14.16.9.1 Spatial Relation Functions That Use Object Shapes

The OpenGIS specification defines the following functions to test the relationship between two geometry values *g1* and *g2*, using precise object shapes. The return values 1 and 0 indicate true and false, respectively, except that distance functions return distance values.

Functions in this section detect arguments in either Cartesian or geographic spatial reference systems (SRSs), and return results appropriate to the SRS.

Unless otherwise specified, functions in this section handle their geometry arguments as follows:

- If any argument is `NULL` or any geometry argument is an empty geometry, the return value is `NULL`.
- If any geometry argument is not a syntactically well-formed geometry, an `ER_GIS_INVALID_DATA` error occurs.
- If any geometry argument is a syntactically well-formed geometry in an undefined spatial reference system (SRS), an `ER_SRS_NOT_FOUND` error occurs.
- For functions that take multiple geometry arguments, if those arguments are not in the same SRS, an `ER_GIS_DIFFERENT_SRIDS` error occurs.
- If any geometry argument is geometrically invalid, either the result is true or false (it is undefined which), or an error occurs.
- For geographic SRS geometry arguments, if any argument has a longitude or latitude that is out of range, an error occurs:
  - If a longitude value is not in the range (−180, 180], an `ER_GEOMETRY_PARAM_LONGITUDE_OUT_OF_RANGE` error occurs.
  - If a latitude value is not in the range [−90, 90], an `ER_GEOMETRY_PARAM_LATITUDE_OUT_OF_RANGE` error occurs.

  Ranges shown are in degrees. If an SRS uses another unit, the range uses the corresponding values in its unit. The exact range limits deviate slightly due to floating-point arithmetic.
- Otherwise, the return value is non-`NULL`.

Some functions in this section permit a unit argument that specifies the length unit for the return value. Unless otherwise specified, functions handle their unit argument as follows:

- A unit is supported if it is found in the `INFORMATION_SCHEMA.ST_UNITS_OF_MEASURE` table. See Section 28.3.37, "The INFORMATION_SCHEMA ST_UNITS_OF_MEASURE Table".
- If a unit is specified but not supported by MySQL, an `ER_UNIT_NOT_FOUND` error occurs.
- If a supported linear unit is specified and the SRID is 0, an `ER_GEOMETRY_IN_UNKNOWN_LENGTH_UNIT` error occurs.
- If a supported linear unit is specified and the SRID is not 0, the result is in that unit.
- If a unit is not specified, the result is in the unit of the SRS of the geometries, whether Cartesian or geographic. Currently, all MySQL SRSs are expressed in meters.

These object-shape functions are available for testing geometry relationships:

- `ST_Contains(`***g1***`, `***g2***`)`

  Returns 1 or 0 to indicate whether ***g1*** completely contains ***g2*** (this means that ***g1*** and ***g2*** must not intersect). This relationship is the inverse of that tested by `ST_Within()`.

  `ST_Contains()` handles its arguments as described in the introduction to this section.

```
mysql> SET @g1 = ST_GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))'),
    ->     @p1 = ST_GeomFromText('Point(1 1)'),
    ->     @p2 = ST_GeomFromText('Point(3 3)'),
    ->     @p3 = ST_GeomFromText('Point(5 5)');
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT
    ->   ST_Contains(@g1, @p1), ST_Within(@p1, @g1),
    ->   ST_Disjoint(@g1, @p1), ST_Intersects(@g1, @p1)\G
*************************** 1. row ***************************
  ST_Contains(@g1, @p1): 1
    ST_Within(@p1, @g1): 1
  ST_Disjoint(@g1, @p1): 0
ST_Intersects(@g1, @p1): 1
1 row in set (0.00 sec)

mysql> SELECT
    ->   ST_Contains(@g1, @p2), ST_Within(@p2, @g1),
    ->   ST_Disjoint(@g1, @p2), ST_Intersects(@g1, @p2)\G
*************************** 1. row ***************************
  ST_Contains(@g1, @p2): 0
    ST_Within(@p2, @g1): 0
  ST_Disjoint(@g1, @p2): 0
ST_Intersects(@g1, @p2): 1
1 row in set (0.00 sec)

mysql>
```

```
    -> SELECT
    ->   ST_Contains(@g1, @p3), ST_Within(@p3, @g1),
    ->   ST_Disjoint(@g1, @p3), ST_Intersects(@g1, @p3)\G
*************************** 1. row ***************************
  ST_Contains(@g1, @p3): 0
    ST_Within(@p3, @g1): 0
  ST_Disjoint(@g1, @p3): 1
ST_Intersects(@g1, @p3): 0
1 row in set (0.00 sec)
```

- `ST_Crosses(g1, g2)`

  Two geometries *spatially cross* if their spatial relation has the following properties:

  - Unless ***g1*** and ***g2*** are both of dimension 1: ***g1*** crosses ***g2*** if the interior of ***g2*** has points in common with the interior of ***g1***, but ***g2*** does not cover the entire interior of ***g1***.
  - If both ***g1*** and ***g2*** are of dimension 1: If the lines cross each other in a finite number of points (that is, no common line segments, only single points in common).

