

MapInfo *Professional*[®]

for Microsoft[®] SQL Server[™]

Functions Guide



Information in this document is subject to change without notice and does not represent a commitment on the part of the vendor or its representatives. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, without the written permission of MapInfo Corporation, One Global View, Troy, New York 12180-8399.

©1992-2002 MapInfo Corporation. All rights reserved.

MapInfo Help ©1992-2002 MapInfo Corporation. All rights reserved.

MapInfo, MapInfo Professional, MapInfo logo, MapInfo GIS Extension, MapX, MapXsite, MapXtend, and SpatialWare are registered trademarks, and MapInfo MapXtreme, StreetPro, and TargetPro are trademarks of MapInfo Corporation in the United States.

Contact MapInfo Corporation on the Internet at: <http://www.mapinfo.com>.

MapInfo Corporate Headquarters:

Voice: (518) 285-6000

Fax: (518) 285-6060

Sales Info Hotline: (800) 327-8627

US Government Sales: (800) 619-2333

Technical Support Hotline: (518) 285-7283

Technical Support Fax: (518) 285-6080

MapInfo Europe Headquarters:

Voice: +44 (0)1753.848.200

Fax: +44 (0)1753.621.140

email: uk@mapinfo.com

Germany:

Voice: +49 (0)6142-203-400

Fax: +49 (0)6142-203-444

email: germany@mapinfo.com

Toll-free telephone support is available in the U.S. and Canada. Contact your MapInfo sales representative for details. For international customers, please use the Technical Support Fax number.

HyperHelp copyright© Bristol Technology Inc. 1991, 1992, 1993

EHelp® is a registered trademark of Foundation Solutions, Inc., ©copyright, 1992, 1993.

Products named herein may be trademarks of their respective manufactures and are hereby recognized. Trademarked names are used editorially, to the benefit of the trademark owner, with no intent to infringe on the trademark.

This documentation reflects the contributions of almost all of the women and men who work for MapInfo Corporation. It was specifically produced by Ursula Toelke, with the help of Anne Thorne, Michael Berner, Marie Costa, Colleen Cox, Juliette Funicello, Lindsay Guttschall, Ed McElroy, Max Morton, Gayle Patenaude, Dianne Ritter, and Larry Strianese. These members of the Documentation Department are indebted to MapInfo's Quality Assurance Department and, of course, to all the members of the Product Development (Toronto SpatialWare division) team that engineered this project.

MapInfo welcomes your comments and suggestions.

MapInfo Professional for SQL Server

June 2002

Table of Contents

About Functions	3
Conventions	4
Open GIS Consortium (OGC)	5
Filter Tolerance	5
Data Tolerance	5
Aggregate Functions	7
Function Descriptions	8
Cast Functions	13
Function Descriptions	14
Constructor Functions	17
Function Descriptions	18
Overview	18
Constructor Formats	19
The ST_Spatial Constructor Function	19
Geometry String Formats	21
Coordinate Functions	37
Function Descriptions	38
General Functions	41
Function Descriptions	42
Measurement Functions	45
Function Descriptions	46
Observer Functions	61
Function Descriptions	62
Spatial Functions	93
Function Descriptions	94
Transformation Calculation	131

Spatial Predicates 133

 Function Descriptions 134

Using Coordinate Systems 161

 Performing Coordinate Transformations 162

 OGC Well-Known Text 163

Coordinate Transformation Reference Tables 167

 Projections 168

 Spheroids/Ellipsoids 169

 Coordinate Units 170

 Datums 170

About Functions

Spatial functions allow you to analyze and manipulate spatial data. They are grouped into nine categories.

All functions have either an HG_ prefix or ST_ prefix. The ST_ prefix indicates that the function or geometry representation is defined according to the international standard ISO/IEC SQL/MM. The HG_ prefix indicates that the function or geometry representation is an enhancement to the standard.



1 Chapter

- **Conventions**
- **Open GIS Consortium (OGC)**
- **Filter Tolerance**
- **Data Tolerance**



Overview

- **Aggregate Functions**
Work across rows in a group and return a single-row result.
- **Cast Functions**
Transform data from one format to another.
- **Constructor Functions**
Create geometry using the ST_Spatial function (described under Geometry Creation).
- **Coordinate Functions**
Applied to work with coordinate system information.
- **General Functions**
Perform operations, make queries, or change settings.
- **Measurement Functions**
Perform calculations on geometries to find a measurable characteristic, such as length, slope, area, or height.
- **Observer Functions**
Return numbers, objects, or conditions from within a geometry.
- **Spatial Functions**
Perform operations on geometries to create new geometries.
- **Spatial Predicates**
Analyze geometries to see if they meet specific conditions. These functions usually return TRUE or FALSE (1 or 0) values and are generally used within a WHERE clause (e.g. find overlapping geometries).

Conventions

The following formatting cues are used in this document:

- Code samples, filenames, and URLs are set in a **monospaced** font.
- Notes are identified by the word "Note" in bold type.
- Bulleted lists present options and features.
- Numbered steps indicate procedures.
- Toolbutton icons are generally shown with procedure steps.
- Menu levels are separated by the greater than (>) sign.
- Text for you to type in is set in *italics*.

Open GIS Consortium (OGC)

The Open GIS Consortium, Inc. (OGC) defines itself as "a unique membership organization dedicated to the development of open system approaches to geoprocessing." They are developing the OpenGIS® Simple Features Specification For SQL. It is currently at version 1.1. Documentation is available at the Open GIS Consortium web site (<http://www.opengis.org>). SpatialWare for SQL Server follows the OGC Standards. It supports and properly defines geometry and its components.

Filter Tolerance

Filtering is done by setting the third parameter to the ST_Buffer function (if you do not want to use filtering you would set it to null) or by calling the HG_Filter function:

```
ST_Buffer(sw_geometry, double precision, double precision)
HG_Filter(sw_geometry, double precision)
```

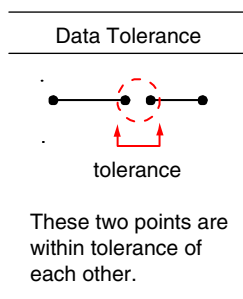
Filtering reduces the number of points that represent a spatial object. If an object is used as input to other functions, like ST_Overlap, reducing its level of detail improves processing time.

Note: Filter tolerance is not the same as data tolerance.

Data Tolerance

SpatialWare functions use a default data tolerance of 1.0 e-10 database units.

Points falling within the tolerance value of one another are considered to occupy the same location, as in the diagram below:



Tolerance applies to all aspects of spatial analysis.

Aggregate Functions

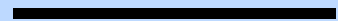
Spatial Aggregate functions work across rows in a group. An aggregate function returns a single-row result of type Spatial Blob. Aggregate functions take as input one or more spatial objects of type ST_Spatial and return a single spatial object.



2

Chapter

► Function Descriptions



Function Descriptions

Aggregate functions include:

HG_Aggspatial
HG_Aggunion
HG_Aggintersection
HG_Aggconvex_Hull
HG_Bounding_Box

HG_Aggspatial

HG_Aggspatial returns a spatial object that contains the geometries from the spatial objects in a group.

Given the following group of spatial values:

- 'point(1,2)'
- 'polyline(list{point(1,2), point(3,2)})'
- 'box(1,2,3,2)'
- 'point(7,5)'

HG_Aggspatial returns:

st_spatial(list{point(1,2), polyline(list{point(1,2), point(3,2)}), box(1,2,3,2), point(7,5)})'

Syntax

HG_Aggspatial(*spatial_obj*)

spatial_obj is an ST_Spatial.

Return Type

HG_Aggunion returns an ST_Spatial.

Example

```
exec sp_spatial_query '  
    select hg_aggunion(flood100.sw_geometry)  
    from flood100;  
,
```

HG_Aggunion

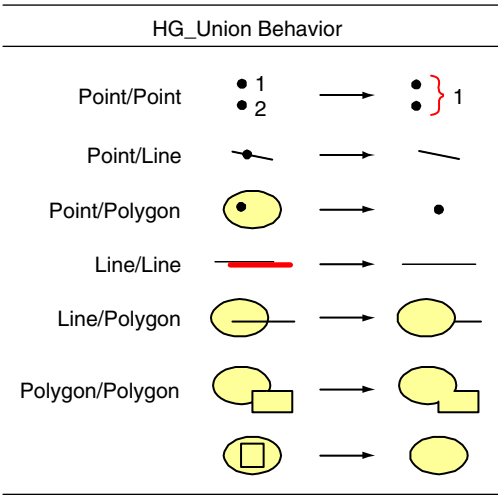
HG_Aggunion returns the union of the spatial objects in the group.

HG_Aggunion returns the union, see also HG_Union on page 120, of each ST_Spatial value in the group. For a group of four ST_Spatial values (a, b, c, d), HG_Aggunion returns the equivalent to:

```
hg_union(hg_union(hg_union(hg_union(a, b), c), d)
```

HG_Union

The following diagram shows the behavior of HG_Union with the basic shape types. The right side of the arrow is the return object. 'Line' represents all Curve objects: ST_CircularArc, ST_Polyline, HG_Curve, and HG_Circle. Collections of objects follow these basic examples.



Syntax

```
HG_Aggunion(spatial_obj)
```

spatial_obj is an ST_Spatial.

Return Type

HG_Aggunion returns an ST_Spatial.

Example

```
exec sp_spatial_query '  
    select hg_aggunion(flood100.sw_geometry)  
    from flood100;  
,
```

HG_Aggintersection

HG_Aggintersection returns the intersection of the spatial objects in the group.

HG_Aggintersection calls HG_Intersection, but is used for groups of ST_Spatial. For a group of four ST_Spatial values (a, b, c, d), HG_Aggintersection returns the equivalent to:

hg_intersection(hg_intersection(hg_intersection(a, b), c), d)

Syntax

HG_Aggintersection(*spatial_obj*)

spatial_obj is an ST_Spatial

Return Type

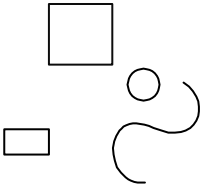
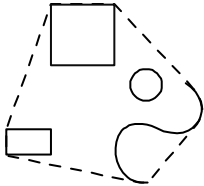
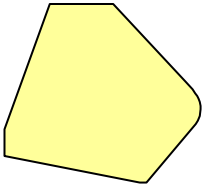
HG_Aggintersection returns an ST_Spatial. If there is no intersection of the spatial objects in this group, an empty geometry is returned.

Example

```
exec sp_spatial_query '  
    select hg_aggintersection(drain.sw_geometry)  
    from drain;  
,
```

HG_Aggconvex_Hull

HG_Aggconvex_Hull returns the convex hull for the group of spatial objects.

Input	Convex Hull Operation	Output
		

Syntax

HG_Aggconvex_Hull(*spatial_obj*)

spatial_obj is an ST_Spatial.

Return Type

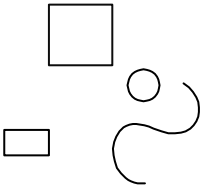
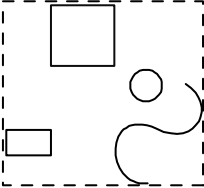
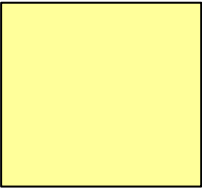
HG_Aggconvex_Hull returns an ST_Spatial.

Example

```
exec sp_spatial_query '  
    select hg_aggconvex_hull(pubbldg.sw_geometry)  
    from pubbldg;  
,
```

HG_Bounding_Box

HG_Bounding_Box returns the bounding box for the group of spatial objects. This is the minimum enclosing rectangle (MER) around all ST_Spatial values in a group.

Input	Convex Hull Operation	Output
		

Syntax

HG_Bounding_Box(*spatial_obj*)

spatial_obj is an ST_Spatial.

Return Type

HG_Bounding_Box returns an ST_Spatial.

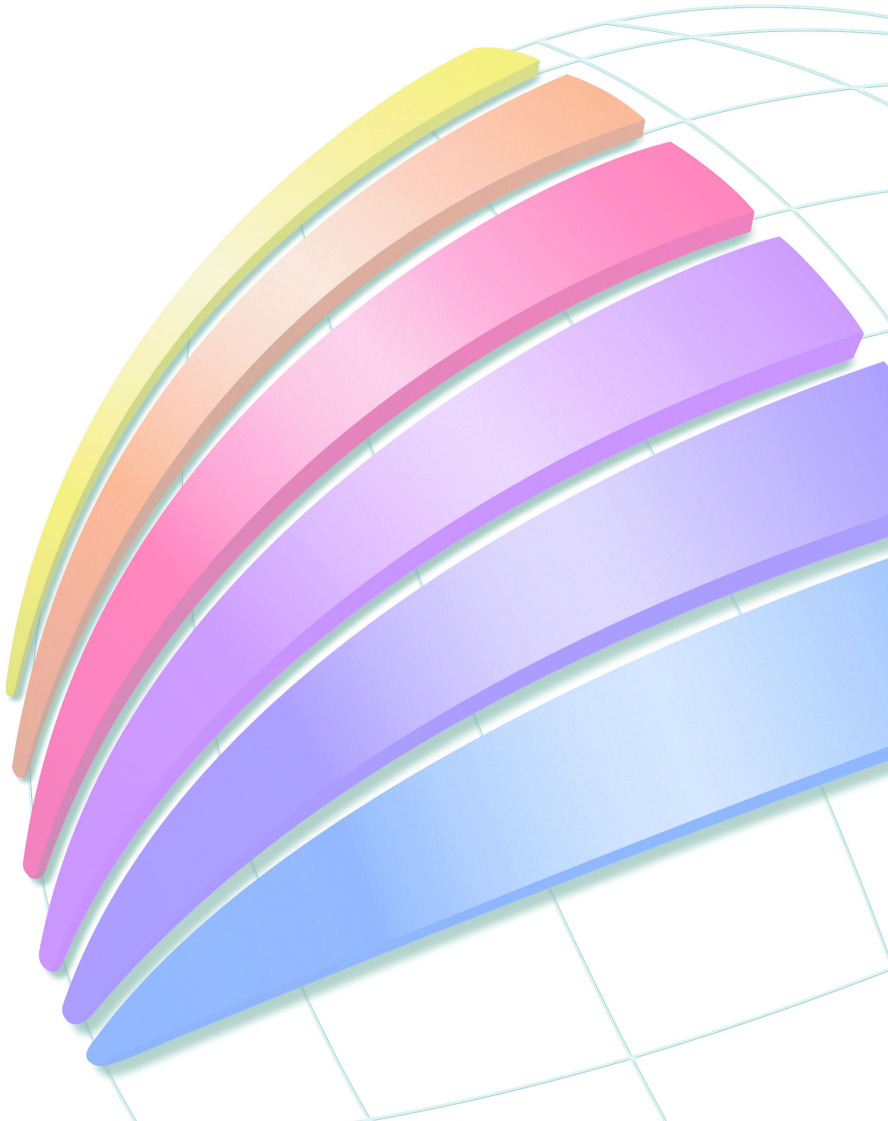
Example

```
exec sp_spatial_query '  
    select hg_bounding_box(pubbldg.sw_geometry)  
    from pubbldg;  
,
```

Cast Functions

Cast functions transform data from one format to another.

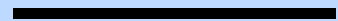
These functions follow the OGC standards, please see Open GIS Consortium (OGC) on page 5.



3

Chapter

► Function Descriptions



Function Descriptions

Cast functions include:

- HG_AsBinary
- HG_AsText
- HG_GeometryFromBinary
- HG_GeometryFromText

HG_AsBinary

The HG_AsBinary function generates an OGC well-known binary from SpatialWare geometry.

Syntax

HG_AsBinary(*spatial_obj*)
spatial_obj is an ST_Spatial.

Return Type

HG_AsBinary returns an OGC well-known binary.

Example

The following example generates a temporary table called tmpbinary. This table contains the OGC well-known binary generated by HG_AsBinary from SpatialWare geometry.

```
exec sp_spatial_query '  
    select HG_AsBinary(sw_geometry) as ogcbinary into tmpbinary  
    from parcel  
'
```

HG_AsText

The HG_AsText function generates OGC well-known text from SpatialWare geometry.

Syntax

HG_AsText(*spatial_obj*)
spatial_obj is an ST_Spatial.

Return

HG_AsText returns an OGC well-known text.

Example

The following example generates a temporary table called tmptxt. This table contains the OGC well-known text generated by HG_AsText from SpatialWare geometry.

```
exec sp_spatial_query '  
    select HG_AsText(sw_geometry) as ogctxt into tmptxt  
    from parcel  
,
```

HG_GeometryFromBinary

The HG_GeometryFromBinary function interprets an OGC well-known binary and creates a SpatialWare geometry from it.

Syntax

HG_GeometryFromBinary(*ogcbinary*)

ogcbinary is an OGC well-known binary.

Return Type

HG_GeometryFromBinary returns an ST_Spatial.

Example

The following example queries a temporary table called tmpbinary. This table contains OGC well-known binaries generated by HG_AsBinary from SpatialWare geometry. Please refer to the description for HG_AsBinary.

```
exec sp_spatial_query '  
    select HG_GetString(HG_GeometryFromBinary(ogcbinary))  
    from tmpbinary  
,
```

HG_GeometryFromText

The HG_GeometryFromText function interprets an OGC well-known text string and creates a SpatialWare geometry from it.

Syntax

`HG_GeometryFromText(ogctext)`

ogctext is an OGC well-known text string passed from a table.

Return Type

`HG_GeometryFromText` returns an `ST_Spatial`.

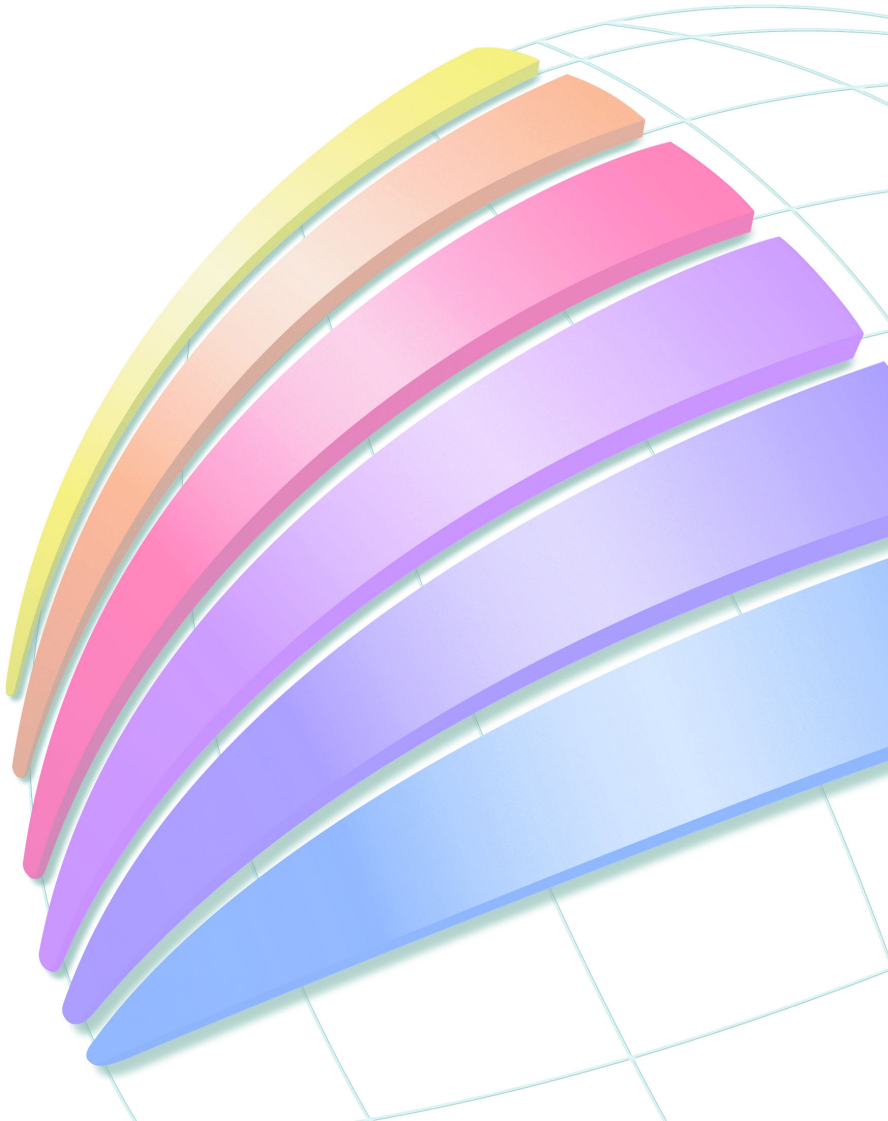
Example

The following example queries a temporary table called `tmptxt`. This table contains OGC well-known text generated by `HG_AsText` from SpatialWare geometry. Please refer to the description for `HG_AsText`.

```
exec sp_spatial_query '  
    select HG_GetString(HG_GeometryFromText(ogctxt))  
    from tmptxt  
'
```

Constructor Functions

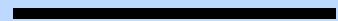
This chapter describes how to construct geometry using SpatialWare's Constructor functions and predefined string formats. The resulting geometry is in the expected format used by SpatialWare's functions.



4

Chapter

- Overview
- Constructor Formats
- The ST_Spatial Constructor Function
- Geometry String Formats



Function Descriptions

Geometry collections, a geometry comprised of multiple geometry strings, are also possible. They can be created using the ST_Spatial function.

Geometry String Formats

HG_Box
HG_Circle – Two Points
HG_Circle – Center Point and Radius
HG_Circle – x, y, and Radius Values
HG_Curve
HG_Triangle
HG_Quad
ST_CircularArc
ST_Path
ST_Point
ST_Polygon – Path of Lines
ST_Polygon – Centroid
ST_Polygon – Interior Boundaries
ST_Polygon – Interior Boundaries and Centroid
ST_Polyline

Constructor Formats

ST_Spatial

Overview

Constructor functions provide a simple interface through which you can create basic geometries, such as points, polygons, and circles. All constructor functions return SW_GEOMETRY.

There are geometry-specific constructor functions such as HG_Circle that create basic geometries, and a more general constructor function, ST_Spatial, to create more complex geometries.

You will notice that some constructor functions have the same name as the geometry string format that they create: HG_Circle, ST_Point, and ST_Polyline (for a description of these three geometry string formats (refer to The ST_Spatial Constructor Function on page 19). These constructor functions are provided as a shortcut for creating some geometry types. You could alternately create these geometry strings using the ST_Spatial constructor function.

Examples use the spatial table called testcon. Refer to the Concepts section for information on data definition for a description of how to create a spatial table.

Dimension

Geometry is three-dimensional if it uses the three-dimensional point construction. For example:

```
ST_Spatial('ST_Polyline(LIST{
    ST_Point(1,3,5),
    ST_Point(10,6,9)
})')
```

Constructor Formats

Geometry-specific constructor functions provide a simplified interface through which you can create basic geometries. Geometry Constructors include HG_Box, HG_Box_Points, HG_Box_WH, HG_Circle, ST_Point, and ST_Polyline.

Constructor functions are shortcuts for creating geometry. Using the specific constructors can cut down on errors and typing. Whether you use the ST_Spatial constructor or one of the constructor functions, the internal representation will be the same.

The following is an example of the HG_Circle Constructor function. This example creates a circle from x and y, and radius values:

```
insert into testcon(sw_member, sw_geometry) values(1,
    HG_Circle(1.1, 2.2, 5.0)
);
```

The sw_member column, in the above example, holds a unique integer key that is used for R-tree indexing.

The ST_Spatial Constructor Function

The ST_Spatial Constructor function uses geometry string formats to generate geometry. The following example uses the ST_Spatial constructor and the predefined string format for circle to insert a circle geometry into a table called testcon:

```
insert into testcon(sw_member, sw_geometry) values(2,
    'ST_Spatial(
        HG_Circle(ST_Point(11.1,22.2),ST_Point(55.5,44.4))
    )'
);
```

ST_Spatial is very versatile because more than one string format can be used to create a multi-element geometry, such as a square with a square hole in it. The following example builds a more complex geometry by using multiple string formats. For example, a geometry containing a point and a circle:

```
insert into testcon(sw_member, sw_geometry) values(3,
  'ST_Spatial(LIST{
    ST_Point(11.1,44.4),
    HG_Circle(ST_Point(11.1,22.2),ST_Point(55.55,44.4))
  })'
)
```

Refer to Geometry String Formats on page 21 for the string formats used with ST_Spatial.

ST_Spatial

The ST_Spatial function takes a textual description of the geometry and constructs a SW_GEOMETRY from it. The textual description are in SpatialWare-SQL/MM format.

ST_Spatial generates geometry consisting of one or more elements. For a description of the string formats, for example HG_Box and HG_CircularArc, refer to The ST_Spatial Constructor Function on page 19.

Syntax

ST_Spatial(*string*)

string is a character string representing a geometry (refer to The ST_Spatial Constructor Function on page 19.).

Return Type

ST_Spatial returns SW_GEOMETRY.

