MySQL 8.4 Reference Manual /
Functions and Operators / Spatial Analysis Functions / Functions That Test Spatial Relations Between Geometry Objects /
Spatial Relation Functions That Use Minimum Bounding Rectangles

14.16.9.2 Spatial Relation Functions That Use Minimum Bounding Rectangles

MySQL provides several MySQL-specific functions that test the relationship between minimum bounding rectangles (MBRs) of two geometries ***g1*** and ***g2***. The return values 1 and 0 indicate true and false, respectively.

The MBR (also known as the bounding box) for a two-dimensional geometry is the smallest rectangle which holds all points in the geometry, and so encloses the area between its greatest extents in both coordinate directions. In other words, it is the rectangle bounded by the points `(min(x), min(y))`, `(min(x), max(y))`, `(max(x), max(y))`, and `(max(x), min(y))`, where `min()` and `max()` represent the geometry's minimum and maximum x-coordinate or y-coordinate, respectively.

When speaking of relationships between geometries, it is important to distinguish between containment and covering, as described here:

- A geometry ***g1*** *contains* another geometry ***g2*** if and only if all points in ***g2*** are also in ***g1***, and their boundaries do not intersect. That is, all points `(a, b)` in ***g2*** must satisfy the conditions `min(x) < a < max(x)` and `min(y) < b < max(y)`. In this case, `ST_Contains(g1, g2)` and `MBRContains(g1, g2)` both return true, as does `ST_Within(g2, g1)`.
- We say that ***g1*** *covers* ***g2*** if all points in ***g2*** are also in ***g1***, including any boundary points. That is, all points `(a, b)` in ***g2*** must satisfy the conditions `min(x) <= a <= max(x)` and `min(y) <= b <= max(y)`. In this case, `MBRCovers(g1, g2)` and `MBRCoveredBy(g2, g1)` both return true.

Let us define a rectangle ***g1*** and points ***p1***, ***p2***, and ***p3*** using the SQL statements shown here:

```
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)');
```

g1 contains and covers ***p1***; ***p1*** is entirely within ***g1*** and does not touch any of its boundaries, as we can see from the `SELECT` statement shown here:

```
mysql> SELECT
    ->   ST_Contains(@g1, @p1), ST_Within(@p1, @g1),
    ->   MBRContains(@g1, @p1),
    ->   MBRCovers(@g1, @p1), MBRCoveredBy(@p1, @g1),
    ->   ST_Disjoint(@g1, @p1), ST_Intersects(@g1, @p1)\G
*************************** 1. row ***************************
  ST_Contains(@g1, @p1): 1
    ST_Within(@p1, @g1): 1
  MBRContains(@g1, @p1): 1
    MBRCovers(@g1, @p1): 1
 MBRCoveredBy(@p1, @g1): 1
  ST_Disjoint(@g1, @p1): 0
ST_Intersects(@g1, @p1): 1
1 row in set (0.01 sec)
```

Using the same query with `@p2` in place of `@p1`, we can see that ***g2*** covers ***p2***, but does not contain it, because ***p2*** is included in the boundary of ***g2***, but does not lie within its interior. (That is, `min(x) <= a <= max(x)` and `min(y) <= b <= max(y)` are true, but `min(x) < a < max(x)` and `min(y) < b < max(y)` are not.)

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

Executing the query—this time using `@p3` rather than `@p2` or `@p1`—shows us that ***p3*** is disjoint from ***g1***; the two geometries have no points in common, and ***g1*** neither contains nor covers ***p3***. `ST_Disjoint(g1, p3)` returns true; `ST_Intersects(g1, p3)` returns false.

```
mysql> SELECT
    ->   ST_Contains(@g1, @p3), ST_Within(@p3, @g1),
    ->   MBRContains(@g1, @p3),
    ->   MBRCovers(@g1, @p3), MBRCoveredBy(@p3, @g1),
    ->   ST_Disjoint(@g1, @p3), ST_Intersects(@g1, @p3)\G
*************************** 1. row ***************************
  ST_Contains(@g1, @p3): 0
    ST_Within(@p3, @g1): 0
  MBRContains(@g1, @p3): 0
    MBRCovers(@g1, @p3): 0
 MBRCoveredBy(@p3, @g1): 0
  ST_Disjoint(@g1, @p3): 1
```

```
ST_Intersects(@g1, @p3): 0
1 row in set (0.00 sec)
```

The function descriptions shown later in this section and in Section 14.16.9.1, “Spatial Relation Functions That Use Object Shapes” provide additional examples.

The bounding box of a point is interpreted as a point that is both boundary and interior.

The bounding box of a straight horizontal or vertical line is interpreted as a line where the interior of the line is also boundary. The endpoints are boundary points.

If any of the parameters are geometry collections, the interior, boundary, and exterior of those parameters are those of the union of all elements in the collection.

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 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 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`.

These MBR functions are available for testing geometry relationships:

- `MBRContains(g1, g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangle of ***g1*** contains the minimum bounding rectangle of ***g2***. This tests the opposite relationship as `MBRWithin()`.

