Referencia de la función SQL ST_Geometry

Tabla de contenido

Funciones de SQL utilizadas con ST_Geometry . . . 6
SQL y Esri ST_Geometry . . . 12
Cargue la biblioteca ST_Geometry de SQLite . . . 15
Funciones del constructor para ST_Geometry . . . 16
Funciones del descriptor de acceso espacial . . . 20
Relaciones espaciales . . . 28
Funciones de la relación espacial . . . 29
Operaciones espaciales . . . 40
Funciones de la operación espacial . . . 42
Círculos, elipses y porciones de círculos paramétricos . . . 48
ST_Aggr_ConvexHull . . . 51
ST_Aggr_Intersection . . . 53
ST_Aggr_Union . . . 56
ST_Area . . . 59
ST_AsBinary . . . 62
ST_AsText . . . 64
ST_Boundary . . . 66
ST_Buffer . . . 70
ST_Centroid . . . 74
ST_Contains . . . 77
ST_ConvexHull . . . 81
ST_CoordDim . . . 85
ST_Crosses . . . 90
ST_Curve . . . 94
ST_Difference . . . 96
ST_Dimension . . . 100
ST_Disjoint . . . 104
ST_Distance . . . 108
ST_DWithin . . . 112
ST_EndPoint . . . 118
ST_Entity . . . 121
ST_Envelope . . . 124

Copyright © 1995-2023 Esri. All rights reserved.

ST_EnvIntersects 130
ST_Equals 133
ST_Equalsrs 136
ST_ExteriorRing 137
ST_GeomCollection 140
ST_GeomCollFromWKB 143
ST_Geometry 145
ST_GeometryN 152
ST_GeometryType 154
ST_GeomFromCollection 158
ST_GeomFromText 160
ST_GeomFromWKB 163
ST_GeoSize 166
ST_InteriorRingN 167
ST_Intersection 169
ST_Intersects 174
ST_Is3d 178
ST_IsClosed 182
ST_IsEmpty 187
ST_IsMeasured 191
ST_IsRing 195
ST_IsSimple 198
ST_Length 201
ST_LineFromText 204
ST_LineFromWKB 206
ST_LineString 209
ST_M 211
ST_MaxM 214
ST_MaxX 217
ST_MaxY 220
ST_MaxZ 223
ST_MinM 226
ST_MinX 229
ST_MinY 232

Copyright © 1995-2023 Esri. All rights reserved.

ST_MinZ . . . 235
ST_MLineFromText . . . 238
ST_MLineFromWKB . . . 240
ST_MPointFromText . . . 243
ST_MPointFromWKB . . . 245
ST_MPolyFromText . . . 248
ST_MPolyFromWKB . . . 250
ST_MultiCurve . . . 253
ST_MultiLineString . . . 254
ST_MultiPoint . . . 256
ST_MultiPolygon . . . 258
ST_MultiSurface . . . 260
ST_NumGeometries . . . 261
ST_NumInteriorRing . . . 264
ST_NumPoints . . . 267
ST_OrderingEquals . . . 270
ST_Overlaps . . . 272
ST_Perimeter . . . 276
ST_Point . . . 281
ST_PointFromText . . . 283
ST_PointFromWKB . . . 285
ST_PointN . . . 288
ST_PointOnSurface . . . 291
ST_PolyFromText . . . 294
ST_PolyFromWKB . . . 296
ST_Polygon . . . 299
ST_Relate . . . 301
ST_SRID . . . 306
ST_StartPoint . . . 308
ST_Surface . . . 311
ST_SymmetricDiff . . . 313
ST_Touches . . . 317
ST_Transform . . . 320
ST_Union . . . 327

Copyright © 1995-2023 Esri. All rights reserved.

ST_Within 331
ST_X 335
ST_Y 338
ST_Z 341

Copyright © 1995-2023 Esri. All rights reserved.

