buildings, and offices, belong to the cultural environment. Still others, such as parks, zoos, and farmland, represent a combination of the natural and cultural environments.

In this discussion, the term spatial information refers to the kind of information that IBM Spatial Support for Db2 for z/OS makes available to its users—namely, facts and figures about the locations of geographic features. Examples of spatial information are:

- Locations of geographic features on the map (for example, longitude and latitude values that define where cities are situated)
- The location of geographic features with respect to one another (for example, points within a city where hospitals and clinics are located, or the proximity of the city's residences to local earthquake zones)
- Ways in which geographic features are related to each other (for example, information that a certain river system is enclosed within a specific region, or that certain bridges in that region cross over the river system's tributaries)
- Measurements that apply to one or more geographic features (for example, the distance between an office building and its lot line, or the length of a bird preserve's perimeter)

Spatial information, either by itself or in combination with traditional relational data, can help you with such activities as defining the areas in which you provide services, and determining locations of possible markets. For example, suppose that the manager of a county welfare district needs to verify which welfare applicants and recipients actually live within the area that the district services. IBM Spatial

© Copyright IBM Corp. 2007, 2013

Support for Db2 for z/OS can derive this information from the serviced area's location and from the addresses of the applicants and recipients.

Or suppose that the owner of a restaurant chain wants to do business in nearby cities. To determine where to open new restaurants, the owner needs answers to such questions as: Where in these cities are concentrations of clientele who typically frequent my restaurants? Where are the major highways? Where is the crime rate lowest? Where are the competition's restaurants located? IBM Spatial Support for Db2 for z/OS can produce information to answer these questions. Furthermore, front-end tools, though not required, can play a part. For example, a visualization tool can put information produced by IBM Spatial Support for Db2 for z/OS, such as the location of concentrations of clientele and the proximity of major highways to proposed restaurants, in graphic form on a map. Business intelligence tools can put associated information, like names and descriptions of competing restaurants, in report form.

How data represents geographic features

A geographic feature can be represented by one or more data items; for example, the data items in a row of a table.

A *data item* is the value or values that occupy a cell of a relational table. For example, consider office buildings and residences. In the following figure, each row of the BRANCHES table represents a branch office of a bank. Similarly, each row of the CUSTOMERS table, taken as a whole, represents a customer of the bank. However, a subset of each row—specifically, the data items that constitute a customer's address—represent the customer's residence.

BRANCHES

ID	NAME	ADDRESS	CITY	POSTAL CODE	STATE_PROV	COUNTRY
937	Airzone-Multern	92467 Airzone Blvd	San Jose	95141	CA	USA

CUSTOMERS

ID	LAST NAME	FIRST NAME	ADDRESS	CITY	POSTAL CODE	STATE_PROV	COUNTRY	CHECKING	SAVINGS
59-6396	Kriner	Endela	9 Concourt Circle	San Jose	95141	CA	USA	A	A

Figure 1. Data that represents geographic features

The tables in this figure contain data that identifies and describes the bank's branches and customers. This discussion refers to such data as business data.

A subset of the business data—the values that denote the branches' and customers' addresses—can be translated into values from which spatial information is generated. For example, as shown in Figure 1 on page 2, one branch office's address is 92467 Airzone Blvd., San Jose, CA 95141, USA. A customer's address is 9 Concourt Circle, San Jose, CA 95141, USA. IBM Spatial Support for Db2 for z/OS can construct a ST_POINT column object by using the geocoded x and y coordinate values. The next figure shows the BRANCHES and CUSTOMERS tables with new columns that are designated to contain such values.

BRANCHES

ID	NAME	ADDRESS	CITY	POSTAL CODE	STATE_PROV	COUNTRY	LOCATION
937	Airzone-Multern	92467 Airzone Blvd	San Jose	95141	CA	USA	

CUSTOMERS

ID	LAST NAME	FIRST NAME	ADDRESS	CITY	POSTAL CODE	STATE_PROV	COU...	LOCATION	CH...	SA...
59-6396	Kriner	Endela	9 Concourt Circle	San Jose	95141	CA	USA		A	A

Figure 2. Tables with spatial columns added

Because spatial information will be derived from the data items stored in the LOCATION column, these data items are referred to in this discussion as spatial data.

The nature of spatial data

Spatial data is made up of coordinates that identify a location. Spatial support works with two-dimensional coordinates specified by x (longitude) and y (latitude) values.

A coordinate is a number that denotes either:

- A position along an axis relative to an origin, given a unit of length.
- A direction relative to a base line or plane, given a unit of angular measure.

For example, latitude is a coordinate that denotes an angle relative to the equatorial plane, usually in degrees. Longitude is a coordinate that denotes an angle relative to the Greenwich meridian, also usually in degrees. Thus, on a map, the position of Yellowstone National Park is defined by latitude 44.45 degrees north of the equator and longitude 110.40 degrees west of the Greenwich meridian. More precisely, these coordinates reference the center of Yellowstone National Park in the USA.

The definitions of latitude and longitude, their points, lines, and planes of reference, units of measure, and other associated parameters are referred to collectively as a coordinate system. Coordinate systems can be based on values other than latitude and longitude. These coordinate systems have their own points, lines, and planes of reference, units of measure, and additional associated parameters (such as the projection transformation).

The simplest spatial data item consists of a single coordinate pair that defines the position of a single geographic location. A more extensive spatial data item consists of several coordinates that define a linear path that a road or river might form. A third kind consists of coordinates that define the boundary of an area; for example, the boundary of a land parcel or flood plain.

Each spatial data item is an instance of a spatial data type. The data type for coordinates that mark a single location is ST_Point; the data type for coordinates that define a linear path is ST_LineString; and the data type for coordinates that define the boundary of an area is ST_Polygon. These types, together with the other spatial data types, are structured types that belong to a single hierarchy.

Where spatial data comes from

You can obtain spatial data by generating it from spatial functions and importing it from external sources.

Using functions to generate spatial data

You can use spatial functions to generate spatial data from input data.

For example, suppose that the bank whose branches are defined in the BRANCHES table wants to know how many customers are located within five miles of each branch. Before the bank can obtain this information from the database, it needs to define the zone that lies within a specified radius around each branch. An IBM Spatial Support for Db2 for z/OS function, ST_Buffer, can create such a definition. Using the coordinates of each branch as input, ST_Buffer can generate the coordinates that demarcate the perimeters of the zones. The following figure shows the BRANCHES table with information that is supplied by ST_Buffer.

BRANCHES

ID	NAME	ADDRESS	CITY	POSTAL CODE	STATE_PROV	COUNTRY	LOCATION	SALES_AREA
937	Airzone-Multern	92467 Airzone Blvd	San Jose	95141	CA	USA	1653 3094	1002 2001, 1192 3564, 2502 3415, 1915 3394, 1002 2001

Figure 3. Table that includes new spatial data derived from existing spatial data

In addition to the ST_Buffer function, IBM Spatial Support for Db2 for z/OS provides several other functions that derive new spatial data from existing spatial data.

Importing spatial data
IBM Spatial Support for Db2 for z/OS provides services to import spatial data in shape file format.

Spatial data in shape file format is available from many sources through the internet.

You can import spatial data from shape files provided by external data sources. These files typically contain data that is applied to maps: street networks, flood plains, earthquake faults, and so on. By using such data in combination with spatial data that you produce, you can augment the spatial information available to you. For example, if a public works department needs to determine what hazards a residential community is vulnerable to, it could use the ST_Buffer function to define a zone around the community. The public works department could then import data on flood plains and earthquake faults to see which of these problem areas overlap this zone.

How features, spatial information, spatial data, and geometries fit together

This section summarizes several basic concepts that underlie the operations of IBM Spatial Support for Db2 for z/OS: geographic features, spatial information, spatial data, and geometries.

IBM Spatial Support for Db2 for z/OS lets you obtain facts and figures that pertain to things that can be defined geographically—that is, in terms of their location on earth, or within a region of the earth. The Db2 documentation refers to such facts and figures as *spatial information*, and to the things as *geographic features* (called *features* here, for short).

For example, you could use IBM Spatial Support for Db2 for z/OS to determine whether any populated areas overlap the proposed site for a landfill. The populated areas and the proposed site are features. A finding as to whether any overlap exists would be an example of spatial information. If overlap is found to exist, the extent of it would also be an example of spatial information.

To produce spatial information, IBM Spatial Support for Db2 for z/OS must process data that defines the locations of features. Such data, called *spatial data*, consists of coordinates that reference the locations on a map or similar projection. For example, to determine whether one feature overlaps another, IBM Spatial Support for Db2 for z/OS must determine where the coordinates of one of the features are situated with respect to the coordinates of the other.

In the world of spatial information technology, it is common to think of features as being represented by symbols called *geometries*. Geometries are partly visual and partly mathematical. Consider their visual aspect. The symbol for a feature that has width and breadth, such as a park or town, is a multisided figure. Such a geometry is called a *polygon*. The symbol for a linear feature, such as a river or a road, is a line. Such a geometry is called a *linestring*.

A geometry has properties that correspond to facts about the feature that it represents. Most of these properties can be expressed mathematically. For example, the coordinates for a feature collectively constitute one of the properties of the feature's corresponding geometry. Another property, called *dimension*, is a numerical value that indicates whether a feature has length or breadth.

Spatial data and certain spatial information can be viewed in terms of geometries. Consider the example, described earlier, of the populated areas and the proposed landfill site. The spatial data for the populated areas includes coordinates stored in a column of a table in a Db2 database. The convention is to regard what is stored not simply as data, but as actual geometries. Because populated areas have width and breadth, you can see that these geometries are polygons.

Like spatial data, certain spatial information is also viewed in terms of geometries. For example, to determine whether a populated area overlaps a proposed landfill site, IBM Spatial Support for Db2 for z/OS must compare the coordinates in the polygon that symbolizes the site with the coordinates of the polygons that represent populated areas. The resulting information—that is, the areas of overlap—are themselves regarded as polygons: geometries with coordinates, dimensions, and other properties.

Chapter 2. About geometries

Entities of information, called geometries, consist of coordinates and represent geographic features.

Geometries

This topic provides an overview of the geometries that are supported by IBM Spatial Support for Db2 for z/OS.

Webster's Revised Unabridged Dictionary defines *geometry* as "That branch of mathematics which investigates the relations, properties, and measurement of solids, surfaces, lines, and angles; the science which treats of the properties and relations of magnitudes; the science of the relations of space." The word geometry has also been used to denote the geometric features that, for the past millennium or more, cartographers have used to map the world. An abstract definition of this new meaning of geometry is "a point or aggregate of points representing a feature on the ground."

In IBM Spatial Support for Db2 for z/OS, the operational definition of geometry is "a model of a geographic feature." The model can be expressed in terms of the feature's coordinates. The model conveys information; for example, the coordinates identify the position of the feature with respect to fixed points of reference. Also, the model can be used to produce information; for example, the ST_Overlaps function can take the coordinates of two proximate regions as input and return information as to whether the regions overlap or not.

The coordinates of a feature that a geometry represents are regarded as properties of the geometry. Several kinds of geometries have other properties as well; for example, area, length, and boundary.

IBM Spatial Support for Db2 for z/OS supports seven distinct geometry types. Six of these geometry types are instantiable, and one is non-instantiable.

The instantiable geometries include:

Point
A single point. Points represent discrete features that are perceived as occupying the locus where an east-west coordinate line (such as a parallel) intersects a north-south coordinate line (such as a meridian). For example, suppose that the notation on a world map shows that each city on the map is located at the intersection of a parallel and a meridian. A point could represent each city.

Linestring
A line between two or more points. It does not have to be a straight line. Linestrings represent linear geographic features (for example, streets, canals, and pipelines).

Polygon
A polygon or surface within a polygon. Polygons represent multisided geographic features (for example, welfare districts, forests, and wildlife habitats).

Multipoint
A multiple point geometry type. Multipoints represent multipart features whose components are each located at the intersection of an east-west coordinate line and a north-south coordinate line (for example, an island chain whose members are each situated at an intersection of a parallel and meridian).

Multilinestring
A multiple curve geometry type with multiple strings. Multilinestrings represent multipart features that are made up (for example, river systems).

Multipolygon
A multiple surface geometry type with multiple polygons. Multipolygons represent multipart features made up of multisided units or components (for example, the collective farmlands in a specific region, or a system of lakes).

The non-instantiable geometry type is:

© Copyright IBM Corp. 2007, 2013

Geometry
Geometry is an abstract data type that you can use for parameter passing in spatial functions. Geometry does not have a corresponding constructor function; therefore, it is not supported as an instantiable spatial data type.

Properties of geometries

This information describes the properties of geometries.

The properties of geometries are:

- The type that a geometry belongs to
- Geometry coordinates
- A geometry's interior, boundary, and exterior
- The quality of being simple or non-simple
- The quality of being empty or not empty
- A geometry's minimum bounding rectangle or envelope
- Dimension
- The identifier of the spatial reference system with which a geometry is associated

Types

IBM Spatial Support for Db2 for z/OS supports seven distinct geometry types.

Six of these geometry types are instantiable, and one is abstract. The six instantiable geometry types are ST_Point, ST_Linestring, ST_Polygon, ST_Multipoint, ST_Multilinestring, and ST_Multipolygon. The abstract type is ST_Geometry.

Geometry coordinates

All geometries include at least one X coordinate and one Y coordinate, unless they are empty geometries, in which case they contain no coordinates at all.

In addition, a geometry can include one or more Z coordinates and M coordinates. X, Y, Z, and M coordinates are represented as double–precision numbers. The following subsections explain:

- X and Y coordinates
- Z coordinates
- M coordinates

X and Y coordinates

An *X coordinate value*, or longitude, denotes a location that is relative to a point of reference to the east or west. A *Y coordinate value*, or latitude, denotes a location that is relative to a point of reference to the north or south.

Z coordinates

Some geometries have an associated altitude or depth. Each of the points that form the geometry of a feature can include an optional Z coordinate that represents an altitude or depth normal to the earth's surface.

M coordinates

An M coordinate (measure) is a value that conveys information about a geographic feature and that is stored together with the coordinates that define the feature's location.

For example, suppose that you are representing highways in your application. If you want your application to process values that denote linear distances or mileposts, you can store these values along with the coordinates that define locations along the highway. M coordinates are represented as double–precision numbers.

Interior, boundary, and exterior

All geometries occupy a position in space defined by their interiors, boundaries, and exteriors.

The exterior of a geometry is all space not occupied by the geometry. The boundary of a geometry serves as the interface between its interior and exterior. The interior is the space occupied by the geometry.

Simple or non-simple

The values of some geometry subtypes (linestrings, multipoints, and multilinestrings) are either simple or non-simple.

A geometry is simple if it obeys all the topological rules imposed on its subtype and non-simple if it doesn't. A linestring is simple if it does not intersect its interior. A multipoint is simple if none of its elements occupy the same coordinate space. Points, polygons, multipolygons, and empty geometries are always simple.

Closed

A linestring is closed if its start and end points are the same.

A multilinestring is closed if all of its elements are closed. A ring is a simple, closed linestring.

Empty or not empty

A geometry is empty if it does not have any points.

The envelope, boundary, interior, and exterior of an empty geometry are not defined and will be represented as null. An empty geometry is always simple. Empty polygons and multipolygons have an area of 0.

Minimum bounding rectangle (MBR)

The minimum bounding rectangle (MBR) of a geometry is the bounding geometry formed by the minimum and maximum (X,Y) coordinates.

Except for the following special cases, the MBRs of geometries form a boundary rectangle:

- The MBR of any point is the point itself, because its minimum and maximum X coordinates are the same and its minimum and maximum Y coordinates are the same.
- The MBR of a horizontal or vertical linestring is a linestring represented by the boundary (the endpoints) of the source linestring.

Dimension

A geometry can have a dimension of –1, 0, 1, or 2.

The dimensions are listed as follows:

–1
Is empty

0
Has no length and an area of 0 (zero)

1
Has a length larger than 0 (zero) and an area of 0 (zero)

2
Has an area that is larger than 0 (zero)

The point and multipoint subtypes have a dimension of zero. Points represent dimensional features that can be modeled with a single tuple of coordinates, while multipoint subtypes represent data that must be modeled with a set of points.

The linestring and multilinestring subtypes have a dimension of one. They store road segments, branching river systems and any other features that are linear in nature.

Polygon and multipolygon subtypes have a dimension of two. Features whose perimeter encloses a definable area, such as forests, parcels of land, and lakes, can be represented by either the polygon or multipolygon data type.

Spatial reference system identifier

The numeric identifier for a spatial reference system determines which spatial reference system is used to represent the geometry.

All spatial reference systems known to the database can be accessed through the DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS catalog view.

Chapter 3. Getting started with IBM Spatial Support for Db2 for z/OS

This information provides instructions for installing and configuring IBM Spatial Support for Db2 for z/OS.

System requirements for installing IBM Spatial Support for Db2 for z/OS

Before you install IBM Spatial Support for Db2 for z/OS, ensure that your system meets all of the operating system and software requirements.

Operating system requirements

Your z/OS operating system must meet the requirements for z/OS Version 1 Release 13, and later.

Software requirements

To install IBM Spatial Support for Db2 for z/OS, you must have Db2 11 for z/OS installed and operating in new-function mode. Also, you must specify the name of the WLM environment to use for the stored procedures.

In addition, you must install and configure the Db2 ODBC feature for Db2 11 for z/OS.

To access spatial data from a remote client application, you must install the version of DB2 Connect that is required for Db2 11 for z/OS. For more information, see the Program Directory for the Db2 Accessories Suite.

Related information

Setting up and installing spatial support

A spatial support system consists of a Db2 database system, IBM Spatial Support for Db2 for z/OS, and, for most applications, a map viewer.

Before you begin

Before you set up IBM Spatial Support for Db2 for z/OS, you must have the following software installed:

- You must have Db2 11 for z/OS installed and operating in new-function mode.
- You must have the Db2 Open Database Connectivity (ODBC) feature for Db2 11 for z/OS installed and configured.
- To access spatial data from a remote client application, you must install the version of DB2 Connect that is required for Db2 11 for z/OS. For more information, see the Program Directory for the Db2 Accessories Suite.

About this task

A map viewer is not required but is useful for visually rendering the results of spatial queries, generally in the form of maps. An example of an easy-to-use map viewer is ArcMap, which is included as an application for ArcGIS and developed by ESRI. Open source map viewers are also available.

Databases enabled for spatial support are located on the server. You can use client applications to access spatial data through the IBM Spatial Support for Db2 for z/OS stored procedures and spatial queries.

Procedure

To install and set up spatial support:

© Copyright IBM Corp. 2007, 2013

1. Ensure that your system meets all of the system requirements.
2. Install IBM Spatial Support for Db2 for z/OS.

   Ensure that you have the Program Directory for the Db2 Accessories Suite and the IBM Spatial Support for Db2 for z/OS product tape. The program directory provides instructions for the SMP/E installation and includes the correct service levels for all of the required software.
3. Verify the installation by issuing two SQL statements.
4. If necessary, see the troubleshooting tips in Chapter 16, “Identifying IBM Spatial Support for Db2 for z/OS problems,” on page 257 and take appropriate actions to correct any problems.

   To access Db2 documentation, go to the Db2 collection in the IBM Knowledge Center at the following website: http://www-01.ibm.com/support/knowledgecenter/SSEPEK/db2z_prodhome.html. This collection contains documentation for the Db2 database system and related products.

Related information

Verifying the installation of spatial support

You can verify the installation of IBM Spatial Support for Db2 for z/OS by issuing two SQL SELECT statements.

Procedure

To verify the installation of spatial support:

1. Submit the following SELECT statement:

```
SELECT CAST(DB2GSE.ST_ASTEXT(DB2GSE.ST_POINT('POINT (1.0 1.0)', 1)) AS VARCHAR
(32)) AS POINT_DATA FROM SYSIBM.SYSDUMMY1;
```

   The expected output is:

```
POINT_DATA
--------------------------------
POINT (1.000000 1.000000)

1 record(s) selected.
```

2. Then, submit the following SELECT statement:

```
SELECT CAST(DB2GSE.ST_ASTEXT(DB2GSE.ST_POINT('POINT (Invalid SRS ID)', 99999))
AS VARCHAR(32)) AS POINT_DATA FROM SYSIBM.SYSDUMMY1;
```

   The expected output is:

```
POINT_DATA
--------------------------------
SQL0443N Routine "GSEGEOMFROMTEXT" (specific name "STCO00002GFT") has returned
an error SQLSTATE with diagnostic text "GSE3001N Invalid SRS identifier 99999.".
SQLSTATE=38SU1
```

Results

You verified the successful installation of IBM Spatial Support for Db2 for z/OS.

Inventory of resources supplied for your database

To enable a database for spatial support, IBM Spatial Support for Db2 for z/OS provides the database with the required resources.

IBM Spatial Support for Db2 for z/OS provides the following resources:

- Stored procedures. When you request a spatial operation (for example, when you import spatial data), IBM Spatial Support for Db2 for z/OS invokes one of these stored procedures to perform the operation.

- Spatial data types. You must assign a spatial data type to each table or view column that is to contain spatial data.
- Spatial catalog tables and catalog views.
- A spatial grid index, so that you can define grid indexes on spatial columns.
- Spatial functions. You use these to work with spatial data in a number of ways; for example, to determine relationships between geometries and to generate more spatial data.
- Definitions of coordinate systems.
- Default spatial reference systems.
- Two schemas: DB2GSE and SYSPROC.

Enabling spatial support for the first time

If you are a new customer and want to start using IBM Spatial Support for Db2 for z/OS, you need to enable your Db2 subsystem for spatial support.

Before you begin

Before you enable the Db2 subsystem for spatial support, your TSO user ID must have SYSADM authority on the Db2 subsystem.

Procedure

To enable a Db2 subsystem for spatial support:

1. Customize and submit the DSNTIJCL job from the SDSNSAMP library.

   This job binds the Db2 ODBC application plan. The prolog for the DSNTIJCL job contains instructions for modifying the job.
2. Customize and submit the DSN5SENB job from the SDSNSAMP library.

   This job creates the spatial catalog tables, catalog views, and stored procedures. In addition, this job binds all of the spatial packages, and it invokes the DSN5SCLP program, which uses the enable_spatial command to create the spatial user-defined functions and to populate the spatial catalog tables with initial values.

   Before you submit the DSN5SENB job, you also must customize the DSNAOINI initialization file in the SDSNSAMP library. The prolog for the DSN5SENB job contains instructions for modifying the job.
3. Customize and submit the DSN5SBND job from the SDSNSAMP library.

   The expected return code for this job is 0 (zero).

Results

The expected return codes are 0 (zero) and 4 for both the DSNTIJCL job and the DSN5SENB job.

What to do next

Now, the Db2 subsystem is enabled for spatial support. You can submit the DSN5SCMD JCL in the SDSNSAMP library to verify that the Spatial functions are correctly enabled.

Related concepts

"The DSN5SCLP program" on page 231

DSN5SCLP is an ODBC program that you can use to invoke IBM Spatial Support for Db2 for z/OS stored procedures for administrative tasks.

Related reference

"enable_spatial" on page 246

Use the enable_spatial command to supply a Db2 subsystem with the resources that it needs to store spatial data and support spatial operations.

Related information

Enabling spatial support for migration to Db2 11

If you are migrating from Db2 10 to Db2 11 and had spatial support enabled on Db2 10, you must enable your Db2 subsystem for spatial support again.

Before you begin

Before you enable the Db2 subsystem for spatial support, your TSO user ID must have SYSADM authority on the Db2 subsystem.

Procedure

To enable a Db2 subsystem for spatial support if you migrated from the previous version of Db2 for z/OS:

1. Customize and submit the DSNTIJCL job from the SDSNSAMP library.

   This job binds the Db2 ODBC application plan. The prolog for the DSNTIJCL job contains instructions for modifying the job.
2. Customize and submit the DSN5SBND job from the SDSNSAMP library.

   The expected return code for the DSN5SBND job is 0 (zero).

   Note: This step is not required if you applied APAR PM92224/UK98018 to your DB2 10 subsystem before migration and followed the ++HOLD instructions.

Results

The expected return codes are 0 (zero) and 4 for the DSNTIJCL job.

What to do next

Now, the Db2 subsystem is enabled for spatial support. You can submit the DSN5SCMD JCL in the SDSNSAMP library to verify that the Spatial functions are correctly enabled.

Related concepts

"The DSN5SCLP program" on page 231

DSN5SCLP is an ODBC program that you can use to invoke IBM Spatial Support for Db2 for z/OS stored procedures for administrative tasks.

Related reference

"enable_spatial" on page 246

Use the enable_spatial command to supply a Db2 subsystem with the resources that it needs to store spatial data and support spatial operations.

Related information

Chapter 4. Setting up spatial resources

After your Db2 subsystem is enabled for spatial support, you are ready to set up the resources that you need in order to use spatial data.

Among these resources are a coordinate system to which spatial data conforms and a spatial reference system, which defines the extent of the geographical area that is referenced by the data. This information discusses the nature of coordinate systems and tells how to create them. This information also explains what spatial reference systems are and tells how to create them.

How to use coordinate systems

This discussion explains the concept of coordinate systems and introduces the tasks of selecting one to use and creating a new one.

When you plan a project that uses spatial data, you need to determine whether the data should be based on one of the coordinate systems that are registered to the IBM Spatial Support for Db2 for z/OS catalog. If none of these coordinate systems meet your requirements, you can create one that does.

Coordinate systems

A coordinate system is a framework for defining the relative locations of things in a given area; for example, an area on the earth's surface or the earth's surface as a whole.

IBM Spatial Support for Db2 for z/OS supports the following types of coordinate systems to determine the location of a geographic feature:

Geographic coordinate system
A *geographic coordinate* system is a reference system that uses a three-dimensional spherical surface to determine locations on the earth. Any location on earth can be referenced by a point with latitude and longitude coordinates based on angular units of measure.

Projected coordinate system
A *projected coordinate system* is a flat, two-dimensional representation of the earth. It uses rectilinear (Cartesian) coordinates based on linear units of measure. It is based on a spherical (or spheroidal) earth model, and its coordinates are related to geographic coordinates by a projection transformation.

Geographic coordinate system

A *geographic coordinate system* uses a three-dimensional spherical surface to determine locations on the earth.

Any location on earth can be referenced by a point with longitude and latitude coordinates. The values for the points can have the following units of measurement:

- Linear units when the geographic coordinate system has a spatial reference system identifier (SRID) that IBM Spatial Support for Db2 for z/OS recognizes.
- Any of the following units when the geographic coordinate system has an SRID that IBM Spatial Support for Db2 for z/OS does not recognize:
  - Decimal degrees
  - Decimal minutes
  - Decimal seconds
  - Gradians
  - Radians

For example, the following figure shows a geographic coordinate system where a location is represented by the coordinates longitude 80 degree East and latitude 55 degree North.

© Copyright IBM Corp. 2007, 2013

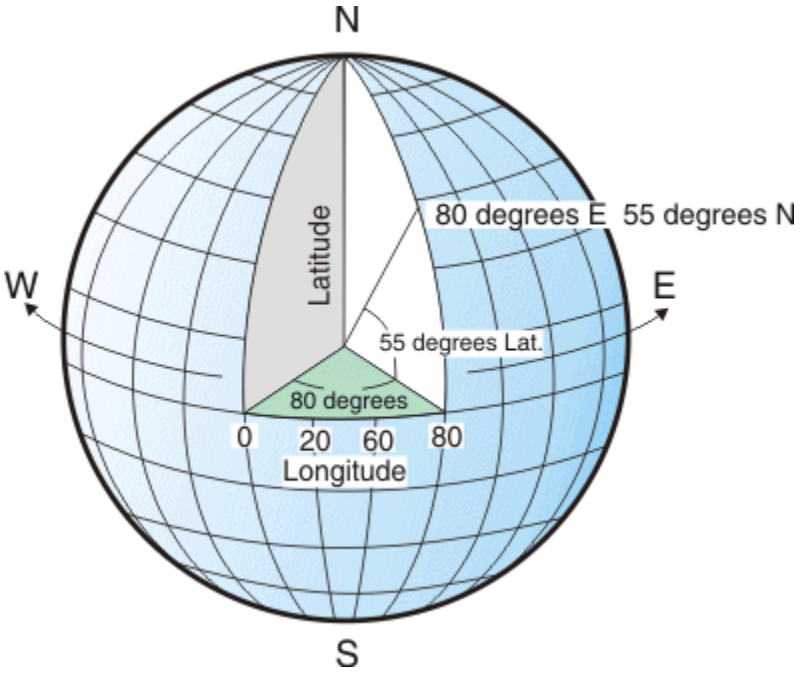


Figure 4. A geographic coordinate system

The lines that run east and west each have a constant latitude value and are called *parallels*. They are equidistant and parallel to one another, and form concentric circles around the earth. The *equator* is the largest circle and divides the earth in half. It is equal in distance from each of the poles, and the value of this latitude line is zero. Locations north of the equator have positive latitudes that range from 0 to +90 degrees, while locations south of the equator have negative latitudes that range from 0 to -90 degrees.

The following figure illustrates latitude lines.

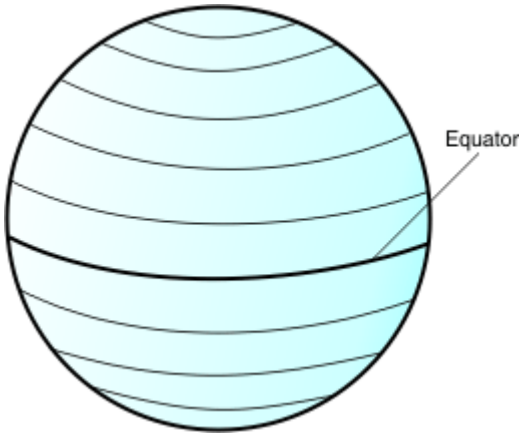


Figure 5. Latitude lines

The lines that run north and south each have a constant longitude value and are called *meridians*. They form circles of the same size around the earth, and intersect at the poles. The *prime meridian* is the line of longitude that defines the origin (zero degrees) for longitude coordinates. One of the most commonly used prime meridian locations is the line that passes through Greenwich, England. However, other longitude lines, such as those that pass through Bern, Bogota, and Paris, have also been used as the prime meridian. Locations east of the prime meridian up to its *antipodal* meridian (the continuation of the prime meridian on the other side of the globe) have positive longitudes ranging from 0 to +180 degrees. Locations west of the prime meridian have negative longitudes ranging from 0 to –180 degrees.

The following figure illustrates longitude lines.

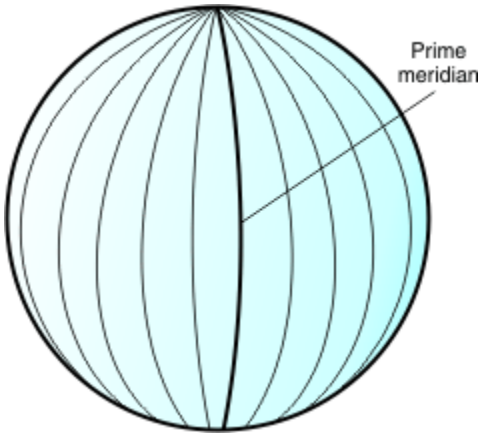


Figure 6. Longitude lines

The latitude and longitude lines can cover the globe to form a grid, called a *graticule*. The point of origin of the graticule is (0,0), where the equator and the prime meridian intersect. The equator is the only place on the graticule where the linear distance corresponding to one degree latitude is approximately equal the distance corresponding to one degree longitude. Because the longitude lines converge at the poles, the distance between two meridians is different at every parallel. Therefore, as you move closer to the poles, the distance corresponding to one degree latitude will be much greater than that corresponding to one degree longitude.

It is also difficult to determine the lengths of the latitude lines using the graticule. The latitude lines are concentric circles that become smaller near the poles. They form a single point at the poles where the meridians begin. At the equator, one degree of longitude is approximately 111.321 kilometers, while at 60 degrees of latitude, one degree of longitude is only 55.802 km (this approximation is based on the Clarke 1866 spheroid). Therefore, because there is no uniform length of degrees of latitude and longitude, the distance between points cannot be measured accurately by using angular units of measure.

The following figure shows the different dimensions between locations on the graticule.

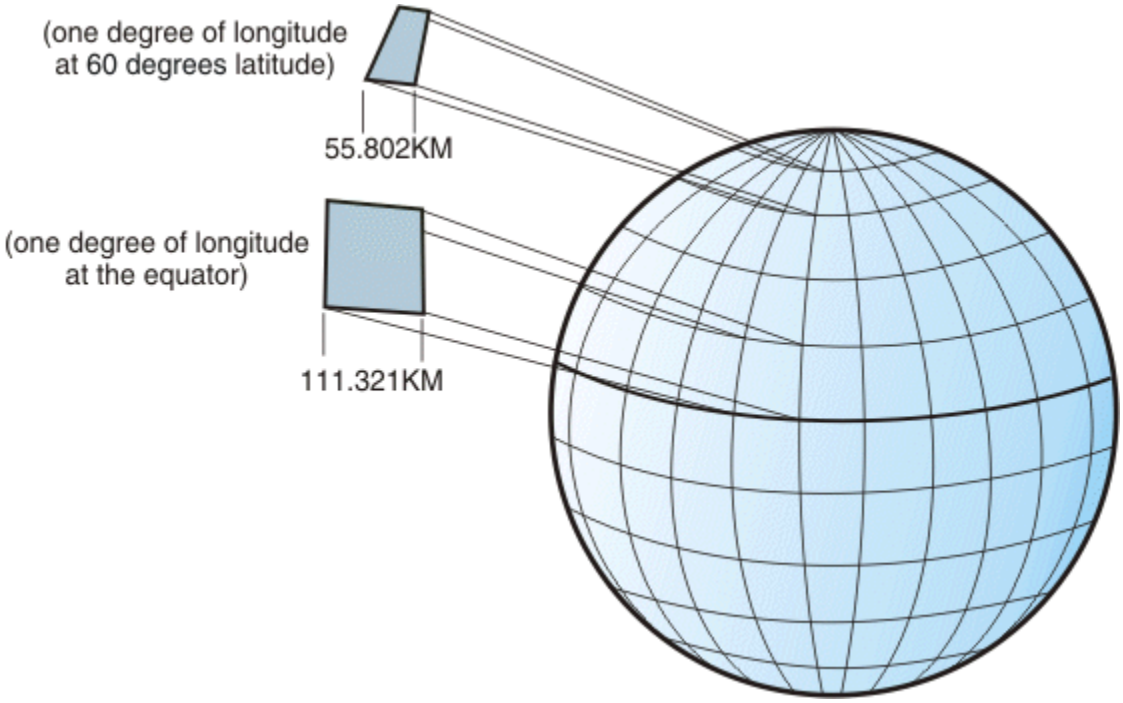


Figure 7. Different dimensions between locations on the graticule

A coordinate system can be defined by either a sphere or a spheroid approximation of the earth's shape. Because the earth is not perfectly round, a spheroid can help maintain accuracy for a map, depending on

the location on the earth. A *spheroid* is an ellipsoid, that is based on an ellipse, whereas a sphere is based on a circle.

The shape of the ellipse is determined by two radii. The longer radius is called the semi-major axis, and the shorter radius is called the semi-minor axis. An ellipsoid is a three-dimensional shape formed by rotating an ellipse around one of its axes.

The following figure shows the sphere and spheroid approximations of the earth and the major and minor axes of an ellipse.

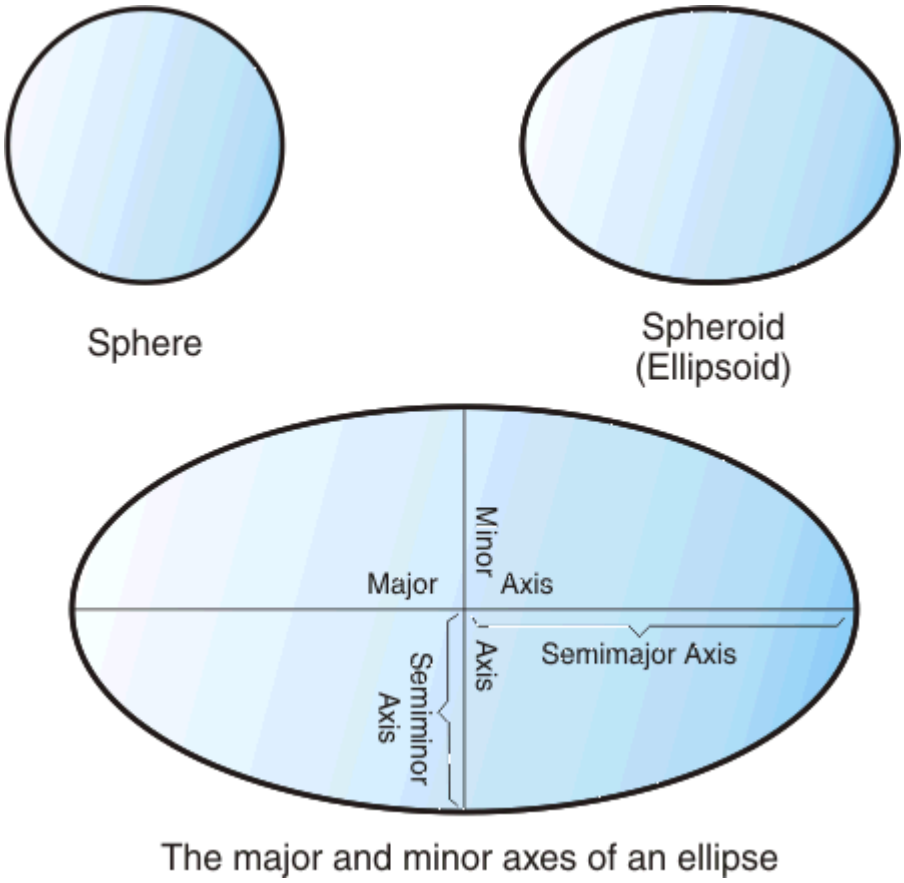


Figure 8. Sphere and spheroid approximations

A *datum* is a set of values that defines the position of the spheroid relative to the center of the earth. The datum provides a frame of reference for measuring locations and defines the origin and orientation of latitude and longitude lines. Some datums are global and intend to provide good average accuracy around the world. A local datum aligns its spheroid to closely fit the earth's surface in a particular area. Therefore, the coordinate system's measurements are not be accurate if they are used with an area other than the one that they were designed.

The following figure shows how different datums align with the earth's surface. The local datum, North American Datum of 1927 (NAD27), more closely aligns with Earth's surface than the Earth-centered datum, World Geodetic System 1984 (WGS84), at this particular location.

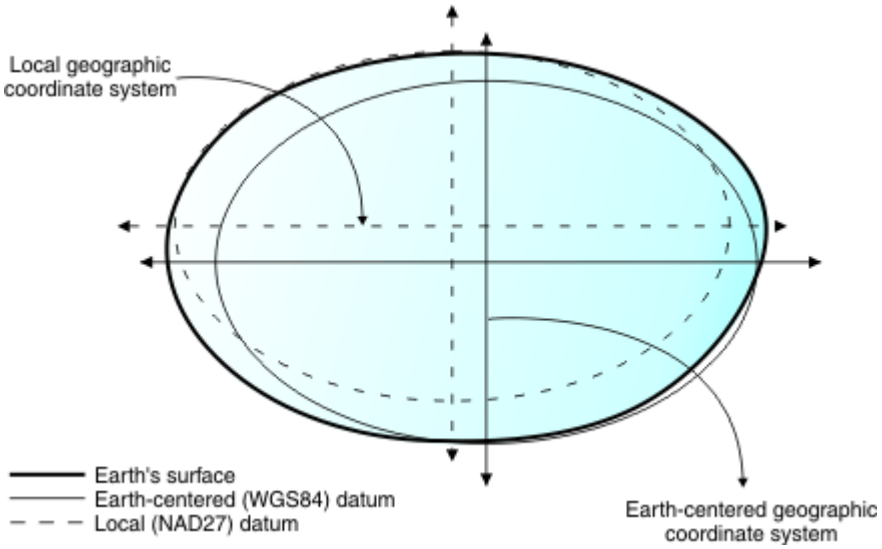


Figure 9. Datum alignments

Whenever you change the datum, the geographic coordinate system is altered and the coordinate values will change. For example, the coordinates in DMS of a control point in Redlands, California using the North American Datum of 1983 (NAD 1983) are: "-117 12 57.75961 34 01 43.77884" The coordinates of the same point on the North American Datum of 1927 (NAD 1927) are: "-117 12 54.61539 34 01 43.72995".

Projected coordinate systems

A *projected coordinate system* is a flat, two-dimensional representation of the Earth.

A projected coordinate system is based on a sphere or spheroid geographic coordinate system, but it uses linear units of measure for coordinates, so that calculations of distance and area are easily done in terms of those same units.

The latitude and longitude coordinates are converted to x, y coordinates on the flat projection. The x coordinate is usually the eastward direction of a point, and the y coordinate is usually the northward direction of a point. The center line that runs east and west is referred to as the x axis, and the center line that runs north and south is referred to as the y axis.

The intersection of the x and y axes is the origin and usually has a coordinate of (0,0). The values above the x axis are positive, and the values below the x axis are negative. The lines parallel to the x axis are equidistant from each other. The values to the right of the y axis are positive, and the values to the left of the y axis are negative. The lines parallel to the y axis are equidistant.

Mathematical formulas are used to convert a three-dimensional geographic coordinate system to a two-dimensional flat projected coordinate system. The transformation is referred to as a *map projection*. Map projections usually are classified by the projection surface used, such as conic, cylindrical, and planar surfaces. Depending on the projection used, different spatial properties will appear distorted. Projections are designed to minimize the distortion of one or two of the data's characteristics, yet the distance, area, shape, direction, or a combination of these properties might not be accurate representations of the data that is being modeled. There are several types of projections available. While most map projections attempt to preserve some accuracy of the spatial properties, there are others that attempt to minimize overall distortion instead, such as the *Robinson* projection. The most common types of map projections include:

Equal area projections
: These projections preserve the area of specific features. These projections distort shape, angle, and scale. The *Albers Equal Area Conic* projection is an example of an equal area projection.

Conformal projections
: These projections preserve local shape for small areas. These projections preserve individual angles to describe spatial relationships by showing perpendicular graticule lines that intersect at 90 degree

angles on the map. All of the angles are preserved; however, the area of the map is distorted. The *Mercator* and *Lambert Conformal Conic* projections are examples of conformal projections.

Equidistant projections
These projections preserve the distances between certain points by maintaining the scale of a given data set. Some of the distances will be true distances, which are the same distances at the same scale as the globe. If you go outside the data set, the scale will become more distorted. The *Sinusoidal* projection and the *Equidistant Conic* projection are examples of equidistant projections.

True-direction or azimuthal projections
These projections preserve the direction from one point to all other points by maintaining some of the great circle arcs. These projections give the directions or azimuths of all points on the map correctly with respect to the center. Azimuthal maps can be combined with equal area, conformal, and equidistant projections. The *Lambert Equal Area Azimuthal* projection and the *Azimuthal Equidistant* projection are examples of azimuthal projections.

Selecting or creating coordinate systems

One of the first steps in planning a project is to determine what coordinate system to use.

Before you begin

Before you create a coordinate system, your user ID must have either SYSADM or DBADM authority on the database that contains the spatial catalog tables.

About this task

After you enable the Db2 subsystem for spatial operations, you are ready to plan projects that use spatial data. You can use a coordinate system that is included with IBM Spatial Support for Db2 for z/OS or one that was created elsewhere. Many coordinate systems are included with IBM Spatial Support for Db2 for z/OS.

To find out about these coordinate systems, and to determine what other coordinate systems are included with IBM Spatial Support for Db2 for z/OS, and what (if any) coordinate systems have been created by other users, consult the DB2GSE.ST_COORDINATE_SYSTEMS catalog view.

Procedure

Run an application that invokes the ST_create_coordsys stored procedure.

For more information about this stored procedure, see “ST_create_coordsys” on page 50.

How to set up spatial reference systems

When you plan a project that uses spatial data, you need to determine whether any of the spatial reference systems available to you can be used for this data.

If none of the available systems are appropriate for the data, you can create one that is. This information explains the concept of spatial reference systems and describes the tasks of selecting which one to use and creating one.

Spatial reference systems

This topic provides an overview of spatial reference systems.

A *spatial reference system* is a set of parameters that includes:

- The name of the coordinate system from which the coordinates are derived.
- The numeric identifier that uniquely identifies the spatial reference system.
- Coordinates that define the maximum possible extent of space that is referenced by a given range of coordinates.
- Numbers that, when applied in certain mathematical operations, convert coordinates received as input into values that can be processed with maximum efficiency.

The following sections discuss the parameter values that define an identifier, a maximum extent of space, and conversion factors.

Spatial reference system identifier

The spatial reference system identifier (SRID) is used as an input parameter for various spatial functions.

Defining the space that encompasses coordinates stored in a spatial column

The coordinates in a spatial column typically define locations that span across part of the Earth. The space over which the span extends—from east to west and from north to south—is called a *spatial extent*. For example, consider a body of flood plains whose coordinates are stored in a spatial column. Suppose that the westernmost and easternmost of these coordinates are latitude values of –24.556 and –19.338, respectively, and that the northernmost and southernmost of the coordinates are longitude values of 18.819 and 15.809 degrees, respectively. The spatial extent of the flood plains is a space that extends on a west-east plane between the two latitudes and on a north-south plane between the two longitudes. You can include these values in a spatial reference system by assigning them to certain parameters. If the spatial column includes Z coordinates and measures, you would need to include the highest and lowest Z coordinates and measures in the spatial reference system as well.

The term spatial extent can refer not only to an actual span of locations, as in the previous paragraph; but also to a potential one. Suppose that the flood plains in the preceding example were expected to broaden over the next five years. You could estimate what the westernmost, easternmost, northernmost, and southernmost coordinates of the planes would be at the end of the fifth year. You could then assign these estimates, rather than the current coordinates, to the parameters for a spatial extent. That way, you could retain the spatial reference system as the plains expand and their wider latitudes and longitudes are added to the spatial column. Otherwise, if the spatial reference system is limited to the original latitudes and longitudes, it would need to be altered or replaced as the flood planes grew.

Converting to values that improve performance

Typically, most coordinates in a coordinate system are decimal values; some are integers. In addition, coordinates to the east of the origin are positive; those to the west are negative. Before being stored by IBM Spatial Support for Db2 for z/OS, the negative coordinates are converted to positive values, and the decimal coordinates are converted into integers. As a result, all coordinates are stored by IBM Spatial Support for Db2 for z/OS as positive integers. The purpose is to enhance performance when the coordinates are processed.

Certain parameters in a spatial reference system are used to make the conversions described in the preceding paragraph. One parameter, called an *offset*, is subtracted from each negative coordinate, which leaves a positive value as a remainder. Each decimal coordinate is multiplied by another parameter, called a *scale factor*, which results in an integer whose precision is the same as that of the decimal coordinate. (The offset is subtracted from positive coordinates as well as negative; and the nondecimal coordinates, as well as the decimal coordinates, are multiplied by the scale factor. This way, all positive and non-decimal coordinates remain commensurate with the negative and decimal ones.)

These conversions take place internally, and remain in effect only until coordinates are retrieved. Input and query results always contain coordinates in their original, unconverted form.

Deciding whether to use a default spatial reference system or create a new system

After you determine what coordinate system to use, you are ready to provide a spatial reference system that suits the coordinate data that you are working with.

About this task

IBM Spatial Support for Db2 for z/OS provides five spatial reference systems for spatial data.

Procedure

Answer the following questions to determine whether you can use one of the default spatial reference systems:

1. Does the coordinate system on which the default spatial reference system is based cover the geographic area that you are working with? These coordinate systems are shown in "Spatial reference systems supplied with IBM Spatial Support for Db2 for z/OS" on page 20.
2. Do the conversion factors associated with one of the default spatial reference systems work with your coordinate data?

   IBM Spatial Support for Db2 for z/OS uses offset values and scale factors to convert the coordinate data that you provide to positive integers. To determine if your coordinate data works with the given offset values and scale factors for one of the default spatial reference systems:

   a. Review the information in "Conversion factors that transform coordinate data into integers" on page 23.
   b. Look at how these factors are defined for the default spatial reference systems. If, after applying the offset value to the minimum X and Y coordinates, these coordinates are not both greater than 0, you must create a new spatial reference system and define the offsets yourself. For more information about how to create a new spatial reference system, see "Creating a spatial reference system" on page 22.
3. Does the data that you are working with include height and depth coordinates (Z coordinates) or measures (M coordinates)?

   If you are working with Z or M coordinates, you might need to create a new spatial reference system with Z or M offsets and scale factors suitable to your data.
4. If the existing spatial reference systems do not work with your data, you need to create a spatial reference system.

   For more information, see "Creating a spatial reference system" on page 22.

What to do next

After you decide which spatial reference system you need, you specify this choice to IBM Spatial Support for Db2 for z/OS. For more information see "Spatial reference systems supplied with IBM Spatial Support for Db2 for z/OS" on page 20.

Spatial reference systems supplied with IBM Spatial Support for Db2 for z/OS

The spatial reference system converts the coordinate data to positive integers.

IBM Spatial Support for Db2 for z/OS provides the spatial reference systems that are shown in the table below, along with the coordinate system on which each spatial reference system is based and the offset values and scale factors that IBM Spatial Support for Db2 for z/OS uses to convert the coordinate data to positive integers. You can find information about these spatial reference systems in the DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS catalog view.

If you are working with decimal-degrees, the offset values and scale factors for the default spatial reference systems support the full range of latitude-longitude coordinates and preserve 6 decimal positions, equivalent to approximately 10 cm.

If none of the default spatial reference systems meet your needs, you can create a new spatial reference system.

Table 1. Spatial reference systems provided with IBM Spatial Support for Db2 for z/OS

Spatial reference system	SRS ID	Coordinate system	Offset values	Scale factors	When to use
DEFAULT _SRS	0	None	xOffset = 0 yOffset = 0 zOffset = 0 mOffset = 0	xScale = 1 yScale = 1 zScale = 1 mScale = 1	You can select this system when your data is independent of a coordinate system or you cannot or do not need to specify one.
NAD83_ SRS_1	1	GCS_NORTH _AMERICAN _1983	xOffset = –180 yOffset = –90 zOffset = 0 mOffset = 0	xScale = 1,000,000 yScale = 1,000,000 zScale = 1 mScale = 1	You can select this spatial reference system if you plan to use the U.S. sample data that was previously available from Db2 Spatial Extender. If the coordinate data that you are working with was collected after 1983, use this system instead of NAD27_SRS_1002.
NAD27_ SRS_1002	1002	GCS_NORTH _AMERICAN _1927	xOffset = –180 yOffset = –90 zOffset = 0 mOffset = 0	xScale = 5,965,232 yScale = 5,965,232 zScale = 1 mScale = 1	You can select this spatial reference system if you plan to use the U.S. sample data that was previously available from Db2 Spatial Extender. If the coordinate data that you are working with was collected before 1983, use this system instead of NAD83_SRS_1. This system provides a greater degree of precision than the other default spatial reference systems.
WGS84_ SRS_1003	1003	GCS_WGS _1984	xOffset = –180 yOffset = –90 zOffset = 0 mOffset = 0	xScale = 5,965,232 yScale = 5,965,232 zScale = 1 mScale = 1	You can select this spatial reference system if you are working with data outside the U.S. (This system handles worldwide coordinates.)

Table 1. Spatial reference systems provided with IBM Spatial Support for Db2 for z/OS (continued)

Spatial reference system	SRS ID	Coordinate system	Offset values	Scale factors	When to use
DE_HDN _SRS_1004	1004	GCS _DEUTSCHES _HAUPTDREI ECKSNETZ	xOffset = –180 yOffset = –90 zOffset = 0 mOffset = 0	xScale = 5,965,232 yScale = 5,965,232 zScale = 1 mScale = 1	This spatial reference system is based on a coordinate system for German addresses.

Creating a spatial reference system

Create a new spatial reference system if none of the spatial reference systems that are provided with IBM Spatial Support for Db2 for z/OS work with your data.

About this task

You use the SYSPROC.ST_create_srs stored procedure to create a spatial reference system. This stored procedure has two variations. The first variation, the SYSPROC.ST_create_srs stored procedure, takes the conversion factors (offsets and scale factors) as input parameters. The second variation, the ST_create_srs_2 stored procedure, takes the extents and the precision as input parameters and calculates the conversion factors internally. For more information, see “ST_create_srs” on page 54 and “ST_create_srs_2” on page 57.

Procedure

To create a spatial reference system by using the SYSPROC.ST_create_srs stored procedure:

1. Specify an appropriate spatial reference system ID (SRID).

   For spatial data in a flat-earth representation, specify an SRID that is not already defined.
2. Decide on the degree of precision that you want.

   You can either:

   - Specify the extents of the geographical area that you are working with and the scale factors that you want to use with your coordinate data. IBM Spatial Support for Db2 for z/OS takes the extents that you specify and calculates the offset for you. To specify the extents, provide the appropriate parameters for the SYSPROC.ST_create_srs2 stored procedure.
   - Specify both the offset values (required for IBM Spatial Support for Db2 for z/OS to convert negative values to positive values) and scale factors (required for IBM Spatial Support for Db2 for z/OS to convert decimal values to integers). Use this method when you need to follow strict criteria for accuracy or precision. To specify the offset values and scale factors, provide the appropriate parameters for the SYSPROC.ST_create_srs stored procedure.
3. Calculate the conversion information that IBM Spatial Support for Db2 for z/OS needs to convert coordinate data to positive integers, and provide this information to the interface that you chose.

   This information differs according to the method that you chose in the previous step.

   - If you chose to specify the extents, you need to calculate the following information:
     – Scale factors. If any of the coordinates that you are working with are decimal values, calculate scale factors. Scale factors are numbers that, when multiplied by decimal coordinates and measures, yields integers with at least the same number of significant digits as the original coordinates and measures. If the coordinates are integers, the scale factors can be set to 1. If the coordinates are decimal values, the scale factor should be set to a number that converts the decimal portion to an integer value. For example, if the coordinate units are meters and the accuracy of the data is 1 cm, you would need a scale factor of 100.

- Minimum and maximum values for your coordinates and measures.

- If you chose to specify the offset values and scale factors, you need to calculate the following information:
  - Offset values

    If your coordinate data includes negative numbers or measures, you need to specify the offset values that you want to use. An offset is a number that is subtracted from all coordinates, leaving only positive values as a remainder. If you are working with positive coordinates, set all offset values to 0. If you are not working with positive coordinates, select an offset that, when applied against the coordinate data, results in integers that are less than the largest positive integer value (9,007,199,254,740,992).
  - Scale factors

    If any of the coordinates for the locations that you are representing are decimal numbers, determine what scale factors to use.

4. Run an application that invokes the SYSPROC.ST_create_srs stored procedure to create the spatial reference system.

Conversion factors that transform coordinate data into integers

IBM Spatial Support for Db2 for z/OS uses offset values and scale factors to convert the coordinate data that you provide to positive integers.

The default spatial reference systems already have offset value and scale factors associated with them. If you are creating a new spatial reference system, determine the scale factors and, optionally, the offset values that work best with your data.

Offset values

An offset value is a number that is subtracted from all coordinates, leaving only positive values as a remainder.

IBM Spatial Support for Db2 for z/OS converts your coordinate data using the following formulas to ensure that all adjusted coordinate values are greater than 0.

Formula notation: In these formulas, the notation "min" represents "the minimum of all". For example, "min(x)" means "the minimum of all x coordinates". The offset for each geographic direction is represented as dimensionOffset. For example, xOffset is the offset value applied to all X coordinates.

```
min(x) – xOffset ≥ 0
min(y) – yOffset ≥ 0
min(z) – zOffset ≥ 0
min(m) – mOffset ≥ 0
```

Scale factors

A scale factor is a value that, when multiplied by decimal coordinates and measures, yields integers with at least the same number of significant digits as the original coordinates and measures.

IBM Spatial Support for Db2 for z/OS converts your decimal coordinate data using the following formulas to ensure that all adjusted coordinate values are positive integers. The converted values cannot exceed 2^{53} (approximately $9 * 10^{15}$).

Formula notation: In these formulas, the notation "max" represents "the maximum of all". The offset for each geographic dimension is represented as dimensionOffset (for example, xOffset is the offset value applied to all X coordinates). The scale factor for each geographic dimension is represented as dimensionScale (for example, xScale is the scale factor applied to X coordinates).

$$(max(x) - xOffset) * xScale \leq 2^{53}$$
$$(max(y) - yOffset) * yScale \leq 2^{53}$$
$$(max(z) - zOffset) * zScale \leq 2^{53}$$
$$(max(m) - mOffset) * mScale \leq 2^{53}$$

When you choose which scale factors work best with your coordinate data, ensure that:

- You use the same scale factor for X and Y coordinates.
- When multiplied by a decimal X coordinate or a decimal Y coordinate, the scale factor yields a value less than 2^{53}. One common technique is to make the scale factor a power of 10. That is, the scale factor should be 10 to the first power (10), 10 to the second power (100), 10 to the third power (1000), or, if necessary, a larger factor.
- The scale factor is large enough to ensure that the number of significant digits in the new integer is the same as the number of significant digits in the original decimal coordinate.

Example

Suppose that the ST_Point function is given input that consists of an X coordinate of 10.01, a Y coordinate of 20.03, and the identifier of a spatial reference system. When ST_Point is invoked, it multiplies the value of 10.01 and the value of 20.03 by the spatial reference system's scale factor for X and Y coordinates. If this scale factor is 10, the resulting integers that IBM Spatial Support for Db2 for z/OS stores will be 100 and 200, respectively. Because the number of significant digits in these integers (3) is less than the number of significant digits in the coordinates (4), IBM Spatial Support for Db2 for z/OS will not be able to convert these integers back to the original coordinates, or to derive from them values that are consistent with the coordinate system to which these coordinates belong. But if the scale factor is 100, the resulting integers that IBM Spatial Support for Db2 for z/OS stores will be 1001 and 2003—values that can be converted back to the original coordinates or from which compatible coordinates can be derived.

Units for offset values and scale factors

Whether you use an existing spatial reference system or create a new one, the units for the offset values and scale factors will vary depending on the type of coordinate system that you are using.

For example, if you are using a geographic coordinate system, the values are in angular units such as decimal degrees; if you are using a projected coordinate system, the values are in linear units such as meters or feet.

Calculating offset values

If you create a spatial reference system and your coordinate data includes negative numbers or measures, you need to specify the offset values that you want to use.

About this task

An offset is a number that is subtracted from all coordinates, leaving only positive values as a remainder. You can improve the performance of spatial operations when the coordinates are positive integers instead of negative numbers or measures.

Procedure

To calculate the offset values for the coordinates that you are working with:

1. Determine the lowest negative X, Y, and Z coordinates within the range of coordinates for the locations that you want to represent. If your data is to include negative measures, determine the lowest of these measures.
2. Optional but recommended: Indicate to IBM Spatial Support for Db2 for z/OS that the domain that encompasses the locations that you are concerned with is larger than it actually is. Thus, after you write data about these locations to a spatial column, you can add data about locations of new features as they are added to outer reaches of the domain, without having to replace your spatial reference system with another one.

   For each coordinate and measure that you identified in step 1, add an amount equal to five to ten percent of the coordinate or measure. The result is referred to as an *augmented value*. For example, if the lowest negative X coordinate is –100, you could add –5 to it, yielding an augmented value of –105. Later, when you create the spatial reference system, you will indicate that the lowest X coordinate is –

105, rather than the true value of –100. IBM Spatial Support for Db2 for z/OS will then interpret –105 as the westernmost limit of your domain.

3. Find a value that, when subtracted from your augmented X value, leaves zero; this is the offset value for X coordinates. IBM Spatial Support for Db2 for z/OS subtracts this number from all X coordinates to produce only positive values.

   For example, if the augmented X value is –105, you need to subtract –105 from it to get 0. IBM Spatial Support for Db2 for z/OS will then subtract –105 from all X coordinates that are associated with the features that you are representing. Because none of these coordinates is greater than –100, all the values that result from the subtraction will be positive.
4. Repeat step 3 for the augmented Y value, augmented Z value, and augmented measure.

Calculating scale factors

Scale factors are numbers that, when multiplied by decimal coordinates and measures, yields integers with at least the same number of significant digits as the original coordinates and measures.

About this task

If you create a spatial reference system and any of the coordinates that you are working with are decimal values, calculate the appropriate scale factors for your coordinates and measures.

After you calculate scale factors, you need to determine the extent values. Then use the ST_create_srs stored procedure to create a spatial reference system.

Procedure

To calculate the scale factors:

1. Determine which X and Y coordinates are, or are likely to be, decimal numbers. For example, suppose that of the various X and Y coordinates that you will be dealing with, you determine that three of them are decimal numbers: 1.23, 5.1235, and 6.789.
2. Find the decimal coordinate that has the longest decimal precision. Then determine by what power of 10 this coordinate can be multiplied to yield an integer of equal precision. For example, of the three decimal coordinates in the current example, 5.1235 has the longest decimal precision. Multiplying it by 10 to the fourth power (10000) yields the integer 51235.
3. Determine whether the integer produced by the multiplication just described is less than 2^{53}. 51235 is not too large. But suppose that, in addition to 1.23, 5.11235, and 6.789, your range of X and Y coordinates includes a fourth decimal value, 10000000006.789876. Because this coordinate's decimal precision is longer than that of the other three, you would multiply this coordinate—not 5.1235—by a power of 10. To convert it to an integer, you could multiply it by 10 to the sixth power (1000000). But the resulting value, 10000000006789876, is greater than 2^{53}. If IBM Spatial Support for Db2 for z/OS tried to store it, the results would be unpredictable.

   To avoid this problem, select a power of 10 that, when multiplied by the original coordinate, yields a decimal number that IBM Spatial Support for Db2 for z/OS can truncate to a storable integer, with minimum loss of precision. In this case, you could select 10 to the fifth power (100000). Multiplying 100000 by 10000000006.789876 yields 1000000000678987.6. IBM Spatial Support for Db2 for z/OS would round this number to 1000000000678988, reducing its accuracy slightly.

Determining minimum and maximum coordinates and measures

Determine minimum and maximum coordinates and measures if you decide to specify extent transformations when you create a spatial reference system.

About this task

Use this process to determine minimum and maximum coordinates and measures if you:

- Decide to create a new spatial reference system because none of the spatial reference systems provided with IBM Spatial Support for Db2 for z/OS work with your data.

- Decide to use extent transformations to convert your coordinates.

After you determine the extent values, if any of the coordinates are decimal values, you need to calculate scale factors. Otherwise, use the ST_create_srs stored procedure to create a spatial reference system.

Procedure

To determine the minimum and maximum coordinates and measures of the locations that you want to represent:

1. Determine the minimum and maximum X coordinates.

   To find the minimum X coordinate, identify the X coordinate in your domain that is furthest west. (If the location lies to the west of the point of origin, this coordinate will be a negative value.) To find the maximum X coordinate, identify the X coordinate in your domain that is furthest east. For example, if you are representing oil wells, and each one is defined by a pair of X and Y coordinates, the X coordinate that indicates the location of the oil well that is furthest west is the minimum X coordinate, and the X coordinate that indicates the location of the oil well that is furthest east is the maximum X coordinate.

   Tip: For multifeature types, such as multipolygons, ensure that you pick the furthest point on the furthest polygon in the direction that you are calculating. For example, if you are trying to identify the minimum X coordinate, identify the westernmost X coordinate of the polygon that is furthest west in the multipolygon.
2. Determine the minimum and maximum Y coordinates.

   To find the minimum Y coordinate, identify the Y coordinate in your domain that is furthest south. (If the location lies to the south of the point of origin, this coordinate will be a negative value.) To determine the maximum Y coordinate, find the Y coordinate in your domain that is furthest north.
3. Determine the minimum and maximum Z coordinates.

   The minimum Z coordinate is the greatest of the depth coordinates and the maximum Z coordinate is the greatest of the height coordinates.
4. Determine the minimum and maximum measures.

   If you are going to include measures in your spatial data, determine which measure has the highest numerical value and which has the lowest.

Chapter 5. Setting up spatial columns

In preparing to obtain spatial data for a project, you not only choose or create a coordinate system and spatial reference system; you also provide one or more table columns to contain the data.

Spatial columns

This information provides an overview of spatial columns and the spatial data types that you can use.

Spatial columns with viewable content

When you use a visualization tool to query a spatial column, the tool returns results in the form of a graphical display; for example, a map of parcel boundaries or the layout of a road system.

Some visualization tools require all rows of the column to use the same spatial reference system. The way you enforce this constraint is to register the column with a spatial reference system.

Spatial data types

When you enable a Db2 subsystem for spatial operations, IBM Spatial Support for Db2 for z/OS provides the database with seven distinct geometry types.

Six of these geometry types are instantiable, and one is abstract.

The six instantiable data types are ST_Point, ST_LineString, ST_Polygon, ST_MultiPoint, ST_MultiLineString, and ST_MultiPolygon.

The data type that is abstract, or not instantiable, is ST_Geometry.

Spatial data types include support for the following:

- Data types for geographic features that can be perceived as forming a single unit; for example, individual residences and isolated lakes.
- Data types for geographic features that are made up of multiple units or components; for example, canal systems and groups of islands in a lake.
- A data type for geographic features of all kinds.

Data types for single-unit features

Use ST_Point, ST_LineString, and ST_Polygon to store coordinates that define the space occupied by features that can be perceived as forming a single unit.

- Use ST_Point when you want to indicate the point in space that is occupied by a discrete geographic feature. The feature might be a very small one, such as a water well; a very large one, such as a city; or one of intermediate size, such as a building complex or park. In each case, the point in space can be located at the intersection of an east-west coordinate line (for example, a parallel) and a north-south coordinate line (for example, a meridian). An ST_Point data item includes an X coordinate and a Y coordinate that define such an intersection. The X coordinate indicates where the intersection lies on the east-west line; the Y coordinate indicates where the intersection lies on the north-south line.
- Use ST_Linestring for coordinates that define the space that is occupied by linear features; for example, streets, canals, and pipelines.
- Use ST_Polygon when you want to indicate the extent of space covered by a multi-sided feature; for example, a voting district, a forest, or a wildlife habitat. An ST_Polygon data item consists of the coordinates that define the boundary of such a feature.

In some cases, ST_Polygon and ST_Point can be used for the same feature. For example, suppose that you need spatial information about an apartment complex. If you want to represent the point in space where each building in the complex is located, you would use ST_Point to store the X and Y coordinates that define each such point. Otherwise, if you want to represent the area occupied by the complex as a whole, you would use ST_Polygon to store the coordinates that define the boundary of this area.

© Copyright IBM Corp. 2007, 2013

Data types for multi-unit features
Use ST_MultiPoint, ST_MultiLineString, and ST_MultiPolygon to store coordinates that define spaces occupied by features that are made up of multiple units.

- Use ST_MultiPoint when you are representing features made up of units whose locations are each referenced by an X coordinate and a Y coordinate. For example, consider a table whose rows represent island chains. The X coordinate and Y coordinate for each island has been identified. If you want the table to include these coordinates and the coordinates for each chain as a whole, define an ST_MultiPoint column to hold these coordinates.
- Use ST_MultiLineString when you are representing features made up of linear units, and you want to store the coordinates for the locations of these units and the location of each feature as a whole. For example, consider a table whose rows represent river systems. If you want the table to include coordinates for the locations of the systems and their components, define an ST_MultiLineString column to hold these coordinates.
- Use ST_MultiPolygon when you are representing features made up of multi-sided units, and you want to store the coordinates for the locations of these units and the location of each feature as a whole. For example, consider a table whose rows represent rural counties and the farms in each county. If you want the table to include coordinates for the locations of the counties and farms, define an ST_MultiPolygon column to hold these coordinates.

Multi-unit is not meant as a collection of individual entities. Rather, multi-unit refers to an aggregate of the parts that makes up the whole.

A data type for all features
You can use ST_Geometry when you are not sure which of the other data types to use.

An ST_Geometry column can contain the same kinds of data items that columns of the other data types can contain.

Creating spatial columns

You must create spatial columns to store and retrieve spatial data. After you choose a coordinate system and determine which spatial reference system to use for your data, you create a spatial column in an existing table or import spatial data into a new table.

Before you begin

Before you create a spatial column, your user ID must hold the authorizations that are needed for the Db2 SQL CREATE TABLE statement or ALTER TABLE statement.

Procedure

Use one of the following approaches:

- For a new table, issue the CREATE TABLE statement to create the table and to include a spatial column within that table.
- For an existing table, issue the ALTER TABLE statement to add a spatial column.
- If you are importing spatial data from a shape file, use the SYSPROC.ST_import_shape stored procedure to create a table and to provide this table with a column to hold the data.

Example

The following example shows how to create a table with a spatial column by using the CREATE TABLE statement:

```
CREATE TABLE CUSTOMERS ( ..., LOCATION DB2GSE.ST_POINT, ...) ;
```

This next example shows how to add a spatial column to an existing table by using the ALTER TABLE statement:

```
ALTER TABLE BRANCHES ADD COLUMN LOCATION DB2GSE.ST_POINT;
```

What to do next

Next, you register the spatial column.

Related tasks

"Importing shape data to a new or existing table" on page 33

You can import shape data to an existing table or view, or you can create a table and import shape data to it in a single operation.

"Registering spatial columns" on page 30

Registering a spatial column creates a constraint on the table, if possible, to ensure that all geometries use the specified spatial reference system.

Related information

Creating inline spatial columns

An *inline spatial column* is defined with a LOB data type, or a distinct type that is based on a LOB data type. After you choose a coordinate system and determine which spatial reference system to use for your data, you can create an inline spatial column in an existing table or import spatial data into a new table.

Before you begin

Before you create an inline spatial column, your user ID must hold the authorizations that are needed for the Db2 SQL CREATE TABLE statement or ALTER TABLE statement.

About this task

You can gain performance improvements for data loading, index creation, and queries by using inline LOB columns to store certain types of geometries. These geometry types are ST_LineString, ST_Polygon, ST_MultiPoint, ST_MultiLineString and ST_MultiPolygon, which are based on the BLOB data type. However, inline LOB columns use more storage for a base table space than LOB or non-LOB columns.

Procedure

Use one of the following approaches:

- If your Db2 subsystem is not enabled for spatial support, to create an inline spatial column:
  a. Set the LOB_INLINE_LENGTH subsystem parameter to the appropriate value.
  b. Restart Db2.
  c. Enable you Db2 subsystem for spatial support.
  d. Issue the CREATE TABLE statement to create a new table with a spatial column. The non-point column is created as an inline spatial column. The length of the column is inherited from the value of the LOB_INLINE_LENGTH subsystem parameter.
- If your Db2 subsystem is enabled for spatial support, issue the CREATE TABLE statement and specify the INLINE LENGTH clause to specify the inline length for the column.
- For an existing table with non-point geometry columns, issue the ALTER TABLE ALTER *column-alternation* statement to change the column to be an inline spatial column.

  Tip: After changing an existing column to an inline spatial column, run the REORG TABLESPACE utility.
- For an existing table without geometry columns, issue the ALTER TABLE ADD *column-definition* statement to add an inline spatial column.

Example

The following example assumes that you set the LOB_INLINE_LENGTH subsystem parameter to 1000 bytes, and then enabled spatial support for your Db2 subsystem. To create a new table with an inline spatial column, issue the following statement:

```
CREATE TABLE SYSADM.REAL_ESTATE(ID INTEGER NOT NULL,
PROPERTY  DB2GSE.ST_MULTIPOLYGON);
```

The PROPERTY column is defined as an inline spatial column with a length of 1000 bytes.

You can override the value of the LOB_INLINE_LENGTH subsystem parameter by issuing the CREATE TABLE statement with the INLINE LENGTH clause, as in the following example:

```
CREATE TABLE SYSADM.REAL_ESTATE(ID INTEGER NOT NULL,
PROPERTY DB2GSE.ST_MULTIPOLYGON INLINE LENGTH 500 );
```

The new length of the PROPERTY column is 500 bytes.

In the next example, assume you want to change a column to be an inline spatial column in an existing table with non-point geometry columns. After you issue the following ALTER TABLE ALTER *column-alternation* statement and run the REORG TABLESPACE utility, the PROPERTY column will have a length of 700 bytes:

```
ALTER TABLE SYSADM.REAL_ESTATE ALTER PROPERTY SET INLINE LENGTH 700;
```

Finally, assume you created a table by issuing the following CREATE TABLE statement:

```
CREATE TABLE SYSADM.REAL_ESTATE1(ID INTEGER NOT NULL, PRICE DECIMAL(9,2));
```

Now you want to add an inline spatial column to this existing table that does not have geometry columns. Issue the following ALTER TABLE ADD *column-definition* statement:

```
ALTER TABLE SYSADM.REAL_ESTATE1 ADD COLUMN PROPERTY DB2GSE.ST_MULTIPOLYGON;
```

By issuing this statement, the PROPERTY column inherits the inline LOB length from the LOB_INLINE_LENGTH subsystem parameter, which is set to 1000 bytes. If you want to override the LOB_INLINE_LENGTH subsystem parameter, issue the following statement:

```
ALTER TABLE SYSADM.REAL_ESTATE1 ADD COLUMN PROPERTY1
DB2GSE.ST_MULTIPOLYGON INLINE LENGTH 700;
```

The length of the PROPERTY column is 700 bytes.

What to do next

Next, you register the inline spatial column.

Related tasks

“Registering spatial columns” on page 30

Registering a spatial column creates a constraint on the table, if possible, to ensure that all geometries use the specified spatial reference system.

Related information

Registering spatial columns

Registering a spatial column creates a constraint on the table, if possible, to ensure that all geometries use the specified spatial reference system.

About this task

You might want to register a spatial column in the following situations:

- Access by visualization tools

  If you want certain visualization tools to generate graphical displays of the data in a spatial column, you need to ensure the integrity of the column's data. You do this by imposing a constraint that requires all rows of the column to use the same spatial reference system. To impose this constraint, register the column, specifying both its name and the spatial reference system that applies to it.
- Access by spatial indexes

  Use the same coordinate system for all data in a spatial column on which you want to create an index to ensure that the spatial index returns the correct results. You register a spatial column to constrain all data to use the same spatial reference system and, correspondingly, the same coordinate system.

Procedure

To register a spatial column:

1. Run an application that invokes the SYSPROC.ST_register_spatial_column stored procedure.

   For more information about this stored procedure, see “ST_register_spatial_column” on page 73.
2. Refer to the SRS_NAME column in the DB2GSE.ST_GEOMETRY_COLUMNS view to check the spatial reference system you chose for a particular column after you register the column.

Chapter 6. Populating spatial columns

After you create spatial columns, and register the ones to be accessed by these visualization tools, you are ready to populate the columns with spatial data.

You can supply the data in the following two ways:

- Import the data
- Use spatial functions to create the data or to derive it from business data or other spatial data

About importing spatial data

You can use IBM Spatial Support for Db2 for z/OS to import spatial data from external data sources.

More precisely, you can import spatial data from external sources by transferring it to your database in files, called data exchange files. This section suggests some of the reasons for importing spatial data, and describes the nature of the data exchange files that IBM Spatial Support for Db2 for z/OS supports.

Reasons for importing spatial data

By importing spatial data, you can obtain a great deal of spatial information that is already available in the industry. Consider the following scenario.

Your database contains spatial data that represents your sales offices, customers, and other business concerns. You want to supplement this data with spatial data that represents your organization's cultural environment—cities, streets, points of interest, and so on. The data that you want is available from a map vendor. You can use IBM Spatial Support for Db2 for z/OS to import it from a data exchange file that the vendor supplies. You must place the input files in an HFS directory.

Shape files

IBM Spatial Support for Db2 for z/OS supports importing shape files. The term shape file refers to a collection of files with the same file name but different file extensions. The collection can include up to four files. The files are:

- A file that contains spatial data in shape format, an industry-standard format developed by ESRI. Such data is often called shape data. The extension of a file containing shape data is .shp.
- A file that contains business data that pertains to locations defined by shape data. This file's extension is .dbf. The content of the .dbf file is ASCII data.
- A file that contains an index to shape data. This file's extension is .shx.
- A file that contains a specification of the coordinate system on which the data in a .shp file is based. This file's extension is .prj.

When you use IBM Spatial Support for Db2 for z/OS to import shape data, you receive at least one .shp file. In most cases, you receive one or more of the other three kinds of shape files as well.

Importing spatial data

This information provides an overview of the task of importing shape data to your database.

Importing shape data to a new or existing table

You can import shape data to an existing table or view, or you can create a table and import shape data to it in a single operation.

Before you begin

Before you import shape data, your user ID must hold the following privileges:

© Copyright IBM Corp. 2007, 2013

- Privileges to access the directories where the input files and the error files are located
- Read access to the input files
- Write access to the error files

In addition, before you import shape data to an existing table or view, your user ID must hold one of the following authorities or privileges:

- SYSADM or DBADM authority on the database that contains the table or view
- The INSERT and SELECT privilege on the table or view

Before you begin to create a table automatically and import shape data to the new table, your user ID must hold the authorizations that are needed for the Db2 CREATE TABLE statement.

About this task

You can import shape data in the following two ways:

- You can import the shape data and attribute data to an existing table that has a spatial column and attribute columns with the file's data. This method is the recommended way for importing shape data.
- IBM Spatial Support for Db2 for z/OS can create a table that has a spatial column and attribute columns and load the new table's columns with the file's data. If you choose this method, you cannot create or customize the table spaces for the resulting table.

Procedure

Run an application that calls the SYSPROC.ST_import_shape stored procedure.

Chapter 7. Using indexes to access spatial data

Before you query spatial columns, you can create indexes that will facilitate access to them.

This information describes the nature of the indexes that IBM Spatial Support for Db2 for z/OS uses to expedite access to spatial data, and explains how to create such indexes.

Spatial indexes

Good query performance is related to having efficient indexes defined on the columns of the base tables in a database.

The performance of the query is directly related to how quickly values in the column can be found during the query. Queries that use an index can execute more quickly and can provide a significant performance improvement.