Example

The following example calls HG_Center_In, which returns a copy of an geometry, where all polygons are replaced by the line-strings forming its boundary. The ST_Spatial constructor is used to create a point geometry for input the HG_Center_In function.

```
exec sp_spatial_query '
select HG_GetString(HG_Center_In(
  flood100.sw_geometry, ST_Spatial(''
    ST_Point(1753960.182000, 10698421.103000)
  ''
))
from flood100
'
```

Note: HG_GetString has been added to the example so that it will return geometry in text format.

Geometry String Formats

This section describes predefined string formats available for creating geometry with the ST_Spatial constructor function.

HG_Box

HG_Box is the string format of a four sided object. It takes four double precision numbers: x1, y1, x2, and y2 as input. These parameters specify the lower left (x1, y1) and upper right corners of a box (x2, y2).

String Format

`'HG_Box(x1, y1, x2, y2)'`

x1, y1 are double precision numbers representing the x and y values of a coordinate.

x2, y2 are double precision numbers representing the x and y values of a coordinate.

Constructor

The HG_Box syntax of the ST_Spatial constructor is:

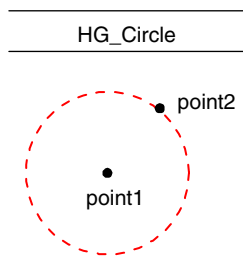
```
ST_Spatial('HG_Box(x1, y1, x2, y2)')
```

Example

```
insert into testcon(sw_member, sw_geometry) values(
    20, 'ST_Spatial(HG_Box(1.1,2.2,3.3,4.4))'
)
```

HG_Circle – Two Points

HG_Circle is the string format of a circle. This syntax (syntax two of three) defines a circle by specifying a center point and a point that lies on the circle, in that order.



HG_Circle generates a linear geometry – a circular path.

String Format

'HG_Circle(LIST{*point1*, *point2*})'

point1 is the string format of an ST_Point.

point2 is the string format of an ST_Point.

Constructor

The HG_Circle syntax of the ST_Spatial constructor is:

```
ST_Spatial('HG_Circle(LIST{point, point})')
```

Example

```
insert into testcon(sw_member, sw_geometry) values(  
    21,  
    'ST_Spatial(HG_Circle(LIST{  
        ST_Point(11.11,22.22),  
        ST_Point(55.55,44.44)  
    }))'  
)
```

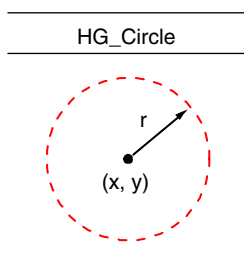
Caveat

To generate a circular area, instead of a circular path, buffer around a point or create a polygon from a circle. For example:

```
ST_Buffer(x,y,r)  
ST_Spatial('ST_Polygon(HG_Circle(point,r))')
```

HG_Circle – Center Point and Radius

HG_Circle is the string format of a circle. This syntax (syntax one of three) defines a circle by specifying a center point and radius value (x, y, and radius values).



HG_Circle generates a linear geometry – a circular path.

String Format

'HG_Circle(*point*, *radius*)'

point is the string format of an ST_Point.

radius is a double precision value.

Constructor

The HG_Circle syntax of the ST_Spatial constructor is:

```
ST_Spatial('HG_Circle(point, radius)')
```

Example

```
insert into testcon(sw_member, sw_geometry) values(
    22, 'ST_Spatial(HG_Circle(ST_Point(11.11,22.22), 10))'
)
```

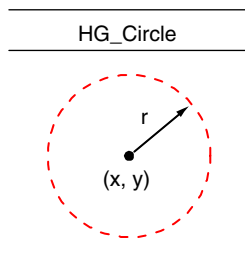
Caveat

To generate a circular area, instead of a circular path, buffer around a point or create a polygon from a circle. For example:

```
ST_Buffer(x,y,r)
ST_Spatial('ST_Polygon(HG_Circle(point,r))')
```

HG_Circle – x, y, and Radius Values

HG_Circle is the string format of a circle. This syntax (syntax three of three) defines a circle by specifying the x and y ordinates of the circle centre point, and a radius value.



HG_Circle generates a linear geometry – a circular path.

String Format

`'HG_Circle(x, y, radius)'`

x, y, are double precision values.

radius is a double precision value.

Constructor

The HG_Circle syntax of the ST_Spatial constructor is:

```
ST_Spatial('HG_Circle(x, y, radius)')
```

Example

```
insert into testcon(sw_member, sw_geometry) values(  
    23, 'ST_Spatial(HG_Circle(11.11, 22.22, 10))'  
)
```

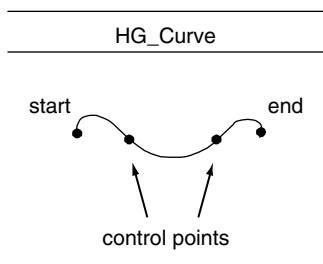
Caveat

To generate a circular area, instead of a circular path, buffer around a point or create a polygon from a circle. For example:

```
ST_Buffer(x,y,r)  
ST_Spatial('ST_Polygon(HG_Circle(point,r))')
```

HG_Curve

HG_Curve is the string format of a curve. It is a smoothly curving line defined by a list of points: a start point and an end point, with zero or more control points in between.

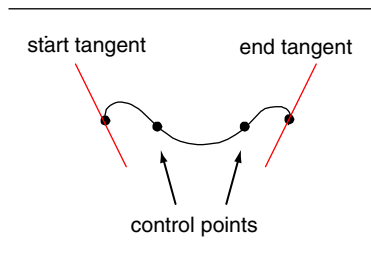


Optional start-tangent and end-tangent parameters give more control over the shape of the line. As a result, HG_Curve can be defined in four ways:

1. A list of ST_Points.
2. A start-tangent and a list of ST_Points.

3. A list of ST_Points and an end-tangent.
4. A start-tangent, a list of ST_Points, and an end-tangent.

By setting the start and end tangents, you have more control over the shape of the curve. Tangents are entered as double precision numbers representing a degree.



For brevity, only the fourth case will be detailed below. Just note that the start-tangent and end-tangent parameters are optional. Default values will be assigned if they are left out.

String Format

`'HG_Curve(start_tan, LIST{point, point, point,...}, end_tan)'`

start_tan is a double precision number representing degrees. (This parameter is optional.)

LIST is an internal list mechanism.

point is the string format of an ST_Point.

end_tan is a double precision number representing degrees. (This parameter is optional.)

Constructor

The HG_Curve syntax of the ST_Spatial constructor is:

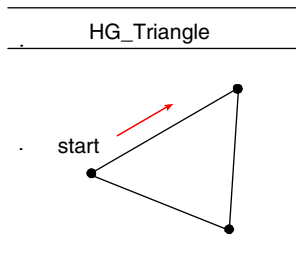
```
ST_Spatial('HG_Curve(start_tan, LIST{point,point,point},
end_tan)')
```

Example

```
insert into testcon(sw_member, sw_geometry) values(
24,
'ST_Spatial(HG_Curve(55.0, LIST{
ST_Point(11.11,22.22),
ST_Point(33.33,44.44),
ST_Point(55.55,66.66)
}, 40.0))')
)
```

HG_Triangle

HG_Triangle is a triangular polygon, which is defined by three points.



String Format

'HG_Triangle(LIST{*point1*, *point2*, *point3*})'

point is the string format of ST_Point.

Constructor

The HG_Triangle syntax of the ST_Spatial constructor is:

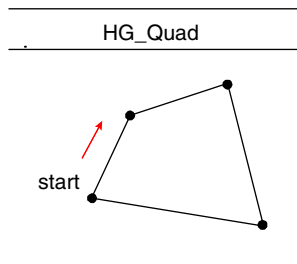
```
ST_Spatial('HG_Triangle(LIST{point, point, point})')
```

Example

```
insert into testcon(sw_member, sw_geometry) values(  
    25,  
    'ST_Spatial(HG_Triangle(LIST{  
        ST_Point(11.22, 33.44),  
        ST_Point(55.66, 77.88),  
        ST_Point(99.1010, 1111.1212),  
    })))'  
)
```

HG_Quad

HG_Quad is a four-sided polygon of any shape, size, or rotation, which is defined by four points.



String Format

'HG_Quad(LIST{point1, point2, point3, point4})'

point is the string format of ST_Point.

Constructor

The HG_Quad syntax of the ST_Spatial constructor is:

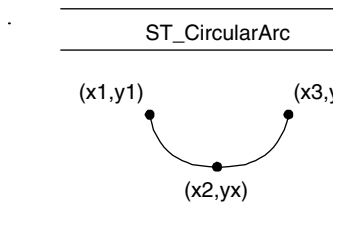
```
ST_Spatial('HG_Quad(LIST{point, point, point, point})')
```

Example

```
insert into testcon(sw_member, sw_geometry) values(
  26,
  'ST_Spatial(HG_Quad(LIST{
    ST_Point(11.22, 33.44),
    ST_Point(55.66, 77.88),
    ST_Point(99.1010, 1111.1212),
    ST_Point(1313.1414, 1515.1616)
  })))'
```

ST_CircularArc

ST_CircularArc is the string format of a curved line defined by three points. The first and last points define the beginning and end, and the middle point determines the shape of the arc.



String Format

'ST_CircularArc(LIST{point1, point2, point3})'

point is the string format of ST_Point.

Constructor

The ST_CircularArc syntax of the ST_Spatial constructor is:

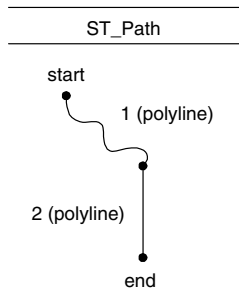
```
ST_Spatial('ST_CircularArc(LIST{point, point, point})')
```

Example

```
insert into testcon(sw_member, sw_geometry) values(27,
  'ST_Spatial(ST_CircularArc(LIST{
    ST_Point(11.11,22.22),
    ST_Point(33.33,44.44),
    ST_Point(55.55,66.66)
  })))'
```

ST_Path

ST_Path is the string format defining a compound line made up of one or more curves. In this case, a curve refers to a family of geometries: ST_Polyline, ST_CircularArc, HG_Circle, or HG_Curve.



String Format

'ST_Path(LIST{*object*, *object*,...})'

LIST is an internal list mechanism.

object is the string format of any curve (ST_Polyline, ST_CircularArc, HG_Circle, or HG_Curve).

Constructor

The ST_Path constructor is:

```
ST_Spatial('ST_Path(LIST{curve, curve})')
```

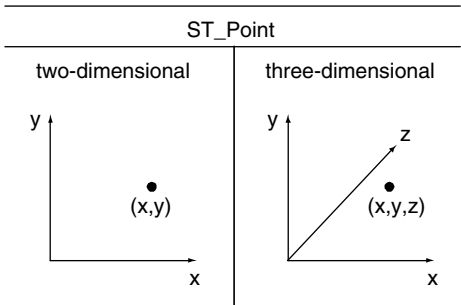
Example

```
insert into testcon(sw_member, sw_geometry) values(
28,
'ST_Spatial(ST_Path(LIST{
  ST_Polyline(LIST{
    ST_Point(11.11,22.22),ST_Point(33.33,44.44)
  }),
  ST_CircularArc(LIST{
    ST_Point(11.11,22.22),
    ST_Point(33.33,44.44),
    ST_Point(55.55,66.66)
  })
})
```

```
    } ) ) '  
)
```

ST_Point

ST_Point is the string format of a point defined by its x, y, and optional z coordinates. The optional z coordinate is used for points located in three-dimensional space. The coordinates are double precision values.



String Format

'ST_Point(x, y)'

The string format of ST_Point in three dimensions is:

'ST_Point(x, y, z)'

x is a double precision number representing the point's x coordinate.

y is a double precision number representing the point's y coordinate.

z is a double precision number representing the point's z coordinate.

Constructor

The ST_Point syntax of the ST_Spatial constructor is:

```
ST_Spatial('ST_Point(x, y)')  
ST_Spatial('ST_Point(x, y, z)')
```

Examples

The following example creates a two-dimensional point.

```
insert into testcon(sw_member, sw_geometry) values(  
29,
```

```
'ST_Spatial(ST_Point(11.1,22.2))'
```

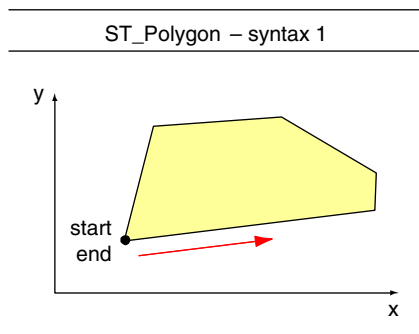
The following example creates a three-dimensional point.

```
insert into testcon(sw_member, sw_geometry) values(
  30,
  'ST_Spatial(ST_Point(11.1,22.2,44.4))'
```

ST_Polygon – Path of Lines

A polygon is the string format of a closed plane figure consisting of three or more straight sides that connect three or more points. None of the sides are intersecting.

This syntax (syntax one of four) of an ST_Polygon defines an ST_Polygon with a path of lines that creates an exterior boundary. The start and end points are the same.



String Format

'ST_Polygon(*path*)'

path is the string format of ST_Path.

Constructor

The ST_Spatial constructor for an ST_Polygon with an exterior boundary is:

```
ST_Spatial('ST_Polygon(path)')
```

Example

The following statement inserts an ST_Polygon into a table called testcon:

```
insert into testcon(sw_member, sw_geometry) values(
  31,
  'ST_Spatial(ST_Polygon(
```



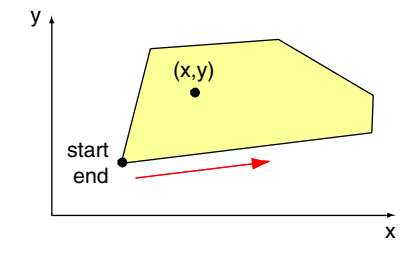
```
ST_Path(LIST{
  ST_Point(1.2,1.0),
  ST_Point(6.4,1.6),
  ST_Point(6.4,2.9),
  ST_Point(5.0,4.2),
  ST_Point(2.1,4.0),
  ST_Point(1.2,1.0)
})
)) '
)
```

ST_Polygon – Centroid

A polygon is the string format of a closed plane figure consisting of three or more straight sides that connect three or more points. None of the sides are intersecting.

This syntax defines a polygon with a centroid. The centroid is the geometric center point of the polygon.

ST_Polygon – syntax 2



String Format

`'ST_Polygon(path, point)'`

path is the string format of ST_Path.

point is the string format of ST_Point.

Constructor

The ST_Spatial constructor for an ST_Polygon with an exterior boundary is:

```
ST_Spatial('ST_Polygon(path, point)')
```

Example

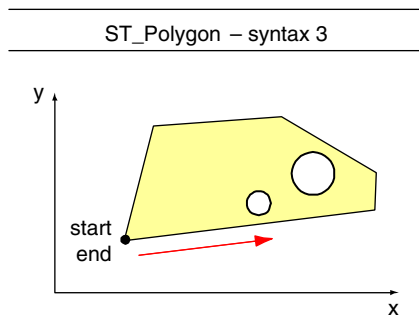
The following statement inserts an ST_Polygon with a centroid into a table called testcon:

```
insert into testcon(sw_member, sw_geometry) values(
  32,
  'ST_Spatial(ST_Polygon(
    ST_Path(LIST{
      ST_Point(1.2,1.0),
      ST_Point(6.4,1.6),
      ST_Point(6.4,2.9),
      ST_Point(5.0,4.2),
      ST_Point(2.1,4.0),
      ST_Point(1.2,1.0)
    }),
    ST_Point(3.7,2.6)
  ))'
```

ST_Polygon – Interior Boundaries

A polygon is the string format of a closed plane figure consisting of three or more straight sides that connect three or more points. None of the sides are intersecting.

This syntax (syntax three of four) defines a polygon with an exterior boundary (a path) and a list of interior boundaries (geometries). The resulting geometry is what is shaded in the below example.



String Format

'ST_Polygon(*path*, LIST{*object1*, *object2* ,...})'

path is the string format of ST_Path.

object1, object2,... are geometries defining interior boundaries.

Constructor

The ST_Spatial constructor for an ST_Polygon with an exterior boundary and one or more interior boundaries is:

```
ST_Spatial('ST_Polygon(path, LIST{object,object})')
```

Example

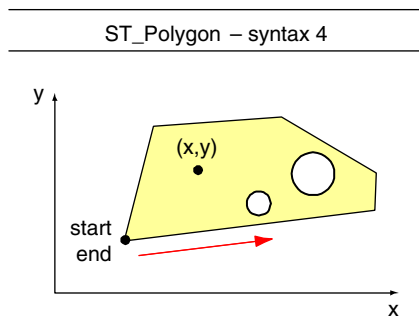
The following statement inserts an ST_Polygon with an exterior boundary and two interior boundaries into a table called testcon:

```
insert into testcon(sw_member, sw_geometry) values (
33,
'ST_Spatial(ST_Polygon(
  ST_Path(LIST{
    ST_Point(1.2,1.0),
    ST_Point(6.4,1.6),
    ST_Point(6.4,2.9),
    ST_Point(5.0,4.2),
    ST_Point(2.1,4.0),
    ST_Point(1.2,1.0)
  }),
  LIST{
    HG_Circle(ST_Point(3.0,1.9),ST_Point(3.2,1.9)),
    HG_Circle(ST_Point(4.6,3.7),ST_Point(5.0,3.7))
  }
))'
)
```

ST_Polygon – Interior Boundaries and Centroid

A polygon is the string format of a closed plane figure consisting of three or more straight sides that connect three or more points. None of the sides are intersecting.

This syntax (syntax four of four) defines a polygon using all possible elements: an exterior boundary (path), a list of interior boundaries (geometries), and a centroid.



String Format

'ST_Polygon(*path*, LIST{*object1*, *object2*, ...}, *point*)'

path is the string format of ST_Path.

object1, *object2*,... are geometries defining interior boundaries.

point is the string format of ST_Point.

Constructor

The ST_Spatial constructor for a polygon with an exterior boundary and one or more interior boundaries is:

```
ST_Spatial('ST_Polygon(path, LIST{object,object}, point)')
```

Example

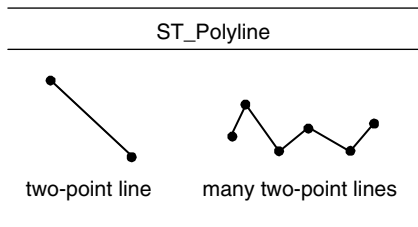
The following statement inserts an ST_Polygon with an exterior boundary, two interior boundaries, and a centroid into the table testcon:

```
insert into testcon(sw_member, sw_geometry) values(
34,
'ST_Spatial(ST_Polygon(
  ST_Path(LIST{
    ST_Point(1.2,1.0),
    ST_Point(6.4,1.6),
    ST_Point(6.4,2.9),
    ST_Point(5.0,4.2),
    ST_Point(2.1,4.0),
    ST_Point(1.2,1.0)}
),
  LIST{
    HG_Circle(ST_Point(3.0,1.9),ST_Point(3.2,1.9)),
    HG_Circle(ST_Point(4.6,3.7),ST_Point(5.0,3.7))
  }
)')
```

```
    },  
    ST_Point(3.7,2.6)  
  )) '  
)
```

ST_Polyline

ST_Polyline is the string format of defining one or more two-point line segments positioned end to end. The line can be of any shape.



String Format

'ST_Polyline(LIST{*point1*, *point2*,...})'

point1, *point2*,... are string formats of ST_Point.

Constructor

The ST_Polyline constructor takes a list as an argument.

```
ST_Spatial('ST_Polyline(LIST{point, point})')
```

Example

The following example creates a two-dimensional polyline.

```
insert into testcon(sw_member, sw_geometry) values(  
  35,  
  'ST_Spatial(ST_Polyline(LIST{  
    ST_Point(11.11,22.22),  
    ST_Point(33.33,44.44)  
  }))) '  
)
```

Coordinate Functions

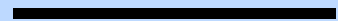
Coordinate functions are applied to transform between coordinate systems.



5

Chapter

► Function Descriptions



Function Descriptions

For information about how SpatialWare handles coordinate systems, please refer to the chapter on coordinate systems.

Coordinate functions include:

HG_CSTransform

HG_CSTransform

HG_CSTransform is a coordinate transformation function. Given a geometry, the description of the coordinate system that it is in, and the target coordinate system, HG_CSTransform performs a transformation. The source and destination coordinate systems are described by the OGC well-known text strings.

The master..HG_SpatialRef table (located in the Master database) contains the well-known text strings for supported coordinate systems.

Coordinate system information is not stored in the system. You must know what coordinate system your data is stored in when working with coordinate systems.

For more information, refer to the chapter on Coordinate Systems.

Syntax

`HG_CSTransform(spatial_obj, from_cs, to_cs)`

spatial_obj is the spatial object being transformed, an ST_Spatial.

from_cs is the coordinate system to transform from a string.

to_cs is the coordinate system to transfer to a string.

Return Type

HG_CSTransform returns an ST_Spatial.

Example

```
exec sp_spatial_query '
    select HG_GetString(HG_CSTransform(
        a.sw_geometry, b.srtext, c.srtext
    ))
```

```
from world a, master..hg_spatialref b,  
    master..hg_spatialref c  
where b.cs_name = ''Longitude / Latitude''  
and c.cs_name = ''Robinson''  
,
```

Coordinate systems are specified by the OGC standard well-known text representation. The well-known text strings for the coordinate systems are available in the table master..HG_SpatialRef (in the Master database). This example assumes the existence of a table called spotelev with geometries in 'Texas 4203, Central Zone (1983, US Survey feet)'.

General Functions

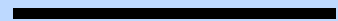
General functions perform operations, make queries, or change settings.



6

Chapter

► Function Descriptions



Function Descriptions

General functions include:

HG_GetString	HG_Version
HG_Morph_Out	

HG_GetString

HG_GetString returns the text representation of a geometry.

A select statement on a column of type ST_Spatial returns a non-text representation of a geometry; HG_GetString is used in a select statement to return the text representation of a selected ST_Spatial value. An ST_Spatial type is the internal, non-text representation of a spatial object.

Syntax

HG_GetString(*spatial_obj*)

spatial_obj is an ST_Spatial.

Return Type

HG_GetString returns a text string.

Examples

This example returns all geometries in the lake table.

```
exec sp_spatial_query '  
    select HG_GetString(sw_geometry)  
    from lake  
,
```

This example returns buffers for every geometry in the lake table.

```
exec sp_spatial_query '  
    select sw_member, HG_GetString(  
        ST_Buffer(sw_geometry, 66.0, 1.0)  
    )  
    from lake  
,
```

HG_Morph_Out

HG_Morph_Out is used to transfer data from a SpatialWare server to a MapInfo client, in MapInfo binary format.

Syntax

HG_Morph_Out(*spatial_obj*)

spatial_obj is any ST_Spatial.

Return Type

HG_Morph_Out returns the input ST_Spatial transformed to MapInfo format.

Example

This example returns all the geometries in the lake table in MapInfo binary format.

```
exec sp_spatial_query '  
    select HG_Morph_Out(sw_geometry)  
    from lake  
,
```

HG_Version

HG_Version returns the current version of SpatialWare as a text string (i.e., '4.5').

Syntax

HG_Version()

No input is required.

Return Type

HG_Version returns a text string.

Examples

```
exec sp_spatial_query '  
    select HG_Version() from mapinfo.mapinfo_mapcatalog  
,
```

Measurement Functions

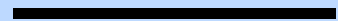
Measurement functions perform calculations on geometries to find a measurable characteristic, such as length, slope, area, or height.



7

Chapter

► Function Descriptions



Function Descriptions

Measurement functions include:

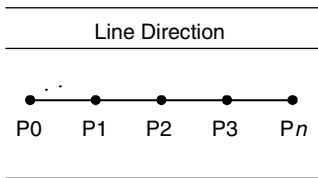
HG_Azimuth	HG_Slope_Min
HG_Azimuth_2pts	HG_SphericalDist
HG_Distance	HG_Width
HG_Height	ST_Area
HG_Separation	ST_Length
HG_Slope	ST_Length_3D
HG_Slope_2pts	ST_Perimeter
HG_Slope_Avg	ST_Perimeter_3D
HG_Slope_Max	

HG_Azimuth

HG_Azimuth returns the compass angle between the end and beginning points of a line. HG_Azimuth finds the beginning and end points of the input line, performs an HG_Azimuth_2pts operation on the two points (see page 47), and returns the result. The line direction attribute determines which point is represented as beginning or end.

Line Direction Attribute

In addition to the global attributes, ST_Line contains a direction attribute that specifies the directional property of the line. Line directions are FORWARD, REVERSE, or NULL. The Direction of a line determines which points are considered the beginning and end points. Line direction is relevant for many functions in SpatialWare, such as HG_Slope. A line has 0 to *n* points.



When a line’s direction is FORWARD or NULL, the beginning point is P0 and the end point is Pn. When a line is REVERSE, the beginning point is Pn and the end point is P0.

Syntax

HG_Azimuth(*spatial_obj*)

spatial_obj is an ST_Spatial representing an ST_Polyline.