Funciones de SQL utilizadas con ST_Geometry

Este documento de referencia proporciona una lista y una descripción de las funciones disponibles para su uso con el tipo de datos espaciales Esri ST_Geometry en Oracle, PostgreSQL y SQLite.

Los tipos y funciones SQL Esri ST_Geometry se crean al realizar alguna de las siguientes acciones:

- Crear una geodatabase en una base de datos de Oracle.
- Usar ST_Geometry al crear una crear una geodatabase en una base de datos de PostgreSQL.
- Instalar el tipo de datos espaciales ST_Geometry en una base de datos de Oracle o PostgreSQL.
- Cree una base de datos de SQLite que incluya el tipo de datos espaciales ST_Geometry mediante la herramienta de geoprocesamiento Crear base de datos de SQLite o la función ArcPy y cargue funciones ST_Geometry para utilizar con la base de datos.
- Cargue las funciones de ST_Geometry para usar con una geodatabase móvil.

En las bases de datos de Oracle y PostgreSQL, el tipo ST_Geometry y sus funciones se crean en un esquema llamado sde. En SQLite, el tipo y las funciones se almacenan en una biblioteca que debe cargar antes de ejecutar el SQL en la base de datos o la geodatabase móvil de SQLite.

Sugerencia:

Para obtener información sobre el tipo Esri ST_Geometry, consulte las siguientes páginas de ayuda de ArcGIS Pro:

- ST_Geometry en PostgreSQL
- ST_Geometry en Oracle
- Bases de datos y ST_Geometry
- Cargue la biblioteca ST_Geometry de SQLite
- Cargar ST_Geometry en una geodatabase móvil para el acceso SQL

Formato de las páginas de funciones SQL

Las páginas de funciones de este documento tienen la siguiente estructura:

- Definición: un breve enunciado de lo que hace la función
- Sintaxis: la sintaxis SQL para usar la función

> **Nota:**
>
> Con los operadores relacionales, el orden en el que se especifican los parámetros es importante: el primer parámetro debe ser para la tabla en la que se hace la selección y el segundo parámetro debe ser para la tabla que se está usando como filtro.

- Tipo de devolución: el tipo de datos que se devuelve cuando se ejecuta la función
- Ejemplo: ejemplos que utilizan la función específica

Lista de las funciones SQL

Haga clic en los vínculos siguientes para ir a las funciones que puede usar con el tipo ST_Geometry en Oracle,

Copyright © 1995-2023 Esri. All rights reserved.

PostgreSQL y SQLite.

Cuando utilice las funciones ST_Geometry en Oracle, debe calificar las funciones y los operadores con `sde`. Por ejemplo, ST_Buffer debería ser sde.ST_Buffer. Al agregar `sde.` se indica al software que la función está almacenada en el esquema del usuario sde. En PostgreSQL, la calificación es opcional, pero es una buena práctica para incluir el calificador. No incluya la calificación cuando use las funciones con SQLite, ya que no hay esquema SDE en las bases de datos de SQLite.

Si proporciona cadenas de texto conocido como entrada con una función ST_Geometry de SQL, puede utilizar notación científica para especificar valores muy grandes o muy pequeños. Por ejemplo, si especifica coordenadas utilizando texto conocido mientras crea una entidad, y una de las coordenadas es 0.00002350001816501026, en su lugar puede escribir `2.3500001816501026e-005`.

Sugerencia:
Para otros tipos, por ejemplo, los tipos de PostGIS, SDO_Geometry de Oracle, los tipos espaciales de Microsoft SQL Server, ST_Geometry de IBM Db2 o ST_Geometry de SAP HANA, consulte la documentación proporcionada por el proveedor del sistema de administración de bases de datos para obtener información sobre las funciones utilizadas por cada uno de ellos.

Las siguientes funciones SQL Esri ST_Geometry están agrupadas según su uso.