You can gain performance improvements for data loading, index creation, and queries by using inline LOB columns to store certain types of geometries. These geometry types are ST_LineString, ST_Polygon, ST_MultiPoint, ST_MultiLineString and ST_MultiPolygon, which are based on the BLOB data type. However, inline LOB columns use more storage for a base table space than LOB or non-LOB columns.

Spatial queries are typically queries that involve two or more dimensions. For example, in a spatial query you might want to know if a point is included within an area (polygon). Due to the multidimensional nature of spatial queries, the Db2 native B-tree indexing is inefficient for these queries.

Spatial queries use a type of index called a spatial grid index. The indexing technology in IBM Spatial Support for Db2 for z/OS utilizes *grid indexing*, which is designed to index multidimensional spatial data, to index spatial columns. IBM Spatial Support for Db2 for z/OS provides a grid index that is optimized for two-dimensional data on a flat projection of the Earth.

Spatial grid indexes

Indexes improve application query performance, especially when the queried table or tables contain many rows. If you create appropriate indexes that the query optimizer chooses to run your query, you can greatly reduce the number of rows to process.

IBM Spatial Support for Db2 for z/OS provides a grid index that is optimized for two dimensional data. The index is created on the X and Y dimensions of a geometry.

The following aspects of a grid index are helpful to understand:

- The generation of the index
- The use of spatial functions in a query
- How a query uses a spatial grid index

Generation of spatial grid indexes

IBM Spatial Support for Db2 for z/OS generates a spatial grid index using the minimum bounding rectangle (MBR) of a geometry.

For most geometries, the MBR is a rectangle that surrounds the geometry.

A spatial grid index divides a region into logical square grids with a fixed size that you specify when you create the index. The spatial index is constructed on a spatial column by making one or more entries for the intersections of each geometry's MBR with the grid cells. An index entry consists of the grid cell identifier, the geometry MBR, and the internal identifier of the row that contains the geometry.

You can define up to three spatial index levels (grid levels). Using several grid levels is beneficial because it allows you to optimize the index for different sizes of spatial data.

If a geometry intersects four or more grid cells, the geometry is promoted to the next larger level. In general, the larger geometrys will be indexed at the larger levels. If a geometry intersects 10 or more grid

© Copyright IBM Corp. 2007, 2013

cells at the largest grid size, a system-defined overflow index level is used. This overflow level prevents the generation of too many index entries. For best performance, define your grid sizes to avoid the use of this overflow level.

For example, if multiple grid levels exist, the indexing algorithm attempts to use the lowest grid level possible to provide the finest resolution for the indexed data. When a geometry intersects more than four grid cells at a given level, it is promoted to the next higher level, (provided that there is another level). Therefore, a spatial index that has the three grid levels of 10.0, 100.0, and 1000.0 will first intersect each geometry with the level 10.0 grid. If a geometry intersects with more than four grid cells of size 10.0, it is promoted and intersected with the level 100.0 grid. If more than four intersections result at the 100.0 level, the geometry is promoted to the 1000.0 level. If more than 10 intersections result at the 1000.0 level, the geometry is indexed in the overflow level.

Use of spatial functions in a query

A spatial index is used as a consideration in certain queries.

The Db2 optimizer considers a spatial index for use when a query contains one the following functions in its WHERE clause:

- ST_Contains
- ST_Crosses
- ST_Distance
- EnvelopesIntersect
- ST_Equals
- ST_Intersects
- ST_Overlaps
- ST_Touches
- ST_Within

In addition, the expression to the right of the predicate must be equal to 1, except when ST_Distance is the function on the left. With the ST_Distance function, the predicate must be less than the numeric value that is on the right.

How a query uses a spatial grid index

When the query optimizer chooses a spatial grid index, the query execution uses a multiple-step filter process.

The filter process includes the following steps:

1. Determine the grid cells that intersect the query window. The *query window* is the geometry that you are interested in and that you specify as the second parameter in a spatial function (see examples below).
2. Scan the index for entries that have matching grid cell identifiers.
3. Compare the geometry MBR values in the index entries with the query window and discard any values that are outside the query window.
4. Perform further analysis as appropriate. The candidate set of geometries from the previous steps might undergo further analysis to determine if they satisfy the spatial function (ST_Contains, ST_Distance, and so on). The spatial function EnvelopesIntersect omits this step and typically has the best performance.

The following examples of spatial queries have a spatial grid index on the column C.GEOMETRY:

```
SELECT name
FROM counties AS c
WHERE EnvelopesIntersect(c.geometry, -73.0, 42.0, -72.0, 43.0, 1) = 1

SELECT name
FROM counties AS c
```

```
WHERE ST_Intersects(c.geometry, :geometry2) = 1
```

In the first example, the four coordinate values define the query window. These coordinate values specify the lower-left and upper-right corners (42.0 –73.0 and 43.0 –72.0) of a rectangle.

In the second example, IBM Spatial Support for Db2 for z/OS computes the MBR of the geometry specified by the host variable :geometry2 and uses it as the query window.

When you create a spatial grid index, you should specify appropriate grid sizes for the most common query window sizes that your spatial application is likely to use. If a grid size is larger, index entries for geometries that are outside of the query window must be scanned because they reside in grid cells that intersect the query window, and these extra scans degrade performance. However, a smaller grid size might generate more index entries for each geometry and more index entries must be scanned, which also degrades query performance.

Considerations for the number of grid levels and grid sizes

Determining the appropriate grid sizes for your spatial grid indexes is the best way to tune the indexes and make your spatial queries most efficient.

Number of grid levels

You can have up to three grid levels.

For each grid level in a spatial grid index, a separate index search is performed during a spatial query. Therefore, if you have more grid levels, your query is less efficient.

If the values in the spatial column are about the same relative size, use a single grid level. However, a typical spatial column does not contain geometries of the same relative size, but geometries in a spatial column can be grouped according to size. You should correspond your grid levels with these geometry groupings.

For example, suppose you have a table of county land parcels with a spatial column that contains groupings of small urban parcels surrounded by larger rural parcels. Because the sizes of the parcels can be grouped into two groups (small urban ones and larger rural ones), you would specify two grid levels for the spatial grid index.

Grid cell sizes

The general rule is to decrease the grid sizes as much as possible to get the finest resolution while minimizing the number of index entries.

A small value should be used for the finest grid size to optimize the overall index for small geometries in the column. This avoids the overhead of evaluating geometries that are not within the search area. However, the finest grid size also produces the highest number of index entries. Consequently, the number of index entries processed at query time increases, as does the amount of storage needed for the index. These factors reduce overall performance.

By using larger grid sizes, the index can be optimized further for larger geometries. The larger grid sizes produce fewer index entries for large geometries than the finest grid size would. Consequently, storage requirements for the index are reduced, increasing overall performance.

The following figures show the effects of different grid sizes.

The first figure shows a map of land parcels, each parcel represented by a polygon geometry. The black rectangle represents a query window. Suppose you want to find all of the geometries whose MBR intersects the query window. This figure shows that 28 geometries (highlighted in pink) have an MBR that intersects the query window.

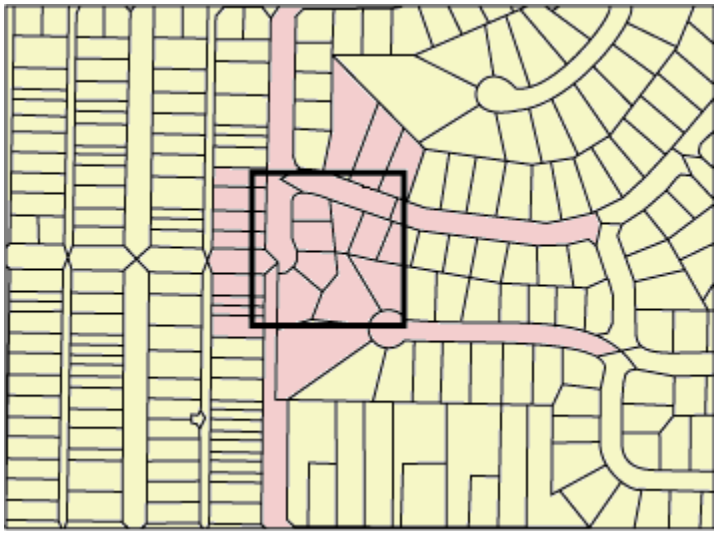


Figure 10. Land parcels in a neighborhood

The next figure shows a small grid size (25) that provides a close fit to the query window. The query returns only the 28 geometries that are highlighted, but the query must examine and discard three additional geometries whose MBRs intersect the query window.

This small grid size results in many index entries per geometry. During execution, the query accesses all index entries for these 31 geometries. The figure shows 256 grid cells that overlay the query window. However, the query execution accesses 578 index entries because many geometries are indexed with the same grid cells. For this query window, this small grid size results in an excessive number of index entries to scan.

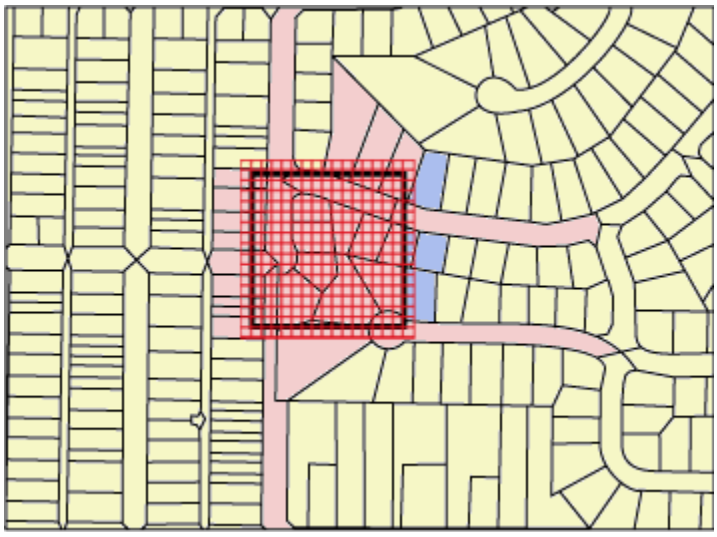


Figure 11. Small grid size (25) on land parcels

Figure 12 on page 39 shows a large grid size (400®) that encompasses a considerably larger area with many more geometries than the query window. This large grid size results in only one index entry per geometry, but the query must examine and discard 59 additional geometries whose MBRs intersect the grid cell.

During execution, the query accesses all index entries for the 28 geometries that intersect the query window, plus the index entries for the 59 additional geometries, for a total of 112 index entries. For this query window, this large grid size results in an excessive number of geometries to examine.

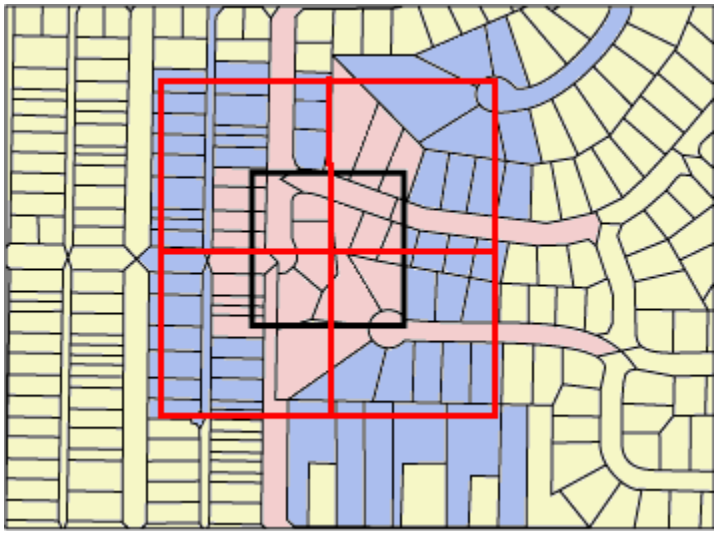


Figure 12. Large grid size (400) on land parcels

Figure 13 on page 39 shows a medium grid size (100) that provides a close fit to the query window. The query returns only the 28 geometries that are highlighted, but the query must examine and discard five additional geometries whose MBRs intersect the query window.

During execution, the query accesses all index entries for the 28 geometries that intersect the query window, plus the index entries for the 5 additional geometries, for a total of 91 index entries. For this query window, this medium grid size is the best because it results in significantly fewer index entries than the small grid size and the query examines fewer additional geometries than the large grid size.

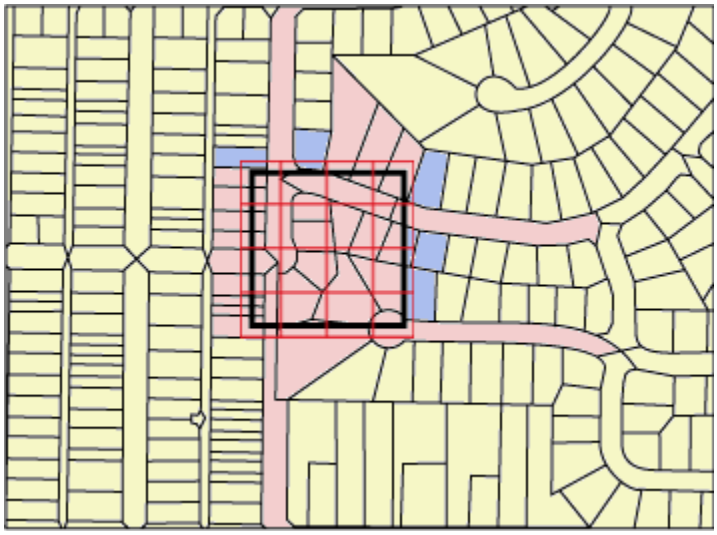


Figure 13. Medium grid size (100) on land parcels

Creating spatial grid indexes

You can use the ST_create_index stored procedure to create spatial grid indexes to define two-dimensional grid indexes on spatial columns to help optimize spatial queries.

Before you begin

Before you create a spatial grid index:

- The user ID that invokes the ST_create_index stored procedure must hold one of the following authorities or privileges:

  - SYSADM or DBADM authority on the database that contains the table where the spatial grid index will be used
  - Ownership or INDEX privilege on the table
- You must know the values that you want to specify for the fully-qualified spatial grid index name and the three grid sizes that the index will use.

About this task

You create spatial grid indexes to improve the performance of queries on spatial columns. When you create a spatial grid index, you give it the following information:

- A name
- The name of the spatial column on which it is to be defined
- The combination of the three grid sizes, which helps optimize performance by minimizing the total number of index entries and the number of index entries that need to be scanned to satisfy a query

Restriction: The Db2 LOAD utility will fail if a spatial grid index is created on the target table. Before running this utility, you must drop the spatial grid index by using the ST_drop_index stored procedure. You can create the spatial grid index again after running the LOAD utility.

Procedure

Invoke the ST_create_index stored procedure.

For information about this stored procedure, see “ST_create_index” on page 52.

Chapter 8. Analyzing and generating spatial information

After you populate spatial columns, you are ready to query them.

These topics describe the environments in which you can submit queries, provide examples of the various types of spatial functions that you can invoke in a query, and provide guidelines on using spatial functions in conjunction with spatial indexes.

Environments for performing spatial analysis

You can retrieve and analyze spatial data through various programming environments.

You can perform spatial analysis by using SQL statements and spatial functions in the following programming environments:

- Issued interactively
- Dynamically prepared and executed
- Dynamically prepared and executed by using the Db2 ODBC function calls
- Application programs in all languages supported by Db2

Examples of how spatial functions operate

This information provides an overview of spatial functions and examples of how you can use them.

IBM Spatial Support for Db2 for z/OS provides functions that perform various operations on spatial data. Generally speaking, these functions can be categorized according to the type of operation that they perform. Table 2 on page 41 lists these categories, along with examples. The text following Table 2 on page 41 shows coding for these examples.

Table 2. Spatial functions and operations

Category of function	Example of operation
Returns information about specific geometries.	Return the extent, in square miles, of the sales area of Store 10.
Makes comparisons.	Determine whether the location of a customer's home lies within the sales area of Store 10.
Derives new geometries from existing ones.	Derive the sales area of a store from its location.

Example 1: Returns information about specific geometries

In this example, the ST_Area function returns a numeric value that represents the sales area of store 10. The function will return the area in the same units as the units of the coordinate system that is being used to define the area's location.

```
SELECT db2gse.ST_Area(sales_area)
FROM   stores
WHERE  id = 10
```

© Copyright IBM Corp. 2007, 2013

Example 2: Makes comparisons

In this example, the ST_Within function compares the coordinates of the geometry representing a customer's residence with the coordinates of a geometry representing the sales area of store 10. The function's output will signify whether the residence lies within the sales area.

```
SELECT c.first_name, c.last_name, db2gse.ST_Within(c.location, s.sales_area)
FROM   customers as c. stores AS s
WHERE  s.id = 10
```

Example 3: Derives new geometries from existing ones

In this example, the ST_Buffer function derives a geometry representing a store's sales area from a geometry representing the store's location. The data type is ST_Geometry for both the store's sales area and location.

```
UPDATE stores
SET    sales_area = db2gse.ST_Polygon(db2gse.ST_Buffer(location, 10, 'KILOMETERS'))
WHERE  id = 10
```

Functions that use indexes to optimize queries

A specialized group of spatial functions, called *comparison functions*, can improve query performance by exploiting a spatial grid index.

Comparison functions compare two geometries with one another. If the results of the comparison meet certain criteria, the function returns a value of 1; if the results fail to meet the criteria, the function returns a value of 0. If the comparison cannot be performed, the function can return a null value.

For example, the function ST_Overlaps compares two geometries that have the same dimension (for example, two linestrings or two polygons). If the geometries overlap partway, and if the space covered by the overlap has the same dimension as the geometries, ST_Overlaps returns a value of 1.

The following list shows the comparison functions that can use a spatial grid index:

- EnvelopesIntersect
- ST_Contains
- ST_Crosses
- ST_Distance
- ST_Equals
- ST_Intersects
- ST_Overlaps
- ST_Touches
- ST_Within

Because of the time and memory required to execute a function, such execution can involve considerable processing. Furthermore, the more complex the geometries that are being compared, the more complex and time-intensive the comparison will be. The specialized functions listed above can complete their operations more quickly if they can use a spatial index to locate geometries. To enable such a function to use a spatial index, observe all of the following rules:

- The function must be specified in a WHERE clause. If it is specified in a SELECT, HAVING, or GROUP BY clause, a spatial index cannot be used.
- The function must be the expression on the left of the predicate.
- The operator that is used in the predicate that compares the result of the function with another expression must be an equal sign, with one exception: the ST_Distance function must use the less than operator.

- The expression on the right of the predicate must be the constant 1, except when ST_Distance is the function on the left.
- The ST_Distance function takes two geometries and, optionally, a unit as input parameters. The value of the unit must be the same type of value as each geometry. For example, if you specify the values for the geometries in degrees, you also must specify the value for the unit in degrees.
- The operation must involve a search in a spatial column on which a spatial index is defined.

For example:

```
SELECT   c.name, c.address, c.phone
FROM customers AS c, bank_branches AS b
WHERE db2gse.ST_Distance(c.location, b.location) < 10000
  and b.branch_id = 3
```

If you have detailed information about your spatial data so that the selectivity of a particular query can be estimated accurately, you can add a SELECTIVITY clause with a numeric constant in the spatial predicate function specification. Db2 uses this value as the new filter factor for determining an access path for the SELECT statement.

The following table shows the correct and incorrect ways of creating spatial queries to utilize a spatial index.

Table 3. Demonstration of how spatial functions can adhere to and violate rules for utilizing a spatial index

Queries that reference spatial functions	Rules violated

`SELECT *`
`FROM stores AS s`
`WHERE db2gse.ST_Contains(s.sales_zone,`
`db2gse.ST_Point(-121.8,37.3, 1)) = 1`
	No condition is violated in this example.

`SELECT *`
`FROM stores AS s`
`WHERE db2gse.ST_Contains(s.sales_zone,`
`db2gse.ST_Point(-121.8,37.3, 1)) = 1 SELECTIVITY 0.001`
	No condition is violated in this example.

`SELECT *`
`FROM stores AS s`
`WHERE db2gse.ST_Length(s.location) > 10`
	The spatial function ST_Length does not compare geometries and cannot utilize a spatial index.

`SELECT *`
`FROM stores AS s`
`WHERE 1=db2gse.ST_Within(s.location,:BayArea)`
	The function must be an expression on the left side of the predicate.

`SELECT *`
`FROM stores AS s`
`WHERE db2gse.ST_Contains(s.sales_zone,`
`db2gse.ST_Point(-121.8,37.3, 1)) <> 0`
	Equality comparisons must use the integer constant 1.

Table 3. Demonstration of how spatial functions can adhere to and violate rules for utilizing a spatial index (continued)

Queries that reference spatial functions	Rules violated

``
SELECT *
FROM stores AS s
WHERE db2gse.ST_Contains(db2gse.ST_Polygon
  ('polygon((10 10, 10 20, 20 20, 20 10, 10 10))', 1),
  db2gse.ST_Point(-121.8, 37.3, 1) = 1
``
	No spatial index exists on either of the arguments for the function, so no index can be utilized.

Chapter 9. Stored procedures

This section provides reference information about the stored procedures that you can use to set up spatial support and create projects that use spatial data.

When you set up IBM Spatial Support for Db2 for z/OS, you invoke these stored procedures implicitly.

Alternatively, you can invoke the stored procedures explicitly in an application program.

Before invoking most IBM Spatial Support for Db2 for z/OS stored procedures on a database, you must enable that database for spatial operations. For more information, see “Enabling spatial support for the first time” on page 11.

After Db2 for z/OS is enabled for spatial operations, you can invoke any IBM Spatial Support for Db2 for z/OS stored procedure, either implicitly or explicitly.

ST_alter_coordsys

Use this stored procedure to update a coordinate system definition in the database.

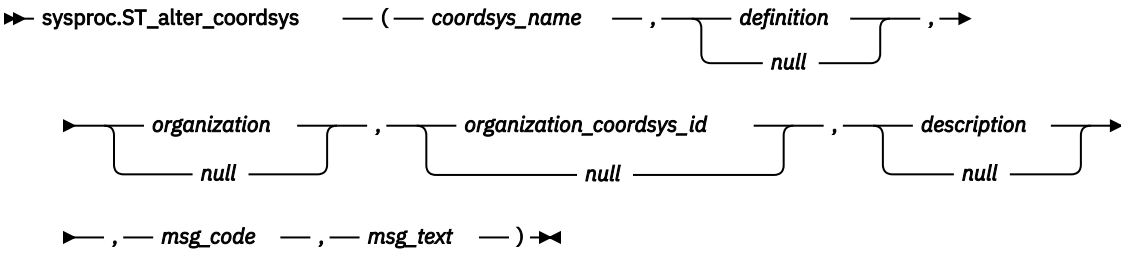
When this stored procedure is processed, information about the coordinate system is updated in the DB2GSE.ST_COORDINATE_SYSTEMS catalog view.

Attention: Use care with this stored procedure. If you use this stored procedure to change the definition of the coordinate system and you have existing spatial data that is associated with a spatial reference system that is based on this coordinate system, you might inadvertently change the spatial data. If spatial data is affected, you are responsible for ensuring that the changed spatial data is still accurate and valid.

Authorization

The user ID under which the stored procedure is invoked must have either SYSADM or DBADM authority.

Syntax



Parameter descriptions

coordsys_name
Uniquely identifies the coordinate system. You must specify a non-null value for this parameter.

Specify the *coordsys_name* value in uppercase letters.

The data type of this parameter is VARCHAR(130).

definition
Defines the coordinate system. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the definition of the coordinate system is not changed.

The data type of this parameter is VARCHAR(2048).

© Copyright IBM Corp. 2007, 2013

organization
Identifies the organization that defined the coordinate system and provided the definition for it; for example, "European Petroleum Survey Group (EPSG)." Although you must specify a value for this parameter, the value can be null.

If this parameter is null, the organization of the coordinate system is not changed. If this parameter is not null, the *organization_coordsys_id* parameter cannot be null; in this case, the combination of the *organization* and *organization_coordsys_id* parameters uniquely identifies the coordinate system.

The data type of this parameter is VARCHAR(128).

organization_coordsys_id
Specifies a numeric identifier that is assigned to this coordinate system by the entity listed in the *organization* parameter. Although you must specify a value for this parameter, the value can be null.

If this parameter is null, the *organization* parameter must also be null; in this case, the organization's coordinate system identifier is not changed. If this parameter is not null, the *organization* parameter cannot be null; in this case, the combination of the *organization* and *organization_coordsys_id* parameters uniquely identifies the coordinate system.

The data type of this parameter is INTEGER.

description
Describes the coordinate system by explaining its application. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the description information about the coordinate system is not changed.

The data type of this parameter is VARCHAR(256).

Output parameters

msg_code
Specifies the message code that is returned from the stored procedure. The value of this output parameter identifies the error, success, or warning condition that was encountered during the processing of the procedure. If this parameter value is for a success or warning condition, the procedure finished its task. If the parameter value is for an error condition, no changes to the database were performed.

The data type of this output parameter is INTEGER.

msg_text
Specifies the actual message text, associated with the message code, that is returned from the stored procedure. The message text can include additional information about the success, warning, or error condition, such as where an error was encountered.

The data type of this output parameter is VARCHAR(4096).

When the message code that is returned is 0 (zero), the message text parameter is set to null.

Example

This example shows how to use the Db2 CALL statement to invoke the ST_alter_coordsys stored procedure. This example uses a Db2 CALL statement to update a coordinate system named NORTH_AMERICAN_TEST. This CALL statement assigns a value of 1002 to the *coordsys_id* parameter:

```
call sysproc.ST_alter_coordsys('NORTH_AMERICAN_TEST',NULL,NULL,1002,NULL,?,?)
```

The two question marks at the end of this CALL statement represent the output parameters, *msg_code* and *msg_text*. The values for these output parameters are returned after the stored procedure is called.

ST_alter_srs

Use this stored procedure to update a spatial reference system definition in the database.

When this stored procedure is processed, information about the spatial reference system is updated in the DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS catalog view.

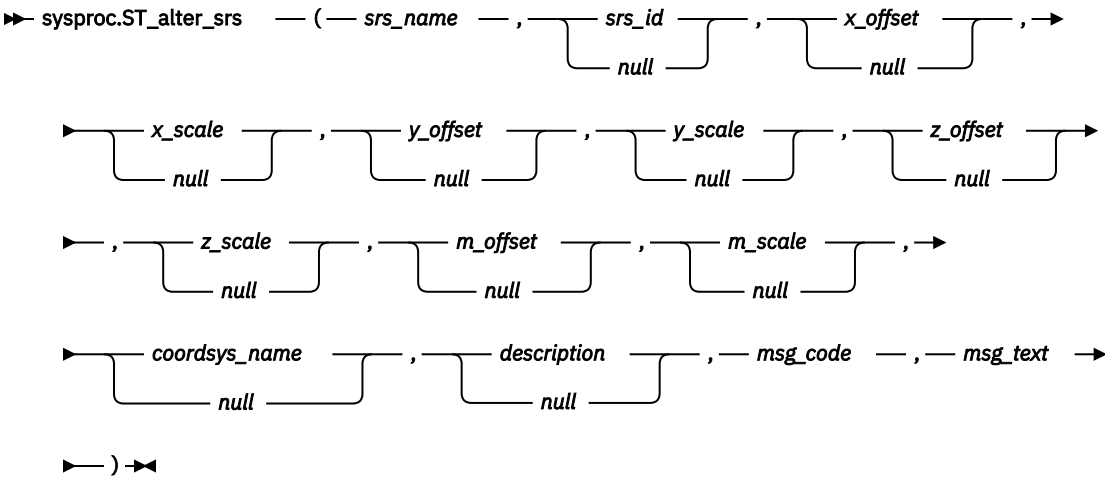
Restriction: You cannot alter a spatial reference system if a registered spatial column uses that spatial reference system.

Attention: Use care with this stored procedure. If you use this stored procedure to change offset, scale, or *coordsys_name* parameters of the spatial reference system, and if you have existing spatial data that is associated with the spatial reference system, you might inadvertently change the spatial data. If spatial data is affected, you are responsible for ensuring that the changed spatial data is still accurate and valid.

Authorization

The user ID under which the stored procedure is invoked must have either SYSADM or DBADM authority.

Syntax



Parameter descriptions

srs_name
Identifies the spatial reference system. You must specify a non-null value for this parameter.

The *srs_name* value is converted to uppercase unless you enclose it in double quotation marks.

The data type of this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

srs_id
Uniquely identifies the spatial reference system. This identifier is used as an input parameter for various spatial functions. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the numeric identifier of the spatial reference system is not changed.

The data type of this parameter is INTEGER.

x_offset
Specifies the offset for all X coordinates of geometries that are represented in this spatial reference system. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the value for this parameter in the definition of the spatial reference system is not changed.

The offset is subtracted before the scale factor *x_scale* is applied when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. (*WKT* is well-known text, and *WKB* is well-known binary.)

The data type of this parameter is DOUBLE.

x_scale
Specifies the scale factor for all X coordinates of geometries that are represented in this spatial reference system. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the value for this parameter in the definition of the spatial reference system is not changed.

The scale factor is applied (multiplication) after the offset *x_offset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation.

The data type of this parameter is DOUBLE.

y_offset
Specifies the offset for all Y coordinates of geometries that are represented in this spatial reference system. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the value for this parameter in the definition of the spatial reference system is not changed.

The offset is subtracted before the scale factor *y_scale* is applied when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation.

The data type of this parameter is DOUBLE.

y_scale
Specifies the scale factor for all Y coordinates of geometries that are represented in this spatial reference system. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the value for this parameter in the definition of the spatial reference system is not changed.

The scale factor is applied (multiplication) after the offset *y_offset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. This scale factor must be the same as *x_scale*.

The data type of this parameter is DOUBLE.

z_offset
Specifies the offset for all Z coordinates of geometries that are represented in this spatial reference system. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the value for this parameter in the definition of the spatial reference system is not changed.

The offset is subtracted before the scale factor *z_scale* is applied when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation.

The data type of this parameter is DOUBLE.

z_scale
Specifies the scale factor for all Z coordinates of geometries that are represented in this spatial reference system. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the value for this parameter in the definition of the spatial reference system is not changed.

The scale factor is applied (multiplication) after the offset *z_offset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation.

The data type of this parameter is DOUBLE.

m_offset
Specifies the offset for all M coordinates of geometries that are represented in this spatial reference system. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the value for this parameter in the definition of the spatial reference system is not changed.

The offset is subtracted before the scale factor *m_scale* is applied when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation.

The data type of this parameter is DOUBLE.

m_scale
Specifies the scale factor for all M coordinates of geometries that are represented in this spatial reference system. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the value for this parameter in the definition of the spatial reference system is not changed.

The scale factor is applied (multiplication) after the offset *m_offset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation.

The data type of this parameter is DOUBLE.

coordsys_name
Uniquely identifies the coordinate system on which this spatial reference system is based. The coordinate system must be listed in the view ST_COORDINATE_SYSTEMS. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the coordinate system that is used for this spatial reference system is not changed.

Specify the *coordsys_name* value in uppercase letters.

The data type of this parameter is VARCHAR(130).

description
Describes the spatial reference system by explaining its application. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the description information about the spatial reference system is not changed.

The data type of this parameter is VARCHAR(256).

Output parameters

msg_code
Specifies the message code that is returned from the stored procedure. The value of this output parameter identifies the error, success, or warning condition that was encountered during the processing of the procedure. If this parameter value is for a success or warning condition, the procedure finished its task. If the parameter value is for an error condition, no changes to the database were performed.

The data type of this output parameter is INTEGER.

msg_text
Specifies the actual message text, associated with the message code, that is returned from the stored procedure. The message text can include additional information about the success, warning, or error condition, such as where an error was encountered.

The data type of this output parameter is VARCHAR(4096).

When the message code that is returned is 0 (zero), the message text parameter is set to null.

Example

This example shows how to use the Db2 CALL statement to invoke the ST_alter_srs stored procedure. This example uses a Db2 CALL statement to change the *description* parameter value of a spatial reference system named SRSDEMO:

```
call sysproc.ST_alter_srs('SRSDEMO',NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,
     NULL,NULL,'SRS for GSE Demo Program: offices table',?,?)
```

The two question marks at the end of this CALL statement represent the output parameters, *msg_code* and *msg_text*. The values for these output parameters are returned after the stored procedure is called.

ST_create_coordsys

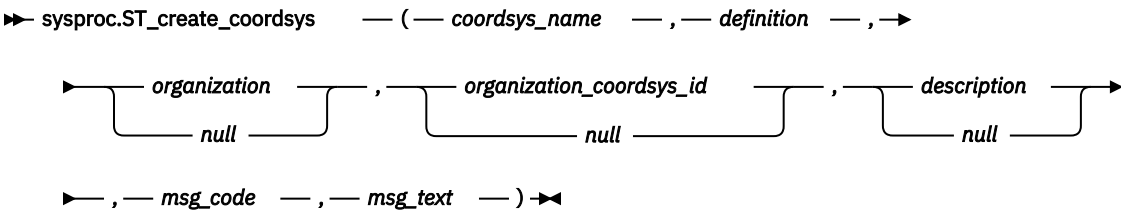
Use this stored procedure to store information in the database about a new coordinate system.

When this stored procedure is processed, information about the coordinate system is added to the DB2GSE.ST_COORDINATE_SYSTEMS catalog view.

Authorization

The user ID under which the stored procedure is invoked must have either SYSADM or DBADM authority.

Syntax



Parameter descriptions

coordsys_name
Uniquely identifies the coordinate system. You must specify a non-null value for this parameter.

Specify the *coordsys_name* value in uppercase letters.

The data type of this parameter is VARCHAR(130).

definition
Defines the coordinate system. You must specify a non-null value for this parameter. The vendor that supplies the coordinate system usually provides the information for this parameter.

The data type of this parameter is VARCHAR(2048).

organization
Identifies the organization that defined the coordinate system and provided the definition for it; for example, "European Petroleum Survey Group (EPSG)." Although you must specify a value for this parameter, the value can be null.

If this parameter is null, the *organization_coordsys_id* parameter must also be null. If this parameter is not null, the *organization_coordsys_id* parameter cannot be null; in this case, the combination of the *organization* and *organization_coordsys_id* parameters uniquely identifies the coordinate system.

The data type of this parameter is VARCHAR(128).

organization_coordsys_id
Specifies a numeric identifier. The entity that is specified in the *organization* parameter assigns this value. This value is not necessarily unique across all coordinate systems. Although you must specify a value for this parameter, the value can be null.

If this parameter is null, the *organization* parameter must also be null. If this parameter is not null, the *organization* parameter cannot be null; in this case, the combination of the *organization* and *organization_coordsys_id* parameters uniquely identifies the coordinate system.

The data type of this parameter is INTEGER.

description
Describes the coordinate system by explaining its application. Although you must specify a value for this parameter, the value can be null. If this parameter is null, no description information about the coordinate system is recorded.

The data type of this parameter is VARCHAR(256).

Output parameters

msg_code
Specifies the message code that is returned from the stored procedure. The value of this output parameter identifies the error, success, or warning condition that was encountered during the processing of the procedure. If this parameter value is for a success or warning condition, the procedure finished its task. If the parameter value is for an error condition, no changes to the database were performed.

The data type of this output parameter is INTEGER.

msg_text
Specifies the actual message text, associated with the message code, that is returned from the stored procedure. The message text can include additional information about the success, warning, or error condition, such as where an error was encountered.

The data type of this output parameter is VARCHAR(4096).

When the message code that is returned is 0 (zero), the message text parameter is set to null.

Example

This example shows how to use the Db2 CALL statement to invoke the ST_create_coordsys stored procedure. This example uses a Db2 CALL statement to create a coordinate system with the following parameter values:

- *coordsys_name* parameter: NORTH_AMERICAN_TEST
- *definition* parameter:

```
GEOGCS["GCS_North_American_1983",
DATUM["D_North_American_1983",
SPHEROID["GRS_1980",6378137.0,298.257222101]],
PRIMEM["Greenwich",0.0],
UNIT["Degree",0.0174532925199433]]
```

- *organization* parameter: EPSG
- *organization_coordsys_id* parameter: 1001
- *description* parameter: Test Coordinate Systems

```
call sysproc.ST_create_coordsys('NORTH_AMERICAN_TEST',
     'GEOGCS["GCS_North_American_1983",DATUM["D_North_American_1983",
     SPHEROID["GRS_1980",6378137.0,298.257222101]],
     PRIMEM["Greenwich",0.0],UNIT["Degree",
     0.0174532925199433]]','EPSG',1001,'Test Coordinate Systems',?,?)
```

The two question marks at the end of this CALL statement represent the output parameters, *msg_code* and *msg_text*. The values for these output parameters are returned after the stored procedure is called.

ST_create_index

Use this stored procedure a create spatial grid index on a spatial column to help optimize spatial queries.

The column that you want to index must be a spatial data type that adheres to the following guidelines:

- The column name cannot be qualified
- If the column is not the ST_Point data type, then the LOB table space that stored the corresponding BLOB column data must exist. Also, if the table space that contains the base table is LOGGED, then the LOB table space must be created with LOGGED, too.
- The column cannot have any field procedure or security label defined.
- Only one spatial index is allowed on a column with a spatial data type.

Determining the correct grid size for a spatial grid index takes experience. Set the grid size in relation to the approximate size of the object that you are indexing. A grid size that is too small or too large can decrease performance. For example, a grid size that is set too small can affect the key to object ratio during an index search. If a grid size is set too large, the initial index search returns a small number of candidates and can decrease the performance during the final table scan.

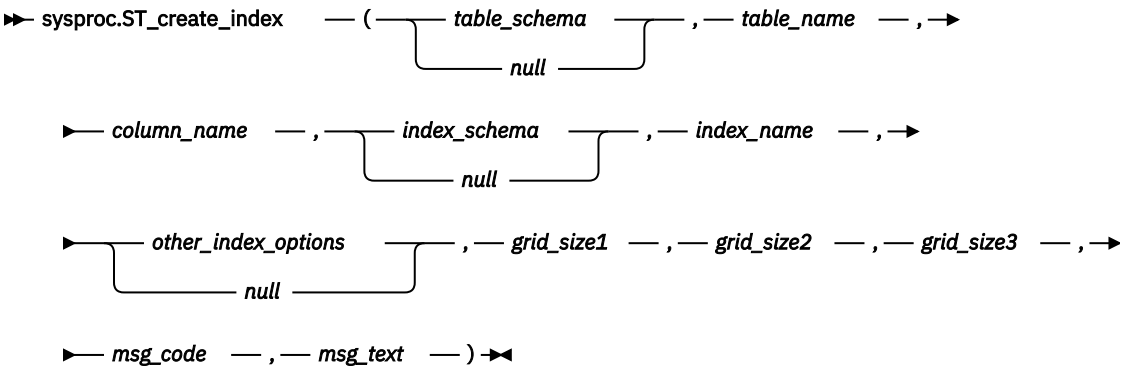
Important: Because a spatial index cannot be rebuilt, create the spatial index with the COPY YES option specified. When you specify this option, Db2 takes an image copy of the index along with an image copy of the table. Also, you cannot alter the spatial index to change any of the options that you specified when you invoked the ST_create_index stored procedure.

Authorization

The user ID under which the stored procedure is invoked must have one of the following authorities or privileges:

- SYSADM or DBADM authority on the database that contains the table where the spatial grid index will be used
- Ownership or INDEX privilege on the table

Syntax



Parameter descriptions

table_schema
Identifies the schema to which the table that is specified in the *table_name* parameter belongs. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the value in the CURRENT SCHEMA special register is used as the schema name for the table or view.

The data type of this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

table_name
Identifies the unqualified name of the table on which the index is to be defined. You must specify a non-null value for this parameter.

The data type of this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

column_name
Identifies the column that contains the spatial data type for the index. You must specify a non-null value for this parameter.

The data type of this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

index_schema
Identifies the schema to which the index that is specified in the *index_name* parameter belongs. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the value in the CURRENT SCHEMA special register is used as the schema name for the table or view.

The *index_schema* value is converted to uppercase unless you enclose it in double quotation marks.

The data type of this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

index_name
Identifies the name of the index that is to be created. You must specify a non-null value for this parameter.

The *index_name* value is converted to uppercase unless you enclose it in double quotation marks.

The data type of this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

other_index_options
Identifies one or more valid options from the CREATE INDEX statement. For example, you can specify FREEPAGE, PCTFREE, and so on. This parameter is nullable. The following options are not valid for a spatial index:

- CLUSTER
- PARTITIONED
- PARTITIONED BY
- DEFER YES

The data type of this parameter is VARCHAR(1024).

grid_size1
A number that indicates the granularity of the smallest index grid. You must specify a non-null value for this parameter.

The data type of this parameter is DOUBLE.

grid_size2
A number that indicates either that there is not a second grid for this index, or the granularity of the second index grid. You must specify a non-null value for this parameter. Specify 0, if there is not a second grid. If you want a second grid for the index, then you must specify a grid size that is larger than the value in *grid_size1*. This value is commonly two to five times larger than the prior grid size.

The data type of this parameter is DOUBLE.

grid_size3
A number that indicates either that there is not a third grid for this index, or the granularity of the third index grid. You must specify a non-null value for this parameter. Specify 0, if there is not a third grid. If you want a third grid for the index, then you must specify a grid size that is larger than the value in *grid_size2*. This value is commonly two to five times larger than the prior grid size.

The data type of this parameter is DOUBLE.

Output parameters

msg_code
Specifies the message code that is returned from the stored procedure. The value of this output parameter identifies the error, success, or warning condition that was encountered during the processing of the procedure. If this parameter value is for a success or warning condition, the procedure finished its task. If the parameter value is for an error condition, no changes to the database were performed.

The data type of this output parameter is INTEGER.

msg_text
Specifies the actual message text, associated with the message code, that is returned from the stored procedure. The message text can include additional information about the success, warning, or error condition, such as where an error was encountered.

The data type of this output parameter is VARCHAR(4096).

When the message code that is returned is 0 (zero), the message text parameter is set to null.

Example

This example shows how to use a Db2 CALL statement to invoke the ST_create_index stored procedure. This example uses a Db2 CALL statement to create a spatial index named INDEXDEMO on column LOCATION in table OFFICE with the following grid sizes values:

- gridSize1: 10.0
- gridSize2: 20.0
- gridSize3: 35.0

```
call sysproc.ST_create_index(NULL,'OFFICE', 'LOCATION',NULL,'INDEXDEMO',
NULL,10.0,20.0,35.0,?,?)
```

The two question marks at the end of this CALL statement represent the output parameters, *msg_code* and *msg_text*. The values for these output parameters are returned after the stored procedure is called.

ST_create_srs

Use this stored procedure to create a spatial reference system. The ST_create_srs stored procedure takes the conversion factors (offsets and scale factors) as input parameters.

A spatial reference system is defined by the coordinate system, the precision, and the extents of coordinates that are represented in this spatial reference system. The extents are the minimum and maximum possible coordinate values for the X, Y, Z, and M coordinates.

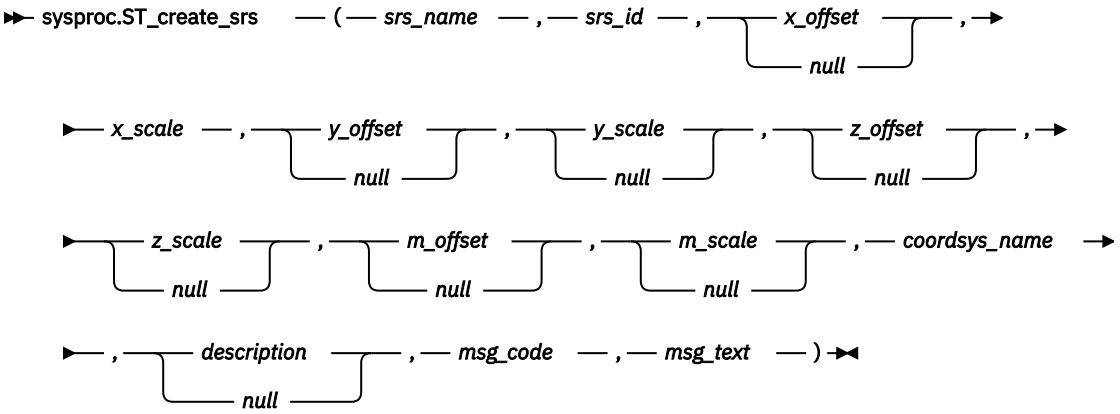
This stored procedure has two variations. This variation takes the conversion factors (offsets and scale factors) as input parameters. The second variation, the ST_create_srs_2 stored procedure, takes the extents and the precision as input parameters and calculates the conversion factors internally.

Authorization

The user ID under which the stored procedure is invoked must have the following authorities or privileges:

- SYSADM or DBADM authority
- INSERT and SELECT privileges on the catalog table or view

Syntax



Parameter descriptions

srs_name
Identifies the spatial reference system. You must specify a non-null value for this parameter.

The *srs_name* value is converted to uppercase unless you enclose it in double quotation marks.

The data type of this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

srs_id
Uniquely identifies the spatial reference system. This numeric identifier is used as an input parameter for various spatial functions. You must specify a non-null value for this parameter.

The data type of this parameter is INTEGER.

x_offset
Specifies the offset for all X coordinates of geometries that are represented in this spatial reference system. The offset is subtracted before the scale factor *x_scale* is applied when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. (*WKT* is well-known text, and *WKB* is well-known binary.) Although you must specify a value for this parameter, the value can be null. If this parameter is null, a value of 0 (zero) is used.

The data type of this parameter is DOUBLE.

x_scale
Specifies the scale factor for all X coordinates of geometries that are represented in this spatial reference system. The scale factor is applied (multiplication) after the offset *x_offset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. You must specify a non-null value for this parameter.

The data type of this parameter is DOUBLE.

y_offset
Specifies the offset for all Y coordinates of geometries that are represented in this spatial reference system. The offset is subtracted before the scale factor *y_scale* is applied when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. Although you must specify a value for this parameter, the value can be null. If this parameter is the null value, a value of 0 (zero) is used.

The data type of this parameter is DOUBLE.

y_scale
Specifies the scale factor for all Y coordinates of geometries that are represented in this spatial reference system. The scale factor is applied (multiplication) after the offset *y_offset* is subtracted

when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the value of the *x_scale* parameter is used. If you specify a value other than null for this parameter, the value that you specify must match the value of the *x_scale* parameter.

The data type of this parameter is DOUBLE.

z_offset
Specifies the offset for all Z coordinates of geometries that are represented in this spatial reference system. The offset is subtracted before the scale factor *z_scale* is applied when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. Although you must specify a value for this parameter, the value can be null. If this parameter is null, a value of 0 (zero) is used.

The data type of this parameter is DOUBLE.

z_scale
Specifies the scale factor for all Z coordinates of geometries that are represented in this spatial reference system. The scale factor is applied (multiplication) after the offset *z_offset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. Although you must specify a value for this parameter, the value can be null. If this parameter is null, a value of 1 is used.

The data type of this parameter is DOUBLE.

m_offset
Specifies the offset for all M coordinates of geometries that are represented in this spatial reference system. The offset is subtracted before the scale factor *m_scale* is applied when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. Although you must specify a value for this parameter, the value can be null. If this parameter is null, a value of 0 (zero) is used.

The data type of this parameter is DOUBLE.

m_scale
Specifies the scale factor for all M coordinates of geometries that are represented in this spatial reference system. The scale factor is applied (multiplication) after the offset *m_offset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. Although you must specify a value for this parameter, the value can be null. If this parameter is null, a value of 1 is used.

The data type of this parameter is DOUBLE.

coordsys_name
Uniquely identifies the coordinate system on which this spatial reference system is based. The coordinate system must be listed in the view ST_COORDINATE_SYSTEMS. You must supply a non-null value for this parameter.

Specify the *coordsys_name* value in uppercase letters.

The data type of this parameter is VARCHAR(130).

description
Describes the spatial reference system by explaining the application's purpose. Although you must specify a value for this parameter, the value can be null. If this parameter is null, no description information is recorded.

The data type of this parameter is VARCHAR(256).

Output parameters

msg_code
Specifies the message code that is returned from the stored procedure. The value of this output parameter identifies the error, success, or warning condition that was encountered during the

processing of the procedure. If this parameter value is for a success or warning condition, the procedure finished its task. If the parameter value is for an error condition, no changes to the database were performed.

The data type of this output parameter is INTEGER.

msg_text
Specifies the actual message text, associated with the message code, that is returned from the stored procedure. The message text can include additional information about the success, warning, or error condition, such as where an error was encountered.

The data type of this output parameter is VARCHAR(4096).

When the message code that is returned is 0 (zero), the message text parameter is set to null.

Example

This example shows how to use a Db2 CALL statement to invoke the ST_create_srs stored procedure. This example uses a Db2 CALL statement to create a spatial reference system named SRSDEMO with the following parameter values:

- *srs_id*: 1000000
- *x_offset*: -180
- *x_scale*: 1000000
- *y_offset*: -90
- *y_scale*: 1000000

```
call sysproc.ST_create_srs('SRSDEMO',1000000,
                     -180,1000000, -90, 1000000,
                     0, 1, 0, 1,'NORTH_AMERICAN',
                     'SRS for GSE Demo Program: customer table',?,?)
```

The two question marks at the end of this CALL statement represent the output parameters, *msg_code* and *msg_text*. The values for these output parameters are returned after the stored procedure is called.

ST_create_srs_2

Use this stored procedure to create a spatial reference system. The ST_create_srs_2 stored procedure takes the extents and the precision as input parameters and calculates the conversion factors internally.

A spatial reference system is defined by the coordinate system, the precision, and the extents of coordinates that are represented in this spatial reference system. The extents are the minimum and maximum possible coordinate values for the X, Y, Z, and M coordinates.

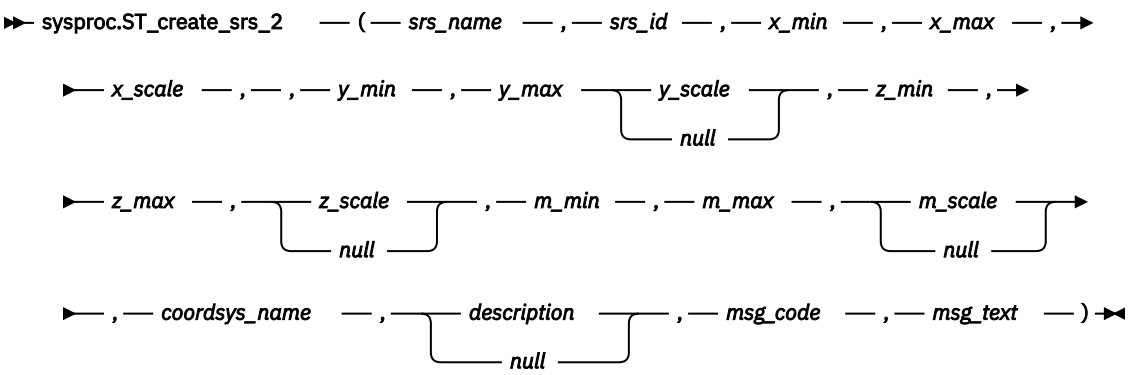
This stored procedure has two variations. This variation takes the extents and the precision as input parameters and calculates the conversion factors internally. The other variation, the ST_create_srs stored procedure, takes the conversion factors (offsets and scale factors) as input parameters.

Authorization

The user ID under which the stored procedure is invoked must have the following authorities or privileges:

- SYSADM or DBADM authority
- INSERT and SELECT privileges on the catalog table or view

Syntax



Parameter descriptions

srs_name
Identifies the spatial reference system. You must specify a non-null value for this parameter.

The *srs_name* value is converted to uppercase unless you enclose it in double quotation marks.

The data type of this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

srs_id
Uniquely identifies the spatial reference system. This numeric identifier is used as an input parameter for various spatial functions. You must specify a non-null value for this parameter.

The data type of this parameter is INTEGER.

x_min
Specifies the minimum possible X coordinate value for all geometries that use this spatial reference system. You must specify a non-null value for this parameter.

The data type of this parameter is DOUBLE.

x_max
Specifies the maximum possible X coordinate value for all geometries that use this spatial reference system. You must specify a non-null value for this parameter.

Depending on the value of *x_scale*, the value that is shown in the view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS might be larger than the value that is specified here. The value from the view is correct.

The data type of this parameter is DOUBLE.

x_scale
Specifies the scale factor for all X coordinates of geometries that are represented in this spatial reference system. The scale factor is applied (multiplication) after the offset *x_offset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. The calculation of the offset *x_offset* is based on the *x_min* value. You must supply a non-null value for this parameter.

If both the *x_scale* and *y_scale* parameters are specified, the values must match.

The data type of this parameter is DOUBLE.

y_min
Specifies the minimum possible Y coordinate value for all geometries that use this spatial reference system. You must supply a non-null value for this parameter.

The data type of this parameter is DOUBLE.

y_max
Specifies the maximum possible Y coordinate value for all geometries that use this spatial reference system. You must supply a non-null value for this parameter.

Depending on the value of *y_scale*, the value that is shown in the view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS might be larger than the value that is specified here. The value from the view is correct.

The data type of this parameter is DOUBLE.

y_scale
Specifies the scale factor for all Y coordinates of geometries that are represented in this spatial reference system. The scale factor is applied (multiplication) after the offset *y_offset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. The calculation of the offset *y_offset* is based on the *y_min* value. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the value of the *x_scale* parameter is used. If both the *y_scale* and *x_scale* parameters are specified, the values must match.

The data type of this parameter is DOUBLE.

z_min
Specifies the minimum possible Z coordinate value for all geometries that use this spatial reference system. You must specify a non-null value for this parameter.

The data type of this parameter is DOUBLE.

z_max
Specifies the maximum possible Z coordinate value for all geometries that use this spatial reference system. You must specify a non-null value for this parameter.

Depending on the value of *z_scale*, the value that is shown in the view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS might be larger than the value that is specified here. The value from the view is correct.

The data type of this parameter is DOUBLE.

z_scale
Specifies the scale factor for all Z coordinates of geometries that are represented in this spatial reference system. The scale factor is applied (multiplication) after the offset *z_offset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. The calculation of the offset *z_offset* is based on the *z_min* value. Although you must specify a value for this parameter, the value can be null. If this parameter is null, a value of 1 is used.

The data type of this parameter is DOUBLE.

m_min
Specifies the minimum possible M coordinate value for all geometries that use this spatial reference system. You must specify a non-null value for this parameter.

The data type of this parameter is DOUBLE.

m_max
Specifies the maximum possible M coordinate value for all geometries that use this spatial reference system. You must specify a non-null value for this parameter.

Depending on the value of *m_scale*, the value that is shown in the view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS might be larger than the value that is specified here. The value from the view is correct.

The data type of this parameter is DOUBLE.

m_scale
Specifies the scale factor for all M coordinates of geometries that are represented in this spatial reference system. The scale factor is applied (multiplication) after the offset *m_offset* is subtracted

when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. The calculation of the offset *m_offset* is based on the *m_min* value. Although you must specify a value for this parameter, the value can be null. If this parameter is null, a value of 1 is used.

The data type of this parameter is DOUBLE.

coordsys_name
Uniquely identifies the coordinate system on which this spatial reference system is based. The coordinate system must be listed in the view ST_COORDINATE_SYSTEMS. You must specify a non-null value for this parameter.

Specify the *coordsys_name* value in uppercase letters.

The data type of this parameter is VARCHAR(130).

description
Describes the spatial reference system by explaining the application's purpose. Although you must specify a value for this parameter, the value can be null. If this parameter is null, no description information is recorded.

The data type of this parameter is VARCHAR(256).

Output parameters

msg_code
Specifies the message code that is returned from the stored procedure. The value of this output parameter identifies the error, success, or warning condition that was encountered during the processing of the procedure. If this parameter value is for a success or warning condition, the procedure finished its task. If the parameter value is for an error condition, no changes to the database were performed.

The data type of this output parameter is INTEGER.

msg_text
Specifies the actual message text, associated with the message code, that is returned from the stored procedure. The message text can include additional information about the success, warning, or error condition, such as where an error was encountered.

The data type of this output parameter is VARCHAR(4096).

When the message code that is returned is 0 (zero), the message text parameter is set to null.

Example

This example shows how to use a Db2 CALL statement to invoke the ST_create_srs_2 stored procedure. This example uses a Db2 CALL statement to create a spatial reference system named SRSDEMO with the following parameter values:

- srs_id: 1000000
- x_offset: -180
- x_scale: 1000000
- y_offset: -90
- y_scale: 1000000

```
call sysproc.ST_create_srs_2('SRSDEMO',1000000, -180,1000000, -90, 1000000,
0, 1, 0, 1,'NORTH_AMERICAN', 'SRS for GSE Demo Program: customer table',?,?)
```

The two question marks at the end of this CALL statement represent the output parameters, *msg_code* and *msg_text*. The values for these output parameters are returned after the stored procedure is called.

ST_drop_coordsys

Use this stored procedure to delete information about a coordinate system from the database.

When this stored procedure is processed, information about the coordinate system is removed from the DB2GSE.ST_COORDINATE_SYSTEMS catalog view.

Restriction:

You cannot drop a coordinate system on which a spatial reference system is based.

Authorization

The user ID under which the stored procedure is invoked must have either SYSADM or DBADM authority.

Syntax



Parameter descriptions

coordsys_name
Uniquely identifies the coordinate system. You must specify a non-null value for this parameter.

Specify the *coordsys_name* value in uppercase letters.

The data type of this parameter is VARCHAR(130).

Output parameters

msg_code
Specifies the message code that is returned from the stored procedure. The value of this output parameter identifies the error, success, or warning condition that was encountered during the processing of the procedure. If this parameter value is for a success or warning condition, the procedure finished its task. If the parameter value is for an error condition, no changes to the database were performed.

The data type of this output parameter is INTEGER.

msg_text
Specifies the actual message text, associated with the message code, that is returned from the stored procedure. The message text can include additional information about the success, warning, or error condition, such as where an error was encountered.

The data type of this output parameter is VARCHAR(4096).

When the message code that is returned is 0 (zero), the message text parameter is set to null.

Example

This example shows how to use a Db2 CALL statement to invoke the ST_drop_coordsys stored procedure. This example uses a Db2 CALL statement to delete a coordinate system named NORTH_AMERICAN_TEST from the database:

```
call sysproc.ST_drop_coordsys('NORTH_AMERICAN_TEST',?,?)
```

The two question marks at the end of this CALL statement represent the output parameters, *msg_code* and *msg_text*. The values for these output parameters are returned after the stored procedure is called.

ST_drop_index

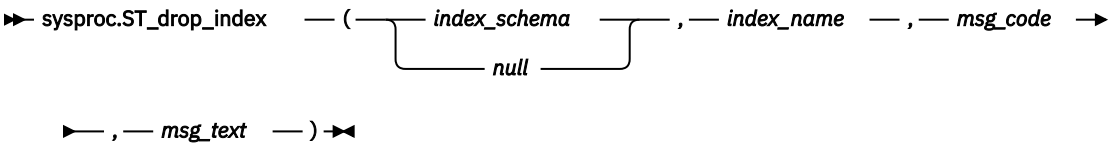
Use this stored procedure to drop a spatial index.

Authorization

The user ID under which the stored procedure is invoked must have one of the following authorities or privileges:

- SYSADM or DBADM authority on the database that contains the table where the spatial grid index will be used
- Ownership or INDEX privilege on the table

Syntax



Parameters

index_schema
Identifies the schema to which the index that is specified in the *index_name* parameter belongs. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the value in the CURRENT SCHEMA special register is used as the schema name for the index.

The *index_schema* value is converted to uppercase unless you enclose it in double quotation marks.

The data type of this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

index_name
Identifies the name of the index that is to be dropped. You must specify a non-null value for this parameter.

The *index_name* value is converted to uppercase unless you enclose it in double quotation marks.

The data type of this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

Output parameters

msg_code
Specifies the message code that is returned from the stored procedure. The value of this output parameter identifies the error, success, or warning condition that was encountered during the processing of the procedure. If this parameter value is for a success or warning condition, the procedure finished its task. If the parameter value is for an error condition, no changes to the database were performed.

The data type of this output parameter is INTEGER.

msg_text
Specifies the actual message text, associated with the message code, that is returned from the stored procedure. The message text can include additional information about the success, warning, or error condition, such as where an error was encountered.

The data type of this output parameter is VARCHAR(4096).

When the message code that is returned is 0 (zero), the message text parameter is set to null.

Example

This example shows how to use a Db2 CALL statement to invoke the ST_drop_index stored procedure. This example uses a Db2 CALL statement to drop the spatial index named INDEXDEMO:

```
call sysproc.ST_drop_index(NULL,'INDEXDEMO',?,?)
```

The two question marks at the end of this CALL statement represent the output parameters, *msg_code* and *msg_text*. The values for these output parameters are returned after the stored procedure is called.

ST_drop_srs

Use this stored procedure to drop a spatial reference system.

When this stored procedure is processed, information about the spatial reference system is removed from the DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS catalog view.

Restriction: You cannot drop a spatial reference system if a spatial column that uses that spatial reference system is registered.

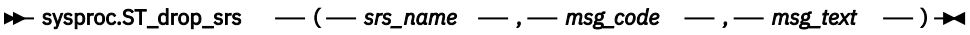
Important:

Use care when you use this stored procedure. If you use this stored procedure to drop a spatial reference system, and if any spatial data is associated with that spatial reference system, you can no longer perform spatial operations on the spatial data.

Authorization

The user ID under which the stored procedure is invoked must have either SYSADM or DBADM authority.

Syntax



Parameter descriptions

srs_name
Identifies the spatial reference system. You must specify a non-null value for this parameter.

The *srs_name* value is converted to uppercase unless you enclose it in double quotation marks.

The data type of this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

Output parameters

msg_code
Specifies the message code that is returned from the stored procedure. The value of this output parameter identifies the error, success, or warning condition that was encountered during the processing of the procedure. If this parameter value is for a success or warning condition, the procedure finished its task. If the parameter value is for an error condition, no changes to the database were performed.

The data type of this output parameter is INTEGER.

msg_text
Specifies the actual message text, associated with the message code, that is returned from the stored procedure. The message text can include additional information about the success, warning, or error condition, such as where an error was encountered.

The data type of this output parameter is VARCHAR(4096).

When the message code that is returned is 0 (zero), the message text parameter is set to null.

Example

This example shows how to use the Db2 CALL statement to invoke the ST_drop_srs stored procedure. This example uses a Db2 CALL statement to delete a spatial reference system named SRSDEMO:

```
call sysproc.ST_drop_srs('SRSDEMO',?,?)
```

The two question marks at the end of this CALL statement represent the output parameters, *msg_code* and *msg_text*. The values for these output parameters are returned after the stored procedure is called.

ST_export_shape

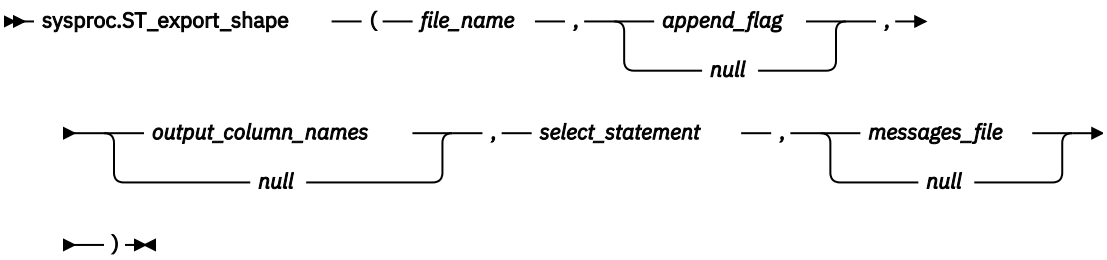
Use the ST_export_shape stored procedure to export a spatial column and its associated table to a shape file.

Authorization

The user ID under which this stored procedure is invoked must have the necessary privileges to successfully execute the SELECT statement from which the data is to be exported.

The stored procedure, which runs as a process that is owned by the Db2 instance owner, must have the necessary privileges on the server machine to create or write to the shape files.

Syntax



Notes:

Parameter descriptions

file_name

Specifies the full path name of a shape file to which the specified data is to be exported. You must specify a non-null value for this parameter.

You can use the ST_export_shape stored procedure to export a new file or to export to an existing file by appending the exported data to it:

- If you are exporting to a new file, you can specify the optional file extension as .shp or .SHP. If you specify .shp or .SHP for the file extension, IBM Spatial Support for Db2 for z/OS creates the file with the specified *file_name* value. If you do not specify the optional file extension, IBM Spatial Support for Db2 for z/OS creates the file that has the name of the *file_name* value that you specify and with an extension of .shp.
- If you are exporting data by appending the data to an existing file, IBM Spatial Support for Db2 for z/OS first looks for an exact match of the name that you specify for the *file_name* parameter. If IBM Spatial Support for Db2 for z/OS does not find an exact match, it looks first for a file with the .shp extension, and then for a file with the .SHP extension.

If the value of the *append_flag* parameter indicates that you are not appending to an existing file, but the file that you name in the *file_name* parameter already exists, IBM Spatial Support for Db2 for z/OS returns an error and does not overwrite the file.

See Usage notes for a list of files that are written on the server machine. The stored procedure, which runs as a process that is owned by the Db2 instance owner, must have the necessary privileges on the server machine to create or write to the files.

The data type of this parameter is VARCHAR(256).

append_flag
Indicates whether the data that is to be exported is to be appended to an existing shape file. Although you must specify a value for this parameter, the value can be null. Indicate whether you want to append to an existing shape file as follows:

- If you want to append data to an existing shape file, specify any value greater than 0 (zero). In this case, the file structure must match the exported data; otherwise, an error is returned.
- If you want to export to a new file, specify 0 (zero) or null. In this case, IBM Spatial Support for Db2 for z/OS does not overwrite any existing files.

The data type of this parameter is SMALLINT.

output_column_names
Specifies one or more column names (separated by commas) that are to be used for non-spatial columns in the output dBASE file. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the names that are derived from the SELECT statement are used.

If you specify this parameter but do not enclose column names in double quotation marks, the column names are converted to uppercase. The number of specified columns must match the number of columns that are returned from the SELECT statement, as specified in the *select_statement* parameter, excluding the spatial column.

The data type of this parameter is VARCHAR(32K).

select_statement
Specifies the subselect that returns the data that is to be exported. The subselect must reference exactly one spatial column and any number of attribute columns. You must specify a non-null value for this parameter.

The data type of this parameter is VARCHAR(32K).

messages_file
Specifies the full path name of the file (on the server machine) that is to contain messages about the export operation. Although you must specify a value for this parameter, the value can be null. If this parameter is null, no file for IBM Spatial Support for Db2 for z/OS messages is created.

The messages that are sent to this messages file can be:

- Informational messages, such as a summary of the export operation
- Error messages for data that could not be exported, for example because of different coordinate systems

The stored procedure, which runs as a process that is owned by the Db2 instance owner, must have the necessary privileges on the server to create the file.

The data type of this parameter is VARCHAR(256).

Output parameters

msg_code
Specifies the message code that is returned from the stored procedure. The value of this output parameter identifies the error, success, or warning condition that was encountered during the processing of the procedure. If this parameter value is for a success or warning condition, the procedure finished its task. If the parameter value is for an error condition, no changes to the database were performed.

The data type of this output parameter is INTEGER.

msg_text
Specifies the actual message text, associated with the message code, that is returned from the stored procedure. The message text can include additional information about the success, warning, or error condition, such as where an error was encountered.

The data type of this output parameter is VARCHAR(1024).

Usage notes

You can export only one spatial column at a time.

The ST_export_shape stored procedure creates or writes to the following four files:

- The main shape file (.shp extension).
- The shape index file (.shx extension).
- A dBASE file that contains data for non-spatial columns (.dbf extension). This file is created only if attribute columns need to be exported. A dBASE file can store only columns with names that are 11 characters in length or less; otherwise, the column name will be truncated. If you have a column name that is longer than 11 characters, you can bypass this limitation by creating a view on the column with a name that is 11 characters in length or less.
- A projection file that specifies the coordinate system that is associated with the spatial data, if the coordinate system is not equal to "UNSPECIFIED" (.prj extension). The coordinate system is obtained from the first spatial record. An error occurs if subsequent records have different coordinate systems.

The following table describes how Db2 data types are stored in dBASE attribute files. All other Db2 data types are not supported.

Table 4. Storage of Db2 data types in attribute files

SQL type	.dbf type	.dbf length	.dbf decimals	Comments
SMALLINT	N	6	0	
INTEGER	N	11	0	
BIGINT	N	20	0	
DECIMAL	N	precision+2	scale	
REAL FLOAT(1) through FLOAT(24)	F	14	6	
DOUBLE FLOAT(25) through FLOAT(53)	F	19	9	
CHARACTER, VARCHAR, LONG VARCHAR, and DATALINK	C	*len*	0	length ≤ 255
DATE	D	8	0	
TIME	C	8	0	
TIMESTAMP	C	26	0	

All synonyms for data types and distinct types that are based on the types listed in the preceding table are supported.

Example

This example shows how to use the Db2 command line processor to invoke the ST_export_shape stored procedure. This example uses a Db2 CALL command to export all rows from the CUSTOMERS table to a shape file that is to be created and named `/tmp/export_file`:

```
call sysproc.ST_export_shape('/tmp/export_file',0,NULL,
     'select * from customers','/tmp/export_msg',?,?)
```

The two question marks at the end of this CALL command represent the output parameters, *msg_code* and *msg_text*. The values for these output parameters are displayed after the stored procedure runs.

ST_import_shape

Use the ST_import_shape stored procedure to import a shape file to a database that is enabled for spatial operations.

This stored procedure can operate in either of two ways, based on the *create_table_flag* parameter:

- IBM Spatial Support for Db2 for z/OS can create a table that has a spatial column and attribute columns, and it can then load the table's columns with the file's data.
- Otherwise, the shape and attribute data can be loaded into an existing table that has a spatial column and attribute columns that match the file's data.

Important: Using a message file is optional; however, consider specifying a message file so that any errors and informational messages are written to the message file. The import process continues even if an error occurs on a row. If many errors occur, the import process will be much slower.

The input files must reside on the HFS file under the z/OS UNIX environment, so the binder and the user must have read access to the given directory. Also, the message file will be generated on a valid HFS directory under the z/OS UNIX environment if specified. Therefore, the binder and the user must have write access to the given directory.

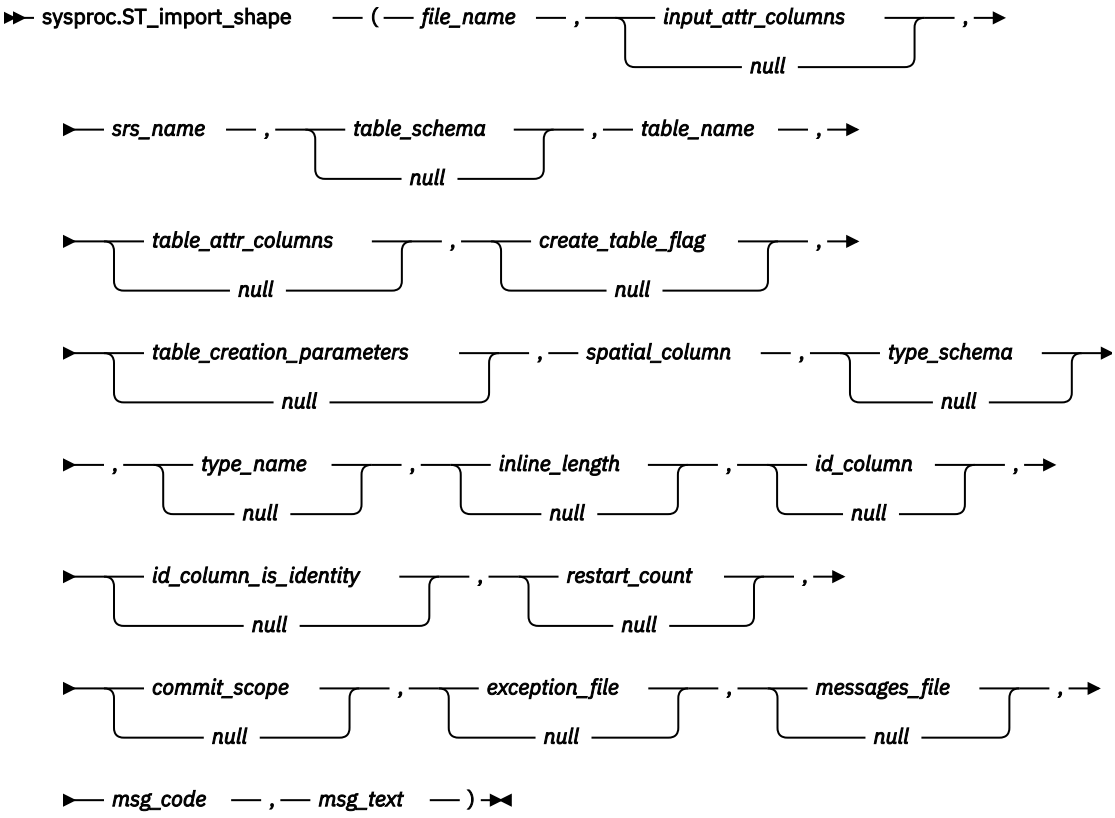
IBM Spatial Support for Db2 for z/OS does not support the inline_length parameter and the exception_file parameter for this stored procedure. If you specify either of these parameters, the parameter will be ignored.

Authorization

You must have the necessary privileges on the server machine for reading the input files and optionally writing error files. Additional authorization requirements vary based on whether you are importing into an existing table or into a new table.

- When importing into an existing table, the user ID under which this stored procedure is invoked must hold INSERT authority.
- When importing into a new table, the user ID under which this stored procedure is invoked must hold CREATE TABLE authority.

Syntax



Parameter descriptions

file_name

Specifies the full path name of the shape file that is to be imported. You must specify a non-null value for this parameter.

If you specify the optional file extension, specify either .shp or .SHP. IBM Spatial Support for Db2 for z/OS first looks for an exact match of the specified file name. If IBM Spatial Support for Db2 for z/OS does not find an exact match, it looks first for a file with the .shp extension, and then for a file with the .SHP extension.

See Usage notes for a list of required files, which must reside on the server machine. The stored procedure, which runs as a task in the WLM environment, must have the necessary privileges on the server to read the files.

The data type of this parameter is VARCHAR(256).

input_attr_columns

Specifies a list of attribute columns to import from the dBASE file. Although you must specify a value for this parameter, the value can be null. If this parameter is null, all columns are imported. If the dBASE file does not exist, this parameter must be the empty string or null.

To specify a non-null value for this parameter, use one of the following specifications:

- **List the attribute column names.** The following example shows how to specify a list of the names of the attribute columns that are to be imported from the dBASE file:

```
N(COLUMN1,COLUMN5,COLUMN3,COLUMN7)
```

  If a column name is not enclosed in double quotation marks, it is converted to uppercase. Each name in the list must be separated by a comma. The resulting names must exactly match the column names in the dBASE file.

- **List the attribute column numbers.** The following example shows how to specify a list of the numbers of the attribute columns that are to be imported from the dBASE file:

```
P(1,5,3,7)
```

  Columns are numbered beginning with 1. Each number in the list must be separated by a comma.
- **Indicate that no attribute data is to be imported.** Specify "", which is an empty string that explicitly specifies that IBM Spatial Support for Db2 for z/OS is to import *no* attribute data.

The data type of this parameter is VARCHAR(32K).

srs_name
Identifies the spatial reference system that is to be used for the geometries that are imported into the spatial column. You must specify a non-null value for this parameter.

The spatial column will not be registered. The spatial reference system (SRS) must exist before the data is imported. The import process does not implicitly create the SRS, but it does compare the coordinate system of the SRS with the coordinate system that is specified in the .prj file (if available with the shape file). The import process also verifies that the extents of the data in the shape file can be represented in the given spatial reference system. That is, the import process verifies that the extents lie within the minimum and maximum possible X, Y, Z, and M coordinates of the SRS.

The *srs_name* value is converted to uppercase unless you enclose it in double quotation marks.

The data type for this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

table_schema
Identifies the schema to which the table that is specified in the *table_name* parameter belongs. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the value in the CURRENT SCHEMA special register is used as the schema name for the table or view.

The data type for this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

table_name
Identifies the unqualified name of the table into which the imported shape file is to be loaded. You must specify a non-null value for this parameter.

The data type for this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

table_attr_columns
Specifies the table column names where attribute data from the dBASE file is to be stored. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the names of the columns in the dBASE file are used.

If this parameter is specified, the number of names must match the number of columns that are imported from the dBASE file. If the table exists, the column definitions must match the incoming data. See Usage notes for an explanation of how attribute data types are mapped to Db2 data types.

The data type of this parameter is VARCHAR(32K).

create_table_flag
Specifies whether the import process is to create a new table. Although you must specify a value for this parameter, the value can be null. If this parameter is null or any other value other than 0 (zero), a new table is created. (If the table already exists, an error is returned.) If this parameter is 0 (zero), no table is created, and the table must already exist.

If you want to create a target table in a separate table space, first create the table, and then create the LOB table space, auxiliary table, and index for the target table before using the import shape operation.

After creating the required LOB table space, auxiliary table, and index for the target table, specify 0 (zero) for the *create_table_flag* option to import shape data and attributes data to the table. The import shape operation does not create a LOB table space, an auxiliary table, or an index for the LOB column.

The data type of this parameter is SMALLINT.

table_creation_parameters
Specifies any options that are to be added to the CREATE TABLE statement that creates a table into which data is to be imported. Although you must specify a value for this parameter, the value can be null. If this parameter is null, no options are added to the CREATE TABLE statement.

To specify any CREATE TABLE options, use the syntax of the Db2 CREATE TABLE statement. For example, to specify a database and Unicode option for character columns, specify:

```
IN dbName CCSID UNICODE
```

The data type of this parameter is VARCHAR(32K).

spatial_column
Identifies the spatial column in the table into which the shape data is to be loaded. You must specify a non-null value for this parameter.

For a new table, this parameter specifies the name of the new spatial column that is to be created. Otherwise, this parameter specifies the name of an existing spatial column in the table.

The *spatial_column* value is converted to uppercase unless you enclose it in double quotation marks.