Return Type

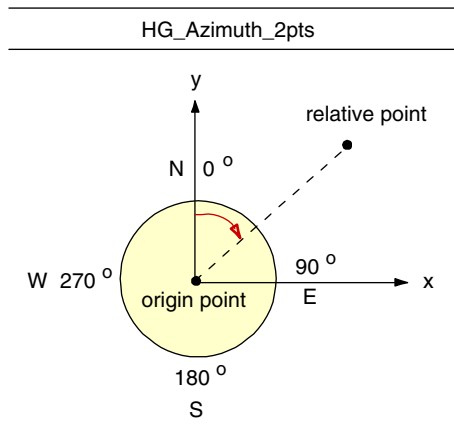
HG_Azimuth returns a double precision number representing degrees.

Example

```
exec sp_spatial_query '
    select HG_Azimuth(rdpaved.sw_geometry)
    from rdpaved;
'
```

HG_Azimuth_2pts

HG_Azimuth_2pts returns the compass angle of the second point, relative to the first. The first parameter becomes the *origin point* and the second parameter, the *relative point*. The *origin point* is always represented as zero degrees North.



The direction of the angle calculation is clockwise from 0 to 360 degrees.

Syntax

HG_Azimuth_2pts(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 must be an ST_Spatial point (the origin).

spatial_obj2 must be an ST_Spatial point (the relative point).

Return Type

HG_Azimuth_2pts returns a double precision number representing degrees or radians.

Example

```
exec sp_spatial_query '
    select HG_Azimuth_2pts(
        spotelev.sw_geometry,
        ST_Spatial(''
            ST_Point(1753960.182000, 10698421.103000,2784.945100)
        '')
    ) from spotelev
,
```

HG_Distance

HG_Distance returns the distance between two geometries in a two-dimensional plane.

Syntax

HG_Distance(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is an ST_Spatial.

spatial_obj2 is an ST_Spatial.

Return Type

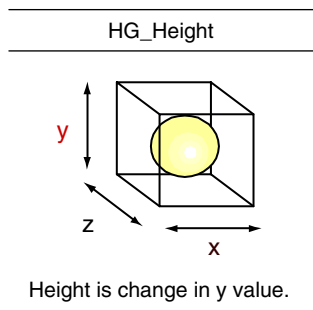
HG_Distance returns a double precision number.

Example

```
exec sp_spatial_query '
    select HG_Distance(
        sw_geometry,
        ST_Spatial(''ST_Point(1753960, 10698421)''))
    from spotelev
,
```

HG_Height

HG_Height returns the height of any ST_Spatial geometry. Height is measured as the change in the y-value of the object's minimum enclosing box. The object can be either two or three dimensional.



Syntax

`HG_Height(spatial_obj)`

spatial_obj is an ST_Spatial.

Return Type

HG_Height returns a double precision number.

Example

```
exec sp_spatial_query '  
    select HG_Height(sw_geometry) from lake  
,
```

HG_Separation

HG_Separation returns the distance between two points in a three-dimensional space.

Syntax

`HG_Separation(spatial_obj1, spatial_obj2)`

spatial_obj1 must be an ST_Spatial representing an ST_Point.

spatial_obj2 must be an ST_Spatial representing an ST_Point.

Return Type

HG_Separation returns a double precision number.

Null is returned if the points are specified as two-dimensional.

Example

```
exec sp_spatial_query '  
    select HG_Separation(a.sw_geometry, b.sw_geometry)  
    from spotelev a, spotelev b  
,
```

HG_Slope

HG_Slope returns the slope of a three-dimensional line by finding the end points and calculating the slope. Points falling between the end points are ignored.

Syntax

HG_Slope(*spatial_obj*)

spatial_obj is an ST_Spatial representing a three-dimensional ST_Polyline.

Return Type

HG_Slope returns an angle in degrees as a double precision number.

NULL is returned if a two-dimensional line is given as a parameter.

Usage

Use this function during a three-dimensional session.

Example

```
exec sp_spatial_query '  
    select HG_Slope(sw_geometry) from testcon  
,
```

HG_Slope_2pts

HG_Slope_2pts returns the slope between two points. Slope is directional, so the order of the parameters is important. The first parameter is the beginning point and the second parameter is the end point.

Syntax

HG_Slope_2pts(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is an ST_Spatial representing an ST_Point (the beginning point).

spatial_obj2 is an ST_Spatial representing an ST_Point (the end point).

Return Type

HG_Slope_2pts returns an angle in degrees as a double precision number.

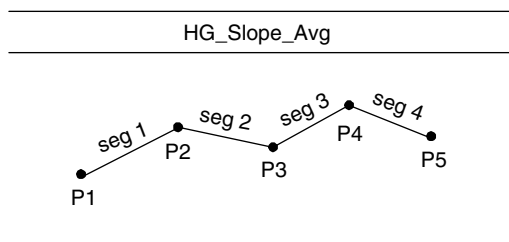
Example

```
exec sp_spatial_query '
select HG_Slope_2pts(
    sw_geometry,
    ST_Spatial(''ST_Point(100.1, 100.2, 100.3)''))
from spotelev where sw_member = 3
'
```

HG_Slope_Avg

HG_Slope_Avg returns the average slope of all two-point line segments within a line.

The function takes a line parameter and checks for points within the line. If there are points within the line, then multiple line segments are created. The slope is determined for each segment and then the average slope of all segments is calculated.



Syntax

HG_Slope_Avg(*spatial_obj*)

spatial_obj is an ST_Spatial representing a three-dimensional ST_Polyline.

Return Type

HG_Slope_Avg returns an angle in degrees as a double precision number.

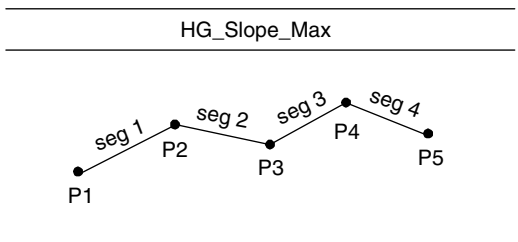
NULL is returned if a two-dimensional line is given as a parameter.

Example

```
exec sp_spatial_query '
select HG_Slope_Avg(sw_geometry) from testcon
where sw_member = 3
'
```

HG_Slope_Max

HG_Slope_Max returns the maximum slope of all two-point line segments within a line. The function receives a line parameter and checks for points within the line. If there are points in the line, then multiple line segments are created. The slope is determined for each segment and the maximum slope is returned.



Syntax

HG_Slope_Max(*spatial_obj*)

spatial_obj is an ST_Spatial representing a three-dimensional ST_Polyline.

Return Type

HG_Slope_Max returns an angle in degrees as a double precision number.

NULL is returned if a two-dimensional line is given as a parameter.

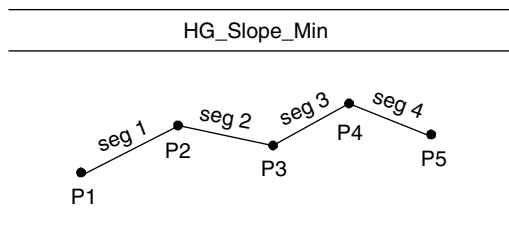
Example

```
exec sp_spatial_query '  
    select HG_Slope_Max(sw_geometry) from testcon  
    where sw_member = 3  
'
```

HG_Slope_Min

HG_Slope_Min returns the minimum slope of all two-point line segments within a line.

If there are points within the input line, then multiple line segments are created. The slope is determined for each segment and the minimum of all slopes is returned.



Syntax

`HG_Slope_Min(spatial_obj)`

spatial_obj is an ST_Spatial representing a three-dimensional ST_Polyline.

Return Type

HG_Slope_Min returns an angle in degrees as a double precision number.

NULL is returned if a two-dimensional line is given as a parameter.

Example

```
exec sp_spatial_query '
    select HG_Slope_Min(sw_geometry) from testcon
    where sw_member =3
    ,
```

HG_SphericalDist

HG_SphericalDist returns the Spherical distance between two points. The distance calculated is in Meters. This function is ideal for telecom users who have data that straddles UTM boundaries.

If your application requires units other than meters it must convert the results. For example, to find the Spherical distance between the cities of Toronto and Las Vegas in Kilometers, divide the value returned by this function by 1000. A list of conversion constants is provided in Appendix: Conversion Constants on page 181.

The input geometries are assumed to be in long/lat, in the same datum. If the geometries are anything other than points, their centroids are used for the distance calculation.

Syntax

`HG_SphericalDist(point1, point2)`

point1 is an ST_Spatial representing an ST_Point in long/lat.

point2 is an ST_Spatial representing an ST_Point in long/lat.

Return Type

HG_SphericalDist returns a double precision number in Meters.

Example

This example is in the form of a script. It creates a table called *na_cities* and populates it with point data representing cities: Toronto, Albany, and Las Vegas. A query is made to view the input geometry by calling *HG_GetString*. The spherical distance between the cities is then calculated using *HG_SphericalDist*. You may want to divide the result by 1000 to convert to Kilometers.

```
create table na_cities (  
    sw_member integer NOT NULL,  
    city varchar(20),  
    sw_geometry st_spatial  
)  
exec sp_sw_spatialize_column  
    'dbo','na_cities','sw_geometry','sw_member',NULL  
insert into na_cities values (1,  
    'Toronto',  
    'ST_Spatial(ST_Point(-79.412672,43.720834))'  
)  
insert into na_cities values (2,  
    'Albany',  
    'ST_Spatial(ST_Point(-73.799017, 42.66575))'  
)  
insert into na_cities values (3,  
    'Las Vegas',  
    'ST_Spatial(ST_Point(-115.222799160,36.205749990))'  
)  
exec sp_spatial_query '  
    select city, HG_GetString(sw_geometry)  
    from na_cities  
'  
exec sp_spatial_query '  
    select a.city, b.city, HG_SphericalDist(  
        a.sw_geometry, b.sw_geometry  
    ) as dist  
    from na_cities a, na_cities b
```

```

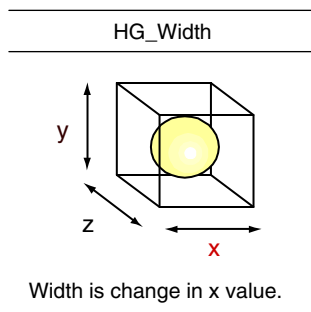
        where a.city < b.city
    ,
    exec sp_sw_despatialize_column
        'dbo','na_cities','sw_geometry','sw_member'
    drop table na_cities

```

The last two lines drop the spatial table: the spatial column is despatialized and then the table is dropped.

HG_Width

HG_Width returns the minimum enclosing rectangle of a geometry. Width is measured as the change in the x-value of the object's minimum enclosing box. The object can be either two or three dimensional.



Syntax

`HG_Width(spatial_obj)`

spatial_obj is an ST_Spatial.

Return Type

HG_Width returns a double precision number.

Example

```

exec sp_spatial_query '
    select HG_Width(sw_geometry) from lake
'

```

ST_Area

ST_Area returns the surface area of polygon geometries; if the input geometry contains entities other than polygons, ST_Area returns a NULL value.

Syntax

ST_Area(*spatial_obj*)

spatial_obj is an ST_Spatial representing a ST_Polygon.

Return Type

ST_Area returns a double precision number.

Example

```
exec sp_spatial_query '
    select ST_Area(sw_geometry) from lake
'
```

Caveat

There are some accuracy considerations when performing area calculations in the Longitude/Latitude coordinate system.

All SpatialWare operators are designed to work in a Cartesian coordinate system. Any of the SpatialWare projections when applied to define a coordinate system will expose a Cartesian coordinate system.

There is however one exception, the Latitude/Longitude coordinate system may not be treated as a Cartesian system. A unit of length, such as area or slope, in a Latitude/Longitude system is undefined. Operators such as ST_Buffer and HG_Circle which accept arguments of "unit" can not provide valid results on a Latitude/Longitude Coordinate System as the operation is mathematically invalid.

Consider the example of the ST_Area function. The mathematically precise way to apply the area function on a geometry stored in Latitude/Longitude coordinate system is to transform it to a suitable coordinate system before applying the function. If you set your current projection to an appropriate coordinate system, the geometry will be transformed to the projected space before the area function is applied. The results will thus be mathematically precise.

ST_Length

ST_Length calculates the length of two dimensional line geometries; if the input geometry contains entities other than lines, ST_Length returns a NULL value.

For three-dimensional line geometries, use ST_Length_3D on page 57.

Syntax

ST_Length(*spatial_obj*)

spatial_obj is an ST_Spatial representing an ST_Polyline.

Return Type

ST_Length returns a double precision number.

Example

```
exec sp_spatial_query '  
    select ST_Length(sw_geometry)  
    from rd paved  
    where sw_member=3  
'
```

ST_Length_3D

ST_Length_3D calculates the length of three dimensional line geometries; if the input geometry contains entities other than lines, ST_Length_3D returns a NULL value. For two-dimensional line geometries, use ST_Length on page 57.

Syntax

ST_Length_3D(*spatial_obj*)

spatial_obj is an ST_Spatial representing a three-dimensional ST_Polyline.

Return Type

ST_Length_3D returns a double precision number.

Usage

Use this function with three-dimensional geometry.

Example

```
exec sp_spatial_query '
    select ST_Length_3D(sw_geometry) from testcon
    '
```

ST_Perimeter

ST_Perimeter calculates the perimeter of two-dimensional polygon geometries; if the input geometry contains entities other than polygons, ST_Perimeter returns a NULL value.

For three-dimensional objects, use ST_Perimeter_3D on page 58.

Syntax

ST_Perimeter(*spatial_obj*)

spatial_obj is an ST_Spatial representing an ST_Polygon.

Return Type

ST_Perimeter returns a double precision number.

Example

```
exec sp_spatial_query '
    select ST_Perimeter(sw_geometry) from lake
    '
```

ST_Perimeter_3D

ST_Perimeter_3D calculates the perimeter of polygon geometries in three-dimensional space; if the input geometry contains entities other than polygons, ST_Perimeter_3D returns a NULL value. For two-dimensional objects, use ST_Perimeter on page 58.

Syntax

ST_Perimeter_3D(*spatial_obj*)

spatial_obj is an ST_Spatial representing a three-dimensional ST_Polygon.

Return Type

ST_Perimeter_3D returns a double precision number.

Usage

Use this function with three-dimensional geometry.

Example

```
exec sp_spatial_query '  
    select ST_Perimeter_3d(sw_geometry) from lake  
,
```

Observer Functions

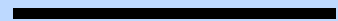
Spatial Observer functions return numbers, objects, or conditions from within a geometry.



8

Chapter

► Function Descriptions



Function Descriptions

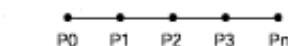
Observer functions include:

HG_Begin_Point	HG_Llb_X	HG_Pointdyn_Ori_Y
HG_Cen_X	HG_Llb_Y	HG_Pointdyn_Rot
HG_Cen_Y	HG_Llb_Z	HG_Pointdyn_Xscale
HG_Cen_Z	HG_Ncoords	HG_Pointdyn_Yscale
HG_Center_Point	HG_Ncurves	HG_PointN
HG_Corner	HG_Nitems	HG_Radians
HG_Curve	HG_Npaths	HG_Start_Arc_Rot
HG_End_Arc_Rot	HG_Npoints	HG_Start_Tangent
HG_End_Point	HG_Npolygons	HG_Start_Tangent_P
HG_End_Tangent	HG_Nsubcurves	HG_Subcurve
HG_End_Tangent_P	HG_Ori_Rotation	HG_Urt_X
HG_Expanded	HG_Ori_X	HG_Urt_Y
HG_Exterior_Path	HG_Ori_Y	HG_Urt_Z
HG_Extract_At	HG_Ori_Z	ST_X
HG_GeometryN	HG_Point	ST_Y
HG_Interior_Path	HG_Pointdyn_Ori_X	ST_Z

HG_Begin_Point

HG_Begin_Point returns the start point of a line. NULL is returned if the input is not a line.

A line has 0 to n points.



Syntax

HG_Begin_Point(*polyline*)

polyline is an ST_Spatial representing an ST_Polyline.

Return Type

HG_Begin_Point returns an ST_Spatial representing an ST_Point.

Example

```
exec sp_spatial_query '
    select HG_GetString(HG_Begin_Point(sw_geometry))
    from rd paved
    ,
```

Caveat

HG_GetString on page 42 has been added to the example so that it will return geometry in text format.

HG_Cen_X

HG_Cen_X returns the x-value of an ST_Spatial's centroid. All ST_Spatials contain information about their centroid: the weighted mathematical center of an ST_Spatial.

Note: The centroid of an ST_Spatial is the weighted, mathematical center point of an object. The x, y, and z coordinates are accessible. It is an ST_Point.

Syntax

HG_Cen_X(*spatial_obj*)

spatial_obj is any ST_Spatial.

Return Type

HG_Cen_X returns a double precision number.

Example

```
exec sp_spatial_query '
    select HG_Cen_X(sw_geometry)
    from lake
    ,
```

HG_Cen_Y

HG_Cen_Y returns the y-value of an ST_Spatial's centroid. All ST_Spatials contain information about their centroid: the weighted mathematical center of an ST_Spatial.

Note: The centroid of an ST_Spatial is the weighted, mathematical center point of an object. The x, y, and z coordinates are accessible. It is an ST_Point.

Syntax

HG_Cen_Y(*spatial_obj*)

spatial_obj is any ST_Spatial.

Return Type

HG_Cen_Y returns a double precision number.

Example

```
exec sp_spatial_query '  
    select HG_Cen_Y(sw_geometry)  
    from lake  
,
```

HG_Cen_Z

HG_Cen_Z returns the z-value of an ST_Spatial's centroid. All ST_Spatials contain information about their centroid: the weighted mathematical center of an ST_Spatial.

Note: The centroid of an ST_Spatial is the weighted, mathematical center point of an object. The x, y, and z coordinates are accessible. It is an ST_Point.

Syntax

HG_Cen_Z(*spatial_obj*)

spatial_obj is any ST_Spatial.

Return Type

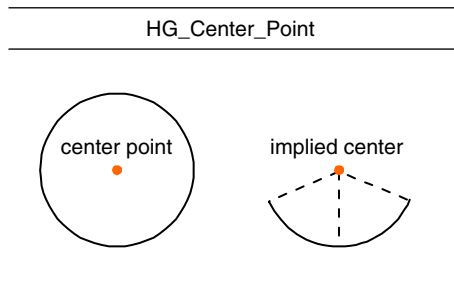
HG_Cen_Z returns a double precision number.

Example

```
exec sp_spatial_query '  
    select HG_Cen_Z(sw_geometry)  
    from lake  
,
```

HG_Center_Point

HG_Center_Point returns the center point of an ST_CircularArc, or HG_Circle. The center point is the center of an arc shape. In the case of an ST_CircularArc, it is an implied center.



Syntax

HG_Center_Point(*spatial_obj*)

spatial_obj is an ST_Spatial representing an ST_CircularArc, or HG_Circle.

Return Type

HG_Center_Point returns an ST_Spatial representing an ST_Point.

Example

```
exec sp_spatial_query '
    select HG_GetString(HG_Center_Point(sw_geometry))
    from testcon
    where HG_Is_Circle(sw_geometry)
    ,
```

Caveat

HG_GetString on page 42 has been added to the example so that it will return geometry in text format.

HG_Corner

HG_Corner is a synonym for HG_Point. It returns a specific ST_Point from an ST_Curve.

ST_Curves contain *zero* to *n* ST_Points, stored sequentially in a LIST. The number parameter in HG_Corner selects a particular ST_Point from that LIST.

Syntax

HG_Corner(*spatial_obj*, *number*)

spatial_obj is an ST_Spatial representing an HG_Curve.

number is an integer representing the position of an object in a LIST.

Return Type

HG_Corner returns an ST_Spatial representing an ST_Point.

Example

```
exec sp_spatial_query '  
    select HG_GetString(HG_Corner(a.sw_geometry, 3))  
    from rdpaved  
    where sw_member = 2  
'
```

Caveat

HG_GetString on page 42 has been added to the example so that it will return geometry in text format.

HG_Curve

HG_Curve returns a specific ST_Curve from an ST_Path. ST_Paths are composed of *zero* to *n* ST_Curves, stored sequentially in a LIST. The number parameter sent to HG_Curve selects a particular ST_Curve from that LIST.

Syntax

HG_Curve(*spatial_obj*, *number*)

spatial_obj is an ST_Spatial representing an ST_Path.

number is an integer representing the position of an object in a LIST.

Return Type

HG_Curve returns an ST_Spatial representing an HG_Curve.

Example

```
exec sp_spatial_query '  
    select HG_GetString(HG_Curve(a.sw_geometry, 3))
```



```

        from testcon
    ,

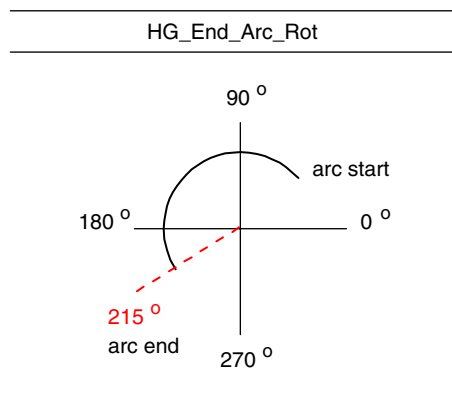
```

Caveat

HG_GetString on page 42 has been added to the example so that it will return geometry in text format.

HG_End_Arc_Rot

HG_End_Arc_Rot returns the ST_CircularArc's end angle in degrees: zero (0) degrees is the positive X-axis, ninety (90) degrees is the positive Y-axis.



Refer also to the description for HG_Start_Arc_Rot on page 86.

Syntax

HG_End_Arc_Rot(*spatial_obj*)

spatial_obj is an ST_Spatial representing an ST_CircularArc.

Return Type

HG_End_Arc_Rot returns a real value.

Example

```

exec sp_spatial_query '
    select HG_End_Arc_Rot(sw_geometry)
    from lake
    ,

```

HG_End_Point

HG_End_Point returns the end point of a line.

Syntax

HG_End_Point(*polyline*)

polyline is an ST_Spatial representing a ST_Polyline.

Return Type

HG_End_Point returns an ST_Spatial representing a ST_Point.

Example

```
exec sp_spatial_query '
    select HG_GetString(HG_End_Point(sw_geometry))
    from rd paved
    ,
```

Caveat

HG_GetString has been added to the example so that it will return geometry in text format.

HG_End_Tangent

HG_End_Tangent returns the tangent of an HG_Curve's end point.

Syntax

HG_End_Tangent(*spatial_obj*)

spatial_obj is an ST_Spatial representing an HG_Curve.

Return Type

HG_End_Tangent returns a real value.

Example

```
exec sp_spatial_query '
    select HG_End_Tangent(sw_geometry)
    from testcon
    where HG_Is_HG_Curve(sw_geometry)
    ,
```

HG_End_Tangent_P

HG_End_Tangent_P returns an ST_Point on the tangent of an HG_Curve's end point.

Syntax

HG_End_Tangent_P(*spatial_obj*)

spatial_obj is an ST_Spatial representing an HG_Curve.

Return Type

HG_End_Tangent_P returns an ST_Spatial representing an ST_Point.

Example

```
exec sp_spatial_query '  
    select HG_GetString(HG_End_Tangent_P(sw_geometry))  
    from testcon  
    where hg_is_hg_curve(sw_geometry)  
,
```

Caveat

HG_GetString on page 42 has been added to the example so that it will return geometry in text format.

HG_Expanded

HG_Expanded returns the point equivalent of an ST_Curve. For example, an ST_CircularArc is defined by three points and an equation. Instead of three points, HG_Expanded returns 129 points. The default conversion rate is 128 points/180 degrees of curve.

Syntax

HG_Expanded(*spatial_obj*)

spatial_obj is an ST_Spatial representing an HG_Curve.

Return Type

HG_Expanded returns an ST_Spatial.

Example

```
exec sp_spatial_query '  
    select HG_GetString(HG_Expanded(sw_geometry))
```

```
        from rdpaved
        where sw_member = 1
    ,
```

Caveat

HG_GetString on page 42 has been added to the example so that it will return geometry in text format.

HG_Exterior_Path

HG_Exterior_Path returns the exterior boundary of a polygon, an ST_Polygon. ST_Polygons have one exterior boundary and *zero* to *n* interior boundaries, defined by ST_Paths.

The input geometry must be a polygon, otherwise, HG_Exterior_Path returns NULL.

Syntax

HG_Exterior_Path(*spatial_obj*)

spatial_obj is an ST_Spatial representing a ST_Polygon.

Return Type

HG_Exterior_Path returns an ST_Spatial representing a ST_Path.

Example

```
exec sp_spatial_query '
    select HG_GetString(HG_Exterior_Path(sw_geometry))
    from lake
    ,
```

Caveat

HG_GetString on page 42 has been added to the example so that it will return geometry in text format.