Funciones del constructor

Las funciones del constructor adoptan un tipo de geometría o una descripción de texto de geometría y crean una geometría. La tabla siguiente muestra las funciones de constructor e indica qué implementaciones de ST_Geometry son compatibles con cada una de ellas.

Funciones del constructor

Función	Oracle	PostgreSQL	SQLite
ST_Centroid	X	X	X
ST_Curve	X		X
ST_GeomCollection	X	X	
ST_GeomCollFromWKB		X	
ST_Geometry	X	X	X
ST_GeomFromText	X		X
ST_GeomFromWKB	X	X	X
ST_LineFromText	X		X
ST_LineFromWKB	X	X	X
ST_LineString	X	X	X
ST_MLineFromText	X		X
ST_MLineFromWKB	X	X	X
ST_MPointFromText	X		X

Copyright © 1995-2023 Esri. All rights reserved.

Función	Oracle	PostgreSQL	SQLite
ST_MPointFromWKB	X	X	X
ST_MPolyFromText	X		X
ST_MPolyFromWKB	X	X	X
ST_MultiCurve	X		
ST_MultiLineString	X	X	X
ST_MultiPoint	X	X	X
ST_MultiPolygon	X	X	X
ST_MultiSurface	X		
ST_Point	X	X	X
ST_PointFromText	X		X
ST_PointFromWKB	X	X	X
ST_PolyFromText	X		X
ST_PolyFromWKB	X	X	X
ST_Polygon	X	X	X
ST_Surface	X		X

Funciones del descriptor de acceso

Existe una cantidad de funciones que toman una o varias geometrías con entrada y devuelven información específica sobre las geometrías.

Algunas de estas funciones del descriptor revisan para ver si una o varias entidades cumplen ciertos criterios. Si la geometría cumple con los criterios, la función devuelve 1 (Oracle y SQLite) o t (PostgreSQL) para true. Si la geometría no cumple con los criterios, devuelve 0 (Oracle y SQLite) o f (PostgreSQL) para false.

Estas funciones son válidas para todas las implementaciones, excepto aquellas en las que se indica lo contrario.

Funciones del descriptor de acceso
ST_Area
ST_AsBinary
ST_AsText
ST_CoordDim
ST_Dimension
ST_EndPoint
ST_Entity
ST_Equalsrs (solo PostgreSQL)
ST_ExteriorRing
ST_GeomFromCollection (solo PostgreSQL)

Copyright © 1995-2023 Esri. All rights reserved.

ST_GeometryType
ST_GeoSize (solo PostgreSQL)
ST_Is3d
ST_IsClosed
ST_IsEmpty
ST_IsMeasured
ST_IsRing
ST_IsSimple
ST_Length
ST_M
ST_MaxM
ST_MaxX
ST_MaxY
ST_MaxZ
ST_MinM
ST_MinX
ST_MinY
ST_MinZ
ST_NumGeometries
ST_NumInteriorRing
ST_NumPoints
ST_Perimeter
ST_SRID
ST_StartPoint
ST_X
ST_Y
ST_Z

Funciones relacionales

Las funciones relacionales toman geometrías como entrada y determinan si existe una relación espacial entre las geometrías. Si se cumplen las condiciones de relación espacial, estas funciones devuelven 1 (Oracle y SQLite) o t (PostgreSQL) para true. Si no se cumplen las condiciones (no existen relaciones), estas funciones devuelven 0 (Oracle y SQLite) o f (PostgreSQL) para false.

Estas funciones son válidas para todas las implementaciones, excepto aquellas en las que se indica lo contrario.

Funciones relacionales

Copyright © 1995-2023 Esri. All rights reserved.

ST_Contains
ST_Crosses
ST_Disjoint
ST_Distance
ST_DWithin
ST_EnvIntersects (solo Oracle y SQLite)
ST_Equals
ST_Intersects
ST_OrderingEquals (solo Oracle y PostgreSQL)
ST_Overlaps
ST_Relate
ST_Touches
ST_Within

Funciones de operación de geometría

Estas funciones toman datos espaciales, realizan operaciones espaciales en ellos y devuelven una geometría.