  `MBRContains()` 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))'),
    ->   @g2 = ST_GeomFromText('Polygon((1 1,1 2,2 2,2 1,1 1))'),
    ->   @g3 = ST_GeomFromText('Polygon((0 0,0 5,5 5,5 0,0 0))'),
    ->   @g4 = ST_GeomFromText('Polygon((5 5,5 10,10 10,10 5,5 5))'),
    ->   @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
    ->   MBRContains(@g1, @g2), MBRContains(@g1, @g4),
    ->   MBRContains(@g2, @g1), MBRContains(@g2, @g4),
    ->   MBRContains(@g2, @g3), MBRContains(@g3, @g4),
    ->   MBRContains(@g3, @g1), MBRContains(@g1, @g3),
    ->   MBRContains(@g1, @p1), MBRContains(@p1, @g1),
    ->   MBRContains(@g1, @p1), MBRContains(@p1, @g1),
    ->   MBRContains(@g2, @p2), MBRContains(@g2, @p3),
    ->   MBRContains(@g3, @p1), MBRContains(@g3, @p2),
    ->   MBRContains(@g3, @p3), MBRContains(@g4, @p1),
    ->   MBRContains(@g4, @p2), MBRContains(@g4, @p3)\G
*************************** 1. row ***************************
MBRContains(@g1, @g2): 1
MBRContains(@g1, @g4): 0
MBRContains(@g2, @g1): 0
MBRContains(@g2, @g4): 0
MBRContains(@g2, @g3): 0
MBRContains(@g3, @g4): 0
MBRContains(@g3, @g1): 1
MBRContains(@g1, @g3): 0
MBRContains(@g1, @p1): 1
MBRContains(@p1, @g1): 0
MBRContains(@g1, @p1): 1
MBRContains(@p1, @g1): 0
MBRContains(@g2, @p2): 0
MBRContains(@g2, @p3): 0
MBRContains(@g3, @p1): 1
MBRContains(@g3, @p2): 1
MBRContains(@g3, @p3): 0
MBRContains(@g4, @p1): 0
MBRContains(@g4, @p2): 0
MBRContains(@g4, @p3): 0
1 row in set (0.00 sec)
```

- MBRCoveredBy(*g1*, *g2*)

  Returns 1 or 0 to indicate whether the minimum bounding rectangle of *g1* is covered by the minimum bounding rectangle of *g2*. This tests the opposite relationship as MBRCovers().

  MBRCoveredBy() 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))');
  mysql> SET @g2 = ST_GeomFromText('Point(1 1)');
  mysql> SELECT MBRCovers(@g1,@g2), MBRCoveredby(@g1,@g2);
  +--------------------+-----------------------+
  | MBRCovers(@g1,@g2) | MBRCoveredby(@g1,@g2) |
  +--------------------+-----------------------+
  |                  1 |                     0 |
  +--------------------+-----------------------+
  mysql> SELECT MBRCovers(@g2,@g1), MBRCoveredby(@g2,@g1);
  +--------------------+-----------------------+
  | MBRCovers(@g2,@g1) | MBRCoveredby(@g2,@g1) |
  +--------------------+-----------------------+
  |                  0 |                     1 |
  +--------------------+-----------------------+
  ```

  See the description of the MBRCovers() function for additional examples.

- MBRCovers(*g1*, *g2*)

  Returns 1 or 0 to indicate whether the minimum bounding rectangle of *g1* covers the minimum bounding rectangle of *g2*. This tests the opposite relationship as MBRCoveredBy(). See the description of MBRCoveredBy() for additional examples.

  MBRCovers() 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))'),
      ->   @g2 = ST_GeomFromText('Polygon((1 1,1 2,2 2,2 1,1 1))'),
      ->   @p1 = ST_GeomFromText('Point(1 1)'),
      ->   @p2 = ST_GeomFromText('Point(3 3)'),
      ->   @p3 = ST_GeomFromText('Point(5 5)');
  Query OK, 0 rows affected (0.02 sec)

  mysql> SELECT
      ->   MBRCovers(@g1, @p1), MBRCovers(@g1, @p2),
      ->   MBRCovers(@g1, @g2), MBRCovers(@g1, @p3)\G
  *************************** 1. row ***************************
  ```

```
MBRCovers(@g1, @p1): 1
MBRCovers(@g1, @p2): 1
MBRCovers(@g1, @g2): 1
MBRCovers(@g1, @p3): 0
1 row in set (0.00 sec)
```

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

  Returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries ***`g1`*** and ***`g2`*** are disjoint (do not intersect).

  `MBRDisjoint()` 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))'),
      ->   @g2 = ST_GeomFromText('Polygon((1 1,1 2,2 2,2 1,1 1))'),
      ->   @g3 = ST_GeomFromText('Polygon((0 0,0 5,5 5,5 0,0 0))'),
      ->   @g4 = ST_GeomFromText('Polygon((5 5,5 10,10 10,10 5,5 5))'),
      ->   @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
      ->   MBRDisjoint(@g1, @g4), MBRDisjoint(@g2, @g4),
      ->   MBRDisjoint(@g3, @g4), MBRDisjoint(@g4, @g4),
      ->   MBRDisjoint(@g1, @p1), MBRDisjoint(@g1, @p2),
      ->   MBRDisjoint(@g1, @p3)\G
  *************************** 1. row ***************************
  MBRDisjoint(@g1, @g4): 1
  MBRDisjoint(@g2, @g4): 1
  MBRDisjoint(@g3, @g4): 0
  MBRDisjoint(@g4, @g4): 0
  MBRDisjoint(@g1, @p1): 0
  MBRDisjoint(@g1, @p2): 0
  MBRDisjoint(@g1, @p3): 1
  1 row in set (0.00 sec)
  ```

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

  Returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries ***`g1`*** and ***`g2`*** are the same.