HG_Extract_At

HG_Extract_At returns the primitive at a given position in a SpatialObject. The position argument can be between 0 and n-1 in a SpatialObject that has n primitives. The returned value is an instance of ST_Spatial. This function is used for extracting individual primitives in a multi-element geometry.

If the specified position is -1, then the whole SpatialObject is returned.

Syntax

HG_Extract_At(*spatial_obj*, *ref_number*)

spatial_obj is any ST_Spatial.

ref_number is an integer.

Return Type

HG_Extract_At returns an ST_Spatial.

Example

This example selects the first primitive of the SpatialObject whose sw_member is 1.

```
exec sp_spatial_query '  
    select HG_GetString(HG_Extract_At(sw_geometry, 0))  
    from rd paved  
    where sw_member = 1  
'
```

Caveat

HG_GetString on page 42 has been added to the example so that it will return geometry in text format.

HG_GeometryN

HG_GeometryN extracts the Nth geometry in a multi-element SW_GEOMETRY, where N is a specified position. The index on HG_GeometryN is zero based. For example, zero (0) is used for the first element and one (1) is used for the second element.

If the input number is greater than the number of points in the geometry, NULL is returned.

Syntax

HG_GeometryN(*spatial_obj*, *position*)

spatial_obj is an ST_Spatial.

position is an integer.

Return Type

HG_GeometryN returns an ST_Spatial.

Example

```
exec sp_spatial_query '
    select HG_GetString(HG_GeometryN(sw_geometry, 1))
    from rdpaved
    where sw_member = 7
    '
```

Caveat

HG_GetString on page 42 has been added to the example so that it will return geometry in text format.

HG_Interior_Path

HG_Interior_Path returns a specific interior boundary from a polygon, an ST_Polygon. ST_Polygons have one exterior boundary and zero to n-1 interior boundaries, defined by ST_Paths. The interior boundaries are stored sequentially in a LIST. The number parameter that is sent to HG_Interior_Path selects a particular interior boundary from that LIST.

Syntax

HG_Interior_Path(*spatial_obj*, *number*)

spatial_obj is an ST_Spatial representing a ST_Polygon.

number is an integer representing the position of an object in a LIST that starts with zero (i.e., 0, 1, 2, ... n).

Return Type

HG_Interior_Path returns an ST_Spatial representing an ST_Path.

Example

This example creates a polygon with a hole in it in the testcon table:

```
insert into testcon (sw_member, sw_geometry) values (63,
    'ST_Spatial(ST_Polygon(
        HG_Box(0,0,5,5),
        ST_Path(LIST{
            ST_Polyline(LIST{
                ST_Point(1,1),
                ST_Point(2,2)
            }),
            ST_CircularArc(LIST{
```

```
        ST_Point(2,2),
        ST_Point(2,3),
        ST_Point(3,3)
    }),
    ST_Polyline(LIST{
        ST_Point(3,3),
        ST_Point(3,1),
        ST_Point(1,1)
    })
})
))'
)
exec sp_spatial_query '
    select HG_GetString(HG_Interior_Path(sw_geometry, 0))
    from testcon
    ,
```

Caveat

HG_GetString on page 42 has been added to the example so that it will return geometry in text format.

HG_Llb_X

HG_Llb_X returns the x-coordinate of the lower left corner of the minimum enclosing rectangle for the geometry.

Syntax

HG_Llb_X(*spatial_obj*)

spatial_obj is an ST_Spatial representing a ST_Polygon.

Return Type

HG_Llb_X returns a double precision number.

Example

```
exec sp_spatial_query '
    select HG_Llb_X(sw_geometry)
    from lake
    ,
```

HG_Llb_Y

HG_Llb_Y returns the y-coordinate of the lower left corner of the minimum enclosing rectangle for the geometry.

Syntax

HG_Llb_Y(*spatial_obj*)

spatial_obj is an ST_Spatial representing a ST_Polygon.

Return Type

HG_Llb_Y returns a double precision number.

Example

```
exec sp_spatial_query '  
    select HG_Llb_Y(sw_geometry)  
    from lake  
,
```

HG_Llb_Z

HG_Llb_Z returns the z-coordinate of the lower left corner of the minimum enclosing rectangle for the geometry.

Syntax

HG_Llb_Z(*spatial_obj*)

spatial_obj is an ST_Spatial representing a ST_Polygon.

Return Type

HG_Llb_Z returns a double precision number.

Example

```
exec sp_spatial_query '  
    select HG_Llb_Z(sw_geometry)  
    from lake  
,
```


HG_Ncoords

HG_Ncoords returns the total number of points in an ST_Spatial, counting points embedded in other geometry constructors (ST_GeometricPrimitives). If you want just the number of ST_Points, use HG_Npoints on page 77.

Syntax

HG_Ncoords(*spatial_obj*)

spatial_obj is any ST_Spatial.

Return Type

HG_Ncoords returns an integer.

Example

```
exec sp_spatial_query '
    select HG_Ncoords(sw_geometry)
    from lake
    ,
```

HG_Ncurves

HG_Ncurves returns the number of ST_Curves in an ST_Spatial. The function does **not** count embedded HG_Curves.

Syntax

HG_Ncurves(*spatial_obj*)

spatial_obj is any ST_Spatial.

Return Type

HG_Ncurves returns an integer.

Example

```
exec sp_spatial_query '
    select HG_Ncurves(sw_geometry)
    from rd paved
    where sw_member < 7
    ,
```

```
select HG_Ncurves(sw_geometry)
from rdpaved
where sw_member < 7;
```

HG_Nitems

HG_Nitems returns the number of geometries in an ST_Spatial.

The function does not count embedded items.

Syntax

HG_Nitems(*spatial_obj*)

spatial_obj is any ST_Spatial.

Return Type

HG_Nitems returns an integer.

Example

```
exec sp_spatial_query '
    select HG_Nitems(sw_geometry)
    from lake
    '
```

HG_Npaths

HG_Npaths returns the number of ST_Paths in an ST_Spatial. The function does not count embedded ST_Paths.

Syntax

HG_Npaths(*spatial_obj*)

spatial_obj is any ST_Spatial.

Return Type

HG_Npaths returns an integer.

Example

```
exec sp_spatial_query '
    select HG_Npaths(sw_geometry)
```

```
        from lake  
,
```

HG_Npoints

HG_Npoints returns the number of ST_Points in an ST_Spatial. The function does not count embedded ST_Points.

Syntax

HG_Npoints(*spatial_obj*)

spatial_obj is any ST_Spatial.

Return Type

HG_Npoints returns an integer.

Example

```
exec sp_spatial_query '  
    select HG_Npoints(sw_geometry)  
    from spotelev  
    where sw_member = 11  
,
```

HG_Npolygons

HG_Npolygons returns the number of ST_Polygons in an ST_Spatial. The function does not count embedded ST_Polygons.

Syntax

HG_Npolygons(*spatial_obj*)

spatial_obj is any ST_Spatial.

Return Type

HG_Npolygons returns an integer.

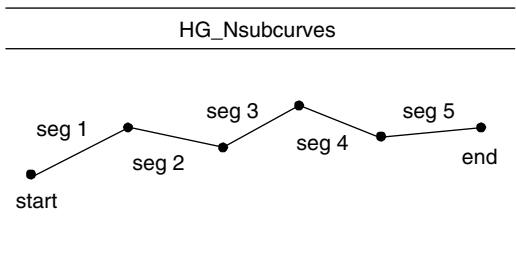
Example

```
exec sp_spatial_query '  
    select HG_Npolygons(sw_geometry)  
    from parcel
```

```
    where sw_member < 10
    ,
```

HG_Nsubcurves

HG_Nsubcurves returns the number of subcurves embedded in the ST_Lines contained by ST_Spatial. Subcurves are two-point line segments based on a line's start, end, and internal points.



Syntax

HG_Nsubcurves(*spatial_obj*)
spatial_obj is any ST_Spatial.

Return Type

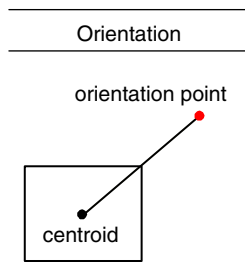
HG_Nsubcurves returns an integer.

Example

```
exec sp_spatial_query '
    select HG_Nsubcurves(sw_geometry)
    from rdpaved
    ,
```

HG_Ori_Rotation

HG_Ori_Rotation returns the angle of rotation of an ST_Spatial's orientation point. The orientation point is an attribute of all ST_Spatials used frequently for attribute positioning. All ST_Spatials have an orientation point, which can be used for map-attribute positioning.



The placement of the orientation point can communicate angle and rotation.

The angle of rotation is determined by the position of the orientation point, relative to the centroid

Syntax

`HG_Ori_Rotation(spatial_obj)`

spatial_obj is an ST_Spatial.

Return Type

HG_Ori_Rotation returns a type real.

Example

```
exec sp_spatial_query '
    select HG_Ori_Rotation(sw_geometry)
    from spotelev
    where sw_member = 1
    ,
```

HG_Ori_X

HG_Ori_X returns the x-coordinate value of an ST_Spatial's orientation point. The orientation point is a property of all ST_Spatials, used frequently for attribute positioning. For more information, see HG_Ori_Rotation on page 78.

Syntax

`HG_Ori_X(spatial_obj)`

spatial_obj is an ST_Spatial.

Return Type

HG_Ori_X returns a double precision number.

Example

```
exec sp_spatial_query '  
    select HG_Ori_X(sw_geometry)  
    from spotelev  
    where sw_member = 11  
,
```

HG_Ori_Y

HG_Ori_Y returns the y-coordinate value of an orientation point. The orientation point is a property of all ST_Spatials, used frequently for attribute positioning. For more information, see HG_Ori_Rotation on page 78.

Syntax

HG_Ori_Y(*spatial_obj*)

spatial_obj is an ST_Spatial.

Return Type

HG_Ori_Y returns a double precision number.

Example

```
exec sp_spatial_query '  
    select HG_Ori_Y(sw_geometry)  
    from spotelev  
    where sw_member = 11  
,
```

HG_Ori_Z

HG_Ori_Z returns the z-coordinate value of an orientation point. The orientation point is a property of all ST_Spatials, used frequently for attribute positioning. For more information, see HG_Ori_Rotation on page 78.

Syntax

HG_Ori_Z(*spatial_obj*)

spatial_obj is an ST_Spatial.

Return Type

HG_Ori_Z returns a double precision number.

Example

```
exec sp_spatial_query '
    select HG_Ori_Z(sw_geometry)
    from spotelev
    where sw_member = 11
    ,
```

HG_Point

HG_Point returns a specific ST_Point from an ST_Curve. ST_Curves contain zero to n ST_Points, stored sequentially in a LIST. The number parameter in HG_Point selects a particular ST_Point from that LIST. HG_Point is identical to the function HG_PointN on page 85.

Syntax

HG_Point(*spatial_obj*, *number*)

spatial_obj is an ST_Spatial representing an HG_Curve.

number is an integer representing the position of an object in a LIST.

Return Type

HG_Point returns an ST_Spatial representing an ST_Point.

Example

```
exec sp_spatial_query '
    select HG_GetString(HG_Point(sw_geometry, 3))
    from rdpaved
    where sw_member < 24
    ,
```

Caveat

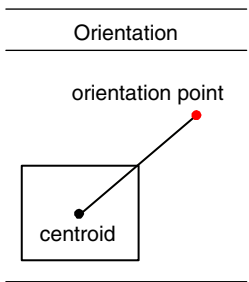
HG_GetString on page 42 has been added to the example so that it will return geometry in text format.

HG_Pointdyn_Ori_X

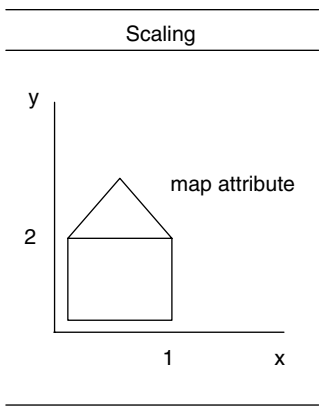
HG_Pointdyn_Ori_X returns the x-coordinate value of an ST_Point's dynamic orientation point. The dynamic orientation point is one of ST_Point's dynamic attributes, used for map attribute positioning and sizing.

ST_Points can store information about scale and orientation, commonly used for map attribute positioning and sizing. These attributes compose the point dynamic.

One element of the point dynamic is orientation. It is similar to the global ST_Spatial orientation, but it is an additional place to store orientation information. So, an ST_Point can have two levels of orientation: its orientation as an ST_Spatial and its orientation as an ST_Point.



The second element of the point dynamic is scale. Scale allows applications that do mapping or plotting to specify the size of an attribute associated with an ST_Point. Scaling can be done for both x and y axes.



The scale is based on *one unit of plot*.

Syntax

HG_Pointdyn_Ori_X(*spatial_obj*)

spatial_obj is an ST_Spatial representing an ST_Point.

Return Type

HG_Pointdyn_Ori_X returns a double precision number.

Example

```
exec sp_spatial_query '  
    select HG_Pointdyn_Ori_X(sw_geometry)  
    from spotelev  
    where sw_member = 1  
,
```

HG_Pointdyn_Ori_Y

HG_Pointdyn_Ori_Y returns the y-coordinate value of an ST_Point's orientation point. The orientation point is one of ST_Point's dynamic attributes, used for map attribute positioning and sizing. For more information, refer to HG_Pointdyn_Ori_X on page 82.

Syntax

HG_Pointdyn_Ori_Y(*spatial_obj*)

spatial_obj is an ST_Spatial representing an ST_Point.

Return Type

HG_Pointdyn_Ori_Y returns a double precision number.

Example

```
exec sp_spatial_query '  
    select HG_Pointdyn_Ori_Y(sw_geometry)  
    from spotelev  
    where sw_member = 1  
,
```

HG_Pointdyn_Rot

HG_Pointdyn_Rot returns the rotation, in degrees, of an ST_Point's dynamic orientation point, relative to an ST_Point. The point dynamic attributes are used frequently for map

attribute positioning and sizing. For more information, refer to HG_Pointdyn_Ori_X on page 82.

Syntax

HG_Pointdyn_Rot(*spatial_obj*)

spatial_obj is an ST_Spatial representing an ST_Point.

Return Type

HG_Pointdyn_Rot returns a value of type real.

Example

```
exec sp_spatial_query '  
    select HG_Pointdyn_Rot(sw_geometry)  
    from spotelev  
    where sw_member = 1  
,
```

HG_Pointdyn_Xscale

HG_Pointdyn_Xscale returns the X-axis scale value of an ST_Point's dynamic scale. Scale is an attribute of all ST_Points, used frequently for map attribute positioning and sizing. The scale is based on one unit of plot scale. For more information, refer to HG_Pointdyn_Ori_X on page 82.

Syntax

HG_Pointdyn_Xscale(*spatial_obj*)

spatial_obj is an ST_Spatial representing an ST_Point.

Return Type

HG_Pointdyn_Xscale returns a value of type real.

Example

```
exec sp_spatial_query '  
    select HG_Pointdyn_Xscale(sw_geometry)  
    from spotelev  
    where sw_member = 1  
,
```

HG_Pointdyn_Yscale

HG_Pointdyn_Yscale returns the Y-axis value of an ST_Point's scale attribute. The scale attribute is one of ST_Point's dynamic attributes, used frequently for attribute positioning and sizing. The scale is based on one unit of plot scale.

Syntax

HG_Pointdyn_Yscale(*spatial_obj*)

spatial_obj is an ST_Spatial representing an ST_Point.

Return Type

HG_Pointdyn_Yscale returns a value of type real.

Example

```
exec sp_spatial_query '
    select HG_Pointdyn_Yscale(sw_geometry)
    from spotelev
    where sw_member = 1
    ,
```

HG_PointN

HG_PointN returns the Nth point in a line geometry, where N is a specified position. The index on HG_Point is zero based. For example, zero (0) is used for the first element and one (1) is used for the second element.

If the input number is greater than the number of points in the geometry, NULL is returned.

Syntax

HG_PointN(*spatial_obj*, *position*)

spatial_obj is an ST_Spatial representing an ST_Polyline.

position is an integer representing the position of a point in a line geometry.

Return Type

HG_PointN returns an ST_Spatial representing an ST_Point.

Example

```
exec sp_spatial_query '  
    select HG_GetString(HG_PointN(sw_geometry, 3))  
    from rdpaved  
    where sw_member = 35  
,
```

Caveat

HG_GetString on page 42 has been added to the example so that it will return geometry in text format.

HG_Radians

HG_Radians returns the length of the radius of an ST_CircularArc, or HG_Circle.

Syntax

HG_Radians(*spatial_obj*)

spatial_obj is an ST_Spatial representing an ST_CircularArc, or HG_Circle.

Return Type

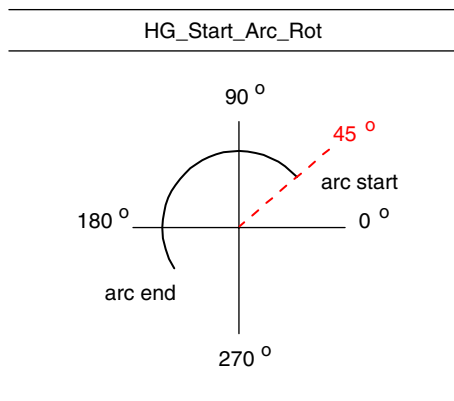
HG_Radians returns a double precision number.

Example

```
exec sp_spatial_query '  
    select HG_Radians(sw_geometry)  
    from testcon  
    where HG_Is_Circle(sw_geometry)  
,
```

HG_Start_Arc_Rot

HG_Start_Arc_Rot returns the ST_CircularArc's start angle in degrees: zero (0) degrees is the positive X-axis, ninety (90) degrees is the positive Y-axis.



Refer also to the description for HG_End_Arc_Rot on page 67.

Syntax

HG_Start_Arc_Rot(*spatial_obj*)

spatial_obj is an ST_Spatial representing an ST_CircularArc.

Return Type

HG_Start_Arc_Rot returns a real value.

HG_Start_Tangent

HG_Start_Tangent returns the tangent of an HG_Curve's start point.

Syntax

HG_Start_Tangent(*spatial_obj*)

spatial_obj is an ST_Spatial representing an HG_Curve.

Return Type

HG_Start_Tangent returns a real value.

Example

```
exec sp_spatial_query '  
    select HG_Start_Tangent(sw_geometry)  
    from testcon  
    where HG_Is_HG_Curve(sw_geometry)  
'
```

HG_Start_Tangent_P

HG_Start_Tangent_P returns an ST_Point on the tangent of an HG_Curve's start point.

Syntax

`HG_Start_Tangent_P(spatial_obj)`

spatial_obj is an ST_Curve.

Return Type

HG_Start_Tangent_P returns an ST_Spatial representing an ST_Point.

Example

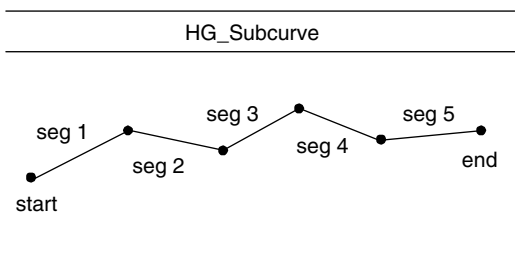
```
exec sp_spatial_query '  
    select HG_GetString(HG_Start_Tangent_P(sw_geometry))  
    from testcon  
    where HG_Is_HG_Curve(sw_geometry)  
'
```

Caveat

HG_GetString on page 42 has been added to the example so that it will return geometry in text format.

HG_Subcurve

HG_Subcurve returns a specific subcurve from an ST_Curve. A subcurve is a two-point line segment contained in an ST_Curve. The *start point*, and *end point* parameters determine which subcurve is returned.



Syntax

`HG_Subcurve(spatial_obj, start_point, end_point)`

spatial_obj is an ST_Spatial representing an HG_Curve.

start_point is an integer representing the position of an ST_Point in a LIST.

end_point is an integer representing the position of an ST_Point in a LIST.

Return Type

HG_Subcurve returns an ST_Spatial.

Example

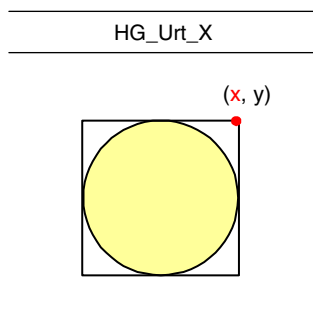
```
exec sp_spatial_query '  
    select HG_GetString(HG_Subcurve(sw_geometry, 1, 2))  
    from rd paved  
    where sw_member = 8  
'
```

Caveat

HG_GetString on page 42 has been added to the example so that it will return geometry in text format.

HG_Urt_X

HG_Urt_X returns the x-coordinate of the upper right corner of the minimum enclosing rectangle for the geometry.



Syntax

HG_Urt_X(*spatial_obj*)

spatial_obj is an ST_Spatial representing an ST_Polygon.

Return Type

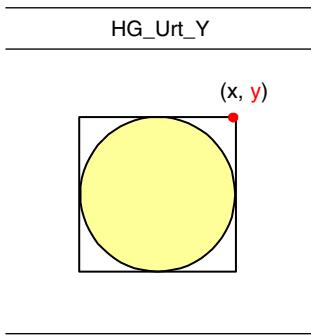
HG_Urt_X returns a double precision number.

Example

```
exec sp_spatial_query '  
    select HG_Urt_X(sw_geometry)  
    from lake  
,
```

HG_Urt_Y

HG_Urt_Y returns the y-coordinate of the upper right corner of the minimum enclosing rectangle for the geometry.



Syntax

HG_Urt_Y(*spatial_obj*)
spatial_obj is an ST_Spatial representing an ST_Polygon.

Return Type

HG_Urt_Y returns a double precision number.

Example

```
exec sp_spatial_query '  
    select HG_Urt_Y(sw_geometry)  
    from lake  
,
```


HG_Urt_Z

HG_Urt_Z returns the z-coordinate of the upper right corner of the minimum enclosing rectangle for the geometry.

Syntax

HG_Urt_Z(*spatial_obj*)

spatial_obj is an ST_Spatial representing an ST_Polygon.

Return Type

HG_Urt_Z returns a double precision number.

Example

```
exec sp_spatial_query '
    select HG_Urt_Z(sw_geometry)
    from lake
    ,
```

ST_X

ST_X returns the x-coordinate of a point geometry. If the geometry is not a point, ST_X returns NULL.

Syntax

ST_X(*spatial_obj*)

spatial_obj is an ST_Spatial representing an ST_Point.

Return Type

ST_X returns a double precision number.

Example

```
exec sp_spatial_query '
    select ST_X(sw_geometry)
    from spotelev
    ,
```

ST_Y

ST_Y returns the y-coordinate of a point geometry. If the geometry is not a point, ST_Y returns NULL.

Syntax

ST_Y(*spatial_obj*)

spatial_obj is an ST_Spatial representing an ST_Point.

Return Type

ST_Y returns a double precision number.

Example

```
exec sp_spatial_query '
    select ST_Y(sw_geometry)
    from spotelev
    '
```

ST_Z

ST_Z returns the z-coordinate of a point geometry. If the geometry is not a point, ST_Z returns NULL.

Syntax

ST_Z(*spatial_obj*)

spatial_obj is an ST_Spatial representing an ST_Point.

Return Type

ST_Z returns a double precision number.

Example

```
exec sp_spatial_query '
    select ST_Z(sw_geometry)
    from spotelev
    '
```

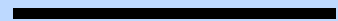
Spatial Functions

Spatial functions perform operations on geometries to create new geometries. Spatial functions include:

9

Chapter

- **Function Descriptions**
- **Transformation Calculation**



Function Descriptions

Please note that HG_GetString has been added to some of the examples in this section, so that they return geometry in text format.

HG_Affine	HG_Connect	HG_Intersect_In
HG_Affine3D	HG_Convex_Hull	HG_LL_Circle
HG_As_Curves	HG_Difference	HG_Split
HG_As_Paths	HG_End_Of	HG_Start_Of
HG_As_Points	HG_Envelope	HG_Sym_Difference
HG_Center_In	HG_Erase	HG_Union
HG_Center_In_3D	HG_Erase_Outside	ST_Adjacent
HG_Centroid	HG_Filter	ST_Buffer
HG_Clean	HG_Filter_Curves	ST_Contain
HG_Clean_I	HG_Filter_Paths	ST_Overlap
HG_Clean_S	HG_Filter_Points	ST_Transform
HG_Combine	HG_Filter_Polygons	ST_Transform3D

HG_Affine

The HG_Affine function performs a 2x3 affine transformation on a geometry.

In a 2x3 affine transformation a matrix is used to specify the coefficients of the general form affine transformation:

$$\begin{aligned}x' &= A*x + B*y + C \\ y' &= D*x + E*y + F\end{aligned}$$

Syntax

HG_Affine(*spatial_obj*, *A*, *B*, *C*, *D*, *E*, *F*)

spatial_obj is an ST_Spatial.

A, *B*, *C*, *D*, *E*, *F* are 2x3 matrix double values.

Return Type

HG_Affine returns an ST_Spatial.