Estas funciones son válidas para todas las implementaciones, excepto aquellas en las que se indica lo contrario.

Funciones de operación de geometría

ST_Aggr_ConvexHull (solo Oracle y SQLite)
ST_Aggr_Intersection (solo Oracle y SQLite)
ST_Aggr_Union
ST_Boundary
ST_Buffer
ST_ConvexHull
ST_Difference
ST_Envelope
ST_ExteriorRing
ST_GeometryN
ST_InteriorRingN
ST_Intersection
ST_PointN
ST_PointOnSurface
ST_SymmetricDiff
ST_Transform

Copyright © 1995-2023 Esri. All rights reserved.

ST_Union

Copyright © 1995-2023 Esri. All rights reserved.

SQL y Esri ST_Geometry

Puede utilizar el Lenguaje estructurado de consultas (SQL), los tipos de datos y los formatos de tabla del sistema de administración de bases de datos para trabajar con la información almacenada en una geodatabase o base de datos en la que esté instalado el tipo ST_Geometry. SQL es un lenguaje de base de datos que admite comandos de definición de datos y manipulación de datos.

Al acceder a los datos a través de SQL, las aplicaciones externas pueden trabajar con los datos tabulares administrados por la geodatabase o la base de datos. Estas aplicaciones externas pueden ser aplicaciones de bases de datos no espaciales o aplicaciones espaciales personalizadas.

Al insertar o editar datos en una geodatabase o una base de datos mediante SQL, emita una sentencia `COMMIT` o `ROLLBACK` después de ejecutar la sentencia SQL para asegurarse de que los cambios se confirman en la base de datos o se deshacen. Esto ayuda a evitar que se mantengan bloqueos en las filas, páginas o tablas que está editando.

Insertar datos ST_Geometry mediante SQL

Puede utilizar SQL para insertar datos espaciales en una tabla de base de datos o de geodatabase que tenga una columna ST_Geometry. Puede usar las funciones del constructor de ST_Geometry para insertar tipos de geometría específicos. También puede especificar que la salida de ciertas funciones de operación espacial se incluya en una tabla existente.

Cuando inserte una geometría en una tabla con SQL, tenga en cuenta lo siguiente:

- Debe especificar un Id. de referencia espacial (SRID) válido.
- Todas las geometrías de la misma columna deben utilizar el mismo identificador de referencia espacial (SRID).
- Para seguir usando la tabla con ArcGIS, el campo utilizado como Id. de objeto no puede ser nulo ni contener valores no únicos.

Id. de referencia espacial

El SRID que especifique al insertar una geometría en una tabla en Oracle que utilice el tipo espacial ST_Geometry debe estar en la tabla ST_SPATIAL_REFERENCES y tener un registro coincidente en la tabla SDE.SPATIAL_REFERENCES. El SRID que especifique al insertar una geometría en una tabla en PostgreSQL que utilice el tipo espacial ST_Geometry debe estar en la tabla public.sde_spatial_references. Estas tablas se rellenan previamente con las referencias espaciales y los SRID.

El SRID que especifique al insertar una geometría en una tabla en SQLite que utilice el tipo espacial ST_Geometry debe estar en la tabla st_spatial_reference_systems.

Si necesita utilizar una referencia espacial personalizada que no esté presente en la tabla, lo más sencillo es cargar o crear una clase de entidad que tenga los valores de referencia espacial que desee. Asegúrese de que la clase de entidad que cree utilice el almacenamiento ST_Geometry. De este modo, se crea un registro en las tablas SDE.SPATIAL_REFERENCES y ST_SPATIAL_REFERENCES en Oracle, un registro en la tabla public.sde_spatial_references en PostgreSQL o un registro en la tabla st_aux_spatial_reference_systems_table en SQLite.