The data type for this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

type_schema
This parameter is not supported and always will be null. If you specify this parameter, the parameter is ignored.

type_name
Identifies the data type that is to be used for the spatial values. Although you must specify a value for this parameter, the value can be null. The valid data types are ST_Point, ST_MultiPoint, ST_MultiLineString, ST_MultiPolygon, or ST_Geometry.

If this parameter is null, the data type is determined by the shape file and is one of the following types:

- ST_Point
- ST_MultiPoint
- ST_MultiLineString
- ST_MultiPolygon

Note that shape files, by definition, allow a distinction between only points and multipoints, but not between polygons and multipolygons or between linestrings and multilinestrings.

If you are importing into a table that does not yet exist, this data type is also used for the data type of the spatial column.

The *type_name* value is converted to uppercase unless you enclose it in double quotation marks.

The data type for this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

inline_length
This parameter is not supported and always will be null. If you specify this parameter, the parameter is ignored.

id_column
Identifies a column that is to be created to contain a unique number for each row of data. The unique values for that column are generated automatically during the import process. Although you must specify a value for this parameter, the value can be null if no column (with a unique ID in each row) exists in the table or if you are not adding such a column to a newly created table. If this parameter is null, no column is created or populated with unique numbers.

Restriction: You cannot specify an *id_column* name that matches the name of any column in the dBASE file.

The requirements and effect of this parameter depend on whether the table already exists.

- **For an existing table**, the data type of the *id_column* parameter can be any integer type (INTEGER, SMALLINT, or BIGINT).
- **For a new table that is to be created**, the column is added to the table when the stored procedure creates it. The column will be defined as follows:

```
INTEGER NOT NULL PRIMARY KEY
```

If the value of the *id_column_is_identity* parameter is not null and not 0 (zero), the definition is expanded as follows:

```
INTEGER NOT NULL PRIMARY KEY GENERATED ALWAYS AS IDENTITY
( START WITH 1 INCREMENT BY 1 )
```

The *id_column* value is converted to uppercase unless you enclose it in double quotation marks.

The data type for this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

id_column_is_identity
Indicates whether the specified *id_column* is to be created using the IDENTITY clause. Although you must specify a value for this parameter, the value can be null. If this parameter is 0 (zero) or null, the column is not created as the identity column. If the parameter is any value other than 0 or null, the column is created as the identity column. This parameter is ignored for tables that already exist.

The data type of this parameter is SMALLINT.

restart_count
Specifies that an import operation is to be started at record *n* + 1. The first *n* records are skipped. Although you must specify a value for this parameter, the value can be null. If this parameter is null, all records (starting with record number 1) are imported.

The data type of this parameter is INTEGER.

commit_scope
Specifies that a COMMIT is to be performed after at least *n* records are imported. Although you must specify a value for this parameter, the value can be null. If this parameter is null, a value of 0 (zero) is used, and no records are committed.

The data type of this parameter is INTEGER.

exception_file
This parameter is not supported and always will be null. If you specify this parameter, the parameter is ignored.

messages_file
Specifies the full path name of the file (on the server machine) that is to contain messages about the import operation. Although you must specify a value for this parameter, the value can be null. If the parameter is null, no file for IBM Spatial Support for Db2 for z/OS messages is created.

The messages that are written to the messages file can be:

- Informational messages, such as a summary of the import operation
- Error messages for data that could not be imported, for example because of different coordinate systems

The user who runs the job that calls the stored procedure must have the necessary privileges on the server to create the file. If the file already exists, the file will be overwritten.

The data type of this parameter is VARCHAR(256).

Output parameters

msg_code
Specifies the message code that is returned from the stored procedure. The value of this output parameter identifies the error, success, or warning condition that was encountered during the processing of the procedure. If this parameter value is for a success or warning condition, the procedure finished its task. If the parameter value is for an error condition, no changes to the database were performed.

The data type of this output parameter is INTEGER.

msg_text
Specifies the actual message text, associated with the message code, that is returned from the stored procedure. The message text can include additional information about the success, warning, or error condition, such as where an error was encountered.

The data type of this output parameter is VARCHAR(4096).

When the message code that is returned is 0 (zero), the message text parameter is set to null.

Usage notes

The ST_import_shape stored procedure uses from one to four files:

- The main shape file (.shp extension). This file is required.
- The shape index file (.shx extension). This file is optional.
- A dBASE file that contains attribute data (.dbf extension). This file is required only if attribute data is to be imported.
- The projection file that specifies the coordinate system of the shape data (.prj extension). This file is optional. If this file is present, the coordinate system that is defined in it is compared with the coordinate system of the spatial reference system that is specified by the *srs_id* parameter.

The following table describes how dBASE attribute data types are mapped to Db2 data types. All other attribute data types are not supported.

Table 5. Relationship between Db2 data types and dBASE attribute data types

.dbf type	.dbf lengthb (See note)	.dbf decimalsb (See note)	SQL type	Comments
N	< 5	0	SMALLINT	
N	< 10	0	INTEGER	
N	< 20	0	BIGINT	
N	*len*	*dec*	DECIMAL(*len*,*dec*)	*len*<32
F	*len*	*dec*	REAL	*len* + *dec* < 7
F	*len*	*dec*	DOUBLE	
C	*len*		CHAR(*len*)	
L			CHAR(1)	

Table 5. Relationship between Db2 data types and dBASE attribute data types (continued)

.dbf type	.dbf lengthb (See note)	.dbf decimalsb (See note)	SQL type	Comments
D			DATE	

Note: This table includes the following variables, both of which are defined in the header of the dBASE file:

- *len*, which represents the total length of the column in the dBASE file. IBM Spatial Support for Db2 for z/OS uses this value for two purposes:
  - To define the precision for the SQL data type DECIMAL or the length for the SQL data type CHAR
  - To determine which of the integer or floating-point types is to be used
- *dec*, which represents the maximum number of digits to the right of the decimal point of the column in the dBASE file. IBM Spatial Support for Db2 for z/OS uses this value to define the scale for the SQL data type DECIMAL.

For example, assume that the dBASE file contains a column of data whose length (*len*) is defined as 20. Assume that the number of digits to the right of the decimal point (*dec*) is defined as 5. When IBM Spatial Support for Db2 for z/OS imports data from that column, it uses the values of *len* and *dec* to derive the following SQL data type: DECIMAL(20,5).

Example

This example shows how to use a Db2 CALL statement to invoke the ST_import_shape stored procedure. This example uses a Db2 CALL statement to import a shape file named /tmp/officesShape into the table named OFFICES:

```
call sysproc.ST_import_shape('/tmp/officesShape',NULL,'USA_SRS_1',NULL,
'OFFICES',NULL,0,NULL,'LOCATION',NULL,NULL,NULL,NULL, NULL,NULL,NULL,NULL,'
/tmp/import_msg',?,?)
```

The two question marks at the end of this CALL statement represent the output parameters, *msg_code* and *msg_text*. The values for these output parameters are returned after the stored procedure is called.

Related information

ST_register_spatial_column

Use this stored procedure to register a spatial column and to associate a spatial reference system (SRS) with it.

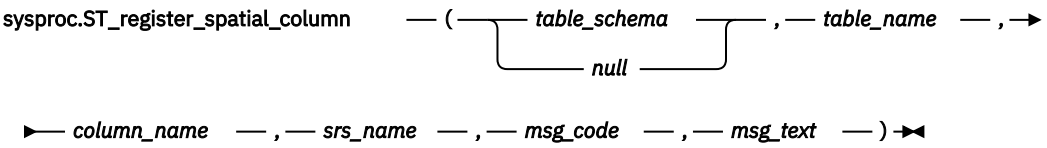
When this stored procedure is processed, information about the spatial column that is being registered is added to the DB2GSE.ST_GEOMETRY_COLUMNS catalog view. Registering a spatial column creates a constraint on the table, if possible, to ensure that all geometries use the specified SRS.

Authorization

The user ID under which this stored procedure is invoked must hold one of the following authorities or privileges:

- SYSADM or DBADM authority on the database that contains the table to which the spatial column that is being registered belongs
- All table or view privileges on this table

Syntax



Parameter descriptions

table_schema
Identifies the schema to which the table or view that is specified in the *table_name* parameter belongs. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the value in the CURRENT SCHEMA special register is used as the schema name for the table or view.

The data type of this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

table_name
Identifies the unqualified name of the table or view that contains the column that is being registered. You must specify a non-null value for this parameter.

The data type of this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

column_name
Identifies the column that is being registered. You must specify a non-null value for this parameter.

The data type of this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

srs_name
Identifies the spatial reference system that is to be used for this spatial column. You must specify a non-null value for this parameter.

The *srs_name* value is converted to uppercase unless you enclose it in double quotation marks.

The data type of this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

Output parameters

msg_code
Specifies the message code that is returned from the stored procedure. The value of this output parameter identifies the error, success, or warning condition that was encountered during the processing of the procedure. If this parameter value is for a success or warning condition, the procedure finished its task. If the parameter value is for an error condition, no changes to the database were performed.

The data type of this output parameter is INTEGER.

msg_text
Specifies the actual message text, associated with the message code, that is returned from the stored procedure. The message text can include additional information about the success, warning, or error condition, such as where an error was encountered.

The data type of this output parameter is VARCHAR(4096).

When the message code that is returned is 0 (zero), the message text parameter is set to null.

Example

This example shows how to use the Db2 CALL statement to invoke the ST_register_spatial_column stored procedure. This example uses a Db2 CALL statement to register the spatial column named LOCATION in

the table named CUSTOMERS. This CALL statement specifies the *srs_name* parameter value as USA_SRS_1:

```
call sysproc.ST_register_spatial_column(NULL,'CUSTOMERS','LOCATION',
     'USA_SRS_1',?,?)
```

The two question marks at the end of this CALL statement represent the output parameters, *msg_code* and *msg_text*. The values for these output parameters are returned after the stored procedure is called.

ST_unregister_spatial_column

Use this stored procedure to remove the registration of a spatial column.

The stored procedure removes the registration by:

- Removing association of the spatial reference system with the spatial column. The DB2GSE.ST_GEOMETRY_COLUMNS catalog view continues to contain the spatial column, but the column is no longer associated with any spatial reference system.
- For a base table, dropping the triggers that IBM Spatial Support for Db2 for z/OS placed on this table to ensure that the geometry values in this spatial column are all represented in the same spatial reference system.

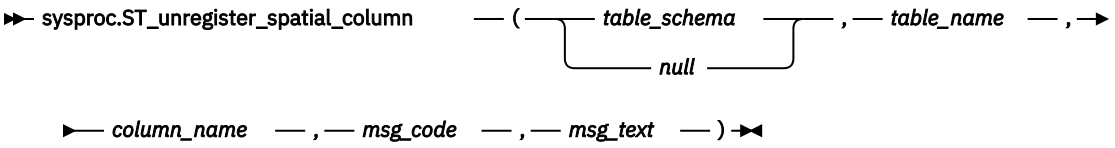
If you drop the table that contains the spatial column before calling the ST_unregister_spatial_column stored procedure, then the triggers are still dropped but an error is returned that the table does not exist.

Authorization

The user ID under which this stored procedure is invoked must hold one of the following authorities or privileges:

- SYSADM or DBADM authority
- All table or view privileges on this table

Syntax



Parameter descriptions

table_schema
Identifies the schema to which the table that is specified in the *table_name* parameter belongs. Although you must specify a value for this parameter, the value can be null. If this parameter is null, the value in the CURRENT SCHEMA special register is used as the schema name for the table or view.

The data type of this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

table_name
Identifies the unqualified name of the table that contains the column that is specified in the *column_name* parameter. You must specify a non-null value for this parameter.

The data type of this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

column_name
Identifies the spatial column that you want to unregister. You must specify a non-null value for this parameter.

The data type of this parameter is VARCHAR(128) or, if you enclose the value in double quotation marks, VARCHAR(130).

Output parameters

msg_code
Specifies the message code that is returned from the stored procedure. The value of this output parameter identifies the error, success, or warning condition that was encountered during the processing of the procedure. If this parameter value is for a success or warning condition, the procedure finished its task. If the parameter value is for an error condition, no changes to the database were performed.

The data type of this output parameter is INTEGER.

msg_text
Specifies the actual message text, associated with the message code, that is returned from the stored procedure. The message text can include additional information about the success, warning, or error condition, such as where an error was encountered.

The data type of this output parameter is VARCHAR(4096).

When the message code that is returned is 0 (zero), the message text parameter is set to null.

Example

This example shows how to use the Db2 CALL statement to invoke the ST_unregister_spatial_column stored procedure. This example uses a Db2 CALL statement to unregister the spatial column named LOCATION in the table named CUSTOMERS:

```
call sysproc.ST_unregister_spatial_column(NULL,'CUSTOMERS','LOCATION',?,?)
```

The two question marks at the end of this CALL statement represent the output parameters, *msg_code* and *msg_text*. The values for these output parameters are returned after the stored procedure is called.

Chapter 10. Catalog views

The catalog views in IBM Spatial Support for Db2 for z/OS give useful information.

Catalog views contain information about:

- Coordinate systems that you can use
- Spatial columns that you can populate or update
- Allowable maximum lengths of values that you can assign to variables
- Spatial reference systems that you can use
- The units of measure (meters, miles, feet, and so on) in which distances generated by spatial functions can be expressed

The DB2GSE.GEOMETRY_COLUMNS catalog view

This catalog view shows selected columns of the catalog table that contains information about layers.

When you create a layer, IBM Spatial Support for Db2 for z/OS registers it by recording its identifier and information relating to it in a catalog table. Selected columns from the catalog table comprise the DB2GSE.GEOMETRY_COLUMNS catalog view, which is described in the following table.

Table 6. Columns in the DB2GSE.GEOMETRY_COLUMNS catalog view

Name	Data Type	Nullable?	Content
LAYER_CATALOG	VARCHAR(30)	Yes	NULL. There is no concept of LAYER_CATALOG in IBM Spatial Support for Db2 for z/OS.
LAYER_SCHEMA	VARCHAR(30)	No	Schema of the table or view that contains the column that was registered as this layer.
LAYER_TABLE	VARCHAR(128)	No	Name of the table or view that contains the column that was registered as this layer.
LAYER_COLUMN	VARCHAR(128)	No	Name of the column that was registered as this layer.
GEOMETRY_TYPE	INTEGER	Yes	Data type of the column that was registered as this layer. If the column has a user-defined subtype of any of the geometry types defined by spatial support, then this value will be null.
SRID	INTEGER	No	Identifier of the spatial reference system used for the values in the column that was registered as this layer.
STORAGE_TYPE	INTEGER	Yes	This field is not being used. The value always will be null.

The DB2GSE.SPATIAL_REF_SYS catalog view

Query the DB2GSE.SPATIAL_REF_SYS catalog view to retrieve information about spatial reference systems.

When you create a spatial reference system, IBM Spatial Support for Db2 for z/OS registers it by recording its identifier and information related to it in a catalog table.

The following table describes the columns from the catalog table that comprise the DB2GSE.SPATIAL_REF_SYS catalog view.

© Copyright IBM Corp. 2007, 2013

Table 7. Columns in the DB2GSE.SPATIAL_REF_SYS catalog view

Name	Data Type	Nullable?	Content
SRID	INTEGER	No	User-defined identifier for this spatial reference system.
SR_NAME	VARCHAR(128)	No	Name of this spatial reference system.
CSID	INTEGER	No	Numeric identifier for the coordinate system that underlies this spatial reference system.
CS_NAME	VARCHAR(128)	No	Name of the coordinate system that underlies this spatial reference system.
AUTH_NAME	VARCHAR(128)	Yes	Name of the organization that sets the standards for this spatial reference system.
AUTH_SRID	INTEGER	Yes	The identifier that the organization specified in the AUTH_NAME column assigns to this spatial reference system.
SRTEXT	VARCHAR(2048)	No	Annotation text for this spatial reference system.
FALSEX	DOUBLE	No	A number that, when subtracted from a negative X coordinate value, leaves a non-negative number (that is, a positive number or a zero).
FALSEY	DOUBLE	No	A number that, when subtracted from a negative Y coordinate value, leaves a non-negative number (that is, a positive number or a zero).
XYUNITS	DOUBLE	No	A number that, when multiplied by a decimal X coordinate or a decimal Y coordinate, yields an integer that can be stored as a 32–bit data item.
FALSEZ	DOUBLE	No	A number that, when subtracted from a negative Z coordinate value, leaves a non-negative number (that is, a positive number or a zero).
ZUNITS	DOUBLE	No	A number that, when multiplied by a decimal Z coordinate, yields an integer that can be stored as a 32–bit data item.
FALSEM	DOUBLE	No	A number that, when subtracted from a negative measure, leaves a non-negative number (that is, a positive number or a zero).
MUNITS	DOUBLE	No	A number that, when multiplied by a decimal measure, yields an integer that can be stored as a 32–bit data item.

The DB2GSE.ST_COORDINATE_SYSTEMS catalog view

Query the DB2GSE.ST_COORDINATE_SYSTEMS catalog view to retrieve information about registered coordinate systems.

IBM Spatial Support for Db2 for z/OS automatically registers coordinate systems in the IBM Spatial Support for Db2 for z/OS catalog at the following times:

- When you enable a database for spatial operations.
- When users define additional coordinate systems to the database.

For a description of columns in this view, see the following table.

Table 8. Columns in the DB2GSE.ST_COORDINATE_SYSTEMS catalog view

Name	Data type	Nullable?	Content
COORDSYS_NAME	VARCHAR(128)	No	Name of this coordinate system. The name is unique within the database.
COORDSYS_TYPE	VARCHAR(128)	No	Type of this coordinate system: **PROJECTED** Two-dimensional. **GEOGRAPHIC** Three-dimensional. Uses X and Y coordinates. **GEOCENTRIC** Three-dimensional. Uses X, Y, and Z coordinates. **UNSPECIFIED** Abstract or non-real world coordinate system. The value for this column is obtained from the DEFINITION column.
DEFINITION	VARCHAR(2048)	No	Well-known text representation of the definition of this coordinate system.
ORGANIZATION	VARCHAR(128)	Yes	Name of the organization (for example, a standards body such as the European Petrol Survey Group, or ESPG) that defined this coordinate system. This column is null if the ORGANIZATION_COORDSYS_ID column is null.
ORGANIZATION_ COORDSYS_ID	INTEGER	Yes	Numeric identifier assigned to this coordinate system by the organization that defined the coordinate system. This identifier and the value in the ORGANIZATION column uniquely identify the coordinate system unless the identifier and the value are both null. If the ORGANIZATION column is null, then the ORGANIZATION_COORDSYS_ID column is also null.
DESCRIPTION	VARCHAR(256)	Yes	Description of the coordinate system that indicates its application.

The DB2GSE.ST_GEOMETRY_COLUMNS catalog view

Use the DB2GSE.ST_GEOMETRY_COLUMNS catalog view to find information about all spatial columns in all tables that contain spatial data in the database.

If a spatial column was registered in association with a spatial reference system, you can also use the view to find out the spatial reference system's name and numeric identifier. For additional information about spatial columns, query the Db2 SYSIBM.SYSCOLUMNS catalog table.

For a description of the DB2GSE.ST_GEOMETRY_COLUMNS catalog view, see the following table.

Table 9. Columns in the DB2GSE.ST_GEOMETRY_COLUMNS catalog view

Name	Data type	Nullable?	Content
TABLE_SCHEMA	VARCHAR(128)	No	Name of the schema to which the table that contains this spatial column belongs.
TABLE_NAME	VARCHAR(128)	No	Unqualified name of the table that contains this spatial column.
COLUMN_NAME	VARCHAR(128)	No	Name of this spatial column. The combination of TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME uniquely identifies the column.
TYPE_SCHEMA	VARCHAR(128)	No	Name of the schema to which the declared data type of this spatial column belongs. This name is obtained from the Db2 catalog.
TYPE_NAME	VARCHAR(128)	No	Unqualified name of the declared data type of this spatial column. This name is obtained from the Db2 catalog.
SRS_NAME	VARCHAR(128)	Yes	Name of the spatial reference system that is associated with this spatial column. If no spatial reference system is associated with the column, then SRS_NAME is null.
SRS_ID	INTEGER	Yes	Numeric identifier of the spatial reference system that is associated with this spatial column. If no spatial reference system is associated with the column, then SRS_ID is null.

The DB2GSE.ST_SIZINGS catalog view

The DB2GSE.ST_SIZINGS catalog view contains information about the allowable maximum lengths of values that you can assign to variables.

Use the DB2GSE.ST_SIZINGS catalog view to retrieve:

- All the variables supported by IBM Spatial Support for Db2 for z/OS; for example, coordinate system name, geocoder name, and variables to which well-known text representations of spatial data can be assigned.
- The allowable maximum length, if known, of values assigned to these variables (for example, the maximum allowable lengths of names of coordinate systems, of names of geocoders, and of well-known text representations of spatial data).

For a description of the columns in this view, see the following table.

Table 10. Columns in the DB2GSE.ST_SIZINGS catalog view

Name	Data type	Nullable?	Content
VARIABLE_NAME	VARCHAR(128)	No	Term that denotes a variable. The term is unique within the database.

Table 10. Columns in the DB2GSE.ST_SIZINGS catalog view (continued)

Name	Data type	Nullable?	Content
SUPPORTED_VALUE	INTEGER	Yes	Allowable maximum length of the values assigned to the variable shown in the VARIABLE_NAME column. Possible values in the SUPPORTED_VALUE column are: **A numeric value other than 0** The allowable maximum length of values assigned to this variable. **0** Either any length is allowed, or the allowable length cannot be determined. **NULL** IBM Spatial Support for Db2 for z/OS does not support this variable.
DESCRIPTION	VARCHAR(256)	Yes	Description of this variable.

The DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS catalog view

Query the DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS catalog view to retrieve information about registered spatial reference systems.

Spatial reference systems are automatically registered in the IBM Spatial Support for Db2 for z/OS catalog at the following times:

- When you enable a database for spatial operations (registering five default spatial reference systems)
- When users create additional spatial reference systems

To get full value from the DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS catalog view, you need to understand that each spatial reference system is associated with a coordinate system. The spatial reference system is designed partly to convert coordinates derived from the coordinate system into values that Db2 can process with maximum efficiency, and partly to define the maximum possible extent of space that these coordinates can reference.

To find out the name and type of the coordinate system associated with a given spatial reference system, query the COORDSYS_NAME and COORDSYS_TYPE columns of the DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS catalog view. For more information about the coordinate system, query the DB2GSE.ST_COORDINATE_SYSTEMS catalog view.

Table 11. Columns in the DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS catalog view

Name	Data type	Nullable?	Content
SRS_NAME	VARCHAR(128)	No	Name of the spatial reference system. This name is unique within the database.
SRS_ID	INTEGER	No	Numerical identifier of the spatial reference system. Each spatial reference system has a unique numerical identifier. Spatial functions specify spatial reference systems by their numerical identifiers rather than by their names.

Table 11. Columns in the DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS catalog view (continued)

Name	Data type	Nullable?	Content
X_OFFSET	DOUBLE	No	Offset to be subtracted from all X coordinates of a geometry. The subtraction is a step in the process of converting the geometry's coordinates into values that Db2 can process with maximum efficiency. A subsequent step is to multiply the figure resulting from the subtraction by the scale factor shown in the X_SCALE column.
X_SCALE	DOUBLE	No	Scale factor by which to multiply the figure that results when an offset is subtracted from an X coordinate. This factor is identical to the value shown in the Y_SCALE column.
Y_OFFSET	DOUBLE	No	Offset to be subtracted from all Y coordinates of a geometry. The subtraction is a step in the process of converting the geometry's coordinates into values that Db2 can process with maximum efficiency. A subsequent step is to multiply the figure resulting from the subtraction by the scale factor shown in the Y_SCALE column.
Y_SCALE	DOUBLE	No	Scale factor by which to multiply the figure that results when an offset is subtracted from a Y coordinate. This factor is identical to the value shown in the X_SCALE column.
Z_OFFSET	DOUBLE	No	Offset to be subtracted from all Z coordinates of a geometry. The subtraction is a step in the process of converting the geometry's coordinates into values that Db2 can process with maximum efficiency. A subsequent step is to multiply the figure resulting from the subtraction by the scale factor shown in the Z_SCALE column.
Z_SCALE	DOUBLE	No	Scale factor by which to multiply the figure that results when an offset is subtracted from a Z coordinate.
M_OFFSET	DOUBLE	No	Offset to be subtracted from all measures associated with a geometry. The subtraction is a step in the process of converting the measures into values that Db2 can process with maximum efficiency. A subsequent step is to multiply the figure resulting from the subtraction by the scale factor shown in the M_SCALE column.
M_SCALE	DOUBLE	No	Scale factor by which to multiply the figure that results when an offset is subtracted from a measure.
MIN_X	DOUBLE	No	Minimum possible value for X coordinates in the geometries to which this spatial reference system applies. This value is derived from the values in the X_OFFSET and X_SCALE columns.

Table 11. Columns in the DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS catalog view (continued)

Name	Data type	Nullable?	Content
MAX_X	DOUBLE	No	Maximum possible value for X coordinates in the geometries to which this spatial reference system applies. This value is derived from the values in the X_OFFSET and X_SCALE columns.
MIN_Y	DOUBLE	No	Minimum possible value for Y coordinates in the geometries to which this spatial reference system applies. This value is derived from the values in the Y_OFFSET and Y_SCALE columns.
MAX_Y	DOUBLE	No	Maximum possible value for Y coordinates in the geometries to which this spatial reference system applies. This value is derived from the values in the Y_OFFSET and Y_SCALE columns.
MIN_Z	DOUBLE	No	Minimum possible value for Z coordinates in geometries to which this spatial reference system applies This value is derived from the values in the Z_OFFSET and Z_SCALE columns.
MAX_Z	DOUBLE	No	Maximum possible value for Z coordinates in geometries to which this spatial reference system applies. This value is derived from the values in the Z_OFFSET and Z_SCALE columns.
MIN_M	DOUBLE	No	Minimum possible value for measures that can be stored with geometries to which this spatial reference system applies. This value is derived from the values in the M_OFFSET and M_SCALE columns.
MAX_M	DOUBLE	No	Maximum possible value for measures that can be stored with geometries to which this spatial reference system applies. This value is derived from the values in the M_OFFSET and M_SCALE columns.
COORDSYS_NAME	VARCHAR(128)	No	Identifying name of the coordinate system on which this spatial reference system is based.
COORDSYS_TYPE	VARCHAR(128)	No	Type of the coordinate system on which this spatial reference system is based.
ORGANIZATION	VARCHAR(128)	Yes	Name of the organization (for example, a standards body) that defined the coordinate system on which this spatial reference system is based. ORGANIZATION is null if ORGANIZATION_COORSYS_ID is null.
ORGANIZATION_ COORDSYS_ID	INTEGER	Yes	Name of the organization (for example, a standards body) that defined the coordinate system on which this spatial reference system is based. ORGANIZATION_COORDSYS_ID is null if ORGANIZATION is null.
DEFINITION	VARCHAR(2048)	No	Well-known text representation of the definition of the coordinate system.
DESCRIPTION	VARCHAR(256)	Yes	Description of the spatial reference system.

The DB2GSE.ST_UNITS_OF_MEASURE catalog view

Consult the DB2GSE.ST_UNITS_OF_MEASURE catalog view to see what units of measure are available.

Certain spatial functions accept or return values that denote a specific distance. In some cases, you can choose what unit of measure the distance is to be expressed in. For example, ST_Distance returns the minimum distance between two specified geometries. On one occasion you might require ST_Distance to return the distance in terms of miles; on another, you might require a distance expressed in terms of meters. To find out what units of measure you can choose from, consult the DB2GSE.ST_UNITS_OF_MEASURE catalog view.

Table 12. Columns in the DB2GSE.ST_UNITS_OF_MEASURE catalog view

Name	Data type	Nullable?	Content
UNIT_NAME	VARCHAR(128)	No	Name of the unit of measure. This name is unique in the database.
UNIT_TYPE	VARCHAR(128)	No	Type of the unit of measure. Possible values are: **LINEAR** The unit of measure is linear. **ANGULAR** The unit of measure is angular.
CONVERSION_FACTOR	DOUBLE	No	Numeric value used to convert this unit of measure to its base unit. The base unit for linear units of measure is METER; the base unit for angular units of measure is RADIAN. The base unit itself has a conversion factor of 1.0.
DESCRIPTION	VARCHAR(256)	Yes	Description of the unit of measure.

Chapter 11. Spatial functions: categories and uses

This information introduces all of the spatial functions that IBM Spatial Support for Db2 for z/OS provides, organizing them by category.

IBM Spatial Support for Db2 for z/OS provides functions that:

- Convert geometries to and from various data exchange formats. These functions are called constructor functions.
- Compare geometries for boundaries, intersections, and other information. These functions are called comparison functions.
- Return information about properties of geometries, such as coordinates and measures within geometries, relationships between geometries, and boundary and other information.
- Generate new geometries from existing geometries.
- Measure the shortest distance between points in geometries.
- Provide information about index parameters.
- Provide projections and conversions between different coordinate systems.

Constructor functions

Constructor functions are spatial functions that you can use to build spatial objects from the following input formats: well-known text (WKT) representation, well-known binary (WKB) representation, ESRI shape representation, and Geography Markup Language (GML).

Constructor functions have the same name as the geometry data type of the column into which the data will be inserted. These functions operate consistently on each of the input data exchange formats.

Spatial support includes the following constructor functions:

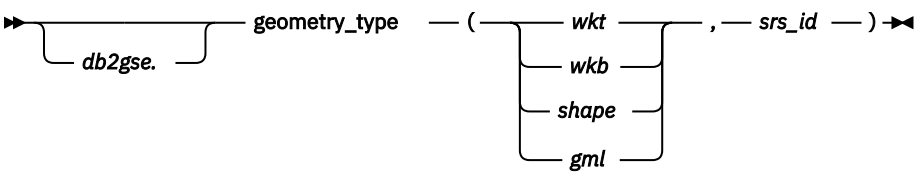
- ST_POINT
- ST_LINESTRING
- ST_POLYGON
- ST_MULTIPOINT
- ST_MULTILINESTRING
- ST_MULTIPOLYGON

The ST_Point function also takes X, Y, Z, and M coordinate values as input.

Functions that operate on data exchange formats

This section provides the syntax for calling functions that operate on data exchange formats, describes the input parameters for the functions, and identifies the type of geometry that these functions return.

Syntax



© Copyright IBM Corp. 2007, 2013

Parameters and other elements of syntax

db2gse
Name of the schema to which the spatial data types supplied by IBM Spatial Support for Db2 for z/OS belong.

geometry_type
One of the following constructor functions:

- ST_Point
- ST_LineString
- ST_Polygon
- ST_MultiPoint
- ST_MultiLineString
- ST_MultiPolygon

wkt
A value of type CLOB(8M) that contains the well-known text representation of the geometry.

wkb
A value of type BLOB(4M) that contains the well-known binary representation of the resulting linestring. If the well-known binary representation is null, then null is returned.

shape
A value of type BLOB(4M) that contains the shape representation of the resulting linestring. If the shape representation is null, then null is returned.

gml
A value of type CLOB(8M) that contains the GML representation of the geometry.

srs_id
A value of type INTEGER that identifies the spatial reference system for the resulting geometry.

Return Type

geometry_type

If **geometry_type** is ST_Geometry, the dynamic type of the returned geometry type corresponds to the geometry indicated by the input value.

If **geometry_type** is any other type, the dynamic type of the returned geometry type corresponds to the function name. If the geometry indicated by the input value does not match the function name or the name of one of its subtypes, an error is returned.

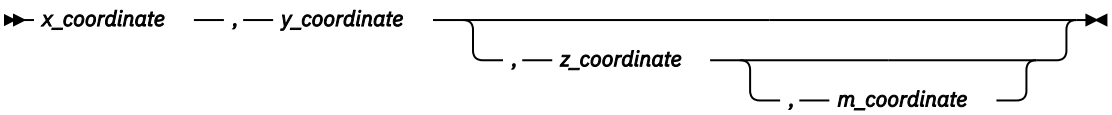
A function that creates geometries from coordinates

The ST_Point function creates geometries not only from data exchange formats, but also from numeric coordinate values—a very useful capability if your location data is already stored in your database.

Syntax



coordinates



Parameters

x_coordinate
A value of type DOUBLE that specifies the X coordinate for the resulting point.

y_coordinate
A value of type DOUBLE that specifies the Y coordinate for the resulting point.

z_coordinate
A value of type DOUBLE that specifies the Z coordinate for the resulting point.

If the *z_coordinate* parameter is omitted, the resulting point will not have a Z coordinate.

m_coordinate
A value of type DOUBLE that specifies the M coordinate for the resulting point.

If the *m_coordinate* parameter is omitted, the resulting point will not have a measure.

srs_id
A value of type INTEGER that identifies the spatial reference system for the resulting point.

If *srs_id* does not identify a spatial reference system listed in the catalog view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS, an exception condition is raised.

Return type

db2gse.ST_Point

Examples

This topic provides examples of code for invoking constructor functions, code for creating tables to contain the output of constructor functions, and code for retrieving the output.

The following example inserts a row into the SAMPLE_GEOMETRY table with ID 100 and a point value with an X coordinate of 30, a Y coordinate of 40, and in spatial reference system 1 using the coordinate representation and well-known text (WKT) representation. It then inserts another row with ID 200 and a linestring value with the coordinates indicated.

```
CREATE TABLE sample_geometry (id INT, geom db2gse.ST_Geometry);

INSERT INTO sample_geometry(id, geom)
       VALUES(100,db2gse.ST_Geometry(db2gse.ST_Point('POINT(30 40)',1)));

INSERT INTO sample_geometry(id, geom)
       VALUES(200,db2gse.ST_Geometry(db2gse.ST_Linestring('linestring(50 50,
                                100 100)', 1)));
```

If you know that the spatial column can only contain ST_Point values, you can use the following example, which inserts two points. Attempting to insert a linestring or any other type which is not a point results in an SQL error. The first insert creates a point geometry from the well-known-text representation (WKT). The second insert creates a point geometry from numeric coordinate values. These input values could also be selected from existing table columns.

```
CREATE TABLE sample_points (id INT, geom db2gse.ST_Point);

INSERT INTO sample_points(id, geom)
       VALUES(100,db2gse.ST_Point('point(30 40)', 1));

INSERT INTO sample_points(id, geom)
       VALUES(101,db2gse.ST_Point(50, 50, 1));
```

The following example uses embedded SQL and assumes that the application fills the data areas with the appropriate values.

```
EXEC SQL BEGIN DECLARE SECTION;
   sqlint32 id = 0;
   SQL TYPE IS CLOB(10000) wkt_buffer;
   SQL TYPE IS BLOB(10000) wkb_buffer;
   SQL TYPE IS BLOB(10000) shape_buffer;
```

```
EXEC SQL END DECLARE SECTION;

// * Application logic to read into buffers goes here */

EXEC SQL INSERT INTO sample_geometry(id, geom)
         VALUES(:id, db2gse.ST_Geometry(:wkt_buffer,1));

EXEC SQL INSERT INTO sample_geometry(id, geom)
         VALUES:id, db2gse.ST_Geometry(:wkb_buffer,1));

EXEC SQL INSERT INTO sample_geometry(id, geom)
         VALUES(:id, db2gse.ST_Geometry(:shape_buffer,1));
```

The following sample Java™ code uses JDBC to insert point geometries using the WKT representation to specify the geometry and using the X, Y numeric coordinate values to specify the geometries.

```
String ins1 = "INSERT into sample_geometry (id, geom)
               VALUES(?, db2gse.ST_Point(CAST( ?
               as VARCHAR(128)), 1))";
PreparedStatement pstmt = con.prepareStatement(ins1);
pstmt.setInt(1, 100);                    // id value
pstmt.setString(2, "point(32.4 50.7)");  // wkt value
int rc = pstmt.executeUpdate();

String ins2 = "INSERT into sample_geometry (id, geom)
               VALUES(?, db2gse.ST_Point(CAST( ? as double),
               CAST(? as double), 1))";
pstmt = con.prepareStatement(ins2);
pstmt.setInt(1, 200);          // id value
pstmt.setDouble(2, 40.3);      // lat
pstmt.setDouble(3, -72.5);     // long
rc = pstmt.executeUpdate();
```

Conversion to well-known text (WKT) representation

Text representations are CLOB values representing ASCII character strings. They allow geometries to be exchanged in ASCII text form.

The ST_AsText function converts a geometry value stored in a table to a WKT string. The following example uses a simple command-line query to select the values that were previously inserted into the SAMPLE_GEOMETRY table. (This example returns SQL code 445, because the data has been truncated.)

```
SELECT id, VARCHAR(db2gse.ST_AsText(geom), 50) AS WKTGEOM
FROM sample_geometry;

ID         WKTGEOM
------     --------------------------------
   100   POINT ( 30.00000000 40.00000000)
   200   LINESTRING ( 50.00000000 50.00000000, 100.00000000 100.00000000)
```

The following example uses embedded SQL to select the values that were previously inserted into the SAMPLE_GEOMETRY table.

```
 EXEC SQL BEGIN DECLARE SECTION;
 sqlint32 id = 0;
 SQL TYPE IS CLOB(10000) wkt_buffer;
 short wkt_buffer_ind = -1;
 EXEC SQL END DECLARE SECTION;

 EXEC SQL
 SELECT id, db2gse.ST_AsText(geom)
 INTO :id, :wkt_buffer :wkt_buffer_ind
 FROM sample_geometry
 WHERE id = 100;
```

In addition, IBM Spatial Support for Db2 for z/OS provides other functions that convert geometries to and from well-known text representations. The following additional functions implement the Open Geospatial

Consortium (OGC) standard Simple Features SQL and the ISO standard SQL multimedia and application packages - Part 3: Spatial:

- ST_GeomFromText
- ST_WKTToSQL

Conversion to well-known binary (WKB) representation

The WKB representation consists of binary data structures that must be BLOB values.

These BLOB values represent binary data structures that must be managed by an application program written in a programming language that Db2 supports and for which Db2 has a language binding.

The ST_AsBinary function converts a geometry value stored in a table to the well-known binary (WKB) representation, which can be fetched into a BLOB variable in program storage. The following example uses embedded SQL to select the values that were previously inserted into the SAMPLE_GEOMETRY table.

```
EXEC SQL BEGIN DECLARE SECTION;
   sqlint32 id = 0;
   SQL TYPE IS BLOB(10000) wkb_buffer;
   short wkb_buffer_ind = -1;
EXEC SQL END DECLARE SECTION;

EXEC SQL
   SELECT id, db2gse.ST_AsBinary(geom)
   INTO :id, :wkb_buffer :wkb_buffer_ind
   FROM sample_geometry
   WHERE id = 200;
```

In addition, IBM Spatial Support for Db2 for z/OS provides other functions that convert geometries to and from well-known binary representations. The following additional functions implement the Open Geospatial Consortium (OGC) standard Simple Features SQL and the ISO standard SQL multimedia and application packages - Part 3: Spatial:

- ST_GeomFromWKB
- ST_WKBToSQL

Conversion to ESRI shape representation

The ESRI shape representation consists of binary data structures that must be managed by an application program written in a supported language.

The **ST_AsShape** function converts a geometry value stored in a table to the ESRI shape representation, which can be fetched into a BLOB variable in program storage. The following example uses embedded SQL to select the values that were previously inserted into the SAMPLE_GEOMETRY table.

```
EXEC SQL BEGIN DECLARE SECTION;
   sqlint32 id;
   SQL TYPE IS BLOB(10000) shape_buffer;
EXEC SQL END DECLARE SECTION;

EXEC SQL
   SELECT id, db2gse.ST_AsShape(geom)
   INTO :id, :shape_buffer
   FROM sample_geometry;
```

Conversion to Geography Markup Language (GML) representation

Geography Markup Language (GML) representations are ASCII strings. GML representations allow geometries to be exchanged in ASCII text format.

The ST_AsGML function converts a geometry value that is stored in a table to a GML text string. The following example selects the values that were previously inserted into the SAMPLE_GEOMETRY table.

The results shown in the example are reformatted for readability. The spacing in your results might vary according to your online display.

```
SELECT id, VARCHAR(db2gse.ST_AsGML(geom, 'EPSG', 4269), 500) AS GMLGEOM
FROM sample_geometry;

ID           GMLGEOM
-----------  ------------------------------------------------------------
        100 <gml:Point srsName="EPSG:4269">
            <gml:coord><gml:X>30</gml:X><gml:Y>40</gml:Y></gml:coord>
            </gml:Point>
        200 <gml:LineString srsName="EPSG:4269">
            <gml:coord><gml:X>50</gml:X><gml:Y>50</gml:Y></gml:coord>
            <gml:coord><gml:X>100</gml:X><gml:Y>100</gml:Y></gml:coord>
            </gml:LineString>
```

Comparison functions

You can use comparison functions to compare two geometries with one another.

Comparison functions return a value of 1 (one) if a comparison meets certain criteria, a value of 0 (zero) if a comparison fails to meet the criteria, and a null value if the comparison could not be performed. Comparisons cannot be performed if the comparison operation has not been defined for the input parameters, or if either of the parameters is null. Comparisons can be performed if geometries with different data types or dimensions are assigned to the parameters.

The Dimensionally Extended 9 Intersection Model (DE-9IM) is a mathematical approach that defines the pair-wise spatial relationship between geometries of different types and dimensions. This model expresses spatial relationships between all types of geometries as pair-wise intersections of their interiors, boundaries, and exteriors, with consideration for the dimension of the resulting intersections.

Given geometries *a* and *b*: I(*a*), B(*a*), and E(*a*) represent the interior, boundary, and exterior of *a*, respectively. And, I(b), B(b), and E(b) represent the interior, boundary, and exterior of *b*. The intersections of I(a), B(a), and E(a) with I(b), B(b), and E(b) produce a 3–by–3 matrix. Each intersection can result in geometries of different dimensions. For example, the intersection of the boundaries of two polygons consists of a point and a linestring, in which case the dim function returns the maximum dimension of 1.

The dim function returns a value of –1, 0, 1 or 2. The –1 corresponds to the null set or dim(null), which is returned when no intersection was found.

Results returned by comparison functions can be understood or verified by comparing the results returned by a comparison function with a pattern matrix that represents the acceptable values for the DE-9IM.

The pattern matrix contains the acceptable values for each of the intersection matrix cells. The possible pattern values are:

T
: An intersection must exist; dim = 0, 1, or 2.

F
: An intersection must not exist; dim = -1.

*****
: It does not matter if an intersection exists; dim = -1, 0, 1, or 2.

0
: An intersection must exist and its exact dimension must be 0; dim = 0.

1
: An intersection must exist and its maximum dimension must be 1; dim = 1.

2
: An intersection must exist and its maximum dimension must be 2; dim = 2.

For example, the following pattern matrix for the ST_Within function includes the values T, F, and *.

Table 13. Matrix for ST_Within. The pattern matrix of the ST_Within function for geometry combinations.

	Geometry b Interior	Geometry b Boundary	Geometry b Exterior
Geometry a Interior	T	*	F
Geometry a Boundary	*	*	F
Geometry a Exterior	*	*	*

The ST_Within function returns a value of 1 when the interiors of both geometries intersect and when the interior or boundary of *a* does not intersect the exterior of *b.* All other conditions do not matter.

Each function has at least one pattern matrix, but some require more than one to describe the relationships of various geometry type combinations.

The DE-9IM was developed by Clementini and Felice, who dimensionally extended the 9 Intersection Model of Egenhofer and Herring. The DE-9IM is a collaboration of four authors (Clementini, Eliseo, Di Felice, and van Osstrom) who published the model in "A Small Set of Formal Topological Relationships Suitable for End-User Interaction," D. Abel and B.C. Ooi (Ed.), *Advances in Spatial Database—Third International Symposium. SSD '93*. LNCS 692. Pp. 277-295. The 9 Intersection model by M. J. Egenhofer and J. Herring (Springer-Verlag Singapore [1993]) was published in "Categorizing binary topological relationships between regions, lines, and points in geographic databases," *Tech. Report, Department of Surveying Engineering*, University of Maine, Orono, ME 1991.

Spatial comparison functions

Spatial comparison functions compare two geometries with one another.

The comparison functions are:

- EnvelopesIntersect
- ST_Contains
- ST_Crosses
- ST_Disjoint
- ST_Equals
- ST_Intersects
- ST_Overlaps
- ST_Relate
- ST_Touches
- ST_Within

Functions that compare geographic features

You can use comparison functions to compare geographic features.

Certain spatial functions return information about ways in which geographic features relate to one another or compare with one another. Other spatial functions return information as to whether two definitions of coordinate systems or two spatial reference systems are the same. In all cases, the information returned is a result of a comparison between geometries, between definitions of coordinate systems, or between spatial reference systems.

Table 14. Comparison functions by purpose

Purpose	Functions
Determines whether the interior of one geometry intersects the interior of another.	• ST_Contains • ST_Within
Returns information about intersections of geometries.	• ST_Crosses • ST_Disjoint • ST_Intersects • ST_Overlaps • ST_Touches
Determines whether the smallest rectangle that encloses one geometry intersects with the smallest rectangle that encloses another geometry.	• EnvelopesIntersect
Determines whether two objects are identical.	• ST_Equals
Determines the shortest distance between any point in the first geometry to any point in the second geometry.	• ST_Distance
Determines whether the geometries that are being compared meet the conditions of the DE-9IM pattern matrix string.	• ST_Relate

Functions that check whether one geometry contains another

ST_Contains and ST_Within both take two geometries as input and determine whether the interior of one intersects the interior of the other.

In colloquial terms, ST_Contains determines whether the first geometry given to it encloses the second geometry (whether the first contains the second). ST_Within determines whether the first geometry is completely inside the second (whether the first is within the second).

ST_Contains

Use ST_Contains to determine whether one geometry is completely contained by another geometry.

ST_Contains returns a value of 1 (one) if the second geometry is completely contained by the first geometry. The ST_Contains function returns the exact opposite result of the ST_Within function.

Figure 14 on page 93 shows examples of ST_Contains:

- A multipoint geometry contains a point or multipoint geometries when all of the points are within the first geometry.
- A polygon geometry contains a multipoint geometry when all of the points are either on the boundary of the polygon or in the interior of the polygon.
- A linestring geometry contains a point, multipoint, or linestring geometries when all of the points are within the first geometry.
- A polygon geometry contains a point, linestring or polygon geometries when the second geometry is in the interior of the polygon.

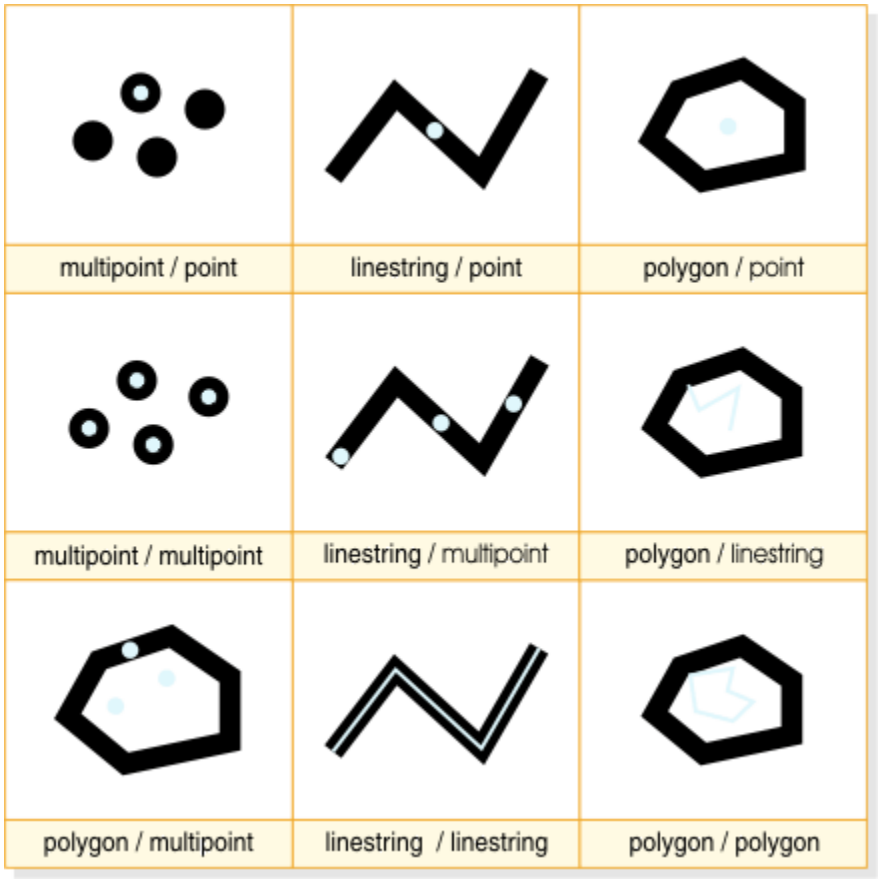


Figure 14. ST_Contains

The pattern matrix of the ST_Contains function states that the interiors of both geometries must intersect and that the interior or boundary of the secondary (geometry *b*) must not intersect the exterior of the primary (geometry *a*). The asterisk (*) indicates that it does not matter if an intersection exists between these parts of the geometries.

Table 15. Matrix for ST_Contains

	Geometry b Interior	Geometry b Boundary	Geometry b Exterior
Geometry a Interior	T	*	*
Geometry a Boundary	*	*	*
Geometry a Exterior	F	F	*

ST_Within

Use ST_Within to determine whether one geometry is completely within another geometry.

ST_Within returns a value of 1 (one) if the first geometry is completely within the second geometry. ST_Within returns the exact opposite result of ST_Contains.

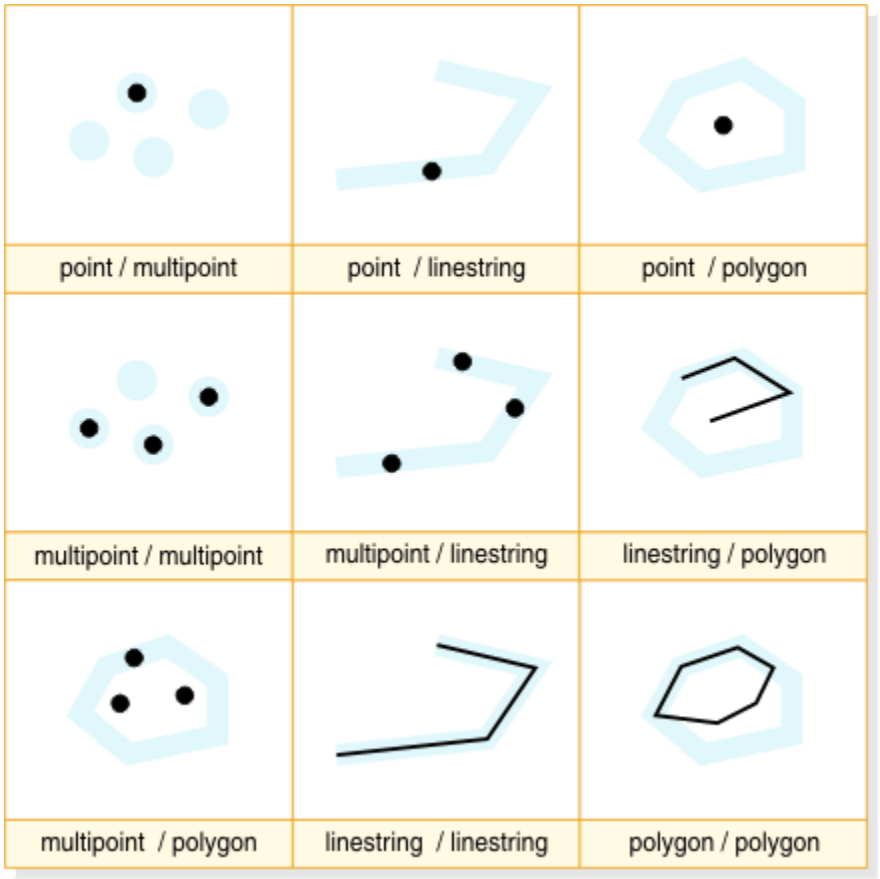


Figure 15. ST_Within

The ST_Within function pattern matrix states that the interiors of both geometries must intersect, and that the interior or boundary of the primary geometry (geometry *a*) must not intersect the exterior of the secondary (geometry *b*). The asterisk (*) indicates that all other intersections do not matter.

Table 16. Matrix for ST_Within

	Geometry b Interior	Geometry b Boundary	Geometry b Exterior
Geometry a Interior	T	*	F
Geometry a Boundary	*	*	F
Geometry a Exterior	*	*	*

Figure 15 on page 94 shows examples of ST_Within:

- A point geometry is within a multipoint geometry when its interior intersects one of the points in the second geometry.
- A multipoint geometry is within a multipoint geometry when the interiors of all points intersect the second geometry.
- A multipoint geometry is within a polygon geometry when all of the points are either on the boundary of the polygon or in the interior of the polygon.
- A point geometry is within a linestring geometry when all of the points are within the second geometry. In Figure 15 on page 94, the point is not within the linestring because its interior does not intersect the linestring; however, the multipoint geometry is within the linestring because all of its points intersect the interior of the linestring.

- A linestring geometry is within another linestring geometries when all of its points intersect the second geometry.
- A point geometry is not within a polygon geometry because its interior does not intersect the boundary or interior of the polygon.
- A linestring geometry is within a polygon geometry when all of its points intersect either the boundary or interior of the polygon.
- A polygon geometry is within a polygon geometry when all of its points intersect either the boundary or interior of the polygon.

Functions that check intersections between geometries

The EnvelopesIntersect, ST_Intersects, ST_Crosses, ST_Overlaps, and ST_Touches functions all determine whether one geometry intersects another.

These functions differ mainly as to the scope of intersection that they test for.

The EnvelopesIntersect function determines if the minimum bounding rectangles of two geometries intersect.

The ST_Intersects function tests to determine whether the two geometries given to it meet one of four conditions: that the geometries' interiors intersect, that their boundaries intersect, that the boundary of the first geometry intersects with the interior of the second, or that the interior of the first geometry intersects with the boundary of the second.

The ST_Crosses function is used to analyze the intersection of geometries of different dimensions, with one exception: it can also analyze the intersection of linestrings. In all cases, the place of intersection is itself considered a geometry; and ST_Crosses requires that this geometry be of a lesser dimension than the greater of the intersecting geometries (or, if both are linestrings, that the place of intersection be of a lesser dimension than a linestring). For example, the dimensions of a linestring and polygon are 1 and 2, respectively. If two such geometries intersect, and if the place of intersection is linear (the linestring's path along the polygon), then that place can itself be considered a linestring. And because a linestring's dimension (1) is lesser than a polygon's (2), ST_Crosses, after analyzing the intersection, would return a value of 1.

The geometries given to the ST_Overlaps function as input must be of the same dimension. ST_Overlaps requires that these geometries overlap part-way, forming a new geometry (the region of overlap) that is the same dimension as they are.

The ST_Touches function determines whether the boundaries of two geometries intersect.

EnvelopesIntersect

Use the EnvelopesIntersect spatial function to determine if the minimum bounding rectangles of two geometries intersect.

The EnvelopesIntersect function accepts two types of input parameters:

- Two geometries

  EnvelopesIntersect returns 1 if the envelope of the first geometry intersects the envelope of the second geometry. Otherwise, 0 (zero) is returned.
- A geometry, four type DOUBLE coordinate values that define the lower-left and upper-right corners of a rectangular window, and the spatial reference system identifier.

  EnvelopesIntersect returns 1 if the envelope of the first geometry intersects with the envelope defined by the four type DOUBLE values. Otherwise, 0 (zero) is returned.

If one of the input geometries is empty or null, null is returned.

ST_Intersects

Use ST_Intersects to determine whether two geometries intersect.

ST_Intersects returns a value of 1 (one) if the intersection does not result in an empty set.

The ST_Intersects function returns 1 (one) if the conditions of any of the following pattern matrices returns TRUE.

Table 17. Matrix for ST_Intersects (1). The ST_Intersects function returns 1 (one) if the interiors of both geometries intersect.

	Geometry b Interior	Geometry b Boundary	Geometry b Exterior
Geometry a Boundary	*	*	*
Geometry a Interior	T	*	*
Geometry a Exterior	*	*	*

Table 18. Matrix for ST_Intersects (2). The ST_Intersects function returns 1 (one) if the boundary of the first geometry intersects the boundary of the second geometry.

	Geometry b Interior	Geometry b Boundary	Geometry b Exterior
Geometry a Boundary	*	*	*
Geometry a Interior	*	T	*
Geometry a Exterior	*	*	*

Table 19. Matrix for ST_Intersects (3). The ST_Intersects function returns 1 (one) if the boundary of the first geometry intersects the interior of the second.

	Geometry b Interior	Geometry b Boundary	Geometry b Exterior
Geometry a Boundary	T	*	*
Geometry a Interior	*	*	*
Geometry a Exterior	*	*	*

Table 20. Matrix for ST_Intersects (4). The ST_Intersects function returns 1 (one) if the boundaries of either geometry intersect.

	Geometry b Interior	Geometry b Boundary	Geometry b Exterior
Geometry a Boundary	*	T	*
Geometry a Interior	*	*	*
Geometry a Exterior	*	*	*

ST_Crosses

Use ST_Crosses to determine whether one geometry crosses another.

ST_Crosses takes two geometries and returns a value of 1 (one) if:

- The intersection results in a geometry whose dimension is less than the maximum dimension of the source geometries.
- The intersection set is interior to both source geometries.

ST_Crosses returns a null if the first geometry is a polygon or multipolygon or if the second geometry is a point or multipoint. For all other combinations, ST_Crosses returns either a value of 1 (indicating that the two geometries cross) or a value of 0 (indicating that they do not cross).

The following figure illustrates multipoints crossing linestring, linestring crossing linestring, multiple points crossing a polygon, and linestring crossing a polygon. In three of the four cases, geometry b

crosses geometry a. In the fourth case geometry a is a multipoint which does not cross the line, but does touch the area inside the geometry b polygon.

The dark geometries represent geometry a; the gray geometries represent geometry b.

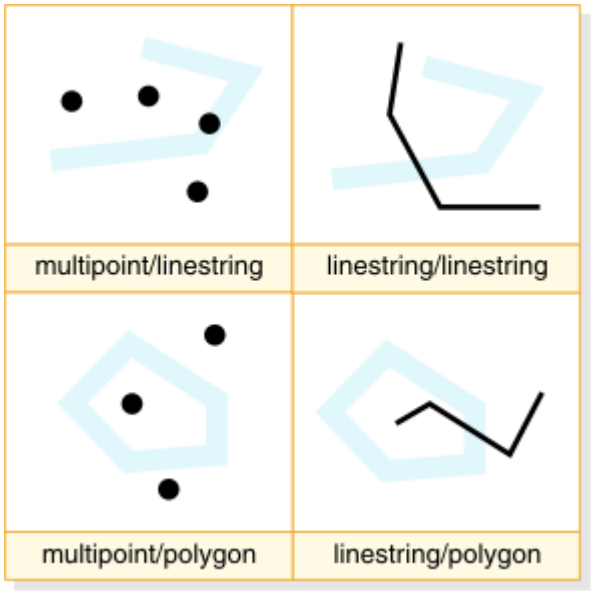


Figure 16. ST_Crosses

The pattern matrix in the following table applies if the first geometry is a point or multipoint, or if the first geometry is a linestring or multilinestring, and the second geometry is a polygon. The matrix states that the interiors must intersect and that the interior of the primary (geometry *a*) must intersect the exterior of the secondary (geometry *b*).

Table 21. Matrix for ST_Crosses (1)

	Geometry b Interior	Geometry b Boundary	Geometry b Exterior
Geometry a Boundary	*	*	*
Geometry a Interior	T	*	T
Geometry a Exterior	*	*	*

The pattern matrix in the following table applies if the first and second geometries are both linestrings or multilinestrings. The 0 indicates that the intersection of the interiors must be a point (dimension 0). If the dimension of this intersection is 1 (intersect at a linestring), the ST_Crosses function returns a value of 0 (indicating that the geometries do not cross); however, the ST_Overlaps function returns a value of 1 (indicating that the geometries overlap).

Table 22. Matrix for ST_Crosses (2)

	Geometry b Interior	Geometry b Boundary	Geometry b Exterior
Geometry a Boundary	*	*	*
Geometry a Interior	0	*	*
Geometry a Exterior	*	*	*

ST_Overlaps

Use ST_Overlaps to determine whether two geometries of the same dimension overlap.

ST_Overlaps compares two geometries of the same dimension. It returns a value of 1 (one) if their intersection set results in a geometry that is different from both, but that has the same dimension.

The dark geometries represent geometry *a*; the gray geometries represent geometry *b*. In all cases, both geometries have the same dimension, and one overlaps the other partway. The area of overlap is a new geometry; it has the same dimension as geometries *a* and *b*.

The following figure illustrates overlaps in geometries. The three examples show overlaps with points, linestrings, and polygons. With points the actual points overlap. With linestrings, a portion of the line overlaps. With polygons a portion of the area overlaps.

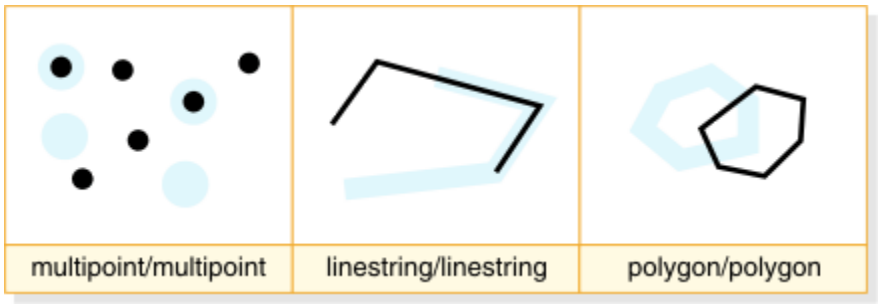


Figure 17. ST_Overlaps

The pattern matrix in Table 23 on page 98 applies if the first and second geometries are both either points, multipoints, polygons, or multipolygons. ST_Overlaps returns a value of 1 if the interior of each geometry intersects the other geometry's interior and exterior.

Table 23. Matrix for ST_Overlaps (1)

	Geometry b Interior	Geometry b Boundary	Geometry b Exterior
Geometry a Boundary	*	*	*
Geometry a Interior	T	*	T
Geometry a Exterior	T	*	*

The pattern matrix in Table 24 on page 98 applies if the first and second geometries are both linestrings or multilinestrings. In this case, the intersection of the geometries must result in a geometry that has a dimension of 1 (another linestring). If the dimension of the intersection of the interiors is 0, ST_Overlaps returns a value of 0 (indicating that the geometries do not overlap); however the ST_Crosses function would return a value of 1 (indicating that the geometries cross).

Table 24. Matrix for ST_Overlaps (2)

	Geometry b Interior	Geometry b Boundary	Geometry b Exterior
Geometry a Boundary	*	*	*
Geometry a Interior	1	*	T
Geometry a Exterior	T	*	*

ST_Touches

Use ST_Touches to determine whether the boundaries of two geometries intersect.

ST_Touches returns a value of 1 (one) if all the points common to both geometries can be found only on the boundaries. The interiors of the geometries must not intersect one another. At least one geometry must be a linestring, polygon, multilinestring, or multipolygon.

The dark geometries represent geometry a; the gray geometries represent geometry b. In all cases, the boundary of geometry b intersects geometry a. The interior of geometry b remains separate from geometry a.

The following figure shows examples of touching with types of geometries, such as point and linestring, linestring and linestring, point and polygon, multipoint and polygon, and linestring and polygon.

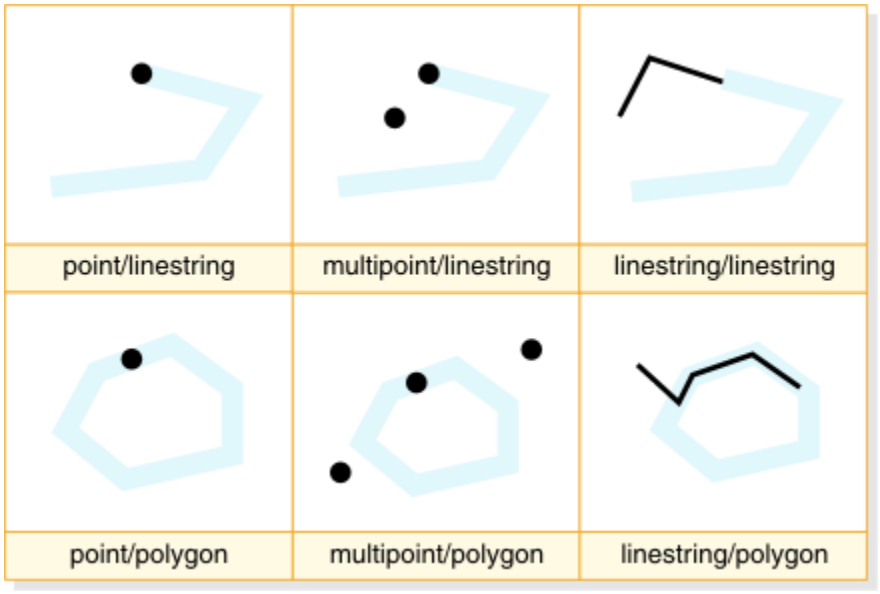


Figure 18. ST_Touches

The pattern matrices show that the ST_Touches function returns 1 (one) when the interiors of the geometry do not intersect, and the boundary of either geometry intersects the other's interior or its boundary.

Table 25. Matrix for ST_Touches (1)

	Geometry b Interior	Geometry b Boundary	Geometry b Exterior
Geometry a Boundary	*	*	*
Geometry a Interior	F	T	*
Geometry a Exterior	*	*	*

Table 26. Matrix for ST_Touches (2)

	Geometry b Interior	Geometry b Boundary	Geometry b Exterior
Geometry a Boundary	T	*	*
Geometry a Interior	F	*	*
Geometry a Exterior	*	*	*

Table 27. Matrix for ST_Touches (3)

	Geometry b Interior	Geometry b Boundary	Geometry b Exterior
Geometry a Boundary	*	T	*
Geometry a Interior	F	*	*
Geometry a Exterior	*	*	*

Function that checks whether two geometries are identical

You can use the ST_Equals comparison function to check whether two geometries are identical.

ST_Equals

Use ST_Equals to determine if two geometries are identical.

ST_Equals returns a value of 1 (one) if two geometries are identical. The order of the points used to define the geometries is not relevant to the test of equality.

In the six examples (point, multipoint, linestring, multistring, polygon, and multipolygon) geometry a and geometry b are the same.

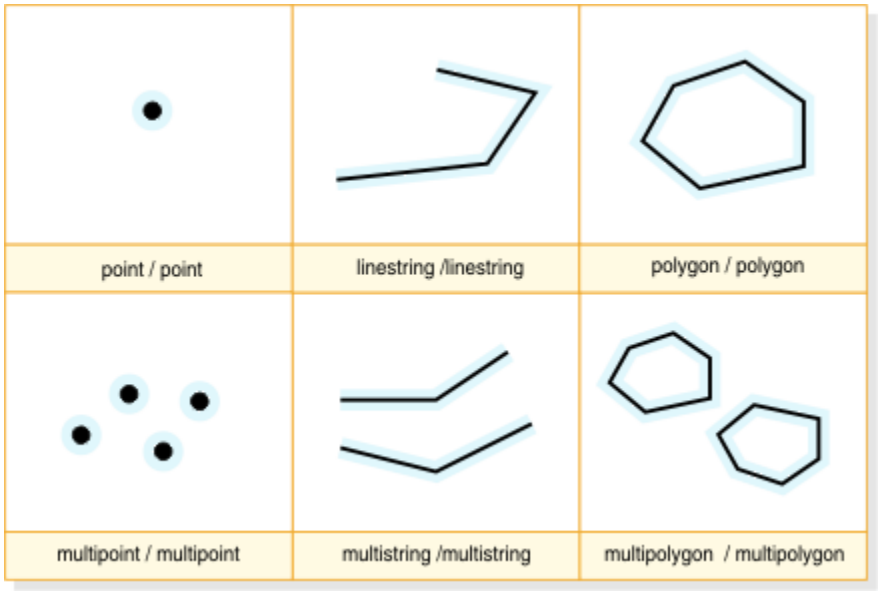


Figure 19. ST_Equals

Table 28. Matrix for equality. The DE-9IM pattern matrix for equality ensures that the interiors intersect and that no part interior or boundary of either geometry intersects the exterior of the other.

	Geometry b Interior	Geometry b Boundary	Geometry b Exterior
Geometry a Boundary	*	*	F
Geometry a Interior	T	*	F
Geometry a Exterior	F	F	*

Functions that return coordinate and measure information

You can use certain functions to return information about the coordinates and measures within a geometry.

For example, ST_X can return the X coordinate within a specified point, ST_MaxX returns the highest X coordinate within a geometry, and ST_MinX returns the lowest X coordinate within a geometry.

These functions are:

- ST_Is3D
- ST_IsMeasured
- ST_IsValid
- ST_M

- ST_MaxM
- ST_MaxX
- ST_MaxY
- ST_MaxZ
- ST_MinM
- ST_MinX
- ST_MinY
- ST_MinZ
- ST_X
- ST_Y
- ST_Z

ST_Is3D

ST_Is3D takes a geometry as an input parameter and returns 1 if the given geometry has Z coordinates. Otherwise, 0 (zero) is returned.

Related reference

"ST_Is3D" on page 150

ST_Is3D takes a geometry as an input parameter and returns 1 if the given geometry has Z coordinates. Otherwise, 0 (zero) is returned.

ST_IsMeasured

ST_IsMeasured takes a geometry as an input parameter and returns 1 if the given geometry has M coordinates (measures). Otherwise 0 (zero) is returned.

Related reference

"ST_IsMeasured" on page 154

ST_IsMeasured takes a geometry as an input parameter and returns 1 if the given geometry has M coordinates (measures). Otherwise 0 (zero) is returned.

ST_IsValid

ST_IsValid takes a geometry as an input parameter and returns 1 if it is valid. Otherwise 0 (zero) is returned.

A geometry is valid only if all of the attributes in the structured type are consistent with the internal representation of geometry data, and if the internal representation is not corrupted.

Related reference

"ST_IsValid" on page 157

ST_IsValid takes a geometry as an input parameter and returns 1 if it is valid. Otherwise 0 (zero) is returned.

ST_M

If a measure is stored with a given point, ST_M can take the point as an input parameter and return the measure.

ST_MaxM

ST_MaxM takes a geometry as an input parameter and returns its maximum measure.

ST_MaxX

ST_MaxX takes a geometry as an input parameter and returns its maximum X coordinate.

ST_MaxY

ST_MaxY takes a geometry as an input parameter and returns its maximum Y coordinate.

ST_MaxZ

ST_MaxZ takes a geometry as an input parameter and returns its maximum Z coordinate.

ST_MinM

ST_MinM takes a geometry as an input parameter and returns its minimum measure.

ST_MinX

ST_MinX takes a geometry as an input parameter and returns its minimum X coordinate.

ST_MinY

ST_MinY takes a geometry as an input parameter and returns its minimum Y coordinate.

ST_MinZ

ST_MinY takes a geometry as an input parameter and returns its minimum Z coordinate.

ST_X

ST_X can take a point as an input parameter and return the point's X coordinate.

ST_Y

ST_Y can take a point as an input parameter and return the point's Y coordinate.

ST_Z

If a Z coordinate is stored with a given point, ST_Z can take the point as an input parameter and return the Z coordinate.

Functions that return information about geometries within a geometry

The following functions return information about geometries within a geometry.

ST_Centroid

ST_Centroid takes a geometry as an input parameter and returns the geometric center, which is the center of the minimum bounding rectangle of the given geometry, as a point.

The resulting point is represented in the spatial reference system of the given geometry. If the given geometry is null, then null is returned.

ST_EndPoint

ST_EndPoint takes a linestring as an input parameter and returns the point that is the last point of the linestring.

The resulting point is represented in the spatial reference system of the given linestring. If the given linestring is null or empty, then null is returned.

ST_GeometryN

ST_GeometryN takes a geometry collection and an index as input parameters and returns the geometry in the collection that is identified by the index.

The resulting geometry is represented in the spatial reference system of the given geometry collection. If the given geometry is null or is empty, then null is returned.

ST_NumGeometries

ST_NumGeometries takes a geometry collection as an input parameter and returns the number of geometries in the collection.

If the given geometry collection is null or empty, then null is returned.

ST_NumPoints

ST_NumPoints takes a geometry as an input parameter and returns the number of points that were used to define that geometry.

For example, if the geometry is a polygon and five points were used to define that polygon, then the returned number is 5.

ST_PointN

ST_PointN takes a linestring or a multipoint and an index as input parameters and returns that point in the linestring or multipoint that is identified by the index. The resulting point is represented in the spatial reference system of the given linestring or multipoint.

If the given linestring or multipoint is null or is empty, then null is returned. If the index is smaller than 1 or larger than the number of points in the linestring or multipoint, then null is returned and a warning is returned.

ST_StartPoint

ST_StartPoint takes a linestring as an input parameter and returns the point that is the first point of the linestring.

The resulting point is represented in the spatial reference system of the given linestring. This result is equivalent to the function call ST_PointN(linestring, 1). If the given linestring is null or empty, then null is returned.

Functions that show information about boundaries, envelopes, and rings

You can use certain functions to return information about demarcations that divide an inner part of a geometry from an outer part, or that divide the geometry itself from the space external to it.

For example, ST_Boundary returns a geometry's boundary in the form of a curve.

The following functions show information about boundaries, envelopes, and rings:

- ST_Boundary
- ST_Envelope
- ST_ExteriorRing
- ST_InteriorRingN
- ST_NumInteriorRing

- ST_Perimeter

Functions that return information about a geometry's dimensions

The ST_Area function and the ST_Length function return information about the dimension of a geometry. For example, ST_Area reports how much area a given geometry covers.

ST_Area

ST_Area takes a geometry and, optionally, a unit as input parameters and returns the area covered by the given geometry in the given unit of measure.

ST_Length

ST_Length takes a linestring or multilinestring and, optionally, a unit as input parameters and returns the length of the given linestring or multilinestring in the given unit of measure.

Functions that reveal whether a geometry is closed, empty, or simple

ST_IsClosed shows whether a given linestring or multilinestring is closed. ST_IsEmpty shows whether a given geometry is empty, and ST_IsSimple shows whether a geometry is simple.

A linestring is closed if the start point and end point of the linestring are the same. A geometry is empty when it is devoid of points, and a geometry is simple when its configuration is typical.

ST_IsClosed

ST_IsClosed takes a linestring or multilinestring as an input parameter and returns 1 if the given linestring or multilinestring is closed. Otherwise, 0 (zero) is returned.

A linestring is closed if the start point and end point are equal. If the linestring has Z coordinates, the Z coordinates of the start point and end point must be equal. Otherwise, the points are not considered equal, and the linestring is not closed. A multilinestring is closed if each of its linestrings are closed.

ST_IsEmpty

ST_IsEmpty takes a geometry as an input parameter and returns 1 if the given geometry is empty. Otherwise 0 (zero) is returned.

A geometry is empty if it does not have any points that define it.

ST_IsSimple

ST_IsSimple takes a geometry as an input parameter and returns 1 if the given geometry is simple. Otherwise, 0 (zero) is returned.

Points, polygons, and multipolygons are always simple. A linestring is simple if it does not pass through the same point twice. A multipoint is simple if it does not contain two equal points, and a multilinestring is simple if all of its linestrings are simple and the only intersections occur at points that are on the boundary of the linestrings in the multilinestring.

Function that identifies a geometry's spatial reference system

The function ST_SRID returns values that identify the spatial reference system that is associated with the geometry.

ST_SRID

ST_SRID takes a geometry as the input parameter and returns the spatial reference system identifier from the geometry.

Functions that generate new geometries from existing geometries

Spatial support provides a category of functions that derive new geometries from existing ones.

This category does not include functions that derive geometries that represent properties of other geometries. Rather, it is for functions that:

- Convert geometries into other geometries
- Create geometries that represent configurations of space
- Derive individual geometries from multiple geometries
- Create modifications of geometries

Function that converts one geometry to another

ST_Polygon constructs a polygon from a closed linestring.

ST_Polygon

ST_Polygon can construct a polygon from a closed linestring.

The linestring will define the exterior ring of the polygon.

Functions that create new geometries with different space configurations

Using existing geometries as a starting point, the following functions create new geometries that represent circular areas or other configurations of space.

For example, given a point that represents the center of a proposed airport, ST_Buffer can create a surface that represents, in circular form, the proposed extent of the airport.

ST_Buffer

The ST_Buffer function can generate a new geometry that extends outward from an existing geometry by a specified radius.

The new geometry is a surface when the existing geometry is buffered or whenever the elements of a collection are so close that the buffers around the single elements of the collection overlap. However, when the buffers are separate, individual buffer surfaces result, in which case ST_Buffer returns a multisurface.

The following figure illustrates the buffer around single and overlapped elements.

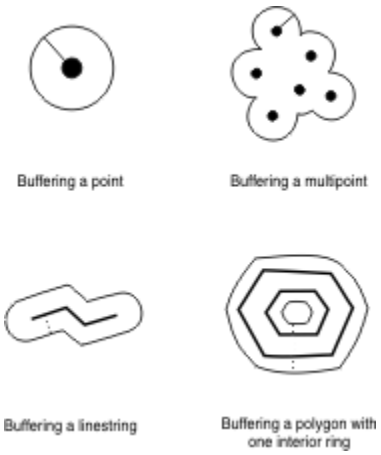


Figure 20. ST_Buffer

The ST_Buffer function accepts both positive and negative distance; however, only geometries with a dimension of two (surfaces and multisurfaces) apply a negative buffer. The absolute value of the buffer distance is used whenever the dimension of the source geometry is less than 2 (all geometries that are not surfaces or multisurfaces).

In general, for exterior rings, positive buffer distances generate surface rings that are away from the center of the source geometry; negative buffer distances generate surface or multisurface rings toward the center. For interior rings of a surface or multisurface, a positive buffer distance generates a buffer ring toward the center, and a negative buffer distance generates a buffer ring away from the center.