Example

This example scales a point (1,1) by 100 in the x direction about origin, by -100 in the y direction about origin, and then translates the result by (1, -1).

```
exec sp_spatial_query '
select HG_Affine (
    ST_Spatial(''ST_Point(1, 1)''),
    100.0, 0.0, 1.0, 0.0, -100.0, -1.0)
from dual
'
```

In this example:

$x = 1$

$y = 1$

The result will be:

$$\begin{aligned}x' &= 100 \cdot x + 0 \cdot y + 1 = 101 \\y' &= 0 \cdot x + (-100) \cdot y + (-1) = -101\end{aligned}$$

HG_Affine3D

The HG_Affine3D function performs a 3x4 affine transformation on three-dimensional geometry.

In a 3x4 affine transformation HG_Matrix_3x4 is used to specify the coefficients of the general form affine transformation:

$$x' = A \cdot x + B \cdot y + C \cdot z + D$$

$$y' = E \cdot x + F \cdot y + G \cdot z + H$$

$$z' = I \cdot x + J \cdot y + K \cdot z + L$$

Syntax

HG_Affine3D(*spatial_obj*, *A*, *B*, *C*, *D*, *E*, *F*, *G*, *H*, *I*, *J*, *K*, *L*)

spatial_obj is an ST_Spatial.

A,B,C,D,E,F,G,H,I,J,K,L are 3x4 matrix double values.

Return Type

HG_Affine3D returns an ST_Spatial.

Usage

Use this function with three-dimensional geometry.

Example

This HG_Affine3D example scales a point (1,1,2) by 100 in the x direction about origin, by -100 in the y direction about origin, by 1 in the z direction about the origin, and then translates the result by (1, -1, 0).

```
exec sp_spatial_query '  
    select HG_Affine3D(  
        ST_Spatial(''ST_Point(1, 1, 2)''),  
        100.0, 0.0, 0.0, 1.0, 0.0, -100.0,  
        0.0, -1.0, 0.0, 0.0, 1.0, 0.0  
    )  
    from dual  
,
```

In this example $x = 1$, $y = 1$, and $z = 2$. The result will be:

$$\begin{aligned}x' &= 100 \cdot x + 0 \cdot y + 0 \cdot z + 1 = 101 \\y' &= 0 \cdot 1 + (-100) \cdot y + 0 \cdot z + (-1) = -101 \\z' &= 0 \cdot x + 0 \cdot y + 1 \cdot z + 0 = 2\end{aligned}$$

HG_As_Curves

HG_As_Curves returns a copy of an instance of a geometry, where all polygon and path elements are replaced by their composite curves (HG_Curve).

Syntax

HG_As_Curves(*spatial_obj*)

spatial_obj is an ST_Spatial.

Return Type

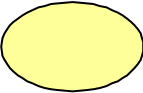
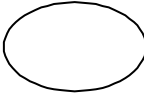
HG_As_Curves returns ST_Spatial representing an ST_Polyline.

Example

```
exec sp_spatial_query '  
    select HG_GetString(HG_As_Curves(sw_geometry))  
    from flood100  
,
```

HG_As_Paths

HG_As_Paths returns a copy of an SW_GEOMETRY, where all polygons are replaced by the line-strings forming their boundary.

Input	Output
	
polygon	line string

Syntax

HG_As_Paths(*spatial_obj*)

spatial_obj is an ST_Spatial.

Return Type



HG_As_Paths returns an ST_Spatial representing an ST_Polyline.

Example

```
exec sp_spatial_query '
    select HG_GetString(HG_As_Paths(sw_geometry))
    from flood100
    ,
```

HG_As_Points

HG_As_Points returns the pointal equivalent of a spatial object by breaking primitives down into their component points. In the case of a circle, only three points are returned: the center point, and two points on the circumference.

Input	Output
	
polygon	points

Syntax

HG_As_Points(*spatial_obj*)
spatial_obj is an ST_Spatial.

Return Type

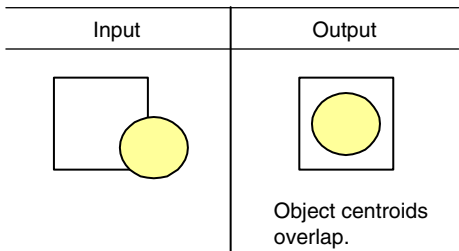
HG_As_Points returns an ST_Spatial representing an ST_Point.

Example

```
exec sp_spatial_query '  
    select HG_GetString(HG_As_Points(sw_geometry))  
    from flood100  
'
```

HG_Center_In

HG_Center_In centers *spatial_obj1* on *spatial_obj2*, in two dimensional space.



spatial_obj1 (e.g., the circle) is transformed so that its centroid overlaps the centroid of *spatial_obj2* (e.g., the box).

Syntax

HG_Center_In(*spatial_obj1*, *spatial_obj2*)
spatial_obj1 is an ST_Spatial.
spatial_obj2 is an ST_Spatial.

Return Type

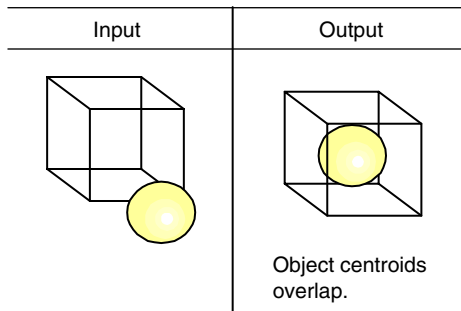
HG_Center_In returns an ST_Spatial.

Example

```
exec sp_spatial_query '
    select HG_GetString(HG_Center_In(
        flood100.sw_geometry,
        ST_Spatial(''ST_Point(1753960.182000, 10698421.103000)'')
    ))
    from flood100
    ,
```

HG_Center_In_3D

HG_Center_In_3D centers *spatial_obj1* in *spatial_obj2*, in three-dimensional space.



spatial_obj1 (e.g., the circle) is transformed so that its centroid overlaps the centroid of *spatial_obj2* (e.g., the box).

Syntax

HG_Center_In_3D(*spatial_obj1*, *spatial_obj2*)
spatial_obj1 is an ST_Spatial to be centered in *spatial_obj2*.
spatial_obj2 is an ST_Spatial.

Return Type

HG_Center_In_3D returns an ST_Spatial.



Example

This example generates a circle in the testcon table and then centers it with a point when HG_Center_In_3D is called.

```
insert into testcon (sw_member, sw_geometry) values (
    67, ST_Spatial('HG_Circle(ST_Point(1,1,1), 10')')
)
exec sp_spatial_query '
    select HG_GetString(HG_Center_In_3D(
        sw_geometry, ST_Spatial(''ST_Point(2,2,2)'')
    ))
    from testcon
'
```

HG_Centroid

HG_Centroid returns the centroid of a given geometry. The centroid is a point representing the weighted center of a shape.

Input	Output
 object	 Point representing the weighted center.

Syntax

HG_Centroid(*spatial_obj*)
spatial_obj is an ST_Spatial.

Return Type

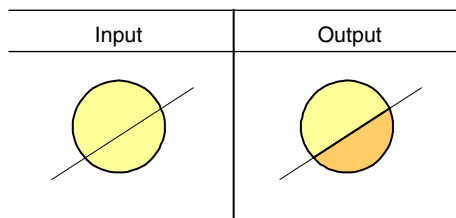
ST_Centroid returns an ST_Spatial representing an ST_Point.

Example

```
exec sp_spatial_query '
    select sw_member, HG_GetString(HG_Centroid(sw_geometry))
    from lake
'
```

HG_Clean

HG_Clean takes an object as input and creates a topologically correct geometry.



Refer also to *HG_Clean_I* and *HG_Clean_S*, which use topology, control, and filter values.

Syntax

`HG_Clean(spatial_obj)`

spatial_obj is an ST_Spatial.

Return Type

HG_Clean returns an ST_Spatial.

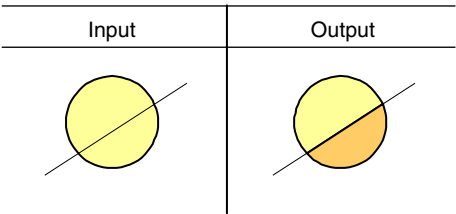
Example

```
exec sp_spatial_query '
    select HG_GetString(HG_Clean(sw_geometry))
    from testcon
    where sw_member=1
    ,
```

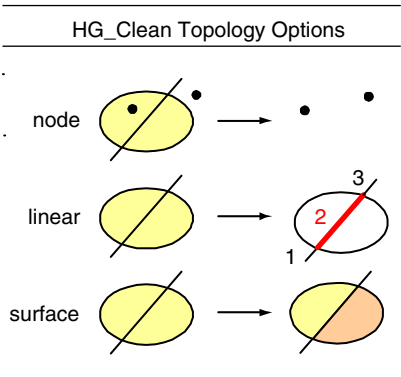
HG_Clean_I

HG_Clean_I takes an object as input and creates a topologically correct geometry. Topology and control parameters are specified as integer values:

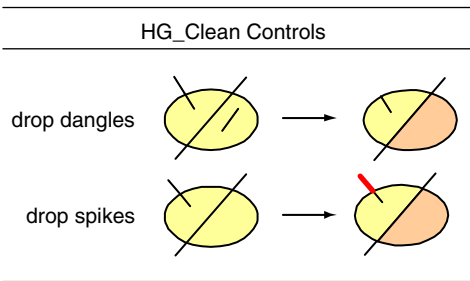
Output Topology Type	Integer Value
undefined	-1
node	0
linear	1
surface	2
Output Control Type	Integer Value
discard	0
no aggregates	1
drop spikes	2
drop dangles	3



The following diagrams show the expected behavior with basic shape types when specifying topology and control values. The right side of the arrow is the resulting geometry. The lines could be any curve geometry: circular arc, polyline, curve, or circle. The following illustrate results for node, linear, and surface output topology options:



The following output controls work on surfaces only: drop spikes, and drop dangles. They will have no effect if set with node or linear output topology options.



By default, the clean functions set the current topology to be 'surface'. If a linear (polyline) is given as input, then an unclosed surface is returned. In this case, the direction of the polyline is not preserved. If the topology is set to be 'linear' or 'auto-detected' then the output would be a linear object with direction preserved.

This function is the same as HG_Clean_S except that it accepts topology and control types as integer values instead of strings. Refer also to HG_Clean.

Syntax

HG_Clean_I(*spatial_obj*, *topology*, *control*, *filter*)

spatial_obj is an ST_Spatial.

topology is the topological type as an integer.

control is the control type to build topology as an integer.

filter is the filter tolerance, specified in double precision database units.

Return Type

HG_Clean_I returns an ST_Spatial.

Example

```
exec sp_spatial_query '
    select HG_GetString(HG_Clean_I(sw_geometry, 2, 3, 2.0))
    from testcon
    where sw_member=1
    ,
```

HG_Clean_S

HG_Clean_S takes an object as input and creates a topologically correct geometry. Topology and control parameters are specified as strings:

Output Topology Type	Input String
undefined node linear surface	'undef' 'node' 'linear' 'surface'
Output Control Type	Input String
discard no aggregates drop spikes drop dangles	'discard' 'no_agg' 'drop_spikes' 'drop_dangles'

This function is the same as HG_Clean_I except that it accepts topology and control types as strings instead of as integer values. Please refer to HG_Clean_I for a more detailed description of what is returned by these two functions.

Refer also to the related function HG_Clean.

Syntax

HG_Clean_S(*spatial_obj*, *topology*, *control*, *filter*)

spatial_obj is an ST_Spatial.

topology is the topological type as a Boolean value (true or false).

control is the control type to build topology as a Boolean value (true or false).

filter is the filter tolerance, specified in double precision database units.

Return Type

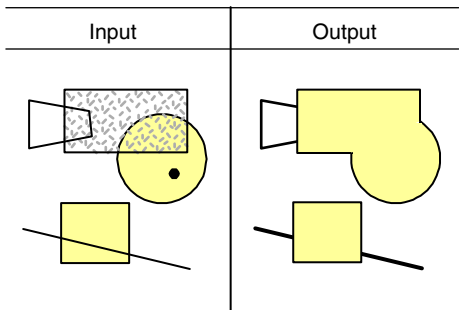
HG_Clean_S returns an ST_Spatial.

Example

```
exec sp_spatial_query '
    select HG_GetString(
        HG_Clean_S(sw_geometry, 'surface', 'drop_dangles', 2.0)
    )
    from testcon
    where sw_member=1
    ,
```

HG_Combine

HG_Combine is a union, it dissolves the boundaries in a spatial object. HG_Combine will drop any common primitives and form only one object as shown in the following image.



Syntax

HG_Combine(*spatial_obj*)

spatial_obj is a geometry collection of type ST_Spatial.

Return Type

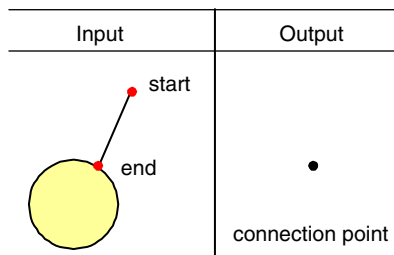
HG_Combine returns an ST_Spatial.

Example

```
exec sp_spatial_query '
    select HG_GetString(HG_Combine(sw_geometry))
    from testcon
    where sw_member=9
    ,
```

HG_Connect

HG_Connect returns the points of connection between a line (ST_Polyline) and an SW_GEOMETRY. Connection points occur when the boundary of the SW_GEOMETRY shares a point with either the start or end of the line (or both).



Syntax

HG_Connect(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is an ST_Spatial.

spatial_obj2 is an ST_Spatial representing an ST_Polyline.

Return Type

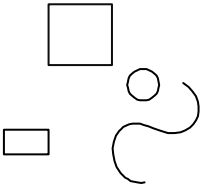
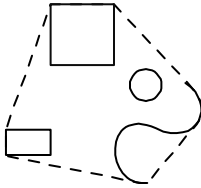
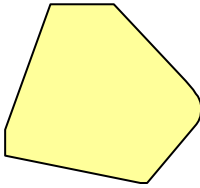
HG_Connect returns an ST_Spatial representing an ST_Point.

Example

```
exec sp_spatial_query '
    select HG_GetString(
        HG_Connect(flood100.sw_geometry, rdpaved.sw_geometry)
    )
    from flood100, rdpaved
    ,
```

HG_Convex_Hull

HG_Convex_Hull returns the minimum boundary around a spatial object without the boundary being concave. The result is the spatial object representing the convex hull of the perimeter.

Input	Convex Hull Operation	Output
 <p>Object containing multiple elements.</p>		 <p>convex hull</p>

Syntax

HG_Convex_Hull(*spatial_obj*)
spatial_obj is an ST_Spatial.

Return Type

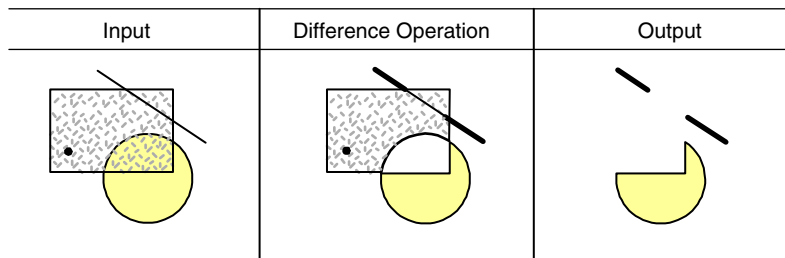
HG_Convex_Hull returns an ST_Spatial representing a ST_Polygon.

Example

```
exec sp_spatial_query '
    select HG_GetString(HG_Convex_Hull(sw_geometry))
    from testcon
    where sw_member=2
    ,
```


HG_Difference

HG_Difference removes those places where two spatial objects overlap. The point set difference is returned, (A-B).



The HG_Difference function is very similar to HG_Erase except that the cutter object does not need to be a closed surface, it could be a linear object.

Syntax

HG_Difference(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is an ST_Spatial.

spatial_obj2 is an ST_Spatial.

Return Type

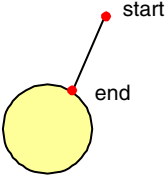

HG_Difference returns an ST_Spatial.

Example

```
exec sp_spatial_query '
select HG_GetString(
    HG_Difference(a.sw_geometry, b.sw_geometry)
)
from flood100 b, lake a
'
```

HG_End_Of

HG_End_Of returns the end point of a line if it matches a point in the boundary of an ST_Spatial.

Input	Output
	 end point

Syntax

HG_End_Of(*spatial_obj1*, *spatial_obj2*)
spatial_obj1 is an ST_Spatial.
spatial_obj2 is an ST_Spatial representing an ST_Polyline.

Return Type




HG_End_Of returns an ST_Spatial.

Example

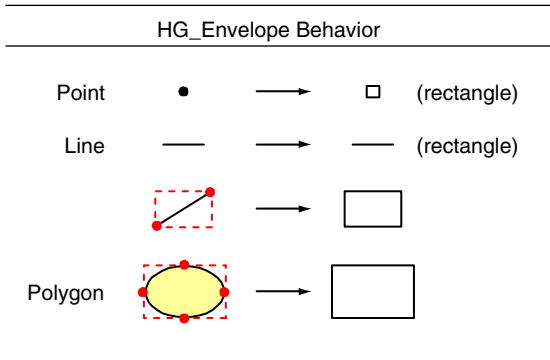
```
exec sp_spatial_query '  
  select HG_GetString(  
    HG_End_Of(a.sw_geometry, b.sw_geometry)  
  )  
  from flood100 a, rdpaved b  
,
```

HG_Envelope

HG_Envelope returns the smallest rectangle, orthogonal to the axes, that can contain a geometry. The rectangle is an ST_Polygon.

Input	Envelope Process	Output
		 minimum enclosing rectangle

The following diagram shows the behavior of HG_Envelope with the basic shape types. The right side of the arrow is the return object. The line type represents all curve objects: circular arc, polyline, curve, and circle.



Note: The rectangle surrounding the point and horizontal line is conceptual only. For example, the point rectangle's lower left point and upper right point are the same.

Syntax

HG_Envelope(*spatial_obj*)
spatial_obj is an ST_Spatial.

Return Type

HG_Envelope returns an ST_Spatial representing an ST_Polygon.

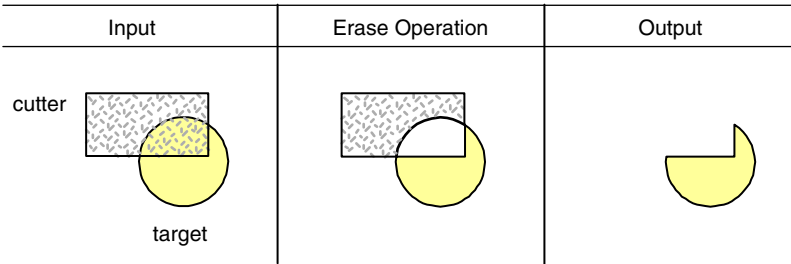
Example

This query retrieves the smallest rectangle that contains the given SW_GEOMETRY:

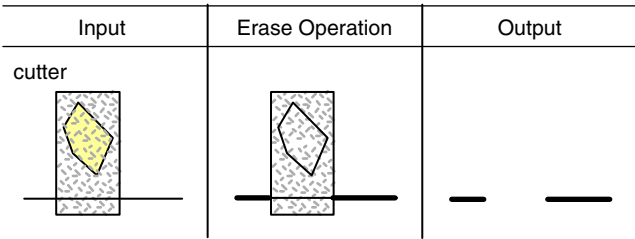
```
exec sp_spatial_query '  
    select lake.sw_member,  
           HG_GetString(HG_Envelope(lake.sw_geometry))  
    from lake  
'
```

HG_Erase

HG_Erase removes those places of a target object that overlap with a cutting object. Any places where the target object overlaps with the cutter object are removed. HG_Erase produces the opposite result to HG_Erase_Outside.



HG_Erase performs a cutting operation; one spatial object is used as the target object to be cut, and another spatial object is used as the cutter. A new geometry is generated as output. Those places where the cutter object intersect with the target object are removed. If the cutter object does not intersect with the target object, then the target object remains unchanged. If the cutter object contains the target object, then the target object is removed from the result. If using a binary relationship, where object A is the target and object B is the cutter, results in object B are subtracted from object A. For example:



Note that the polygon was completely contained within the cutter object and has been removed from the result. Although the linear object has been cut into multiple pieces, the result is a single region object.

The HG_Erase function is very similar to HG_Difference except that the cutter object must be a closed surface (a closed polygon). The Difference function accepts a cutter object that is not closed, such as a linear.

Syntax

`HG_Erase(spatial_obj1, spatial_obj2)`
spatial_obj1 is the target ST_Spatial.
spatial_obj2 is the cutter ST_Spatial (a closed polygon).

Return Type

HG_Erase returns an ST_Spatial.

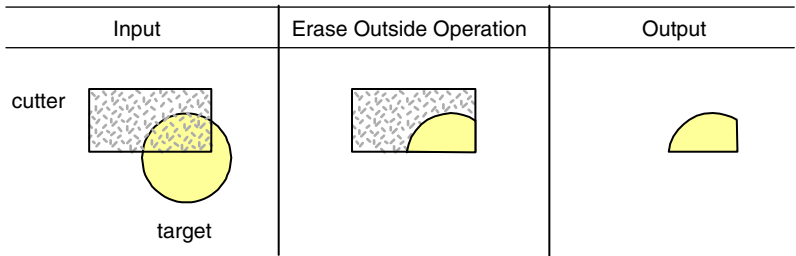
Example

```
exec sp_spatial_query '
    select HG_GetString(
        HG_Erase(flood100.sw_geometry, lake.sw_geometry)
    )
    from flood100, lake
    ,
```

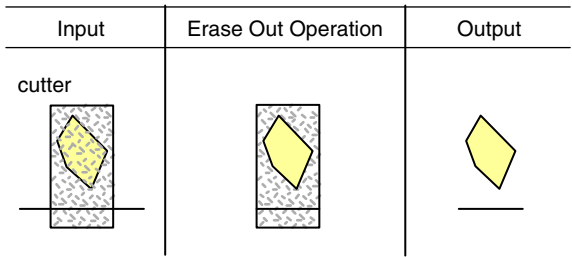
HG_Erase_Outside

HG_Erase_Outside removes those places of a target object that do not overlap with a cutting object. HG_Erase_Outside produces the opposite result to HG_Erase on page 109.

The cutter object must be a closed surface (a closed polygon).



A set of objects are target objects, and another set of objects are cutter objects. The portion of a target object, which overlaps with the cutter object is retained, and all else is discarded. A new geometry is generated as output. If the target object is contained within the cutter object then it remains in the result unchanged. If the target object does not intersect with the cutting object, then it is removed from the result. Using a binary relationship where object A is the target, and object B is the cutter, results in A intersected with B. For example:



Note: The polygon was completely contained within the cutter object and remains unchanged in the result. Those portions of the linear object that lie outside of the cutter object are not included within the result.

Syntax

HG_Erase_Outside(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is the target ST_Spatial.

spatial_obj2 is the cutter ST_Spatial (a closed polygon).

Return Type

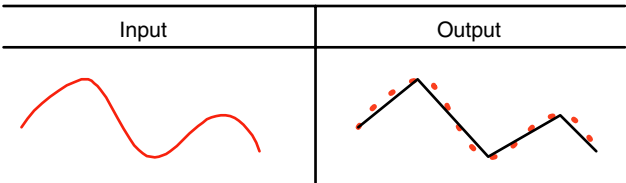
HG_Erase_Outside returns an ST_Spatial.

Example

```
exec sp_spatial_query '
    select HG_GetString(
        HG_Erase_Outside(drain.sw_geometry, flood100.sw_geometry)
    )
from drain, flood100
'
```

HG_Filter

HG_Filter returns a filtered line strings, either a polyline (ST_Polyline), or curve (HG_Curve). Often, the detail of the given line may be more than is required, making processing inefficient. HG_Filter allows you to reduce the detail to a desired level, by specifying a filter tolerance.



Points falling under the filter tolerance will be ignored, forming a simplified curve, more like the second (above). HG_Filter performs the following on the input object:

- Creates a temporary straight line from the start point to the end point of the object.
- Determines if any points in the object have a distance from the temporary straight line that exceeds the filter tolerance.

If no, then all points between the start and end points are ignored and a two point line is created from start to end.

If yes, then two lines are created, start-to-point, and point-to-end. The process continues recursively until no more points fall outside the filter tolerance.

The filter operation uses the Douglas-Puecker algorithm (The Canadian Cartographer, Vol.10 No.2, Dec. 1973, pp.112-122), to reduce the number of points to represent a line.

Syntax

`HG_Filter(spatial_obj, tolerance)`

spatial_obj is an ST_Spatial representing an ST_Polyline or HG_Curve.

tolerance is the tolerance level used in the filter calculation, in double precision database units.

Return Type

HG_Filter returns an ST_Spatial representing an ST_Polyline or HG_Curve.

Example

```
exec sp_spatial_query '  
    select HG_GetString(HG_Filter(lake.sw_geometry, 0.1))  
    from lake  
,
```

HG_Filter_Curves

HG_Filter_Curves returns all of the curves from within a spatial object of type SW_GEOMETRY.