  This function returns 1 or 0 to indicate whether ***g1*** spatially crosses ***g2***.

  `ST_Crosses()` handles its arguments as described in the introduction to this section except that the return value is `NULL` for these additional conditions:

  - ***g1*** is of dimension 2 (`Polygon` or `MultiPolygon`).
  - ***g2*** is of dimension 1 (`Point` or `MultiPoint`).

- `ST_Disjoint(g1, g2)`

  Returns 1 or 0 to indicate whether ***g1*** is spatially disjoint from (does not intersect) ***g2***.

  `ST_Disjoint()` handles its arguments as described in the introduction to this section.

- `ST_Distance(g1, g2 [, unit])`

  Returns the distance between ***g1*** and ***g2***, measured in the length unit of the spatial reference system (SRS) of the geometry arguments, or in the unit of the optional ***unit*** argument if that is specified.

  This function processes geometry collections by returning the shortest distance among all combinations of the components of the two geometry arguments.

  `ST_Distance()` handles its geometry arguments as described in the introduction to this section, with these exceptions:

- `ST_Distance()` detects arguments in a geographic (ellipsoidal) spatial reference system and returns the geodetic distance on the ellipsoid. `ST_Distance()` supports distance calculations for geographic SRS arguments of all geometry types.
- If any argument is geometrically invalid, either the result is an undefined distance (that is, it can be any number), or an error occurs.
- If an intermediate or final result produces `NaN` or a negative number, an `ER_GIS_INVALID_DATA` error occurs.

`ST_Distance()` permits specifying the linear unit for the returned distance value with an optional ***`unit`*** argument which `ST_Distance()` handles as described in the introduction to this section.

```
mysql> SET @g1 = ST_GeomFromText('POINT(1 1)');
mysql> SET @g2 = ST_GeomFromText('POINT(2 2)');
mysql> SELECT ST_Distance(@g1, @g2);
+----------------------+
| ST_Distance(@g1, @g2) |
+----------------------+
|    1.4142135623730951 |
+----------------------+

mysql> SET @g1 = ST_GeomFromText('POINT(1 1)', 4326);
mysql> SET @g2 = ST_GeomFromText('POINT(2 2)', 4326);
mysql> SELECT ST_Distance(@g1, @g2);
+----------------------+
| ST_Distance(@g1, @g2) |
+----------------------+
|     156874.3859490455 |
+----------------------+
mysql> SELECT ST_Distance(@g1, @g2, 'metre');
+--------------------------------+
| ST_Distance(@g1, @g2, 'metre') |
+--------------------------------+
|              156874.3859490455 |
+--------------------------------+
mysql> SELECT ST_Distance(@g1, @g2, 'foot');
+--------------------------------+
| ST_Distance(@g1, @g2, 'foot') |
+--------------------------------+
|             514679.7439273146 |
+--------------------------------+
```

For the special case of distance calculations on a sphere, see the `ST_Distance_Sphere()` function.

- `ST_Equals(`***`g1, g2`***`)`

Returns 1 or 0 to indicate whether ***g1*** is spatially equal to ***g2***.

ST_Equals() handles its arguments as described in the introduction to this section, except that it does not return NULL for empty geometry arguments.

```
mysql> SET @g1 = Point(1,1), @g2 = Point(2,2);
mysql> SELECT ST_Equals(@g1, @g1), ST_Equals(@g1, @g2);
+--------------------+--------------------+
| ST_Equals(@g1, @g1) | ST_Equals(@g1, @g2) |
+--------------------+--------------------+
|                  1 |                  0 |
+--------------------+--------------------+
```

- ST_FrechetDistance(***g1***, ***g2*** [, ***unit***])

Returns the discrete Fréchet distance between two geometries, reflecting how similar the geometries are. The result is a double-precision number measured in the length unit of the spatial reference system (SRS) of the geometry arguments, or in the length unit of the ***unit*** argument if that argument is given.

This function implements the discrete Fréchet distance, which means it is restricted to distances between the points of the geometries. For example, given two LineString arguments, only the points explicitly mentioned in the geometries are considered. Points on the line segments between these points are not considered.

ST_FrechetDistance() handles its geometry arguments as described in the introduction to this section, with these exceptions:

- The geometries may have a Cartesian or geographic SRS, but only LineString values are supported. If the arguments are in the same Cartesian or geographic SRS, but either is not a LineString, an ER_NOT_IMPLEMENTED_FOR_CARTESIAN_SRS or ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS error occurs, depending on the SRS type.