The buffering process merges surfaces that overlap. Negative distances greater than one half the maximum interior width of a polygon result in an empty geometry.

ST_ConvexHull
The ST_ConvexHull function returns the convex hull of any geometry that has at least three vertices forming a convex.

Vertices are the pairs of X and Y coordinates within geometries. A *convex hull* is the smallest convex polygon that can be formed by all vertices within a given set of vertices.

The following illustration shows four examples of convex hull. In the first example, an irregular shape resembling the letter c has been drawn. The c is closed by the convex hull. In the fourth example, there are four points with lines in a zig-zag pattern. The convex line goes between points four and two on one side and three and one on the other side.



Figure 21. ST_ConvexHull

ST_Difference
ST_Difference takes two geometries of the same dimension as input and returns that portion of the first geometry that is not intersected by the second geometry.

This operation is the spatial equivalent of the logical operator AND NOT. The portion of geometry returned by ST_Difference is itself a geometry—a collection that has the same dimension as the geometries taken as input. If these two geometries are equal—that is, if they occupy the same space— the returned geometry is empty.

To the left of each arrow are two geometries that are given to ST_Difference as input. To the right of each arrow is the output of ST_Difference. If part of the first geometry is intersected by the second, the output is that part of the first geometry that is not intersected. If the geometries given as input are equal, the output is an empty geometry (denoted by the term nil)

This figure illustrates input and output for ST_Difference. For example, if input is points, and point A and point B are the same, the output is null. If point A and point B are different, the output is a new point between the two. If the input is a polygon for Band a smaller but identical polygon for geometry A inside the first, the outcome is null. If the polygons are overlapping, the output is the outer edges of the combined polygons.

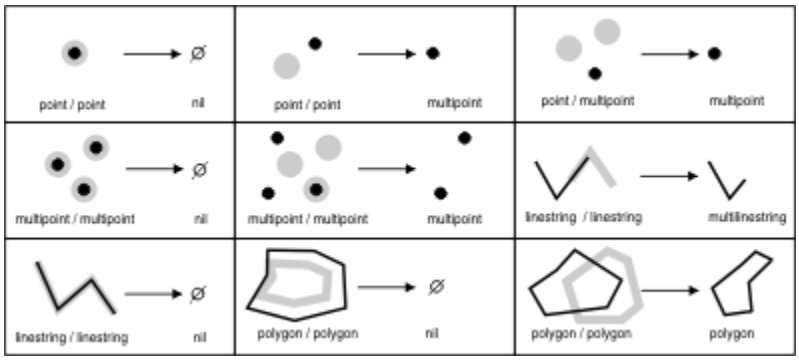


Figure 22. ST_Difference

ST_Intersection

The ST_Intersection function returns a set of points, represented as a geometry, that define the intersection of two given geometries.

If the geometries given to ST_Intersection as input do not intersect, or if they do intersect and the dimension of their intersection is less than the geometries' dimensions, ST_Intersection returns an empty geometry.

This figure illustrates ten examples of output for ST_Intersection, which returns information on where given geometries intersect. To the left of each arrow are two intersecting geometries that are given to ST_Intersection as input. To the right of each arrow is the output of ST_Intersection, which is a geometry that represents the intersection created by the geometries at the left.

For example, if B is a linestring and geometry A is a point on the line, the output is the multipoint where geometry A and geometry B converge. If geometry A and geometry B are overlapping polygons, the output is a new multipolygon of only that portion that overlaps.

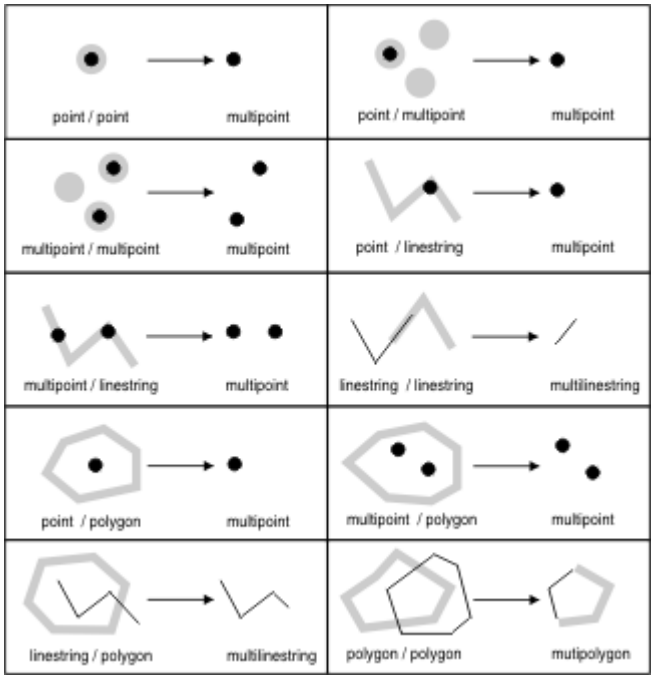


Figure 23. ST_Intersection

ST_SymDifference
The ST_SymDifference function returns the symmetric difference (the spatial equivalent of the logical XOR operation) of two intersecting geometries that have the same dimension.

If these geometries are equal, ST_SymDifference returns an empty geometry. If they are not equal, then a portion of one or both of them lies outside the area of intersection.

Function that derives one geometry from many

Use the ST_Union function to derive individual geometries from multiple geometries. ST_Union combines two geometries into a single geometry.

ST_Union
The ST_Union function returns the union set of two geometries.

This operation is the spatial equivalent of the logical operator OR. The two geometries must be of the same dimension. ST_Union always returns the result as a collection.

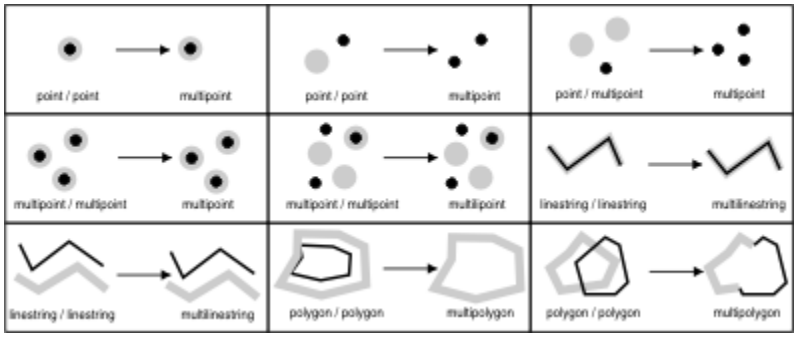


Figure 24. ST_Union

ST_UnionAggr
The ST_UnionAggr function is a union aggregate function that works as a scalar function. This function returns a result for each row.

The result is the union of the geometry on that row and all of the geometries of the previous rows. The result of the final row is the union of all the geometries of that column.

Function that returns distance information

The ST_Distance function takes two geometries and, optionally, a unit as input parameters and returns the shortest distance between any point in the first geometry to any point in the second geometry, measured in the given units.

If the second geometry is not represented in the same spatial reference system as the first geometry, it will be converted to the other spatial reference system.

If any of the two given geometries is null or is empty, then null is returned.

For example, ST_Distance could report the shortest distance an aircraft must travel between two locations. The following figure illustrates this information.

The figure shows a map of the United States with a straight line between points labeled Los Angeles and Chicago.



Figure 25. Minimum distance between two cities

Function that returns index information

ST_GetIndexParms takes either the identifier for a spatial index or for a spatial column as an input parameter and returns the parameters used to define the index or the index on the spatial column.

If an additional parameter number is specified, only the parameter identified by the number is returned.

Chapter 12. Spatial functions: syntax and parameters

This information introduces the spatial functions and discusses certain factors that are common to all or most spatial functions. The functions are documented in alphabetical order.

Considerations for spatial functions

When you code spatial functions, you must specify the schema to which the spatial functions belong.

Before a spatial function can be called, its name must be qualified by the name of the schema to which the spatial functions belong. This schema is DB2GSE. One way to do this is to explicitly specify the schema in the SQL statement that references the function, as in the following example:

```
CREATE TABLE CUSTOMERS ( ..., LOCATION DB2GSE.ST_POINT, ... ) ;
ALTER TABLE BRANCHES ADD COLUMN LOCATION DB2GSE.ST_POINT;
```

Alternatively, to avoid specifying the schema each time a function is to be called, you can add DB2GSE to the CURRENT PATH special register. To obtain the current settings for this special register, use the following SQL statement:

```
 SELECT CURRENT PATH FROM SYSIBM.SYSDUMMY1;
```

To update the CURRENT PATH special register with DB2GSE, use the following SQL statement:

```
set CURRENT PATH = CURRENT PATH, db2gse;
```

EnvelopesIntersect

The EnvelopesIntersect spatial function accepts two types of input parameters to determine if the minimum bounding rectangles of two geometries intersect.

This function accepts the following two types of input parameters:

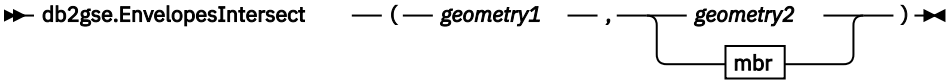
- Two geometries

  EnvelopesIntersect returns 1 if the envelope of the first geometry intersects the envelope of the second geometry. Otherwise, 0 (zero) is returned.
- A geometry, four type DOUBLE coordinate values that define the lower-left and upper-right corners of a rectangular window, and the spatial reference system identifier.

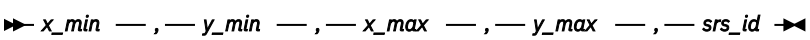
  EnvelopesIntersect returns 1 if the envelope of the first geometry intersects with the envelope defined by the four type DOUBLE values. Otherwise, 0 (zero) is returned.

If one of the input geometries is empty or null, null is returned.

Syntax



mbr



© Copyright IBM Corp. 2007, 2013

Parameters

geometry1
One of the seven distinct spatial data types that represents the geometry whose envelope is to be tested for intersection with the envelope of either *geometry2* or the minimum bounding rectangle (MBR) defined by the four type DOUBLE values.

geometry2
One of the seven distinct spatial data types that represents the geometry whose envelope is to be tested for intersection with the envelope of *geometry1*.

x_min
Specifies the minimum X coordinate value for the envelope. You must specify a non-null value for this parameter.

The data type of this parameter is DOUBLE.

y_min
Specifies the minimum Y coordinate value for the envelope. You must specify a non-null value for this parameter.

The data type of this parameter is DOUBLE.

x_max
Specifies the maximum X coordinate value for the envelope. You must specify a non-null value for this parameter.

The data type of this parameter is DOUBLE.

y_max
Specifies the maximum Y coordinate value for the envelope. You must specify a non-null value for this parameter.

The data type of this parameter is DOUBLE.

srs_id
Uniquely identifies the spatial reference system. The spatial reference system identifier must match the spatial reference system identifier of the geometry parameter. You must specify a non-null value for this parameter.

The data type of this parameter is INTEGER.

Return type

INTEGER

Example

This example creates two polygons that represent counties and then determines if any of them intersect a geographic area specified by the four type DOUBLE values.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE counties (id INTEGER, name CHAR(20), geometry ST_Polygon);

INSERT INTO counties VALUES
       (1, 'County_1', ST_Polygon('polygon((0 0, 30 0, 40 30, 40 35,
        5 35, 5 10, 20 10, 20 5, 0 0))' ,0));

INSERT INTO counties VALUES
       (2, 'County_2', ST_Polygon('polygon((15 15, 15 20, 60 20, 60 15,
        15 15))' ,0));

INSERT INTO counties VALUES
       (3, 'County_3', ST_Polygon('polygon((115 15, 115 20, 160 20, 160 15,
        115 15))' ,0));


SELECT name
FROM counties as c
```

```
WHERE EnvelopesIntersect(c.geometry, 15, 15, 60, 20, 0) =1;
```

Results:

```
Name
--------------------
County_1
County_2
```

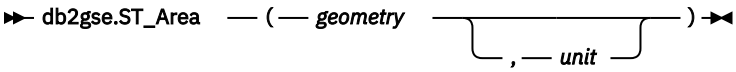
ST_Area

ST_Area takes a geometry and, optionally, a unit as input parameters and returns the area covered by the geometry in either the default or given unit of measure.

If the geometry is a polygon or multipolygon, then the area covered by the geometry is returned. The area of points, linestrings, multipoints, and multilinestrings is 0 (zero).

If the geometry is null or is an empty geometry, null is returned.

Syntax



Parameters

geometry
A value of one of the seven distinct spatial data types that represents the geometry that determines the area.

unit
A VARCHAR(128) value that identifies the units in which the area is measured. The supported units of measure are listed in the DB2GSE.ST_UNITS_OF_MEASURE catalog view.

If the *unit* parameter is omitted, the following rules are used to determine the unit in which the area is measured:

- If *geometry* is in a projected or geocentric coordinate system, the linear unit associated with this coordinate system is used.
- If *geometry* is in a geographic coordinate system, the angular unit associated with this coordinate system is used.

Restrictions on unit conversions: An error (SQLSTATE 38SU4) is returned if any of the following conditions occur:

- The geometry is in an unspecified coordinate system and the *unit* parameter is specified.
- The geometry is in a projected coordinate system and an angular unit is specified.
- The geometry is in a geographic coordinate system, and a linear unit is specified.

Return type

DOUBLE

Examples

Example 1

The spatial analyst needs a list of the area covered by each sales region. The sales region polygons are stored in the SAMPLE_POLYGONS table. The area is calculated by applying the ST_Area function to the geometry column.

```
DSN5SCLP /create_srs STLEC1  -srsId 4000 -srsName new_york1983 -xOffset 0
    -yOffset 0 -xScale 1 -yScale 1
    -coordsysName NAD_1983_StatePlane_New_York_East_FIPS_3101_Feet

SET current path db2gse;
CREATE TABLE sample_polygons (id INTEGER, geometry ST_POLYGON);

INSERT INTO sample_polygons (id, geometry)
VALUES
  (1, ST_Polygon('polygon((0 0, 0 10, 10 10, 10 0, 0 0))', 4000) );

INSERT INTO sample_polygons (id, geometry)
VALUES
  (2, ST_Polygon('polygon((20 0, 30 20, 40 0, 20 0 ))', 4000) );

INSERT INTO sample_polygons (id, geometry)
VALUES
  (3, ST_Polygon('polygon((20 30, 25 35, 30 30, 20 30))', 4000));
```

The following SELECT statement retrieves the sales region ID and area:

```
SELECT id, ST_Area(geometry) AS area
FROM   sample_polygons;
```

Results:

```
ID        AREA
-------- ------------------------
       1   +1.00000000000000E+002
       2   +2.00000000000000E+002
       3   +2.50000000000000E+001
```

Example 2

The following SELECT statement retrieves the sales region ID and area in various units:

```
SELECT id,
       ST_Area(geometry) square_feet,
       ST_Area(geometry, 'METER') square_meters,
       ST_Area(geometry, 'STATUTE MILE') square_miles
FROM   sample_polygons;
```

Results:

```
ID  SQUARE_FEET              SQUARE_METERS            SQUARE_MILES
--- ------------------------ ------------------------ ------------------------
  1 +1.00000000000000E+002   +9.29034116132748E+000   +3.58702077598427E-006
  2 +2.00000000000000E+002   +1.85806823226550E+001   +7.17404155196855E-006
  3 +2.50000000000000E+001   +2.32258529033187E+000   +8.96755193996069E-007
```

Example 3

This example finds the area of a polygon defined in State Plane coordinates.

The State Plane spatial reference system with an ID of 3 is created with the following command:

```
DSN5SCLP /create_srs SAMP_DB  -srsId 3 -srsName z3101a -xOffset 0
```

```
    -yOffset 0 -xScale 1 -yScale 1
    -coordsysName NAD_1983_StatePlane_New_York_East_FIPS_3101_Feet
```

The following SQL statements add the polygon, in spatial reference system 3, to the table and determines the area in square feet, square meters, and square miles.

```
SET current path db2gse;
 CREATE TABLE Sample_Poly3 (id integer, geometry ST_Polygon);
 INSERT into Sample_Poly3 VALUES
    (1, ST_Polygon('polygon((567176.0 1166411.0,
                             567176.0 1177640.0,
                             637948.0 1177640.0,
                             637948.0 1166411.0,
                             567176.0 1166411.0 ))', 3));
 SELECT id, ST_Area(geometry) "Square Feet",
            ST_Area(geometry, 'METER') "Square Meters",
            ST_Area(geometry, 'STATUTE MILE') "Square Miles"
 FROM Sample_Poly3;
```

Results:

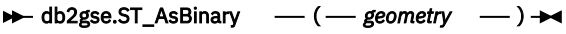
```
ID   Square Feet              Square Meters            Square Miles
--- ----------------------- ----------------------- -----------------------
  1 +7.94698788000000E+008   +7.38302286101346E+007   +2.85060106320552E+001
```

ST_AsBinary

ST_AsBinary takes a geometry as an input parameter and returns its well-known binary representation. The Z and M coordinates are discarded and will not be represented in the well-known binary representation.

If the given geometry is null, then null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types to be converted to the corresponding well-known binary representation.

Return type

BLOB(4M)

Examples

Example 1

The following code illustrates how to use the ST_AsBinary function to convert the points in the geometry columns of the SAMPLE_POINTS table into well-known binary (WKB) representation in the BLOB column.

```
SET CURRENT PATH = CURRENT PATH, db2gse;

CREATE TABLE SAMPLE_POINTS (id integer, geometry ST_POINT, wkb BLOB(32K));

INSERT INTO SAMPLE_POINTS (id, geometry)
VALUES
    (1100, ST_Point(10, 20, 1));
```

Example 2

This example populates the WKB column, with an ID of 1111, from the GEOMETRY column, with an ID of 1100.

```
INSERT INTO sample_points(id, wkb)
VALUES (1111,
  (SELECT  ST_AsBinary(geometry)
   FROM    sample_points
   WHERE   id = 1100));

SELECT id, cast(ST_AsText(ST_Point(wkb,1)) AS varchar(35)) AS point
FROM   sample_points
WHERE  id = 1111;
```

Results:

```
ID          Point
----------- -----------------------------------
       1111 POINT ( 10.00000000 20.00000000)
```

Example 3

This example displays the WKB binary representation.

```
SELECT id, HEX(substr(ST_AsBinary(geometry), 1, 21)) AS point_wkb
FROM   sample_points
WHERE  id = 1100;
```

Results:

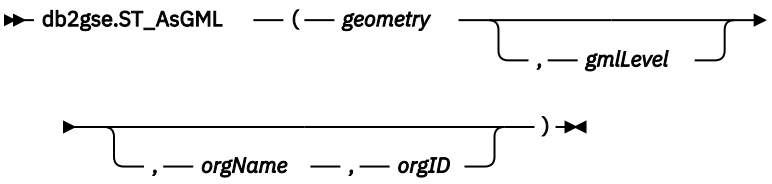
```
ID     POINT_WKB
------ ---------------------------------------------
  1100 00000000014024000000000000403400000000000000
```

ST_AsGML

ST_AsGML takes a geometry as an input parameter and returns its representation using the Geography Markup Language (GML).

If the given geometry is null, then null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types that represents the geometry that is to be converted to the corresponding GML representation.

gmlLevel
Specifies an integer that represents the GML level that is being used. The supported values are 2 and 3.

orgName
Specifies the organization name for the coordinate system that is being used.

orgID
Specifies the organization ID for the coordinate system that is being used.

Return type

CLOB(8M)

Example

In the following example, the lines of results have been reformatted for readability. The spacing in your results will vary according to your online display.

The following code fragment illustrates how to use the ST_AsGML function to view the GML fragment. This example populates the GML column, from the geometry column, with an ID of 2222.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse

CREATE TABLE SAMPLE_POINTS (id integer, geometry ST_POINT, gml CLOB(32K))

INSERT INTO SAMPLE_POINTS (id, geometry)
VALUES
    (1100, ST_Point(10, 20, 1))

INSERT INTO sample_points(id, gml)
VALUES (2222,
  (SELECT  ST_AsGML(geometry, 'EPSG', 4269)
   FROM    sample_points
   WHERE   id = 1100))
```

The following SELECT statement lists the ID and the GML representation of the geometries. The geometry is converted to a GML fragment by the ST_AsGML function.

```
SELECT id, cast(ST_AsGML(geometry, 'EPSG', 4269)
AS varchar(110)) AS gml_fragment
FROM   sample_points
WHERE  id = 1100
```

Results:

```
The SELECT statement returns the following result set:

ID          GML_FRAGMENT

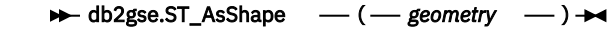
----------- ------------------------------------------------------------
       1100 <gml:Point srsName="EPSG:4269"><gml:coord>
            <gml:X>10</gml:X><gml:Y>20</gml:Y>
            </gml:coord></gml:Point>
```

ST_AsShape

St_AsShape takes a geometry as an input parameter and returns its ESRI shape representation.

If the given geometry is null, then null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types to be converted to the corresponding ESRI shape representation.

Return type

BLOB(4M)

Example

The following code fragment illustrates how to use the ST_AsShape function to convert the points in the geometry column of the SAMPLE_POINTS table into shape binary representation in the shape BLOB column. This example populates the shape column from the geometry column. The shape binary representation is used to display the geometries in geobrowsers, which require geometries to comply with the ESRI shapefile format, or to construct the geometries for the *.SHP file of the shape file.

```
SET CURRENT PATH = CURRENT PATH, db2gse;

CREATE TABLE SAMPLE_POINTS (id integer, geometry ST_POINT, shape BLOB(32K));

INSERT INTO SAMPLE_POINTS (id, geometry)
VALUES
    (1100, ST_Point(10, 20, 1));

INSERT INTO sample_points(id, shape)
VALUES (2222,
  (SELECT  ST_AsShape(geometry)
   FROM    sample_points
   WHERE   id = 1100));

SELECT id, HEX(substr(ST_AsShape(geometry), 1, 20)) AS shape
FROM   sample_points
WHERE  id = 1100;
```

Returns:

```
ID     SHAPE

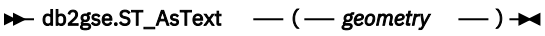
------ ----------------------------------------
  1100 01000000000000000000244000000000000003440
```

ST_AsText

ST_AsText takes a geometry as an input parameter and returns its well-known text representation.

If the given geometry is null, then null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types to be converted to the corresponding well-known text representation.

Return type

CLOB(8M)

Example

In the following example, the lines of results have been reformatted for readability.

After capturing and inserting the data into the SAMPLE_GEOMETRIES table, an analyst wants to verify that the values inserted are correct by looking at the well-known text representation of the geometries.

```
SET CURRENT PATH = CURRENT PATH, db2gse;

CREATE TABLE sample_geometries(id SMALLINT, spatial_type varchar(18),
    geometry ST_GEOMETRY);

INSERT INTO sample_geometries(id, spatial_type, geometry)
VALUES
    (1, 'st_point', ST_GEOMETRY(ST_Point(50,50,1)));

INSERT INTO sample_geometries(id, spatial_type, geometry)
VALUES
    (2, 'st_linestring', ST_GEOMETRY(ST_LineString('linestring
        (200 100, 210 130, 220 140)',  1)));

INSERT INTO sample_geometries(id, spatial_type, geometry)
VALUES
    (3, 'st_polygon', ST_GEOMETRY(ST_Polygon('polygon((110 120, 110 140,
        130 140, 130 120, 110 120))', 1)));
```

The following SELECT statement lists the spatial type and the WKT representation of the geometries. The geometry is converted to text by the ST_AsText function. It is then cast to a varchar(150), because the default output of the ST_AsText function is CLOB(8M).

```
SELECT id, spatial_type, cast(ST_AsText(geometry)
   AS varchar(150)) AS wkt
FROM   sample_geometries
```

Results:

```
ID     SPATIAL_TYPE       WKT
------ ------------------ -------------------------------------------------
     1 st_point           POINT ( 50.000000 50.000000)

     2 st_linestring      LINESTRING ( 200.000000 100.000000,
                            210.000000 130.000000, 220.000000
                            140.000000)

     3 st_polygon         POLYGON (( 110.000000 120.000000,
                            130.000000 120.000000, 130.000000
                            140.000000, 110.000000 140.000000,
                            110.000000 120.000000))
```

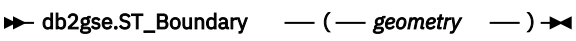
ST_Boundary

ST_Boundary takes a geometry as an input parameter and returns its boundary as a new geometry. The resulting geometry is represented in the spatial reference system of the given geometry.

If the given geometry is a point, multipoint, closed linestring, or closed multilinestring, or if it is empty, then the result is an empty geometry of type ST_Point. For linestrings or multilinestrings that are not closed, the start points and end points of the linestrings are returned as an ST_MultiPoint value, unless such a point is the start or end point of an even number of linestrings. For polygons and multipolygons, the linestring defining the boundary of the given geometry is returned, either as an ST_LineString or an ST_MultiLineString value. If the given geometry is null, then null is returned.

If possible, the specific type of the returned geometry will be ST_Point, ST_LineString, or ST_Polygon. For example, the boundary of a polygon with no holes is a single linestring, represented as ST_LineString. The boundary of a polygon with one or more holes consists of multiple linestrings, represented as ST_MultiLineString.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types that represents the geometry. The boundary of this geometry is returned.

Return type

db2gse.ST_Geometry

Example

In the following example, the lines of results have been reformatted for readability. The spacing in your results will vary according to your online display.

This example creates several geometries and determines the boundary of each geometry.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_geoms (id INTEGER, geometry ST_Geometry)

INSERT INTO sample_geoms VALUES
       (1, ST_Polygon('polygon((40 120, 90 120, 90 150, 40 150, 40 120))', 0))

INSERT INTO sample_geoms VALUES
       (2, ST_Polygon('polygon((40 120, 90 120, 90 150, 40 150, 40 120),
                               (70 130, 80 130, 80 140, 70 140, 70 130))' ,0))

INSERT INTO sample_geoms VALUES
       (3, ST_Geometry('linestring(60 60, 65 60, 65 70, 70 70)' ,0))

INSERT INTO sample_geoms VALUES
       (4, ST_Geometry('multilinestring((60 60, 65 60, 65 70, 70 70),
                                        (80 80, 85 80, 85 90, 90 90),
                                        (50 50, 55 50, 55 60, 60 60))' ,0))

INSERT INTO sample_geoms VALUES
       (5, ST_Geometry('point(30 30)' ,0))

SELECT id, CAST(ST_AsText(ST_Boundary(geometry)) as VARCHAR(320)) Boundary
FROM   sample_geoms

Results
ID      BOUNDARY
------- ------------------------------------------------------------------------
      1 LINESTRING ( 40.00000000 120.00000000, 90.00000000 120.00000000,
          90.00000000 150.00000000, 40.00000000 150.00000000, 40.00000000
          120.00000000)

      2 MULTILINESTRING (( 40.00000000 120.00000000, 90.00000000 120.00000000,
          90.00000000 150.00000000, 40.00000000 150.00000000, 40.00000000
          120.00000000),( 70.00000000 130.00000000, 70.00000000 140.00000000,
          80.00000000 140.00000000, 80.00000000 130.00000000, 70.00000000
          130.00000000))

      3 MULTIPOINT ( 60.00000000 60.00000000, 70.00000000 70.00000000)

      4 MULTIPOINT ( 50.00000000 50.00000000, 70.00000000 70.00000000,
          80.00000000 80.00000000, 90.00000000 90.00000000)

      5 POINT EMPTY
```

ST_Buffer

ST_Buffer takes a geometry, a distance, and, optionally, a unit as input parameters and returns the geometry that surrounds the given geometry by the specified distance, measured in the given unit.

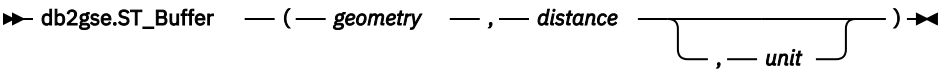
Each point on the boundary of the resulting geometry is the specified distance away from the given geometry. The resulting geometry is represented in the spatial reference system of the given geometry.

Any circular curve in the boundary of the resulting geometry is approximated by linear strings. For example, the buffer around a point, which would result in a circular region, is approximated by a polygon whose boundary is a linestring.

If the given geometry is null or is empty, null will be returned.

Note: After you apply APAR PM92224, the results that are returned by the ST_Buffer function might be different than before the APAR was applied.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types that represents the geometry to create the buffer around.

distance
A DOUBLE PRECISION value that specifies the distance to be used for the buffer around *geometry*.

unit
A VARCHAR(128) value that identifies the unit in which *distance* is measured. The supported units of measure are listed in the DB2GSE.ST_UNITS_OF_MEASURE catalog view.

If the *unit* parameter is omitted, the following rules are used to determine the unit of measure that is used for distance:

- If *geometry* is in a projected or geocentric coordinate system, the linear unit associated with this coordinate system is the default.
- If *geometry* is in a geographic coordinate system, the angular unit associated with this coordinate system is the default.
- If *geometry* is in a geographic coordinate system, and a linear unit is specified, the geometry type must be ST_Point. If the distance is shorter than 1 meter, ST_Buffer regards the distance as 1 meter.

Restrictions on unit conversions: An error (SQLSTATE 38SU4) is returned if any of the following conditions occur:

- The geometry is in an unspecified coordinate system and the *unit* parameter is specified.
- The geometry is in a projected coordinate system and an angular unit is specified.
- The geometry in the geographic coordinate system is not ST_Point, and a linear unit is specified.

Return type

db2gse.ST_Geometry

Examples

In the following examples, the results have been reformatted for readability.

Example 1

The following code creates a spatial reference system, creates the SAMPLE_GEOMETRIES table, and populates it.

```
DSN5SCLP /create_srs STLEC1  -srsId 4000 -srsName new_york1983
    -xOffset 0 -yOffset 0 -xScale 1 -yScale 1
    -coordsysName NAD_1983_StatePlane_New_York_East_FIPS_3101_Feet

SET CURRENT PATH = CURRENT PATH, db2gse;

CREATE TABLE
    sample_geometries (id INTEGER, spatial_type varchar(18),
    geometry ST_GEOMETRY);

INSERT INTO sample_geometries(id, spatial_type, geometry)
VALUES
    (1, 'st_point', ST_GEOMETRY(ST_Point(50, 50, 4000)));

INSERT INTO sample_geometries(id, spatial_type, geometry)
VALUES
    (2, 'st_linestring',
         ST_GEOMETRY(ST_LineString('linestring(200 100, 210 130,
         220 140)', 4000)));

INSERT INTO sample_geometries(id, spatial_type, geometry)
VALUES
    (3, 'st_polygon',
         ST_GEOMETRY(ST_Polygon('polygon((110 120, 110 140, 130 140,
         130 120, 110 120))',4000)));

INSERT INTO sample_geometries(id, spatial_type, geometry)
VALUES
    (4, 'st_multipolygon',
         ST_GEOMETRY(ST_MultiPolygon('multipolygon(((30 30, 30 40,
         35 40, 35 30, 30 30),(35 30, 35 40, 45 40,
         45 30, 35 30)))', 4000)));
```

Example 2

The following SELECT statement uses the ST_Buffer function to apply a buffer of 10.

```
SELECT id, spatial_type,
       cast(ST_AsText(ST_Buffer(geometry, 10)) AS varchar(470)) AS buffer_10
FROM   sample_geometries
```

Results:

```
ID          SPATIAL_TYPE       BUFFER_10
----------- ------------------ ----------------------------------------
1           st_point           POLYGON (( 60 50, 59 55, 54 59, 49
   60, 44 58, 41 53, 40 48, 42 43, 47 41, 52 40, 57 42, 60 50))

2             st_linestring    POLYGON (( 230 140, 229 145, 224 149,
     219 150, 213 147, 203 137, 201 133, 191 103, 191 99, 192 95,
   196 91, 200 91, 204 91, 209 97, 218 124, 227 133, 230 140))

3              st_polygon         POLYGON (( 140 120, 140 140, 139 145,
     130 150, 110 150, 105 149, 100 140, 100 120, 101  115, 110 110,
     130 110, 135 111, 140 120))

4              st_multipolygon    POLYGON (( 55 30, 55 40, 54 45, 45
   50, 30 50, 25 49, 20 40, 20 30, 21 25, 30 20, 45 20, 50 21, 55 30))
```

Example 3

The following SELECT statement uses the ST_Buffer function to apply a negative buffer of 5.

```
SELECT id, spatial_type,
       cast(ST_AsText(ST_Buffer(geometry, -5)) AS varchar(150))
       AS buffer_negative_5
```

```
FROM   sample_geometries
WHERE  id = 3
```

Results:

```
ID          SPATIAL_TYPE       BUFFER_NEGATIVE_5
----------- ------------------ ------------------------------------
3           st_polygon         POLYGON (( 115 125, 125 125, 125 135,
     115 135, 115 125))
```

Example 4

The following SELECT statement shows the result of applying a buffer with the unit parameter specified.

```
SELECT id, spatial_type,
    cast(ST_AsText(ST_Buffer(geometry, 10, 'METER')) AS varchar(680))
    AS buffer_10_meter
FROM   sample_geometries
WHERE  id = 3
```

Results:

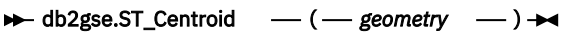
```
ID          SPATIAL_TYPE       BUFFER_10_METER
----------- ------------------ ------------------------------------
3           st_polygon         POLYGON (( 163 120, 163 140, 162 149,
     159 157, 152 165, 143 170, 130 173, 110 173, 101 172, 92 167,
     84 160, 79 151, 77 140, 77 120, 78 111, 83 102, 90 94, 99 89,
     110 87, 130 87, 139 88, 147 91, 155 98, 160 107, 163 120))
```

ST_Centroid

ST_Centroid takes a geometry as an input parameter and returns the geometric center, which is the center of the minimum bounding rectangle of the given geometry, as a point. The resulting point is represented in the spatial reference system of the given geometry.

If the given geometry is null or is empty, then null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types that represents the geometry to determine the geometric center.

Return type

db2gse.ST_Point

Example

This example creates two geometries and finds the centroid of them.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_geoms (id INTEGER, geometry ST_Geometry)

INSERT INTO sample_geoms VALUES
    (1, ST_Polygon('polygon
```

```
    ((40 120, 90 120, 90 150, 40 150, 40 120),
    (50 130, 80 130, 80 140, 50 140, 50 130))',0))

INSERT INTO sample_geoms VALUES
    (2, ST_MultiPoint('multipoint(10 10, 50 10, 10 30)' ,0))


SELECT id, CAST(ST_AsText(ST_Centroid(geometry))
   as VARCHAR(40)) Centroid
FROM sample_geoms
```

Results:

```
ID          CENTROID
----------- ----------------------------------------
          1 POINT ( 65.00000000 135.00000000)
          2 POINT ( 30.00000000 20.00000000)
```

ST_Contains

ST_Contains takes two geometries as input parameter and returns 1 if the first geometry completely contains the second; otherwise it returns 0 (zero) to indicate that the first geometry does not completely contain the second.

If any of the given geometries is null or is empty, then null is returned.

If the second geometry is not represented in the same spatial reference system as the first geometry, it will be converted to the other spatial reference system.

Syntax



Parameter

geometry1
A value of one of the seven distinct spatial data types that represents the geometry that is to be tested to completely contain *geometry2*.

geometry2
A value of one of the seven distinct spatial data types that represents the geometry that is to be tested to be completely within *geometry1*.

Return type

INTEGER

Examples

Example 1

The following code creates and populates these tables.

```
SET CURRENT PATH = CURRENT PATH, db2gse;

CREATE TABLE sample_points(id SMALLINT, geometry ST_POINT);

CREATE TABLE sample_lines(id SMALLINT, geometry ST_LINESTRING);

CREATE TABLE sample_polygons(id SMALLINT, geometry ST_POLYGON);

INSERT INTO sample_points (id, geometry)
VALUES
    (1, ST_Point(10, 20, 1));
```

```
INSERT INTO sample_points (id, geometry)
VALUES
    (2, ST_Point('point(41 41)', 1));

INSERT INTO sample_lines (id, geometry)
VALUES
    (10, ST_LineString('linestring (1 10, 3 12, 10 10)', 1) );

INSERT INTO sample_lines (id, geometry)
VALUES
    (20, ST_LineString('linestring (50 10, 50 12, 45 10)', 1) );

INSERT INTO sample_polygons(id, geometry)
VALUES
      (100, ST_Polygon('polygon((0 0, 0 40, 40 40, 40 0, 0 0))',  1) );
```

Example 2

The following code fragment uses the ST_Contains function to determine which points are contained by a particular polygon.

```
SELECT poly.id AS polygon_id,
       CASE ST_Contains(poly.geometry, pts.geometry)
          WHEN 0 THEN 'does not contain'
          WHEN 1 THEN 'does contain'
       END AS contains,
       pts.id AS point_id
FROM   sample_points pts, sample_polygons poly;
```

Results:

```
POLYGON_ID CONTAINS         POINT_ID
---------- ---------------- --------
       100 does contain            1
       100 does not contain        2
```

Example 3

The following code fragment uses the ST_Contains function to determine which lines are contained by a particular polygon.

```
SELECT poly.id AS polygon_id,
       CASE ST_Contains(poly.geometry, line.geometry)
          WHEN 0 THEN 'does not contain'
          WHEN 1 THEN 'does contain'
       END AS contains,
       line.id AS line_id
FROM   sample_lines line, sample_polygons poly;
```

Results:

```
POLYGON_ID CONTAINS         LINE_ID
---------- ---------------- -------
       100 does contain          10
       100 does not contain      20
```

ST_ConvexHull

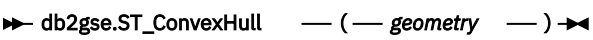
ST_ConvexHull takes a geometry as an input parameter and returns the convex hull of it.

The resulting geometry is represented in the spatial reference system of the given geometry.

If possible, the specific type of the returned geometry will be ST_Point, ST_LineString, or ST_Polygon. For example, the boundary of a polygon with no holes is a single linestring, represented as ST_LineString. The boundary of a polygon with one or more holes consists of multiple linestrings, represented as ST_MultiLineString.

If the given geometry is null or is empty, then null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types that represents the geometry to compute the convex hull.

Return type

db2gse.ST_Geometry

Example

In the following example, the lines of results have been reformatted for readability. The spacing in your results will vary according to your online display.

The following code creates and populates the SAMPLE_GEOMETRIES table.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse

CREATE TABLE sample_geometries(id INTEGER, spatial_type varchar(18),
   geometry ST_GEOMETRY)

INSERT INTO sample_geometries(id, spatial_type, geometry)
  VALUES (1, 'ST_LineString', ST_LineString
        ('linestring(20 20, 30 30, 20 40, 30 50)', 0)),
INSERT INTO sample_geometries(id, spatial_type, geometry)
  VALUES (2, 'ST_Polygon', ST_Polygon('polygon
        ((110  120,  110 140, 120 130, 110 120))', 0) ),
INSERT INTO sample_geometries(id, spatial_type, geometry)
  VALUES (3, 'ST_Polygon', ST_Polygon('polygon((30 30, 25 35, 15 50,
        35 80, 40 85, 80 90,70 75, 65 70, 55 50, 75 40, 60 30,
        30 30))', 0) ),
INSERT INTO sample_geometries(id, spatial_type, geometry)
  VALUES (4, 'ST_MultiPoint', ST_MultiPoint('multipoint(20 20, 30 30,
        20 40, 30 50)', 1))
```

The following SELECT statement calculates the convex hull for all of the geometries that are constructed above and displays the result.

```
SELECT id, spatial_type, cast( ST_AsText( ST_ConvexHull(g))
AS varchar(300)) "convexhull"
FROM sample_geometries
```

Results:

```
ID     SPATIAL_TYPE       CONVEXHULL
----- ------------------ ----------------------------------------------------
    1 ST_LineString      POLYGON (( 20.00000000 40.00000000,
                           20.00000000 20.00000000, 30.00000000
                           30.00000000, 30.00000000 50.00000000,
                           20.00000000 40.00000000))

    2 ST_Polygon         POLYGON (( 110.00000000 140.00000000,
                           110.00000000 120.00000000, 120.00000000
```

```
                          130.00000000,  110.00000000 140.00000000))

    3 ST_Polygon         POLYGON (( 15.00000000 50.00000000,
                          25.00000000 35.00000000, 30.00000000
                          30.00000000, 60.00000000 30.00000000,
                          75.00000000 40.00000000, 80.00000000
                          90.00000000, 40.00000000 85.00000000,
                          35.00000000 80.00000000, 15.00000000
                          50.00000000))

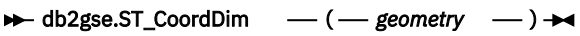
    4 ST_MultiPoint      POLYGON (( 20.00000000 40.00000000,
                          20.00000000 20.00000000, 30.00000000
                          30.00000000, 30.00000000 50.00000000,
                          20.00000000 40.00000000))
```

ST_CoordDim

ST_CoordDim takes a geometry as an input parameter and returns the dimensionality of its coordinates.

If the given geometry does not have Z and M coordinates, the dimensionality is 2. If the given geometry has Z coordinates and no M coordinates, or if it has M coordinates and no Z coordinates, the dimensionality is 3. If it has Z and M coordinates, the dimensionality is 4. If the geometry is null, then null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types that represents the geometry to retrieve the dimensionality from.

Return type

INTEGER

Example

The following example creates several geometries and then determines the dimensionality of their coordinates.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_geoms (id CHARACTER(15), geometry ST_Geometry)

INSERT INTO sample_geoms VALUES
       ('Empty Point', ST_Geometry('point EMPTY',0))

INSERT INTO sample_geoms VALUES
       ('Linestring', ST_Geometry('linestring (10 10, 15 20)',0))

INSERT INTO sample_geoms VALUES
       ('Polygon', ST_Geometry('polygon((40 120, 90 120, 90 150,
         40 150, 40 120))' ,0))

INSERT INTO sample_geoms VALUES
       ('Multipoint M', ST_Geometry('multipoint m (10 10 5, 50 10
         6, 10 30 8)' ,0))

INSERT INTO sample_geoms VALUES
       ('Multipoint Z', ST_Geometry('multipoint z (47 34 295,
         23 45 678)' ,0))

INSERT INTO sample_geoms VALUES
       ('Point ZM',     ST_Geometry('point zm (10 10 16 30)' ,0))


SELECT id, ST_CoordDim(geometry)  COORDDIM
```

```
FROM sample_geoms
```

Results:

```
ID              COORDDIM
--------------- -----------
Empty Point               2
Linestring                2
Polygon                   2
Multipoint M              3
Multipoint Z              3
Point ZM                  4
```

ST_Crosses

ST_Crosses takes two geometries as input parameters and returns 1 if the first geometry crosses the second. Otherwise, 0 (zero) is returned.

If the second geometry is not represented in the same spatial reference system as the first geometry, it will be converted to the other spatial reference system.

If the first geometry is a polygon or a multipolygon, or if the second geometry is a point or multipoint, or if any of the geometries is null value or is empty, then null is returned. If the intersection of the two geometries results in a geometry that has a dimension that is one less than the maximum dimension of the two given geometries, and if the resulting geometry is not equal any of the two given geometries, then 1 is returned. Otherwise, the result is 0 (zero).

Syntax



Parameter

geometry1
A value of one of the seven distinct spatial data types that represents the geometry that is to be tested for crossing *geometry2*.

geometry2
A value of one of the seven distinct spatial data types that represents the geometry that is to be tested to determine if it is crossed by *geometry1*.

Return Type

INTEGER

Example

This code determines if the constructed geometries cross each other.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_geoms (id INTEGER, geometry ST_Geometry);

INSERT INTO sample_geoms VALUES
       (1, ST_Geometry(ST_Polygon('polygon((30 30, 30 50, 50 50, 50 30,
                      30 30))' ,0)));

INSERT INTO sample_geoms VALUES
       (2, ST_Geometry(ST_Linestring('linestring(40 50, 50 40)' ,0)));

INSERT INTO sample_geoms VALUES
       (3, ST_Geometry(ST_Linestring('linestring(20 20, 60 60)' ,0)));

SELECT a.id, b.id, ST_Crosses(a.geometry, b.geometry) Crosses
```

```
FROM   sample_geoms a, sample_geoms b;
```

Results:

```
ID          ID          CROSSES
----------- ----------- -----------
          1           1           -
          2           1           0
          3           1           1
          1           2           -
          2           2           0
          3           2           1
          1           3           -
          2           3           1
          3           3           0
```

ST_Difference

ST_Difference takes two geometries as input parameters and returns the part of the first geometry that does not intersect with the second geometry.

Both geometries must be of the same dimension. If either geometry is null, null is returned. If the first geometry is empty, an empty geometry of type ST_Point is returned. If the second geometry is empty, then the first geometry is returned unchanged.

If the second geometry is not represented in the same spatial reference system as the first geometry, it will be converted to the other spatial reference system.

Syntax



Parameter

geometry1
A value of one of the seven distinct spatial data types that represents the first geometry to use to compute the difference to *geometry2*.

geometry2
A value of one of the seven distinct spatial data types that represents the second geometry that is used to compute the difference to *geometry1*.

Return type

db2gse.ST_Geometry

The dimension of the returned geometry is the same as that of the input geometries.

Examples

In the following examples, the results have been reformatted for readability. The spacing in your results will vary according to your display.

Example 1

The following example creates and populates the SAMPLE_GEOMETRIES table.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_geoms (id INTEGER, geometry ST_Geometry)
```

```
INSERT INTO sample_geoms VALUES
       (1, ST_Geometry('polygon((10 10, 10 20, 20 20, 20 10, 10 10))' ,0))

INSERT INTO sample_geoms VALUES
       (2, ST_Geometry('polygon((30 30, 30 50, 50 50, 50 30, 30 30))' ,0))

INSERT INTO sample_geoms VALUES
       (3, ST_Geometry('polygon((40 40, 40 60, 60 60, 60 40, 40 40))' ,0))

INSERT INTO sample_geoms VALUES
       (4, ST_Geometry('linestring(70 70, 80 80)' ,0))

INSERT INTO sample_geoms VALUES
       (5, ST_Geometry('linestring(75 75, 90 90)' ,0))
```

Example 2

This example finds the difference between two disjoint polygons.

```
SELECT a.id, b.id, CAST(ST_AsText(ST_Difference(a.geometry, b.geometry))
   as VARCHAR(200))  Difference
FROM sample_geoms a, sample_geoms b
WHERE a.id = 1 and b.id = 2
```

Results:

```
ID        ID      DIFFERENCE
--------- -------- ----------------------------------------------------
       1        2 POLYGON (( 10.00000000 10.00000000, 20.00000000
                    10.00000000, 20.00000000 20.00000000,
                    10.00000000 20.00000000, 10.00000000 10.00000000))
```

Example 3

This example finds the difference between two intersecting polygons.

```
SELECT a.id, b.id, CAST(ST_AsText(ST_Difference(a.geometry, b.geometry))
   as VARCHAR(200))  Difference
FROM sample_geoms a, sample_geoms b
WHERE a.id = 2 and b.id = 3
```

Results:

```
ID       ID       DIFFERENCE
-------- -------- ----------------------------------------------------
       2        3 POLYGON (( 30.00000000 30.00000000, 50.00000000
                    30.00000000, 50.00000000 40.00000000, 40.00000000
                    40.00000000, 40.00000000 50.00000000, 30.00000000
                    50.00000000, 30.00000000 30.00000000))
```

Example 4

This example finds the difference between two overlapping linestrings.

```
SELECT a.id, b.id, CAST(ST_AsText(ST_Difference(a.geometry, b.geometry))
   as VARCHAR(100))  Difference
FROM sample_geoms a, sample_geoms b
WHERE a.id = 4 and b.id = 5
```

Results:

```
ID       ID       DIFFERENCE
```

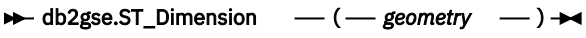
```
-------- -------- --------------------------------------------------
       4        5 LINESTRING ( 70.00000000 70.00000000, 75.00000000
                    75.00000000)
```

ST_Dimension

ST_Dimension takes a geometry as an input parameter and returns its dimension.

If the given geometry is empty, then -1 is returned. For points and multipoints, the dimension is 0 (zero); for linestrings and multilinestrings, the dimension is 1; and for polygons and multipolygons, the dimension is 2. If the given geometry is null, then null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types that represents the geometry for which the dimension is returned.

Return type

INTEGER

Example

The following example creates several different geometries and finds their dimensions.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_geoms (id char(15), geometry ST_Geometry)

INSERT INTO sample_geoms VALUES
       ('Empty Point', ST_Geometry('point EMPTY',0))

INSERT INTO sample_geoms VALUES
       ('Point ZM', ST_Geometry('point zm (10 10 16 30)' ,0))

INSERT INTO sample_geoms VALUES
       ('MultiPoint M', ST_Geometry('multipoint m (10 10 5,
         50 10 6, 10 30 8)' ,0))

INSERT INTO sample_geoms VALUES
       ('LineString', ST_Geometry('linestring (10 10, 15 20)',0))

INSERT INTO sample_geoms VALUES
       ('Polygon', ST_Geometry('polygon((40 120, 90 120, 90 150,
         40 150, 40 120))' ,0))

SELECT id, ST_Dimension(geometry)  Dimension
FROM sample_geoms
```

Results:

```
ID              DIMENSION
--------------- -----------
Empty Point              -1
Point ZM                  0
MultiPoint M              0
LineString                1
Polygon                   2
```

ST_Disjoint

ST_Disjoint takes two geometries as input parameters and returns 1 if the given geometries do not intersect. If the geometries do intersect, then 0 (zero) is returned.

If the second geometry is not represented in the same spatial reference system as the first geometry, it will be converted to the other spatial reference system.

If any of the two geometries is null or is empty, then null value is returned.

Syntax



Parameter

geometry1
A value of one of the seven distinct spatial data types that represents the geometry that is tested to be disjoint with *geometry2*.

geometry2
A value of one of the seven distinct spatial data types that represents the geometry that is tested to be disjoint with *geometry1*.

Return type

INTEGER

Examples

Example 1

This code creates several geometries in the SAMPLE_GEOMETRIES table.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_geoms (id INTEGER, geometry ST_Geometry)

INSERT INTO sample_geoms VALUES
       (1, ST_Geometry(ST_Polygon('polygon((20 30, 30 30, 30 40, 20 40, 20 30))',
        0)))

INSERT INTO sample_geoms VALUES
       (2, ST_Geometry(ST_Polygon('polygon((30 30, 30 50, 50 50, 50 30, 30 30))',
        0)))

INSERT INTO sample_geoms VALUES
       (3, ST_Geometry(ST_Polygon('polygon((40 40, 40 60, 60 60, 60 40, 40 40))',
        0)))

INSERT INTO sample_geoms VALUES
       (4, ST_Geometry(ST_Linestring('linestring(60 60, 70 70)' ,0)))

INSERT INTO sample_geoms VALUES
       (5, ST_Geometry(ST_Linestring('linestring(30 30, 40 40)' ,0)))
```

Example 2

This example determines if the first polygon is disjoint from any of the geometries.

```
SELECT a.id, b.id, ST_Disjoint(a.geometry, b.geometry) DisJoint
FROM sample_geoms a, sample_geoms b
WHERE a.id = 1
```

Results:

```
ID          ID          DISJOINT
----------- ----------- -----------
          1           1           0
          1           2           0
          1           3           1
          1           4           1
          1           5           0
```

Example 3

This example determines if the third polygon is disjoint from any of the geometries.

```
SELECT a.id, b.id, ST_Disjoint(a.geometry, b.geometry) DisJoint
FROM sample_geoms a, sample_geoms b
WHERE a.id = 3
```

Results:

```
ID          ID          DISJOINT
----------- ----------- -----------
          3           1           1
          3           2           0
          3           3           0
          3           4           0
          3           5           0
```

Example 4

This example determines if the second linestring is disjoint from any of the geometries.

```
SELECT a.id, b.id, ST_Disjoint(a.geometry, b.geometry) DisJoint
FROM sample_geoms a, sample_geoms b
WHERE a.id = 5
```

Results:

```
ID          ID          DISJOINT
----------- ----------- -----------
          5           1           0
          5           2           0
          5           3           0
          5           4           1
          5           5           0
```

ST_Distance

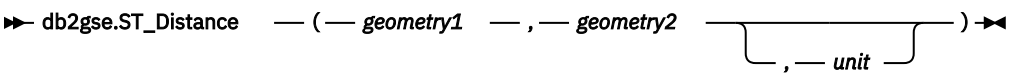
ST_Distance takes two geometries and, optionally, a unit as input parameters and returns the shortest distance between any point in the first geometry to any point in the second geometry, measured in the default or given units.

If any of the two geometries is null or is empty, null is returned.

If the second geometry is not represented in the same spatial reference system as the first geometry, it will be converted to the other spatial reference system.

Note: After you apply APAR PM92224, the results that are returned by the ST_Distance function might be different than before the APAR was applied, unless the function uses point data exclusively.

Syntax



Parameter

geometry1
A value of one of the seven distinct spatial data types that represents the geometry that is used to compute the distance to *geometry2*.

geometry2
A value of one of the seven distinct spatial data types that represents the geometry that is used to compute the distance to *geometry1*.

unit
VARCHAR(128) value that identifies the unit in which the result is measured. The supported units of measure are listed in the DB2GSE.ST_UNITS_OF_MEASURE catalog view.

If the *unit* parameter is omitted, the following rules are used to determine the unit of measure used for the result:

- If *geometry1* is in a projected or geocentric coordinate system, the linear unit associated with this coordinate system is the default.
- If *geometry1* is in a geographic coordinate system, the angular unit associated with this coordinate system is the default.

If the geometry is in a geographic coordinate system, you can specify a linear unit as the value.

Restrictions on unit conversions: An error (SQLSTATE 38SU4) is returned if any of the following conditions occur:

- The geometry is in an unspecified coordinate system and the *unit* parameter is specified.
- The geometry is in a projected coordinate system and an angular unit is specified.

Return type

DOUBLE

Examples

Example 1

The following SQL statements create and populate the SAMPLE_GEOMETRIES1 and SAMPLE_GEOMETRIES2 tables.

```
SET CURRENT PATH = CURRENT PATH, db2gse;

CREATE TABLE sample_geometries1(id SMALLINT, spatial_type varchar(13),
   geometry ST_GEOMETRY);

CREATE TABLE sample_geometries2(id SMALLINT, spatial_type varchar(13),
   geometry ST_GEOMETRY);

INSERT INTO sample_geometries1(id, spatial_type, geometry)
VALUES
    ( 1, 'ST_Point', ST_GEOMETRY(ST_Point('point(100 100)', 1) ));

INSERT INTO sample_geometries1(id, spatial_type, geometry)
VALUES
    (10, 'ST_LineString', ST_GEOMETRY(ST_LineString('linestring(125 125,
                          125 175)', 1) ));

INSERT INTO sample_geometries1(id, spatial_type, geometry)
VALUES
    (20, 'ST_Polygon', ST_GEOMETRY(ST_Polygon('polygon
                       ((50 50, 50 150, 150 150, 150 50, 50 50))', 1) ));
```

```
INSERT INTO sample_geometries2(id, spatial_type, geometry)
VALUES
    (101, 'ST_Point', ST_GEOMETRY(ST_Point('point(200 200)', 1) ));

INSERT INTO sample_geometries2(id, spatial_type, geometry)
VALUES
    (102, 'ST_Point', ST_GEOMETRY(ST_Point('point(200 300)', 1) ));

INSERT INTO sample_geometries2(id, spatial_type, geometry)
VALUES
    (103, 'ST_Point', ST_GEOMETRY(ST_Point('point(200 0)', 1) ));

INSERT INTO sample_geometries2(id, spatial_type, geometry)
VALUES
    (110, 'ST_LineString', ST_GEOMETRY(ST_LineString('linestring(200 100,
                           200 200)', 1) ));

INSERT INTO sample_geometries2(id, spatial_type, geometry)
VALUES
    (120, 'ST_Polygon', ST_GEOMETRY(ST_Polygon('polygon
                        ((200 0, 200 200, 300 200, 300 0, 200 0))', 1) ));
```

Example 2

The following SELECT statement calculates the distance between the various geometries in the SAMPLE_GEOMETRIES1 and SAMPLE_GEOMETRIES2 tables.

```
SELECT   sg1.id AS sg1_id, sg1.spatial_type AS sg1_type,
         sg2.id AS sg1_id, sg2.spatial_type AS sg2_type,
         cast(ST_Distance(sg1.geometry, sg2.geometry)
         AS Decimal(8, 4)) AS distance
FROM     sample_geometries1 sg1, sample_geometries2 sg2
ORDER BY sg1.id;
```

Results:

```
SG1_ID SG1_TYPE      SG1_ID SG2_TYPE      DISTANCE
------ ------------- ------ ------------- ----------
     1 ST_Point         101 ST_Point        141.4213
     1 ST_Point         102 ST_Point        223.6067
     1 ST_Point         103 ST_Point        141.4213
     1 ST_Point         110 ST_LineString   100.0000
     1 ST_Point         120 ST_Polygon      100.0000
    10 ST_LineString    101 ST_Point         79.0569
    10 ST_LineString    102 ST_Point        145.7737
    10 ST_LineString    103 ST_Point        145.7737
    10 ST_LineString    110 ST_LineString    75.0000
    10 ST_LineString    120 ST_Polygon       75.0000
    20 ST_Polygon       101 ST_Point         70.7106
    20 ST_Polygon       102 ST_Point        158.1138
    20 ST_Polygon       103 ST_Point         70.7106
    20 ST_Polygon       110 ST_LineString    50.0000
    20 ST_Polygon       120 ST_Polygon       50.0000
```

Example 3

The following SELECT statement illustrates how to find all the geometries that are within a distance of 100 of each other.

```
SELECT   sg1.id AS sg1_id, sg1.spatial_type AS sg1_type,
         sg2.id AS sg1_id, sg2.spatial_type AS sg2_type,
         cast(ST_Distance(sg1.geometry, sg2.geometry)
         AS Decimal(8, 4)) AS distance
FROM     sample_geometries1 sg1, sample_geometries2 sg2
WHERE    ST_Distance(sg1.geometry, sg2.geometry)  <= 100;
```

Results:

```
SG1_ID SG1_TYPE      SG1_ID SG2_TYPE      DISTANCE
------ ------------- ------ ------------- ----------
     1 ST_Point         110 ST_LineString   100.0000
     1 ST_Point         120 ST_Polygon      100.0000
    10 ST_LineString    101 ST_Point         79.0569
    10 ST_LineString    110 ST_LineString    75.0000
    10 ST_LineString    120 ST_Polygon       75.0000
    20 ST_Polygon       101 ST_Point         70.7106
    20 ST_Polygon       103 ST_Point         70.7106
    20 ST_Polygon       110 ST_LineString    50.0000
    20 ST_Polygon       120 ST_Polygon       50.0000
```

Example 4

The following SELECT statement calculates the distance in kilometers between the various geometries.

```
SAMPLE_GEOMETRIES1 and SAMPLE_GEOMETRIES2 tables.
SELECT   sg1.id AS sg1_id, sg1.spatial_type AS sg1_type,
         sg2.id AS sg1_id, sg2.spatial_type AS sg2_type,
         cast(ST_Distance(sg1.geometry, sg2.geometry, 'KILOMETER')
         AS DECIMAL(10, 4)) AS distance
FROM     sample_geometries1 sg1, sample_geometries2 sg2
ORDER BY sg1.id;
```

Results:

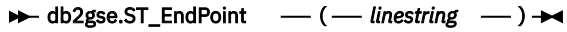
```
SG1_ID SG1_TYPE      SG1_ID SG2_TYPE      DISTANCE
------ ------------- ------ ------------- ------------
     1 ST_Point         101 ST_Point        12373.2168
     1 ST_Point         102 ST_Point        16311.3816
     1 ST_Point         103 ST_Point         9809.4713
     1 ST_Point         110 ST_LineString    1707.4463
     1 ST_Point         120 ST_Polygon      12373.2168
    10 ST_LineString    101 ST_Point         8648.2333
    10 ST_LineString    102 ST_Point        11317.3934
    10 ST_LineString    103 ST_Point        10959.7313
    10 ST_LineString    110 ST_LineString    3753.5862
    10 ST_LineString    120 ST_Polygon      10891.1254
    20 ST_Polygon       101 ST_Point         7700.5333
    20 ST_Polygon       102 ST_Point        15039.8109
    20 ST_Polygon       103 ST_Point         7284.8552
    20 ST_Polygon       110 ST_LineString    6001.8407
    20 ST_Polygon       120 ST_Polygon      14515.8872
```

ST_Endpoint

ST_Endpoint takes a linestring as an input parameter and returns the point that is the last point of the linestring. The resulting point is represented in the spatial reference system of the given linestring.

If the given linestring is null or is empty, then null is returned.

Syntax



Parameter

linestring
A value of type ST_Linestring that represents the geometry from which the last point is returned.

Return type

db2gse.ST_Point

Example

The SELECT statement finds the endpoint of each of the geometries in the SAMPLE_LINES table.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_lines(id INTEGER, line ST_Linestring)


INSERT INTO sample_lines VALUES
    (1, ST_LineString('linestring (10 10, 5 5, 0 0, 10 0, 5 5, 0 10)', 0) )

INSERT INTO sample_lines VALUES
    (2, ST_LineString('linestring z (0 0 4, 5 5 5, 10 10 6, 5 5 7)', 0) )


SELECT id, CAST(ST_AsText(ST_EndPoint(line)) as VARCHAR(50)) Endpoint
FROM   sample_lines
```

Results:

```
ID          ENDPOINT
----------- --------------------------------------------------
          1 POINT ( 0.00000000 10.00000000)
          2 POINT Z ( 5.00000000 5.00000000 7.00000000)
```

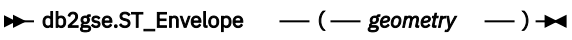
ST_Envelope

ST_Envelope takes a geometry as an input parameter and returns an envelope around the geometry. The envelope is a rectangle that is represented as a polygon.

If the given geometry is a point, a horizontal linestring, or a vertical linestring, then a rectangle, which is slightly larger than the given geometry, is returned. Otherwise, the minimum bounding rectangle of the geometry is returned as the envelope.

If the given geometry is null or is empty, then null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types that represents the geometry to return the envelope for.

Return type

db2gse.ST_Polygon

Example

In the following examples, the lines of results have been reformatted for readability.

This example creates several geometries and then determines their envelopes. For the non-empty point and the linestring (which is horizontal), the envelope is a rectangle that is slightly larger than the geometry.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_geoms (id INTEGER, geometry ST_Geometry);
```

```
INSERT INTO sample_geoms VALUES
    (1, ST_Geometry(ST_Point('point EMPTY',0)));

INSERT INTO sample_geoms VALUES
    (2, ST_Geometry(ST_Point('point zm (10 10 16 30)' ,0)));

INSERT INTO sample_geoms VALUES
    (3, ST_Geometry(ST_Multipoint('multipoint m (10 10 5, 50 10 6,
             10 30 8)' ,0)));

INSERT INTO sample_geoms VALUES
    (4, ST_Geometry(ST_Linestring('linestring (10 10, 20 10)',0)));

INSERT INTO sample_geoms VALUES
    (5, ST_Geometry(ST_Polygon('polygon((40 120, 90 120, 90 150,
             40 150, 40 120))',0)));


SELECT id, CAST(ST_AsText(ST_Envelope(geometry)) as VARCHAR(160))  Envelope
FROM sample_geoms;
```

Results:

```
ID          ENVELOPE
----------- ----------------------------------------------------------------
          1 -

          2 POLYGON (( 9 9, 11 9, 11 11, 9 11, 9 9))

          3 POLYGON (( 10 10, 50 10, 50 30, 10 30, 10 10))

          4 POLYGON (( 10 9, 20 9, 20  11, 10 11, 10 9))

          5 POLYGON (( 40 120, 90 120, 90 150, 40 150, 40 120))
```

ST_Equals

ST_Equals takes two geometries as input parameters and returns 1 if the geometries are equal. Otherwise 0 (zero) is returned. The order of the points used to define the geometry is not relevant for the test for equality.

If the second geometry is not represented in the same spatial reference system as the first geometry, it will be converted to the other spatial reference system.

If any of the two given geometries is null, then null is returned. If both of the geometries are empty, then 1 is returned. If one out of the two given geometries is empty, then 0 (zero) is returned.

Syntax



Parameter

geometry1
A value of one of the seven distinct spatial data types that represents the geometry that is to be compared with *geometry2*.

geometry2
A value of one of the seven distinct spatial data types that represents the geometry that is to be compared with *geometry1*.

Return type

INTEGER

Examples

Example 1

This example creates two polygons that have their coordinates in a different order. ST_Equal is used to show that these polygons are considered equal.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_geoms (id INTEGER, geometry ST_Geometry);

INSERT INTO sample_geoms VALUES
       (1, ST_Geometry(ST_Polygon('polygon((50 30, 30 30, 30 50, 50 50,
                                  50 30))' ,0)));

INSERT INTO sample_geoms VALUES
       (2, ST_Geometry(ST_Polygon('polygon((50 30, 50 50, 30 50, 30 30,
                                  50 30))' ,0)));


SELECT a.id, b.id, ST_Equals(a.geometry, b.geometry) Equals
FROM sample_geoms a, sample_geoms b
WHERE a.id = 1 and b.id = 2;
```

Results:

```
ID          ID          EQUALS
----------- ----------- -----------
          1           2           1
```

Example 2

In this example, two geometries are created with the same X and Y coordinates, but different M coordinates (measures). When the geometries are compared with the ST_Equal function, a 0 (zero) is returned to indicate that these geometries are not equal.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_geoms (id INTEGER, geometry ST_Geometry);

INSERT INTO sample_geoms VALUES
       (3, ST_Geometry(ST_MultiPoint('multipoint m(80 80 6, 90 90 7)' ,0)));

INSERT INTO sample_geoms VALUES
       (4, ST_Geometry(ST_MultiPoint('multipoint m(80 80 6, 90 90 4)' ,0)));

SELECT a.id, b.id, ST_Equals(a.geometry, b.geometry) Equals
FROM sample_geoms a, sample_geoms b
WHERE a.id = 3 and b.id = 4;
```

Results:

```
ID          ID          EQUALS
----------- ----------- -----------
          3           4           0
```

Example 3

In this example, two geometries are created with a different set of coordinates, but both represent the same geometry. ST_Equal compares the geometries and indicates that both geometries are indeed equal.

```
SET current path = current path, db2gse;
CREATE TABLE sample_geoms ( id INTEGER, geometry ST_Geometry );

INSERT INTO sample_geoms VALUES
   (5, ST_GEOMETRY(ST_LineString('linestring ( 10 10, 40 40 )', 0)));

INSERT INTO sample_geoms VALUES
   (6, ST_GEOMETRY(ST_LineString('linestring ( 10 10, 20 20, 40 40)', 0)));

SELECT a.id, b.id, ST_Equals(a.geometry, b.geometry) Equals
FROM   sample_geoms a, sample_geoms b
WHERE  a.id = 5 AND b.id = 6;
```

Results:

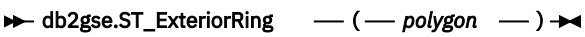
```
ID          ID          EQUALS
----------- ----------- -----------
          5           6           1
```

ST_ExteriorRing

ST_ExteriorRing takes a polygon as an input parameter and returns its exterior ring as a linestring. The resulting linestring is represented in the spatial reference system of the given polygon.

If the given polygon is null or is empty, then null is returned. If the polygon does not have any interior rings, the returned exterior ring is identical to the boundary of the polygon.

Syntax



Parameter

polygon
A value of type ST_Polygon that represents the polygon for which the exterior ring is to be returned.

Return type

db2gse.ST_Linestring

Example

In the following examples, the lines of results have been reformatted for readability. The spacing in your results will vary according to your online display.

This example creates two polygons, one with two interior rings and one with no interior rings, then it determines their exterior rings.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_polys (id INTEGER, geometry ST_Polygon)

INSERT INTO sample_polys VALUES
       (1, ST_Polygon('polygon((40 120, 90 120, 90 150, 40 150, 40 120),
                      (50 130, 60 130, 60 140, 50 140, 50 130),
                      (70 130, 80 130, 80 140, 70 140, 70 130))' ,0))

INSERT INTO sample_polys VALUES
       (2, ST_Polygon('polygon((10 10, 50 10, 10 30, 10 10))' ,0))


SELECT id, CAST(ST_AsText(ST_ExteriorRing(geometry))
   AS VARCHAR(180)) Exterior_Ring
```

```
FROM sample_polys
```

Results:

```
ID          EXTERIOR_RING
----------- ------------------------------------------------------------------
          1 LINESTRING ( 40.00000000 120.00000000, 90.00000000
   120.00000000,  90.00000000 150.00000000, 40.00000000 150.00000000,
   40.00000000 120.00000000)

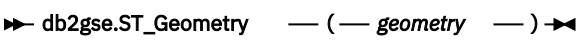
          2 LINESTRING ( 10.00000000 10.00000000, 50.00000000
   10.00000000, 10.00000000 30.00000000, 10.00000000 10.00000000)
```

ST_Geometry

The ST_Geometry function takes a geometry as an input parameter and casts the output type to ST_Geometry.

If the given geometry is null, then null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types.

Return type

db2gse.ST_Geometry

Examples

The following code example shows how you can use the ST_Geometry function to recast any spatial data type.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_geometries(id INTEGER, geometry ST_GEOMETRY)
INSERT INTO sample_geometries(id, geometry)
       VALUES (7001, ST_Geometry(ST_point ( point(1 2), 1) )
INSERT INTO sample_geometries(id, geometry)
       VALUES (7002, ST_Geometry(ST_line string ( linestring(33 2, 34 3,
                                                  35 6), 1) )
INSERT INTO sample_geometries(id, geometry)
       VALUES (7003, ST_Geometry(ST_polygon ( polygon((3 3, 4 6, 5 3,
                                             3 3)), 1)))
SELECT id, cast(ST_AsText (geometry) AS varchar(120))
  AS geometry FROM sample_geometries
```

Results:

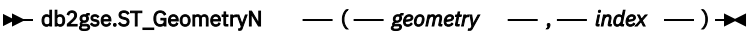
```
 ID          GEOMETRY
----------- ---------------------
   7001     POINT ( 1.00000000 2.00000000)
   7002     LINESTRING ( 33.00000000 2.00000000, 34.00000000 3.00000000,
                        35.00000000 6.00000000)
   7003     POLYGON (( 3.00000000 3.00000000, 5.00000000 3.00000000,
                      4.00000000 6.00000000, 3.00000000 3.00000000))
```

ST_GeometryN

ST_GeometryN takes a geometry type and an index as input parameters and returns the geometry in the type that is identified by the index. The resulting geometry is represented in the spatial reference system of the given geometry type.

If the given geometry is null or is empty, then null is returned.

Syntax



Parameter

geometry
A value of type ST_MultiLineString, ST_MultiPolygon, or ST_MultiPoint that represents the geometry type to locate the *n*th geometry within.

index
A value of type INTEGER that identifies the *n*th geometry that is to be returned from *geometry*.

If *index* is smaller than 1 or larger than the number of geometries in the collection, then null is returned and a warning is returned (SQLSTATE 01HS0).

Return type

db2gse.ST_Geometry

Example

The following code illustrates how to choose the second geometry inside a geometry type.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse

CREATE TABLE sample_geomcollections (id INTEGER,
  geometry ST_Geometry)

INSERT INTO sample_geomcollections(id, geometry)
VALUES
    (4001, ST_Geometry( ST_MultiPoint('multipoint(1 2, 4 3)', 1)) ),
INSERT INTO sample_geomcollections(id, geometry)
VALUES
    (4002, ST_Geometry( ST_MultiLineString('multilinestring(
                           (33 2, 34 3, 35 6),
                           (28 4, 29 5, 31 8, 43 12),
                           (39 3, 37 4, 36 7))', 1)) ),
INSERT INTO sample_geomcollections(id, geometry)
VALUES
    (4003, ST_Geometry( ST_MultiPolygon('multipolygon(((3 3, 4 6, 5 3, 3 3),
                               (8 24, 9 25, 1 28, 8 24),
                               (13 33, 7 36, 1 40, 10 43, 13 33)))', 1)) )

SELECT id, cast(ST_AsText(ST_GeometryN(geometry, 2)) AS varchar(110))
    second_geometry
FROM sample_geomcollections
```

Results:

```
ID          SECOND_GEOMETRY
----------- ------------------------------------------------------------
       4001 POINT ( 4.00000000 3.00000000)

       4002 LINESTRING ( 28.00000000 4.00000000, 29.00000000 5.00000000,
   31.00000000 8.00000000, 43.00000000 12.00000000)

       4003 POLYGON (( 8.00000000 24.00000000, 9.00000000 25.00000000,
   1.00000000 28.00000000, 8.00000000 24.00000000))
```

ST_GeometryType

ST_GeometryType takes a geometry as the input parameter and returns the fully qualified type name of the dynamic type of that geometry.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types for which the geometry type is to be returned.

Return type

CHAR(32)

Examples

The following code illustrates how to determine the type of a geometry.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse

CREATE TABLE sample_geometries (id INTEGER, geometry ST_GEOMETRY)

INSERT INTO sample_geometries(id, geometry)
VALUES
    (7101, ST_Geometry('point(1 2)', 1) ),
    (7102, ST_Geometry('linestring(33 2, 34 3, 35 6)', 1) ),
    (7103, ST_Geometry('polygon((3 3, 4 6, 5 3, 3 3))', 1)),
    (7104, ST_Geometry('multipoint(1 2, 4 3)', 1) )

SELECT id, geometry..ST_GeometryType AS geometry_type
FROM   sample_geometries
```

Results:

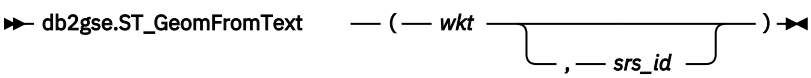
```
ID          GEOMETRY_TYPE
----------- ------------------------------
       7101 DB2GSE.ST_POINT
       7102 DB2GSE.ST_LINESTRING
       7103 DB2GSE.ST_POLYGON
       7104 DB2GSE.ST_MULTIPOINT
```

ST_GeomFromText

ST_GeomFromText takes a well-known text representation of a geometry and a spatial reference system identifier as input parameters and returns the corresponding geometry.

If the given well-known text representation is null, then null is returned.

Syntax



Parameter

wkt
A value of type CLOB(8M) that contains the well-known text representation of the resulting geometry. If the well-known text representation is null, null is returned.

srs_id
A value of type INTEGER that identifies the spatial reference system for the resulting geometry. This parameter is required.

If *srs_id* does not identify a spatial reference system that is listed in the catalog view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS, an error is returned (SQLSTATE 38SU1).

Return type

db2gse.ST_Geometry

Example

In this example the ST_GeomFromText function is used to create and insert a geometry from a well known text (WKT) point representation.

The following code inserts rows into the SAMPLE_POINTS table with IDs and geometries in spatial reference system 1 using WKT representation.

```
SET CURRENT PATH = CURRENT PATH, db2gse

CREATE TABLE sample_geometries(id INTEGER, geometry ST_GEOMETRY)

INSERT INTO sample_geometries(id, geometry)
VALUES
    (1251, ST_GeomFromText('point(1 2)', 1) ),
    (1252, ST_GeomFromText('linestring(33 2, 34 3, 35 6)', 1) ),
    (1253, ST_GeomFromText('polygon((3 3, 4 6, 5 3, 3 3))', 1))
```

The following SELECT statement will return the ID and GEOMETRIES from the SAMPLE_GEOMETRIES table.

```
SELECT id, cast(DB2GSE.ST_AsText(geometry)  AS varchar(105))
   AS geometry
FROM   sample_geometries
```

Results:

```
ID          GEOMETRY
----------- ----------------------------------------------------------------
       1251 POINT ( 1.00000000 2.00000000)

       1252 LINESTRING ( 33.00000000 2.00000000, 34.00000000 3.00000000,
              35.00000000 6.00000000)

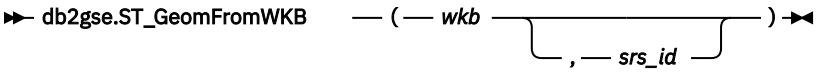
       1253 POLYGON (( 3.00000000 3.00000000, 5.00000000 3.00000000,
              4.00000000 6.00000000, 3.00000000 3.00000000))
```

ST_GeomFromWKB

ST_GeomFromWKB takes a well-known binary representation of a geometry and a spatial reference system identifier as input parameters and returns the corresponding geometry.

If the given well-known binary representation is null, null is returned.

Syntax



Parameter

wkb
A value of type BLOB(4M) that contains the well-known binary representation of the resulting geometry. If the well-known binary representation is null, null is returned.

srs_id
A value of type INTEGER that identifies the spatial reference system for the resulting geometry. This parameter is required.

If the specified *srs_id* parameter does not identify a spatial reference system that is listed in the catalog view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS, an error is returned (SQLSTATE 38SU1).

Return type

db2gse.ST_Geometry

Examples

The following code illustrates how the ST_GeomFromWKB function can be used to create and insert a geometry from a well-known binary (WKB) line representation.