Syntax

`HG_Filter_Curves(spatial_obj)`

spatial_obj is an ST_Spatial.

Return Type

HG_Filter_Curves returns an ST_Spatial containing ST_Polyline.

Example

```
exec sp_spatial_query '  
    select HG_GetString(HG_Filter_Curves(sw_geometry))
```

```
from testcon  
,
```

HG_Filter_Paths

HG_Filter_Paths returns all of the paths within a spatial object of type SW_GEOMETRY.

Syntax

HG_Filter_Paths(*spatial_obj*)

spatial_obj is an ST_Spatial.

Return Type

HG_Filter_Paths returns an ST_Spatial representing an ST_Polyline.

Example

```
exec sp_spatial_query '  
select HG_GetString(HG_Filter_Paths(sw_geometry))  
from testcon  
,
```

HG_Filter_Points

HG_Filter_Points returns all of the points from within a spatial object of type SW_GEOMETRY.

Syntax

HG_Filter_Points(*spatial_obj*)

spatial_obj is an ST_Spatial.

Return Type

HG_Filter_Points returns an ST_Spatial representing a ST_Point.

Example

```
exec sp_spatial_query '  
select HG_GetString(HG_Filter_Points(sw_geometry))  
from testcon  
,
```


HG_Filter_Polygons

HG_Filter_Polygons returns all of the polygons from within a spatial object.

Syntax

HG_Filter_Polygons(*spatial_obj*)

spatial_obj is an ST_Spatial.

Return Type

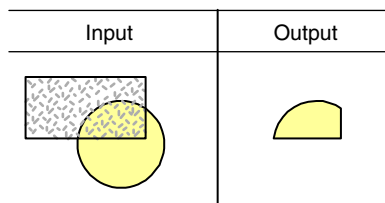
HG_Filter_Polygons returns an ST_Spatial representing an ST_Polygon.

Example

```
exec sp_spatial_query '
    select HG_GetString(HG_Filter_Polygons(sw_geometry))
    from testcon
    ,
```

HG_Intersect_In

HG_Intersect_In is identical to ST_Overlap. It returns all of the overlapping elements between two spatial objects. Two elements are overlapping when they share common points. If there are no overlapping elements, then NULL is returned.



Syntax

HG_Intersect_In(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is an ST_Spatial.

spatial_obj2 is an ST_Spatial.

Return Type

HG_Intersect_In returns an ST_Spatial.

Example

```
exec sp_spatial_query '
    select HG_GetString(
        HG_Intersect_In(flood100.sw_geometry, lake.sw_geometry)
    )
    from flood100, lake
    ,
```

HG_LL_Circle

HG_LL_Circle accepts the Longitude/Latitude of a center point and a radial distance in meters, and creates a geometry representing a search rectangle. The rectangle is a good approximation of a circle in Longitude/Latitude.

This function provides for the curvature of the Earth and takes into account the non-uniformity of the lat/long coordinate system. It uses a spherical model of Earth to generate the output polygon. It is particularly useful when the search area is large, straddles UTM zones, or is located at relatively high (or low) latitudes.

All the vertices of the polygon generated by this function will be at a spherical distance (refer to HG_SphericalDist on page 53) of <dist_meters> from the <longitude> <latitude> point.

Note: This is a more precise circle for radial search on lat/long points than a circular region generated by either, ST_Spatial('ST_Polygon(HG_Circle(x, y, radius))') or ST_Buffer(ST_Spatial('ST_Point(x, y)', dist, filter)

Syntax

HG_LL_Circle(*longitude, latitude, dist_meters*)

longitude is the longitude of the center point (x value), in degrees.

latitude is the latitude of the center point (y value), in degrees.

dist_meters is the radius of the circle to be generated in meters.

Return Type

HG_LL_Circle returns an ST_Spatial.

Example

The following example uses the World sample database. Please refer to your product Release Notes (or readme file) for a description of how to install this sample database.

This example finds all the capitals that lie within a spherical distance of 1000 kilometers from Paris.

```
exec sp_spatial_query '
select a.capital, HG_SphericalDist(
    a.sw_geometry,b.sw_geometry
) as dist_meters
from worldcap a, worldcap b
where b.capital = 'Paris'
and ST_Overlaps(a.sw_geometry, HG_LL_Circle(
    ST_x(b.sw_geometry), ST_y(b.sw_geometry), 1000000
))
'
```

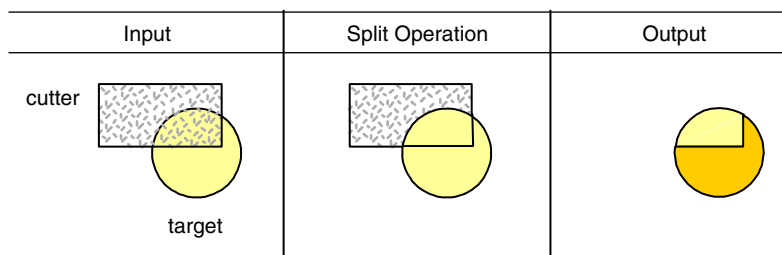
Caveat

This function generates an *approximation* of a circular area in the lat/long coordinate system. It assumes a spherical model of Earth, and therefore may be unsuitable for applications where a high degree precision is needed. For such applications, it may be more appropriate to transform the point to a suitable projected coordinate system, perform a circular buffer, and then transform it back to Longitude/Latitude.

This function does not handle regions straddling the lat/long limits. For example, it will not handle regions straddling the international date line.

HG_Split

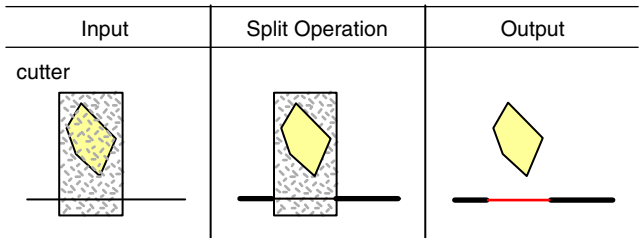
HG_Split splits those places of a target object that overlap the cutting object.



HG_Split combines the results of both the HG_Erase and HG_Erase_Outside functions. One object is used as the target object and a second object is used as the cutting object. A new geometry is generated as output.

The target object is compared to the cutter object. If the target and cutter objects intersect, then the result will contain one object which represents the result as if an erase operation was

performed, and another object which represents the result as if an erase outside operation was performed. If a target object is completely contained within a target cutter or completely outside of it, then it will remain in the result unchanged. For example:



Note that the polygon was completely contained within the cutter object and remains unchanged in the result. Although the linear object has been cut into multiple pieces, the result is a single region object.

The cutter object must be a closed surface (a closed polygon).

Syntax

HG_Split(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is the target ST_Spatial.

spatial_obj2 is the cutter ST_Spatial (a closed polygon).

Return Type

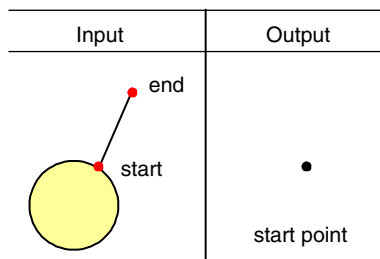
HG_Split returns an ST_Spatial.

Example

```
exec sp_spatial_query '  
    select HG_GetString(  
        HG_Split(flood100.sw_geometry, lake.sw_geometry)  
    )  
    from lake, flood100  
'
```

HG_Start_Of

HG_Start_Of returns the start point of a line if it matches a point in the boundary of an ST_Spatial.



Syntax

`HG_Start_Of(spatial_obj1, spatial_obj2)`

spatial_obj1 is any ST_Spatial.

spatial_obj2 is an ST_Spatial representing an ST_Polyline.

Return Type

HG_Start_Of returns an ST_Spatial.

Example

```
exec sp_spatial_query '
  select HG_GetString(
    HG_Start_Of(lake.sw_geometry, flood100.sw_geometry))
  from flood100, lake
  '
```

HG_Sym_Difference

HG_Sym_Difference performs the symmetric difference (x-or) on the parameters ((A-B) Union (B-A)).

Syntax

`HG_Sym_Difference(spatial_obj1, spatial_obj2)`

spatial_obj1 is an ST_Spatial.

spatial_obj2 is an ST_Spatial.

Return Type

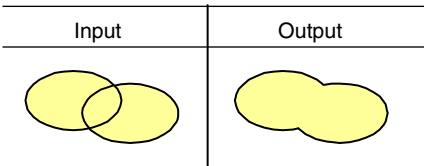
HG_Sym_Difference returns an ST_Spatial.

Example

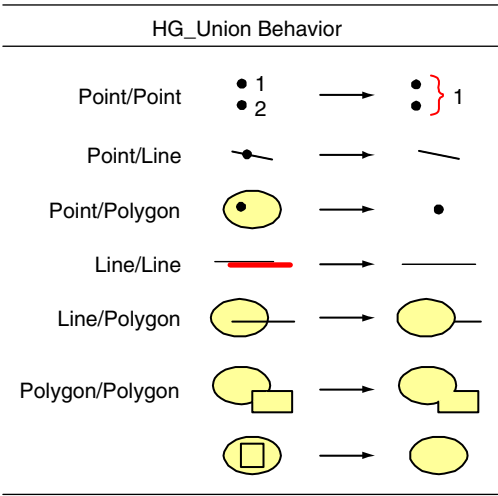
```
exec sp_spatial_query '
    select HG_GetString(
        HG_Sym_Difference(lake.sw_geometry, flood100.sw_geometry))
    from flood100, lake
    ,
```

HG_Union

HG_Union combines two geometries, and returns the union.



The following diagram shows the behavior of HG_Union with the basic shape types. The right side of the arrow is the return object. In the illustration, Line represents all Curve objects: ST_CircularArc, ST_Polyline, HG_Curve, and HG_Circle.



Collections of objects follow these basic examples. For more information on how complex collections of objects are treated in SpatialWare, see Understanding Geometry Collections.

Syntax

```
HG_Union(spatial_obj1, spatial_obj2)
```

spatial_obj1 is an ST_Spatial.
spatial_obj2 is an ST_Spatial.

Return Type

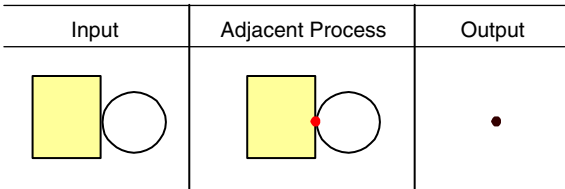
HG_Union returns an ST_Spatial.

Example


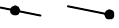
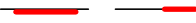

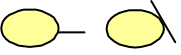
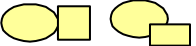
```
exec sp_spatial_query '
    select HG_GetString(
        HG_Union(lake.sw_geometry, flood100.sw_geometry))
    from lake, flood100
    where lake.lake_cm = 17400001
    and flood100.flood100_cm = 39300021
    ,
```

ST_Adjacent

ST_Adjacent returns a geometry made up of the points of intersection and common line segments between two spatial objects. NULL is returned if the two spatial objects have no common values, or if the two objects share interior points.



The following diagram shows the basic valid adjacency conditions. Note that polygons can only share their boundary points with other shapes. If another shape encroaches a polygon's interior, then it is no longer adjacent.

Adjacency Conditions	
Point/Point	
Line/Point	
Line/Line	
Polygon/Point	
Polygon/Line	
Polygon/Polygon	

Syntax

ST_Adjacent(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is an ST_Spatial.

spatial_obj2 is an ST_Spatial.

Return Type


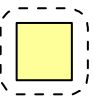

ST_Adjacent returns an ST_Spatial representing an ST_Point.

Example

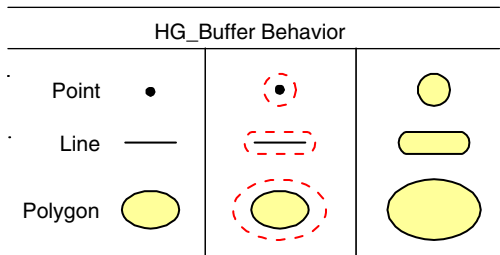
```
exec sp_spatial_query '  
    select HG_GetString(  
        ST_Adjacent(flood100.sw_geometry, publdg.sw_geometry))  
    from flood100, publdg where flood100.sw_member=2  
'
```

ST_Buffer

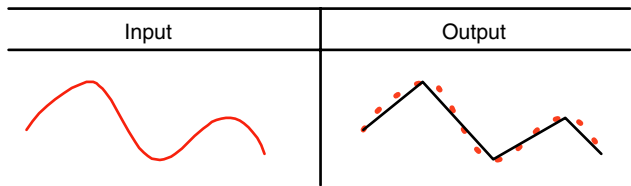
ST_Buffer takes a geometry, buffer distance, and filter tolerance. The filter parameter specifies a filter tolerance on the buffer creation process.

Input	Buffer Process	Output
		

The following diagram shows the behavior of ST_Buffer with basic shape types. The object on the right side of the arrow is the return object. The Line shape type represents all curve objects: ST_CircularArc, ST_Polyline, HG_Curve, and HG_Circle.



Filter tolerance allows you to control the buffer's detail level. By default, ST_Buffer creates a buffer with the same detail as the input object. Detail is represented by the number of points in the boundary of the object.



ST_Buffer then uses the simplified curve as input and creates a buffer.

Syntax

ST_Buffer(*spatial_obj1*, *width*, *filter*)

spatial_obj is an ST_Spatial.

width is the size of the buffer, in double precision database units.

filter is the filter tolerance, specified in double precision database units, possibly NULL.

Return Type

ST_Buffer returns an ST_Spatial representing an ST_Polygon.

Example

The following query creates a buffer 66 feet around a lake.

```
exec sp_spatial_query '
select sw_member, HG_GetString(
    ST_Buffer(SW_GEOMETRY, 66.0, 0.1))
```

```

    from lake
    ,
    select sw_member, ST_Buffer(sw_geometry, 66.0, 0.1)
    from lake;
```

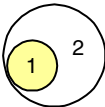

Caveat

Please be aware that ST_Buffer may produce surprising results in the Longitude/Latitude coordinate system. The MapInfo GIS Extension assumes a Cartesian coordinate system when performing this function, and performs no adjustment to account for where the coordinate is on the Earth.

In the Longitude/Latitude coordinate system, buffer units are in the units of Longitude/Latitude. A "unit" is not defined well in Longitude/Latitude, as it changes depending on where you are in the world. Moreover, a "unit" means different distances on x and y directions.

ST_Contain

ST_Contain returns geometry1 if it is entirely contained within geometry2 and NULL if it is not. Boundaries can touch, but the inner object cannot have any points outside the containing boundary.

Input	Output
	 Geometry 1 is returned, because it is entirely contained within geometry 2.

Syntax

ST_Contain(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is an ST_Spatial.

spatial_obj2 is an ST_Spatial.

Return Type

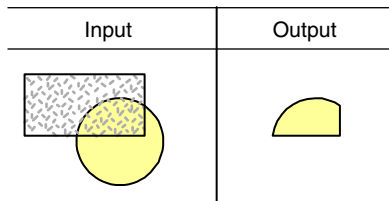
ST_Contain returns an ST_Spatial.

Example

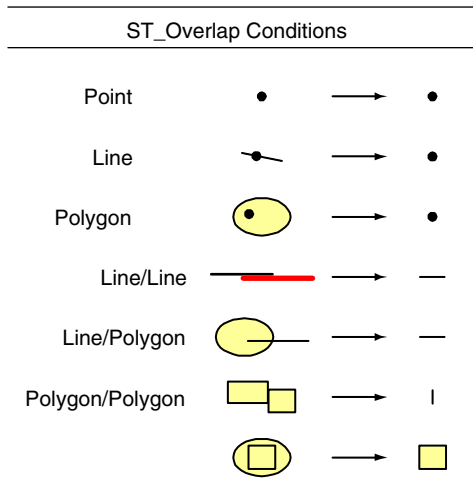
```
exec sp_spatial_query '
    select HG_GetString(
        ST_Contain(pubbldg.sw_geometry, flood100.sw_geometry))
    from pubbldg, flood100
    ,
```

ST_Overlap

ST_Overlap returns all of the overlapping elements between two spatial objects. Two elements are overlapping when they share common points. If there are no overlapping elements, then a NULL is returned.



The following diagram shows the behavior of ST_Overlap with the basic geometries. The output geometry is shown on the right side of the arrow. The line type represents all curve geometries: circular arc, polyline, curve, and circle. Note that this diagram can also be useful for understanding ST_Overlaps. The basic conditions below that return an object, will also return true with ST_Overlaps.



Syntax

`ST_Overlap(spatial_obj1, spatial_obj2)`

spatial_obj1 is an ST_Spatial.

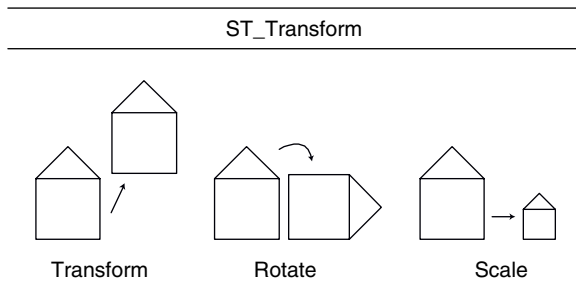
spatial_obj2 is an ST_Spatial.

Return Type

`ST_Overlap` returns an ST_Spatial.

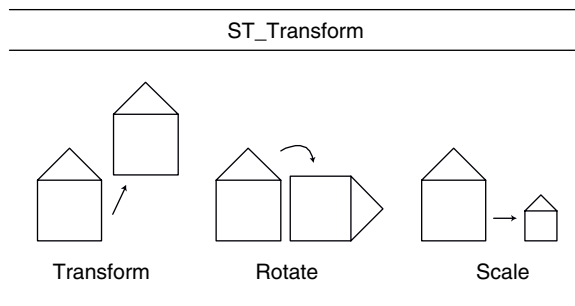
Example

```
exec sp_spatial_query '  
    select HG_GetString(ST_Overlap(a.sw_geometry, b.sw_geometry))  
    from flood100 a, lake b  
'
```



ST_Transform

`ST_Transform` moves, rotates, or scales two-dimensional geometry using a transformation matrix (`sx, tx, rx, sy, ty, ry`), where `sx` is the x factor and `x` is a scale factor, translation (transform) in the x-direction, and `rx` is rotation (in degrees) about the x axis. The `sy, ty, ry` values behave in the same way for the y direction.



ST_Transform requires an *origin* parameter, which specifies the point of rotation. Rotation is in degrees. If there is no rotation in your transformation then a NULL value can be used for the origin parameter, the default rotation point is (0,0).

All transformations can be performed at the same time, but it is not necessary to specify all three. For example, if you only want to rotate a geometry, include zeros in the matrix for scale and transform. You can set the following:

- To specify no scaling use 1.0 for x or y.
- To specify no transform use 0.0 for x or y.
- To specify no rotation use 0.0 for x or y.

For information on how the transformation is calculated, refer to Transformation Calculation.

Refer also to ST_Transform3D on page 128 to transform a three-dimensional object.

Syntax

ST_Transform(*spatial_obj*, *origin*, *sx*, *tx*, *rx*, *sy*, *ty*, *ry*)

spatial_obj is an ST_Spatial.

origin is the point of origin, an ST_Spatial.

sx and *sy* are scale in x and y.

rx and *ry* are rotation in x and y.

tx and *ty* are transformation in x and y.

All values are in database units.

Return Type

ST_Transform returns an ST_Spatial.

Usage

Use this function with two-dimensional geometry.

Example

This example scales a box to create a rectangle. The point of origin is (0,0). The scale values are 2 along the x axis, and 3 along the y axis.

```
exec sp_spatial_query '  
    select HG_GetString(ST_Transform(  
        ST_Spatial('HG_Box(1, 1, 2, 2)'),  
        ST_Spatial('ST_Point(0, 0)'),  
        2.0, 0.0, 0.0, 3.0, 0.0, 0.0))  
    from dual  
,
```

This example moves a box along the x and y axis. The point of origin is (0,0). The transform values are 3 along the x axis, and -3 along the y axis.

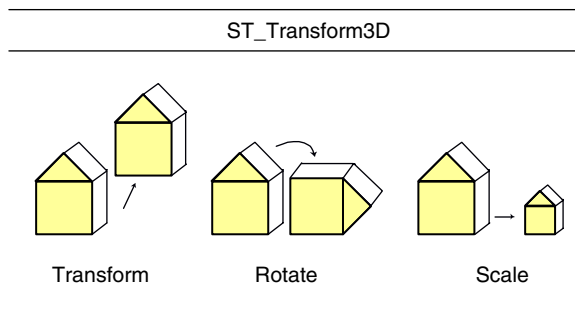
```
exec sp_spatial_query '  
    select HG_GetString(ST_Transform(  
        ST_Spatial('HG_Box(1, 1, 2, 2)'),  
        ST_Spatial('ST_Point(0, 0)'),  
        0.0, 3.0, 0.0, 0.0, -3.0, 0.0))  
    from dual  
,
```

This example rotates a box around the point of origin by 35 degrees. The point of origin is (0,0). The rotation values are 35 for x, and 20 for y.

```
exec sp_spatial_query '  
    select HG_GetString(ST_Transform(  
        ST_Spatial('HG_Box(1, 1, 2, 2)'),  
        ST_Spatial('ST_Point(0, 0)'),  
        0.0, 0.0, 35.0, 0.0, 0.0, 20.0  
    ))  
    from dual  
,
```

ST_Transform3D

ST_Transform moves, rotates, or scales three-dimensional geometry using a transformation matrix (sx, tx, rx, sy, ty, ry, sz, tz, rz), where sx is the x scale factor, tx is translation (transform) in the x-direction, and rx is rotation (in degrees) about the axis. The sy, ty, ry, and sz, tz, rz values behave in the same way in the y and z directions.



ST_Transform3D requires an origin parameter, which specifies the point of rotation. Rotation is in degrees. If there is no rotation in your transformation then a NULL value can be used for the origin parameter, the default rotation point is (0,0,0).

All transformations can be performed at the same time, but it is not necessary to specify all three. For example, if you only want to rotate a geometry, include zeros in the matrix for scale and transform. You can set the following:

- To specify no scaling use 1.0 for x, y, or z.
- To specify no transform use 0.0 for x, y, or z.
- To specify no rotation use 0.0 for x, y, or z.

For information on how the transformation is calculated see Transformation Calculation.

Syntax

ST_Transform3D(*spatial_obj*, *origin*, *sx*, *tx*, *rx*, *sy*, *ty*, *ry*, *sz*, *tz*, *rz*)

spatial_obj is an ST_Spatial.

origin is the point of origin, an ST_Spatial.

sx, *sy*, and *sz* are scale in x, y, and z.

rx, *ry*, and *rz* are rotation in x, y, and z.

tx, *ty*, and *tz* are transformation in x, y, and z.

All values are in database units.

Return Type

ST_Transform3D returns an ST_Spatial.

Usage

Use this function with three-dimensional geometry.

Example

This example scales a box to create a rectangle. The point of origin is (0,0,0). The scale values are 3 along the x axis, 2 along the y axis, and 4 along the z axis.

```
exec sp_spatial_query '  
    select HG_GetString(ST_Transform3D(  
        ST_Spatial('HG_Box(1, 1, 2, 2)'),  
        ST_Spatial('ST_Point(0, 0, 0)'),  
        3, 0, 0, 2, 0, 0, 4, 0, 0  
    ))  
    from dual  
,
```

This example moves a box along the x, y, and z axis. The point of origin is (0,0,0). The transform values are 3 along the x axis, -3 along the y axis, and 2 along the z axis.

```
exec sp_spatial_query '  
    select HG_GetString(ST_Transform3D(  
        ST_Spatial('HG_Box(1, 1, 2, 2)'),  
        ST_Spatial('ST_Point(0, 0, 0)'),  
        0, 3, 0, 0, -3, 0, 0, 2, 0  
    ))  
    from dual  
,
```

This example rotates a box around the point of origin by 35 degrees. The point of origin is (0,0,0). The rotation values are 35 for x, 20 for y, and 40 for z.

```
exec sp_spatial_query '  
    select HG_GetString(ST_Transform3D(  
        ST_Spatial('HG_Box(1, 1, 2, 2)'),  
        ST_Spatial('ST_Point(0, 0, 0)'),  
        0, 0, 35, 0, 0, 20, 0, 0, 40  
    ))  
    from dual  
,
```


Transformation Calculation

The ST_Transform function performs two dimensional transformation and ST_Transform3D performs a three dimensional transformation using a textbook transformation calculations. The calculations are described below.

There are two matrix types used to perform a transformation, 2x3 or 3x3:

2x3 Matrix = '(sx, tx, rx, sy, ty, ry)' where rx and ry must be equal.

3x3 Matrix = '(sx, tx, rx, sy, ty, ry, sz, tz, rz)'.