ST_FrechetDistance() handles its optional ***unit*** argument as described in the introduction to this section.

```
mysql> SET @ls1 = ST_GeomFromText('LINESTRING(0 0,0 5,5 5)');
mysql> SET @ls2 = ST_GeomFromText('LINESTRING(0 1,0 6,3 3,5 6)');
mysql> SELECT ST_FrechetDistance(@ls1, @ls2);
+-------------------------------+
| ST_FrechetDistance(@ls1, @ls2) |
+-------------------------------+
|             2.8284271247461903 |
+-------------------------------+
```

```
mysql> SET @ls1 = ST_GeomFromText('LINESTRING(0 0,0 5,5 5)', 4326);
mysql> SET @ls2 = ST_GeomFromText('LINESTRING(0 1,0 6,3 3,5 6)', 4326);
mysql> SELECT ST_FrechetDistance(@ls1, @ls2);
+------------------------------+
| ST_FrechetDistance(@ls1, @ls2) |
+------------------------------+
|             313421.1999416798 |
+------------------------------+
mysql> SELECT ST_FrechetDistance(@ls1, @ls2, 'foot');
+---------------------------------------+
| ST_FrechetDistance(@ls1, @ls2, 'foot') |
+---------------------------------------+
|                    1028284.7767115477 |
+---------------------------------------+
```

- `ST_HausdorffDistance(g1, g2 [, unit])`

  Returns the discrete Hausdorff distance between two geometries, reflecting how similar the geometries are. The result is a double-precision number measured in the length unit of the spatial reference system (SRS) of the geometry arguments, or in the length unit of the *`unit`* argument if that argument is given.

  This function implements the discrete Hausdorff distance, which means it is restricted to distances between the points of the geometries. For example, given two `LineString` arguments, only the points explicitly mentioned in the geometries are considered. Points on the line segments between these points are not considered.

  `ST_HausdorffDistance()` handles its geometry arguments as described in the introduction to this section, with these exceptions:

  - If the geometry arguments are in the same Cartesian or geographic SRS, but are not in a supported combination, an `ER_NOT_IMPLEMENTED_FOR_CARTESIAN_SRS` or `ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS` error occurs, depending on the SRS type. These combinations are supported:

    - `LineString` and `LineString`
    - `Point` and `MultiPoint`
    - `LineString` and `MultiLineString`
    - `MultiPoint` and `MultiPoint`
    - `MultiLineString` and `MultiLineString`

  `ST_HausdorffDistance()` handles its optional *`unit`* argument as described in the introduction to this section.

```
mysql> SET @ls1 = ST_GeomFromText('LINESTRING(0 0,0 5,5 5)');
mysql> SET @ls2 = ST_GeomFromText('LINESTRING(0 1,0 6,3 3,5 6)');
mysql> SELECT ST_HausdorffDistance(@ls1, @ls2);
+---------------------------------+
| ST_HausdorffDistance(@ls1, @ls2) |
+---------------------------------+
|                               1 |
+---------------------------------+

mysql> SET @ls1 = ST_GeomFromText('LINESTRING(0 0,0 5,5 5)', 4326);
mysql> SET @ls2 = ST_GeomFromText('LINESTRING(0 1,0 6,3 3,5 6)', 4326);
mysql> SELECT ST_HausdorffDistance(@ls1, @ls2);
+---------------------------------+
| ST_HausdorffDistance(@ls1, @ls2) |
+---------------------------------+
|              111319.49079326246 |
+---------------------------------+
mysql> SELECT ST_HausdorffDistance(@ls1, @ls2, 'foot');
+----------------------------------------+
| ST_HausdorffDistance(@ls1, @ls2, 'foot') |
+----------------------------------------+
|                       365221.4264870815 |
+----------------------------------------+
```

- `ST_Intersects(g1, g2)`

  Returns 1 or 0 to indicate whether *g1* spatially intersects *g2*.

  `ST_Intersects()` handles its arguments as described in the introduction to this section.

- `ST_Overlaps(g1, g2)`

  Two geometries *spatially overlap* if they intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

  This function returns 1 or 0 to indicate whether *g1* spatially overlaps *g2*.

  `ST_Overlaps()` handles its arguments as described in the introduction to this section except that the return value is `NULL` for the additional condition that the dimensions of the two geometries are not equal.

- `ST_Touches(g1, g2)`

  Two geometries *spatially touch* if their interiors do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

  This function returns 1 or 0 to indicate whether *g1* spatially touches *g2*.

`ST_Touches()` handles its arguments as described in the introduction to this section except that the return value is `NULL` for the additional condition that both geometries are of dimension 0 (`Point` or `MultiPoint`).

- `ST_Within(`***g1, g2***`)`

  Returns 1 or 0 to indicate whether ***g1*** is spatially within ***g2***. This tests the opposite relationship as `ST_Contains()`.

  `ST_Within()` handles its arguments as described in the introduction to this section.