The following example inserts a record into the SAMPLE_GEOMETRIES table with an ID and a geometry in spatial reference system 1 in a WKB representation.

```
SET CURRENT PATH = CURRENT PATH, db2gse

CREATE TABLE sample_geometries (id INTEGER, geometry ST_GEOMETRY,
    wkb BLOB(32K))

INSERT INTO sample_geometries(id, geometry)
VALUES
    (1901, ST_GeomFromText('point(1 2)', 1) ),
    (1902, ST_GeomFromText('linestring(33 2, 34 3, 35 6)', 1) ),
    (1903, ST_GeomFromText('polygon((3 3, 4 6, 5 3, 3 3))', 1))

UPDATE sample_geometries AS temp_correlated
SET    wkb = DB2GSE.ST_AsBinary(geometry)
WHERE  id = temp_correlated.id

SELECT id, cast(DB2GSE.ST_AsText(ST_GeomFromWKB(wkb)) AS varchar(190))
   AS geometry
FROM   sample_geometries
```

Results:

```
 ID
GEOMETRY
 ----------- ------------------------------------------------------------
1901         POINT ( 1.00000000 2.00000000)

1902         LINESTRING ( 33.00000000 2.00000000, 34.00000000
   3.00000000, 35.00000000 6.00000000)

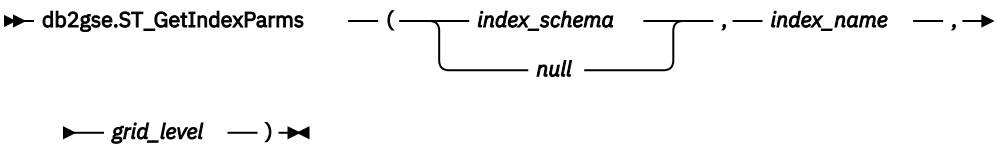
1903         POLYGON (( 3.00000000 3.00000000, 5.00000000 3.00000000,
   4.00000000 6.00000000, 3.00000000 3.00000000))
```

ST_GetIndexParms

ST_GetIndexParms takes the index schema, index name, and grid level value for a spatial index as input parameters and returns the grid size that was used to define the index on the spatial column.

If there is no match found for the given input parameters, then null is returned.

Syntax



Parameter

index_schema
A value of type VARCHAR(128) that identifies the schema in which the spatial index with the unqualified name *index_name* is in.

If this parameter is null, then the value of the CURRENT SCHEMA special register is used as the schema name for the spatial index.

index_name
A value of type VARCHAR(128) that contains the unqualified name of the spatial index for which the index parameters are returned.

grid_level
An INTEGER value that identifies the parameter whose value or values are to be returned. If this value is smaller than 1 or larger than 3, then an error is raised (SQLSTATE 38SQ1).

Return type

DOUBLE

Example

After creating a spatial index, you can use the following code example to return the grid size that was used to define the index on the spatial column.

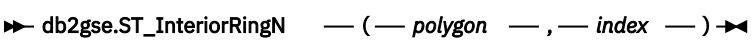
```
SELECT DB2GSE.ST_GETINDEXPARMS('schema', 'PT_IDX', 1)
FROM schema.DUMMY;
```

ST_InteriorRingN

ST_InteriorRingN takes a polygon and an index as input parameters and returns the interior ring identified by the given index as a linestring. The interior rings are organized according to the rules defined by the internal geometry verification routines.

If the given polygon is null or is empty, or if it does not have any interior rings, then null is returned.

Syntax



Parameter

polygon
A value of type ST_Polygon that represents the geometry from which the interior ring identified by *index* is returned.

index
A value of type INTEGER that identifies the *n*th interior ring that is returned. If there is no interior ring identified by *index*, then a warning is returned (01HS1).

Return type

db2gse.ST_LineString

Example

In this example, a polygon is created with two interior rings. The ST_InteriorRingN call is then used to retrieve the second interior ring.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_polys (id INTEGER, geometry ST_Polygon)

INSERT INTO sample_polys VALUES
       (1, ST_Polygon('polygon((40 120, 90 120, 90 150, 40 150, 40 120),
                      (50 130, 60 130, 60 140, 50 140, 50 130),
                      (70 130, 80 130, 80 140, 70 140, 70 130))' ,0))


SELECT id, CAST(ST_AsText(ST_InteriorRingN(geometry, 2)) as VARCHAR(180))
       Interior_Ring
FROM sample_polys
```

Results:

```
ID          INTERIOR_RING
----------- ----------------------------------------------------------------------
                  1 LINESTRING ( 70.00000000 130.00000000, 70.00000000 140.00000000,
80.00000000 140.00000000, 80.00000000 130.00000000, 70.00000000 130.00000000)
```

ST_Intersection

ST_Intersection takes two geometries as input parameters and returns the geometry that is the intersection of the two given geometries. The intersection is the common part of the first geometry and the second geometry. The resulting geometry is represented in the spatial reference system of the first geometry.

If possible, the specific type of the returned geometry will be ST_Point, ST_LineString, or ST_Polygon. For example, the intersection of a point and a polygon is either empty or a single point, represented as ST_MultiPoint.

If any of the two geometries is null, null is returned.

If the second geometry is not represented in the same spatial reference system as the first geometry, it will be converted to the other spatial reference system.

Syntax



Parameter

geometry1
A value of one of the seven distinct spatial data types that represents the first geometry to compute the intersection with *geometry2*.

geometry2
A value of one of the seven distinct spatial data types that represents the second geometry to compute the intersection with *geometry1*.

Return type

db2gse.ST_Geometry

The dimension of the returned geometry is that of the input with the lower dimension.

Example

In the following example, the results have been reformatted for readability. The spacing in your results will vary according to your display.

This example creates several different geometries and then determines the intersection (if any) with the first one.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_geoms (id INTEGER, geometry ST_Geometry)

INSERT INTO sample_geoms
  VALUES (1, ST_Geometry(ST_Polygon('polygon((30 30, 30 50, 50 50,
         50 30, 30 30))' ,0)))

INSERT INTO sample_geoms
  VALUES (2, ST_Geometry(ST_Polygon('polygon((20 30, 30 30, 30 40,
         20 40, 20 30))' ,0)))

INSERT INTO sample_geoms
  VALUES (3, ST_Geometry(ST_Polygon('polygon((40 40, 40 60, 60 60,
         60 40, 40 40))' ,0)))

INSERT INTO sample_geoms
  VALUES (4, ST_Geometry(ST_LineString('linestring(60 60, 70 70)' ,0)))

INSERT INTO sample_geoms VALUES
       (5, ST_Geometry(ST_LineString('linestring(30 30, 60 60)' ,0)))

SELECT a.id, b.id, CAST(ST_AsText(ST_Intersection(a.geometry, b.geometry))
         as VARCHAR(150)) Intersection
FROM sample_geoms a, sample_geoms b
WHERE a.id = 1
```

Results:

```
ID          ID          INTERSECTION
----------- ----------- --------------------------------------------------
          1           1 POLYGON (( 30.00000000 30.00000000, 50.00000000
30.00000000, 50.00000000 50.00000000, 30.00000000 50.00000000, 30.00000000
30.00000000))

          1           2 LINESTRING ( 30.00000000 40.00000000, 30.00000000
30.00000000)

          1           3 POLYGON (( 40.00000000 40.00000000, 50.00000000
40.00000000, 50.00000000 50.00000000, 40.00000000 50.00000000, 40.00000000
40.00000000))

          1           4 POINT EMPTY

          1           5 LINESTRING ( 30.00000000 30.00000000, 50.00000000
50.00000000)

  5 record(s) selected.
```

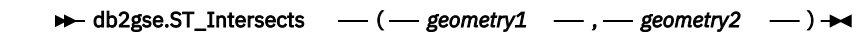
ST_Intersects

ST_Intersects takes two geometries as input parameters and returns 1 if the given geometries intersect. If the geometries do not intersect, 0 (zero) is returned.

If any of the two geometries is null or is empty, null is returned.

If the second geometry is not represented in the same spatial reference system as the first geometry, it will be converted to the other spatial reference system.

Syntax



Parameter

geometry1
A value of one of the seven distinct spatial data types that represents the geometry to test for intersection with *geometry2*.

geometry2
A value of one of the seven distinct spatial data types that represents the geometry to test for intersection with *geometry1*.

Return type

INTEGER

Example

The following statements create and populate the SAMPLE_GEOMETRIES1 and SAMPLE_GEOMETRIES2 tables.

```
SET CURRENT PATH = CURRENT PATH, db2gse;

CREATE TABLE sample_geometries1(id SMALLINT, spatial_type varchar(13),
    geometry ST_GEOMETRY);
CREATE TABLE sample_geometries2(id SMALLINT, spatial_type varchar(13),
    geometry ST_GEOMETRY);

INSERT INTO sample_geometries1(id, spatial_type, geometry)
VALUES
    ( 1, 'ST_Point', ST_GEOMETRY(ST_Point('point(550 150)', 1) ));

INSERT INTO sample_geometries1(id, spatial_type, geometry)
VALUES
    (10, 'ST_LineString', ST_GEOMETRY(ST_LineString('linestring(800 800,
          900 800)', 1)));

INSERT INTO sample_geometries1(id, spatial_type, geometry)
VALUES
    (20, 'ST_Polygon', ST_GEOMETRY(ST_Polygon('polygon((500 100, 500 200, 700 200,
          700 100, 500 100))', 1) ));

INSERT INTO sample_geometries2(id, spatial_type, geometry)
VALUES
    (101, 'ST_Point', ST_GEOMETRY(ST_Point('point(550 150)', 1) ));

INSERT INTO sample_geometries2(id, spatial_type, geometry)
VALUES
    (102, 'ST_Point', ST_GEOMETRY(ST_Point('point(650 200)', 1) ));

INSERT INTO sample_geometries2(id, spatial_type, geometry)
VALUES
    (103, 'ST_Point', ST_GEOMETRY(ST_Point('point(800 800)', 1) ));

INSERT INTO sample_geometries2(id, spatial_type, geometry)
VALUES
    (110, 'ST_LineString', ST_GEOMETRY(ST_LineString('linestring(850 250,
           850 850)', 1)));
```

```
INSERT INTO sample_geometries2(id, spatial_type, geometry)
VALUES
    (120, 'ST_Polygon', ST_GEOMETRY(ST_Polygon('polygon((650 50, 650 150, 800 150,
           800 50, 650 50))', 1)));

INSERT INTO sample_geometries2(id, spatial_type, geometry)
VALUES
    (121, 'ST_Polygon', ST_GEOMETRY(ST_Polygon('polygon((20 20, 20 40, 40 40,
           40 20, 20 20))', 1) ));
```

The following SELECT statement determines whether the various geometries in the SAMPLE_GEOMETRIES1 and SAMPLE_GEOMETRIES2 tables intersect.

```
SELECT   sg1.id AS sg1_id, sg1.spatial_type AS sg1_type,
         sg2.id AS sg2_id, sg2.spatial_type AS sg2_type,
         CASE ST_Intersects(sg1.geometry, sg2.geometry)
            WHEN 0 THEN 'Geometries do not intersect'
            WHEN 1 THEN 'Geometries intersect'
         END AS intersects
FROM     sample_geometries1 sg1, sample_geometries2 sg2
ORDER BY sg1.id;
```

Results:

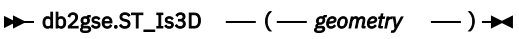
```
SG1_ID SG1_TYPE      SG2_ID SG2_TYPE      INTERSECTS
------ ------------- ------ ------------- --------------------------
     1 ST_Point         101 ST_Point      Geometries intersect
     1 ST_Point         102 ST_Point      Geometries do not intersect
     1 ST_Point         103 ST_Point      Geometries do not intersect
     1 ST_Point         110 ST_LineString Geometries do not intersect
     1 ST_Point         120 ST_Polygon    Geometries do not intersect
     1 ST_Point         121 ST_Polygon    Geometries do not intersect
    10 ST_LineString    101 ST_Point      Geometries do not intersect
    10 ST_LineString    102 ST_Point      Geometries do not intersect
    10 ST_LineString    103 ST_Point      Geometries intersect
    10 ST_LineString    110 ST_LineString Geometries intersect
    10 ST_LineString    120 ST_Polygon    Geometries do not intersect
    10 ST_LineString    121 ST_Polygon    Geometries do not intersect
    20 ST_Polygon       101 ST_Point      Geometries intersect
    20 ST_Polygon       102 ST_Point      Geometries intersect
    20 ST_Polygon       103 ST_Point      Geometries do not intersect
    20 ST_Polygon       110 ST_LineString Geometries do not intersect
    20 ST_Polygon       120 ST_Polygon    Geometries intersect
    20 ST_Polygon       121 ST_Polygon    Geometries do not intersect
```

ST_Is3D

ST_Is3D takes a geometry as an input parameter and returns 1 if the given geometry has Z coordinates. Otherwise, 0 (zero) is returned.

If the given geometry is null or is empty, null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types that represents the geometry that is to be tested for the existence of Z coordinates.

Return type

INTEGER

Example

In this example, several geometries are created with and without Z coordinates and M coordinates (measures). ST_Is3D is then used to determine which of them contain Z coordinates.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_geoms (id INTEGER, geometry ST_Geometry)

INSERT INTO sample_geoms VALUES
       (1, ST_Geometry(ST_Point('point EMPTY',0)))

INSERT INTO sample_geoms VALUES
       (2, ST_Geometry(ST_Polygon('polygon((40 120, 90 120, 90 150, 40 150, 40 120))' ,0)))

INSERT INTO sample_geoms VALUES
       (3, ST_Geometry(ST_Multipoint('multipoint m (10 10 5, 50 10 6, 10 30 8)' ,0)))

INSERT INTO sample_geoms VALUES
       (4, ST_Geometry(ST_Linestring('linestring z (10 10 166, 20 10 168)',0)))

INSERT INTO sample_geoms VALUES
       (5, ST_Geometry(ST_Point('point zm (10 10 16 30)' ,0)))


SELECT id, ST_Is3d(geometry)  Is_3D
FROM sample_geoms
```

Results:

```
ID          IS_3D
----------- -----------
          1           0
          2           0
          3           0
          4           1
          5           1
```

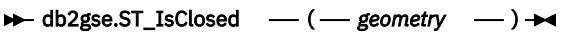
ST_IsClosed

ST_IsClosed takes a linestring or a multilinestring as an input parameter and returns 1 if the given linestring or multilinestring is closed. Otherwise, 0 (zero) is returned.

A linestring is closed if the start point and end point are equal. If the linestring has Z coordinates, the Z coordinates of the start point and end point must be equal. Otherwise, the points are not considered equal, and the linestring is not closed. A multilinestring is closed if each of its linestrings are closed.

If the given linestring or multilinestring is empty, then 0 (zero) is returned. If the geometry is null, then null is returned.

Syntax



Parameter

geometry
A value of type ST_LineString or ST_MultiLineString that represents the linestring or multilinestring that is to be tested.

Return type

INTEGER

Examples

Example 1

This example creates several linestrings. The last two linestrings have the same X and Y coordinates, but one linestring contains varying Z coordinates that cause the linestring to not be closed, and the other linestring contains varying M coordinates (measures) that do not affect whether the linestring is closed.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_lines (id INTEGER, geometry ST_Linestring)

INSERT INTO sample_lines VALUES
       (1, ST_Linestring('linestring EMPTY',0))

INSERT INTO sample_lines VALUES
       (2, ST_Linestring('linestring(10 10, 20 10, 20 20)' ,0))

INSERT INTO sample_lines VALUES
       (3, ST_Linestring('linestring(10 10, 20 10, 20 20, 10 10)' ,0))

INSERT INTO sample_lines VALUES
       (4, ST_Linestring('linestring m(10 10 1, 20 10 2, 20 20 3,
        10 10 4)' ,0))

INSERT INTO sample_lines VALUES
       (5, ST_Linestring('linestring z(10 10 5, 20 10 6, 20 20 7,
        10 10 8)' ,0))

SELECT id, ST_IsClosed(geometry)  Is_Closed
FROM sample_lines
```

Results:

```
ID          IS_CLOSED
----------- -----------
          1           0
          2           0
          3           1
          4           1
          5           0
```

Example 2

In this example, two multilinestrings are created. ST_IsClosed is used to determine if the multilinestrings are closed. The first one is not closed, even though all of the curves together form a complete closed loop. This is because each curve itself is not closed.

The second multilinestring is closed because each curve itself is closed.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_mlines (id INTEGER, geometry ST_MultiLinestring)
INSERT INTO sample_mlines VALUES
       (6, ST_MultiLinestring('multilinestring((10 10, 20 10, 20 20),
                                               (20 20, 30 20, 30 30),
                                               (30 30, 10 30, 10 10))',0))

INSERT INTO sample_mlines VALUES
       (7, ST_MultiLinestring('multilinestring((10 10, 20 10, 20 20, 10 10 ),
                                                (30 30, 50 30, 50 50,
                                                 30 30 ))',0))

SELECT id, ST_IsClosed(geometry)  Is_Closed
FROM sample_mlines
```

Results:

```
ID          IS_CLOSED
----------- -----------
          6           0
          7           1
```

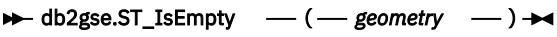
ST_IsEmpty

ST_IsEmpty takes a geometry as an input parameter and returns 1 if the given geometry is empty. Otherwise 0 (zero) is returned.

A geometry is empty if it does not have any points that define it.

If the given geometry is null, then null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types that represents the geometry that is to be tested.

Return type

INTEGER

Example

The following example creates three geometries, and then determines if the geometries are empty.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_geoms (id INTEGER, geometry ST_Geometry)

INSERT INTO sample_geoms VALUES
       (1, ST_Geometry(ST_Point('point EMPTY',0)))

INSERT INTO sample_geoms VALUES
       (2, ST_Geometry(ST_Polygon('polygon((40 120, 90 120, 90 150,
       40 150, 40 120))' ,0)))

INSERT INTO sample_geoms VALUES
       (3, ST_Geometry(ST_MultiPoint('multipoint m (10 10 5, 50 10 6,
       10 30 8)' ,0)))

INSERT INTO sample_geoms VALUES
       (4, ST_Geometry(ST_LineString('linestring z (10 10 166,
       20 10 168)',0)))

INSERT INTO sample_geoms VALUES
       (5, ST_Geometry(ST_Point('point zm (10 10 16 30)' ,0)))
```

```
SELECT id, ST_IsEmpty(geometry)  Is_Empty
FROM sample_geoms
```

Results:

```
ID          IS_EMPTY
----------- -----------
          1           1
          2           0
          3           0
          4           0
          5           0
```

ST_IsMeasured

ST_IsMeasured takes a geometry as an input parameter and returns 1 if the given geometry has M coordinates (measures). Otherwise 0 (zero) is returned.

If the given geometry is null or is empty, null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types that represents the geometry to be tested for the existence of M coordinates (measures).

Return type

INTEGER

Example

In this example, several geometries are created with and without Z coordinates and M coordinates (measures). ST_IsMeasured is then used to determine which of them contained measures.

```
SET CURRENT  PATH = CURRENT PATH, db2gse
CREATE TABLE sample_geoms (id INTEGER, geometry ST_Geometry)

INSERT INTO sample_geoms VALUES
       (1, ST_Geometry(ST_Point('point EMPTY',0)))

INSERT INTO sample_geoms VALUES
       (2, ST_Geometry(ST_Polygon('polygon((40 120, 90 120, 90 150, 40 150, 40 120))' ,0)))

INSERT INTO sample_geoms VALUES
       (3, ST_Geometry(ST_Multipoint('multipoint m (10 10 5, 50 10 6, 10 30 8)' ,0)))

INSERT INTO sample_geoms VALUES
       (4, ST_Geometry(ST_Linestring('linestring z (10 10 166, 20 10 168)',0)))

INSERT INTO sample_geoms VALUES
       (5, ST_Geometry(ST_Point('point zm (10 10 16 30)' ,0)))


SELECT id, ST_IsMeasured(geometry)  Is_Measured
FROM sample_geoms
```

Results:

```
ID          IS_MEASURED
----------- -----------
          1           0
          2           0
          3           1
          4           0
```

```
          5           1


```

ST_IsRing

ST_IsRing takes a linestring as an input parameter and returns 1 if it is a ring. Otherwise, 0 (zero) is returned. A linestring is a ring if it is simple and closed.

If the given linestring is empty, then 0 (zero) is returned. If the linestring is null, then null is returned.

Syntax

Parameter

linestring
A value of type ST_LineString that represents the linestring that is to be tested.

Return type

INTEGER

Examples

In this example, four linestrings are created. ST_IsRing is used to check if they are rings. The last one is not considered a ring even though it is closed, because the path crosses over itself.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_lines (id INTEGER, geometry ST_Linestring)

INSERT INTO sample_lines VALUES
       (1, ST_Linestring('linestring EMPTY',0))

INSERT INTO sample_lines VALUES
       (2, ST_Linestring('linestring(10 10, 20 10, 20 20)' ,0))

INSERT INTO sample_lines VALUES
       (3, ST_Linestring('linestring(10 10, 20 10, 20 20, 10 10)' ,0))

INSERT INTO sample_lines VALUES
       (4, ST_Linestring('linestring(10 10, 20 10, 10 20, 20 20, 10 10)' ,0))


SELECT id, ST_IsClosed(geometry) Is_Closed, ST_IsRing(geometry)  Is_Ring
FROM sample_lines
```

Results:

```
ID          IS_CLOSED   IS_RING
----------- ----------- -----------
          1           1           0
          2           0           0
          3           1           1
          4           1           0


```

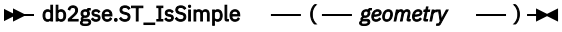
ST_IsSimple

ST_IsSimple takes a geometry as an input parameter and returns 1 if the given geometry is simple. Otherwise, 0 (zero) is returned.

Points, polygons, and multipolygons are always simple. A linestring is simple if it does not pass through the same point twice; a multipoint is simple if it does not contain two equal points; and a multilinestring is simple if all of its linestrings are simple and the only intersections occur at points that are on the boundary of the linestrings in the multilinestring.

If the given geometry is empty, then 1 is returned. If the geometry is null, null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types that represents the geometry that is to be tested.

Return type

INTEGER

Examples

In this example, several geometries are created and checked if they are simple. The geometry with an ID of 4 is not considered simple because it contains more than one point that is the same. The geometry with an ID of 6 is not considered simple, because the linestring crosses over itself.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_geoms (id INTEGER, geometry ST_Geometry)

INSERT INTO sample_geoms VALUES
       (1, ST_Geometry(ST_Point('point EMPTY' ,0)))

INSERT INTO sample_geoms VALUES
       (2, ST_Geometry(ST_Point('point (21 33)' ,0)))

INSERT INTO sample_geoms VALUES
       (3, ST_Geometry(ST_MultiPoint('multipoint(10 10, 20 20, 30 30)' ,0)))

INSERT INTO sample_geoms VALUES
       (4, ST_Geometry(ST_MultiPoint('multipoint(10 10, 20 20, 30 30,
  20 20)' ,0)))

INSERT INTO sample_geoms VALUES
       (5, ST_Geometry(ST_LineString('linestring(60 60, 70 60, 70 70)' ,0)))

INSERT INTO sample_geoms VALUES
       (6, ST_Geometry(ST_LineString('linestring(20 20, 30 30, 30 20,
  20 30  )' ,0))

INSERT INTO sample_geoms VALUES
       (7, ST_Geometry(ST_Polygon('polygon((40 40, 50 40, 50 50,
  40 40  ))' ,0)))

SELECT id, ST_IsSimple(geometry) Is_Simple
FROM sample_geoms
```

Results:

```
ID          IS_SIMPLE
----------- -----------
          1           1
```

```
          2           1
          3           1
          4           0
          5           1
          6           0
          7           1
```

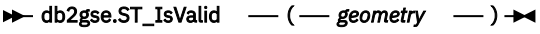
ST_IsValid

ST_IsValid takes a geometry as an input parameter and returns 1 if it is valid. Otherwise 0 (zero) is returned.

A geometry is valid only if all of the attributes in the structured type are consistent with the internal representation of geometry data, and if the internal representation is not corrupted.

If the given geometry is null, null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types that represents the geometry that is to be tested.

Return type

INTEGER

Example

This example creates several geometries and uses ST_IsValid to check if they are valid. All of the geometries are valid because the constructor routines, such as ST_Geometry, do not allow invalid geometries to be constructed.

```
SET CURRENT  PATH = CURRENT PATH, db2gse
CREATE TABLE sample_geoms (id INTEGER, geometry ST_Geometry)

INSERT INTO sample_geoms VALUES
       (1, ST_Geometry(ST_Point('point EMPTY',0)))

INSERT INTO sample_geoms VALUES
       (2, ST_Geometry(ST_Polygon('polygon((40 120, 90 120, 90 150, 40 150, 40 120))' ,0)))

INSERT INTO sample_geoms VALUES
       (3, ST_Geometry(ST_Multipoint('multipoint m (10 10 5, 50 10 6, 10 30 8)' ,0)))

INSERT INTO sample_geoms VALUES
       (4, ST_Geometry(ST_Linestring('linestring z (10 10 166, 20 10 168)',0)))

INSERT INTO sample_geoms VALUES
       (5, ST_Geometry(ST_Point('point zm (10 10 16 30)' ,0)))


SELECT id, ST_IsValid(geometry)  Is_Valid
FROM sample_geoms
```

Results:

```
 ID          IS_VALID
 ----------- -----------
           1           1
```

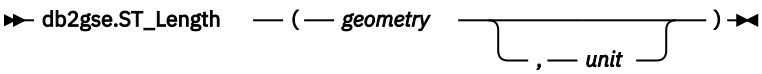
```
          2           1
          3           1
          4           1
          5           1
```

ST_Length

ST_Length takes a geometry of type ST_LineString or ST_MultiLineString and, optionally, a unit as input parameters and returns the length of the given geometry either in the default or given unit of measure.

If the given geometry is null or empty, null is returned.

Syntax



Parameter

geometry
A value of type ST_Linestring or ST_MultiLineString that represents the geometry for which the length is returned.

unit
A VARCHAR(128) value that identifies the units in which the length of the curve is measured. The supported units of measure are listed in the DB2GSE.ST_UNITS_OF_MEASURE catalog view.

If the *unit* parameter is omitted, the following rules are used to determine the unit in which the area is measured:

- If *geometry* is in a projected or geocentric coordinate system, the linear unit associated with this coordinate system is used.
- If *geometry* is in a geographic coordinate system, the angular unit associated with this coordinate system is used.

Restrictions on unit conversions: An error (SQLSTATE 38SU4) is returned if any of the following conditions occur:

- The geometry is in an unspecified coordinate system and the *unit* parameter is specified.
- The geometry is in a projected coordinate system and an angular unit is specified.
- The geometry is in a geographic coordinate system, and a linear unit is specified.

Return type

DOUBLE

Examples

Example 1

The following SQL statements create the SAMPLE_GEOMETRIES table and insert a line and a multiline into the table.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_geometries(id SMALLINT, spatial_type varchar(20),
  geometry ST_GEOMETRY)
INSERT INTO sample_geometries(id, spatial_type, geometry)
VALUES (1110, 'ST_LineString', ST_Geometry(ST_LineString
                        ('linestring(50 10, 50 20)', 1))),
INSERT INTO sample_geometries(id, spatial_type, geometry)
VALUES (1111, 'ST_MultiLineString', ST_Geometry(ST_MultiLineString
                        ('multilinestring((33 2, 34 3, 35 6),
```

```
                          (28 4, 29 5, 31 8, 43 12),
                          (39 3, 37 4, 36 7))', 1)))
```

Example 2

The following SELECT statement calculates the length of the line in the SAMPLE_GEOMETRIES table.

```
SELECT id, spatial_type, cast(ST_Length(geometry)
  AS DECIMAL(7, 2)) AS "Line Length"
FROM   sample_geometries
WHERE  id = 1110
```

Example 3

The following SELECT statement calculates the length of the multiline in the SAMPLE_GEOMETRIES table.

```
SELECT id, spatial_type, ST_Length(geometry)
  AS "multiline_length"
FROM   sample_geometries
WHERE  id = 1111
```

Results:

```
  ID     SPATIAL_TYPE         MULTILINE_LENGTH
------ -------------------- ------------------------
  1111 ST_MultiLineString     +2.76437123387202E+001
```

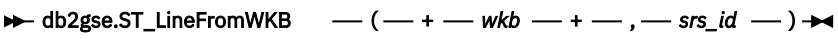
ST_LineFromWKB

ST_LineFromWKB takes a well-known binary representation of a linestring and a spatial reference system identifier as input parameters and returns the corresponding linestring.

If the given well-known binary representation is null, then null is returned.

The preferred version for this functionality is ST_LineString.

Syntax



Parameter

wkb
A value of type BLOB(4M) that contains the well-known binary representation of the resulting linestring.

srs_id
A value of type INTEGER that identifies the spatial reference system for the resulting linestring.

Return type

db2gse.ST_LineString

Example

In the following example, the lines of results have been reformatted for readability. The spacing in your results will vary according to your online display.

The following code uses the ST_LineFromWKB function to create and insert a line from a well-known binary representation. The row is inserted into the SAMPLE_LINES table with an ID and a line in spatial reference system 1 in WKB representation.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse

CREATE TABLE sample_lines(id SMALLINT, geometry ST_LineString, wkb BLOB(32k))
```

```
INSERT INTO sample_lines(id, geometry)
VALUES
    (1901, ST_LineString('linestring(850 250, 850 850)', 1) ),
INSERT INTO sample_lines(id, geometry)
VALUES
    (1902, ST_LineString('linestring(33 2, 34 3, 35 6)', 1) )

UPDATE sample_lines AS temp_correlated
SET    wkb = ST_AsBinary( geometry)
WHERE  id = temp_correlated.id

SELECT id, cast( ST_AsText (ST_LineFromWKB(wkb)) AS varchar(90)) line
FROM   sample_lines
```

Results:

```
ID     LINE
------ ------------------------------------------------------------------
  1901 LINESTRING ( 850.00000000 250.00000000, 850.00000000 850.00000000)

  1902 LINESTRING ( 33.00000000 2.00000000, 34.00000000 3.00000000,
35.00000000 6.00000000)
```

ST_LineString

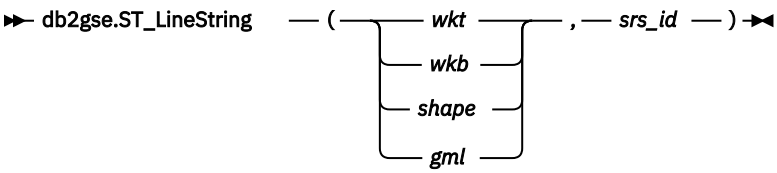
The ST_LineString function has two variations.

In the first variation, ST_LineString constructs a linestring from a well-known text representation, a well-known binary representation, an ESRI shape representation, or a Geography Markup Language (GML) representation. A spatial reference system identifier can be provided optionally to identify the spatial reference system that the resulting linestring is in.

In the second variation, ST_LineString takes ST_Geometry as an input parameter and casts the output type to ST_LineString. If the given geometry is null, then null is returned.

Syntax

Variation 1



Parameters

wkt
A value of type CLOB(8M) that contains the well-known text representation of the resulting linestring. If the well-known text representation is null, then null is returned.

wkb
A value of type BLOB(4M) that contains the well-known binary representation of the resulting linestring. If the well-known binary representation is null, then null is returned.

shape
A value of type BLOB(4M) that contains the shape representation of the resulting linestring. If the shape representation is null, then null is returned.

gml
A value of type CLOB(8M) that contains the GML representation of the geometry. If the GML representation is null, then null is returned.

srs_id
A value of type INTEGER that identifies the spatial reference system for the resulting linestring.

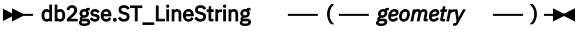
If *srs_id* does not identify a spatial reference system listed in the catalog view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS, an error is returned (SQLSTATE 38SU1).

Return type

db2gse.ST_LineString

Syntax

Variation 2



Parameter

geometry
A value of type ST_Geometry.

Return type

ST_LineString

Example

The following example uses the ST_LineString function to create linestrings from WKT and GML representations and inserts two rows into the sample_lines table with an ID and line in spatial reference system 1.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_lines(id SMALLINT, geometry ST_LineString)
INSERT INTO sample_lines(id, geometry)
VALUES
    (1110, ST_LineString('linestring(850 250, 850 850)', 1) ),
    (1111, ST_LineString('<gml:LineString srsName=";EPSG:4269";><gml:coord>
                         <gml:X>90</gml:X><gml:Y>90</gml:Y>
                         <gml:coord><gml:coord><gml:X>100</gml:X>
                         <gml:Y>100</gml:Y><gml:coord>
                         <gml:LineString>', 1) )
SELECT id, cast(geometry..ST_AsText AS varchar(75)) AS linestring
FROM   sample_lines
```

Results:

```
ID     LINESTRING
------ ---------------------------------------------------------------
1110   LINESTRING ( 850.00000000 250.00000000, 850.00000000 850.00000000)
1111   LINESTRING ( 90.00000000 90.00000000, 100.00000000 100.00000000)
```

ST_LocateAlong

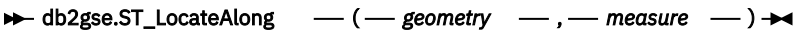
ST_LocateAlong takes a geometry and a measure as input parameters and returns a multipoint or multilinestring of that part of the given geometry that has exactly the specified measure of the given geometry that contains the specified measure.

For points and multipoints, all the points with the specified measure are returned. For linestrings, multilinestrings, polygons, and multipolygons, interpolation is performed to compute the result. The computation for polygons and multipolygons is performed on the boundary of the geometry.

For points and multipoints, if the given measure is not found, then an empty geometry is returned. For all other geometries, if the given measure is lower than the lowest measure in the geometry or higher than the highest measure in the geometry, then an empty geometry is returned.

If the given geometry is null, then null is returned.

Syntax



Parameters

geometry
A value of one of the seven distinct spatial data types that represents the geometry.

measure
A value of type DOUBLE that is the measure that the parts of *geometry* that must be included in the result.

Return type

db2gse.ST_Geometry

Examples

Example 1

The following CREATE TABLE statement creates the SAMPLE_GEOMETRIES table. The SAMPLE_GEOMETRIES table has two columns. The ID column uniquely identifies each row, and the GEOMETRY column ST_Geometry stores sample geometry.

```
CREATE TABLE sample_geometries(id SMALLINT, geometry ST_GEOMETRY)
```

The following INSERT statements insert two rows. The first is a linestring, and the second is a multipoint.

```
INSERT INTO sample_geometries(id, geometry)
  VALUES (1, ST_Geometry(ST_LineString('linestring m (2 2 3, 3 5 3, 3 3 6,
         4 4 8)', 1))),
INSERT INTO sample_geometries(id, geometry)
  VALUES (2, ST_Geometry(ST_MultiPoint('multipoint m (2 2 3, 3 5 3, 3 3 6,
         4 4 6, 5 5 6, 6 6 8)', 1)))
```

Example 2

In the following SELECT statement and the corresponding result set, the ST_LocateAlong function finds points with a measure of 7. The first row returns a point, and the second row returns an empty point. For linear features (a geometry with a dimension that is greater than 0), the ST_LocateAlong function can interpolate the point; however, for multipoints the target measure must match exactly.

```
SELECT id, cast(ST_AsText(ST_LocateAlong(geometry, 7))
AS varchar(45)) AS measure_7
FROM sample_geometries
```

Results:

```
  ID       MEASURE_7
--------- ------------------------------
   1      POINT M ( 3.50000000 3.50000000 7.00000000)
   2      POINT EMPTY
```

Example 3

In the following SELECT statement and the corresponding result set, the ST_LocateAlong function returns a point and a multipoint. The target measure of 6 matches the measures in both the ST_LocateAlong and the multipoint source data.

```
SELECT id, cast(ST_AsText(ST_LocateAlong(geometry, 6))
AS varchar(120)) AS measure_6
FROM sample_geometries
```

Results:

```
  ID       MEASURE_6
-------- ------------------------------
   1      POINT M ( 3.00000000 3.00000000 6.00000000)
   2      MULTIPOINT M ( 3.00000000 3.00000000 6.00000000,
          4.00000000 4.00000000 6.00000000,
          5.00000000 5.00000000 6.00000000)
```

ST_LocateBetween

ST_LocateBetween takes a geometry and two M coordinates (measures) as input parameters and returns that part of the given geometry that represents the set of disconnected paths or points between the two M coordinates.

For linestrings, multilinestrings, polygons, and multipolygons, interpolation is performed to compute the result. The resulting geometry is represented in the spatial reference system of the given geometry. If the given geometry is a polygon or multipolygon, then ST_LOCATEBETWEEN is applied to the exterior and interior rings of the geometry.

If none of the parts of the given geometry are in the interval defined by the given M coordinates, then an empty geometry is returned. If the given geometry is null , then null is returned.

The resulting geometry is represented in the most appropriate spatial type. If it can be represented as a point, linestring, or polygon, then one of those types is used. Otherwise, the multipoint, multilinestring, or multipolygon type is used.

Syntax



Parameters

geometry
A value of one of the seven distinct spatial data types that represents the geometry.

startM
A value of type DOUBLE that represents the lower bound of the measure interval. If this value is null, no lower bound is applied.

endM
A value of type DOUBLE that represents the upper bound of the measure interval. If this value is null, no upper bound is applied.

Return type

db2gse.ST_Geometry

Example

In this example, a researcher uses the M coordinate to record data that she collects about pH values. The researcher collects the pH values of the soil at specific locations along a highway. She records the X and Y coordinates of each location and the pH value of the soil.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_lines (id INTEGER, geometry ST_LineString)
INSERT INTO sample_lines
  VALUES (1, ST_Geometry(ST_LineString ('linestring m (2 2 3, 3 5 3, 3 3 6,
         4 4 6, 5 5 6, 6 6 8)', 1 )) )
```

To find the path where the acidity of the soil varies between 4 and 6, the researcher uses the following SELECT statement:

```
SELECT id, CAST( ST_AsText( ST_LocateBetween( geometry, 4, 6) )
AS VARCHAR(150) ) MEAS_BETWEEN_4_AND_6
FROM sample_lines
```

Results:

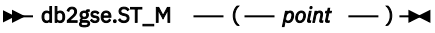
```
  ID       MEAS_BETWEEN_4_AND_6
-------- ----------------------------
   1      LINESTRING M (3.00000000 4.33333300 4.00000000,
          3.00000000 3.00000000 6.00000000,
          4.00000000 4.00000000 6.00000000,
          5.00000000 5.00000000 6.00000000)
```

ST_M

The ST_M function takes a point as an input parameter and return its M (measure) coordinate.

If the specified point is null or is empty, then null is returned.

Syntax



Parameters

point
A value of type ST_Point for which the M coordinate is returned or modified.

Return type

DOUBLE

Examples

Example 1

This example illustrates the use of the ST_M function. Three points are created and inserted into the SAMPLE_POINTS table. They are all in the spatial reference system that has an ID of 1.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_points (id INTEGER, geometry ST_Point);

INSERT INTO sample_points
  VALUES (1, ST_Point (2, 3, 32, 5, 1));

INSERT INTO sample_points
  VALUES (2, ST_Point (4, 5, 20, 4, 1));

INSERT INTO sample_points
  VALUES (3, ST_Point (3, 8, 23, 7, 1));
```

Example 2

This example finds the M coordinate of the points in the SAMPLE_POINTS table.

```
SELECT id, ST_M (geometry) M_COORD
  FROM sample_points;
```

Results:

```
ID          M_COORD
----------- ------------------------
          1 +5.00000000000000E+000
```

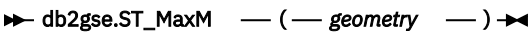
```
          2 +4.00000000000000E+000
          3 +7.00000000000000E+000
```

ST_MaxM

ST_MaxM takes a geometry as an input parameter and returns its maximum M coordinate.

If the given geometry is null or is empty, or if it does not have M coordinates, then null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types for which the maximum M coordinate is returned.

Return type

DOUBLE

Examples

Example 1

This example illustrates the use of the ST_MaxM function. Three polygons are created and inserted into the SAMPLE_POLYS table.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_polys (id INTEGER, geometry ST_Polygon);

INSERT INTO sample_polys
  VALUES (1, ST_Polygon('polygon zm ((110 120 20 3,
                                      110 140 22 3,
                                      120 130 26 4,
                                      110 120 20 3))', 0) );

INSERT INTO sample_polys
  VALUES (2, ST_Polygon('polygon zm ((0 0 40 7,
                                      0 4 35 9,
                                      5 4 32 12,
                                      5 0 31 5,
                                      0 0 40 7))', 0) );

INSERT INTO sample_polys
  VALUES (3, ST_Polygon('polygon zm ((12 13 10 16,
                                      8 4 10 12,
                                      9 4 12 11,
                                      12 13 10 16))', 0) );
```

Example 2

This example finds the maximum M coordinate of each polygon in SAMPLE_POLYS.

```
SELECT id, CAST ( ST_MaxM(geometry) AS INTEGER) MAX_M
  FROM sample_polys;
```

Results:

```
ID          MAX_M
----------- ------------
          1            4
```

```
          2           12
          3           16
```

Example 3

This example finds the maximum M coordinate that exists for all polygons in the GEOMETRY column.

```
SELECT CAST ( MAX ( ST_MaxM(geometry) ) AS INTEGER) OVERALL_MAX_M
  FROM sample_polys;
```

Results:

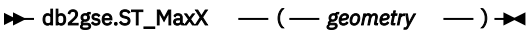
```
OVERALL_MAX_M
--------------
            16
```

ST_MaxX

ST_MaxX takes a geometry as an input parameter and returns its maximum X coordinate.

If the given geometry is null or is empty, then null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct data types for which the maximum X coordinate is returned.

Return type

DOUBLE

Examples

Example 1

This example illustrates the use of the ST_MaxX function. Three polygons are created and inserted into the SAMPLE_POLYS table. The third example illustrates how you can use all of the functions that return the maximum and minimum coordinate values to assess the spatial range of the geometries that are stored in a particular spatial column.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_polys (id INTEGER, geometry ST_Polygon);

INSERT INTO sample_polys
  VALUES (1, ST_Polygon('polygon zm ((110 120 20 3,
                                      110 140 22 3,
                                      120 130 26 4,
                                      110 120 20 3))', 0) );

INSERT INTO sample_polys
  VALUES (2, ST_Polygon('polygon zm ((0 0 40 7,
                                      0 4 35 9,
                                      5 4 32 12,
                                      5 0 31 5,
                                      0 0 40 7))', 0) );

INSERT INTO sample_polys
  VALUES (3, ST_Polygon('polygon zm ((12 13 10 16,
                                      8 4 10 12,
                                      9 4 12 11,
                                      12 13 10 16))', 0) );
```

Example 2

This example finds the maximum X coordinate of each polygon in SAMPLE_POLYS.

```
SELECT id, CAST ( ST_MaxX(geometry) AS INTEGER) MAX_X_COORD
  FROM sample_polys;
```

Results:

```
ID          MAX_X_COORD
----------- ------------
          1          120
          2            5
          3           12
```

Example 3

This example finds the maximum X coordinate that exists for all polygons in the GEOMETRY column.

```
SELECT CAST ( MAX ( ST_MaxX(geometry) ) AS INTEGER) OVERALL_MAX_X
  FROM sample_polys;
```

Results:

```
OVERALL_MAX_X
--------------
          120
```

Example 4

This example finds the spatial extent (overall minimum to overall maximum) of all the polygons in the SAMPLE_POLYS table. This calculation is typically used to compare the actual spatial extent of the geometries to the spatial extent of the spatial reference system associated with the data to determine if the data has room to grow.

```
SELECT CAST ( MIN (ST_MinX (geometry)) AS INTEGER) MIN_X,
       CAST ( MIN (ST_MinY (geometry)) AS INTEGER) MIN_Y,
       CAST ( MIN (ST_MinZ (geometry)) AS INTEGER) MIN_Z,
       CAST ( MIN (ST_MinM (geometry)) AS INTEGER) MIN_M,
       CAST ( MAX (ST_MaxX (geometry)) AS INTEGER) MAX_X,
       CAST ( MAX (ST_MaxY (geometry)) AS INTEGER) MAX_Y,
       CAST ( MAX (ST_MaxZ (geometry)) AS INTEGER) MAX_Z,
       CAST ( MAX (ST_MaxM (geometry)) AS INTEGER) MAX_M,
  FROM sample_polys;
```

Results:

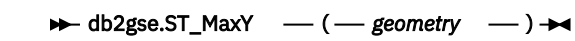
```
MIN_X     MIN_Y     MIN_Z     MIN_M     MAX_X     MAX_Y     MAX_Z     MAX_M
--------- --------- --------- --------- --------- --------- --------- ----------
        0         0        10         3       120       140        40         16
```

ST_MaxY

ST_MaxY takes a geometry as an input parameter and returns its maximum Y coordinate.

If the given geometry is null or is empty, then null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types for which the maximum Y coordinate is returned.

Return type

DOUBLE

Examples

Example 1

This example illustrates the use of the ST_MaxY function. Three polygons are created and inserted into the SAMPLE_POLYS table.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_polys (id INTEGER, geometry ST_Polygon);

INSERT INTO sample_polys
  VALUES (1, ST_Polygon('polygon zm ((110 120 20 3,
                                      110 140 22 3,
                                      120 130 26 4,
                                      110 120 20 3))', 0) );

INSERT INTO sample_polys
  VALUES (2, ST_Polygon('polygon zm ((0 0 40 7,
                                      0 4 35 9,
                                      5 4 32 12,
                                      5 0 31 5,
                                      0 0 40 7))', 0) );

INSERT INTO sample_polys
  VALUES (3, ST_Polygon('polygon zm ((12 13 10 16,
                                      8 4 10 12,
                                      9 4 12 11,
                                      12 13 10 16))', 0) );
```

Example 2

This example finds the maximum Y coordinate of each polygon in SAMPLE_POLYS.

```
SELECT id, CAST ( ST_MaxY(geometry) AS INTEGER) MAX_Y
  FROM sample_polys;
```

Results:

```
ID          MAX_Y
----------- ------------
          1          140
          2            4
          3           13
```

Example 3

This example finds the maximum Y coordinate that exists for all polygons in the GEOMETRY column.

```
SELECT CAST ( MAX ( ST_MaxY(geometry) ) AS INTEGER) OVERALL_MAX_Y
  FROM sample_polys;
```

Results:

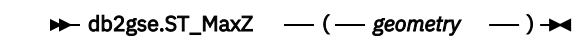
```
OVERALL_MAX_Y
--------------
           140
```

ST_MaxZ

ST_MaxZ takes a geometry as an input parameter and returns its maximum Z coordinate.

If the given geometry is null or is empty, or if it does not have Z coordinates, then null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types for which the maximum Z coordinate is returned.

Return type

DOUBLE

Examples

Example 1

This example illustrates the use of the ST_MaxZ function. Three polygons are created and inserted into the SAMPLE_POLYS table.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_polys (id INTEGER, geometry ST_Polygon);

INSERT INTO sample_polys
  VALUES (1, ST_Polygon('polygon zm ((110 120 20 3,
                                      110 140 22 3,
                                      120 130 26 4,
                                      110 120 20 3))', 0) );

INSERT INTO sample_polys
  VALUES (2, ST_Polygon('polygon zm ((0 0 40 7,
                                      0 4 35 9,
                                      5 4 32 12,
                                      5 0 31 5,
                                      0 0 40 7))', 0) );

INSERT INTO sample_polys
  VALUES (3, ST_Polygon('polygon zm ((12 13 10 16,
                                      8 4 10 12,
                                      9 4 12 11,
                                      12 13 10 16))', 0) );
```

Example 2

This example finds the maximum Z coordinate of each polygon in SAMPLE_POLYS.

```
SELECT id, CAST ( ST_MaxZ(geometry) AS INTEGER) MAX_Z
  FROM sample_polys;
```

Results:

```
ID          MAX_Z
----------- ------------
          1           26
          2           40
          3           12
```

Example 3

This example finds the maximum Z coordinate that exists for all polygons in the GEOMETRY column.

```
SELECT CAST ( MAX ( ST_MaxZ(geometry) ) AS INTEGER) OVERALL_MAX_Z
  FROM sample_polys;
```

Results:

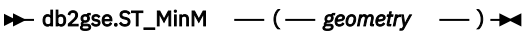
```
OVERALL_MAX_Z
-------------
           40
```

ST_MinM

ST_MinM takes a geometry as an input parameter and returns its minimum M coordinate.

If the given geometry is null or is empty, or if it does not have M coordinates, then null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types for which the minimum M coordinate is returned.

Return type

DOUBLE

Examples

Example 1

This example illustrates the use of the ST_MinM function. Three polygons are created and inserted into the SAMPLE_POLYS table.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_polys (id INTEGER, geometry ST_Polygon);

INSERT INTO sample_polys
  VALUES (1, ST_Polygon('polygon zm ((110 120 20 3,
                                      110 140 22 3,
                                      120 130 26 4,
                                      110 120 20 3))', 0) );

INSERT INTO sample_polys
  VALUES (2, ST_Polygon('polygon zm ((0 0 40 7,
                                      0 4 35 9,
                                      5 4 32 12,
                                      5 0 31 5,
                                      0 0 40 7))', 0) );

INSERT INTO sample_polys
  VALUES (3, ST_Polygon('polygon zm ((12 13 10 16,
                                      8 4 10 12,
                                      9 4 12 11,
                                      12 13 10 16))', 0) );
```

Example 2

This example finds the minimum M coordinate of each polygon in SAMPLE_POLYS.

```
SELECT id, CAST ( ST_MinM(geometry) AS INTEGER) MIN_M
  FROM sample_polys;
```

Results:

```
ID          MIN_M
----------- -----------
          1           3
          2           5
          3          11
```

Example 3

This example finds the minimum M coordinate that exists for all polygons in the GEOMETRY column.

```
SELECT CAST ( MIN ( ST_MinM(geometry) ) AS INTEGER) OVERALL_MIN_M
  FROM sample_polys;
```

Results:

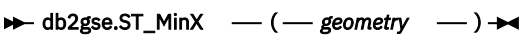
```
OVERALL_MIN_M
-------------
            3
```

ST_MinX

ST_MinX takes a geometry as an input parameter and returns its minimum X coordinate.

If the given geometry is null or is empty, then null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types for which the minimum X coordinate is returned.

Return type

DOUBLE

Examples

Example 1

This example illustrates the use of the ST_MinX function. Three polygons are created and inserted into the SAMPLE_POLYS table.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_polys (id INTEGER, geometry ST_Polygon);

INSERT INTO sample_polys
  VALUES (1, ST_Polygon('polygon zm ((110 120 20 3,
                                      110 140 22 3,
                                      120 130 26 4,
                                      110 120 20 3))', 0) );

INSERT INTO sample_polys
  VALUES (2, ST_Polygon('polygon zm ((0 0 40 7,
                                      0 4 35 9,
                                      5 4 32 12,
                                      5 0 31 5,
                                      0 0 40 7))', 0) );

INSERT INTO sample_polys
  VALUES (3, ST_Polygon('polygon zm ((12 13 10 16,
```

```
                                        8 4 10 12,
                                        9 4 12 11,
                                        12 13 10 16))', 0) );
```

Example 2

This example finds the minimum X coordinate of each polygon in SAMPLE_POLYS.

```
SELECT id, CAST ( ST_MinX(geometry) AS INTEGER) MIN_X
  FROM sample_polys;
```

Results:

```
ID          MIN_X
----------- ------------
          1          110
          2            0
          3            8
```

Example 3

This example finds the minimum X coordinate that exists for all polygons in the GEOMETRY column.

```
SELECT CAST ( MIN ( ST_MinX(geometry) ) AS INTEGER) OVERALL_MIN_X
  FROM sample_polys;
```

Results:

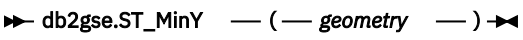
```
OVERALL_MIN_X
-------------
            0
```

ST_MinY

ST_MinY takes a geometry as an input parameter and returns its minimum Y coordinate.

If the given geometry is null or is empty, then null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types for which the minimum Y coordinate is returned.

Return type

DOUBLE

Examples

Example 1

This example illustrates the use of the ST_MinY function. Three polygons are created and inserted into the SAMPLE_POLYS table.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_polys (id INTEGER, geometry ST_Polygon);

INSERT INTO sample_polys
  VALUES (1, ST_Polygon('polygon zm ((110 120 20 3,
```

```
                                      110 140 22 3,
                                      120 130 26 4,
                                      110 120 20 3))', 0) );

INSERT INTO sample_polys
  VALUES (2, ST_Polygon('polygon zm ((0 0 40 7,
                                      0 4 35 9,
                                      5 4 32 12,
                                      5 0 31 5,
                                      0 0 40 7))', 0) );

INSERT INTO sample_polys
  VALUES (3, ST_Polygon('polygon zm ((12 13 10 16,
                                      8 4 10 12,
                                      9 4 12 11,
                                      12 13 10 16))', 0) );
```

Example 2

This example finds the minimum Y coordinate of each polygon in SAMPLE_POLYS.

```
SELECT id, CAST ( ST_MinY(geometry) AS INTEGER) MIN_Y
  FROM sample_polys;
```

Results:

```
ID          MIN_Y
----------- ------------
          1          120
          2            0
          3            4
```

Example 3

This example finds the minimum Y coordinate that exists for all polygons in the GEOMETRY column.

```
SELECT CAST ( MIN ( ST_MinY(geometry) ) AS INTEGER) OVERALL_MIN_Y
  FROM sample_polys;
```

Results:

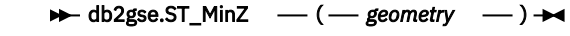
```
OVERALL_MIN_Y
-------------
            0
```

ST_MinZ

ST_MinZ takes a geometry as an input parameter and returns its minimum Z coordinate.

If the given geometry is null or is empty, or if it does not have Z coordinates, then null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types for which the minimum Z coordinate is returned.

Return type

DOUBLE

Examples

Example 1

This example illustrates the use of the ST_MinZ function. Three polygons are created and inserted into the SAMPLE_POLYS table.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_polys (id INTEGER, geometry ST_Polygon);

INSERT INTO sample_polys
  VALUES (1, ST_Polygon('polygon zm ((110 120 20 3,
                                      110 140 22 3,
                                      120 130 26 4,
                                      110 120 20 3))', 0) );

INSERT INTO sample_polys
  VALUES (2, ST_Polygon('polygon zm ((0 0 40 7,
                                      0 4 35 9,
                                      5 4 32 12,
                                      5 0 31 5,
                                      0 0 40 7))', 0) );

INSERT INTO sample_polys
  VALUES (3, ST_Polygon('polygon zm ((12 13 10 16,
                                       8 4 10 12,
                                       9 4 12 11,
                                      12 13 10 16))', 0) );
```

Example 2

This example finds the minimum Z coordinate of each polygon in SAMPLE_POLYS.

```
SELECT id, CAST ( ST_MinZ(geometry) AS INTEGER) MIN_Z
  FROM sample_polys;
```

Results:

```
ID          MIN_Z
----------- ------------
          1           20
          2           31
          3           10
```

Example 3

This example finds the minimum Z coordinate that exists for all polygons in the GEOMETRY column.

```
SELECT CAST ( MIN ( ST_MinZ(geometry) ) AS INTEGER) OVERALL_MIN_Z
  FROM sample_polys;
```

Results:

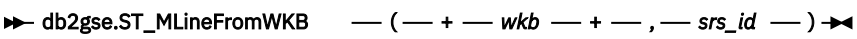
```
OVERALL_MIN_Z
--------------
             10
```

ST_MLineFromWKB

ST_MLineFromWKB takes a well-known binary representation of a multilinestring and a spatial reference system identifier as input parameters and returns the corresponding multilinestring.

If the given well-known binary representation is null, then null is returned.

Syntax



Parameters

wkb
A value of type BLOB(4M) that contains the well-known binary representation of the resulting multilinestring.

srs_id
A value of type INTEGER that identifies the spatial reference system for the resulting multilinestring.

Return type

db2gse.ST_MultiLineString

Example

In the following example, the lines of results have been reformatted for readability. The spacing in your results will vary according to your online display.

This example illustrates how ST_MLineFromWKB can be used to create a multilinestring from its well-known binary representation. The geometry is a multilinestring in spatial reference system 1. In this example, the multilinestring gets stored with ID = 10 in the GEOMETRY column of the SAMPLE_MLINES table, and then the WKB column is updated with its well-known binary representation (using the ST_AsBinary function). Finally, the ST_MLineFromWKB function is used to return the multilinestring from the WKB column. The X and Y coordinates for this geometry are:

- Line 1: (61, 2) (64, 3) (65, 6)
- Line 2: (58, 4) (59, 5) (61, 8)
- Line 3: (69, 3) (67, 4) (66, 7) (68, 9)

The SAMPLE_MLINES table has a GEOMETRY column, where the multilinestring is stored, and a WKB column, where the multilinestring's well-known binary representation is stored.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_mlines (id INTEGER, geometry ST_MultiLineString,
       wkb BLOB(32K))

INSERT INTO sample_mlines
  VALUES (10, ST_MultiLineString ('multilinestring
       ( (61 2, 64 3, 65 6),
         (58 4, 59 5, 61 8),
         (69 3, 67 4, 66 7, 68 9) )', 1) )

UPDATE sample_mlines AS temporary_correlated
  SET wkb = ST_AsBinary( geometry )
  WHERE id = temporary_correlated.id
```

In the following SELECT statement, the ST_MLineFromWKB function is used to retrieve the multilinestring from the WKB column.

```
SELECT id, CAST( ST_AsText( ST_MLineFromWKB (wkb) )
       AS VARCHAR(280) ) MULTI_LINE_STRING
  FROM sample_mlines
  WHERE id = 10
```

Results:

```
ID         MULTI_LINE_STRING
---------- ------------------------------------------------------------------
        10 MULTILINESTRING (( 61.00000000 2.00000000, 64.00000000 3.00000000,
```

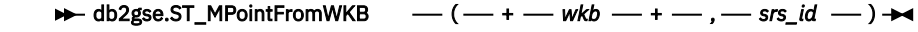
```
              65.00000000 6.00000000),
                            ( 58.00000000 4.00000000, 59.00000000 5.00000000,
              61.00000000 8.0000000),
                            ( 69.00000000 3.00000000, 67.00000000 4.00000000,
              66.00000000 7.00000000, 68.00000000 9.00000000 ))
```

ST_MPointFromWKB

ST_MPointFromWKB takes a well-known binary representation of a multipoint and a spatial reference system identifier as input parameters and returns the corresponding multipoint.

If the given well-known binary representation is null, then null is returned.

Syntax



Parameters

wkb
A value of type BLOB(4M) that contains the well-known binary representation of the resulting multipoint.

srs_id
A value of type INTEGER that identifies the spatial reference system for the resulting multipoint.

Return type

db2gse.ST_MultiPoint

Example

In the following example, the lines of results have been reformatted for readability. The spacing in your results will vary according to your online display.

This example illustrates how ST_MPointFromWKB can be used to create a multipoint from its well-known binary representation. The geometry is a multipoint in spatial reference system 1. In this example, the multipoint gets stored with ID = 10 in the GEOMETRY column of the SAMPLE_MPOINTS table, and then the WKB column is updated with its well-known binary representation (using the ST_AsBinary function). Finally, the ST_MPointFromWKB function is used to return the multipoint from the WKB column. The X and Y coordinates for this geometry are: (44, 14) (35, 16) (24, 13).

The SAMPLE_MPOINTS table has a GEOMETRY column, where the multipoint is stored, and a WKB column, where the multipoint's well-known binary representation is stored.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_mpoints (id INTEGER, geometry ST_MultiPoint,
    wkb BLOB(32K))

INSERT INTO sample_mpoints
  VALUES (10, ST_MultiPoint ('multipoint ( 4 14, 35 16, 24 13)', 1))

UPDATE sample_mpoints AS temporary_correlated
  SET wkb = ST_AsBinary( geometry )
  WHERE id = temporary_correlated.id
```

In the following SELECT statement, the ST_MPointFromWKB function is used to retrieve the multipoint from the WKB column.

```
SELECT id, CAST( ST_AsText( ST_MPointFromWKB (wkb)) AS VARCHAR(100)) MULTIPOINT
  FROM sample_mpoints
  WHERE id = 10
```

Results:

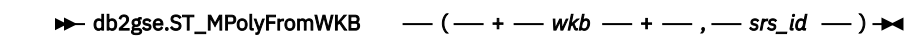
```
ID         MULTIPOINT
---------- ------------------------------------------------------------------
        10 MULTIPOINT (44.00000000 14.00000000, 35.00000000
              16.00000000, 24.00000000 13.00000000)
```

ST_MPolyFromWKB

ST_MPolyFromWKB takes a well-known binary representation of a multipolygon and a spatial reference system identifier as input parameters and returns the corresponding multipolygon.

If the given well-known binary representation is null, then null is returned.

Syntax



Parameters

wkb
A value of type BLOB(4M) that contains the well-known binary representation of the resulting multipolygon.

srs_id
A value of type INTEGER that identifies the spatial reference system for the resulting multipolygon.

Return type

db2gse.ST_MultiPolygon

Example

In the following example, the lines of results have been reformatted for readability. The spacing in your results will vary according to your online display.

This example illustrates how ST_MPolyFromWKB can be used to create a multipolygon from its well-known binary representation. The geometry is a multipolygon in spatial reference system 1. In this example, the multipolygon gets stored with ID = 10 in the GEOMETRY column of the SAMPLE_MPOLYS table, and then the WKB column is updated with its well-known binary representation (using the ST_AsBinary function). Finally, the ST_MPolyFromWKB function is used to return the multipolygon from the WKB column. The X and Y coordinates for this geometry are:

- Polygon 1: (1, 72) (4, 79) (5, 76) (1, 72)
- Polygon 2: (10, 20) (10, 40) (30, 41) (10, 20)
- Polygon 3: (9, 43) (7, 44) (6, 47) (9, 43)

The SAMPLE_MPOLYS table has a GEOMETRY column, where the multipolygon is stored, and a WKB column, where the multipolygon's well-known binary representation is stored.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_mpolys (id INTEGER,
    geometry ST_MultiPolygon, wkb BLOB(32K))

INSERT INTO sample_mpolys
  VALUES (10, ST_MultiPolygon ('multipolygon
        (( (1 72, 4 79, 5 76, 1 72),
         (10 20, 10 40, 30 41, 10 20),
         (9 43, 7 44, 6 47, 9 43) ))', 1))

UPDATE sample_mpolys AS temporary_correlated
  SET wkb = ST_AsBinary( geometry )
```

```
  WHERE id = temporary_correlated.id
```

In the following SELECT statement, the ST_MPolyFromWKB function is used to retrieve the multipolygon from the WKB column.

```
SELECT id, CAST( ST_AsText( ST_MPolyFromWKB (wkb) )
    AS VARCHAR(320) ) MULTIPOLYGON
  FROM sample_mpolys
  WHERE id = 10
```

Results:

```
ID         MULTIPOLYGON
---------- ----------------------------------------------------------------
        10 MULTIPOLYGON ((( 10.00000000 20.00000000, 30.00000000
              41.00000000, 10.00000000 40.00000000, 10.00000000
              20.00000000)),
                          ( 1.00000000 72.00000000, 5.00000000
              76.00000000, 4.00000000 79.0000000, 1.00000000
              72,00000000)),
                          ( 9.00000000 43.00000000, 6.00000000
               47.00000000, 7.00000000 44.00000000, 9.00000000
               43.00000000 )))
```

ST_MultiLineString

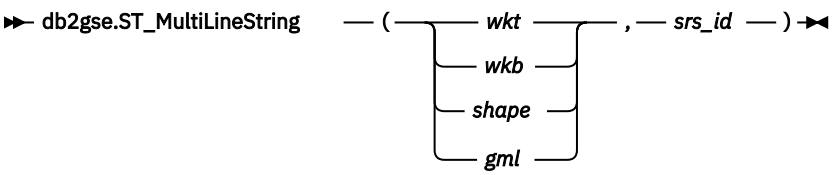
The ST_MultiLineString function has two variations.

In the first variation, ST_MultiLineString constructs a multilinestring from a well-known text representation, a well-known binary representation, an ESRI shape representation, or a Geography Markup Language (GML) representation. An optional spatial reference system identifier can be specified to identify the spatial reference system that the resulting multilinestring is in.

In the second variation, ST_MultiLineString takes ST_Geometry as an input parameter and casts the output type to ST_MultiLineString. If the given geometry is null, then null is returned.

Syntax

Variation 1



Parameters

wkt
A value of type CLOB(8M) that contains the well-known text representation of the resulting multilinestring. If the well-known text representation is null, then null is returned.

wkb
A value of type BLOB(4M) that contains the well-known binary representation of the resulting multilinestring. If the well-known binary representation is null, then null is returned.

shape
A value of type BLOB(4M) that contains the shape representation of the resulting multilinestring. If the shape representation is null, then null is returned.

gml
A value of type CLOB(8M) that contains the GML representation of the geometry. If the GML representation is null, then null is returned.

srs_id
A value of type INTEGER that identifies the spatial reference system for the resulting multilinestring.

If *srs_id* does not identify a spatial reference system listed in the catalog view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS, an error is returned (SQLSTATE 38SU1).

Return type

db2gse.ST_MultiLineString

Syntax

Variation 2



Parameter

geometry
A value of type ST_Geometry.

Return type

ST_MultiLineString

Example

In the following example, the lines of results have been reformatted for readability.

This example illustrates how ST_MultiLineString can be used to create and insert a multilinestring from its well-known text representation. The record that is inserted has ID = 1110, and the geometry is a multilinestring in spatial reference system 1. The multilinestring is in the well-known text representation of a multilinestring. The X and Y coordinates for this geometry are:

- Line 1: (33, 2) (34, 3) (35, 6)
- Line 2: (28, 4) (29, 5) (31, 8) (43, 12)
- Line 3: (39, 3) (37, 4) (36, 7)

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_mlines (id INTEGER,
                            geometry ST_MultiLineString);

INSERT INTO sample_mlines
  VALUES (1110,
          ST_MultiLineString ('multilinestring ( (33 2, 34 3, 35 6),
                                      (28 4, 29 5, 31 8, 43 12),
                                      (39 3, 37 4, 36 7) )', 1) );
```

The following SELECT statement returns the multilinestring that was recorded in the table:

```
SELECT id,
       CAST( ST_AsText( geometry ) AS VARCHAR(280) )
  MULTI_LINE_STRING
  FROM sample_mlines
  WHERE id = 1110;
```

Results:

```
ID         MULTI_LINE_STRING
------- ------------------------------------------------------------------
   1110 MULTILINESTRING (( 33.000000 2.000000, 34.000000 3.000000,
              35.000000 6.000000),
                        ( 28.000000 4.000000, 29.000000 5.000000,
              31.000000 8.000000, 43.000000 12.000000),
                        ( 39.000000 3.000000, 37.000000 4.000000,
              36.000000 7.000000 ))
```

ST_MultiPoint

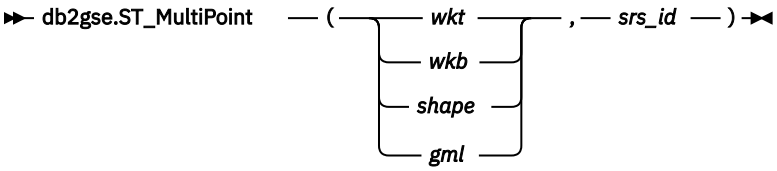
The ST_MultiPoint function has two variations.

In the first variation, ST_MultiPoint constructs a multipoint from a well-known text representation, a well-known binary representation, or an ESRI shape representation. An optional spatial reference system identifier can be specified to indicate the spatial reference system the resulting multipoint is in.

In the second variation, ST_MultiPoint takes ST_Geometry as an input parameter and casts the output type to ST_MultiPoint. If the given geometry is null, then null is returned.

Syntax

Variation 1



Parameters

wkt
A value of type CLOB(8M) that contains the well-known text representation of the resulting multipoint. If the well-known text representation is null, then null is returned.

wkb
A value of type BLOB(4M) that contains the well-known binary representation of the resulting multipoint. If the well-known binary representation is null, then null is returned.

shape
A value of type BLOB(4M) that contains the shape representation of the resulting multipoint. If the shape representation is null, then null is returned.

gml
A value of type CLOB(8M) that contains the GML representation of the geometry. If the GML representation is null, then null is returned.

srs_id
A value of type INTEGER that identifies the spatial reference system for the resulting multipoint.

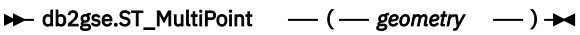
If *srs_id* does not identify a spatial reference system listed in the catalog view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS, an error is returned (SQLSTATE 38SU1).

Return type

db2gse.ST_MultiPoint

Syntax

Variation 2



Parameter

geometry
A value of type ST_Geometry.

Return type

ST_MultiPoint

Example

In the following example, the lines of results have been reformatted for readability.

This example illustrates how ST_MultiPoint can be used to create and insert a multipoint from its well-known text representation. The record that is inserted has ID = 1110, and the geometry is a multipoint in spatial reference system 1. The multipoint is in the well-known text representation of a multipoint. The X and Y coordinates for this geometry are: (1, 2) (4, 3) (5, 6).

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_mpoints (id INTEGER, geometry ST_MultiPoint);

INSERT INTO sample_mpoints
  VALUES (1110, ST_MultiPoint ('multipoint (1 2, 4 3, 5 6)', 1));
```

The following SELECT statement returns the multipoint that was recorded in the table:

```
SELECT id, CAST( ST_AsText(geometry) AS VARCHAR(90)) MULTIPOINT
  FROM sample_mpoints
  WHERE id = 1110;
```

Results:

```
ID         MULTIPOINT
---------- ----------------------------------------------------
      1110 MULTIPOINT (1.000000 2.000000, 4.000000
           3.000000, 5.000000 6.000000)
```

ST_MultiPolygon

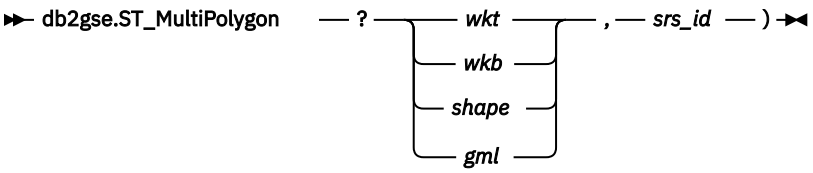
The ST_MultiPolygon function has two variations.

In the first variation, ST_MultiPolygon constructs a multipolygon from a well-known text representation, a well-known binary representation, an ESRI shape representation, or a Geography Markup Language (GML) representation. An optional spatial reference system identifier can be specified to identify the spatial reference system that the resulting multipolygon is in.

In the second variation, ST_MultiPolygon takes ST_Geometry as an input parameter and casts the output type to ST_MultiPolygon. If the given geometry is null, then null is returned.

Syntax

Variation 1



Parameters

wkt
A value of type CLOB(8M) that contains the well-known text representation of the resulting multipolygon. If the well-known text representation is null, then null is returned.

wkb
A value of type BLOB(4M) that contains the well-known binary representation of the resulting multipolygon. If the well-known binary representation is null, then null is returned.

shape
A value of type BLOB(4M) that contains the shape representation of the resulting multipolygon. If the shape representation is null, then null is returned.

gml
A value of type CLOB(8M) that contains the GML representation of the geometry. If the GML representation is null, then null is returned.

srs_id
A value of type INTEGER that identifies the spatial reference system for the resulting multipolygon.

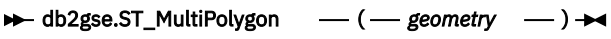
If *srs_id* does not identify a spatial reference system listed in the catalog view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS, an error is returned (SQLSTATE 38SU1).

Return type

db2gse.ST_MultiPolygon

Syntax

Variation 2



Parameter

geometry
A value of type ST_Geometry.

Return type

ST_MultiPolygon

Example

In the following example, the lines of results have been reformatted for readability.

This example illustrates how ST_MultiPolygon can be used to create and insert a multipolygon from its well-known text representation. The record that is inserted has ID = 1110, and the geometry is a multipolygon in spatial reference system 1. The multipolygon is in the well-known text representation of a multipolygon. The X and Y coordinates for this geometry are:

- Polygon 1: (3, 3) (4, 6) (5, 3) (3, 3)
- Polygon 2: (8, 24) (9, 25) (1, 28) (8, 24)

- Polygon 3: (13, 33) (7, 36) (1, 40) (10, 43) (13, 33)

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_mpolys (id INTEGER, geometry ST_MultiPolygon);

INSERT INTO sample_mpolys
  VALUES (1110,
          ST_MultiPolygon ('multipolygon (( (3 3, 4 6, 5 3, 3 3),
                                (8 24, 9 25, 1 28, 8 24),
                                (13 33, 7 36, 1 40, 10 43, 13 33) ))', 1) );
```

The following SELECT statement returns the multipolygon that was recorded in the table:

```
SELECT id, CAST( ST_AsText( geometry ) AS VARCHAR(350) ) MULTI_POLYGON
  FROM sample_mpolys
  WHERE id = 1110;
```

Results:

```
ID         MULTI_POLYGON
------- ------------------------------------------------------------------
   1110 MULTIPOLYGON ((( 13.000000 33.000000, 10.000000 43.000000,
           1.000000 40.000000, 7.000000 36.000000,
          13.000000 33.000000)),
                     (( 8.000000 24.000000, 9.000000 25.000000,
           1.000000 28.00000, 8.000000 24.000000)),
                      (( 3.000000 3.000000, 5.000000 3.000000,
           4.000000 6.000000, 3.000000 3.000000)))
```

ST_NumGeometries

ST_NumGeometries takes a geometry type as an input parameter and returns the number of geometries in the collection.

If the given geometry type is null or is empty, then null is returned.

Syntax



Parameter

geometry
A value of type ST_MultiPoint, ST_MultiLineString, or ST_MultiPolygon that represents the geometry type for which the number of geometries is returned.

Return Type

INTEGER

Example

Two geometry types are stored in the SAMPLE_GEOMCOLL table. One is a multipolygon, and the other is a multipoint. The ST_NumGeometries function determines how many individual geometries are within each geometry type.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_geomcoll (id INTEGER, geometry ST_Geometry)

INSERT INTO sample_geomcoll
  VALUES (1, ST_Geometry(ST_MultiPolygon
                              ('multipolygon (( (3 3, 4 6, 5 3, 3 3),
                              (8 24, 9 25, 1 28, 8 24),
                              (13 33, 7 36, 1 40, 10 43, 13 33) ))', 1)) )
```

```
INSERT INTO sample_geomcoll
  VALUES (2, ST_Geometry(ST_MultiPoint
                              ('multipoint (1 2, 4 3, 5 6, 7 6, 8 8)', 1)) )

SELECT id, ST_NumGeometries (geometry) NUM_GEOMS_IN_COLL
  FROM sample_geomcoll
```

Results:

```
ID          NUM_GEOMS_IN_COLL
----------- -----------------
          1                 3
          2                 5
```

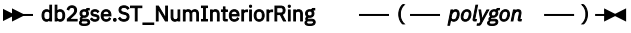
ST_NumInteriorRing

ST_NumInteriorRing takes a polygon as an input parameter and returns the number of its interior rings.

If the given polygon is null or is empty, then null is returned.

If the polygon has no interior rings, then 0 (zero) is returned.

Syntax



Parameter

polygon
A value of type ST_Polygon that represents the polygon for which the number of interior rings is returned.

Return type

INTEGER

Example

The following example creates two polygons:

- One with two interior rings
- One without any interior rings

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_polys (id INTEGER, geometry ST_Polygon)

INSERT INTO sample_polys
  VALUES (1, ST_Polygon('polygon
         ((40 120, 90 120, 90 150, 40 150, 40 120),
          (50 130, 60 130, 60 140, 50 140, 50 130),
          (70 130, 80 130, 80 140, 70 140, 70 130))' , 0) )

INSERT INTO sample_polys
  VALUES (2, ST_Polygon('polygon ((5 15, 50 15, 50 105, 5 15))' , 0) )
```

The ST_NumInteriorRing function is used to return the number of rings in the geometries in the table:

```
SELECT id, ST_NumInteriorRing(geometry) NUM_RINGS
FROM sample_polys
```

Results:

```
ID         NUM_RINGS
---------- --------------
         1              2
         2              0
```

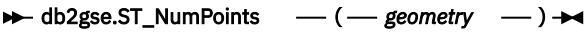
ST_NumPoints

ST_NumPoints takes a geometry as an input parameter and returns the number of points that were used to define that geometry.

For example, if the geometry is a polygon and five points were used to define that polygon, then the returned number is 5.

If the given geometry is null or is empty, then null is returned.

Syntax



Parameter

geometry
A value of one of the seven distinct spatial data types that represents the geometry for which the number of points is returned.

Return type

INTEGER

Example

A variety of geometries are stored in the table. The ST_NumPoints function determines how many points are within each geometry in the SAMPLE_GEOMETRIES table.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_geometries (spatial_type VARCHAR(18), geometry ST_Geometry);

INSERT INTO sample_geometries
  VALUES ('st_point',
          ST_GEOMETRY(ST_Point (2, 3, 0) ));

INSERT INTO sample_geometries
  VALUES ('st_linestring',
          ST_GEOMETRY(ST_LineString ('linestring (2 5, 21 3, 23 10)', 0) ));

INSERT INTO sample_geometries
  VALUES ('st_polygon',
          ST_GEOMETRY(ST_Polygon ('polygon ((110 120, 110 140, 120 130,
          110 120))', 0) ));

SELECT spatial_type, ST_NumPoints (geometry) NUM_POINTS
  FROM sample_geometries;
```

Results:

```
SPATIAL_TYPE    NUM_POINTS
--------------- ----------
st_point                 1
st_linestring            3
st_polygon               4
```

ST_Overlaps

The ST_Overlaps function takes two geometries as input parameters and returns 1 if the intersection of the geometries results in a geometry of the same dimension but is not equal to either of the given geometries. Otherwise, 0 (zero) is returned.

If any of the two geometries is null or is empty, then null is returned.

If the second geometry is not represented in the same spatial reference system as the first geometry, it will be converted to the other spatial reference system.

Syntax



Parameters

geometry1
A value of one of the seven distinct spatial data types that represents the geometry that is tested to overlap with *geometry2*.

geometry2
A value of one of the seven distinct spatial data types that represents the geometry that is tested to overlap with *geometry1*.