  `MBREquals()` 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 = ST_GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))'),
    ->   @g2 = ST_GeomFromText('Polygon((1 1,1 2,2 2,2 1,1 1))'),
    ->   @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
    ->   MBREquals(@g1, @g1), MBREquals(@g1, @g2),
    ->   MBREquals(@g1, @p1), MBREquals(@g1, @p2), MBREquals(@g2, @g2),
    ->   MBREquals(@p1, @p1), MBREquals(@p1, @p2), MBREquals(@p2, @p2)\G
*************************** 1. row ***************************
MBREquals(@g1, @g1): 1
MBREquals(@g1, @g2): 0
MBREquals(@g1, @p1): 0
MBREquals(@g1, @p2): 0
MBREquals(@g2, @g2): 1
MBREquals(@p1, @p1): 1
MBREquals(@p1, @p2): 0
MBREquals(@p2, @p2): 1
1 row in set (0.00 sec)
```

- `MBRIntersects(g1, g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* intersect.

  `MBRIntersects()` 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))'),
    ->   @g2 = ST_GeomFromText('Polygon((1 1,1 2,2 2,2 1,1 1))'),
    ->   @g3 = ST_GeomFromText('Polygon((0 0,0 5,5 5,5 0,0 0))'),
    ->   @g4 = ST_GeomFromText('Polygon((5 5,5 10,10 10,10 5,5 5))'),
    ->   @g5 = ST_GeomFromText('Polygon((2 2,2 8,8 8,8 2,2 2))'),
    ->   @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
    ->   MBRIntersects(@g1, @g1), MBRIntersects(@g1, @g2),
    ->   MBRIntersects(@g1, @g3), MBRIntersects(@g1, @g4), MBRIntersects(@g1
    ->   MBRIntersects(@g1, @p1), MBRIntersects(@g1, @p2), MBRIntersects(@g1
    ->   MBRIntersects(@g2, @p1), MBRIntersects(@g2, @p2), MBRIntersects(@g2
*************************** 1. row ***************************
MBRIntersects(@g1, @g1): 1
MBRIntersects(@g1, @g2): 1
MBRIntersects(@g1, @g3): 1
MBRIntersects(@g1, @g4): 0
MBRIntersects(@g1, @g5): 1
MBRIntersects(@g1, @p1): 1
MBRIntersects(@g1, @p2): 1
MBRIntersects(@g1, @p3): 0
MBRIntersects(@g2, @p1): 1
MBRIntersects(@g2, @p2): 0
MBRIntersects(@g2, @p3): 0
1 row in set (0.00 sec)
```

- `MBROverlaps(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 the minimum bounding rectangles of the two geometries ***g1*** and ***g2*** overlap.

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

- `MBRTouches(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 the minimum bounding rectangles of the two geometries ***g1*** and ***g2*** touch.

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

- `MBRWithin(g1, g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangle of ***g1*** is within the minimum bounding rectangle of ***g2***. This tests the opposite relationship as `MBRContains()`.

  `MBRWithin()` 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))'),
    ->   @g2 = ST_GeomFromText('Polygon((1 1,1 2,2 2,2 1,1 1))'),
    ->   @g3 = ST_GeomFromText('Polygon((0 0,0 5,5 5,5 0,0 0))'),
    ->   @g4 = ST_GeomFromText('Polygon((5 5,5 10,10 10,10 5,5 5))'),
    ->   @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
```

```
    ->   MBRWithin(@g1, @g2), MBRWithin(@g1, @g4),
    ->   MBRWithin(@g2, @g1), MBRWithin(@g2, @g4),
    ->   MBRWithin(@g2, @g3), MBRWithin(@g3, @g4),
    ->   MBRWithin(@g1, @p1), MBRWithin(@p1, @g1),
    ->   MBRWithin(@g1, @p1), MBRWithin(@p1, @g1),
    ->   MBRWithin(@g2, @p2), MBRWithin(@g2, @p3)\G
*************************** 1. row ***************************
MBRWithin(@g1, @g2): 0
MBRWithin(@g1, @g4): 0
MBRWithin(@g2, @g1): 1
MBRWithin(@g2, @g4): 0
MBRWithin(@g2, @g3): 1
MBRWithin(@g3, @g4): 0
MBRWithin(@g1, @p1): 0
MBRWithin(@p1, @g1): 1
MBRWithin(@g1, @p1): 0
MBRWithin(@p1, @g1): 1
MBRWithin(@g2, @p2): 0
MBRWithin(@g2, @p3): 0
1 row in set (0.00 sec)
```