En las geodatabases, puede consultar la tabla LAYERS (Oracle) o sde_layers (PostgreSQL) para averiguar el SRID asignado a la tabla espacial. A continuación, puede utilizar ese SRID cuando cree tablas espaciales e inserte datos con SQL.

Copyright © 1995-2023 Esri. All rights reserved.

Nota:
Con el propósito de utilizar las muestras en este documento, se ha agregado un registro a las tablas ST_SPATIAL_REFERENCES y sde_spatial_references para indicar una referencia espacial desconocida. Este registro tiene un SRID de 0. Puede usar este SRID para los ejemplos de este documento. Sin embargo, no es un SRID oficial; se proporciona con el fin de realizar un ejemplo de código SQL. Se recomienda no usar este SRID para los datos de producción.

Id. de objeto

Para que ArcGIS consulte los datos, es necesario que la tabla contenga un campo de identificador de objeto único.

Las clases de entidad creadas con ArcGIS siempre tienen un campo de Id. de objeto que se utiliza como campo de identificador. Cuando se insertan registros en la clase de entidad con ArcGIS, siempre se inserta un valor único en el campo de Id. de objeto. ArcGIS mantiene el campo de Id. de objeto de una tabla de geodatabase. El sistema de administración de bases de datos mantiene el campo de Id. de objeto de una tabla de base de datos creada en ArcGIS.

Al insertar registros en una tabla de geodatabase con SQL, debe insertar un valor de Id. de objeto único válido.

Las tablas de base de datos que cree fuera de ArcGIS deben tener un campo (o un conjunto de campos) que ArcGIS pueda utilizar como Id. de objeto. Si utiliza el tipo de datos autoincrementado nativo de su base de datos para el campo de Id. de su tabla, la base de datos rellenará este campo cuando inserte un registro con SQL. Si mantiene manualmente los valores en su campo de identificador único, asegúrese de proporcionar un valor único para el Id. al editar la tabla desde SQL.

Nota:
No puede publicar datos de tablas que tengan un campo de identificador único que no esté mantenido por ArcGIS o el sistema de administración de bases de datos.

Editar datos ST_Geometry mediante SQL

Las ediciones de SQL de registros existentes afectan a menudo a los atributos no espaciales almacenados en la tabla; sin embargo, puede editar los datos de la columna ST_Geometry con funciones del constructor dentro de sentencias SQL `UPDATE`.

Si los datos están almacenados en una geodatabase, hay algunas pautas adicionales que debe seguir al editar con SQL:

- No actualice los registros con SQL si los datos se han registrado como versionados o se han habilitado para el archivado de geodatabases.
- No modifique ningún atributo que afecte a otros objetos de la base de datos que participen en el comportamiento de la geodatabase, como clases de relación, anotación vinculada a entidad, topología, reglas de atributos o redes.
- No utilice SQL para modificar los esquemas de tabla.

Copyright © 1995-2023 Esri. All rights reserved.

⚠ **Precaución:**

Utilizar SQL para acceder a la geodatabase omite la funcionalidad de la geodatabase, como el versionado, topología, redes, terrenos, anotación vinculada a entidad u otras extensiones de espacio de trabajo o de clase. Se podrían utilizar las entidades del sistema de administración de bases de datos como desencadenadores y los procedimientos almacenados para mantener las relaciones entre las tablas que son necesarias para ciertas funcionalidades de geodatabase. Sin embargo, ejecutar comandos SQL en la geodatabase sin tener en cuenta esta funcionalidad adicional, por ejemplo, emitir sentencias `INSERT` para agregar registros a una tabla que tiene habilitado el archivado de geodatabases o agregar una columna a una clase de entidad existente, sorteará la funcionalidad de la geodatabase y posiblemente dañará las relaciones entre los datos de la geodatabase.

Copyright © 1995-2023 Esri. All rights reserved.