Return type

INTEGER

Examples

Example 1

This example illustrates the use of ST_Overlaps. Various geometries are created and inserted into the SAMPLE_GEOMETRIES table

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_geometries (id INTEGER, geometry ST_Geometry);

INSERT INTO sample_geometries
  VALUES
               (1, ST_GEOMETRY(ST_Point (10, 20, 1)));

INSERT INTO sample_geometries
  VALUES
        (2, ST_GEOMETRY(ST_Point ('point (41 41)', 1) ));

INSERT INTO sample_geometries
  VALUES
        (10, ST_GEOMETRY(ST_LineString ('linestring (1 10, 3 12, 10 10)', 1) ));

INSERT INTO sample_geometries
  VALUES
        (20, ST_GEOMETRY(ST_LineString ('linestring (50 10, 50 12, 45 10)', 1) ));

INSERT INTO sample_geometries
  VALUES
        (30, ST_GEOMETRY(ST_LineString ('linestring (50 12, 50 10, 60 8)', 1) ));

INSERT INTO sample_geometries
  VALUES
        (100, ST_GEOMETRY(ST_Polygon ('polygon ((0 0, 0 40, 40 40, 40 0,
              0 0))', 1) ));

INSERT INTO sample_geometries
  VALUES
        (110, ST_GEOMETRY(ST_Polygon ('polygon ((30 10, 30 30, 50 30, 50 10,
              30 10))', 1) ));

INSERT INTO sample_geometries
```

```
  VALUES
         (120, ST_GEOMETRY(ST_Polygon ('polygon ((0 50, 0 60, 40 60, 40 60,
               0 50))', 1) ));
```

Example 2

This example finds the IDs of points that overlap.

```
SELECT sg1.id, sg2.id
  CASE ST_Overlaps (sg1.geometry, sg2.geometry)
    WHEN 0 THEN 'Points_do_not_overlap'
    WHEN 1 THEN 'Points_overlap'
    END
  AS OVERLAP
  FROM sample_geometries sg1, sample_geometries sg2
  WHERE sg1.id < 10 AND sg2.id < 10 AND sg1.id >= sg2.id;
```

Results:

```
ID          ID         OVERLAP
----------- ---------- ------------------------
          1          1 Points_do_not_overlap
          2          1 Points_do_not_overlap
          2          2 Points_do_not_overlap
```

Example 3

This example finds the IDs of lines that overlap.

```
SELECT sg1.id, sg2.id
  CASE ST_Overlaps (sg1.geometry, sg2.geometry)
    WHEN 0 THEN 'Lines_do_not_overlap'
    WHEN 1 THEN 'Lines_overlap'
    END
  AS OVERLAP
  FROM sample_geometries sg1, sample_geometries sg2
  WHERE sg1.id >= 10 AND sg1.id < 100
    AND sg2.id >= 10 AND sg2.id < 100
    AND sg1.id >= sg2.id;
```

Results:

```
ID          ID         OVERLAP
----------- ---------- ------------------------
         10         10 Lines_do_not_overlap
         20         10 Lines_do_not_overlap
         30         10 Lines_do_not_overlap
         20         20 Lines_do_not_overlap
         30         20 Lines_overlap
         30         30 Lines_do_not_overlap
```

Example 4

This example finds the IDs of polygons that overlap.

```
SELECT sg1.id, sg2.id
  CASE ST_Overlaps (sg1.geometry, sg2.geometry)
    WHEN 0 THEN 'Polygons_do_not_overlap'
    WHEN 1 THEN 'Polygons_overlap'
    END
  AS OVERLAP
  FROM sample_geometries sg1, sample_geometries sg2
  WHERE sg1.id >= 100 AND sg2.id >= 100 AND sg1.id >= sg2.id;
```

Results:

```
ID          ID         OVERLAP
----------- ---------- ------------------------
        100        100 Polygons_do_not_overlap
```

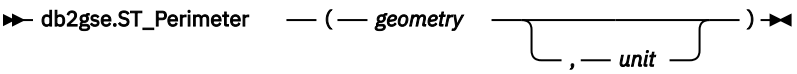
```
        110         100 Polygons_overlap
        120         100 Polygons_do_not_overlap
        110         110 Polygons_do_not_overlap
        120         110 Polygons_do_not_overlap
        120         120 Polygons_do_not_overlap
```

ST_Perimeter

ST_Perimeter takes a geometry type of polygon or multipolygon, and optionally, a unit as input parameters and returns the perimeter of the polygon or multipolygon. The perimeter is the length of its boundary as measured in the given units.

If the given polygon or multipolygon is null or is empty, null is returned.

Syntax



Parameters

geometry
A value of type ST_polygon or ST_Multipolygon for which the perimeter is returned.

unit
A VARCHAR(128) value that identifies the units in which the perimeter is measured. The supported units of measure are listed in the DB2GSE.ST_UNITS_OF_MEASURE catalog view.

If the *unit* parameter is omitted, the following rules are used to determine the unit in which the perimeter is measured:

- If *geometry* is in a projected or geocentric coordinate system, the linear unit associated with this coordinate system is used.
- If *geometry* is in a geographic coordinate system, the angular unit associated with this coordinate system is used.

Restrictions on unit conversions: An error (SQLSTATE 38SU4) is returned if any of the following conditions occur:

- The geometry is in an unspecified coordinate system and the *unit* parameter is specified.
- The geometry is in a projected coordinate system and an angular unit is specified.
- The geometry is in a geographic coordinate system and a linear unit is specified.

Check for an error case if the result is an overflow.

Return type

DOUBLE

Examples

The following examples illustrate the use of the ST_Perimeter function. These examples assume that you created a SAMPLE_POLYS table to hold a geometry with a perimeter of 18.

```
SET CURRENT PATH = CURRENT PATH, db2gse

CREATE TABLE sample_polys (id SMALLINT, geometry ST_Polygon)

INSERT INTO sample_polys
  VALUES (1, ST_Polygon ('polygon ((0 0, 0 4, 5 4, 5 0, 0 0))', 1))
```

Example 1

This example lists the ID and perimeter of the polygon.

```
SELECT id, ST_Perimeter (geometry) AS PERIMETER
  FROM sample_polys
```

Results:

```
ID          PERIMETER
----------- ------------------------
          1 +1.80000000000000E+001
```

Example 2

This example lists the ID and perimeter of the polygon with the perimeter measured in meters.

```
SELECT id, ST_Perimeter (geometry, 'METER') AS PERIMETER_METER
  FROM sample_polys
```

Results:

```
ID          PERIMETER_METER
----------- ------------------------
          1 +5.48641097282195E+000
```

ST_Point

The ST_Point function has two variations.

In the first variation, ST_Point constructs a point from one of the following inputs:

- A set of coordinates
- A well-known text representation
- A well-known binary representation
- An ESRI shape representation
- A Geography Markup Language (GML) representation

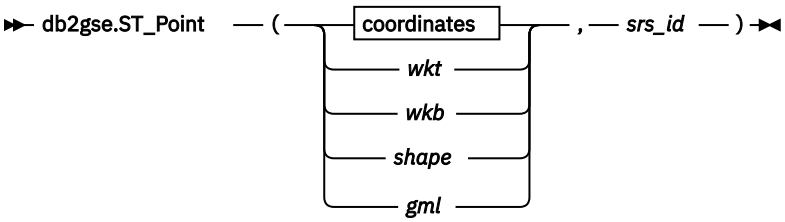
An optional spatial reference system identifier can be specified to indicate the spatial reference system that the resulting point is in.

If the point is constructed from coordinates, and if the X or Y coordinate is null, then an exception condition is raised (SQLSTATE 38SUP). If the Z or M coordinate is null, then the resulting point will not have a Z or M coordinate, respectively. If the point is constructed from its well-known text representation, and if the representation is null, then null is returned.

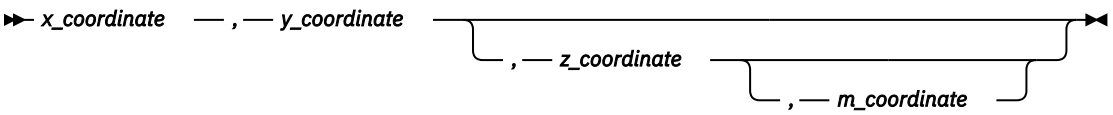
In the second variation, ST_Point takes ST_Geometry as an input parameter and casts the output type to ST_Point. If the given geometry is null, then null is returned.

Syntax

Variation 1



coordinates



Parameters

wkt
A value of type CLOB(8M) that contains the well-known text representation of the resulting point. If the well-known text representation is null, then null is returned.

wkb
A value of type BLOB(4M) that contains the well-known binary representation of the resulting point. If the well-known binary representation is null, then null is returned.

shape
A value of type BLOB(4M) that contains the shape representation of the resulting point. If the shape representation is null, then null is returned.

gml
A value of type CLOB(8M) that contains the GML representation of the geometry. If the GML representation is null, then null is returned.

srs_id
A value of type INTEGER that identifies the spatial reference system for the resulting point. This parameter is required.

If *srs_id* does not identify a spatial reference system listed in the catalog view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS, an error is returned (SQLSTATE 38SU1).

x_coordinate
A value of type DOUBLE that specifies the X coordinate for the resulting point.

y_coordinate
A value of type DOUBLE that specifies the Y coordinate for the resulting point.

z_coordinate
A value of type DOUBLE that specifies the Z coordinate for the resulting point.

If the *z_coordinate* parameter is omitted, the resulting point will not have a Z coordinate.

m_coordinate
A value of type DOUBLE that specifies the M coordinate for the resulting point.

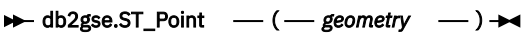
If the *m_coordinate* parameter is omitted, the resulting point will not have a measure.

Return type

db2gse.ST_Point

Syntax

Variation 2



Parameter

geometry
A value of type ST_Geometry.

Return type

ST_Point

Example

In the following examples, the lines of results have been reformatted for readability.

Example 1

This example illustrates how ST_Point can be used to create and insert points. The first point is created using a set of X and Y coordinates. The second point is created using its well-known text representation. Both points are geometries in spatial reference system 1.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_points (id INTEGER, geometry ST_Point);

INSERT INTO sample_points
  VALUES (1100, ST_Point (10, 20, 1) );

INSERT INTO sample_points
  VALUES (1101, ST_Point ('point (30 40)', 1) );
```

The following SELECT statement returns the points that were recorded in the table:

```
SELECT id, CAST( ST_AsText( geometry ) AS VARCHAR(90)) POINTS
  FROM sample_points;
```

Results:

```
ID         POINTS
---------- -----------------------------------
      1110 POINT ( 10.000000 20.000000)
      1101 POINT ( 30.000000 40.000000)
```

Example 2

This example inserts a record into the SAMPLE_POINTS table with ID 1103 and a point value with an X coordinate of 120, a Y coordinate of 358, an M coordinate of 34, but no Z coordinate.

```
INSERT INTO SAMPLE_POINTS(ID, GEOMETRY)
 VALUES(1103, db2gse.ST_Point(120, 358, CAST(NULL AS DOUBLE), 34, 1));

SELECT id, CAST( ST_AsText( geometry ) AS VARCHAR(90) ) POINTS
 FROM sample_points;
```

Results:

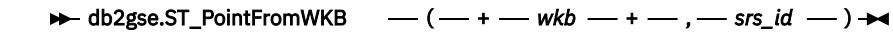
```
ID         POINTS
 ---------- -------------------------------------------------
       1100 POINT ( 10.000000 20.000000)
       1101 POINT ( 30.000000 40.000000)
       1103 POINT M ( 120.00000 358.00000 34)
```

ST_PointFromWKB

ST_PointFromWKB takes a well-known binary representation of a point and a spatial reference system identifier as input parameters and returns the corresponding point.

If the given well-known binary representation is null, then null is returned.

Syntax



Parameters

wkb
A value of type BLOB(4M) that contains the well-known binary representation of the resulting point.

srs_id
A value of type INTEGER that identifies the spatial reference system for the resulting point.

Return type

db2gse.ST_Point

Example

This example illustrates how ST_PointFromWKB can be used to create a point from its well-known binary representation. The geometries are points in spatial reference system 1. In this example, the points get stored in the GEOMETRY column of the SAMPLE_POLYS table, and then the WKB column is updated with their well-known binary representations (using the ST_AsBinary function). Finally, the ST_PointFromWKB function is used to return the points from the WKB column.

The SAMPLE_POINTS table has a GEOMETRY column, where the points are stored, and a WKB column, where the points' well-known binary representations are stored.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_points (id INTEGER, geometry ST_Point, wkb BLOB(32K))

INSERT INTO sample_points
  VALUES (10, ST_Point ('point (44 14)', 1) ),
INSERT INTO sample_points
  VALUES (11, ST_Point ('point (24 13)', 1))

UPDATE sample_points AS temporary_correlated
  SET wkb = ST_AsBinary( geometry )
  WHERE id = temporary_correlated.id
```

In the following SELECT statement, the ST_PointFromWKB function is used to retrieve the points from the WKB column.

```
SELECT id, CAST( ST_AsText( ST_PointFromWKB (wkb) ) AS VARCHAR(35) ) POINTS
  FROM sample_points
```

Results:

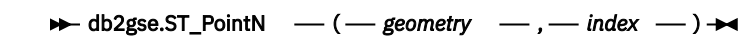
```
ID         POINTS
---------- -----------------------------------
        10 POINT ( 44.00000000 14.00000000)
        11 POINT ( 24.00000000 13.00000000)
```

ST_PointN

ST_PointN takes a linestring or a multipoint and an index as input parameters and returns that point in the linestring or multipoint that is identified by the index. The resulting point is represented in the spatial reference system of the given linestring or multipoint.

If the given linestring or multipoint is null or is empty, then null is returned.

Syntax



Parameters

geometry
A value of type ST_LineString or ST_MultiPoint that represents the geometry from which the point that is identified by *index* is returned.

index
A value of type INTEGER that identifies the *n*th point that is to be returned from *geometry*. If the index is smaller than 1 or larger than the number of points in the linestring or multipoint, then null is returned and a warning is returned (SQLSTATE 01HS2).

Return type

db2gse.ST_Point

Example

The following example shows the use of ST_PointN:

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_lines (id INTEGER, line ST_LineString)

INSERT INTO sample_lines
  VALUES (1, ST_LineString ('linestring (10 10, 5 5, 0 0, 10 0, 5 5, 0 10)', 0) )

SELECT id, CAST ( ST_AsText (ST_PointN (line, 2) ) AS VARCHAR(60) ) SECOND_INDEX
  FROM sample_lines
```

Results:

```
ID        SECOND_INDEX
--------- --------------------------------
        1 POINT (5.00000000 5.00000000)
```

ST_PointOnSurface

ST_PointOnSurface takes a polygon or a multipolygon as an input parameter and returns a point that is guaranteed to be in the interior of the polygon or multipolygon. This point is the paracentroid of the polygon.

The resulting point is represented in the spatial reference system of the given polygon or multipolygon.

If the given polygon or multipolygon is null or is empty, then null is returned.

Syntax



Parameter

geometry
A value of type ST_Polygon or ST_MultiPolygon that represents the geometry for which a point is returned.

Return type

db2gse.ST_Point

Example

In the following example, two polygons are created and then ST_PointOnSurface is used. One of the polygons has a hole in its center. The returned points are on the surface of the polygons. They are not necessarily at the exact center of the polygons.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_polys (id INTEGER, geometry ST_Polygon)

INSERT INTO sample_polys
  VALUES (1,
          ST_Polygon ('polygon ( (40 120, 90 120, 90 150, 40 150, 40 120) ,
                        (50 130, 80 130, 80 140, 50 140, 50 130) )' ,0) )
INSERT INTO sample_polys
  VALUES (2,
          ST_Polygon ('polygon ( (10 10, 50 10, 10 30, 10 10) )', 0) )

SELECT id, CAST (ST_AsText (ST_PointOnSurface (geometry) ) AS VARCHAR(80) )
  POINT_ON_SURFACE
  FROM sample_polys
```

Results:

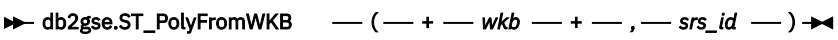
```
ID          POINT_ON_SURFACE
----------- ------------------------------------
          1 POINT ( 65.00000000 125.00000000)
          2 POINT ( 30.00000000 15.00000000)
```

ST_PolyFromWKB

ST_PolyFromWKB takes a well-known binary representation of a polygon and a spatial reference system identifier as input parameters and returns the corresponding polygon.

If the given well-known binary representation is null, then null is returned.

Syntax



Parameters

wkb
A value of type BLOB(4M) that contains the well-known binary representation of the resulting polygon.

srs_id
A value of type INTEGER that identifies the spatial reference system for the resulting polygon.

Return type

db2gse.ST_Polygon

Example

In the following example, the lines of results have been reformatted for readability. The spacing in your results will vary according to your online display.

This example illustrates how ST_PolyFromWKB can be used to create a polygon from its well-known binary representation. The geometry is a polygon in spatial reference system 1. In this example, the polygon gets stored with ID = 1115 in the GEOMETRY column of the SAMPLE_POLYS table, and then the WKB column is updated with its well-known binary representation (using the ST_AsBinary function).

Finally, the ST_PolyFromWKB function is used to return the multipolygon from the WKB column. The X and Y coordinates for this geometry are: (50, 20) (50, 40) (70, 30).

The SAMPLE_POLYS table has a GEOMETRY column, where the polygon is stored, and a WKB column, where the polygon's well-known binary representation is stored.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_polys (id INTEGER, geometry ST_Polygon,
  wkb BLOB(32K))

INSERT INTO sample_polys
  VALUES (10, ST_Polygon ('polygon ((50 20, 50 40, 70 30, 50 20))', 1) )

UPDATE sample_polys AS temporary_correlated
  SET wkb = ST_AsBinary( geometry )
  WHERE id = temporary_correlated.id
```

In the following SELECT statement, the ST_PolyFromWKB function is used to retrieve the polygon from the WKB column.

```
SELECT id, CAST( ST_AsText( ST_PolyFromWKB (wkb) )
  AS VARCHAR(120) ) POLYGON
  FROM sample_polys
  WHERE id = 1115
```

Results:

```
ID         POLYGON
---------- ------------------------------------------------------------
      1115 POLYGON (( 50.00000000 20.00000000, 70.00000000
              30.00000000,50.00000000 40.00000000, 50.00000000
              20.00000000))
```

ST_Polygon

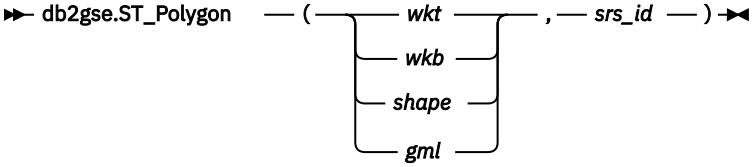
The ST_Polygon function has two variations.

In the first variation, ST_Polygon constructs a polygon from a well-known text representation, a well-known binary representation, an ESRI shape representation, or a Geography Markup Language (GML) representation. An optional spatial reference system identifier can be specified to identify the spatial reference system that the resulting polygon is in.

In the second variation, ST_Polygon takes ST_Geometry as an input parameter and casts the output type to ST_Polygon. If the given geometry is null, then null is returned.

Syntax

Variation 1



Parameters

wkt

A value of type CLOB(8M) that contains the well-known text representation of the resulting polygon. If the well-known text representation is null, then null is returned.

wkb
A value of type BLOB(4M) that contains the well-known binary representation of the resulting polygon. If the well-known binary representation is null, then null is returned.

shape
A value of type BLOB(4M) that contains the shape representation of the resulting polygon. If the shape representation is null, then null is returned.

gml
A value of type CLOB(8M) that contains the GML representation of the geometry. If the GML representation is null, then null is returned.

srs_id
A value of type INTEGER that identifies the spatial reference system for the resulting polygon.

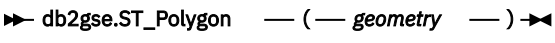
If *srs_id* does not identify a spatial reference system listed in the catalog view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS, an error is returned (SQLSTATE 38SU1).

Return type

db2gse.ST_Polygon

Syntax

Variation 2



Parameter

geometry
A value of type ST_Geometry.

Return type

ST_Polygon

Example

In the following example, the lines of results have been reformatted for readability.

This example illustrates how ST_Polygon can be used to create and insert a polygon. This polygon is created using its well-known text representation. The X and Y coordinates for this polygon are: (110, 120) (110, 140) (120, 130). The geometry is in spatial reference system 1.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_polys (id INTEGER, geometry ST_Polygon);

INSERT INTO sample_polys
  VALUES (1101,
          ST_Polygon ('polygon
                    ((110 120, 110 140, 120 130, 110 120))', 1));
```

The following SELECT statement returns the polygon that was recorded in the table:

```
SELECT id, CAST( ST_AsText( geometry ) AS VARCHAR(120) ) POLYGONS
  FROM sample_polys;
```

Results:

```
ID         POLYGONS
------- ------------------------------------------------------------
```

```
   1101 POLYGON (( 110.000000 120.000000, 120.000000 130.000000,
                   110.000000 140.000000, 110.000000 120.000000))
```

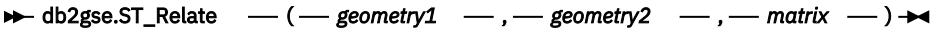
ST_Relate

ST_Relate takes two geometries and a Dimensionally Extended 9 Intersection Model (DE-9IM) matrix as input parameters and returns 1 if the given geometries meet the conditions specified by the matrix. Otherwise, 0 (zero) is returned.

If any of the given geometries is null or empty, then null is returned.

If the second geometry is not represented in the same spatial reference system as the first geometry, the second geometry will be converted to the other spatial reference system.

Syntax



Parameters

geometry1
A value of one of the seven distinct spatial data types that represents the geometry that is tested against *geometry2*.

geometry2
A value of one of the seven distinct spatial data types that represents the geometry that is tested against *geometry1*.

matrix
A value of CHAR(9) that represents the DE-9IM matrix that is to be used for the test of *geometry1* and *geometry2*.

Return type

INTEGER

Example

The following example creates two separate polygons. Then, the ST_Relate function is used to determine several relationships between the two polygons. For example, whether the two polygons overlap.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_polys (id INTEGER, geometry ST_Polygon)

INSERT INTO sample_polys
  VALUES (1,
          ST_Polygon('polygon ( (40 120, 90 120, 90 150, 40 150, 40 120) )', 0))
INSERT INTO sample_polys
  VALUES (2,
          ST_Polygon('polygon ( (30 110, 50 110, 50 130, 30 130, 30 110) )', 0))

SELECT ST_Relate(a.geometry, b.geometry, CHAR('T*T***T**')  "Overlaps  ",
       ST_Relate(a.geometry, b.geometry, CHAR('T*T***FF*')  "Contains  ",
       ST_Relate(a.geometry, b.geometry, CHAR('T*F**F***')  "Within    "
       ST_Relate(a.geometry, b.geometry, CHAR('T********')  "Intersects",
       ST_Relate(a.geometry, b.geometry, CHAR('T*F**FFF2')  "Equals    "
  FROM sample_polys a, sample_polys b
  WHERE a.id = 1 AND b.id = 2
```

Results:

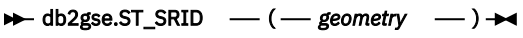
```
Overlaps    Contains    Within      Intersects  Equals
----------- ----------- ----------- ----------- -----------
          1           0           0           1           0
```

ST_SRID

The ST_SRID function takes a geometry as the input parameter and returns the spatial reference system identifier from the geometry.

If the given geometry is null, then null is returned.

Syntax



Parameters

geometry
A value of one of the seven distinct spatial data types that represents the geometry for which the spatial reference system identifier is to be set or returned.

Return types

INTEGER

Example

In the following example, two points are created in two different spatial reference systems. You can use the ST_SRID function to find the ID of the spatial reference system that is associated with each point.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_points (id INTEGER, geometry ST_Point)
INSERT INTO sample_points
  VALUES (1, ST_Point( point (80 180), 0 ) )
INSERT INTO sample_points
  VALUES (2, ST_Point( point (-74.21450127 + 42.03415094), 1 ) )
SELECT id, ST_SRID (geometry) SRID FROM sample_points
```

Results:

```
 ID     SRID
------ ---------
   1    0
   2    1
```

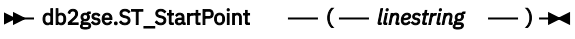
ST_StartPoint

ST_StartPoint takes a linestring as an input parameter and returns the point that is the first point of the linestring. The resulting point is represented in the spatial reference system of the given linestring.

This result is equivalent to the function call ST_PointN(linestring, 1).

If the given curve is null or is empty, then null is returned.

Syntax



Parameters

linestring
A value of type ST_LineString that represents the geometry from which the first point is returned.

Return type

db2gse.ST_Point

Example

In the following example, two linestrings are added to the SAMPLE_LINES table. The first one is a linestring with X and Y coordinates. The second one is a linestring with X, Y, and Z coordinates. The ST_StartPoint function is used to return the first point in each linestring.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_lines (id INTEGER, line ST_LineString)

INSERT INTO sample_lines
  VALUES (1, ST_LineString ('linestring
         (10 10, 5 5, 0 0, 10 0, 5 5, 0 10)', 0))

INSERT INTO sample_lines
  VALUES (1, ST_LineString ('linestring z
         (0 0 4, 5 5 5, 10 10 6, 5 5 7, 0 0 8)', 0))

SELECT id, CAST( ST_AsText( ST_StartPoint( line ) ) AS VARCHAR(80))
  START_POINT
  FROM sample_lines
```

Results:

```
ID          START_POINT
----------- ----------------------------------------
          1 POINT ( 10.00000000 10.00000000)
          2 POINT Z ( 0.00000000 0.00000000 4.00000000)
```

ST_SymDifference

ST_SymDifference takes two geometries as input parameters and returns the geometry that is the symmetrical difference of the two geometries.

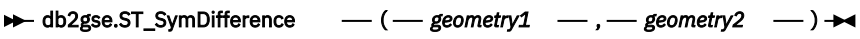
The symmetrical difference is the non-intersecting part of the two given geometries. The resulting geometry is represented in the spatial reference system of the first geometry. The dimension of the returned geometry is the same as that of the input geometries. Both geometries must be of the same dimension.

If the second geometry is not represented in the same spatial reference system as the first geometry, the second geometry is converted to the other spatial reference system.

If the geometries are equal, an empty geometry of type ST_Point is returned. If either geometry is null, then null is returned.

The resulting geometry is represented in the most appropriate spatial type. If it can be represented as a point, linestring, or polygon, then one of those types is used. Otherwise, the multipoint, multilinestring, or multipolygon type is used.

Syntax



Parameters

geometry1
A value of one of the seven distinct spatial data types that represents the first geometry to compute the symmetrical difference with *geometry2*.

geometry2
A value of one of the seven distinct spatial data types that represents the second geometry to compute the symmetrical difference with *geometry1*.

Return type

db2gse.ST_Geometry

Examples

Example 1

This example shows the use of the ST_SymDifference function. The geometries are stored in the SAMPLE_GEOMS table.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_geoms (id INTEGER, geometry ST_Geometry)

INSERT INTO sample_geoms
  VALUES (1,
          ST_Geometry (ST_Polygon('polygon ( (10 10, 10 20, 20 20,
                      20 10, 10 10) )', 0)))

INSERT INTO sample_geoms
  VALUES
    (2, ST_Geometry (ST_Polygon('polygon ( (30 30, 30 50, 50 50,
                     50 30, 30 30) )', 0)))

INSERT INTO sample_geoms
  VALUES
    (3,ST_Geometry (ST_Polygon('polygon ( (40 40, 40 60, 60 60,
                    60 40, 40 40) )', 0)))

INSERT INTO sample_geoms
  VALUES
    (4, ST_Geometry (ST_LineString('linestring (70 70, 80 80)' , 0)) )

INSERT INTO sample_geoms
  VALUES
    (5, ST_Geometry(ST_LineString('linestring(75 75, 90 90)' ,0)));
```

In the following examples, the results have been reformatted for readability. Your results will vary according to your display.

Example 2

This example uses ST_SymDifference to return the symmetric difference of two disjoint polygons in the SAMPLE_GEOMS table.

```
SELECT a.id, b.id,
    CAST (ST_AsText (ST_SymDifference (a.geometry, b.geometry) )
    AS VARCHAR(350) ) SYM_DIFF
  FROM sample_geoms a, sample_geoms b
  WHERE a.id = 1 AND b.id = 2
```

Results:

```
ID    ID    SYM_DIFF
----- ----- ------------------------------------------------------------------
    1     2 MULTIPOLYGON ((( 10.00000000 10.00000000, 20.00000000 10.00000000,
                20.00000000 20.00000000, 10.00000000 20.00000000,
                10.00000000 10.00000000)),
                              (( 30.00000000 30.00000000, 50.00000000 30.00000000,
                50.00000000 50.00000000, 30.00000000 50.00000000,
                30.00000000 30.00000000)))
```

Example 3

This example uses ST_SymDifference to return the symmetric difference of two intersecting polygons in the SAMPLE_GEOMS table.

```
SELECT a.id, b.id,
    CAST (ST_AsText (ST_SymDifference (a.geometry, b.geometry) )
    AS VARCHAR(500) ) SYM_DIFF
  FROM sample_geoms a, sample_geoms b
  WHERE a.id = 2 AND b.id = 3
```

Results:

```
ID  ID  SYM_DIFF
--- --- ------------------------------------------------------------------
  2   3  MULTIPOLYGON ((( 40.00000000 50.00000000, 50.00000000 50.00000000,
           50.00000000 40.00000000, 60.00000000 40.00000000,
           60.00000000 60.00000000, 40.00000000 60.00000000,
           40.00000000 50.00000000)),
                      (( 30.00000000 30.00000000, 50.00000000 30.00000000,
           50.00000000 40.00000000, 40.00000000 40.00000000,
           40.00000000 50.00000000, 30.00000000 50.00000000,
           30.00000000 30.00000000)))
```

Example 4

This example uses ST_SymDifference to return the symmetric difference of two intersecting linestrings in the SAMPLE_GEOMS table.

```
SELECT a.id, b.id,
    CAST (ST_AsText (ST_SymDifference (a.geometry, b.geometry) )
    AS VARCHAR(350) ) SYM_DIFF
  FROM sample_geoms a, sample_geoms b
  WHERE a.id = 4 AND b.id = 5
```

Results:

```
ID   ID  SYM_DIFF
---- --- ------------------------------------------------------------------
   4   5 MULTILINESTRING (( 70.00000000 70.00000000, 75.00000000 75.00000000),
              ( 80.00000000 80.00000000, 90.00000000 90.00000000))
```

ST_Touches

ST_Touches takes two geometries as input parameters and returns 1 if the given geometries spatially touch. Otherwise, 0 (zero) is returned.

Two geometries touch if the interiors of both geometries do not intersect, but the boundary of one of the geometries intersects with either the boundary or the interior of the other geometry.

If the second geometry is not represented in the same spatial reference system as the first geometry, it will be converted to the other spatial reference system.

If both of the given geometries are points or multipoints, or if any of the given geometries is null or empty, then null is returned.

Syntax



Parameters

geometry1
A value of one of the seven distinct spatial data types that represents the geometry that is to be tested to touch *geometry2*.

geometry2
A value of one of the seven distinct spatial data types that represents the geometry that is to be tested to touch *geometry1*.

Return type

INTEGER

Example

Several geometries are added to the SAMPLE_GEOMS table. The ST_Touches function is then used to determine which geometries touch each other.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_geoms (id INTEGER, geometry ST_Geometry);

INSERT INTO sample_geoms
  VALUES (1, ST_Geometry(ST_Polygon ('polygon ( (20 30, 30 30, 30 40, 20 40,
                      20 30) )' , 0)) );

INSERT INTO sample_geoms
   VALUES (2, ST_Geometry(ST_Polygon ('polygon ( (30 30, 30 50, 50 50, 50 30,
              30 30) )' ,0)) );

INSERT INTO sample_geoms
  VALUES (3, ST_Geometry(ST_Polygon ('polygon ( (40 40, 40 60, 60 60, 60 40,
                      40 40) )' , 0)) );

INSERT INTO sample_geoms
  VALUES (4, ST_Geometry(ST_Linestring ('linestring( 60 60, 70 70 )' , 0)) );

INSERT INTO sample_geoms
  VALUES (5, ST_Geometry(ST_Linestring ('linestring( 30 30, 60 60 )' , 0)) );

SELECT a.id, b.id, ST_Touches (a.geometry, b.geometry) TOUCHES
  FROM sample_geoms a, sample_geoms b
  WHERE b.id >= a.id;
```

Results:

```
     ID          ID          TOUCHES
     ----------- ----------- -----------
               1           1           0
               1           2           1
               1           3           0
               1           4           0
               1           5           1
               2           2           0
               2           3           0
               2           4           0
               2           5           0
               3           3           0
               3           4           1
               3           5           0
               4           4           0
               4           5           1
               5           5           0
```

ST_Union

ST_Union takes two geometries as input parameters and returns the geometry that is the union of the given geometries. The resulting geometry is represented in the spatial reference system of the first geometry.

Both geometries must be of the same dimension. If any of the two given geometries is null, null is returned.

If the second geometry is not represented in the same spatial reference system as the first geometry, the second geometry is converted to the other spatial reference system.

The resulting geometry is represented in the most appropriate spatial type. If it can be represented as a point, linestring, or polygon, then one of those types is used. Otherwise, the multipoint, multilinestring, or multipolygon type is used.

Syntax



Parameters

geometry1
A value of one of the seven distinct spatial data types that is combined with *geometry2*.

geometry2
A value of one of the seven distinct spatial data types that is combined with *geometry1*.

Return type

db2gse.ST_Geometry

Examples

Example 1

The following SQL statements create and populate the SAMPLE_GEOMS table.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE sample_geoms (id INTEGER, geometry ST_Geometry)

INSERT INTO sample_geoms
  VALUES (1, ST_Geometry(ST_Polygon ( 'polygon
         ((10 10, 10 20, 20 20, 20 10, 10 10) )', 0)))

INSERT INTO sample_geoms
  VALUES (2, ST_Geometry(ST_Polygon ( 'polygon
         ((30 30, 30 50, 50 50, 50 30, 30 30) )', 0)))

INSERT INTO sample_geoms
  VALUES (3, ST_Geometry(ST_Polygon ( 'polygon
         ((40 40, 40 60, 60 60, 60 40, 40 40) )', 0)))

INSERT INTO sample_geoms
  VALUES (4, ST_Geometry(ST_LineString ('linestring (70 70, 80 80)', 0)))

INSERT INTO sample_geoms
  VALUES (5, ST_Geometry(ST_LineString ('linestring (80 80, 100 70)', 0)))
```

In the following examples, the results have been reformatted for readability. Your results will vary according to your display.

Example 2

This example finds the union of two disjoint polygons.

```
SELECT a.id, b.id, CAST ( ST_AsText( ST_Union( a.geometry, b.geometry) )
    AS VARCHAR (350) ) UNION
  FROM sample_geoms a, sample_geoms b
  WHERE a.id = 1 AND b.id = 2
```

Results:

```
ID    ID    UNION
----- ----- ------------------------------------------------------
```

```
    1     2 MULTIPOLYGON ((( 10.00000000 10.00000000, 20.00000000
                    10.00000000, 20.00000000 20.00000000, 10.00000000
                    20.00000000, 10.00000000 10.00000000))
                             (( 30.00000000 30.00000000, 50.00000000
                    30.00000000,50.00000000 50.00000000, 30.00000000
                    50.00000000,30.00000000 30.00000000)))
```

Example 3

This example finds the union of two intersecting polygons.

```
SELECT a.id, b.id, CAST ( ST_AsText( ST_Union(a.geometry, b.geometry))
    AS VARCHAR (250)) UNION
  FROM sample_geoms a, sample_geoms b
  WHERE a.id = 2 AND b.id = 3
```

Results:

```
ID    ID    UNION
----- ----- ----------------------------------------------------
    2     3 POLYGON (( 30.00000000 30.00000000, 50.00000000
                30.00000000,50.00000000 40.00000000, 60.00000000
                40.00000000,60.00000000 60.00000000, 40.00000000
                60.00000000 40.00000000 50.00000000, 30.00000000
                50.00000000, 30.00000000 30.00000000))
```

Example 4

Find the union of two linestrings.

```
SELECT a.id, b.id, CAST ( ST_AsText( ST_Union( a.geometry, b.geometry) )
    AS VARCHAR (250) ) UNION
  FROM sample_geoms a, sample_geoms b
  WHERE a.id = 4 AND b.id = 5
```

Results:

```
ID    ID    UNION
----- ----- ------------------------------------------------------------
    4     5 MULTILINESTRING (( 70.00000000 70.00000000, 80.00000000 80.00000000),
                ( 80.00000000 80.00000000, 100.00000000 70.00000000))
```

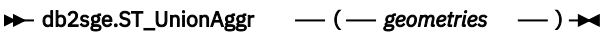
ST_UnionAggr

The ST_UnionAggr function is a union aggregate function that works as a scalar function. This function returns a result for each row.

The result is the union of the geometry on that row and all of the geometries of the previous rows. The result of the final row is the union of all the geometries of that column.

If all of the geometries to be combined in the union are null, then null is returned for each row. If each of the geometries to be combined in the union are either null or are empty, an empty geometry of type ST_Point is returned.

Syntax



Parameters

geometries
A column in a table that has a type of one of the seven distinct spatial data types. The geometries of the column are combined into a union.

Return type

db2gse.ST_Geometry

Examples

In the following examples, the lines of results have been reformatted for readability. The spacing in your results will vary according to your online display.

Example 1: This example illustrates how you can use a union aggregate function to combine a set of points into multipoints. Several points are added to the SAMPLE_POINTS table. The ST_UnionAggr function is used to construct the union of the points for each row.

```
SET CURRENT FUNCTION PATH = CURRENT FUNCTION PATH, db2gse
CREATE TABLE SYSADM.SAMPLE_POINTS (id INTEGER, geometry ST_Point)

INSERT INTO SYSADM.SAMPLE_POINTS
  VALUES (1, ST_Point (2, 3, 1) )
INSERT INTO SYSADM.SAMPLE_POINTS
  VALUES (2, ST_Point (4, 5, 1) )
INSERT INTO SYSADM.SAMPLE_POINTS
  VALUES (3, ST_Point (13, 15, 1) )
INSERT INTO SYSADM.SAMPLE_POINTS
  VALUES (4, ST_Point (12, 5, 1) )
INSERT INTO SYSADM.SAMPLE_POINTS
  VALUES (5, ST_Point (23, 2, 1) )
INSERT INTO SYSADM.SAMPLE_POINTS
  VALUES (6, ST_Point (11, 4, 1) )

SELECT CAST (ST_AsText( ST_UNIONAGGR(geometry) )
       AS VARCHAR(160) ) POINT_AGGREGATE
  FROM SYSADM.SAMPLE_POINTS
```

Results:

```
POINT_AGGREGATE
------------------------------------------------------------
POINT      (2.000000 3.000000)
MULTIPOINT (2.000000 3.000000, 4.000000 5.000000)
MULTIPOINT (2.000000 3.000000, 4.000000 5.000000, 13.000000 15.000000)
MULTIPOINT (2.000000 3.000000, 4.000000 5.000000, 12.000000 5.000000,
           13.000000 15.000000)
MULTIPOINT (2.000000 3.000000, 4.000000 5.000000, 12.000000 5.000000,
           13.000000 15.000000, 23.000000 2.000000)
MULTIPOINT (2.000000 3.000000, 4.000000 5.000000, 11.000000 4.000000,
           12.000000 5.000000, 13.000000 15.000000, 23.000000 2.000000)
```

Example 2: The following examples shows how you can return the last row of the result set from the ST_UnionAggr function.

```
   EXEC SQL BEGIN DECLARE SECTION;

/* User Defined Variables                 */

    struct
         {  short   len;
            char    data(256);

    } HOSTVAR;
    short I_HOSTVAR;
   EXEC SQL END DECLARE SECTION;



/* Declare static scroll cursor*/
```

```
     EXEC SQL DECLARE C1 SENSITIVE STATIC SCROLL
     CURSOR  FOR   SELECT
     CAST(DB2GSE.ST_ASTEXT(DB2GSE.ST_UNIONAGGR(GEOMETRY) ) AS VARCHAR(256) )
                    FROM SYSADM.SYSADM.SAMPLE_POINTS;
.
.
.
/* open cursor*/
     EXEC SQL OPEN C1;

/* Fetch the last row to HOSTVAR */
     EXEC SQL
     FETCH  ABSOLUTE -1   C1
     INTO :HOSTVAR:I_HOSTVAR;

/* Close the cursor*/
    EXEC SQL
         CLOSE C1 ;
```

ST_Within

ST_Within takes two geometries as input parameters and returns 1 if the first geometry is completely within the second geometry. Otherwise, 0 (zero) is returned.

If any of the given geometries is null or is empty, null is returned.

If the second geometry is not represented in the same spatial reference system as the first geometry, it will be converted to the other spatial reference system.

ST_Within performs the same logical operation that ST_Contains performs with the parameters reversed.

Syntax



Parameters

geometry1
A value of one of the seven distinct spatial data types that is to be tested to be fully within *geometry2*.

geometry2
A value of one of the seven distinct spatial data types that is to be tested to be fully within *geometry1*.

Return type

INTEGER

Examples

Example 1

This example illustrates use of the ST_Within function. Geometries are created and inserted into three tables, SAMPLE_POINTS, SAMPLE_LINES, and SAMPLE_POLYGONS.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_points (id INTEGER, geometry ST_Point);
CREATE TABLE sample_lines (id INTEGER, line ST_LineString);
CREATE TABLE sample_polygons (id INTEGER, geometry ST_Polygon);

INSERT INTO sample_points (id, geometry)
  VALUES (1, ST_Point (10, 20, 1) );

INSERT INTO sample_points (id, geometry)
  VALUES (2, ST_Point ('point (41 41)', 1) );

INSERT INTO sample_lines (id, line)
  VALUES (10, ST_LineString ('linestring (1 10, 3 12, 10 10)', 1) );

INSERT INTO sample_lines (id, line)
  VALUES (20, ST_LineString ('linestring (50 10, 50 12, 45 10)', 1) );
```

```
INSERT INTO sample_polygons (id, geometry)
  VALUES (100, ST_Polygon ('polygon (( 0 0, 0 40, 40 40, 40 0, 0 0))', 1) );
```

Example 2

This example finds points from the SAMPLE_POINTS table that are in the polygons in the SAMPLE_POLYGONS table.

```
SELECT a.id POINT_ID_WITHIN_POLYGONS
  FROM sample_points a, sample_polygons b
  WHERE ST_Within( a.geometry, b.geometry) = 1;
```

Results:

```
POINT_ID_WITHIN_POLYGONS
------------------------
1
2
```

Example 3

This example finds linestrings from the SAMPLE_LINES table that are in the polygons in the SAMPLE_POLYGONS table.

```
SELECT a.id LINE_ID_WITHIN_POLYGONS
  FROM sample_lines a, sample_polygons b
  WHERE ST_Within( a.line, b.geometry) = 1;
```

Results:

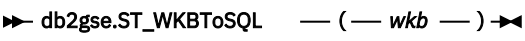
```
LINE_ID_WITHIN_POLYGONS
-----------------------
10
20
```

ST_WKBToSQL

ST_WKBToSQL takes a well-known binary representation of a geometry and returns the corresponding geometry. The spatial reference system with the identifier 0 (zero) is used for the resulting geometry.

If the given well-known binary representation is null, null is returned.

Syntax



Parameter

wkb
A value of type BLOB(4M) that contains the well-known binary representation of the resulting geometry. If the well-known binary representation is null, null is returned.

Return type

db2gse.ST_Geometry

Example

This example illustrates use of the ST_WKBToSQL function. First, geometries are stored in the SAMPLE_GEOMETRIES table in its GEOMETRY column. Then, their well-known binary representations are

stored in the WKB column using the ST_AsBinary function in the UPDATE statement. Finally, the ST_WKBToSQL function is used to return the coordinates of the geometries in the WKB column.

```
SET CURRENT PATH = CURRENT PATH, db2gse
CREATE TABLE sample_geometries
  (id INTEGER, geometry ST_Geometry, wkb BLOB(32K) )

INSERT INTO sample_geometries (id, geometry)
  VALUES (10, ST_Point ( 'point (44 14)', 0 ) ),
         (11, ST_Point ( 'point (24 13)', 0 ) ),
         (12, ST_Polygon ('polygon ((50 20, 50 40, 70 30, 50 20))', 0 ) )
UPDATE sample_geometries AS temp_correlated
  SET wkb = ST_AsBinary(geometry)
  WHERE id = temp_correlated.id
```

Use this SELECT statement to see the geometries in the WKB column.

```
SELECT id, CAST( ST_AsText( ST_WKBToSQL(wkb) ) AS VARCHAR(120) ) GEOMETRIES
  FROM sample_geometries
```

Results:

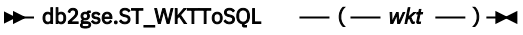
```
ID          GEOMETRIES
----------- ------------------------------------------------------------
         10 POINT ( 44.00000000 14.00000000)
         11 POINT ( 24.00000000 13.00000000)
         12 POLYGON (( 50.00000000 20.00000000, 70.00000000 30.00000000,
                       50.00000000 40.00000000, 50.00000000 20.00000000))
```

ST_WKTToSQL

ST_WKTToSQL takes a well-known text representation of a geometry and returns the corresponding geometry. The spatial reference system with the identifier 0 (zero) is used for the resulting geometry.

If the given well-known text representation is null, null is returned.

Syntax



Parameter

wkt
A value of type CLOB(8M) that contains the well-known text representation of the resulting geometry. If the well-known text representation is null, null is returned.

Return type

db2gse.ST_Geometry

Example

This example illustrates how ST_WKTToSQL can create and insert geometries using their well-known text representations.

```
SET CURRENT PATH = CURRENT PATH, db2gse
CREATE TABLE sample_geometries (id INTEGER, geometry ST_Geometry)

INSERT INTO sample_geometries
  VALUES (10, ST_WKTToSQL( 'point (44 14)' ) ),
         (11, ST_WKTToSQL( 'point (24 13)' ) ),
         (12, ST_WKTToSQL ('polygon ((50 20, 50 40, 70 30, 50 20))' ) )
```

This SELECT statement returns the geometries that have been inserted.

```
SELECT id, CAST( ST_AsText(geometry) AS VARCHAR(120) ) GEOMETRIES
  FROM sample_geometries
```

Results:

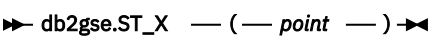
```
ID          GEOMETRIES
----------- ------------------------------------------------------------
         10 POINT ( 44.00000000 14.00000000)
         11 POINT ( 24.00000000 13.00000000)
         12 POLYGON (( 50.00000000 20.00000000, 70.00000000 30.00000000,
                       50.00000000 40.00000000, 50.00000000 20.00000000))
```

ST_X

The function ST_X takes a point as an input parameter and returns its X coordinate.

If the given point is null or is empty, then null is returned.

Syntax



Parameters

point
A value of type ST_Point for which the X coordinate is returned or modified.

Return types

DOUBLE

Examples

Example 1

This example illustrates use of the ST_X function. Geometries are created and inserted into the SAMPLE_POINTS table.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_points (id INTEGER, geometry ST_Point);

INSERT INTO sample_points (id, geometry)
  VALUES (1, ST_Point (2, 3, 32, 5, 1) );

INSERT INTO sample_points (id, geometry)
  VALUES (2, ST_Point (4, 5, 20, 4, 1) );

INSERT INTO sample_points (id, geometry)
  VALUES (3, ST_Point (3, 8, 23, 7, 1) );
```

Example 2

This example finds the X coordinates of the points in the table.

```
SELECT id, ST_X (geometry) X_COORD
  FROM sample_points;
```

Results:

```
ID         X_COORD
---------- ----------------------
         1 +2.00000000000000E+000
         2 +4.00000000000000E+000
```

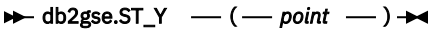
```
         3 +3.00000000000000E+000
```

ST_Y

The ST_Y function takes a point as an input parameter and returns its Y coordinate.

If the given point is null or is empty, then null is returned.

Syntax



Parameters

point
A value of type ST_Point for which the Y coordinate is returned or modified.

Return types

DOUBLE

Examples

Example 1

This example illustrates use of the ST_Y function. Geometries are created and inserted into the SAMPLE_POINTS table.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_points (id INTEGER, geometry ST_Point);

INSERT INTO sample_points (id, geometry)
  VALUES (1, ST_Point (2, 3, 32, 5, 1) );

INSERT INTO sample_points (id, geometry)
  VALUES (2, ST_Point (4, 5, 20, 4, 1) );

INSERT INTO sample_points (id, geometry)
  VALUES (3, ST_Point (3, 8, 23, 7, 1) );
```

Example 2

This example finds the Y coordinates of the points in the table.

```
SELECT id, ST_Y (geometry) Y_COORD
  FROM sample_points;
```

Results:

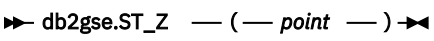
```
ID         Y_COORD
---------- ----------------------
         1 +3.00000000000000E+000
         2 +5.00000000000000E+000
         3 +8.00000000000000E+000
```

ST_Z

The ST_Z function takes a point as an input parameter and returns its Z coordinate.

If the specified point is null or empty, then null is returned.

Syntax



Parameters

point
A value of type ST_Point for which the Z coordinate is returned or modified.

Return types

DOUBLE

Examples

Example 1

This example illustrates use of the ST_Z function. Geometries are created and inserted into the SAMPLE_POINTS table.

```
SET CURRENT PATH = CURRENT PATH, db2gse;
CREATE TABLE sample_points (id INTEGER, geometry ST_Point);

INSERT INTO sample_points (id, geometry)
  VALUES (1, ST_Point (2, 3, 32, 5, 1) );

INSERT INTO sample_points (id, geometry)
  VALUES (2, ST_Point (4, 5, 20, 4, 1) );

INSERT INTO sample_points (id, geometry)
  VALUES (3, ST_Point (3, 8, 23, 7, 1) );
```

Example 2

This example finds the Z coordinates of the points in the table.

```
SELECT id, ST_Z (geometry) Z_COORD
  FROM sample_points;
```

Results:

```
ID         Z_COORD
---------- ----------------------
         1 +3.20000000000000E+001
         2 +2.00000000000000E+001
         3 +2.30000000000000E+001
```

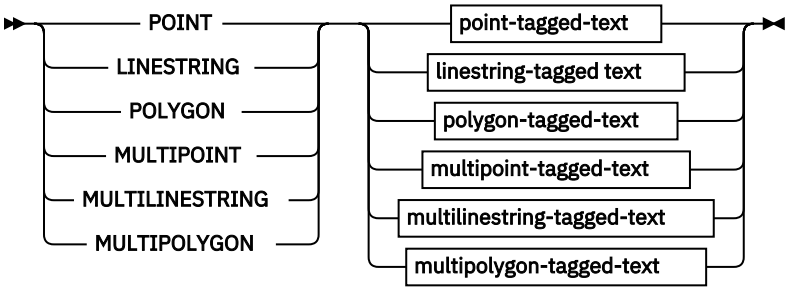
Chapter 13. Supported data formats

IBM Spatial Support for Db2 for z/OS supports several industry standard spatial data formats, such as well-known text representation, well-known binary representation, shape representation, and Geography Markup Language (GML) representation.

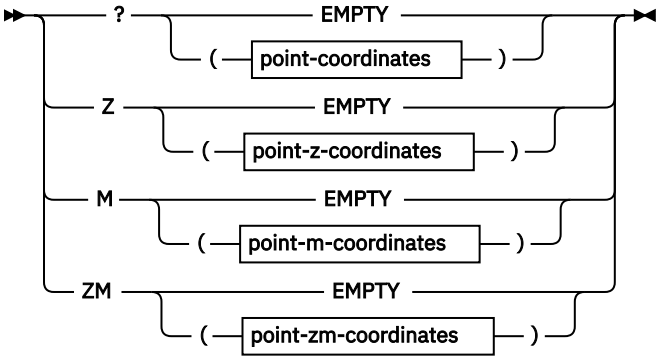
Well-known text (WKT) representation

The OpenGIS Consortium "Simple Features for SQL" specification defines the well-known text representation to exchange geometry data in ASCII format. This representation is also referenced by the ISO "SQL/MM Part: 3 Spatial" standard.

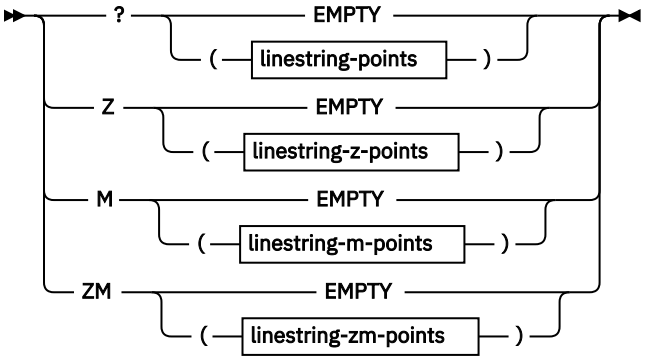
The well-known text representation of a geometry is defined as follows:



point-tagged-text

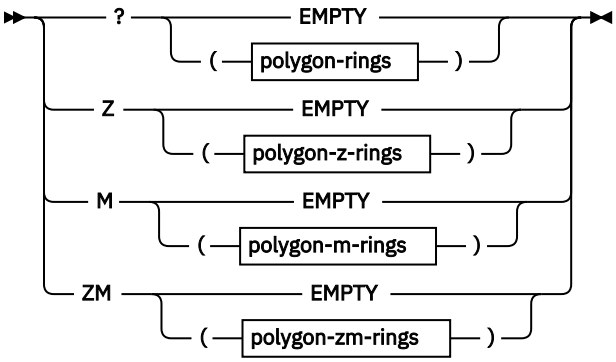


linestring-tagged-text

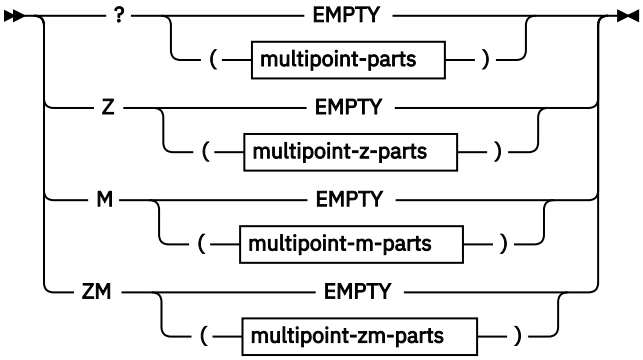


polygon-tagged-text

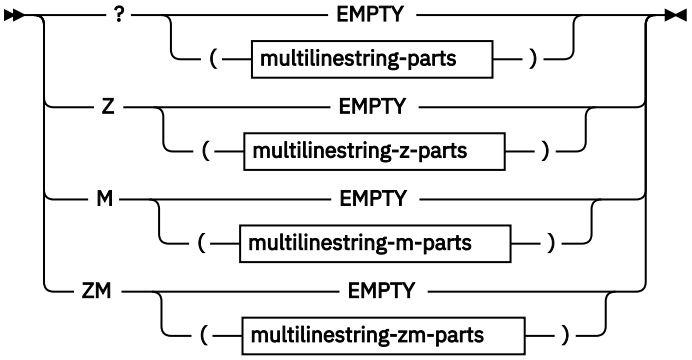
© Copyright IBM Corp. 2007, 2013



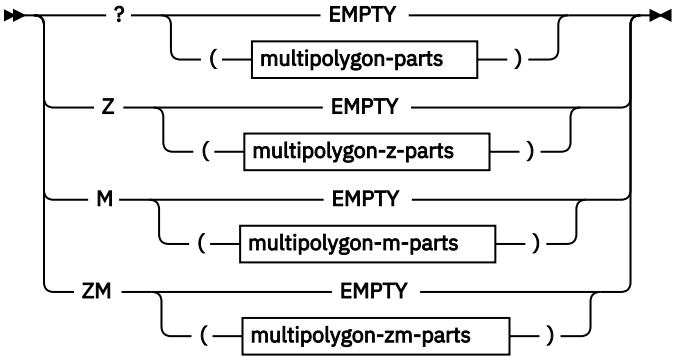
multipoint-tagged-text



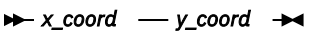
multilinestring-tagged-text



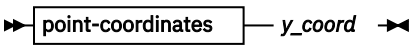
multipolygon-tagged-text



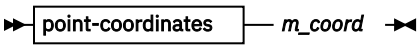
point-coordinates



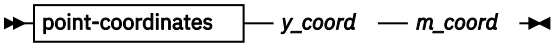
point-z-coordinates



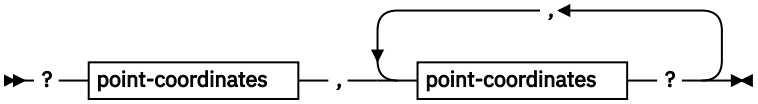
point-m-coordinates



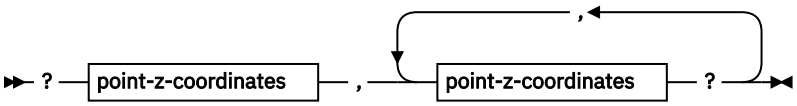
point-zm-coordinates



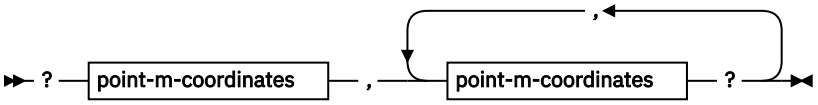
linestring-points



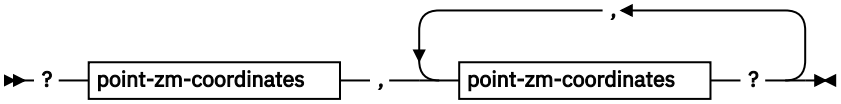
linestring-z-points



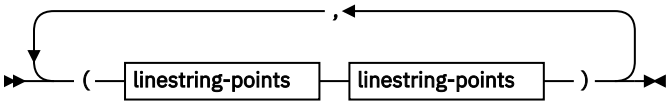
linestring-m-points



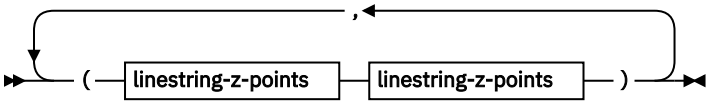
linestring-zm-points



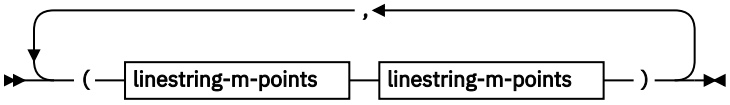
polygon-rings



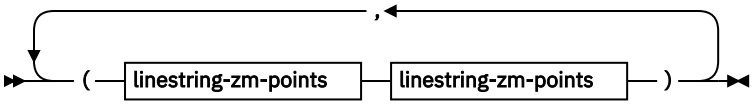
polygon-z-rings



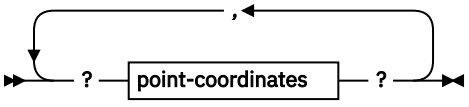
polygon-m-rings



polygon-zm-rings



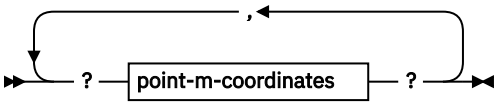
multipoint-parts



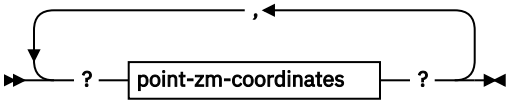
multipoint-z-parts



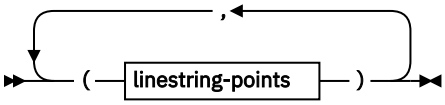
multipoint-m-parts



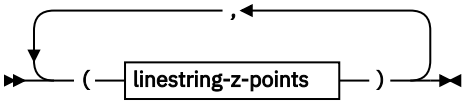
multipoint-zm-parts



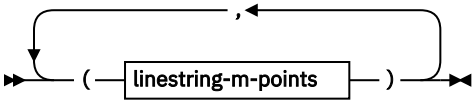
multilinestring-parts



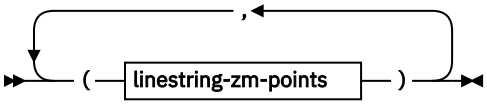
multilinestring-z-parts



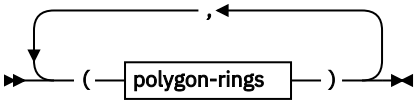
multilinestring-m-parts



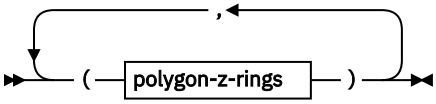
multilinestring-zm-parts



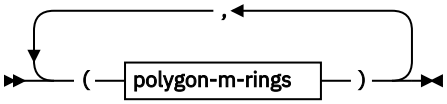
multipolygon-parts



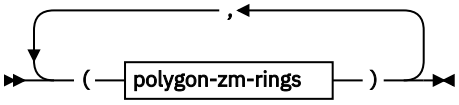
multipolygon-z-parts



multipolygon-m-parts



multipolygon-zm-parts



Parameters

x_coord

A numerical value (fixed, integer, or floating point), which represents the X coordinate of a point.

y_coord

A numerical value (fixed, integer, or floating point), which represents the Y coordinate of a point.

z_coord

A numerical value (fixed, integer, or floating point), which represents the Z coordinate of a point.

m_coord

A numerical value (fixed, integer, or floating point), which represents the M coordinate (measure) of a point.

If the geometry is empty, then the keyword EMPTY is to be specified instead of the coordinate list. The EMPTY keyword must not be embedded within the coordinate list

The following table provides some examples of possible text representations.

Table 29. Geometry types and their text representations

Geometry type	WKT representation	Comment
point	POINT EMPTY	empty point
point	POINT (10.05 10.28)	point
point	POINT Z(10.05 10.28 2.51)	point with Z coordinate
point	POINT M(10.05 10.28 4.72)	point with M coordinate
point	POINT ZM(10.05 10.28 2.51 4.72)	point with Z coordinate and M coordinate
linestring	LINESTRING EMPTY	empty linestring
polygon	POLYGON ((10 10, 10 20, 20 20, 20 15, 10 10))	polygon
multipoint	MULTIPOINT Z(10 10 2, 20 20 3)	multipoint with Z coordinates
multilinestring	MULTILINESTRING M((310 30 1, 40 30 20, 50 20 10)(10 10 0, 20 20 1))	multilinestring with M coordinates
multipolygon	MULTIPOLYGON ZM(((1 1 1 1, 1 2 3 4, 2 2 5 6, 2 1 7 8, 1 1 1 1)))	multipolygon with Z coordinates and M coordinates

Well-known binary (WKB) representation

The OpenGIS Consortium "Simple Features for SQL" specification defines the well-known binary representation for geometries.

This representation is also defined by the International Organization for Standardization (ISO) "SQL/MM Part: 3 Spatial" standard. See the related reference section at the end of this topic for information on functions that accept and produce the WKB.

The basic building block for well-known binary representations is the byte stream for a point, which consists of two double values. The byte streams for other geometries are built using the byte streams for geometries that are already defined.

The following example illustrates the basic building block for well-known binary representations.

```
// Basic Type definitions
```

```
// byte : 1 byte
// uint32 : 32 bit unsigned integer  (4 bytes)
// double : double precision number (8 bytes)

// Building Blocks : Point, LinearRing

Point {
  double x;
  double y;
};
LinearRing   {
  uint32  numPoints;
  Point   points[numPoints];
};
enum wkbGeometryType {
  wkbPoint = 1,
  wkbLineString = 2,
  wkbPolygon = 3,
  wkbMultiPoint = 4,
  wkbMultiLineString = 5,
  wkbMultiPolygon = 6
};
enum wkbByteOrder {
  wkbXDR = 0,     // Big Endian
  wkbNDR = 1     // Little Endian
};
WKBPoint {
  byte     byteOrder;
  uint32   wkbType;     // 1=wkbPoint
  Point    point;
};
WKBLineString {
  byte     byteOrder;
  uint32   wkbType;     // 2=wkbLineString
  uint32   numPoints;
  Point    points[numPoints];
};

WKBPolygon    {
  byte                byteOrder;
  uint32            wkbType;     // 3=wkbPolygon
  uint32            numRings;
  LinearRing        rings[numRings];
};
WKBMultiPoint    {
  byte                byteOrder;
  uint32            wkbType;     // 4=wkbMultipoint
  uint32            num_wkbPoints;
  WKBPoint            WKBPoints[num_wkbPoints];
};
WKBMultiLineString    {
  byte              byteOrder;
  uint32            wkbType;     // 5=wkbMultiLineString
  uint32            num_wkbLineStrings;
  WKBLineString     WKBLineStrings[num_wkbLineStrings];
};

wkbMultiPolygon {
  byte              byteOrder;
  uint32            wkbType;     // 6=wkbMultiPolygon
  uint32            num_wkbPolygons;
  WKBPolygon        wkbPolygons[num_wkbPolygons];
};

WKBGeometry  {
  union {
    WKBPoint                point;
    WKBLineString           linestring;
    WKBPolygon              polygon;
    WKBMultiPoint           mpoint;
    WKBMultiLineString      mlinestring;
    WKBMultiPolygon         mpolygon;
  }
};
```

The following figure shows an example of a geometry in well-known binary representation using NDR coding.

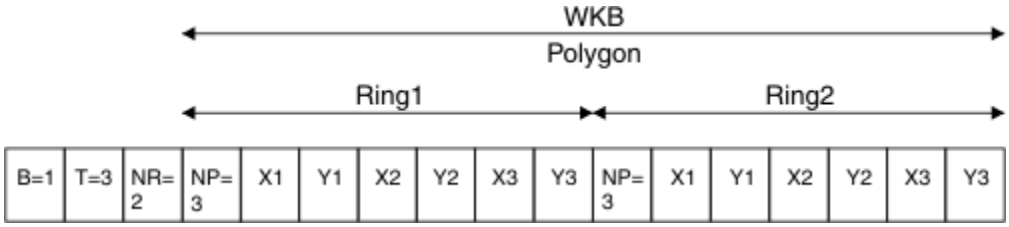


Figure 26. Geometry representation in NDR format

Shape representation

Shape representation is a widely used industry standard defined by ESRI.

For a full description of shape representation, see the ESRI website at http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf.

Geography Markup Language (GML) representation

IBM Spatial Support for Db2 for z/OS has several functions that generate geometries from representations in Geography Markup Language representation.

The Geography Markup Language (GML) is an XML encoding for geographic information that is defined by the OpenGIS Consortium "Geography Markup Language V2" specification. This OpenGIS Consortium specification can be found at http://www.opengis.org/techno/implementation.htm.

Chapter 14. Supported coordinate systems

This information provides an explanation of coordinate systems syntax and lists the coordinate system values that are supported by IBM Spatial Support for Db2 for z/OS.

Coordinate systems syntax

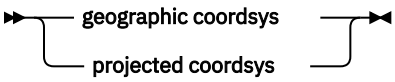
The well-known text (WKT) representation of spatial reference systems provides a standard textual representation for coordinate system information.

The definitions of the well-known text representation are defined by the OpenGIS Consortium "Simple Features for SQL" specification and the ISO "SQL/MM Part 3: Spatial" standard.

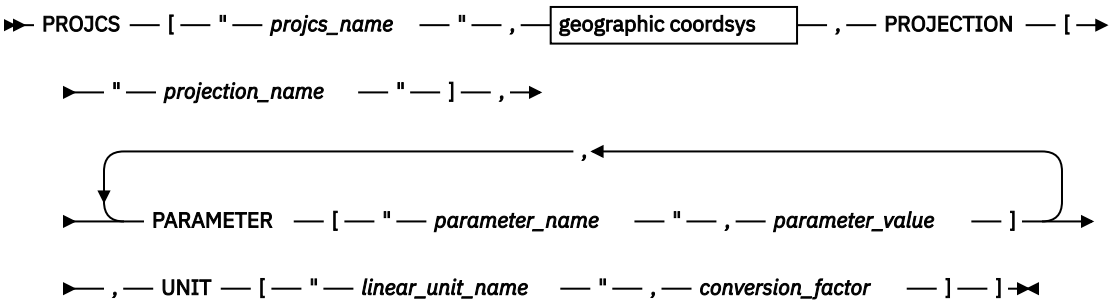
A coordinate system is a geographic (latitude-longitude) coordinate system, or a projected (X,Y) coordinate system. A coordinate system is composed of several objects. Each object has a keyword (for example, DATUM or UNIT) that is followed by a comma-delimited list of the parameters that define the object. The list is enclosed in brackets. Some objects are composed of other objects, so the result is a nested structure.

Note: IBM Spatial Support for Db2 for z/OS also accepts standard brackets ( ) in the place of square brackets [ ]. Keywords are not case-sensitive.

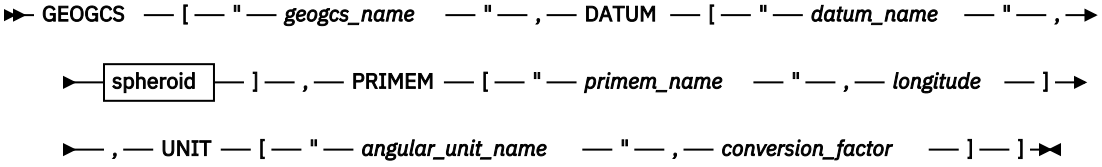
Syntax



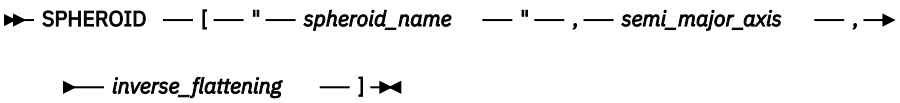
projected coordsys:



geographic coordsys:



spheroid:



Parameters

The type of coordinate system is identified by the first keyword used in the WKT string. This section describes the parameters for each type of coordinate system.

© Copyright IBM Corp. 2007, 2013

PROJCS

A geometry's coordinate system is identified by the PROJCS keyword if the coordinate values are projected.

All projected coordinate systems are based on a geographic coordinate system. The PROJCS keyword is followed by all of the components that define the projected coordinate system. Several objects follow the projected coordinate system name: the geographic coordinate system, the map projection, one or more parameters for the projection, and the linear unit of measure that is applicable to the projected coordinate system.

PROJECTION *"projection_name"*

Specifies the name of the projection algorithm that is to be used for the conversion from the underlying geographic coordinate system to the projected coordinate system. The algorithm for reverse projections is also implied.

For more information about supported projection algorithms, see "Supported map projections" on page 227.

PARAMETER *"parameter_name", parameter_value*

Defines a single parameter for the projected coordinate system. The parameter is identified by its name and a value.

The semantics and units of measure for the parameter and its value are dependent on the parameter itself. For example, a projected coordinate system might require the specification for the longitude of the central meridian and additional offsets and scale factors for the projected coordinate values.

UNIT *"unit_name", conversion_factor*

Defines the linear unit for the projected coordinate system. For example, the unit in which the distance between two coordinate values is measured. The base unit is METER, and the *conversion_factor* specifies how many meters represent a single unit in the projected coordinate system.

For more information about supported units of measure, see "Supported linear units" on page 224.

GEOGCS

A geometry's coordinate system is identified by the GEOGCS keyword if the coordinate values are geographic coordinates.

The name of the geographic coordinate system and several objects are needed to define a geographic coordinate system object: the datum, the prime meridian, and the angular unit of measure that is applicable to the geographic coordinate system.

DATUM *"datum_name"*, spheroid

The datum, based on a spheroid, defines the shape and position of the spheroid that is used to approximate the Earth's surface.

The *datum_name* uniquely identifies the datum. For more information about datums, see "Geographic coordinate system" on page 13.

SPHEROID *"spheroid_name", semi_major_axis, inverse_flattening*

Defines the shape that is used to approximate the Earth's surface. The *semi_major_axis* variable specifies the radius of the spheroid at its equator. This value is measured in meters and must be greater than 0 (zero). The minor axis is calculated based on the major axis and the *inverse_flattening*. The inverse flattening variable *if* gives the proportion of the semi-minor axis *minor* to the semi-major axis *major* and is calculated using the following formula:

if = major / (major - minor)

An inverse flattening value of 0 (zero) is a special case, which implies that the semi-minor axis is equal to the semi-major axis, and therefore, the spheroid is actually a sphere.

PRIMEM *"primem_name", longitude*
Defines the prime meridian, which means the meridian that has a longitude of 0 (zero) assigned to it in the geographic coordinate system. All longitude values are measured relative to that meridian. For example, all points on the prime meridian have a longitude of 0 (zero).

The longitude of the prime meridian is specified relative to the Greenwich meridian and measured in degrees. Greenwich is often chosen as the prime meridian; however, this is not mandatory. A positive value for longitude places the prime meridian to the east of Greenwich, and a negative value for longitude places the prime meridian to the west of Greenwich.

For more information about supported prime meridians, see “Supported prime meridians” on page 227.

UNIT *"angular_unit_name", conversion_factor*
Defines the angular unit for the geographic coordinate system. For example, the unit in which the distance between latitude values or longitude values is measured. The base unit is RADIAN, and the *conversion_factor* specifies how many radians represent a single unit in the geographic coordinate system. The conversion factor must be greater than 0 (zero).

For more information about supported angular units of measure, see “Supported angular units” on page 224.

Examples

The following WKT representation, known as GCS_North_American_1983, shows a geographic coordinate system that uses the spheroid GRS_1980 and the datum D_North_American_1983. The spheroid has a semi-major axis of 6378.137 kilometers and a semi-minor axis of 6356.752 kilometers. That results in an inverse flattening of 298.257222101. The primary meridian is placed at Greenwich (longitude 0), and the units are measured in degrees.

```
GEOGCS["GCS_North_American_1983",
    DATUM["D_North_American_1983",
    SPHEROID["GRS_1980", 6378137, 298.257222101]],
    PRIMEM["Greenwich", 0],
    UNIT["Degree", 0.0174532925199433]]
```

In the following example, UTM zone 10N is a projected coordinate system that is based on the above geographic coordinate system, which used datum NAD83. The Transverse Mercator projection algorithm is used to calculate the projected coordinates from the geographic (latitude-longitude) coordinates for each geometry. The resulting projected coordinate systems are shifted by 50 kilometers to the east, as the parameter named "False_Easting" indicates. Other parameters for the projected coordinate system define the central meridian and a scale factor, for example. All units are measured in meters in the projected coordinate system.

```
PROJCS["NAD_1983_UTM_Zone_10N",
    GEOGCS["GCS_North_American_1983",
        DATUM["D_North_American_1983",
        SPHEROID["GRS_1980", 6378137, 298.257222101]],
        PRIMEM["Greenwich", 0],
        UNIT["Degree", 0.0174532925199433]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["False_Easting", 500000.0],
    PARAMETER["False_Northing", 0.0],
    PARAMETER["Central_Meridian", -123.0],
    PARAMETER["Scale_Factor", 0.9996],
    PARAMETER["Latitude_of_Origin", 0.0],
    UNIT["Meter", 1.0]]
```

Supported linear units

This table shows the supported linear units.

Table 30. Supported linear units

Unit	Conversion factor
Meter	1.0
Foot (International)	0.3048
U.S. Foot	12/39.37
Modified American Foot	12.0004584/39.37
Clarke's Foot	12/39.370432
Indian Foot	12/39.370141
Link	7.92/39.370432
Link (Benoit)	7.92/39.370113
Link (Sears)	7.92/39.370147
Chain (Benoit)	792/39.370113
Chain (Sears)	792/39.370147
Yard (Indian)	36/39.370141
Yard (Sears)	36/39.370147
Fathom	1.8288
Nautical Mile	1852.0

Supported angular units

This table shows the supported angular units.

Table 31. Supported angular units

Unit	Valid range for latitude	Valid range for longitude	Conversion factor
Radian	–pi/2 and pi/2 radians (inclusive)	–pi and pi radians (inclusive)	1.0
Decimal Degree	–90 and 90 degrees (inclusive)	–180 and 180 degrees (inclusive)	pi/180
Decimal Minute	–5400 and 5400 minutes (inclusive)	–10800 and 10800 minutes (inclusive)	(pi/180)/60

Table 31. Supported angular units (continued)

Unit	Valid range for latitude	Valid range for longitude	Conversion factor
Decimal Second	–324000 and 324000 seconds (inclusive)	–648000 and 648000 seconds (inclusive)	(pi/180)*3600
Gon	–100 and 100 gradians (inclusive)	–200 and 200 gradians (inclusive)	pi/200
Grad	–100 and 100 gradians (inclusive)	–200 and 200 gradians (inclusive)	pi/200

Supported spheroids

This table shows the supported spheroids.

Table 32. Supported spheroids

Name	Semi-major axis	Inverse flattening
Airy 1830	6377563.396	299.3249646
Airy Modified 1849	6377340.189	299.3249646
Average Terrestrial System 1977	6378135.0	298.257
Australian National Spheroid	6378160.0	298.25
Bessel 1841	6377397.155	299.1528128
Bessel Modified	6377492.018	299.1528128
Bessel Namibia	6377483.865	299.1528128
Clarke 1858	6378293.639	294.260676369
Clarke 1866	6378206.4	294.9786982
Clarke 1866 (Michigan)	6378450.047	294.978684677
Clarke 1880	6378249.138	293.466307656
Clarke 1880 (Arc)	6378249.145	293.466307656
Clarke 1880 (Benoit)	6378300.79	293.466234571
Clarke 1880 (IGN)	6378249.2	293.46602
Clarke 1880 (RGS)	6378249.145	293.465
Clarke 1880 (SGA 1922)	6378249.2	293.46598
Everest (1830 Definition)	6377299.36	300.8017

Table 32. Supported spheroids (continued)

Name	Semi-major axis	Inverse flattening
Everest 1830 Modified	6377304.063	300.8017
Everest Adjustment 1937	6377276.345	300.8017
Everest 1830 (1962 Definition)	6377301.243	300.8017255
Everest 1830 (1967 Definition)	6377298.556	300.8017
Everest 1830 (1975 Definition)	6377299.151	300.8017255
Everest 1969 Modified	6377295.664	300.8017
Fischer 1960	6378166.0	298.3
Fischer 1968	6378150 .0	298.3
Modified Fischer	6378155 .0	298.3
GEM 10C	6378137.0	298.257222101
GRS 1967	6378160.0	298.247167427
GRS 1967 Truncated	6378160.0	298.25
GRS 1980	6378137.0	298.257222101
Helmert 1906	6378200.0	298.3
Hough 1960	6378270.0	297.0
Indonesian National Spheroid	6378160.0	298.247
International 1924	6378388.0	297.0
International 1967	6378160.0	298.25
Krassowsky 1940	6378245.0	298.3
NWL 9D	6378145.0	298.25
NWL 10D	6378135.0	298.26
OSU 86F	6378136.2	298.25722
OSU 91A	6378136.3	298.25722
Plessis 1817	6376523.0	308.64
Sphere	6371000.0	0.0
Sphere (ArcInfo)	6370997.0	0.0

Table 32. Supported spheroids (continued)

Name	Semi-major axis	Inverse flattening
Struve 1860	6378298.3	294.73
Walbeck	6376896.0	302.78
War Office	6378300.0	296.0
WGS 1966	6378145.0	298.25
WGS 1972	6378135.0	298.26
WGS 1984	6378137.0	298.257223563

Supported prime meridians

This table shows the supported prime meridians.