Two Dimensional Calculation

In a two dimensional transformation only sx, tx, rx, sy, ty, ry are used with the constraint that $rx == ry$:

sx is the Scale in the X-axis,
tx is the Displacement in the X-axis,
sy is the Scale in the Y-axis,
ty is the Displacement in the Y-axis, and
 $rx = ry$ is the rotation in the xy planes in degrees.

In two dimensions, sz, tz, and rz have no meaning and must be initialized to 0.0 (or sz may be 1.0).

The effective algorithm is:

1. If the source point to be transformed is $p = [x, y]$ then let $PV = [x, y, 1]$.
2. If rotation point $r = [rotx, roty]$ is specified, then let $RV = [rotx, roty, 0]$.
3. Otherwise, let $RV = [0, 0, 0]$.
set $MA = [[\cos(rx), -\sin(rx), tx], [\sin(rx), \cos(rx), 0], [0, 0, 1]]$
4. Set $MB = [[sx, 0, tx], [0, sy, ty], [0, 0, 1]]$
5. Let $TV = [x', y', 1]$ be the result of the matrix operation:
 $TV = ((PV - RV) * MA * MB) + RV$
6. Let the transformed point, p' be $[x', y']$.

Three Dimensional Calculation

In a three dimensional transformation all nine values are used, where:

sx is the Scale in the X-axis,
tx is the Displacement in the X-axis
rx is the rotation in the yz plane,

sy is the Scale in the Y-axis,
ty is the Displacement in the Y-axis,
ry is the rotation in the xz plane,
sz is the Scale in the Y-axis,
tz is the Displacement in the Y-axis, and
rz is the rotation in the xy planes in degrees.

The effective algorithm is:

1. If the source point to be transformed is $p = [x, y, z]$ then let $PV = [x, y, z, 1]$.
2. If rotation point $r = [rotx, roty, rotz]$ is specified, then let $RV = [rotx, roty, rotz, 0]$; otherwise, let $RV = [0, 0, 0, 0]$.
3. Set $MA = [[1, 0, 0, 0], [0, \cos(rx), -\sin(rx), 0], [0, \sin(rx), \cos(rx), 0], [0, 0, 0, 1]]$ (yz plane rotation).
4. Set $MB = [[\cos(ry), 0, -\sin(ry), 0], [0, 1, 0, 0], [\sin(ry), 0, \cos(ry), 0], [0, 0, 0, 1]]$ (xz plane rotation).
5. Set $MC = [[\cos(rz), -\sin(rz), 0, 0], [\sin(rz), \cos(rz), 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]]$ (xy plane rotation).
6. Set $MD = [[sx, 0, 0, tx], [0, sy, 0, ty], [0, 0, sz, tz], [0, 0, 0, 1]]$.
7. Let $TV = [x', y', z', 1]$ be the result of the matrix operation:
$$TV = ((PV - RV) * MA * MB * MC * MD) + RV$$
8. Let the transformed point, p' be $[x', y', z']$.

Spatial Predicates

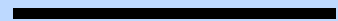
Spatial Predicates analyze geometries against specific conditions. These functions return 'TRUE' or 'FALSE' values and are generally used within a WHERE clause.



10

Chapter

► Function Descriptions



Function Descriptions

Spatial Predicates include:

HG_Above	HG_Is_Empty	HG_Is_Znull
HG_Assembled	HG_Is_Forward	HG_Level
HG_At_End_Of	HG_Is_HG_Curve	ST_Adjacent_To
HG_At_Start_Of	HG_Is_Invalid	ST_Contained_By
HG_Below	HG_Is_Nulldir	ST_Contains
HG_Connected_To	HG_Is_Path	ST_Equals
HG_Identical	HG_Is_Point	ST_Meets
HG_Is_Box	HG_Is_Polygon	ST_Not_Equals
HG_Is_Circle	HG_Is_Polyline	ST_Outside
HG_Is_CircularArc	HG_Is_Quad	ST_Overlaps
HG_Is_Closed	HG_Is_Reverse	ST_Within
HG_Is_Contiguous	HG_Is_Triangle	
HG_Is_Curve	HG_Is_Valid	

HG_Above

HG_Above returns TRUE if point1 has a higher z coordinate value than point2. In heterogeneous geometries, the z ordinate of the first point geometry defines the level. If both points have the same z ordinate values, neither point is higher than the other, then HG_Above returns FALSE.

This function only works with point geometry. The results are only relevant if the input points are in three dimensions (if they have z values).

Syntax

HG_Above(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is an ST_Spatial representing a three-dimensional point (ST_Point).

spatial_obj2 is an ST_Spatial representing a three-dimensional point (ST_Point).

Return Type

HG_Above returns a Boolean value.

Example

```
exec sp_spatial_query '
select sw_member from spotelev
where HG_Above(
    spotelev.sw_geometry,
    ST_Spatial(''ST_Point(5.0, 5.0, 5.0)'')
)
```

HG_Assembled

This function returns TRUE if the input ST_Spatial is assembled. All ST_Spatials contain the assembled attribute, which can be used by applications that require it. For example, an application may set Assembled to TRUE when a shape is created. If the object is manipulated at a later time by the application, the flag will be set to FALSE until it can be validated.

Syntax

HG_Assembled(*spatial_obj*)

spatial_obj is an ST_Spatial.

Return Type

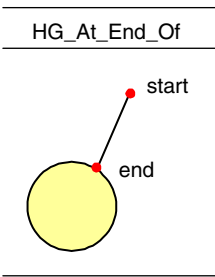
HG_Assembled returns a Boolean value.

Example

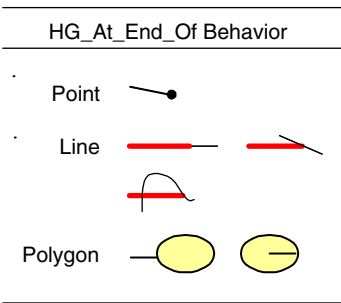
```
exec sp_spatial_query '
select sw_member from rd paved where HG_Assembled(sw_geometry)
```

HG_At_End_Of

HG_At_End_Of returns TRUE if the end point of the line matches a point in the ST_Spatial's boundary.



The following diagram shows the behavior of HG_At_End_Of with the basic shape types. The Line type represents all curve objects: ST_CircularArc, ST_Polyline, HG_Curve, and HG_Circle. The end of the line is the right end.



Collections of objects follow these basic examples.

Syntax

HG_At_End_Of(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is an ST_Spatial.

spatial_obj2 is an ST_Spatial representing an ST_Polyline.

Return Type

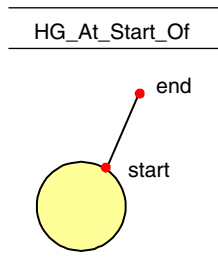
HG_At_End_Of returns a Boolean value.

Example

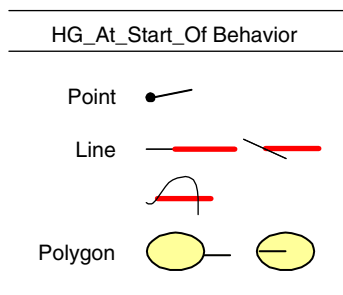
```
exec sp_spatial_query '
  select a.sw_member, b.sw_member
  from rdpaved a, rdpaved b
  where HG_At_End_Of(a.sw_geometry,b.sw_geometry)
  and a.sw_member = 28
'
```

HG_At_Start_Of

HG_At_Start_Of returns TRUE if the start point of a line matches a point in the boundary of an object.



The following diagram shows the behavior of HG_At_Start_Of with the basic shape types. HG_At_Start_Of returns a Boolean value (of TRUE or FALSE). The following diagram illustrates the behavior of HG_At_Start_Of with the basic valid shapes. All cases below return TRUE. For the diagram, the start point is always the left end of the line. Collections of objects follow these basic examples.



Syntax

HG_At_Start_Of(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is an ST_Spatial.

spatial_obj2 is an ST_Spatial representing an ST_Polyline.

Return Type

HG_At_Start_Of returns a Boolean value.

Example

```
exec sp_spatial_query '
    select a.sw_member, b.sw_member
    from rd paved a, rd paved b
    where HG_At_Start_Of(a.sw_geometry, b.sw_geometry)
    and a.sw_member = 30
    ,
```

HG_Below

HG_Below returns TRUE if *point1* has a lower z coordinate value than *point2*. In heterogeneous geometries, the z ordinate of the first point geometry defines the level. If both points have the same z ordinate values, neither point is higher than the other, then HG_Below returns FALSE.

This function only works with point geometry. The results are only relevant if the input points are in three dimensions (if they have z values).

Syntax

HG_Below(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is an ST_Spatial representing a three-dimensional ST_Point.

spatial_obj2 is an ST_Spatial representing a three-dimensional ST_Point.

Return Type

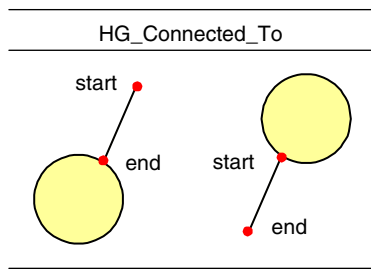
HG_Below returns a Boolean value.

Example

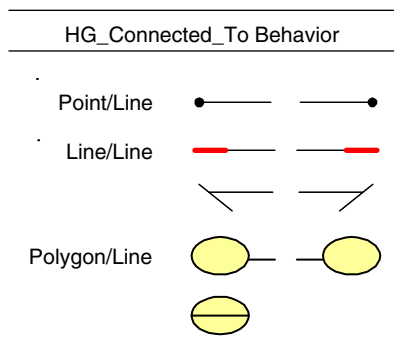
```
exec sp_spatial_query '
    select sw_member from spotelev
    where HG_Above(
        spotelev.sw_geometry,
        ST_Spatial(''ST_Point(5.0, 5.0, 55.0)'')
    )
    ,
```

HG_Connected_To

HG_Connected_To returns TRUE if the end point or start point of a line matches a point in the object.



The following diagram shows the behavior of `HG_Connected_To` with the basic shape types. 'Line' represents all Curve objects: `ST_CircularArc`, `ST_Polyline`, `HG_Curve`, and `HG_Circle`. Collections of objects follow these basic examples. Line direction in these examples is from left to right. All of these examples will return `TRUE`.



Syntax

`HG_Connected_To(spatial_obj1, spatial_obj2)`

spatial_obj1 is an `ST_Spatial`.

spatial_obj2 is an `ST_Spatial` representing an `ST_Polyline`.

Return Type

`HG_Connected_To` returns a Boolean value.

Example

```
exec sp_spatial_query '
select rdpaved.sw_member
from rdpaved, flood100
where HG_Connected_To(
    rdpaved.sw_geometry, flood100.sw_geometry
```

```
)  
,
```

HG_Identical

HG_Identical returns TRUE if two spatial objects are equal (both objects must have the same primitives in the same order).

Note: This function returns the same value as ST_Equals on page 155.

Syntax

HG_Identical(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is an ST_Spatial.

spatial_obj2 is an ST_Spatial.

Return Type

HG_Identical returns a Boolean value.

Example

```
exec sp_spatial_query '  
    select sw_member  
    from spotelev  
    where HG_Identical(  
        spotelev.sw_geometry,  
        ST_Spatial(''  
            ST_Point(1753960.182000, 10698421.103000, 2784.945100)  
        ''  
    )  
,
```

HG_Is_Box

HG_Is_Box returns TRUE if the input object contains only one HG_Box.

Syntax

HG_Is_Box(*spatial_obj*)

spatial_obj is an ST_Spatial.

Return Type

HG_Is_Box returns a Boolean value.

Example

```
exec sp_spatial_query '  
    select *  
    from parcel  
    where HG_Is_Box(parcel.sw_geometry)  
,
```

HG_Is_Circle

HG_Is_Circle returns TRUE if the input object contains only one HG_Circle.

Syntax

HG_Is_Circle(*spatial_obj1*)

spatial_obj1 is an ST_Spatial.

Return Type

HG_Is_Circle returns a Boolean value.

Example

```
exec sp_spatial_query '  
    select sw_member  
    from lake  
    where HG_Is_Circle(sw_geometry)  
,
```

HG_Is_CircularArc

HG_Is_CircularArc returns TRUE if the input object contains only one ST_CircularArc.

Syntax

HG_Is_CircularArc(*spatial_obj*)

spatial_obj is an ST_Spatial.

Return Type

HG_Is_CircularArc returns Boolean value.

Example

```
exec sp_spatial_query '
    select sw_member
    from parcel
    where HG_Is_CircularArc(sw_geometry)
    ,
```

HG_Is_Closed

HG_Is_Closed returns true if the input polygon, ST_Polygon, or line string, ST_Polyline, is closed. An ST_Polygon or ST_Polyline is closed if the start and end points are the same.

Syntax

HG_Is_Closed(*spatial_obj1*)

spatial_obj1 is an ST_Spatial representing a ST_Polygon or ST_Polyline.

Return Type

HG_Is_Closed returns a Boolean value.

Example

```
exec sp_spatial_query '
    select sw_member
    from parcel
    where HG_Is_Closed(sw_geometry)
    ,
```

HG_Is_Contiguous

HG_Is_Contiguous returns TRUE if the input path, ST_Path, is contiguous. An ST_Path is contiguous when all of the component curves, HG_Curve, are touching, so that the path is uninterrupted.

Syntax

HG_Is_Contiguous(*spatial_obj1*)

spatial_obj1 is an ST_Spatial representing a ST_Polyline.

Return Type

HG_Is_Contiguous returns a Boolean value.

Example

```
insert into testcon (sw_member, sw_geometry)
values (64, 'ST_Spatial(ST_Path(LIST{
  ST_Polyline(LIST{
    ST_Point(1,1),
    ST_Point(2,2)
  }),
  ST_CircularArc(LIST{
    ST_Point(2,2),
    ST_Point(2,3),
    ST_Point(3,3)
  }),
  ST_Polyline(LIST{
    ST_Point(3,3),
    ST_Point(3,1),
    ST_Point(1,1)
  })
}))')
exec sp_spatial_query '
select HG_Is_Contiguous(sw_geometry) from testcon
,
```

HG_Is_Curve

HG_Is_Curve returns TRUE if the input ST_Spatial contains only one HG_Curve.

Syntax

HG_Is_Curve(*spatial_obj*)

spatial_obj is an ST_Spatial.

Return Type

HG_Is_Curve returns a Boolean value.

Example

```
exec sp_spatial_query '
select sw_member
from parcel
where HG_Is_Curve(sw_geometry)
,
```

HG_Is_Empty

HG_Is_Empty returns TRUE if the geometry is empty, FALSE if it is not.

Syntax

HG_Is_Empty(*spatial_obj1*)

spatial_obj1 is an ST_Spatial.

Return Type

HG_Is_Empty returns a Boolean value.

Example

This example returns the sw_member of the empty geometry.

```
exec sp_spatial_query '  
    select sw_member from parcel  
    where HG_Is_Empty(sw_geometry)  
'
```

HG_Is_Forward

HG_Is_Forward returns true if the direction attribute of the input ST_Line is FORWARD. All ST_Lines have a direction attribute of FORWARD, REVERSE, or NULL.

Syntax

HG_Is_Forward(*spatial_obj1*)

spatial_obj1 is an ST_Spatial representing an ST_Polyline.

Return Type

HG_Is_Forward returns a Boolean value.

Example

```
exec sp_spatial_query '  
    select sw_member  
    from rdpaved  
    where HG_Is_Forward(sw_geometry)  
'
```

HG_Is_HG_Curve

HG_Is_HG_Curve returns TRUE if the input ST_Spatial contains only one HG_Curve.

Syntax

HG_Is_HG_Curve(*spatial_obj*)

spatial_obj is an ST_Spatial.

Return Type

HG_Is_HG_Curve returns a Boolean value.

Example

This example returns the sw_member of the geometry containing only one HG_Curve.

```
exec sp_spatial_query '
    select sw_member
    from flood100
    where HG_Is_hg_Curve(sw_geometry)
    ,
```

HG_Is_Invalid

HG_Is_Invalid is used to determine if a curve is invalid. If a curve is invalid, then HG_Is_Invalid returns TRUE.

Syntax

HG_Is_Invalid(*spatial_obj1*)

spatial_obj1 is an ST_Spatial representing a ST_Polyline.

Return Type

HG_Is_Invalid returns a Boolean value.

Example

```
insert into testcon (sw_member, sw_geometry)
values ( 65, 'ST_Spatial(ST_Path(LIST{
    ST_Polyline(LIST{
        ST_Point(1,1),
        ST_Point(2,2)
    }) ,
```

```
        ST_CircularArc (LIST{
            ST_Point(2,2),
            ST_Point(2,3),
            ST_Point(3,3)
        }),
        ST_Polyline (LIST{
            ST_Point(3,3),
            ST_Point(3,1),
            ST_Point(1,1)
        })
    )) ' )
exec sp_spatial_query '
    select HG_Is_Invalid(sw_geometry) from testcon
    ,
```

HG_Is_Nulldir

HG_Is_Nulldir returns TRUE if the direction property of the input ST_Line is 'NULL'. All ST_Lines have a direction attribute of FORWARD, REVERSE, or NULL.

Syntax

HG_Is_Nulldir(*spatial_obj1*)

spatial_obj1 is an ST_Spatial representing an ST_Polyline.

Return Type

HG_Is_Nulldir returns a Boolean value.

Example

```
exec sp_spatial_query '
    select sw_member
    from rdpaved
    where HG_Is_Nulldir(sw_geometry)
    ,
```

HG_Is_Path

HG_Is_Path returns TRUE if the input object contains only one ST_Path.

Syntax

HG_Is_Path(*spatial_obj1*)

spatial_obj1 is an ST_Spatial.

Return Type

HG_Is_Path returns a Boolean value.

Example

```
exec sp_spatial_query '  
    select sw_member  
    from parcel  
    where HG_Is_Path(sw_geometry)  
'
```

HG_Is_Point

HG_Is_Point returns TRUE if the input object contains only one ST_Point.

Syntax

HG_Is_Point(*spatial_obj1*)

spatial_obj1 is an ST_Spatial.

Return Type

HG_Is_Point returns a Boolean value.

Example

```
exec sp_spatial_query '  
    select sw_member  
    from spotelev  
    where HG_Is_Point(sw_geometry)  
'
```

HG_Is_Polygon

HG_Is_Polygon returns TRUE if the input object contains one ST_Polygon.

Syntax

HG_Is_Polygon(*spatial_obj*)

spatial_obj1 is an ST_Spatial.

Return Type

HG_Is_Polygon returns a Boolean value.

Example

```
exec sp_spatial_query '  
    select sw_member  
    from lake  
    where HG_Is_Polygon(sw_geometry)  
,
```

HG_Is_Polyline

HG_Is_Polyline returns TRUE if the input object contains only one ST_Polyline.

Syntax

HG_Is_Polyline(*spatial_obj1*)

spatial_obj1 is an ST_Spatial.

Return Type

HG_Is_Polyline returns a Boolean value.

Example

```
exec sp_spatial_query '  
    select sw_member  
    from rd paved  
    where HG_Is_Polyline(sw_geometry)  
,
```

HG_Is_Quad

HG_Is_Quad returns TRUE if the input object contains only one HG_Quad.

Syntax

HG_Is_Quad(*spatial_obj1*)

spatial_obj1 is an ST_Spatial.

Return Type

HG_Is_Quad returns a Boolean value.

Example

```
exec sp_spatial_query '  
    select sw_member  
    from flood100  
    where HG_Is_Quad(sw_geometry)  
,
```

HG_Is_Reverse

HG_Is_Reverse returns TRUE if the direction attribute of the input ST_line is set to REVERSE. All ST_Lines have a direction attribute of FORWARD, REVERSE, or NULL.

Syntax

HG_Is_Reverse(*spatial_obj1*)

spatial_obj1 is an ST_Spatial representing an ST_Polyline.

Return Type

HG_Is_Reverse returns a Boolean value.

Example

```
exec sp_spatial_query '  
    select sw_member from rdpaved  
    where HG_Is_Reverse(sw_geometry)  
,
```

HG_Is_Triangle

HG_Is_Triangle returns TRUE if the input object contains only one HG_Triangle.

Syntax

HG_Is_Triangle(*spatial_obj1*)

spatial_obj1 is an ST_Spatial.

Return Type

HG_Is_Triangle returns a Boolean.

Example

```
exec sp_spatial_query '  
    select sw_member  
    from flood100  
    where HG_Is_Triangle(sw_geometry)  
,
```

HG_Is_Valid

HG_Is_Valid returns TRUE if the geometry is valid, FALSE if it is not. Valid conditions of the different geometries are:

Polygon – Valid if it is closed and has valid paths.

Path – Valid if it has two or more points and all points are unique.

Polyline – Valid if it has two or more points and all points are unique.

Circular Arc – Valid if it has exactly three points.

Circle – Valid if it has a radius greater than zero.

Point – Always valid.

Syntax

HG_Is_Valid(*spatial_obj*)

spatial_obj1 is an ST_Spatial.

Return Type

HG_Is_Valid returns a Boolean value.

Example

```
exec sp_spatial_query '  
    select sw_member  
    from lake  
    where HG_Is_Valid(lake.sw_geometry)  
,
```

HG_Is_Znull

HG_Is_Znull returns TRUE if the geometry has NULL z coordinates and FALSE otherwise.

Syntax

HG_Is_Znull(*spatial_obj*)

spatial_obj1 is an ST_Spatial.

Return Type

HG_Is_Znull returns a Boolean value.

Example

```
exec sp_spatial_query '
    select sw_member
    from spotelev
    where HG_Is_Znull(sw_geometry)
    ,
```

HG_Level

HG_Level returns TRUE if point 1 has the same z coordinate value as point 2.

Syntax

HG_Level(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is an ST_Spatial representing a three dimensional ST_Point.

spatial_obj2 is an ST_Spatial representing a three dimensional ST_Point.

Return Type

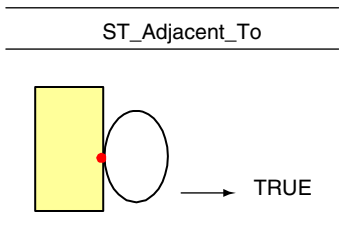
HG_Level returns a Boolean value.

Example

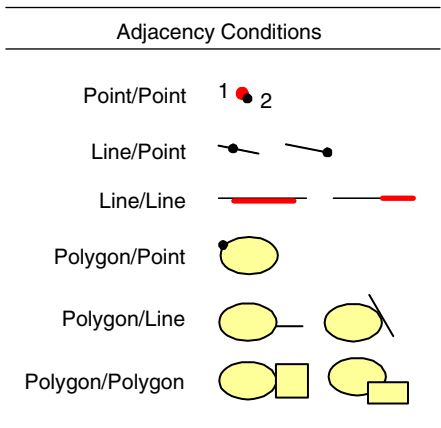
```
exec sp_spatial_query '
    select sw_member
    from spotelev
    where HG_Level(
        spotelev.sw_geometry,
        ST_Spatial(''
            ST_Point(1753960.182000, 10698421.103000, 2784.945100)
        ''
    )
    ,
```

ST_Adjacent_To

ST_Adjacent_To returns TRUE if two objects touch. They touch if they have one or more common boundary points, but no common interior points. ST_Adjacent_To is identical to ST_Meets.



The following diagram shows the basic valid adjacency conditions. Note that polygons can only share their boundary points with other shapes. If another shape encroaches a polygon's interior, then it is no longer adjacent.



Syntax

ST_Adjacent_To(*spatial_obj1*, *spatial_obj2*)
spatial_obj1 is an ST_Spatial.
spatial_obj2 is an ST_Spatial.

Return Type

ST_Adjacent_To returns a Boolean value.

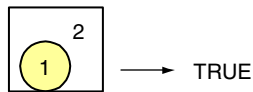
Example

```
exec sp_spatial_query '  
    select rdpaved.sw_member  
    from rdpaved, lake  
    where ST_Adjacent_To(rdpaved.sw_geometry, lake.sw_geometry)  
,
```

ST_Contained_By

ST_Contained_By returns TRUE if geometry1 is entirely contained within geometry2, and FALSE if it is not. Boundaries can touch, but the inner object cannot have any points outside the containing boundary.

ST_Contained_By



ST_Contained_By returns the inverse of what ST_Contains returns.

Syntax

ST_Contained_By(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is an ST_Spatial.

spatial_obj2 is an ST_Spatial.

Return Type

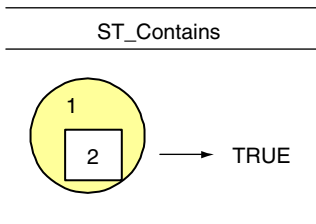
ST_Contained_By returns a Boolean value.

Example

```
exec sp_spatial_query '  
    select a.sw_member, b.sw_member  
    from pubbldg a, flood100 b  
    where ST_Contained_By(a.sw_geometry, b.sw_geometry)  
,
```

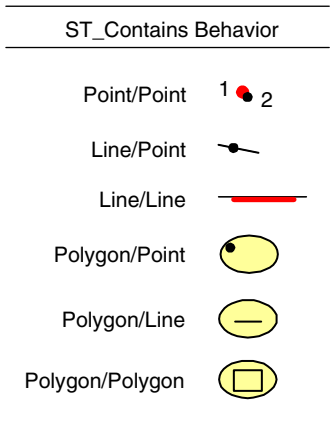
ST_Contains

ST_Contains returns TRUE if geometry1 entirely contains geometry2, and FALSE if it is not. Boundaries can touch, but the inner object cannot have any points outside the containing boundary. See also ST_Contain.