Cargue la biblioteca ST_Geometry de SQLite

Haga lo siguiente antes de ejecutar comandos SQL que contengan funciones ST_Geometry en una base de datos de SQLite:

1. Descargue el archivo zip de Bibliotecas de ArcGIS Pro ST_Geometry (SQLite) desde My Esri y descomprímalo.
2. Instale un editor de SQL en el mismo equipo que la base de datos.
3. Coloque el archivo ST_Geometry en una ubicación accesible para la base de datos de SQLite y el editor de SQL desde el que cargará vaya a cargar ST_Geometry.
   Si la base de datos de SQLite está en un equipo Microsoft Windows, utilice el archivo `stgeometry_sqlite.dll`.
   Si la base de datos de SQLite está en un equipo Linux, utilice el archivo `libstgeometry_sqlite.so`.
4. Abra el editor SQL y conéctese a la base de datos de SQLite.
5. Cargue la biblioteca ST_Geometry.
   En el primer ejemplo que se ve a continuación, la biblioteca se carga para una base de datos de SQLite en un equipo Windows. El segundo ejemplo carga la biblioteca para una base de datos de SQLite en un equipo Linux.

```
--Load the ST_Geometry library on Windows.
SELECT load_extension(
 'stgeometry_sqlite.dll',
 'SDE_SQL_funcs_init'
);

--Load the ST_Geometry library on Linux.
SELECT load_extension(
 'libstgeometry_sqlite.so',
 'SDE_SQL_funcs_init'
);
```

Ahora puede ejecutar comandos SQL que contienen funciones ST_Geometry en la base de datos de SQLite.

Copyright © 1995-2023 Esri. All rights reserved.

Funciones del constructor para ST_Geometry

Las funciones del constructor crean una geometría a partir de una descripción de texto conocido o un binario conocido.

Cuando se proporciona una descripción de texto conocido para construir una geometría, la coordenada de medición se debe especificar al final. Por ejemplo, si el texto incluye coordenadas para x, y, z y m, se deben proporcionar en ese orden.

Una geometría puede tener cero o más puntos. Una geometría se considera vacía si tiene cero puntos. El subtipo punto es la única geometría que está restringida a cero o un punto; todos los demás subtipos pueden tener cero o más.

Las siguientes secciones describen la geometría superclase y las geometrías de subclase, e incluyen las funciones que cada una puede construir.

También puede construir geometrías como resultado de una operación espacial realizada en geometrías existentes.

Geometría superclase

No se puede crear una instancia de la superclase ST_Geometry; aunque puede definir una columna como un tipo ST_Geometry, los datos reales insertados se definen como entidades de punto, cadena de línea, polígono, multipunto, cadena de multilínea o multipolígono.

Las siguientes funciones se pueden utilizar para crear una superclase que contenga cualquiera de estos tipos de entidad.

- ST_Geometry
- ST_GeomFromText (solo Oracle y SQLite)
- ST_GeomFromWKB

Subclases

Puede definir una entidad como una subclase específica, en cuyo caso sólo el tipo de entidad permitido para esa subclase se pueden insertar. Por ejemplo, ST_PointFromWKB sólo puede crear entidades de punto.

ST_Point

Un ST_Point es una geometría de cero dimensión que ocupa una sola ubicación en un espacio de coordenadas. Un ST_Point que tiene un solo valor de coordenada x,y siempre es simple y tiene un límite NULL. ST_Point puede utilizarse para definir entidades, como pozos petroleros, hitos y sitios de recolección de muestras de agua.

Las siguientes funciones crean un punto:

- ST_Point
- ST_PointFromText (solo Oracle y SQLite)
- ST_PointFromWKB

ST_MultiPoint

Un ST_MultiPoint es un conjunto de ST_Points y, como sus elementos, tiene una dimensión 0. El tipo ST_MultiPoint es simple si ninguno de sus elementos ocupa el mismo espacio de coordenadas. El límite de un ST_MultiPoint es NULL. ST_MultiPoints pueden definir cosas como patrones de transmisión aérea e incidentes del brote de una

Copyright © 1995-2023 Esri. All rights reserved.

enfermedad.