Table 33. Supported prime meridians

Location	Coordinates
Greenwich	0° 0' 0"
Bern	7° 26' 22.5" E
Bogota	74° 4' 51.3" W
Brussels	4° 22' 4.71" E
Ferro	17° 40' 0" W
Jakarta	106° 48' 27.79" E
Lisbon	9° 7' 54.862" W
Madrid	3° 41' 16.58" W
Paris	2° 20' 14.025" E
Rome	12° 27' 8.4" E
Stockholm	18° 3' 29" E

Supported map projections

These tables show the supported map projections, which include cylindrical projections, conic projections, and the map projection parameters.

Table 34. Cylindrical projections

Cylindrical projections	Pseudocylindrical projections
Behrmann	Craster parabolic

Table 34. Cylindrical projections (continued)

Cylindrical projections	Pseudocylindrical projections
Cassini	Eckert I
Cylindrical equal area	Eckert II
Equirectangular	Eckert III
Gall's stereographic	Eckert IV
Gauss-Kruger	Eckert V
Mercator	Eckert VI
Miller cylindrical	McBryde-Thomas flat polar quartic
Oblique	Mercator (Hotine) Mollweide
Plate-Carée	Robinson
Times	Sinusoidal (Sansom-Flamsteed)
Transverse Mercator	Winkel I

Table 35. Conic projections

Name	Conic projection
Albers conic equal-area	Chamberlin trimetric
Bipolar oblique conformal conic	Two-point equidistant
Bonne	Hammer-Aitoff equal-area
Equidistant conic	Van der Grinten I
Lambert conformal conic	Miscellaneous
Polyconic	Alaska series E
Simple conic	Alaska Grid (Modified-Stereographic by Snyder)

Table 36. Map projection parameters

Parameter	Description
central_meridian	The line of longitude chosen as the origin of x-coordinates.
scale_factor	Scale_factor is used generally to reduce the amount of distortion in a map projection.
standard_parallel_1	A line of latitude that has no distortion generally. Also used for "latitude of true scale."

Table 36. Map projection parameters (continued)

Parameter	Description
standard_parallel_2	A line of longitude that has no distortion generally.
longitude_of_center	The longitude that defines the center point of the map projection.
latitude_of_center	The latitude that defines the center point of the map projection.
longitude_of_origin	The longitude chosen as the origin of x-coordinates.
latitude_of_origin	The latitude chosen as the origin of y-coordinates.
false_easting	A value added to x-coordinates so that all x-coordinate values are positive.
false_northing	A value added to y-coordinates so that all y-coordinates are positive.
azimuth	The angle east of north that defines the center line of an oblique projection.
longitude_of_point_1	The longitude of the first point needed for a map projection.
latitude_of_point_1	The latitude of the first point needed for a map projection.
longitude_of_point_2	The longitude of the second point needed for a map projection.
latitude_of_point_2	The latitude of the second point needed for a map projection.
longitude_of_point_3	The longitude of the third point needed for a map projection.
latitude_of_point_3	The latitude of the third point needed for a map projection.
landsat_number	The number of a Landsat satellite.
path_number	The orbital path number for a particular satellite.
perspective_point_height	The height above the earth of the perspective point of the map projection.
fipszone	State Plane Coordinate System zone number.
zone	UTM zone number.

Chapter 15. The DSN5SCLP program

DSN5SCLP is an ODBC program that you can use to invoke IBM Spatial Support for Db2 for z/OS stored procedures for administrative tasks.

DSN5SCLP is located in the SDSNLOAD library. You can run the DSN5SCLP program using JCL job DSN5SCMD, which is located in the SDSNSAMP library. Before you run DSN5SCMD, follow the customization instructions in the job prolog.

Results are returned in the system log. If the job runs without errors, then a return code of 0 is returned. For example:

```
GSE0000I  The operation was completed successfully.
```

Otherwise, a return code of 8 and a description of the error is returned. For example:

```
GSE1040N  A spatial reference system with the numeric identifier 9999
already exists.
```

A return code of 4 might be returned, indicating that the job completed but with a warning. For example:

```
GSE3010W Invalid ring number 2.
```

If the value of a parameter is broken by a space, surround the value in quotations marks. For example, “COORDSYS 1”.

Use the plus sign (+) to wrap text to another line. For example:

```
-xMin 23 +
-yMin 0
```

If you use the plus sign within quotation marks, all of the white space before the plus sign is preserved. For example:

```
“COORDSYS NUMBER +
ONE” => COORDSYS NUMBER ONE
“COORDSYS NUMBER+
ONE” => COORDSYS NUMBERONE
```

To use quotation marks within quotation marks, use the backslash symbol (\) with the inner quotation marks. For example:

```
“\”Coordsys 1\””
```

To use a left bracket ([ ) and a right bracket ( ]) in the definition of a coordinate system, you must use the correct hex value for each bracket. The hex value for a left bracket is 0xAD. The hex value for a right bracket is 0xBD.

Commands for the DSN5SCLP program

These topics provide descriptions, parameters, and examples for each command that you can use with the DSN5SCLP program.

You can use the following commands with the DSN5SCLP program:

- alter_cs
- alter_srs
- create_cs
- create_idx
- create_srs
- create_srs_2
- disable_spatial

© Copyright IBM Corp. 2007, 2013

- drop_cs
- drop_idx
- drop_srs
- enable_spatial
- function_level
- import_shape
- register_spatial_column
- unregister_spatial_column

You can run all of these commands only in the JCL DSN5SCMD job. The *SUBSYSLOC* value in each command is the Db2 location name.

You can find additional information about the descriptions of parameters in the topics for the corresponding stored procedures. For more information, see Chapter 9, “Stored procedures,” on page 45.

alter_cs

Use the alter_cs command to update a coordinate system definition in the database.

When this command is processed, information about the coordinate system is updated in the DB2GSE.ST_COORDINATE_SYSTEMS catalog view.



Attention: Use care with this command. If you use this command to change the definition of the coordinate system and you have existing spatial data that is associated with a spatial reference system that is based on this coordinate system, you might inadvertently change the spatial data. If spatial data is affected, you are responsible for ensuring that the changed spatial data is still accurate and valid.

Authorization

The user ID under which the command is invoked must have either SYSADM or DBADM authority.

Command syntax

```
DSN5SCLP /alter_cs DALLAS
    -coordsysName cs_name
    [-definition def_string]
    [-organization org_name]
    [-organizationCoordsysId org_cs_id]
    [-description description_string]
```

Parameter descriptions

All parameters are required and case-sensitive unless otherwise indicated.

-coordsysName
Uniquely identifies the coordinate system. You must specify a non-empty value for this parameter.

-definition
Defines the coordinate system. This parameter is optional.

If you do not specify this parameter, the definition of the coordinate system is not changed.

-organization
Identifies the organization that defined the coordinate system and provided the definition for it; for example, "European Petroleum Survey Group (EPSG)." This parameter is optional.

If you do not specify this parameter, the organization of the coordinate system is not changed. If you specify this parameter, then you must also specify the *-organizationCoordsysId* parameter. The combination of the *-organization* and *-organizationCoordsysId* parameters uniquely identifies the coordinate system.

-organizationCoordsysId
Specifies a numeric identifier that is assigned to this coordinate system by the entity listed in the *organization* parameter. This parameter is optional.

If this parameter is not specified, the *-organization* parameter must also be unspecified; in this case, the organization's coordinate system identifier is not changed. If this parameter is specified, the *-organization* parameter must be specified; in this case, the combination of the *-organization* and *-organizationCoordsysId* parameters uniquely identifies the coordinate system.

-description
Describes the coordinate system by explaining its application. This parameter is optional.

If this parameter is not specified, the description information about the coordinate system is not changed.

Example

This example shows the alter_cs command with all of the parameters specified.

```
DSN5SCLP /alter_cs DALLAS +
-coordsysName COORD_SYS1 +
-definition GEOGCS[\"GCS_Swiss_TRF_1996\",DATUM[+
\"D_Swiss_TRF_1995\",SPHEROID[GRS_1980,+
6378137,298.257222101]],PRIMEM[\"Greenwich\",0],+\
UNIT[\"Degree\",0.0174532925199432955]] +
-organization ESRIX -organizationCoordsysId 56029 +
-description Camp_Area_AstroX
```

alter_srs

Use the alter_srs command to update a spatial reference system definition in the database.

When this command is processed, information about the spatial reference system is updated in the DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS catalog view.

Restriction: You cannot alter a spatial reference system if a registered spatial column uses that spatial reference system.



Attention: Use care with this stored procedure. If you use this stored procedure to change offset, scale, or coordinate system name parameters of the spatial reference system, and if you have existing spatial data that is associated with the spatial reference system, you might inadvertently change the spatial data. If spatial data is affected, you are responsible for ensuring that the changed spatial data is still accurate and valid.

Authorization

The user ID under which the command is invoked must have either SYSADM or DBADM authority.

Command syntax

```
DSN5SCLP /alter_srs SUBSYSLOC
    -srsName srs_name
    [-srsId srs_id]
    [-xOffset x_offset]
    [-xScale x_scale]
    [-yOffset y_offset]
    [-yScale y_scale]
    [-zOffset z_offset]
    [-zScale z_scale]
    [-mOffset m_offset]
    [-mScale m_scale]
    [-coordsysName cs_name]
    [-description description_string]
```

Parameter descriptions

All parameters are required and case-sensitive unless otherwise indicated.

-srsName
Identifies the name of the spatial reference system. You must specify a non-empty value for this parameter.

-srsId
Uniquely identifies the spatial reference system. This parameter is optional.

This numeric identifier is used as an input parameter for various spatial functions. If you do not specify this parameter, the numeric identifier of the spatial reference system is not changed.

-xOffset
Specifies the offset for all X coordinates of geometries that are represented in this spatial reference system. This parameter is optional.

If you do not specify this parameter, the value for this parameter in the definition of the spatial reference system is not changed.

The offset is subtracted before the scale factor *-xScale* is applied when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. (*WKT* is well-known text, and *WKB* is well-known binary.)

-xScale
Specifies the scale factor for all X coordinates of geometries that are represented in this spatial reference system. This parameter is optional.

If this parameter is not specified, the value for this parameter in the definition of the spatial reference system is not changed.

The scale factor is applied (multiplication) after the offset *-xOffset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation.

-yOffset
Specifies the offset for all Y coordinates of geometries that are represented in this spatial reference system. This parameter is optional.

If this parameter is not specified, the value for this parameter in the definition of the spatial reference system is not changed.

The offset is subtracted before the scale factor *-yScale* is applied when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation.

-yScale
Specifies the scale factor for all Y coordinates of geometries that are represented in this spatial reference system. This parameter is optional.

If this parameter is not specified, the value for this parameter in the definition of the spatial reference system is not changed.

The scale factor is applied (multiplication) after the offset *-yOffset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. This scale factor must be the same as *-xScale*.

-zOffset
Specifies the offset for all Z coordinates of geometries that are represented in this spatial reference system. This parameter is optional.

If this parameter is not specified, the value for this parameter in the definition of the spatial reference system is not changed.

The offset is subtracted before the scale factor *-zScale* is applied when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation.

-zScale
Specifies the scale factor for all Z coordinates of geometries that are represented in this spatial reference system. This parameter is optional.

If this parameter is not specified, the value for this parameter in the definition of the spatial reference system is not changed.

The scale factor is applied (multiplication) after the offset *-zOffset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation.

-mOffset
Specifies the offset for all M coordinates of geometries that are represented in this spatial reference system. This parameter is optional.

If this parameter is not specified, the value for this parameter in the definition of the spatial reference system is not changed.

The offset is subtracted before the scale factor *-mScale* is applied when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation.

-mScale
Specifies the scale factor for all M coordinates of geometries that are represented in this spatial reference system. This parameter is optional.

If this parameter is not specified, the value for this parameter in the definition of the spatial reference system is not changed.

The scale factor is applied (multiplication) after the offset *-mOffset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation.

-coordsysName
Uniquely identifies the coordinate system on which this spatial reference system is based. This parameter is optional.

If this parameter is not specified, the coordinate system that is used for the spatial reference system is not changed.

The coordinate system must be listed in the view DB2GSE.ST_COORDINATE_SYSTEMS.

-description
Describes the spatial reference system by explaining its application. This parameter is optional.

If this parameter is not specified, the description information about the spatial reference system is not changed.

Example

This example shows the alter_srs command with all of the parameters specified.

```
DSN5SCLP alter_srs SUBSYSLOC +
-srsName NEW_SRS_2006 -srsId 2002 -xOffset -180 +
-xScale 5000000 -yOffset -90 -yScale 5000000 +
-zOffset 0 -zScale 1 -mOffset 0 -mScale 1 +
-coordsysName COORD_SYS1 -description NEW_SRS_2006v2
```

create_cs

Use the create_cs command to store information in the database about a new coordinate system.

When this command is processed, information about the coordinate system is added to the DB2GSE.ST_COORDINATE_SYSTEMS catalog view.

Authorization

The user ID under which the command is invoked must have either SYSADM or DBADM authority.

Command syntax

```
DSN5SCLP /create_cs DALLAS
    -coordsysName cs_name
    -definition define_string
    [-organization organization]
    [-organizationCoordsysId org_cs_id]
    [-description description_string]
```

Parameter descriptions

All parameters are required and case-sensitive unless otherwise indicated.

-coordsys_name
Uniquely identifies the coordinate system. You must specify a non-empty value for this parameter.

-definition
Defines the coordinate system. You must specify a non-empty value for this parameter. The vendor that supplies the coordinate system usually provides the information for this parameter.

You must surround the definition with quotation marks. If you use quotation marks within the definition, you must combine the quotation mark with a backslash. You can use additional symbols for concatenation. For example:

```
-definition "GEOGCS[\"GCS_Swiss_TRF_1995\",+
DATUM[\"D_Swiss_TRF_1995\",SPHEROID[\"GRS_1980\",+
6378137,298.257222101]],PRIMEM[\"Greenwich\",0],+
UNIT[\"Degree\",0.0174532925199432955]] " +
-organization DB2
```

-organization
Identifies the organization that defined the coordinate system and provided the definition for it; for example, "European Petroleum Survey Group (EPSG)." This parameter is optional.

If this parameter is not specified, the *-organizationCoordsysId* parameter must also be unspecified. If this parameter is specified, the *-organizationCoordsysId* parameter must be specified; in this case, the combination of the *-organization* and *-organizationCoordsysId* parameters uniquely identifies the coordinate system.

-organizationCoordsysId
Specifies a numeric identifier. The entity that is specified in the *organization* parameter assigns this value. This value is not necessarily unique across all coordinate systems. This parameter is optional.

If this parameter is not specified, the *-organization* parameter must also be unspecified. If this parameter is specified, the *-organization* parameter must be specified; in this case, the combination of the *-organization* and *-organizationCoordsysId* parameters uniquely identifies the coordinate system.

-description
Describes the coordinate system by explaining its application. This parameter is optional.

Example

This example shows the create_cs command with all of the parameters specified.

```
DSN5SCLP /create_cs DALLAS +
-coordsysName "COORD_SYS2" +
-definition "GEOGCS[\"GCS_Swiss_TRF_1995\",+
DATUM[\"D_Swiss_TRF_1995\",SPHEROID[\"GRS_1980\",+
6378137,298.257222101]],PRIMEM[\"Greenwich\",0],+
UNIT[\"Degree\",0.0174532925199432955]] " +
```

```
-organization DB2 -organizationCoordsysId 5601 +
-description Camp_Area_Astro
```

create_idx

Use the create_idx command to create a spatial grid index on a spatial column to help optimize spatial queries.

The column that you want to index must be a spatial data type that adheres to the following guidelines:

- The column name cannot be qualified
- If the column is not the ST_Point data type, then the LOB table space that stored the corresponding BLOB column data must exist. Also, if the table space that contains the base table is LOG YES, then the LOB table space must be created with LOG YES, too.
- The column cannot have any field procedure or security label defined.
- Only one spatial index is allowed on a column with a spatial data type.

Determining the correct grid size for a spatial grid index takes experience. Set the grid size in relation to the approximate size of the object that you are indexing. A grid size that is too small or too large can decrease performance. For example, a grid size that is set too small can affect the key to object ratio during an index search. If a grid size is set too large, the initial index search returns a small number of candidates and can decrease the performance during the final table scan.

Important: Because a spatial index cannot be rebuilt, create the spatial index with the COPY YES option specified. When you specify this option, Db2 takes an image copy of the index along with an image copy of the table. Also, you cannot alter the spatial index to change any of the options that you specified when you invoked the create_idx command.

Authorization

The user ID under which the command is invoked must have one of the following authorities or privileges:

- SYSADM or DBADM authority on the database that contains the table where the spatial grid index will be used
- Ownership or INDEX privilege on the table

Command syntax

```
DSN5SCLP /create_idx DALLAS
    [-tableSchema tab_schema]
    -tableName tab_name
    -columnName col_name
    [-indexSchema idx_schema]
    -indexName idx_name
    [-otherIdxOpts other_idx_opts]
    -gridSize1 gsize1
    -gridSize2 gsize2
    -gridSize3 gsize3
```

Parameter descriptions

All parameters are required and case-sensitive unless otherwise indicated.

-tableSchema
Identifies the schema to which the table that is specified in the *-tableName* parameter belongs. This parameter is optional.

If you specify this parameter, then a value must be present. If you do not specify this parameter, the CURRENT SCHEMA special register is used as the schema name for the table.

-tableName
Identifies the unqualified name of the table on which the index is to be defined. You must specify a non-empty value for this parameter.

-columnName
Identifies the column that contains the spatial data type for the index. You must specify a non-empty value for this parameter.

-indexSchema
Identifies the schema to which the index that is specified in the *index_name* parameter belongs. This parameter is optional.

If you specify this parameter, then a value must be present. If you do not specify this parameter, the CURRENT SCHEMA special register is used as the schema name for the index.

-indexName
Identifies the name of the index that is to be created. You must specify a non-empty value for this parameter.

-otherIdxOpts
Identifies one or more valid options from the CREATE INDEX statement. This parameter is optional.

For example, you can specify FREEPAGE, PCTFREE, and so on. If you specify a value for this parameter, the value must be non-empty. The following options are not valid for a spatial index:

- CLUSTER
- PARTITIONED
- PARTITION BY
- DEFER YES

The value of this parameter is not case-sensitive.

-gridSize1
A number that indicates the granularity of the smallest index grid. You must specify a non-empty value for this parameter.

-gridSize2
A number that indicates either that there is not a second grid for this index, or the granularity of the second index grid. You must specify a non-empty value for this parameter.

Specify 0, if there is not a second grid. If you want a second grid for the index, then you must specify a grid size that is larger than the value in *-gridSize1*. This value is commonly two to five times larger than the prior grid size.

-gridSize3
A number that indicates either that there is not a third grid for this index, or the granularity of the third index grid. You must specify a non-empty value for this parameter. Specify 0, if there is not a third grid. If you want a third grid for the index, then you must specify a grid size that is larger than the value in *-gridSize2*. This value is commonly two to five times larger than the prior grid size.

Examples

Example 1

This example shows the create_idx command with only the required parameters specified.

```
DSN5SCLP /create_idx DALLAS +
-tableName POINTS -columnName P +
-indexName P_IDX +
-gridSize1 10.0 -gridSize2 20.0 -gridSize3 35.0
```

Example 2

This example shows the create_idx command with all of the parameters specified.

```
DSN5SCLP /create_idx DALLAS +
-tableSchema DB2GSE -tableName POINTS -columnName P +
-indexSchema DB2GSE -indexName P_IDX2 +
```

```
-otherIdxOpts "FREEPAGE 0" +
-gridSize1 10.0 -gridSize2 20.0 -gridSize3 35.0
```

create_srs

Use the create_srs command to create a spatial reference system. This command takes the conversion factors (offsets and scale factors) as input parameters.

A spatial reference system is defined by the coordinate system, the precision, and the extents of coordinates that are represented in this spatial reference system. The extents are the minimum and maximum possible coordinate values for the X, Y, Z, and M coordinates.

This command has two variations. This variation takes the conversion factors (offsets and scale factors) as input parameters. The second variation, the create_srs_2 command, takes the extents and the precision as input parameters and calculates the conversion factors internally.

Authorization

The user ID under which the command is invoked must have the following authorities or privileges:

- SYSADM or DBADM authority
- INSERT and SELECT privileges on the catalog table or view

Command syntax

```
DSN5SCLP /create_srs DALLAS
    -srsName srs_name
    -srsId srs_id
    [-xOffset xoffset]
    -xScale xscale
    [-yOffset xoffset]
    [-yScale xscale]
    [-zOffset xoffset]
    [-zScale xscale]
    [-mOffset xoffset]
    [-mScale xscale]
    -coordsysName cs_name
    [-description description_string]
```

Parameter descriptions

All parameters are required and case-sensitive unless otherwise indicated.

-srsName
Identifies the spatial reference system. You must specify a non-empty value for this parameter.

-srsId
Uniquely identifies the spatial reference system.

This numeric identifier is used as an input parameter for various spatial functions. You must specify a non-empty value for this parameter.

-xOffset
Specifies the offset for all X coordinates of geometries that are represented in this spatial reference system. This parameter is optional.

The offset is subtracted before the scale factor *-xScale* is applied when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. (*WKT* is well-known text, and *WKB* is well-known binary.) If this parameter is not specified, a value of 0 (zero) is used.

-xScale
Specifies the scale factor for all X coordinates of geometries that are represented in this spatial reference system. The scale factor is applied (multiplication) after the offset *-xOffset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial

Support for Db2 for z/OS internal representation. Either the *-xOffset* value is specified explicitly, or a default *-xOffset* value of 0 is used. You must specify a non-empty value for this parameter.

The data type of this parameter is DOUBLE.

-yOffset
Specifies the offset for all Y coordinates of geometries that are represented in this spatial reference system. This parameter is optional.

The offset is subtracted before the scale factor *-yScale* is applied when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. If this parameter is not specified, a value of 0 (zero) is used.

-yScale
Specifies the scale factor for all Y coordinates of geometries that are represented in this spatial reference system. This parameter is optional.

The scale factor is applied (multiplication) after the offset *-yOffset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. Either the *-yOffset* value is specified explicitly, or a default *-yOffset* value of 0 is used. If this parameter is not specified, the value of the *-xScale* parameter is used. If you specify a value for this parameter, the value that you specify must match the value of the *-xScale* parameter.

-zOffset
Specifies the offset for all Z coordinates of geometries that are represented in this spatial reference system. This parameter is optional.

The offset is subtracted before the scale factor *-zScale* is applied when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. If this parameter is not specified, a value of 0 (zero) is used.

-zScale
Specifies the scale factor for all Z coordinates of geometries that are represented in this spatial reference system. This parameter is optional.

The scale factor is applied (multiplication) after the offset *-zOffset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. Either the *-zOffset* value is specified explicitly, or a default *-zOffset* value of 0 is used. If this parameter is not specified, a value of 1 is used.

-mOffset
Specifies the offset for all M coordinates of geometries that are represented in this spatial reference system. This parameter is optional.

The offset is subtracted before the scale factor *-mScale* is applied when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. If this parameter is not specified, a value of 0 (zero) is used.

-mScale
Specifies the scale factor for all M coordinates of geometries that are represented in this spatial reference system. This parameter is optional.

The scale factor is applied (multiplication) after the offset *-mOffset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. Either the *-mOffset* value is specified explicitly, or a default *-mOffset* value of 0 is used. If this parameter is not specified, a value of 1 is used.

-coordsysName
Uniquely identifies the coordinate system on which this spatial reference system is based.

The coordinate system must be listed in the view DB2GSE.ST_COORDINATE_SYSTEMS. You must supply a non-empty value for this parameter.

-description
Describes the spatial reference system by explaining the application's purpose. This parameter is optional.

If this parameter is not specified, no description information is recorded.

Example

This example shows the create_srs command with all of the parameters specified.

```
DSN5SCLP /create_srs DALLAS +
-srsName MAX -coordsysName UNSPECIFIED  -srsId 107 +
-xscale 100 -xoffset 1 -yscale 100 -yoffset 2 -zscale 300 +
-zoffset 400 -mscale 500 -moffset 600 -description ALL_PARMS
```

create_srs_2

Use the create_srs_2 command to create a spatial reference system. This command takes the extents and the precision as input parameters and calculates the conversion factors internally.

A spatial reference system is defined by the coordinate system, the precision, and the extents of coordinates that are represented in this spatial reference system. The extents are the minimum and maximum possible coordinate values for the X, Y, Z, and M coordinates.

This command has two variations. This variation takes the extents and the precision as input parameters and calculates the conversion factors internally. The other variation, the create_srs command, takes the conversion factors (offsets and scale factors) as input parameters.

Authorization

The user ID under which the command is invoked must have the following authorities or privileges:

- SYSADM or DBADM authority
- INSERT and SELECT privileges on the catalog table or view

Command syntax

```
DSN5SCLP /create_srs_2 DALLAS
    -srsName srs_name
    -srsId srs_id
    -xMin x_min
    -xMax x_max
    -xScale x_scale
    -yMin y_min
    -yMax y_max
    [-yScale y_scale]
    -zMin z_min
    -zMax z_max
    [-zScale z_scale]
    -mMin m_min
    -mMax m_max
    [-mScale m_scale]
    -coordsysName cs_name
    [-description description_string]
```

Parameter descriptions

All parameters are required and case-sensitive unless otherwise indicated.

-srsName
Identifies the spatial reference system. You must specify a non-empty value for this parameter.

-srsId
Uniquely identifies the spatial reference system. This numeric identifier is used as an input parameter for various spatial functions. You must specify a non-empty value for this parameter.

The data type of this parameter is INTEGER.

-xMin
Specifies the minimum possible X coordinate value for all geometries that use this spatial reference system. You must specify a non-empty value for this parameter.

-xMax
Specifies the maximum possible X coordinate value for all geometries that use this spatial reference system. You must specify a non-empty value for this parameter.

Depending on the value of *-xScale*, the value that is shown in the view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS might be larger than the value that is specified here. The value from the view is correct.

-xScale
Specifies the scale factor for all X coordinates of geometries that are represented in this spatial reference system.

The scale factor is applied (multiplication) after the offset *-xOffset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. The calculation of the offset *-xOffset* is based on the *-xMin* value. You must supply a non-empty value for this parameter.

-yMin
Specifies the minimum possible Y coordinate value for all geometries that use this spatial reference system. You must supply a non-empty value for this parameter.

-yMax
Specifies the maximum possible Y coordinate value for all geometries that use this spatial reference system. You must supply a non-empty value for this parameter.

Depending on the value of *-yScale*, the value that is shown in the view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS might be larger than the value that is specified here. The value from the view is correct.

-yScale
Specifies the scale factor for all Y coordinates of geometries that are represented in this spatial reference system. This parameter is optional.

The scale factor is applied (multiplication) after the offset *-yOffset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. The calculation of the offset *-yOffset* is based on the *-yMin* value.

If you do not specify this parameter, the value of the *-xScale* parameter is used. If you specify a value for this parameter, the value that you specify must match the value of the *-xScale* parameter.

-zMin
Specifies the minimum possible Z coordinate value for all geometries that use this spatial reference system. You must specify a non-empty value for this parameter.

The data type of this parameter is DOUBLE.

-zMax
Specifies the maximum possible Z coordinate value for all geometries that use this spatial reference system. You must specify a non-empty value for this parameter.

Depending on the value of *-zScale*, the value that is shown in the view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS might be larger than the value that is specified here. The value from the view is correct.

-zScale
Specifies the scale factor for all Z coordinates of geometries that are represented in this spatial reference system. This parameter is optional.

The scale factor is applied (multiplication) after the offset *-zOffset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for

z/OS internal representation. The calculation of the offset *-zOffset* is based on the *-zMin* value. If this parameter is not specified, a value of 1 is used.

-mMin
Specifies the minimum possible M coordinate value for all geometries that use this spatial reference system. You must specify a non-empty value for this parameter.

The data type of this parameter is DOUBLE.

-mMax
Specifies the maximum possible M coordinate value for all geometries that use this spatial reference system. You must specify a non-empty value for this parameter.

Depending on the value of *-mScale*, the value that is shown in the view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS might be larger than the value that is specified here. The value from the view is correct.

-mScale
Specifies the scale factor for all M coordinates of geometries that are represented in this spatial reference system. This parameter is optional.

The scale factor is applied (multiplication) after the offset *-mOffset* is subtracted when geometries are converted from external representations (WKT, WKB, shape) to the IBM Spatial Support for Db2 for z/OS internal representation. The calculation of the offset *-mOffset* is based on the *-mMin* value.

If this parameter is not specified, a value of 1 is used.

-coordsysName
Uniquely identifies the coordinate system on which this spatial reference system is based.

The coordinate system must be listed in the view DB2GSE.ST_COORDINATE_SYSTEMS. You must specify a non-empty value for this parameter.

-description
Describes the spatial reference system by explaining the application's purpose. This parameter is optional.

If this parameter is not specified, no description information is recorded.

Example

This example shows the create_srs_2 command with all of the parameters specified.

```
DSN5SCLP /create_srs_2 DALLAS +
-srsName MAX14 +
-coordsysName UNSPECIFIED  -srsId 10714 -xmin 310 -xmax 310 +
-xscale 310 -ymin 310 -ymax 310 -yscale 310 +
-zmin 10 -zmax 10 -zscale 10  -mmin 10 -mmax 10 -mscale 300 +
-description ALL_PARMS
```

disable_spatial

Use the disable_spatial command to remove all resources that enable IBM Spatial Support for Db2 for z/OS to store spatial data and support operations that are performed on this data.

If you did not define any spatial columns or import any spatial data, you can invoke this command to remove all spatial resources from the subsystem. Because of the interdependency between spatial columns and type definitions, you cannot drop the type definitions if columns of those types exist. If you already defined spatial columns and still want to disable the subsystem, you must specify a value other than 0 (zero) for the *-force* parameter to remove all spatial resources in the subsystem that do not have dependencies on them.

Important: Running this command will disable IBM Spatial Support for Db2 for z/OS for the entire Db2 subsystem.

Authorization

The user ID under which the command is invoked must have SYSADM authority.

Command syntax

```
DSN5SCLP /disable_spatial DALLAS
    [-force force]
```

Parameter descriptions

-force

Specifies that you want to disable a subsystem for spatial operations, even though you might have database objects that are dependent on the spatial types or spatial functions. This parameter is optional.

If you specify a value other than 0 (zero) for the *-force* parameter, the subsystem is disabled and all resources for IBM Spatial Support for Db2 for z/OS are removed (if possible). If you specify 0 (zero), or do not specify this parameter, the subsystem is not disabled if any database objects are dependent on spatial types or spatial functions. Database objects that might have dependencies include tables, views, constraints, triggers, generated columns, methods, functions, procedures, and other data types (subtypes or structured types with a spatial attribute).

Example

Example 1

This example shows the disable_spatial command with the *-force* parameter specified as 0 (zero), which does not disable the subsystem if any database objects are dependent on spatial types or spatial functions.

```
DSN5SCLP /drop_spatial DALLAS +
-force 0
```

Example 2

This example shows the disable_spatial command with the *-force* parameter specified as 1, which disables the subsystem and removes all IBM Spatial Support for Db2 for z/OS resources.

```
DSN5SCLP /drop_spatial DALLAS +
-force 1
```

drop_cs

Use the drop_cs command to delete information about a coordinate system from the database.

When this command is processed, information about the coordinate system is removed from the DB2GSE.ST_COORDINATE_SYSTEMS catalog view.

Restriction: You cannot drop a coordinate system on which a spatial reference system is based.

Authorization

The user ID under which the command is invoked must have either SYSADM or DBADM authority.

Command syntax

```
DSN5SCLP /drop_cs DALLAS
    -coordsysName cs_name
```

Parameter descriptions

All parameters are required and case-sensitive unless otherwise indicated.

-coordsysName
Uniquely identifies the coordinate system. You must specify a non-empty value for this parameter.

Example

This example shows how you can use the drop_cs command to delete information about a coordinate system.

```
DSN5SCLP /drop_cs DALLAS +
-coordsysName "COORD_SYS2"
```

drop_idx

Use the drop_idx command to drop a spatial index.

Authorization

The user ID under which the command is invoked must have one of the following authorities and privileges:

- SYSADM or DBADM authority on the database that contains the table where the spatial grid index is used
- Ownership or INDEX privilege on the table

Command syntax

```
DSN5SCLP /drop_idx DALLAS
    [-indexSchema idx_schema]
    -indexName idx_name
```

Parameter descriptions

All parameters are required and case-sensitive unless otherwise indicated.

-indexSchema
Identifies the schema to which the index that is specified in the *-indexName* parameter belongs. This parameter is optional.

If this parameter is not specified, the value in the CURRENT SCHEMA special register is used as the schema name for the index.

-indexName
Identifies the name of the index that is to be dropped. You must specify a non-empty value for this parameter.

Example

This example shows how you can use the drop_idx command to drop the spatial index named LOCIDX from the DB2GSE schema.

```
DSN5SCLP /drop_idx DALLAS +
-indexSchema DB2GSE -indexName LOCIDX
```

drop_srs

Use the drop_srs command to drop a spatial reference system.

When this command is processed, information about the spatial reference system is removed from the DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS catalog view.

Restriction: You cannot drop a spatial reference system if a spatial column that uses that spatial reference system is registered.

Important: Use care when you use this command. If you use this command to drop a spatial reference system, and if any spatial data is associated with that spatial reference system, you can no longer perform spatial operations on the spatial data.

Authorization

The user ID under which the command is invoked must have either SYSADM or DBADM authority.

Command syntax

```
DSN5SCLP /drop_srs DALLAS
    -srsName srs_name
```

Parameter descriptions

All parameters are required and case-sensitive unless otherwise indicated.

-srsName
Uniquely identifies the name of the index to be dropped. You must specify a non-empty value for this parameter.

Example

This example shows how you can use the drop_srs command to drop the NEWSRS2006B spatial reference system.

```
DSN5SCLP /drop_srs DALLAS +
-srsName NEWSRS2006B
```

enable_spatial

Use the enable_spatial command to supply a Db2 subsystem with the resources that it needs to store spatial data and support spatial operations.

These resources include spatial data types, spatial index types, catalog views, supplied functions, and stored procedures.

Authorization

The user ID under which the command is invoked must have SYSADM authority.

Command syntax

```
DSN5SCLP /enable_spatial DALLAS
    -wlmName wlm_name [-update update_mode]
```

Parameter descriptions

All parameters are case-sensitive unless otherwise indicated.

-wlmName
Identifies the name of the Workload Manager (WLM) in which the stored procedures will run. This parameter is required.

-update
Specifies when you need to install the latest spatial support functions for Db2 for z/OS. Do not use this parameter if you are installing IBM Spatial Support for Db2 for z/OS for the first time. The value of this parameter is `v10`.

Example 1: The following example shows how you can use the enable_spatial command to enable spatial support for the first time.

```
DSN5SCLP /enable_spatial DALLAS +
-wlmName WLMENV3
```

Example 2: The following example shows how you can use the enable_spatial command during migration (if you need to migrate) to enable spatial support and install the latest spatial support functions.

```
DSN5SCLP /enable_spatial DALLAS +
-wlmName WLMENV3 -update v10
```

Related tasks

"Enabling spatial support for the first time" on page 11
If you are a new customer and want to start using IBM Spatial Support for Db2 for z/OS, you need to enable your Db2 subsystem for spatial support.

"Enabling spatial support for migration to Db2 11" on page 12
If you are migrating from Db2 10 to Db2 11 and had spatial support enabled on Db2 10, you must enable your Db2 subsystem for spatial support again.

function_level

Use the function_level command to report the spatial functions that are installed on a Db2 subsystem.

Authorization

The user ID under which the command is invoked must have DBADM authority.

Command syntax

```
DSN5SCLP /function_level SUBSYSTEM -REPORT filename
```

Parameter descriptions

All parameters are required and case-sensitive unless otherwise indicated.

filename
Identifies the name of the file or data set where the report output will be written. You can specify an absolute file name, or 'D'.

If you specify an absolute file name, the report is written to a text file in UNIX System Services (USS). If you specify 'D', the report is written to the data set that is specified for the DD name FUNCLVL.

Examples

Example 1: The following example shows how to specify an absolute file name and have the output of the function_level command written to a text file:

```
DSN5SCLP /function_level   DALLAS -REPORT   /tmp/functionlevel.txt
```

Example 2: The following example shows how to have the output of the function_level command written to the data set that is specified for DD name FUNCLVL. First, you must allocate a sequential, fixed-block 80 record format data set. Then, to the job that invokes the DSN5SCLP program, add the DD name FUNCLVL and specify the data set name.

In this example, assume that you allocated data set 'USER.SPATIAL.REPORT' for DD name FUNCLVL:

```
//FUNCLVL   DD  DSN=USER.SPATIAL.REPORT, DISP=SHR

  Invoke DSN5SCLP with the option REPORT = D

  DSN5SCLP /function_level   DALLAS -REPORT  D
```

The following output shows a sample of the report:

```
STOB00088ZMB:    1
ST_UNIONAGGR7:   1
STCV00055GMV:    1
DB2GSE.DSN5SK48: 0
DB2GSE.DSN5SK30: 0
SDE, TYPE = P:   0
SDE, TYPE = F:   0
Count DB2GSE:    1244
Count SYSPROC:   13

LIST ALL DB2GSE FUNCTIONS NAMESSCHEMAS,SPECIFICNAMES AS BELLOW
NAME                  SCHEMA      SPECIFICNAME
--------------------  ----------  ------------------------------
BLOB                  DB2GSE      BLOB
BLOB                  DB2GSE      BLODRMWKFJFPSIF
...

LIST ALL STORED PROCEDURES NAMESSTART WITH 'ST_'
NAME
--------------------------------------------------
ST_ALTER_COORDSYS
ST_ALTER_SRS
ST_CREATE_COORDSYS
ST_CREATE_INDEX
ST_CREATE_SRS
ST_CREATE_SRS_2
ST_DROP_COORDSYS
ST_DROP_INDEX
ST_DROP_SRS
ST_EXPORT_SHAPE
ST_IMPORT_SHAPE
ST_REGISTER_SPATIAL_COLUMN
ST_UNREGISTER_SPATIAL_COLUMN
```

import_shape

Use the import_shape command to import a shape file to a database that is enabled for spatial operations.

This command can operate in either of two ways, based on the *-createTableFlag* parameter:

- IBM Spatial Support for Db2 for z/OS can create a table that has a spatial column and attribute columns, and it can then load the table's columns with the file's data.
- Otherwise, the shape and attribute data can be loaded into an existing table that has a spatial column and attribute columns that match the file's data.

Important: Using a message file is optional; however, consider specifying a message file so that any errors and informational messages are written to the message file. The import process continues even if an error occurs on a row. If many errors occur, the import process will be much slower.

The input files must reside on the HFS file under the z/OS UNIX environment, so the binder and the user must have read access to the given directory. Also, the message file will be generated on a valid HFS directory under the z/OS UNIX environment if specified. Therefore, the binder and the user must have write access to the given directory.

IBM Spatial Support for Db2 for z/OS does not support the *-inlineLength* parameter and the *-exceptionFile* parameter for this command. If you specify either of these parameters, the parameter will be ignored.

Authorization

The user ID must have the necessary privileges for reading the input files and optionally writing error files. Additional authorization requirements vary based on whether you are importing into an existing table or into a new table.

- When importing shape data to an existing table, your user ID must hold one of the following authorities and privileges:
  - SYSADM or DBADM authority on the database that contains the table or view

  - INSERT and SELECT privilege on the table or view
- When creating a table automatically and importing shape data to the new table, your user ID under must hold the authorizations that are needed for the CREATE TABLE statement.

Command syntax

```
DSN5SCLP /import_shape DALLAS
    -fileName file_name
    [-inputAttrColumns input_columns]
    -srsName srs_name
    [-tableSchema table_schema]
    -tableName table_name
    [-tableAttrColumns attr_columns]
    [-createTableFlag create_flag]
    [-tableCreationParameters tc_params]
    -spatialColumn spatial_column
    [-typeSchema type_schema]
    [-typeName type_name]
    [-inlineLength length]
    [-idColumn id_column]
    [-idColumnIsIdentity id_flag]
    [-restartCount rs_count]
    [-commitScope commit_count]
    [-exceptionFile efile_name]
    [-messagesFile mfile_name]
    [-client client_flag]
```

Parameter descriptions

All parameters are required and case-sensitive unless otherwise indicated.

-fileName
Specifies the full path name of the shape file that is to be imported. You must specify a non-empty value for this parameter.

If you specify the optional file extension, specify either .shp or .SHP. IBM Spatial Support for Db2 for z/OS first looks for an exact match of the specified file name. If IBM Spatial Support for Db2 for z/OS does not find an exact match, it looks first for a file with the .shp extension, and then for a file with the .SHP extension.

See Usage notes for a list of required files, which must reside on the HFS file under the z/OS UNIX environment.

-inputAttrColumns
Specifies a list of attribute columns to import from the dBASE file. This parameter is optional.

If this parameter is not specified, all columns are imported. If the dBASE file does not exist, this parameter must not be specified.

To specify a non-empty value for this parameter, use one of the following specifications:

- **List the attribute column names.** The following example shows how to specify a list of the names of the attribute columns that are to be imported from the dBASE file:

  ```
  N(COLUMN1,COLUMN5,COLUMN3,COLUMN7)
  ```

  If a column name is not enclosed in double quotation marks, it is converted to uppercase. Each name in the list must be separated by a comma. The resulting names must exactly match the column names in the dBASE file.
- **List the attribute column numbers.** The following example shows how to specify a list of the numbers of the attribute columns that are to be imported from the dBASE file:

  ```
  P(1,5,3,7)
  ```

  Columns are numbered beginning with 1. Each number in the list must be separated by a comma.

- **Indicate that no attribute data is to be imported.** Specify "", which is an empty string that explicitly specifies that IBM Spatial Support for Db2 for z/OS is to import *no* attribute data.

This parameter is not case-sensitive.

-srsName
Identifies the spatial reference system that is to be used for the geometries that are imported into the spatial column. You must specify a non-empty value for this parameter.

The spatial column will not be registered. The spatial reference system (SRS) must exist before the data is imported. The import process does not implicitly create the SRS, but it does compare the coordinate system of the SRS with the coordinate system that is specified in the .prj file (if available with the shape file). The import process also verifies that the extents of the data in the shape file can be represented in the given spatial reference system. That is, the import process verifies that the extents lie within the minimum and maximum possible X, Y, Z, and M coordinates of the SRS.

-tableSchema
Identifies the schema to which the table that is specified in the *-tableName* parameter belongs. This parameter is optional.

If this parameter is not specified, the value in the CURRENT SCHEMA special register is used as the schema name for the table.

-tableName
Identifies the unqualified name of the table into which the imported shape file is to be loaded. You must specify a non-empty value for this parameter.

-tableAttrColumns
Specifies the table column names where attribute data from the dBASE file is to be stored. This parameter is optional.

If this parameter is not specified, the names of the columns in the dBASE file are used.

If this parameter is specified, the number of names must match the number of columns that are imported from the dBASE file. If the table exists, the column definitions must match the incoming data. See Usage notes for an explanation of how attribute data types are mapped to Db2 data types.

This parameter is not case-sensitive.

-createTableFlag
Specifies whether the import process is to create a new table. This parameter is optional.

If this parameter is not specified, or is any value other than 0 (zero), a new table is created. (If the table already exists, an error is returned.) If this parameter is 0 (zero), no table is created, and the table must already exist.

If you want to create a target table in a separate table space, first create the table, and then create the LOB table space, auxiliary table, and index for the target table before using the import shape operation.

After creating the required LOB table space, auxiliary table, and index for the target table, specify 0 (zero) for the *-createTableFlag* option to import shape data and attributes data to the table. The import shape operation does not create a LOB table space, an auxiliary table, or an index for the LOB column.

-tableCreationParameters
Specifies any options that are to be added to the CREATE TABLE statement that creates a table into which data is to be imported. This parameter is optional.

If this parameter is not specified, no options are added to the CREATE TABLE statement.

To specify any CREATE TABLE options, use the syntax of the Db2 CREATE TABLE statement. For example, to specify a database and Unicode option for character columns, specify:

```
IN dbName CCSID UNICODE
```

This parameter is not case-sensitive.

-spatialColumn
Identifies the spatial column in the table into which the shape data is to be loaded. You must specify a non-empty value for this parameter.

For a new table, this parameter specifies the name of the new spatial column that is to be created. Otherwise, this parameter specifies the name of an existing spatial column in the table.

-typeSchema
Specifies the schema name of the spatial data type (specified by the *-typeName* parameter) that is to be used when creating a spatial column in a new table. This parameter is optional.

If this parameter is not specified, a value of DB2GSE is used.

-typeName
Identifies the data type that is to be used for the spatial values. This parameter is optional. The valid data types are ST_Point, ST_MultiPoint, ST_MultiLineString, ST_MultiPolygon, or ST_Geometry.

If this parameter is not specified, the data type is determined by the shape file and is one of the following types:

- ST_Point
- ST_MultiPoint
- ST_MultiLineString
- ST_MultiPolygon

Note that shape files, by definition, allow a distinction between only points and multipoints, but not between polygons and multipolygons or between linestrings and multilinestrings.

If you are importing into a table that does not yet exist, this data type is also used for the data type of the spatial column.

This parameter is not case-sensitive.

-inlineLength
This parameter is not supported and always will be null. If you specify this parameter, the parameter is ignored.

-idColumn
Identifies a column that is to be created to contain a unique number for each row of data. The unique values for that column are generated automatically during the import process. This parameter is optional.

If this parameter is not specified, no column is created or populated with unique numbers.

Restriction: You cannot specify an *-idColumn* name that matches the name of any column in the dBASE file.

The requirements and effect of this parameter depend on whether the table already exists.

- **For an existing table**, the data type of the *-idColumn* parameter can be any integer type (INTEGER, SMALLINT, or BIGINT).
- **For a new table that is to be created**, the column is added to the table when the stored procedure creates it. The column will be defined as follows:

```
INTEGER NOT NULL PRIMARY KEY
```

If the value of the *-idColumnIsIdentity* parameter is not null and not 0 (zero), the definition is expanded as follows:

```
INTEGER NOT NULL PRIMARY KEY GENERATED ALWAYS AS IDENTITY
( START WITH 1 INCREMENT BY 1 )
```

-idColumnIsIdentity
Indicates whether the specified *id_column* is to be created using the IDENTITY clause. This parameter is optional.

If this parameter is 0 (zero) or not specified, the column is not created as the identity column. If the parameter is any value other than 0, the column is created as the identity column. This parameter is ignored for tables that already exist.

-restartCount
Specifies that an import operation is to be started at record *n* + 1. The first *n* records are skipped. This parameter is optional.

If this parameter is not specified, all records (starting with record number 1) are imported.

-commitScope
Specifies that a COMMIT is to be performed after at least *n* records are imported. This parameter is optional.

If this parameter is not specified, a value of 0 (zero) is used, and no records are committed.

-exceptionFile
This parameter is not supported and always will be null. If you specify this parameter, the parameter is ignored.

-messagesFile
Specifies the full path name of the file (HFS file under the z/OS UNIX environment) that is to contain messages about the import operation. This parameter is optional.

If the parameter is not specified, no file for IBM Spatial Support for Db2 for z/OS messages is created.

The messages that are written to the messages file can be:

- Informational messages, such as a summary of the import operation
- Error messages for data that could not be imported, for example because of different coordinate systems

The user who runs the job that calls the command must have the necessary privileges on the server to create the file. If the file already exists, the file will be overwritten.

Usage notes

The import_shape command uses from one to four files:

- The main shape file (.shp extension). This file is required.
- The shape index file (.shx extension). This file is optional.
- A dBASE file that contains attribute data (.dbf extension). This file is required only if attribute data is to be imported.
- The projection file that specifies the coordinate system of the shape data (.prj extension). This file is optional. If this file is present, the coordinate system that is defined in it is compared with the coordinate system of the spatial reference system that is specified by the *-srsId* parameter.

The following table describes how dBASE attribute data types are mapped to Db2 data types. All other attribute data types are not supported.

Table 37. Relationship between Db2 data types and dBASE attribute data types

.dbf type	.dbf lengthb (See note)	.dbf decimalsb (See note)	SQL type	Comments
N	< 5	0	SMALLINT	
N	< 10	0	INTEGER	
N	< 20	0	BIGINT	

Table 37. Relationship between Db2 data types and dBASE attribute data types (continued)

.dbf type	.dbf lengthb (See note)	.dbf decimalsb (See note)	SQL type	Comments
N	*len*	*dec*	DECIMAL(*len*,*dec*)	*len*<32
F	*len*	*dec*	REAL	*len* + *dec* < 7
F	*len*	*dec*	DOUBLE	
C	*len*		CHAR(*len*)	
L			CHAR(1)	
D			DATE	

Note: This table includes the following variables, both of which are defined in the header of the dBASE file:

- *len*, which represents the total length of the column in the dBASE file. IBM Spatial Support for Db2 for z/OS uses this value for two purposes:
  - To define the precision for the SQL data type DECIMAL or the length for the SQL data type CHAR
  - To determine which of the integer or floating-point types is to be used
- *dec*, which represents the maximum number of digits to the right of the decimal point of the column in the dBASE file. IBM Spatial Support for Db2 for z/OS uses this value to define the scale for the SQL data type DECIMAL.

For example, assume that the dBASE file contains a column of data whose length (*len*) is defined as 20. Assume that the number of digits to the right of the decimal point (*dec*) is defined as 5. When IBM Spatial Support for Db2 for z/OS imports data from that column, it uses the values of *len* and *dec* to derive the following SQL data type: DECIMAL(20,5).

Example

This example shows how you can use the import_shape command to import a shape file.

```
DSN5SCLP /import_shape DALLAS +
-fileName /tmp/shapes/zipcode.shp +
-createTableFlag 1 +
-inputAttrColumns "N (AREA, ZIPCODE)" +
-srsName SANDIEGO +
-tableSchema NEWTON -tableName TABLE11 +
-tableCreationParameters "IN DATABASE TMP" +
-typeSchema DB2GSE -typeName ST_MULTIPOLYGON +
-spatialColumn "loc" +
-messagesFile /tmp/shapes/ut07_msg
```

Related information

register_spatial_column

Use the register_spatial_column command to register a spatial column and to associate a spatial reference system (SRS) with it.

When this command is processed, information about the spatial column that is being registered is added to the DB2GSE.ST_GEOMETRY_COLUMNS catalog view. Registering a spatial column creates a constraint on the table, if possible, to ensure that all geometries use the specified SRS.

Note: Even if you drop the table that was used to register the spatial column, you must still run the unregister_spatial_column command to remove the entry for the spatial column. Otherwise, if you create the table again and use the register_spatial_column command to register the same spatial column, error GSE1003N will occur. For more information about this message, see "GSE1003N" on page 269.

Authorization

The user ID under which this command is invoked must hold one of the following authorities or privileges:

- SYSADM or DBADM authority on the database that contains the table to which the spatial column that is being registered belongs
- All table or view privileges on this table

Command syntax

```
DSN5SCLP /register_spatial_column DALLAS
    [-tableSchema schema]
    -tableName table_name
    -columnName column_name
    -srsName srs_name
```

Parameter descriptions

All parameters are required and case-sensitive unless otherwise indicated.

-tableSchema
Identifies the schema to which the table or view that is specified in the *-tableName* parameter belongs. This parameter is optional.

If this parameter is not specified, the value in the CURRENT SCHEMA special register is used as the schema name for the table or view.

-tableName
Identifies the unqualified name of the table or view that contains the column that is being registered. You must specify a non-empty value for this parameter.

-columnName
Identifies the column that is being registered. You must specify a non-empty value for this parameter.

-srsName
Identifies the spatial reference system that is to be used for this spatial column. You must specify a non-empty value for this parameter.

Example

This example shows the register_spatial_column command with all of the parameters specified.

```
DSN5SCLP /register_spatial_column DALLAS +
-tableSchema DB2GSE -tableName TABLE2 -columnName SPATIALCOL +
-srsName NAD83_SRS_1
```

unregister_spatial_column

Use the unregister_spatial_column command to remove the registration of a spatial column.

This command removes the registration by:

- Removing association of the spatial reference system with the spatial column. The DB2GSE.ST_GEOMETRY_COLUMNS catalog view continues to contain the spatial column, but the column is no longer associated with any spatial reference system.
- For a base table, dropping the triggers that IBM Spatial Support for Db2 for z/OS placed on this table to ensure that the geometry values in this spatial column are all represented in the same spatial reference system.

Note: If you drop the table that contains the spatial column before unregistering the spatial column, the triggers are still dropped. However, the entry for the spatial column will still exist in the DB2GSE.ST_GEOMETRY_COLUMNS catalog view. You must still run the unregister_spatial_column command to remove the entry for the spatial column.

Authorization

The user ID under which this command is invoked must hold one of the following authorities or privileges:

- SYSADM or DBADM authority
- All table or view privileges on this table

Command syntax

```
DSN5SCLP /unregister_spatial_column DALLAS
    [-tableSchema schema]
    -tableName table_name
    -columnName column_name
```

Parameter descriptions

All parameters are required and case-sensitive unless otherwise indicated.

-tableSchema
Identifies the schema to which the table that is specified in the *-tableName* parameter belongs. This parameter is optional.

If this parameter is not specified, the value in the CURRENT SCHEMA special register is used as the schema name for the table or view.

-tableName
Identifies the unqualified name of the table that contains the column that is specified in the *-columnName* parameter. You must specify a non-empty value for this parameter.

-columnName
Identifies the spatial column that you want to unregister. You must specify a non-empty value for this parameter.

Example

This example shows the unregister_spatial_column command with all of the parameters specified.

```
DSN5SCLP /unregister_spatial_column DALLAS +
-tableSchema DB2GSE -tableName TABLE2 -columnName SPATIALCOL +
```

Chapter 16. Identifying IBM Spatial Support for Db2 for z/OS problems

If you encounter a problem working with IBM Spatial Support for Db2 for z/OS, you need to determine the cause of the problem.

You can troubleshoot problems in the following ways:

- You can use message information to diagnose the problem.
- When working with spatial support stored procedures and functions, Db2 returns information about the success or failure of the stored procedure or function. The information returned will be a message code (in the form of an integer), message text, or both depending on the interface that you use to work with IBM Spatial Support for Db2 for z/OS.
- If you have a recurring and reproducible problem, an IBM customer support representative might ask you to use the Db2 trace facility, as provided by the Db2 instrumentation facility component (ICF), to help them diagnose the problem.

How to interpret spatial support messages

This topic explains how to interpret the IBM Spatial Support for Db2 for z/OS messages.

You can work with spatial support through several different interfaces:

- IBM Spatial Support for Db2 for z/OS stored procedures
- IBM Spatial Support for Db2 for z/OS functions

These interfaces return messages to help you determine whether the spatial operation that you requested completed successfully or resulted in an error.

The following table explains each part of this sample message text:

```
GSE0000I: The operation was completed successfully.
```

Table 38. The parts of the IBM Spatial Support for Db2 for z/OS message text

Message text part	Description
GSE	The message identifier. All IBM Spatial Support for Db2 for z/OS messages begin with the three-letter prefix GSE.
0000	The message number. A four digit number that ranges from 0000 through 9999.
I	The message type. A single letter that indicates the severity of message: **C** Critical error messages **N** Non-critical error messages **W** Warning messages **I** Informational messages
The operation was completed successfully.	The message explanation.

© Copyright IBM Corp. 2007, 2013

The explanation that appears in the message text is the brief explanation. To obtain additional information about the message, which includes the detailed explanation and suggestions to avoid or correct the problem, see Chapter 17, “GSE Messages,” on page 261.

Output parameters for spatial support stored procedures

You can use stored procedure output parameters to diagnose problems when IBM Spatial Support for Db2 for z/OS stored procedures are invoked explicitly in application programs.

You can invoke IBM Spatial Support for Db2 for z/OS stored procedures explicitly in an application program or from a remote DB2 Connect client command line processor. To diagnose stored procedures that are invoked implicitly, use the messages that are returned by IBM Spatial Support for Db2 for z/OS.

IBM Spatial Support for Db2 for z/OS stored procedures have two output parameters: the message code (msg_code) and the message text (msg_text). The parameter values indicate the success or failure of a stored procedure.

msg_code

The msg_code parameter is an integer, which can be positive, negative, or zero (0). Positive numbers are used for warnings, negative numbers are used for errors (both critical and non-critical), and zero (0) is used for informational messages.

The absolute value of the msg_code is included in the msg_text as the message number. For example

- If the msg_code is 0, the message number is 0000.
- If the msg_code is –219 , the message number is 0219. The negative msg_code indicates that the message is a critical or non-critical error.
- If the msg_code is +1036, the message number is 1036. The positive msg_code number indicates that the message is a warning.

The msg_code numbers for IBM Spatial Support for Db2 for z/OS stored procedures are divided into the three categories shown in the following table:

Table 39. Stored procedure message codes

Codes	Category
0000 – 0999	Common messages
1000 – 1999	Administrative messages
2000 – 2999	Import and export messages

msg_text

The msg_text parameter is comprised of the message identifier, the message number, the message type, and the explanation. An example of a stored procedure msg_text value is:

```
GSE0219N   An EXECUTE IMMEDIATE statement
           failed. SQLERROR = "<sql-error>".
```

The explanation that appears in the msg_text parameter is the brief explanation. You can retrieve additional information about the message that includes the detailed explanation and suggestions to avoid or correct the problem.

For a detailed explanation of the parts of the msg_text parameter, and information on how to retrieve additional information about the message, see “How to interpret spatial support messages” on page 257.

Working with stored procedures in applications

When you call a IBM Spatial Support for Db2 for z/OS stored procedure from an application, you will receive the msg_code and msg_text as output parameters. You can:

- Program your application to return the output parameter values to the application user.
- Perform some action based on the type of msg_code value returned.

Messages for spatial support stored procedures

Most of the messages returned through IBM Spatial Support for Db2 for z/OS are for stored procedures.

When a stored procedure is invoked implicitly, you receive message text that indicates the success or failure of the stored procedure.

The message text is comprised of the message identifier, the message number, the message type, and the explanation. For example, if you enable a database by using the DSN5SCLP program and specify the parameters as `enable_spatial TESTDB -wlmName WLMENV1`, the following message text is returned:

```
GSE0000I  The operation was completed successfully.
```

Likewise, the same message is returned if you disable the database (`disable_spatial TESTDB`).

The explanation that appears in the message text is the brief explanation. You can retrieve additional information about the message that includes the detailed explanation and suggestions to avoid or correct the problem. The steps to retrieve this information, and a detailed explanation of how to interpret the parts of the message text, are discussed in a separate topic.

Spatial support function messages

The messages returned by IBM Spatial Support for Db2 for z/OS functions are typically embedded in an SQL message.

The SQLCODE returned in the message indicates if an error occurred with the function or that a warning is associated with the function. For example:

- The SQLCODE –443 (message number SQL0443) indicates that an error occurred with the function.
- The SQLCODE +462 (message number SQL0462) indicates that a warning is associated with the function.

The following table explains the significant parts of this sample message:

```
DB21034E  The command was processed as an SQL statement because it was
not a valid Command Line Processor command.  During SQL processing it
returned: SQL0443N  Routine "DB2GSE.GSEGEOMFROMWKT"
(specific name "GSEGEOMWKT1") has returned an error
SQLSTATE with diagnostic text "GSE3421N  Polygon is not closed.".
SQLSTATE=38SSL
```

Table 40. The significant parts of IBM Spatial Support for Db2 for z/OS function messages

Message part	Description
SQL0443N	The SQLCODE indicates the type of problem.
GSE3421N	The IBM Spatial Support for Db2 for z/OS message number and message type. The message numbers for functions range from GSE3000 to GSE3999. Additionally, common messages can be returned when you work with IBM Spatial Support for Db2 for z/OS functions. The message numbers for common messages range from GSE0001 to GSE0999.
Polygon is not closed	The message explanation.

Table 40. The significant parts of IBM Spatial Support for Db2 for z/OS function messages (continued)

Message part	Description
SQLSTATE=38SSL	An SQLSTATE code that further identifies the error. An SQLSTATE code is returned for each statement or row. • The SQLSTATE codes for spatial support function errors are 38S*xx*, where each *x* is a character letter or number. • The SQLSTATE codes for spatial support function warnings are 01HS*x*, where the *x* is a character letter or number.

An example of an SQL0443 error message

Suppose that you attempt to insert the values for a polygon into the table POLYGON_TABLE, as shown below:

```
INSERT INTO polygon_table ( geometry )
VALUES ( ST_Polygon ( 'polygon (( 0 0, 0 2, 2 2, 1 2)) ') )
```

This results in an error message because you did not provide the end value to close the polygon. The error message returned is:

```
DB21034E  The command was processed as an SQL statement because it was
not a valid Command Line Processor command.  During SQL processing it
returned: SQL0443N  Routine "DB2GSE.GSEGEOMFROMWKT"
(specific name "GSEGEOMWKT1") has returned an error
SQLSTATE with diagnostic text "GSE3421N  Polygon is not closed.".
SQLSTATE=38SSL
```

The SQL message number SQL0443N indicates that an error occurred and the message includes the message text GSE3421N Polygon is not closed.

When you receive this type of message, locate the GSE message number within the Db2 or SQL error message. The message is repeated, along with a detailed explanation and recommended user response.

Chapter 17. GSE Messages

This section contains the messages for IBM Spatial Support for Db2 for z/OS. The messages are listed in numeric sequence.

GSE0001C **An internal error occurred.**

Explanation

IBM Spatial Support for Db2 for z/OS encountered an unexpected internal error.

User response

Repeat the command. If the problem persists, contact IBM Software Support.

msgcode: -1

sqlstate: 38S01

GSE0002C **IBM Spatial Support for Db2 for z/OS could not access its memory pool. Reason code = *reason-code*.**

Explanation

IBM Spatial Support for Db2 for z/OS tried unsuccessfully to access its memory pool.

User response

Note the reason code *reason-code* and contact IBM Software Support.

msgcode: -2

sqlstate: 38S02

GSE0003N **IBM Spatial Support for DB2 for z/OS could not allocate *number* bytes of memory.**

Explanation

Not enough memory was available. Possible reasons are that the supply of memory was too low, or that memory was being used by other applications.

User response

Resolve the memory shortage and repeat the command.

msgcode: -3

sqlstate: 38S03

GSE0004C **An internal parameter error occurred.**

Explanation

IBM Spatial Support for Db2 for z/OS encountered an unexpected error in a parameter passed to an internal function. The operation cannot be completed successfully.

User response

Repeat the command. If the problem persists, contact IBM Software Support.

msgcode: -4

sqlstate: 38S04

GSE0006N **An internal string error occurred.**

Explanation

IBM Spatial Support for Db2 for z/OS encountered an unexpected error in an internal string operation. The operation cannot be completed successfully.

User response

Repeat the command. If the problem persists, contact IBM Software Support.

msgcode: -6

sqlstate: 38S06

GSE0007N **The string *string* is missing either a closing quotation mark or a closing pair of quotation marks.**

Explanation

This string lacks a closing delimiter and therefore is not terminated correctly.

User response

Terminate the string correctly. If it starts with a quotation mark, close it with a quotation mark. If it starts with a pair of quotation marks, close it with a pair of quotation marks.

msgcode: -7

sqlstate: 38S07

GSE0008N **An invalid error code *error-code* was used to raise an error.**

© Copyright IBM Corp. 2007, 2013

Explanation

There was an attempt to raise an error identified by an invalid *error-code*.

User response

Contact IBM Software Support.

msgcode: -8

sqlstate: 38S08

GSE0100N **IBM Spatial Support for DB2 for z/OS could not open a file named *file-name*. Reason code = *reason-code*.**

Explanation

Reasons why a file cannot be opened, preceded by their reason codes, are as follows:

111
Permission is denied.

112
The resource is temporarily unavailable.

117
The file exists.

119
The file is too large.

122
An I/O error occurred.

123
The file specified is a directory.

126
The file name specified is too long.

129
No such file or directory exists.

132
Not enough space is available.

141
The specified file system is read-only.

162
HFS encountered a system error.

All other reason codes indicate an internal error.

User response

Verify the authorization for the file and the directories, then repeat the command.

If a reason code that indicates an internal error was encountered, contact IBM Software Support.

msgcode: -100

sqlstate: 38S10

GSE0101N **An I/O error occurred while a file named *file-name* was being processed. Reason code = *reason-code*.**

Explanation

Reasons why an I/O error can occur during file processing, preceded by their reason codes, are as follows:

111
Permission is denied.

112
The resource is temporarily unavailable.

119
The file is too large.

162
HFS encountered a system error.

All other reason codes indicate an internal error.

User response

Verify that the file exists, that you have the appropriate access to the file, and that the file is not in use by another process.

If a reason code that indicates an internal error was encountered, contact IBM Software Support.

msgcode: -101

sqlstate: 38S11

GSE0102N **IBM Spatial Support for DB2 for z/OS could not close a file named *file-name*. Reason code = *reason-code*.**

Explanation

Reasons why an error can occur during an attempt to close a file, preceded by their reason codes, are as follows:

122
An I/O error occurred.

162
HFS encountered a system error.

All other reason codes indicate an internal error.

User response

Verify that the file system is in fully working condition and that enough disk space is available.

If a reason code that indicates an internal error was encountered, contact IBM Software Support.

msgcode: -102

sqlstate: 38S12

GSE0103N **IBM Spatial Support for DB2 for z/OS could not delete a file named *file-name*. Reason code = *reason-code*.**

Explanation

Reasons why an error can occur during an attempt to delete a file, preceded by their reason codes, are as follows:

111
Permission is denied.

112
The resource is temporarily unavailable.

122
An I/O error occurred.

126
The file name specified is too long.

129
No such file or directory exists.

141
The specified file system is read-only.

162
HFS encountered a system error.

All other reason codes indicate an internal error.

User response

Verify the authorization for the file and the directories, then repeat the command.

If a reason code that indicates an internal error was encountered, contact IBM Software Support.

msgcode: -103

sqlstate: 38S13

GSE0200N **An attempt to connect to the database failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS was not able to connect to the database. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: -200

sqlstate: 38S20

GSE0201W **An attempt to disconnect from the database failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS was not able to disconnect from the database. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: +201

sqlstate: 38S21

GSE0202N **No connection to a database exists.**

Explanation

IBM Spatial Support for Db2 for z/OS cannot connect to a database. The command cannot be executed successfully.

User response

Verify the IBM Spatial Support for Db2 for z/OS and database setup. Make sure that a connection to the database can be established.

msgcode: -202

sqlstate: 38S22

GSE0203W **IBM Spatial Support for DB2 for z/OS was already connected to database *database-name*.**

Explanation

IBM Spatial Support for Db2 for z/OS tried to connect to the database *database-name* but was already connected to it.

User response

Contact IBM Software Support.

msgcode: +203

sqlstate: 38S23

GSE0204N **An attempt to commit a transaction failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not commit the current transaction successfully. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: -204

sqlstate: 38S24

GSE0205W **An attempt to roll back a transaction failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not roll back the current transaction. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: +205

sqlstate: 38S25

GSE0206N **A SELECT statement failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not execute a SELECT statement successfully. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: -206

sqlstate: 38S26

GSE0207N **A VALUES statement failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not execute a VALUES statement successfully. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: -207

sqlstate: 38S27

GSE0208N **A PREPARE statement failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not prepare an SQL statement successfully. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: -208

sqlstate: 38S28

GSE0209N **An attempt to open an SQL cursor failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not open a cursor over a result set successfully. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: -209

sqlstate: 38S29

GSE0210W **An attempt to close an SQL cursor failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not close a cursor over a result set successfully. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: +210

sqlstate: 38S2A

GSE0211N **A fetch from an SQL cursor failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not fetch a result from a cursor successfully. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: -211

sqlstate: 38S2B

GSE0212N **An attempt to drop an object failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not drop the specified database object. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: -212

sqlstate: 38S2C

GSE0214N **An INSERT statement failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not execute an INSERT statement successfully. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: -214

sqlstate: 38S2E

GSE0215N **An UPDATE statement failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not execute an UPDATE statement successfully. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: -215

sqlstate: 38S2F

GSE0216N **A DELETE statement failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not execute a DELETE statement successfully. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: -216

sqlstate: 38S2G

GSE0217N **A LOCK TABLE statement failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not execute a LOCK TABLE statement successfully. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: -217

sqlstate: 38S2H

GSE0218N **A DECLARE GLOBAL TEMPORARY TABLE statement failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not execute a DECLARE GLOBAL TEMPORARY TABLE statement successfully. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: -218

sqlstate: 38S2I

GSE0219N **An EXECUTE IMMEDIATE statement failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not execute an EXECUTE IMMEDIATE statement successfully. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: -219

sqlstate: 38S2J

GSE0220N **The setting of a savepoint failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not set a savepoint successfully. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: -220

sqlstate: 38S2K

GSE0221N **No database name was specified.**

Explanation

IBM Spatial Support for Db2 for z/OS could not connect to a database because the name of the database was not specified.

User response

Specify a database name.

msgcode: -221

sqlstate: 38S2L

GSE0222N **An attempt to retrieve the authorization list from Db2 failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not retrieve the authorization list for the current user. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: -222

sqlstate: 38S2M

GSE0223N **An attempt to quiesce a table space failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not quiesce a table space successfully. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: -223

sqlstate: 38S2N

GSE0224N **An attempt to import data into a table failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not import data into a table successfully. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: -224

sqlstate: 38S2O

GSE0226N **An attempt to create a trigger failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not create a trigger successfully. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: -226

sqlstate: 38S2Q

GSE0227N **An ALTER TABLE statement failed. SQLERROR = *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS could not execute an ALTER TABLE statement successfully. Db2 returned *sql-error*.

User response

Refer to the description of *sql-error*.

msgcode: -227

sqlstate: 38S2R

GSE0228N **An attempt to retrieve the message for error *gse-error* and SQLCODE = *sqlcode* from the IBM Spatial Support for Db2 for z/OS message catalog failed.**

Explanation

IBM Spatial Support for Db2 for z/OS could not retrieve the message for error *gse-error* and SQLCODE = *sqlcode* successfully.

User response

Verify the installation of IBM Spatial Support for Db2 for z/OS. Verify also that the message catalog in the language that you want to use is installed.

msgcode: -228

sqlstate: 38S2S

GSE0230N **The system catalog could not be updated.**

Explanation

IBM Spatial Support for Db2 for z/OS encountered an error while attempting to use the Db2 service to update the system catalog.

User response

Contact IBM Software Support.

msgcode: -230

sqlstate: 38S2U

GSE0231N **A PREPARE statement encountered a warning condition. SQLWARNING = *sql-warning*.**

Explanation

IBM Spatial Support for Db2 for z/OS encountered a warning condition when preparing an SQL statement. Db2 returned *sql-warning*. The PREPARE statement was completed successfully.

User response

Refer to the description of *sql-warning*.

msgcode: -231

sqlstate: 38S2V

GSE0300N **The specified password is too long.**

Explanation

The password used in your attempt to connect to the database is too long.

User response

Verify that the password you specified is correct. If it is the correct password, then shorten it and try the operation again.

msgcode: -300

sqlstate: 38S40

GSE0301N **The specified schema name, *schema-name*, is too long.**

Explanation

The requested operation cannot be completed successfully because the length of the schema name exceeds the limit for schema names in Db2.

User response

Specify a valid, shorter schema name. For more information on the schema name length limit, refer to IBM Spatial Support for Db2 for z/OS User's Guide and Reference and try the operation again.

msgcode: -301

sqlstate: 38S41

GSE0302N **The specified table name, *table-name*, is too long.**

Explanation

The requested operation cannot be completed successfully because the length of the table name exceedslimit for table names in Db2.

User response

Specify a valid, shorter table name.

For more information on the table name length limit, refer to IBM Spatial Support for Db2 for z/OS User's Guide and Reference, and then try the operation again.

msgcode: -302

sqlstate: 38S42

GSE0303N **The specified column name, *column-name*, is too long.**

Explanation

The requested operation cannot be completed successfully because the length of the column name exceeds the limit for column names in Db2.

User response

Specify a valid, shorter column name.

For more information on the column name length limit, refer to IBM Spatial Support for Db2 for z/OS User's Guide and Reference, and then try the operation again.

msgcode: -303

sqlstate: 38S43

GSE0304N **The specified index name, *index-name*, is too long.**

Explanation

The requested operation cannot be completed successfully because the length of the index name exceeds limit for index names in Db2.

User response

Specify a valid, shorter index name. For more information on the index name length limit, refer to IBM Spatial Support for Db2 for z/OS User's Guide and Reference, and then try the operation again.

msgcode: -304

sqlstate: 38S44

GSE0305N **The specified data type name, *type-name*, is too long.**

Explanation

The requested operation cannot be completed successfully because the length of the data type name exceeds limit for data type names in Db2.

User response

Specify a valid, shorter type name. For more information on the data type name length limit, refer to IBM Spatial Support for Db2 for z/OS User's Guide and Reference, and then try the operation again.

msgcode: -305

sqlstate: 38S45

GSE0306N **A complete path that starts with *path* would exceed the acceptable limit of *limit* bytes.**

Explanation

The file in a path that starts with *path* cannot be accessed because the length of the complete path would exceed the limit of *limit* bytes. As a result, the statement that you submitted cannot be executed successfully.

User response

Change the location of the file to be accessed so that it can be found using a shorter path and re-submit the statement that you specified. On UNIX systems, symbolic links can be used to establish a shorter path name.

msgcode: -306

sqlstate: 38S46

GSE0307N **The length of a dynamic SQL statement *statement-length* would exceed the acceptable limit of *limit* bytes.**

Explanation

The statement cannot be constructed because it would be too long.

User response

If the statement is constructed in the context of a stored procedure, verify that the WHERE clause is not too long. If necessary, shorten the WHERE clause and retry the operation. If the problem persists, contact IBM Software Support.

msgcode: -307

sqlstate: 38S47

GSE0308N **The string, *string*, exceeds the limit of *limit* bytes.**

Explanation

The requested operation cannot be completed successfully because the string, *string*, is too long.

User response

Specify a shorter string. If necessary, contact IBM Software Support.

msgcode: -308

sqlstate: 38S48

GSE1000N **IBM Spatial Support for DB2 for z/OS could not perform an operation *operation-name* that was requested under user id *user-id*.**

Explanation

You requested this operation under a user id that does not hold the privilege or authority to perform the operation.

User response

Consult the IBM Spatial Support for Db2 for z/OS User's Guide and Reference to find out what the required authorization for the operation is.

msgcode: -1000

sqlstate: 38S50

GSE1001N **The specified value, *value*, is not valid for the *argument-name* argument.**

Explanation

The value *value* that you entered for argument *argument-name* was incorrect or misspelled.

User response

Consult the IBM Spatial Support for Db2 for z/OS User's Guide and Reference to find out what value or range of values you need to specify.

msgcode: -1001

sqlstate: 38S51

GSE1002N **A required argument, *argument-name*, was not specified.**

Explanation

The requested operation cannot be completed successfully because an argument that it requires was not specified.

User response

Specify argument *argument-name* with the value that you want; then request the operation again.

msgcode: -1002

sqlstate: 38S52

GSE1003N **The spatial column, *schema-name.table-name.column-name*, could not be registered with the spatial reference system *srs-name* because it is already registered with another spatial reference system.**

Explanation

A spatial reference system is already registered with the spatial column. It cannot be registered again unless it is unregistered first.

User response

Either unregister the spatial column and then register it with the spatial reference system you want or do not attempt to register it again.

msgcode: -1003

sqlstate: 38S53

GSE1006N **The spatial column *schema-name.table-name.column-name* is not registered.**

Explanation

This spatial column was not registered with a spatial reference system. Therefore, it cannot be unregistered.

User response

Specify a spatial column that is already registered, or do not attempt to unregister the column.

msgcode: -1006

sqlstate: 38S56

GSE1009N **A table named *schema-name.table-name* does not exist.**

Explanation

The requested operation cannot be completed successfully because the table *schema-name.table-name* does not exist.

User response

Specify a valid table name and retry the operation.

msgcode: -1009

sqlstate: 38S59

GSE1010N **A spatial column named *schema-name.table-name.column-name* does not exist.**

Explanation

The requested operation cannot be completed successfully because *schema-name.table-name.column-name* does not identify an existing column.

User response

Specify a valid spatial column name and retry the operation.

msgcode: -1010

sqlstate: 38S5A

GSE1011N **A data type named *schema-name.type-name* does not exist.**

Explanation

The requested operation cannot be completed successfully because a data type *schema-name.type-name* does not exist.

User response

Specify a valid data type name and retry the operation.

msgcode: -1011

sqlstate: 38S5B

GSE1012N **The database has not been enabled for spatial operations.**

Explanation

The requested operation cannot be completed successfully because the database has not been enabled for spatial operations and, therefore, a IBM Spatial Support for Db2 for z/OS catalog has not been created.

User response

Enable the database for spatial operations.

msgcode: -1012

sqlstate: 38S5C

GSE1013N **The database is already enabled for spatial operations.**

Explanation

The database is already enabled for spatial operations. It cannot be enabled again.