ST_Contains returns the inverse of what ST_Contained_By returns.

The following diagram illustrates the behavior of ST_Contains with various geometry combinations. In each of the geometry pairs, the first geometry contains the second. These cases represent the basic combinations that yield a true. Note that points can't contain anything but points. Lines can't contain polygons. In the case of a polygon, boundaries may touch, as long as the interior boundary stays within the exterior boundary.



Syntax

ST_Contains(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is an ST_Spatial.

spatial_obj2 is an ST_Spatial.

Return Type

ST_Contains returns a Boolean value.

Example

```
exec sp_spatial_query '  
    select a.sw_member, b.sw_member  
    from pubbldg a, flood100 b  
    where ST_Contains(a.sw_geometry, b.sw_geometry)  
,
```

ST_Equals

ST_Equals returns TRUE if *spatial_obj1* is identical to *spatial_obj2* (both objects must have the same primitives in the same order).

Note: This function returns the same value as HG_Identical on page 140.

Syntax

ST_Equals(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is an ST_Spatial.

spatial_obj2 is an ST_Spatial.

Return Type

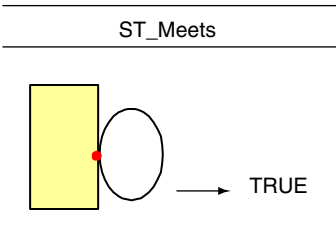
ST_Equals returns a Boolean value.

Example

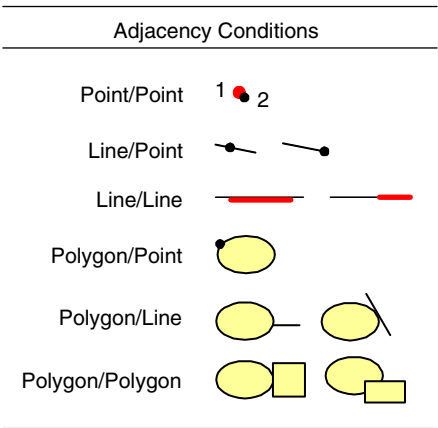
```
exec sp_spatial_query '  
    select a.sw_member, a.sw_geometry, b.sw_member, b.sw_geometry  
    from flood100 a, flood100 b  
    where ST_Equals(a.sw_geometry, b.sw_geometry)  
,
```

ST_Meets

ST_Meets is the same as ST_Adjacent_To. It returns TRUE if the two objects touch, but do not share common interior points.



The following diagram shows the basic valid adjacency conditions. Note that polygons can only share their boundary points with other shapes. If another shape encroaches a polygon's interior, then it is no longer adjacent.



Syntax

`ST_Meets(spatial_obj1, spatial_obj2)`

spatial_obj1 is an ST_Spatial.

spatial_obj2 is an ST_Spatial.

Return Type

ST_Meets returns a Boolean value.

Example

```
exec sp_spatial_query '  
    select rdpaved.sw_member  
    from rdpaved, lake  
    where ST_Meets(rdpaved.sw_geometry, lake.sw_geometry)  
'
```

ST_Not_Equals

ST_Not_Equals returns TRUE if *spatial_obj1* is not exactly identical to *spatial_obj2* (no tolerance applies).

Syntax

ST_Not_Equals(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is an ST_Spatial.

spatial_obj2 is an ST_Spatial.

Return Type

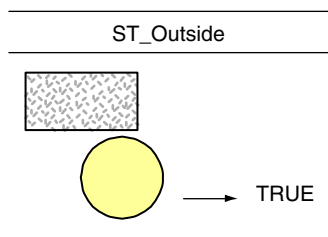
ST_Not_Equals returns a Boolean value.

Example

```
exec sp_spatial_query '  
    select HG_GetString(pubbldg.sw_geometry)  
    from pubbldg where (ST_Not_Equals(pubbldg.sw_geometry,  
    ST_Spatial(''HG_Box(1,1,2,2)'')))
```

ST_Outside

ST_Outside returns TRUE only if there are no common points between the two objects.



This function is the opposite of ST_Overlaps.

Syntax

ST_Outside(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is an ST_Spatial.

spatial_obj2 is an ST_Spatial.

Return Type

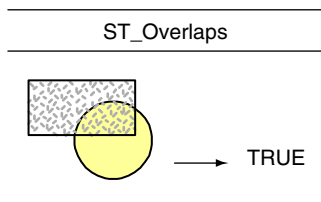
ST_Outside returns a Boolean value.

Example

```
exec sp_spatial_query '  
    select a.sw_member from lake a, flood100 b  
    where ST_Outside(a.sw_geometry, b.sw_geometry)  
'
```

ST_Overlaps

ST_Overlaps returns TRUE if elements of two geometries overlap. Two elements are overlapping when they share common points. If there are no overlapping elements, then ST_Overlaps returns FALSE. See also ST_Overlap on page 125.



This function is the opposite of ST_Outside on page 157.

Syntax

ST_Overlaps(*spatial_obj1*, *spatial_obj2*)

spatial_obj1 is an ST_Spatial.

spatial_obj2 is an ST_Spatial.

Return Type

ST_Overlaps returns a Boolean value.

Example

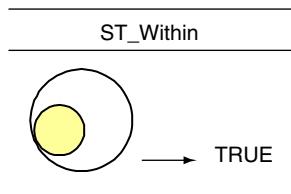
This example does not use the spatial index on the table. For large tables, use SDO_FILTER to make use of spatial indices defined on the table.

```
exec sp_spatial_query '  
    select ST_Overlap(flood100.sw_geometry, lake.sw_geometry)  
    from flood100, lake
```

```
    where ST_Overlaps(flood100.sw_geometry, lake.sw_geometry)  
,
```

ST_Within

ST_Within returns TRUE if geometry1 is entirely contained within geometry 2, and FALSE if it is not. Boundaries can touch, but the inner object cannot have any points outside the containing boundary.



This is the same as ST_Contained_By on page 153.

Syntax

`ST_Within(spatial_obj1, spatial_obj2)`

spatial_obj1 is an ST_Spatial.

spatial_obj2 is an ST_Spatial.

Return Type

ST_Within returns a Boolean value.

Example

```
exec sp_spatial_query '  
    select rdpaved.sw_member from rdpaved, flood100  
    where ST_Within(rdpaved.sw_geometry, flood100.sw_geometry)  
,
```

Using Coordinate Systems

This section describes how to perform coordinate system transformations.

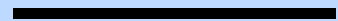
The following chapter provides supporting material to this chapter on using coordinate systems. It lists reference tables for Projections, Spheroids/Ellipsoids, Coordinate Units, and Datums.



11

Chapter

- **Performing Coordinate Transformations**
- **OGC Well-Known Text**



Performing Coordinate Transformations

SpatialWare for SQL Server provides SQL function for coordinate transformation. Geometries can be transformed from one coordinate system to another using HG_CSTransform.

The HG_CSTransform function performs an explicit transformation: applications must provide the geometry, the description of the coordinate system the geometry is in, and the description of the coordinate system to transform to. The description of the coordinate system is in the form of an OGC well-known text representation. Coordinate system descriptions are stored in a table called master..SpatialWare.HG_SpatialRef.

For more information about the transformation function, refer to the chapter describing Coordinate functions in the SpatialWare SQL/Spatial Section.

Note: Geometries are not tagged with the coordinate system that they are in. You may want to transform data from Longitude/Latitude to a planar coordinate system in order to perform accurate area calculations.

The SpatialWare.HG_SpatialRef Table

A table of supported coordinate systems, called HG_SpatialRef, is provided. This table contains the OGC well-known text strings. (These are the same as the coordinate systems defined in MapInfo Professional's MAPINFOW.PRJ.) The values in this table may be used in a join to perform coordinate system transformations.

Note: SQL Server users working with MapInfo Professional may notice that this table excludes non-earth projections. This is because they are not defined in the OpenGIS Consortium's model. You can, however, implement transformation between non-earth systems using HG_Transform or HG_Affine.

The following generates a complete list of supported coordinate systems stored in the HG_SpatialRef table.

```
select cs_name from master..SpatialWare.HG_SpatialRef
order by cs_name
```

A coordinate system is identified by a string of not more than 67 characters. This identifier is unique across the database instance, and is shared by all schemas in the instance. The following are examples of valid identifiers:

- 'British National Grid'
- 'Longitude / Latitude (NAD 83)'
- 'UTM Zone 28 (ED 50)'
- 'Vermont (1983)'

Note: Since identifiers are strings, please take note of white spaces when using them. You can find a coordinate system identifier by querying the HG_SpatialRef table. The following example shows how to find the definition of the Robinson coordinate system:

```
select srtext from HG_SpatialRef
where cs_name = 'Robinson'
```

OGC Well-Known Text

Coordinates are defined according to the standards outlined by the OpenGIS Consortium. For more information about OGC well-known text, refer to the Simple Features Specification for SQL (<http://www.opengis.org/techno/specs/99-049.pdf>).

Coordinate system definitions are stored in the HG_SpatialRef table.

The MapInfo GIS Extension supports projected coordinates (PROJCS) and geographic coordinates (GEOGCS). Geocentric coordinates (GEOCCS) are not supported.

Projected Coordinates

The following is the Extended Backus Naur Form (EBNF) definition of the string representation of a projected coordinate system:

```
<coordinate system> = <projected cs> | <geographic cs> |
<geocentric cs>
<projected cs> = PROJCS['<name>', <geographic cs>,
<projection>, {<parameter>,*} <linear unit>]
<projection> = PROJECTION['<name>']
<parameter> = PARAMETER['<name>', <value>]
<value> = <number>
```

The projected coordinate system is defined with a name ('<name>') followed by the geographic coordinate system, the map projection, zero (0) or more parameters, and the linear unit of measure.

An example of a projected coordinate system is British National Grid. Its string representation is:

```
PROJCS['British National Grid',
GEOGCS['Ordnance Survey GreatBrit',
DATUM['Ordnance Survey Great Brit',
SPHEROID['Airy 1930',6377563.396000,299.324965]],
PRIMEM['Greenwich',0],
UNIT['Decimal Degree',0.017453292520]],
```



```
PROJECTION['Transverse Mercator'],
PARAMETER['Scale_Factor',0.999601],
PARAMETER['Central_Meridian',-2.000000],
PARAMETER['Latitude_Of_Origin',49.000000],
PARAMETER['False_Easting',400000.000000],
PARAMETER['False_Northing',-100000.000000],
UNIT['Meter',1.000000000000]]
```

UNIT represents either an angular or linear unit of measure.

```
<angular unit> = <unit>
<linear unit> = <unit>
<unit> = UNIT['<name>', <conversion factor>]
<conversion factor> = <number>
```

The <conversion factor> must be larger than zero; it specifies per unit the number of meters for a linear unit or the number of radians for an angular unit.

Projected coordinate systems are based upon a geographic coordinate system.

Geographic Coordinates

The following is the EBNF definition of the string representation of a geographic coordinate system:

```
<geographic cs> = GEOGCS['<name>', <datum>, <prime meridian>,
<angular unit>]
<datum> = DATUM['<name>', <spheroid> {, <shift-x>, <shift-y>,
<shift-z> {, <rot-x>, <rot-y>, <rot-z>, <scale_adjust> } } ]
<spheroid> = SPHEROID['<name>', <semi-major axis>, <inverse
flattening>]
<semi-major axis> = <number> NOTE: semi-major axis is measured
in meters and must be > 0.
<inverse flattening> = <number>
<prime meridian> = PRIMEM['<name>', <longitude>]
<longitude> = <number>
```

The geographic coordinate system is defined with a name ('<name>') followed by the datum, the prime meridian, and the angular unit of measure. In the MapInfo GIS Extension, datum has been enhanced from the OGC well-known text standard to support user defined projections. Datum includes the following information:

- shift-x, shift-y, shift-z are in meters.
- rot-x, rot-y, rot-z are in seconds.
- scale_adjust is in parts per million.

The sign is specified by subtracting the datum's values from WGS84.

An example of a geographic coordinate system is Longitude/Latitude (NAD 83); its string representation is:

```
GEOGCS['Longitude / Latitude (NAD 83)',  
  DATUM['GRS 80',  
    SPHEROID['GRS 80',6378137.000000,298.257222]],  
  PRIMEM['Greenwich',0],  
  UNIT['Decimal Degree',0.017453292520]]
```

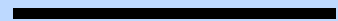
Coordinate Transformation Reference Tables

This chapter lists reference tables for Projections,
Spheroids/Ellipsoids, Coordinate Units, and Datums.

12

Chapter

- **Projections**
- **Spheroids/Ellipsoids**
- **Coordinate Units**
- **Datums**



Projections

The following table lists the projections supported. .

Table 1:

Geographic (Lat/Long)	Interrupted Goode Homolosine
Albers Conical Equal Area	Mollweide
Lambert Conformal Conic	Interrupted Mollweide
Mercator	Hammer
Polyconic	Wagner IV
Equidistant Conic	Wagner VII
Transverse Mercator	Oblated Equal Area
Stereographic	Non-earth
Lambert Azimuthal Equal Area	Transverse Mercator Danish System 45 Bornholm
Azimuthal Equidistant	Transverse Mercator Danish System 34 Jylland-Fyn
Gnomonic	Transverse Mercator Sjaelland
Orthographic	Transverse Mercator Finnish KKJ
General Vertical Near-Side Perspective	Eckert IV
Sinusiodal	Eckert VI
Equirectangular	Gall
Miller Cylindrical	Lambert Conformal Conic (Belgium 1972)
Van der Grinten	New Zealand Map Grid
Hotine Oblique Mercator	Cylindrical Equal Area
Robinson	Swiss Oblique Mercator
Space oblique mercator	Bonne
Alaska Conformal	Cassini

Note: The SQL Server implementation does not use Project IDs.

Spheroids/Ellipsoids

The following table lists the ellipsoids supported.

Table 2:

Clarke 1866	Everest (Kertau)	Walbeck
WGS 72	Fischer 1960 (Mercury)	Bessel 1841 (NGO 1948)
Australian	Fischer 1960 (South Asia)	South American 1969
Krassovsky	Fischer 1968	Clarke 1858
International 1924	GRS 67	Clarke 1880 (Jamaica)
Hayford	Helmert 1906	Clarke 1880 (Palestine)
Clarke 1880	Hough	Everest (Timbalai)
GRS 80	South American 1969	Everest (Kalianpur)
Clarke 1866 (Michigan)	War Office	Indonesian
Airy 1930	WGS 60	NWL 9D
Bessel 1841	WGS 66	NWL 10D
Everest	WGS 84	OSU86F
Sphere	Clarke 1880 (IGN)	OSU91A
Airy 1930(Ireland 1965)	OIAG 75	Plessis 1817
Bessel 1841 (Schwarzeck)	MERIT 83	Struve 1860
Clarke 1880 (Arc 1950)	New International 1967	Sphere (Unity)
Clarke 1880 (Merchich)		

Note: The SQL Server implementation does not use Ellipsoid IDs.

Coordinate Units

The following table lists the coordinate units supported. These units (except the last three) are also used for distance units.

Table 3:

Meter	Chain	Indian Foot
Kilometer	Rod	Link (Benoit)
Centimeter	Link	Link (Sears)
Millimeter	Decimal degree	Chain (Benoit)
Mile	Gon	Chain (Sears)
Nautical Mile	Grad	Yard (Indian)
Survey foot	Modified American Foot	Yard (Sears)
Foot	Clarke's Foot	Fathom
Inch	Decimal second	
Yard	Decimal minute	

Note: The SQL Server implementation does not use coordinate IDs.

Datums

The datum is established by tying a reference ellipsoid to a particular point on the earth. The following table lists:

- The number used to identify the datum (the same numbers used in MapInfo Professional's MAPINFOW.PRJ file).
- The datums.
- The maps (area) where they are typically used.
- Their reference ellipsoid.

Table 4:

Number	Datum	Area	Ellipsoid
1	Adindan	Ethiopia, Mali, Senegal, Sudan	Clarke 1880

Table 4:

Number	Datum	Area	Ellipsoid
2	Afgooye	Somalia	Krassovsky
3	Ain el Abd 1970	Bahrain Island	International
4	Anna 1 Astro 1965	Cocos Islands	Australian National
5	Arc 1950	Botswana, Lesotho, Malawi, Swaziland, Zaire, Zambia, Zimbabwe	Clarke 1880
6	Arc 1960	Kenya, Tanzania	Clarke 1880
7	Ascension Island 1958	Ascension Island	International
8	Astro Beacon "E"	Iwo Jima Island	International
9	Astro B4 Sorol Atoll	Tern Island	International
10	Astro DOS 71/4	St. Helena Island	International
11	Astronomic Station 1952	Marcus Island	International
12	Australian Geodetic 1966 (AGD 66)	Australia and Tasmania Island	Australian National
13	Australian Geodetic 1984 (AGD 84)	Australia and Tasmania Island	Australian National
110	Belgium	Belgium	International
14	Bellevue (IGN)	Efate and Erromango Islands	International
15	Bermuda 1957	Bermuda Islands	Clarke 1866
16	Bogota Observatory	Colombia	International
17	Campo Inchauspe	Argentina	International
18	Canton Astro 1966	Phoenix Islands	International
19	Cape	South Africa	Clarke 1880
20	Cape Canaveral	Florida and Bahama Islands	Clarke 1866
21	Carthage	Tunisia	Clarke 1880
22	Chatham 1971	Chatham Island (New Zealand)	International
23	Chua Astro	Paraguay	International

Table 4:

Number	Datum	Area	Ellipsoid
24	Corrego Alegre	Brazil	International
9999	Custom (see Appendix G)		
1000	Deutsches Hauptdreick-snetz (DHDN)	Germany	Bessel
25	Djakarta (Batavia)	Sumatra Island (Indonesia)	Bessel 1841
26	DOS 1968	Gizo Island (New Georgia Islands)	International
27	Easter Island 1967	Easter Island	International
28	European 1950 (ED 50)	Austria, Belgium, Denmark, Finland, France, Germany, Gibraltar, Greece, Italy, Luxembourg, Netherlands, Norway, Portugal, Spain, Sweden, Switzerland	International
29	European 1979 (ED 79)	Austria, Finland, Netherlands, Norway, Spain, Sweden, Switzerland	International
108	European 1987 (ED 87)	Europe	International
30	Gandajika Base	Republic of Maldives	International
31	Geodetic Datum 1949	New Zealand	International
32	Geodetic Reference System 1967 (GRS 67)	Worldwide	GRS 67
33	Geodetic Reference System 1980 (GRS 80)	Worldwide	GRS 80
34	Guam 1963	Guam Island	Clarke 1866
35	GUX 1 Astro	Guadalcanal Island	International
36	Hito XVIII 1963	South Chile (near 53°S)	International
37	Hjorsey 1955	Iceland	International
38	Hong Kong 1963	Hong Kong	International
39	Hu-Tzu-Shan	Taiwan	International
40	Indian	Thailand and Vietnam	Everest
41	Indian	Bangladesh, India, Nepal	Everest

Table 4:

Number	Datum	Area	Ellipsoid
42	Ireland 1965	Ireland	Modified Airy
43	ISTS 073 Astro 1969	Diego Garcia	International
44	Johnston Island 1961	Johnston Island	International
45	Kandawala	Sri Lanka	Everest
46	Kerguelen Island	Kerguelen Island	International
47	Kertau 1948	West Malaysia and Singapore	Modified Everest
48	L.C. 5 Astro	Cayman Brac Island	Clarke 1866
49	Liberia 1964	Liberia	Clarke 1880
113	Lisboa (DLx)	Portugal	International
50	Luzon	Philippines (excluding Mindanao Island)	Clarke 1866
51	Luzon	Mindanao Island	Clarke 1866
52	Mahe 1971	Mahe Island	Clarke 1880
53	Marco Astro	Salvage Islands	International
54	Massawa	Eritrea (Ethiopia)	Bessel 1841
114	Melrica 1973 (D73)	Portugal	International
55	Merchich	Morocco	Clarke 1880
56	Midway Astro 1961	Midway Island	International
57	Minna	Nigeria	Clarke 1880
58	Nahrwan	Masirah Island (Oman)	Clarke 1880
59	Nahrwan	United Arab Emirates	Clarke 1880
60	Nahrwan	Saudi Arabia	Clarke 1880
61	Naparima, BWI	Trinidad and Tobago	International
109	Netherlands	Netherlands	Bessel
62	North American 1927 (NAD 27)	Continental US	Clarke 1866

Table 4:

Number	Datum	Area	Ellipsoid
63	North American 1927 (NAD 27)	Alaska	Clarke 1866
64	North American 1927 (NAD 27)	Bahamas (excluding San Salvador Island)	Clarke 1866
65	North American 1927 (NAD 27)	San Salvador Island	Clarke 1866
66	North American 1927 (NAD 27)	Canada (including Newfoundland Island)	Clarke 1866
67	North American 1927 (NAD 27)	Canal Zone	Clarke 1866
68	North American 1927 (NAD 27)	Caribbean (Turks and Caicos Islands)	Clarke 1866
69	North American 1927 (NAD 27)	Central America (Belize, Costa Rica, El Salvador, Guatemala, Honduras, Nicaragua)	Clarke 1866
70	North American 1927 (NAD 27)	Cuba	Clarke 1866
71	North American 1927 (NAD 27)	Greenland (Hayes Peninsula)	Clarke 1866
72	North American 1927 (NAD 27)	Mexico	Clarke 1866
73	North American 1927 (NAD 27)	Michigan (used only for State Plane Coordinate System 1927)	Modified Clarke 1866
74	North American 1983 (NAD 83)	Alaska, Canada, Central America, Continental US, Mexico	GRS 80
107	Nouvelle Triangulation Francaise (NTF)	France	Modified Clarke 1880
1002	Nouvelle Triangulation Francaise (NTF) Greenwich Prime Meridian	France	Modified Clarke 1880
111	NWGL 10	Worldwide	WGS 72
75	Observatorio 1966	Corvo and Flores Islands (Azores)	International

Table 4:

Number	Datum	Area	Ellipsoid
76	Old Egyptian	Egypt	Helmert 1906
77	Old Hawaiian	Hawaii	Clarke 1866
78	Oman	Oman	Clarke 1880
79	Ordnance Survey of Great Britain 1936	England, Isle of Man, Scotland, Shetland Islands, Wales	Airy
80	Pico de las Nieves	Canary Islands	International
81	Pitcairn Astro 1967	Pitcairn Island	International
1000	Potsdam	Germany	Bessel
36	Provisional South Chilean 1963	South Chile (near 53°S)	International
82	Provisional South American 1956	Bolivia, Chile, Colombia, Ecuador, Guyana, Peru, Venezuela	International
83	Puerto Rico	Puerto Rico and Virgin Islands	Clarke 1866
1001	Pulkovo 1942	Germany	Krassovsky
84	Qatar National	Qatar	International
85	Qornoq	South Greenland	International
1000	Rauenberg	Germany	Bessel
86	Reunion	Mascarene Island	International
112	Rikets Triangulering 1990 (RT 90)	Sweden	Bessel
87	Rome 1940	Sardinia Island	International
88	Santo (DOS)	Espirito Santo Island	International
89	São Braz	São Miguel, Santa Maria Islands (Azores)	International
90	Sapper Hill 1943	East Falkland Island	International
91	Schwarzeck	Namibia	Modified Bessel 1841

Table 4:

Number	Datum	Area	Ellipsoid
92	South American 1969	Argentina, Bolivia, Brazil, Chile, Colombia, Ecuador, Guyana, Paraguay, Peru, Venezuela, Trinidad, and Tobago	South American 1969
93	South Asia	Singapore	Modified Fischer 1960
94	Southeast Base	Porto Santo and Madeira Islands	International
95	Southwest Base	Faial, Graciosa, Pico, Sao Jorge, Terceira Islands (Azores)	International
1003	Switzerland (CH 1903)	Switzerland	Bessel
96	Timbalai 1948	Brunei and East Malaysia (Sarawak and Sabah)	Everest
97	Tokyo	Japan, Korea, Okinawa	Bessel 1841
98	Tristan Astro 1968	Tristan da Cunha	International
9999	User-defined		
99	Viti Levu 1916	Viti Levu Island (Fiji Islands)	Clarke 1880
100	Wake-Eniwetok 1960	Marshall Islands	Hough
101	World Geodetic System 1960 (WGS 60)	Worldwide	WGS 60
102	World Geodetic System 1966 (WGS 66)	Worldwide	WGS 66
103	World Geodetic System 1972 (WGS 72)	Worldwide	WGS 72
104	World Geodetic System 1984 (WGS 84)	Worldwide	WGS 84
105	Yacare	Uruguay	International
106	Zanderij	Surinam	International