User response

Verify that the database has been enabled as you expected. If necessary, disable the database.

msgcode: -1013

sqlstate: 38S5D

GSE1014N **IBM Spatial Support for DB2 for z/OS was unable to register a column named *schema-name.table-name.column-name* because it is not a spatial column.**

Explanation

Either this column does not have a spatial data type, or it does not belong to a local table.

User response

Define a spatial data type for column *schema-name.table-name.column-name*, or specify a column with a spatial data type as declared type.

msgcode: -1014

sqlstate: 38S5E

GSE1015N **A spatial reference system named *srs-name* does not exist.**

Explanation

The requested operation cannot be completed successfully because a spatial reference system with the name *srs-name* does not exist.

User response

Specify an existing spatial reference system and retry the operation.

msgcode: -1015

sqlstate: 38S5F

GSE1016N **A spatial reference system whose numeric identifier is *srs-id* does not exist.**

Explanation

The requested operation could not be completed successfully because a spatial reference system with the specified numeric identifier *srs-id* does not exist.

User response

Specify an existing spatial reference system identifier and retry the operation.

msgcode: -1016

sqlstate: 38S5G

GSE1017N **A coordinate system named *coordsys-name* already exists.**

Explanation

A coordinate system named *coordsys-name* already exists. Another coordinate system with the same name cannot be created.

User response

Specify a unique name for the new coordinate system.

msgcode: -1017

sqlstate: 38S5H

GSE1018N **A coordinate system named *coordsys-name* does not exist.**

Explanation

The requested operation cannot be completed successfully because a coordinate system with the name *coordsys-name* does not exist.

User response

Specify the name of an existing coordinate system.

msgcode: -1018

sqlstate: 38S5I

GSE1019N **No values of the spatial coordinate system *coordsys-name* are specified.**

Explanation

You attempted to alter the coordinate system *coordsys-name*, but did not specify any new values.

User response

Specify at least one new value for the coordinate system.

msgcode: -1019

sqlstate: 38S5J

GSE1020N **A spatial reference system named *srs-name* already exists.**

Explanation

A spatial reference system named *srs-name* already exists. Another spatial reference system with the same name cannot be created.

User response

Specify a unique name for the spatial reference system to be created and retry the operation.

msgcode: -1020

sqlstate: 38S5K

GSE1021N **A spatial reference system named *srs-name* does not exist.**

Explanation

The requested operation cannot be completed successfully because a spatial reference system with the name *srs-name* does not exist.

User response

Specify a name of an existing spatial reference system and retry the operation.

msgcode: -1021

sqlstate: 38S5L

GSE1022N **A spatial reference system whose numeric identifier is *srs-id* does not exist.**

Explanation

The requested operation cannot be completed successfully because a spatial reference system with the numeric identifier *srs-id* does not exist.

User response

Specify an existing numeric identifier for the spatial reference system.

msgcode: -1022

sqlstate: 38S5M

GSE1023N **A coordinate system whose numeric identifier is *coordsys-id* does not exist.**

Explanation

The requested operation cannot be completed successfully because a coordinate system with the numeric identifier *coordsys-id* does not exist.

User response

Specify an existing numeric identifier for the coordinate system and retry the operation.

msgcode: -1023

sqlstate: 38S5N

GSE1024N **No values of the spatial reference system *srs-name* are specified.**

Explanation

You attempted to alter the spatial reference system *srs-name*, but did not specify any new values.

User response

Specify at least one new value for the spatial reference system and then try the operation again.

msgcode: -1024

sqlstate: 38S5O

GSE1025N **A geocoder whose function name is *schema-name.function-name* could not be found in the database.**

Explanation

The requested operation cannot be completed successfully because IBM Spatial Support for Db2 for z/OS could not locate a function named *schema-name.function-name* for the geocoder.

User response

Specify a geocoder with an existing function name or create the function, then try the operation again.

msgcode: -1025

sqlstate: 38S5P

GSE1034N **The parameters passed to the stored procedure do not include a parameter *parameter-name*.**

Explanation

The SQLDA that was passed to the stored procedure is too small. It does not contain an entry for parameter *parameter-name*.

User response

Correct the parameters that are being passed to the stored procedure.

msgcode: -1034

sqlstate: 38S5Y

GSE1035N **The *parameter-name* parameter, which is being passed to the stored procedure, has an incorrect data type.**

Explanation

The data type for parameter *parameter-name* that is passed to the stored procedure is not correct.

User response

Correct the parameters that are being passed to the stored procedure.

msgcode: -1035

sqlstate: 38S5Z

GSE1037N **The definition of the specified coordinate system named *coordsys-name* is invalid.**

Explanation

A coordinate system named *coordsys-name* cannot be created because the definition given for it is invalid.

User response

Specify a correct definition for the coordinate system.

The function ST_EqualCoordsys can be used to verify the definition by comparing the coordinate system with itself.

msgcode: -1037

sqlstate: 38S61

GSE1038N **The WHERE clause specified for the geocoder named *geocoder-name* is invalid. When IBM Spatial Support for Db2 for z/OS attempted to verify the clause, it encountered SQL error *sql-error*.**

Explanation

The geocoding that you requested cannot be completed successfully because the where clause that determines which rows are to be geocoded is invalid.

User response

Specify a syntactically correct WHERE clause.

msgcode: -1038

sqlstate: 38S62

GSE1039N **A coordinate system identified by the specified identifier *organization-coordsys-id* in combination with the specified organization *organization* already exists.**

Explanation

Your request to create a coordinate system could not be met because the combination of identifiers that you specified for the coordinate system (the name of the organization that defined the system and a number that this organization assigned to it) was not unique. Either these two values must be unique in combination, or they must be null.

User response

Specify a unique set of values for *organization* and *organization-coordsys-id*, or choose null values for both.

msgcode: -1039

sqlstate: 38S63

GSE1040N **A spatial reference system with the numeric identifier *srs-id* already exists.**

Explanation

Your request to create a spatial reference system could not be met because the numeric identifier *srs-id* that you assigned to it already identifies another spatial reference system. A spatial reference system's identifier must be unique.

User response

Specify a unique numeric identifier for the spatial reference system.

msgcode: -1040

sqlstate: 38S64

GSE1041N **A coordinate system with the numeric identifier *coordsys-id* already exists.**

Explanation

Your request to create a coordinate system could not be met because the numeric identifier *coordsys-id* that you assigned to it already identifies another spatial coordinate system. A spatial coordinate system's identifier must be unique.

User response

Specify a unique value *coordsys-id* for the coordinate system.

msgcode: -1041

sqlstate: 38S65

GSE1043N **The specified grid index *schema-name.index-name* already exists.**

Explanation

This index already exists. It must be dropped before an index with the same name can be created.

User response

Specify a name for the index that does not yet exist, or drop the existing index and retry the operation.

msgcode: -1043

sqlstate: 38S67

GSE1044N **The specified coordinate system *coordsys-name* cannot be dropped because an existing spatial reference system is based on this coordinate system.**

Explanation

At least one spatial reference system exists that is based on the specified coordinate system *coordsys-name*. The coordinate system cannot be dropped.

User response

Drop all spatial reference systems that are based on the specified coordinate system. Then try to drop the coordinate system again.

msgcode: -1044

sqlstate: 38S68

GSE1045N **The specified spatial reference system *srs-name* cannot be dropped because a spatial column is registered with this spatial reference system.**

Explanation

At least one spatial column exists that is associated with the specified spatial reference system *srs-name*. The spatial reference system cannot be dropped.

User response

Unregister all spatial columns that are associated with the specified spatial reference system. Then try to drop the spatial reference system again.

msgcode: -1045

sqlstate: 38S69

GSE1048N **The spatial reference system with numeric identifier *srs-id* is a predefined geodetic spatial reference system and cannot be altered.**

Explanation

The spatial reference system was not altered. Spatial reference systems with numeric identifiers in the range 2000000000 to 2000000317 are predefined geodetic spatial reference systems and cannot be altered.

User response

Do not attempt to alter this spatial reference system. If a geodetic spatial reference system with a different definition is needed, you can create a new geodetic spatial reference system with the numeric identifier in the range 2000000318 to 2000001000.

msgcode: -1048

sqlstate: 38SP3

GSE1049N **The spatial reference system with numeric identifier *srs-id* is a predefined geodetic spatial reference system and cannot be dropped.**

Explanation

The spatial reference system was not altered. Spatial reference systems with numeric identifiers in the range 2000000000 to 2000000317 are predefined geodetic spatial reference systems and cannot be dropped.

User response

Do not attempt to drop this spatial reference system. If a geodetic spatial reference system with a different definition is needed, you can create a new geodetic spatial reference system with the numeric identifier in the range 2000000318 to 2000001000.

msgcode: -1049

sqlstate: 38SP4

GSE2100N **The number of attribute columns being imported (*input-columns* columns) does not match the number of attribute columns in the target table (*table-columns* columns).**

Explanation

If you are importing columns that contain attribute data, you have the choice of either specifying or not specifying which attribute columns are being imported and which columns are in the target table. If you specify these values, this error occurs when the specified number of attribute columns being imported differs from the specified number of columns in the target table. If you do not specify these values, this error occurs when the actual number of columns being imported differs from the actual number of attribute columns in the target table.

User response

Make sure that the number of specified or actual attribute columns being imported matches the number of specified or actual columns in the target table.

msgcode: -2100

sqlstate: 38S70

GSE2101N **The data type *schema-name.type-name* to be used during import is unknown to Db2.**

Explanation

The spatial data type *schema-name.type-name* cannot be used during the import of spatial data because it does not exist in the database.

User response

Create the data type in the database or use a data type that exists.

msgcode: -2101

sqlstate: 38S71

GSE2102N **The table specified for import, *schema-name.table-name*, does not exist.**

Explanation

A table named *schema-name.table-name* does not exist in the database. Also, IBM Spatial Support for Db2 for z/OS was not asked to create a table to hold the data that is to be imported. The data was not imported.

User response

If the table is to be created by IBM Spatial Support for Db2 for z/OS, specify the appropriate flag. Otherwise, create the table and retry the operation.

msgcode: -2102

sqlstate: 38S72

GSE2103N **The table specified for import *schema-name.table-name* already exists.**

Explanation

IBM Spatial Support for Db2 for z/OS was asked to create a table named *schema-name.table-name* for the imported data, but a table with that name already exists in the database. No data was imported.

User response

If the table is not to be created by IBM Spatial Support for Db2 for z/OS, do not indicate that the table is to be created. Otherwise, specify the name for a table which does not yet exist in the database.

msgcode: -2103

sqlstate: 38S73

GSE2104N **The column *schema-name.table-name.column-name* to import data into does not exist.**

Explanation

The column into which you want to import data *column-name* does not exist in the table *schema-name.table-name*. No data can be imported into it.

User response

Correct the column name or create the column in the table that is to be imported, or correct the table name.

msgcode: -2104

sqlstate: 38S74

GSE2105W **The import operation completed successfully but not all records from the file were imported.**

Explanation

The import operation completed successfully but not all records from the file were imported. The exception file contains the records that could not be imported, and the messages file the contains information why those records were not imported.

User response

Consult the messages file for the reason why not all records were imported, correct the problem and repeat the operation with the original file or the exception file.

msgcode: +2105

sqlstate: 38S75

GSE2106N **The data type of the column *schema-name.table-name.column-name* is *column-type*, which does not match the expected type *expected-type* for the data to be imported from file.**

Explanation

The column *column-name* in the table *schema-name.table-name* to import data into has a declared type *column-type*. *column-type* does not match the type name *expected-type* for the data to be imported from the file. No data can be imported.

User response

Verify the definition of the table with the structure of the file to be imported.

msgcode: -2106

sqlstate: 38S76

GSE2107N **The table to import data into could not be created due to error *sql-error*.**

Explanation

IBM Spatial Support for Db2 for z/OS was asked to create a table to import data into, but the table could not be created successfully. Db2 returned *sql-error*.

User response

Refer to the description of this *sql-error*.

msgcode: -2107

sqlstate: 38S77

GSE2108N **The method specification *method* to identify the attribute columns to be imported from the file is not correct.**

Explanation

Either no method specification was given or *method* is not a valid method specification. Only 'N' and 'P' are supported method specifications for importing spatial data from a file.

User response

Correct the method specification and try the method again.

msgcode: -2108

sqlstate: 38S78

GSE2109N **A character *found-char* was found when a character *expected-char* was expected.**

Explanation

An unexpected character *found-char* was found in the string that identifies the attribute columns to be imported from the file but *expected-char* was expected. The statement cannot be processed successfully.

User response

Correct the string that identifies the attribute columns to be imported from the file.

msgcode: -2109

sqlstate: 38S79

GSE2110N **The column position identifier *position* in the string *string* is invalid.**

Explanation

The column position identifier *position* specified in the string starting with *string* is not in the valid range. Only values greater than 0 (zero) and less than or equal to the number of columns in the file to be imported can be specified. The statement cannot be processed successfully.

User response

Correct the column position identifier.

msgcode: -2110

sqlstate: 38S7A

GSE2111N **A column named *dbf-column-name* in the dBASE file is too long.**

Explanation

The name of column *dbf-column-name* in the dBASE file (.dbf) exceeds the limit for column names in Db2.

User response

Specify a *dbf-column-name* that does not exceed the Db2 length limit.

msgcode: -2111

sqlstate: 38S7B

GSE2112N **The column *dbf-column-name* cannot be found in the dBASE file.**

Explanation

The name *dbf-column-name* does not identify an existing attribute column in the dBASE file (.dbf). The operation cannot be completed successfully.

User response

Specify a column name that exists in the dBASE file.

msgcode: -2112

sqlstate: 38S7C

GSE2113N **The dBASE file data type *dbf-data-type* for the column *dbf-column-name* in the dBASE file is not supported.**

Explanation

The dBASE file data type *dbf-data-type* for the attribute column *dbf-column-name* in the dBASE file (.dbf) cannot be mapped to a data type in the Db2 database. The shape file cannot be imported.

User response

Exclude the column from the column list.

msgcode: -2113

sqlstate: 38S7D

GSE2114N **The column position *position* is out of range. The dBASE file contains *dbf-column-number* columns.**

Explanation

The specified column position *position* must be a value within the valid range. A valid value must greater than 0 (zero) and less than or equal to the *dbf-column-number*.

User response

Specify a valid position.

msgcode: -2114

sqlstate: 38S7E

GSE2115N **A spatial reference system whose numeric identifier is *srs-id* does not exist.**

Explanation

A spatial reference system whose numeric identifier is *srs-id* does not exist. The data cannot be imported.

User response

Either specify an existing spatial reference system, or create the spatial reference system before attempting the import operation.

msgcode: -2115

sqlstate: 38S7F

GSE2116N **The coordinate system definition *coordsys-def* is too long.**

Explanation

The coordinate system definition *coordsys-def* used for the spatial data to be imported is too long. It could not be verified with the coordinate system that underlies the spatial reference system that is to be used for the imported data.

User response

Verify that the coordinate system defined in the projection file (.prj) is correct. To skip the verification step, do not supply the projection file.

msgcode: -2116

sqlstate: 38S7G

GSE2117N **The coordinate system definition *coordsys-def* does not match the coordinate system definition on which the spatial reference system *srs-id* is based.**

Explanation

The coordinate system *coordsys-def* does not match the coordinate system on which the spatial reference system *srs-id* is based. Both coordinate systems must be semantically identical.

User response

Verify that the coordinate system defined in the projection file (.prj) matches the coordinate system of the spatial reference system. To skip the verification step, do not supply the projection file.

msgcode: -2117

sqlstate: 38S7H

GSE2118N **The spatial data does not fit into the spatial reference system with the numeric identifier *srs-id*.**

Explanation

The spatial data covers an area that exceeds the minimum and maximum coordinates of the spatial reference system with the numeric identifier *srs-id*.

User response

Specify a spatial reference system which may fully contain the spatial data to be imported. Refer to the DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS catalog view for the minimum and maximum coordinate values applicable for the spatial reference system.

msgcode: -2118

sqlstate: 38S7I

GSE2119N **The imported data's spatial reference system, whose numerical identifier is *srs-id1*, does not match the target column's spatial reference system, whose numeric identifier is *srs-id2*. The target column's name is *schema-name.table-name.column-name*.**

Explanation

The spatial column *schema-name.table-name.column-name* was registered with a spatial reference system *srs-id2* assigned to it. This spatial reference system does not match the spatial reference system *srs-id1*, which is used for the spatial data to be imported into that column. The data cannot be imported.

User response

Either unregister the spatial column, or specify the same spatial reference system for the data to be imported that the column uses.

msgcode: -2119

sqlstate: 38S7J

GSE2120N **No data was imported.**

Explanation

None of the shape data could be imported. All rows were rejected and written to the exception file.

User response

Consult the message file for the reasons why all the spatial data was rejected.

msgcode: -2120

sqlstate: 38S7K

GSE2121N **The value *restart-count* specifying the record to restart the operation is out of range. The shape file contains *record-count* records.**

Explanation

The specified restartCount *restart-count* must be a greater than or equal to 0 (zero), and less than or equal to *record-count*.

User response

Specify a valid number for restartCount, or specify a null value for the restartCount.

msgcode: -2121

sqlstate: 38S7L

GSE2122N **The SQL statement used to import the shape data does not fit into the internal buffer.**

Explanation

The SQL statement used to import the shape data into the table does not fit into the internal buffer. A possible reason for this might be too many columns are in the file.

User response

Import a smaller set of the attribute columns in the shape file.

msgcode: -2122

sqlstate: 38S7M

GSE2123N **A buffer to hold the data for *row-count* rows cannot be allocated.**

Explanation

IBM Spatial Support for Db2 for z/OS tried to use a single INSERT statement to import at least *row-count* rows, but a buffer to hold the data for these rows could not be allocated. Too much memory was required.

User response

Specify a commit count for the import that is less than *row-count*. Or, specify a smaller set of columns to be imported. This will reduce the amount of memory required.

msgcode: -2123

sqlstate: 38S7N

GSE2124N **An invalid type identifier *type-id* was found in the header of the shape file to be imported.**

Explanation

The data in the shape file does not appear to have a valid spatial data type. The shape file is possibly corrupted. The data was not imported.

User response

Verify that the shape file is valid.

msgcode: -2124

sqlstate: 38S7O

GSE2125N **A column in the shape file has an unsupported data type *type*.**

Explanation

The shape file contains a column whose data type is not supported by IBM Spatial Support for Db2 for z/OS. The shape file could not be imported.

User response

Import only a smaller set of the columns of the shape file and omit the column with the unsupported data type.

msgcode: -2125

sqlstate: 38S7P

GSE2126N **The header of the shape file *shape-file* is invalid.**

Explanation

The header of the shape file *shape-file* is invalid. The shape file cannot be imported.

The extension of the file name *shape-file* indicates in which part of the shape file the error was encountered. The file extensions include:

.shp
main file

.shx
index file

.dbf
dBASE file

.pri
projection file

User response

Verify and correct the header of the shape file.

msgcode: -2126

sqlstate: 38S7Q

GSE2127N **The offset *offset* for the record *record-number* in the shape index file *shx-file* is invalid.**

Explanation

The offset *offset* for the record *record-number* in the index file (.shx) *shx-file* is invalid. The offset must be greater than or equal to 50 and less than the total length of the main file (.shp) of the shape file. The offset is measured in 16 bit words.

User response

Verify and correct the shape file.

msgcode: -2127

sqlstate: 38S7R

GSE2128N **The length of the shape in record *record-number* of the shape index file *shx-file* is too short.**

Explanation

The length of the shape in record *record-number* found in the shape index file *shx-file* is too short. Each shape must consist of at least 4 bytes (two 16 bit words).

User response

Verify and correct the shape file.

msgcode: -2128

sqlstate: 38S7S

GSE2129N **IBM Spatial Support for DB2 for z/OS found an incorrect record number *record-number* in the shape file *shp-file* when expecting record number *expected-number*.**

Explanation

IBM Spatial Support for Db2 for z/OS found an incorrect record number *record-number* in the shape file *shp-file* when expecting record number *expected-number*.

User response

Verify and correct the shape file.

msgcode: -2129

sqlstate: 38S7T

GSE2130N **The size of the shape data *record-size* indicated in the shape file *shp-file* does not match the size indicated in the shape index file *index-size*.**

Explanation

The size of the shape data *record-size* indicated in the shape file *shp-file* does not match the size indicated in the shape index file *index-size*.

The main file of the shape file (.shp) is not consistent with the index file (.shx) and cannot be processed further.

User response

Verify and correct the shape file.

msgcode: -2130

sqlstate: 38S7U

GSE2131N **The data for record *record-number* in the dBASE file *dbf-file* is invalid.**

Explanation

The data for record *record-number* in the dBASE file *dbf-file* that contains the attribute information associated with the geometries in the shape file is invalid.

Possible explanations are:

- The first byte of the record is neither an asterisk ('*') nor a space (' ').
- The sum of all lengths of the columns in the dBASE file (.dbf) must be equal to the record size indicated in the header of the file.

User response

Verify and correct the dBASE file.

msgcode: -2131

sqlstate: 38S7V

GSE2132N **The data in shape file *shape-file* is invalid.**

Explanation

The data in shape file *shape-file* is corrupted. This shape file cannot be imported.

The file name *shape-file* indicates in which part of the shape file the error was encountered.

User response

Verify and correct the shape file.

msgcode: -2132

sqlstate: 38S7W

GSE2133N **The import operation failed because the column *schema-name.table-name.column-name* is not nullable.**

Explanation

The definition of the column *column-name* in the existing table *schema-name.table-name* indicates that the column may not contain nulls. The column is not included in the list of columns to be imported and Db2 would not produce the values for that column by any other means like default values, a generated column definition or any triggers.

The import operation cannot be completed successfully.

User response

Include the column in the list of columns to be imported, identify the column as id-column, or define an alternate way for Db2 to generate the values for that column during the import operation.

msgcode: -2133

sqlstate: 38S7X

GSE2134N **The spatial reference system associated with the data to be imported is not identical with the spatial reference system with the numeric identifier *srs-id*.**

Explanation

The spatial data in the file to be imported uses a spatial reference system with different offsets and scale factors than the spatial reference system with the numeric identifier *srs-id*. The data cannot be imported successfully.

User response

Specify a spatial reference system which has the same definition as the spatial reference system required by the data in the file to be imported. Refer to the DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS catalog view for the minimum and maximum coordinate values

and the offsets and scale factors applicable for the spatial reference system.

msgcode: -2134

sqlstate: 38S7Y

GSE2299N The shape file *file-name* has an invalid file size.

Explanation

Shape file *file-name* has an invalid file size. Shape files have a file size that is a multiple of 16-bit words. Therefore, their size is always even. The shape file might be corrupt. It cannot be used.

User response

Verify and correct the shape file.

msgcode: -2299

sqlstate: 38S9H

GSE3000N Null SRS identifier.

Explanation

A null value was passed to the function or method instead of a numeric spatial reference system identifier.

User response

Specify a numeric spatial reference system identifier for an existing spatial reference system. Refer to the IBM Spatial Support for Db2 for z/OS catalog view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS for the defined spatial reference systems.

msgcode: -3000

sqlstate: 38SU0

GSE3001N Invalid SRS identifier *srs-id*.

Explanation

The spatial reference system identifier *srs-id* that was provided to the spatial function or method does not identify an existing spatial reference system.

User response

Specify an existing numeric spatial reference system identifier that is defined in the IBM Spatial Support for Db2 for z/OS catalog view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS, or create a spatial reference system that is identified by *srs-id*.

msgcode: -3001

sqlstate: 38SU1

GSE3002N Null unit name.

Explanation

A null was specified as a unit of measure. The specification for a unit of measure must be the unit itself (for example, “meter”). It cannot be a null.

User response

Either omit the unit of measure when calling the spatial function or method, or specify an existing unit of measure. Consult the IBM Spatial Support for Db2 for z/OS catalog view DB2GSE.ST_UNITS_OF_MEASURE for supported units.

msgcode: -3002

sqlstate: 38SU2

GSE3003N Unknown unit *unit-name*.

Explanation

The unit *unit-name* that was provided to the spatial function or method does not identify an existing unit of measure.

User response

Either omit the unit of measure when calling the spatial function or method, or specify an existing unit of measure. Consult the IBM Spatial Support for Db2 for z/OS catalog view DB2GSE.ST_UNITS_OF_MEASURE for supported units.

msgcode: -3003

sqlstate: 38SU3

GSE3004N Unsupported conversion to unit *unit-name*.

Explanation

The conversion to the unit *unit-name* is not supported.

The functions ST_Area, ST_Buffer, ST_Length, and ST_Perimeter cannot accept a linear unit of measure if the given geometry is not in a projected coordinate system.

User response

Use one of the following methods:

- Omit the unit of measure when calling the spatial function or method.
- Specify an angular unit of measure.
- Project the geometry into a projected coordinate system using the ST_Transform function. Consult the

IBM Spatial Support for Db2 for z/OS catalog view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS for applicable spatial reference system.

msgcode: -3004

sqlstate: 38SU4

GSE3005N No unit in SRS.

Explanation

The spatial reference system for the geometry does not have an associated linear or angular unit. The operation cannot be performed in the requested unit of measure.

User response

Either represent the geometry in a correct spatial reference system, which does have an associated linear or angular unit of measure, or omit the unit parameter when you request the operation.

msgcode: -3005

sqlstate: 38SU5

GSE3006N Invalid internal type id.

Explanation

The internal data type identifier for this geometry is a null value and therefore invalid.

This error can occur if the internal representation of the geometry is corrupted, or if the geometry was not constructed by one of the supported constructor functions or methods.

User response

Construct the geometry again by using one of the supported constructor functions or methods.

msgcode: -3006

sqlstate: 38SU6

GSE3007N Unknown internal type id *type-id*.

Explanation

The value of the internal type identifier *type-id* for the geometry is not valid.

This error can occur if the internal representation of the geometry is corrupted, or if the geometry was not constructed by one of the supported constructor functions or methods.

User response

Construct the geometry again by using one of the supported constructor functions or methods.

msgcode: -3007

sqlstate: 38SU7

GSE3008N Internal type id mismatch (*type-id1*, *type-id2*).

Explanation

A mismatch of internal data type identifiers was found. IBM Spatial Support for Db2 for z/OS expected to retrieve a geometry whose internal data type identifier is *type-id2*, but instead retrieved a geometry whose internal data type identifier is *type-id1*.

This error can occur if the internal representation of the geometry is corrupted, or if the geometry was not constructed by one of the supported constructor functions or methods.

User response

Construct the geometry again by using one of the supported constructor functions or methods.

msgcode: -3008

sqlstate: 38SU8

GSE3009W Invalid part number *part-number*.

Explanation

The specified part number *part-number* is not valid. A null value was returned.

User response

If the geometry is not empty, then specify a valid part number, which should be greater than 0 (zero) and less then or equal to the maximum number of parts in the geometry type.

You can use the ST_NumGeometries function to determine the number of parts of the geometry type.

If the geometry is empty, the method should not be applied.

msgcode: +3009

sqlstate: 01HS0

GSE3010W Invalid ring number *ring-number*.

Explanation

The specified number *ring-number* for an internal ring is not valid. A null value was returned.

User response

If the polygon value is not empty, then specify a valid ring number, which should be greater than or equal to 1 (one) and less than or equal to the maximum number of interior rings in the polygon.

If the polygon is empty, the function or method should not be applied. You can use the function ST_NumInteriorRings to determine the number of interior rings of the polygon.

msgcode: +3010

sqlstate: 01HS1

GSE3011W **Invalid point number *point-number*.**

Explanation

The specified point number *point-number* is not valid. A null value was returned.

User response

If the curve value is not empty, then specify a valid point number, which should be greater than 0 (zero) and less than or equal to the maximum number of points in the curve. If the curve is empty, the function or method should not be applied.

You can use the ST_NumPoints function to determine the number of points used to define the curve.

msgcode: +3011

sqlstate: 01HS2

GSE3012N **Invalid DE9-IM *matrix*.**

Explanation

The intersection matrix *matrix* specified for the ST_Relate function is not valid. The matrix must be exactly 9 characters long, and each character in the matrix must be one of the following: 'T', 'F', '0', '1', '2', or '*'.

User response

Specify a valid intersection matrix.

msgcode: -3012

sqlstate: 38SU9

GSE3013N **Exterior ring is no ring.**

Explanation

The linestring that is to to serve as the new exterior ring for the polygon is not a ring. To be a ring, the linestring must be both simple and closed. One or both of these two conditions is not met.

User response

Specify a simple and closed linestring for the new exterior ring of the polygon.

msgcode: -3013

sqlstate: 38SUA

GSE3014N **Interior ring is no ring.**

Explanation

The linestring that is to serve as a new interior ring for the polygon is not a ring. To be a ring, the linestring must be both simple and closed. At least one of these two conditions is not met.

User response

Specify a simple and closed linestring for the new interior ring of the polygon.

msgcode: -3014

sqlstate: 38SUB

GSE3015N **Reason code = *reason-code*. Transformation to SRS *srs-id* failed.**

Explanation

The geometry could not be transformed from the spatial reference system it is represented into the spatial reference system with the numeric identifier *srs-id*. The transform failed with reason code *reason-code*.

The reason codes have the following meanings:

-2008
The geometry is invalid.

-2018
Not enough memory is available to successfully complete the transformation.

-2020
The spatial reference systems are not compatible. Both spatial reference systems must be based directly or indirectly on the same geographic coordinate system.

-2021
One or more points of the resulting geometry would be outside the maximum possible extent for the new spatial reference system. The resulting geometry cannot be represented in the new spatial reference system.

-2025
The definition of the new spatial reference system is not valid.

-2026
An internal error occurred during the projection of the geometry.

User response

Represent the geometry in a spatial reference system that can be transformed into the spatial reference system identified by *srs-id*, or specify a different spatial reference system identifier to transform the geometry into.

msgcode: -3015

sqlstate: 38SUC

GSE3016N **Unsupported cast *type-id1*, *type-id2*.**

Explanation

The attempted cast operation from the data type with the internal type identifier *type-id1* to the data type with the internal type identifier *type-id2* is not supported. The geometry cannot be processed further.

User response

Specify a supported cast operation. For more information, refer to the IBM Db2 SQL Reference for the supported cast functions.

msgcode: -3016

sqlstate: 38SUD

GSE3020N **Invalid Z coordinate and measure combination.**

Explanation

The geometries that are to be processed by the function or method are not represented using the same dimensions with respect to their Z coordinates and measures.

All the geometries must either contain Z coordinates or contain no Z coordinates. All the geometries must either contain measures or contain no measures.

User response

Provide geometries to the function or method that are represented using the same dimensions with respect to their Z coordinates and measures.

msgcode: -3020

sqlstate: 38SUH

GSE3021N **Reason code =*reason-code*. Locator failure.**

Explanation

An internal error occurred when a spatial function or method operated on a LOB locator. The reason code *reason-code* was returned by a locator function.

User response

Refer to the Db2 Application Development Guide to determine the meaning of *reason-code* returned from the LOB locator operation and correct the problem. If the problem persists, contact IBM Software Support.

msgcode: -3021

sqlstate: 38SUI

GSE3022N **Representation too long (*append-length* vs. *written-length* bytes).**

Explanation

The representation of the geometry in Geographic Markup Language (GML), well-known text (WKT), well-known binary (WKB), or the shape representation would be too long. From *append-length* bytes, only *written-length* bytes could be appended to the encoding. A representation of the geometry cannot be created.

User response

Simplify the geometry by omitting points that are not essential for the geometry. You can use the ST_Generalize function for this procedure. Alternatively, break down the geometry into several smaller geometries.

msgcode: -3022

sqlstate: 38SUJ

GSE3023N **Representation too short (*length* bytes).**

Explanation

The representation of the geometry in well-known binary (WKB) representation or the shape representation is only *length* bytes long. It needs to have at least 4 bytes for the shape representation, exactly 5 bytes for the well-known binary representation for empty geometries, and at least 9 bytes for the well-known binary representation for non-empty geometries. The binary representation must also be long enough to contain all of the geometry points.

User response

Provide a valid well-known binary representation or shape representation to the function or method.

msgcode: -3023

sqlstate: 38SUK

GSE3024N Internal geometry too short.

Explanation

The internal representation of the geometry is too short. It could not be processed further.

This error can occur if the internal representation of the geometry is corrupted, or if the geometry was not constructed by one of the supported constructor functions or methods.

User response

Construct the geometry again using one of the supported constructor functions or methods.

msgcode: -3024

sqlstate: 38SUL

GSE3025N Geometry inconsistent.

Explanation

The geometry value is inconsistent and cannot be processed any further.

User response

Recreate the geometry from a valid binary or text representation.

msgcode: -3025

sqlstate: 38SUM

GSE3026N Inconsistent no. of points (*indicated-number* vs. *data-number*).

Explanation

An internal parameter of the geometry indicates that the geometry data contains *indicated-number* points. But the actual geometry data contains *data-number* points. Because of this inconsistency, the geometry will not be used further in the processing.

This error can occur if the internal representation of the geometry is corrupted, or if the geometry was not constructed by one of the supported constructor functions or methods.

User response

Recreate the geometry using the functions or methods supported by IBM Spatial Support for Db2 for z/OS.

msgcode: -3026

sqlstate: 38SUN

GSE3027N Point is empty.

Explanation

It is invalid to specify an X coordinate, Y coordinate, Z coordinate, or measure for an empty point.

If the point is constructed by the constructor function ST_Point, the point's X and Y coordinates must both be null. Furthermore, no Z coordinate or measure should be specified unless it is a null value.

If the mutators ST_X, ST_Y, ST_Z, or ST_M are used to modify an empty point, the point's X and Y coordinates must both be null. No Z coordinate or measure should be specified unless it is null.

User response

Use mutators ST_X, ST_Y, ST_Z, or ST_M to modify points that are not empty, or construct the point by specifying both X and Y coordinates with values that are not null.

msgcode: -3027

sqlstate: 38SUO

GSE3028N Inconsistent coordinates.

Explanation

If a new point is constructed, both the X and Y coordinates must be specified. Both coordinates must be either null or not null.

If both coordinate values are null, the resulting point will be empty. In that case, no Z coordinate or measure should be specified unless it is null.

User response

Specify null values for both the X and Y coordinates, or specify values that are not null for both coordinates.

msgcode: -3028

sqlstate: 38SUP

GSE3029N Invalid byte order *byte-order*.

Explanation

The byte order in the binary representation of the geometry must be either 0 (zero) or 1 (one), but it is *byte-order*.

In the well-known binary representation, a byte order of 0 (zero) indicates big endian format, and a byte order of 1 (one) indicates little endian format.

User response

Correct the byte order in the binary representation so that it is either 0 (zero) or 1 (one).

msgcode: -3029

sqlstate: 38SUQ

GSE3030N **Invalid number of points *num-points* in geometry.**

Explanation

The geometry has an invalid number of points *num-points*. This number must be greater than or equal to 0 (zero).

If the geometry is not empty, then the following conditions must be met:

point
The geometry must have exactly one point.

linestring
The geometry must have 2 or more points defining it.

polygon
The geometry must have 3 or more points defining it.

User response

Construct the geometry by using the functions or methods supported by IBM Spatial Support for Db2 for z/OS.

msgcode: -3030

sqlstate: 38SUR

GSE3031N **Invalid extent (*min-coord* vs. *max-coord*) in geometry.**

Explanation

The extent of the geometry in one of the dimensions is invalid. The minimum coordinate *min-coord* must be less than or equal to the maximum coordinate *max-coord* for all dimensions of the geometry.

User response

Construct the geometry by using the functions or methods supported by IBM Spatial Support for Db2 for z/OS.

msgcode: -3031

sqlstate: 38SUS

GSE3032N **Aggregation failure.**

Explanation

A mismatch between internal identifiers was encountered for the computation of a spatial aggregate.

Aggregate functions are not supported if used in any of the following situations:

- In the partitioned environment.
- A GROUP BY clause is used in the query that contains the spatial aggregate.
- Any function other than the Db2 aggregate function MAX is used.
- The aggregate function is not used in the correct context.

User response

Make sure that you use the aggregate function in a way that is supported by IBM Spatial Support for Db2 for z/OS.

msgcode: -3032

sqlstate: 38SUT

GSE3033N **Invalid binary data (type ids *type-id1*, *type-id2*).**

Explanation

A binary representation that is passed as input to this spatial function or method has to represent a geometry whose data type identifier is *type-id2*. But the representation that was actually passed to the function or method represents a geometry whose data type identifier is *type-id1*. No geometry could be constructed.

User response

Either call the correct function or method which constructs geometries of type *type-id2* or correct the binary representation to represent a geometry of *type-id1*.

msgcode: -3033

sqlstate: 38SUU

GSE3034N **Invalid text data (type ids *type-id1*, *type-id2*).**

Explanation

A text representation that is passed as input to this spatial function or method has to represent a geometry whose data type identifier is *type-id2*. But the representation that was actually passed to the function represents a geometry whose data type

identifier is *type-id1*. No geometry could be constructed.

User response

Either call the correct function which constructs geometries of type *type-id1* or correct the text representation to represent a geometry of *type-id2*.

msgcode: -3034

sqlstate: 38SUV

GSE3035W Curve not changed.

Explanation

The curve was not changed because the specified point to be appended to the curve was empty.

User response

Append a point that is not empty to the curve.

msgcode: +3035

sqlstate: 01HS3

GSE3036W Geometry not accurate.

Explanation

The resulting geometry could not be represented accurately in the spatial reference system. One of the scale factors is too small and does not allow for a high enough precision to represent each point that defines the resulting geometry.

For example, consider a linestring with a well-known text representation of 'linestring m (10 10 8, 10 11 12)' represented in a spatial reference system that includes a scale factor of 1 (one) for X coordinates and a scale factor of 1 (one) also for Y coordinates. If the function ST_MeasureBetween is applied to that linestring, and the upper and lower bounds for the measures are 9 and 10, respectively, the resulting linestring, represented in its well-known text representation, would have to be 'linestring m (10 10.25 9, 10 10.50 10)'. However, the scale factor of 1 (one) for the Y coordinates prevents the representation of fractions. The coordinates 10.25 and 10.50 cannot be represented without rounding that would produce an incorrect result. Such coordinates will be removed from the geometry.

User response

Represent the geometry in a spatial reference system that uses larger scale factors. Alternatively, choose different parameters that influence the resulting geometry.

msgcode: +3036

sqlstate: 01HS4

GSE3037N Invalid GML, expecting *char* instead of *string* at position *position*.

Explanation

A character *char* was expected in the Geography Markup Language of the geometry, but the text *string* was found instead at position *position*. The GML representation is not valid. IBM Spatial Support for Db2 for z/OS cannot construct the geometry successfully.

User response

Correct the GML representation and construct the geometry again.

msgcode: -3037

sqlstate: 38SUW

GSE3038N Invalid GML, expecting *expected-tag* instead of *given-tag* at position *position*.

Explanation

The tag *given-tag* was found in the Geography Markup Language of the geometry at position *position*, but a tag *expected-tag* was expected. The GML representation is not valid. IBM Spatial Support for Db2 for z/OS cannot construct the geometry successfully.

User response

Correct the GML representation and construct the geometry again.

msgcode: -3038

sqlstate: 38SUX

GSE3039N Invalid GML, expecting number instead of *text* at position *position*.

Explanation

Unexpected text *text* was found in the Geography Markup Language of the geometry at position *position*. A number representing a coordinate was expected instead. The GML representation is not valid. IBM Spatial Support for Db2 for z/OS cannot construct the geometry successfully.

User response

Correct the GML representation and construct the geometry again.

msgcode: -3039

sqlstate: 38SUY

GSE3040N Invalid GML type *type*.

Explanation

An unknown type *type* was specified in the Geography Markup Language of the geometry. The GML supports points, linestrings, polygons, multipoints, multilinestrings, and multipolygons. IBM Spatial Support for Db2 for z/OS cannot construct the geometry successfully.

User response

Correct the GML representation and construct the geometry again.

msgcode: -3040

sqlstate: 38SUZ

GSE3041N GML point has been incorrectly specified.

Explanation

The problem occurred due to one of the following reasons:

- A point, represented using the Geography Markup Language, can only have one set of coordinates. The given point had either no set of coordinates or more than one set.
- The set of coordinates is not enclosed by corresponding <gml:coord> or <gml:coordinates> tags.

The GML representation is not valid. IBM Spatial Support for Db2 for z/OS cannot construct the geometry successfully.

User response

Correct the GML representation and construct the geometry again.

msgcode: -3041

sqlstate: 38SV0

GSE3042N Could not read *number-bytes* bytes from locator at offset *offset*. Total length of data is *length*.

Explanation

An attempt was made to read *number-bytes* bytes from the locator, starting at the offset *offset*. This exceeds the total length of the data *length* that is referenced by the locator. The data might be truncated.

For binary representations of a geometry, the binary representation might indicate an invalid binary encoding. The encoded geometry has fewer points than the header indicates.

User response

Verify and correct the representation of the geometry. Make sure that the binary or textual representation does not get truncated before it is passed to the IBM Spatial Support for Db2 for z/OS function.

msgcode: -3042

sqlstate: 38SV1

GSE3043N Invalid number of parts *number-parts*.

Explanation

The number of parts *number-parts* indicated in the binary representation of the geometry is invalid. The number of parts must be larger than 0 (zero) and match the actual number of parts supplied in the encoding.

User response

Specify the correct number of parts or supply all parts for the geometry.

msgcode: -3043

sqlstate: 38SV2

GSE3044N Invalid number of rings *number-rings*.

Explanation

The number of rings *number-rings* indicated in the binary representation of the polygon or multipolygon is invalid. The number of rings must be larger than 0 (zero) and match the actual number of parts supplied in the encoding.

User response

Specify the correct number of rings or supply all rings for the geometry.

msgcode: -3044

sqlstate: 38SV3

GSE3045N Invalid part offset *part-offset* in shape.

Explanation

An invalid offset *part-offset* for a part in the shape representation of the geometry was encountered. A part offset must be larger than or equal to 0 (zero), and each part offset must be larger than the preceding one. The shape representation is not valid. IBM Spatial Support for Db2 for z/OS cannot construct the geometry successfully.

User response

Correct the shape representation and construct the geometry again.

msgcode: -3045

sqlstate: 38SV4

GSE3046N Invalid type ID *type-id* in shape.

Explanation

The shape representation of the geometry contains an invalid type identifier *type-id*. The shape data is possibly corrupted. IBM Spatial Support for Db2 for z/OS cannot construct the geometry successfully.

User response

Verify and correct the shape representation of the geometry.

msgcode: -3046

sqlstate: 38SV5

GSE3047N Invalid length *shape-length* of shape encoding for type *type*, expecting only *expected-length* bytes.

Explanation

The shape encoding contains *shape-length* bytes, which is too long. To encode a geometry of the specified type *type*, only *expected-length* bytes are required. The shape data is possibly corrupted. IBM Spatial Support for Db2 for z/OS cannot construct the geometry successfully.

User response

Verify and correct the shape representation of the geometry.

msgcode: -3047

sqlstate: 38SV6

GSE3048N Invalid WKT format, expecting *char* instead of *string*.

Explanation

A character *char* was expected in the well-known text representation of the geometry, but the text *string* was found instead. The well-known text representation is not valid. IBM Spatial Support for Db2 for z/OS cannot construct the geometry successfully.

User response

Correct the well-known text representation and construct the geometry again.

msgcode: -3048

sqlstate: 38SV7

GSE3049N Invalid WKT format, expecting a number instead of *text*.

Explanation

An unexpected text *text* was found in the well-known text representation of the geometry. A number representing a coordinate was expected instead. The well-known text representation is not valid. IBM Spatial Support for Db2 for z/OS cannot construct the geometry successfully.

User response

Correct the well-known text representation and construct the geometry again.

msgcode: -3049

sqlstate: 38SV8

GSE3050N Unexpected parenthesis in WKT format at *text*.

Explanation

An unexpected opening or closing parenthesis was found in the well-known text representation of the geometry at *text*. The well-known text representation is not valid. IBM Spatial Support for Db2 for z/OS cannot construct the geometry successfully.

User response

Correct the well-known text representation and construct the geometry again.

msgcode: -3050

sqlstate: 38SV9

GSE3051N Parenthesis mismatch in WKT format, expecting *parenthesis*.

Explanation

The end of the well-known text representation was reached unexpectedly. A parenthesis *parenthesis* was expected. The well-known text representation is not valid. IBM Spatial Support for Db2 for z/OS cannot construct the geometry successfully.

User response

Correct the well-known text representation and construct the geometry again.

msgcode: -3051

sqlstate: 38SVA

GSE3052N Unknown type *type* in WKT.

Explanation

The well-known text representation of the geometry contains an unknown type name of *type*. The well-known text representation is not valid. IBM Spatial Support for Db2 for z/OS cannot construct the geometry successfully.

User response

Correct the well-known text representation and construct the geometry again.

msgcode: -3052

sqlstate: 38SVB

GSE3053N Invalid type id *type-id* in WKB.

Explanation

The well-known binary representation of the geometry contains an invalid type identifier *type-id*. The data is possibly corrupted. IBM Spatial Support for Db2 for z/OS cannot construct the geometry successfully.

The type identifiers of separate parts in a geometry type (multipoint, multilinestring, or multipolygon) must have the same indicators for the Z and M coordinates as the geometry type itself.

User response

Verify and correct the well-known binary representation of the geometry.

msgcode: -3053

sqlstate: 38SVC

GSE3300N Invalid grid size *grid-size-number*.

Explanation

The grid size identified by its position *grid-size-number* is invalid. One of the following invalid specifications was made when the grid index was created with the CREATE INDEX statement:

- A number less than 0 (zero) was specified as the grid size for the first, second, or third grid level.
- 0 (zero) was specified as the grid size for the first grid level.
- The grid size specified for the second grid level is less than the grid size of the first grid level but it is not 0 (zero).
- The grid size specified for the third grid level is less than the grid size of the second grid level but it is not 0 (zero).
- The grid size specified for the third grid level is greater than 0 (zero) but the grid size specified for the second grid level is 0 (zero).

The function ST_GetIndexParms can be used to retrieve the values used for the parameters specified when the index was created.

User response

Drop the grid index and create a new grid index using valid grid sizes only.

msgcode: -3300

sqlstate: 38SI0

GSE3301N Invalid z-order parameter *parameter-number*.

Explanation

The parameter identified by its position *parameter-number* for a Z-Order index contains an invalid value. One of the following invalid specifications was made in the CREATE INDEX statement that was used to create the index to which the geometry is to be added:

- A null value was specified for the parameter.
- A negative number was specified for a scale factor (this rule applies to parameter numbers 2 and 4 only).

The function ST_GetIndexParms can be used to retrieve the values used for the parameters specified when the index was created.

User response

Drop the spatial z-order index and create a new index using only valid parameters.

msgcode: -3301

sqlstate: 38SI1

GSE3302N No point to be indexed.

Explanation

The geometry to be indexed using a Z-Order index is not a point. The Z-Order index supports only points, and the index entry cannot be generated.

User response

Do not insert a geometry that is not a point into a column that has a Z-Order index defined on it. Either drop the index or do not insert the geometry.

msgcode: -3302

sqlstate: 38SI2

GSE3303N Invalid quad tree parameter *parameter-number*.

Explanation

An invalid parameter was specified when the quad tree index was created. The parameter is identified by its position *grid-size-number*.

One of the following invalid specifications was made:

- A null value was specified for the parameter.
- A negative number was specified for a scale factor (this rule applies to parameter numbers 3 and 5 only).
- A value less than 1 (one) was specified for the first parameter.

The function ST_GetIndexParms can be used to retrieve the values used for the parameters specified when the index was created.

User response

Drop the spatial quad tree index and create a new index using only valid parameters.

msgcode: -3303

sqlstate: 38SI3

GSE3400C Unknown error *error-code*.

Explanation

An internal error with code *error-code* was encountered when a geometry was processed.

User response

Note the error and contact IBM Software Support.

msgcode: -3400

sqlstate: 38SS0

GSE3402C Insufficient memory.

Explanation

Not enough memory was available for the spatial function or method that you invoked.

User response

Make more memory available to the Db2 process that executes the function or method.

msgcode: -3402

sqlstate: 38SS2

GSE3403N Invalid geometry type.

Explanation

An invalid type of geometry was passed to the function or method that you invoked.

User response

Specify a valid geometry. For more information, refer to the IBM Spatial Support for Db2 for z/OS User's Guide and Reference.

msgcode: -3403

sqlstate: 38SS3

GSE3405N Too many parts specified.

Explanation

The number of parts indicated in the binary or text representation of the geometry is greater than the actual number of parts supplied. Either the number of parts indicated is too high or not all the parts were supplied.

User response

Specify the correct number of parts or supply all parts for the geometry.

msgcode: -3405

sqlstate: 38SS5

GSE3406N Incorrect geometry type.

Explanation

The wrong type of geometry was passed to the function or method that you invoked. For example, a linestring might have been passed to a function or method that takes only polygons as input.

User response

Either pass to the function or method a type of geometry that it can process, or use a function or method that accepts the type of geometry that you want to pass.

msgcode: -3406

sqlstate: 38SS6

GSE3407N Text is too long.

Explanation

The geometry contains too much detail to be converted to its well-known text representation. The well-known text representation exceeds the maximum allowable length (2 gigabytes).

User response

Simplify the geometry - for example, by using the ST_Generalize function - or convert the geometry to its well-known binary representation.

msgcode: -3407

sqlstate: 38SS7

GSE3408N Invalid parameter value.

Explanation

An invalid parameter was encountered.

User response

Refer to the IBM Spatial Support for Db2 for z/OS User's Guide and Reference for the function's correct syntax and retry the operation. If the problem persists, contact IBM Software Support.

msgcode: -3408

sqlstate: 38SS8

GSE3409N Invalid geometry produced.

Explanation

The parameters provided for the function or method have produced an invalid geometry; for example, an invalid shape representation. An invalid geometry is one that violates a geometry property.

User response

Construct the geometry again from a valid representation.

msgcode: -3409

sqlstate: 38SS9

GSE3410N Incompatible geometries.

Explanation

The function or method expected two geometries of a certain type and did not receive them. For example, the ST_AddPoint function expects two geometries, one a representation and the other a point.

User response

Specify geometries that the function or method accepts as valid input. To determine what types of geometries are valid for this function or method, refer to the IBM Spatial Support for Db2 for z/OS User's Guide and Reference.

msgcode: -3410

sqlstate: 38SSA

GSE3411N Invalid geometry.

Explanation

The function or method cannot process the geometry passed to it because one or more properties of the geometry violate the geometry's integrity.

User response

Use the ST_IsValid function to validate the geometry. Construct the geometry again from a correct representation if it is not valid.

msgcode: -3411

sqlstate: 38SSB

GSE3412N Too many points.

Explanation

The construction of a geometry has exceeded the 1-megabyte storage limit; the geometry has too many points.

User response

Construct a geometry that contains fewer points. Or, if possible, remove some points. For performance and storage considerations, include only those points that are needed to render a geometry.

msgcode: -3412

sqlstate: 38SSC

GSE3413N Geometry too small.

Explanation

The geometry returned by the ST_Difference, ST_Intersection, ST_SymDifference, or ST_Union function is too small to be represented accurately in the current spatial reference system.

For example, this can happen if the internal computation constructs a very thin polygon, but the scale factor of the spatial reference system is so low that the geometry would collapse to a linestring if it were to be represented in this spatial reference system. It would lose its property as a polygon.

User response

Use a spatial reference system for the calculation which allows for a higher resolution. The ST_Transform function can be used to convert a geometry from one spatial reference system into another.

msgcode: -3413

sqlstate: 38SSD

GSE3414N Buffer out of bounds.

Explanation

The ST_Buffer function has created a buffer around the provided geometry that is outside the range of the coordinates to which the spatial reference system applies.

Refer to the IBM Spatial Support for Db2 for z/OS catalog view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS to determine the minimum and maximum absolute values for each of the dimensions. These values must not be exceeded by the calculated buffer.

User response

Either reduce the distance to be used for the buffer calculation, or change the spatial reference system in which the calculation is done. The ST_Transform function can be used to convert geometries from one spatial reference system into another.

msgcode: -3414

sqlstate: 38SSE

GSE3415N Invalid scale factor.

Explanation

A scale factor for any of the four dimensions (X, Y, Z, and M) must be greater than or equal to 1 (one).

User response

Use a correctly defined spatial reference system to represent the geometry.

msgcode: -3415

sqlstate: 38SSF

GSE3416N Coordinate out of bounds.

Explanation

A coordinate cannot be represented in the spatial reference system because, in at least one dimension, it exceeds the possible minimum or maximum absolute value within the system's range of values.

Refer to the IBM Spatial Support for Db2 for z/OS catalog view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS to determine the minimum and maximum absolute values for each of the dimensions.

User response

Determine whether the coordinate is correct. If it is, determine whether it fits within the extent of the spatial reference system that you are using. For information about this spatial reference system, consult the DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS catalog view.

msgcode: -3416

sqlstate: 38SSG

GSE3417N Invalid coordsys definition.

Explanation

There are one or more errors in the text representation of the definition of the coordinate system on which the geometry's spatial reference system is based. The representation cannot be converted into a valid projection.

User response

Verify the coordinate system definition of the spatial reference system. Alternatively, construct the geometry in a spatial reference system that is associated with a valid coordinate system. The ST_EqualCoordsys function can be used to verify the coordinate system definition by comparing it with itself.

msgcode: -3417

sqlstate: 38SSH

GSE3418N Projection error.

Explanation

An error occurred during an attempt to project a geometry to another spatial reference system.

User response

Make sure that the geometry is within the legal domain of the projection.

msgcode: -3418

sqlstate: 38SSI

GSE3419N Polygon rings overlap.

Explanation

The rings of a polygon overlap. By definition, the inner and outer rings of a polygon must not overlap. They can intersect only at a tangent, which means the rings can only touch but not cross each other.

User response

Specify the coordinates for the polygon that will not produce overlapping rings. Note that the scale factors of the spatial reference system for the geometry have an influence on the precision.

msgcode: -3419

sqlstate: 38SSJ

GSE3420N Too few points.

Explanation

The error is a result of one of the following:

- Linestrings must consist of at least two points, and polygons must consist of at least four points.
- The geometry cannot be constructed from the points that you have specified.

Note that if the geometry to be constructed is empty, these rules do not apply.

User response

Construct the geometry again from a valid set of points.

msgcode: -3420

sqlstate: 38SSK

GSE3421N Polygon is not closed.

Explanation

The inner and outer rings that define the polygon must be closed. A ring is closed if the start and end points are identical in the X and Y dimensions. If the polygon has Z coordinates, then the start and end points must also be identical to the Z coordinates. Note that this rule does not apply to measures, which can be different for the start and end points.

User response

Specify inner and outer rings for the polygon that have the same points for the start and end points in the X and Y dimension. If the polygon has Z coordinates, the start and end points of the Z coordinate points also have to be identical. If the polygon has measures, the start and end points can be different.

msgcode: -3421

sqlstate: 38SSL

GSE3422N Invalid exterior ring.

Explanation

The exterior ring of the polygon is not valid.

The exterior ring of a polygon must enclose all interior rings of the polygon. All interior rings have to be completely inside the area that is defined by the outer ring and must not cross the exterior ring.

User response

Specify a geometry that consists of a valid set of interior and exterior rings, where the interior rings lie fully within the area that is enclosed by the exterior ring to represent it.

If the geometry has multiple polygons, use a multipolygon.

msgcode: -3422

sqlstate: 38SSM

GSE3423N Polygon has no area.

Explanation

The specified polygon lacks an interior that covers an area that is not the empty set in the X and Y dimensions.

A geometry is a polygon only if its coordinates span two dimensions in the 2-dimensional space defined by the X and Y coordinates.

User response

Specify a polygon that encloses an area that is not empty. If the polygon is empty, construct an empty polygon.

msgcode: -3423

sqlstate: 38SSN

GSE3424N Exterior rings overlap.

Explanation

The exterior rings of distinct polygons in a multipolygon overlap. Distinct polygons in a multipolygon must not overlap, and the boundaries must touch only at a finite number of points. That means the polygons must not share line segments.

The scale factors of the spatial reference system that is used to represent the geometry influences the precision that applies to the coordinates. Rounding operations performed when the geometry is converted to the representation in the spatial reference system might cause a loss in precision and, subsequently, this error.

User response

Specify coordinates for the polygon that will not produce overlapping rings.

Note that the scale factors of the spatial reference system have an influence on precision.

Refer to the IBM Spatial Support for Db2 for z/OS catalog view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS for the scale factor used for the spatial reference system in which the geometry will be represented.

msgcode: -3424

sqlstate: 38SSO

GSE3425N Polygon intersects itself.

Explanation

A ring of a polygon cannot intersect itself. The start and end points on each ring of the polygon must be reached twice when traversing the ring. All other points must only be reached once. This holds true also for the line segments that define the rings of the polygon.

The scale factors of the spatial reference system that is used to represent the geometry influences the precision that applies to the coordinates. Rounding operations performed when the geometry is converted to the representation in the spatial reference system might cause a loss in precision and, subsequently, this error.

User response

Construct a valid polygon in which the rings do not intersect themselves.

Refer to the IBM Spatial Support for Db2 for z/OS catalog view DB2GSE.ST_SPATIAL_REFERENCE_SYSTEMS for the scale factor used for the spatial reference system in which the geometry will be represented.

msgcode: -3425

sqlstate: 38SSP

GSE3426N Invalid number of parts.

Explanation

The number of parts indicated in the binary or text representation of the geometry is not equal to the actual number of parts supplied. Either the number is too low or too many parts were supplied to the function or method.

User response

Specify the correct number of parts or supply all parts for the geometry.

msgcode: -3426

sqlstate: 38SSQ

GSE3427N Incompatible SRSs.

Explanation

The two spatial reference systems are not compatible. They cannot be transformed into or compared with one another. The operation cannot be completed successfully.

User response

Specify two compatible spatial reference systems.

msgcode: -3427

sqlstate: 38SSR

GSE3428N BLOB too small.

Explanation

The number of bytes in the specified binary representation of the geometry is too small.

User response

Specify a valid binary representation of the geometry.

msgcode: -3428

sqlstate: 38SSS

GSE3429N Invalid geometry type.

Explanation

An invalid internal geometry type was encountered. The geometry is not valid and will not be processed any further.

User response

Construct the geometry again from a valid binary or text representation.

msgcode: -3429

sqlstate: 38SST

GSE3430N Invalid byte order.

Explanation

The byte order in the binary representation of the geometry has an invalid value. The byte order must be 0 (zero) or 1 (one).

In the well-known binary representation, a byte order of 0 (zero) indicates big endianess, and a byte order of 1 (one) indicates little endianess.

User response

Specify a valid byte order in the binary representation for the geometry.

msgcode: -3430

sqlstate: 38SSU

GSE3431N Empty geometry.

Explanation

An empty geometry was passed to the ST_AsBinary function, even though it is not allowed as input.

User response

Edit the SQL statement that you submitted so that only non-empty geometries will be passed to the ST_AsBinary function. For example, you can use the ST_IsEmpty function in the WHERE clause to exclude empty geometries.

msgcode: -3431

sqlstate: 38SSV

GSE3432N Invalid end point.

Explanation

The specified point is intended to be appended to the curve, but it is not valid.

User response

Specify a valid point to be appended.

msgcode: -3432

sqlstate: 38SSW

GSE3433N Point not found.

Explanation

The specified point is intended to be changed or removed, but it does not exist in the curve.

User response

Specify a point that does exist in the curve.

msgcode: -3433

sqlstate: 38SSX

GSE3600N No index specified.

Explanation

No valid index was specified. The index schema parameter, the index name parameter, or both, are null. The index parameter values cannot be derived.

User response

Specify a valid spatial index to retrieve the parameter information.

msgcode: -3600

sqlstate: 38SQ0

GSE3601N Invalid spatial index name *schema-name.index-name*.

Explanation

The specified name of the index for which you want parameter information retrieved does not exist or does not identify a spatial index. This name is *schema-name.index-name*.

User response

Specify an existing spatial index to retrieve the parameter information.

msgcode: -3601

sqlstate: 38SQ1

GSE3602N Invalid parameter number *number* specified.

Explanation

The parameter number *number* is not valid for the specified spatial index.

The following limits apply for the different types of spatial indexes:

grid index
Parameter numbers between 1 (one) and 3.

z-order index
Parameter numbers between 1 (one) and 4.

quad-tree index
Parameter numbers between 1 (one) and 5.

User response

Specify a valid parameter number for the spatial index. Consult the Db2 system catalog for the type of the spatial index.

msgcode: -3602

sqlstate: 38SQ2

GSE3603N **Invalid column name.**

Explanation

The specified column does not exist in the table. At least one of the following - table schema, table name, or column name - is a null value. The index parameter for an index on a column cannot be derived.

User response

Specify an existing column which has a spatial index defined on it.

msgcode: -3603

sqlstate: 38SQ3

GSE4000N **Required parameter *parameter-name* is missing.**

Explanation

The required parameter was not found.

User response

Specify the required parameter and try to execute the command again.

msgcode: -4000

sqlstate: 38SB0

GSE4001N **An error occurred while IBM Spatial Support for DB2 for z/OS was allocating an environment handle.**

Explanation

An environment handle could not be allocated using the Call Level Interface (CLI). The operation cannot be completed successfully.

User response

Verify the CLI configuration. If the source of the problem cannot be found and corrected, contact IBM Software Support.

msgcode: -4001

sqlstate: 38SB1

GSE4002N **An error occurred while IBM Spatial Support for DB2 for z/OS was allocating a connection handle. CLI error *cli-error* and native error code = *native-error-code*.**

Explanation

An unexpected error *cli-error* with native error code = *native-error-code* occurred while IBM Spatial Support for Db2 for z/OS was allocating a connection handle.

User response

Look up the detailed error message *cli-error*. Correct the error and execute the command again. If the problem persists, contact IBM Software Support.

msgcode: -4002

sqlstate: 38SB2

GSE4003N **An error occurred while IBM Spatial Support for DB2 for z/OS was connecting to the database. CLI error *cli-error* and native error code = *native-error-code*.**

Explanation

An unexpected error *cli-error* with native error code = *native-error-code* occurred while IBM Spatial Support for Db2 for z/OS was connecting to the database.

User response

Look up the detailed error message *cli-error*. Correct the error and execute the command again. If the problem persists, contact IBM Software Support.

msgcode: -4003

sqlstate: 38SB3

GSE4004N **An error occurred while IBM Spatial Support for DB2 for z/OS was allocating a statement handle. CLI error *cli-error* and native error code = *native-error-code*.**

Explanation

An unexpected error *cli-error* with native error code = *native-error-code* occurred while IBM Spatial Support for Db2 for z/OS was allocating a statement handle.

User response

Look up the detailed error message *cli-error*. Correct the error and execute the command again. If the problem persists, contact IBM Software Support.

msgcode: -4004

sqlstate: 38SB4

GSE4005N **An error occurred while an SQL statement was being prepared. CLI error *cli-error* and native error code = *native-error-code*.**

Explanation

An unexpected error *cli-error* with native error code = *native-error-code* occurred while IBM Spatial Support for Db2 for z/OS was preparing an SQL statement.

User response

Look up the detailed error message *cli-error*. Correct the error and execute the command again. If the problem persists, contact IBM Software Support.

msgcode: -4005

sqlstate: 38SB5

GSE4006N **An error occurred while IBM Spatial Support for DB2 for z/OS was binding parameters to an SQL statement. CLI error *cli-error* and native error code = *native-error-code*.**

Explanation

An unexpected error *cli-error* with native error code = *native-error-code* occurred while IBM Spatial Support for Db2 for z/OS was binding parameters to an SQL statement.

User response

Look up the detailed error message *cli-error*. Correct the error and execute the command again. If the problem persists, contact IBM Software Support.

msgcode: -4006

sqlstate: 38SB6

GSE4007N **An error occurred while IBM Spatial Support for DB2 for z/OS was executing an SQL statement. CLI error *cli-error* and native error code = *native-error-code*.**

Explanation

An unexpected error *cli-error* with native error code = *native-error-code* occurred while IBM Spatial Support for Db2 for z/OS was executing an SQL statement.

User response

Look up the detailed error message *cli-error*. Correct the error and execute the command again. If the problem persists, contact IBM Software Support.

msgcode: -4007

sqlstate: 38SB7

GSE4008N **An error occurred while IBM Spatial Support for DB2 for z/OS was ending a transaction. CLI error *cli-error* and native error code = *native-error-code*.**

Explanation

An unexpected error *cli-error* with native error code = *native-error-code* occurred while IBM Spatial Support for Db2 for z/OS was ending a transaction.

User response

Look up the detailed error message *cli-error*. Correct the error and execute the command again. If the problem persists, contact IBM Software Support.

msgcode: -4008

sqlstate: 38SB8

GSE4009N **The option, *option*, is invalid.**

Explanation

The specified option, *option*, is invalid.

User response

Specify a valid option and repeat the command.

msgcode: -4009

sqlstate: 38SB9

GSE9990C **An internal error occurred: *error-text*.**

Explanation

IBM Spatial Support for Db2 for z/OS encountered an unexpected internal error with the text *error-text*.

User response

Read the given *error-text*. If the problem cannot be resolved, contact IBM Software Support.

msgcode: -9990

sqlstate: 38SZY

GSE9999C Internal message failure.

Explanation

An internal failure occurred while IBM Spatial Support for Db2 for z/OS was retrieving an error message.

User response

Contact IBM Software Support.

msgcode: -9999

sqlstate: 38SZZ

Information resources for Db2 11 for z/OS and related products

Information about Db2 11 for z/OS and products that you might use in conjunction with Db2 11 is available online in .

You can find the complete set of product documentation for Db2 11 for z/OS in IBM Knowledge Center.

You can also download other PDF format manuals for Db2 11 for z/OS from in .

© Copyright IBM Corp. 2007, 2013

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785 US

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing Legal and Intellectual Property Law IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785 US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

© Copyright IBM Corp. 2007, 2013

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as shown below:

© (*your company name*) (*year*).
Portions of this code are derived from IBM Corp. Sample Programs.
© Copyright IBM Corp. (*enter the year or years*).

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered marks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at: http://www.ibm.com/legal/copytrade.shtml.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM’s Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled “Cookies, Web Beacons and Other Technologies” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.

Glossary

The glossary is available in IBM Knowledge Center.

See the Glossary topic for definitions of Db2 for z/OS terms.

© Copyright IBM Corp. 2007, 2013

Index

A

accessibility
  keyboard x
  shortcut keys x
alter_cs command 232
alter_srs command 233
angular units
  coordinate systems 221
  supported 224
azimuthal projections 17

C

catalog views
  DB2GSE.GEOMETRY_COLUMNS 77
  DB2GSE.SPATIAL_REF_SYS 77
  ST_COORDINATE_ SYSTEMS 78
  ST_GEOMETRY_ COLUMNS 79
  ST_SIZINGS 80
  ST_SPATIAL_ REFERENCE_SYSTEMS 81
  ST_UNITS_OF_ MEASURE 84
commands
  alter_cs 232
  alter_srs 233
  create_cs 235
  create_idx 237
  create_srs 239
  create_srs_2 241
  disable_spatial 243
  drop_cs 244
  drop_idx 245
  drop_srs 245
  enable_spatial 246
  for DSN5SCLP 231
  function_level 247
  import_shape 248
  register_spatial_column 253
  unregister_spatial_column 254
comparison functions
  container relationships 92
  identical geometries 100
  intersections between geometries 95
  overview 90, 91
  values 90
conformal projections 17
constructor functions
  coding
    examples 87
  ESRI shape representation 89
  Geography Markup Language (GML) representation 89
  well-known binary representation 89
  well-known text representation 88
coordinate systems
  creating 18
  selecting 18
  ST_COORDINATE_ SYSTEMS catalog view 78
coordinate systems *(continued)*
  ST_SPATIAL_ REFERENCE_SYSTEMS catalog view 81
  supported 13
  syntax 221
coordinates
  improving performance 23
  maximum
    determining 25
  minimum
    determining 25
  obtaining 100
  spatial reference system
    converting 18
  spatial reference systems 18
create_cs command 235
create_idx command 237
create_srs command 239
create_srs_2 command 241
creating
  inline spatial columns 29
  spatial columns 28
  spatial grid indexes 39

D

data formats
  Geography Markup Language (GML) 219
  shape representation 219
  well-known binary (WKB) representation 217
  well-known text (WKT) representation 213
data types
  choosing 28
databases
  spatial support
    enabling 10
DB2GSE.GEOMETRY_COLUMNS catalog view 77
DB2GSE.SPATIAL_REF_SYS catalog view 77
DE_HDN _SRS_1004
  spatial reference system 20
DEFAULT _SRS
  spatial reference system 20
dimensions 7
disability x
disable_spatial command 243
distance information
  obtaining
    ST_Distance function 133
drop_cs command 244
drop_idx command 245
drop_srs command 245
DSN5SCLP program
  commands 231

E

enable_spatial command 246
enabling

enabling *(continued)*
spatial support 10–12
equal-area projections 17
equidistant projections 17

F

function_level command 247
functions
messages 259
spatial
data exchange format conversions 85

G

GCS_NORTH _AMERICAN _1927
coordinate system 20
GCS_NORTH _AMERICAN _1983
coordinate system 20
GCS_WGS _1984
coordinate system 20
GCSW _DEUTSCHE _HAUPTDREIECKSNETZ
coordinate system 20
geocoders
ST_SIZINGS catalog view 80
geocoding
formulas 23
geographic coordinate system 13
geographic features
description 1, 2
Geographic Markup Language (GML)
data format 219
geometries
boundary 7
converting 23, 105
distance information 108
empty 7
exterior 7
generating
one from many 108
space configurations 105
indexes 109
interior 7
not empty 7
properties
boundary information 103
configuration information 104
coordinate information 100
dimensional information 104
geometries within a geometry 102
measure information 100
spatial reference system 105
spatial data 4
geometry coordinates 6
geometry subtypes
non-simple 7
simple 7
geometry types 6
grid cells 37
grid indexes 35
grid levels 37
grid sizes 37

I

IBM Spatial Support for Db2 for z/OS
getting started 9
installing 9
messages 261
overview 1
setting up 9
IBM Spatial Support for DB2 for z/OS
overview 1
spatial reference systems 20
import_shape command 248
importing
shape data 33
spatial data 4, 33
indexes
spatial grid indexes 35
inline spatial columns
creating 29
installation
verifying 10
installing
IBM Spatial Support for Db2 for z/OS
system requirements 9

L

linear units
coordinate systems 221
supported 224
linestrings
closed 7

M

M coordinates 6
map projections
coordinate systems 221
supported 227
map viewers 9
MBR (minimum bounding rectangle)
definition 6
spatial grid indexes 35
measure information
obtaining 100
measures
maximum
determining 25
minimum
determining 25
messages
functions 259
GSE 261
parts 257
stored procedures 258, 259
minimum bounding rectangle (MBR)
definition 6
spatial grid indexes 35
multilinestrings 5
multipliers
processing coordinates 23
multipoints 5
multipolygons 5

N

NAD27_ SRS_1002
    spatial reference system 20
NAD83_ SRS_1
    spatial reference system 20

O

offset values
    calculating 24
    units 23, 24
operating system requirements
    IBM Spatial Support for Db2 for z/OS 9

P

performance
    coordinate data conversions 23
points 5
polygons
    geometry type 5
prime meridians
    coordinate systems 221
    supported 227
problems
    identifying 257
projected coordinate systems 13, 17

Q

queries
    performing
        spatial functions 41
    spatial indexes 42
    submitting 41

R

register_spatial_column command 253
registering
    spatial columns 30
requirements
    operating system 9
    software 9
    system 9
rings 6

S

scale factors
    calculating 25
    units 23, 24
setting up
    IBM Spatial Support for Db2 for z/OS 9
shape data
    importing 33
shape files 4
shape representation 219
shortcut keys
    keyboard x
software requirements
    IBM Spatial Support for Db2 for z/OS 9
spatial columns
    creating 28
    populating 33
    registering 30
    viewing 27
spatial data
    analyzing
        functions 41
        indexes 42
        interfaces 41
    columns 27
    data types 27
    generating 3
    importing 4, 33
    obtaining 3
    retrieving
        functions 41
        indexes 42
        interfaces 41
    ST_GEOMETRY_ COLUMNS 79
spatial data formats
    supported 213
spatial data types
    multi-unit features 28
    single-unit features 27
spatial extents 18
spatial functions
    comparing geometries
        container relationships 92
        identical geometries 100
        intersections 95
    converting geometries 85
    data exchange format conversions
        ESRI shape representation 89
        Geography Markup Language (GML) representation 89
        overview 85
        well-known binary representation 89
        well-known text representation 88
    data exchange formats 85
    distance information 108
    EnvelopesIntersect 95, 111
    examples 41
    generating geometries
        converting 105
        one from many 108
        space configurations 105
    geometries
        generating 105
    index information 109
    parameters 111
    properties of geometries
        boundary information 103
        configuration information 104
        coordinate information 100
        dimensional information 104
        geometries within a geometry 102
        measure information 100
        spatial reference system 105
    queries 36
    spatial indexes 42
    ST_Area 104, 113
    ST_AsBinary 115
    ST_AsGML 116

spatial functions *(continued)*
ST_AsShape 117
ST_AsText 118
ST_Boundary 119
ST_Buffer 121
ST_Centroid 123
ST_Contains 92, 124
ST_ConvexHull 125
ST_CoordDim 127
ST_Crosses 96, 128
ST_Difference 129
ST_Dimension 131
ST_Disjoint 132
ST_Distance 133
ST_Endpoint 136
ST_Envelope 137
ST_Equals 100, 138
ST_ExteriorRing 140
ST_Geometry 141
ST_GeometryN 142
ST_GeometryType 143
ST_GeomFromText 143
ST_GeomFromWKB 144
ST_GetIndexParms 146
ST_InteriorRingN 146
ST_Intersection 147
ST_Intersects 95, 149
ST_Is3D 101, 150
ST_IsClosed 151
ST_IsEmpty 153
ST_IsMeasured 101, 154
ST_IsRing 155
ST_IsSimple 156
ST_IsValid 101, 157
ST_Length 104, 158
ST_LineFromWKB 159
ST_LineString 160
ST_LocateAlong 161
ST_LocateBetween 163
ST_M 102, 164
ST_MaxM 102, 165
ST_MaxX 102, 166
ST_MaxY 102, 167
ST_MaxZ 102, 169
ST_MinM 102, 170
ST_MinX 102, 171
ST_MinY 102, 172
ST_MinZ 102, 173
ST_MLineFromWKB 174
ST_MPointFromWKB 176
ST_MPolyFromWKB 177
ST_MultiLineString 178
ST_MultiPoint 180
ST_MultiPolygon 181
ST_NumGeometries 183
ST_NumInteriorRing 184
ST_NumPoints 103, 185
ST_Overlaps 97, 186
ST_Perimeter 188
ST_Point 86, 189
ST_PointFromWKB 191
ST_PointN 192
ST_PointOnSurface 193
ST_PolyFromWKB 194
spatial functions *(continued)*
ST_Polygon 105, 195
ST_Relate 197
ST_SRID 105, 198
ST_StartPoint 198
ST_SymDifference 199
ST_Touches 98, 201
ST_Union 202
ST_UnionAggr 108, 204
ST_Within 93, 206
ST_WKBToSQL 207
ST_WKTToSQL 208
ST_X 102, 209
ST_Y 102, 210
ST_Z 102, 210
syntax 111
spatial grid indexes
creating 39
exploiting 42
generating 35
grid levels 35, 37
grid sizes 35, 37
queries 36
spatial indexes 35
spatial information
analyzing 41
generating 41
spatial reference systems
creating 19, 22, 54, 57
DB2-supplied 20
default 19
NAD27_ SRS_1002 20
NAD83_ SRS_1 20
WGS84_ SRS_1003 20
spatial resources
setting up 13
spatial support
enabling 11, 12
messages 261
overview 1
spheroids
coordinate systems 221
supported 225
ST_alter_coordsys stored procedure 45
ST_alter_srs stored procedure 47
ST_Buffer 105
ST_Centroid 102
ST_ConvexHull 106
ST_COORDINATE_ SYSTEMS 78
ST_create_coordsys stored procedure 50
ST_create_index stored procedure 39, 52
ST_create_srs stored procedure 54, 57
ST_Difference 106
ST_Distance 133
ST_drop_coordsys stored procedure 61
ST_drop_index stored procedure 62
ST_drop_srs stored procedure 63
ST_EndPoint 103
ST_export_shape stored procedure 64
ST_GEOMETRY_ COLUMNS 79
ST_GeometryN 103
ST_import_shape stored procedure 67
ST_Intersection 107
ST_IsClosed 104

ST_IsEmpty 104
ST_IsSimple 104
ST_NumGeometries 103
ST_PointN 103
ST_register_spatial_column stored procedure 73
ST_SIZINGS 80
ST_SPATIAL_ REFERENCE_SYSTEMS 81
ST_SRID 105
ST_StartPoint 103
ST_SymDifference 108
ST_Union 108
ST_UNITS_OF_ MEASURE 84
ST_UNITS_OF_ MEASURE catalog view 84
ST_unregister_spatial_column stored procedure 75
stored procedures
    messages 258, 259
    ST_alter_coordsys 45
    ST_alter_srs 47
    ST_create_coordsys 50
    ST_create_index 52
    ST_create_srs 54, 57
    ST_drop_coordsys 61
    ST_drop_index 62
    ST_drop_srs 63
    ST_export_shape 64
    ST_import_shape 67
    ST_register_spatial_column 73
    ST_unregister_spatial_ column 75
syntax diagram
    how to read xi
system requirements
    IBM Spatial Support for Db2 for z/OS 9

T

troubleshooting
    functions 259
    messages 257
    stored procedures 258

U

union aggregate function 108, 204
unregister_spatial_column command 254

V

views 35
visualization tools 27

W

well-known binary (WKB) representation 217
well-known text (WKT) representation 213
WGS84_ SRS_1003
    spatial reference system 20

X

X and Y coordinates 6

Z

Z coordinates 6



Product Number: 5697-Q04

GC19-4069