Las siguientes funciones crean una geometría multipunto:

- ST_MultiPoint
- ST_MPointFromText (solo Oracle)
- ST_MPointFromWKB

ST_LineString

Un ST_LineString es un objeto unidimensional almacenado como una secuencia de puntos que define una ruta interpolada lineal. El tipo ST_LineString es simple si no se interseca con el interior. Los extremos (el límite) de un tipo ST_LineString cerrado ocupan el mismo punto en el espacio. Un tipo ST_LineString es un anillo si es cerrado y simple. Al igual que otras propiedades inherentes de ST_Geometry de superclase, ST_LineStrings tiene longitud. ST_LineStrings a menudo se utilizan para definir entidades lineales como carreteras, ríos y líneas de alimentación.

Los extremos normalmente forman el límite de un tipo ST_LineString a menos que éste sea cerrado, en cuyo caso el límite es NULL. El interior de ST_LineString es una ruta conectada que está entre los extremos, a menos que esté cerrada y, en tal caso, el interior es continuo.

Entre las funciones que crean cadenas de líneas se incluyen las siguientes:

- ST_LineString
- ST_LineFromText (solo Oracle y SQLite)
- ST_LineFromWKB
- ST_Curve (solo Oracle y SQLite)

ST_MultiLineString

Un ST_MultiLineString es una colección de ST_LineStrings.

El límite de un ST_MultiLineString son los extremos no intersecados de los elementos de ST_LineString. El límite de un ST_MultiLineString es NULL si todos los extremos de todos los elementos se intersecan. Además de las demás propiedades inherentes de ST_Geometry de superclase, ST_MultiLineStrings tiene longitud. ST_MultiLineStrings se usa para definir entidades lineales no contiguas, como arroyos o redes de carreteras.

Las funciones que construyen multicadenas de líneas son las siguientes:

- ST_MultiLineString
- ST_MLineFromText (solo Oracle y SQLite)
- ST_MLineFromWKB
- ST_MultiCurve (solo Oracle)

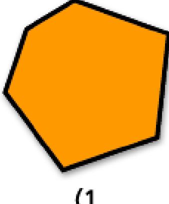
ST_Polygon

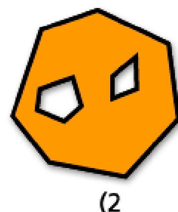
Un ST_Polygon es una superficie de dos dimensiones almacenada como una secuencia de puntos que define el anillo de delimitación exterior y 0 o más anillos interiores. Los ST_Polygons son siempre simples. Los ST_Polygons definen entidades que tienen una extensión espacial, como parcelas de tierra, cuerpos de agua y áreas de jurisdicción.

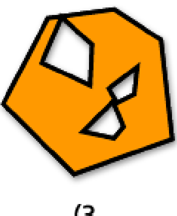
Este gráfico muestra ejemplos de objetos ST_Polygon: 1 es un ST_Polygon cuyo límite viene definido por un anillo

Copyright © 1995-2023 Esri. All rights reserved.

exterior. 2 es un ST_Polygon con un límite definido por un anillo exterior y dos anillos interiores. El área dentro de los anillos interiores forma parte del exterior del ST_Polygon. 3 es un ST_Polygon legal porque los anillos se intersecan en un único punto tangente.







La parte exterior y los anillos interiores definen el límite de un tipo ST_Polygon y el espacio cerrado entre los anillos define el interior de ST_Polygon. Los anillos de un tipo ST_Polygon pueden intersecarse en un punto tangente pero nunca cruzarse. Además de las demás propiedades inherentes de la ST_Geometry de superclase, ST_Polygons tiene área.

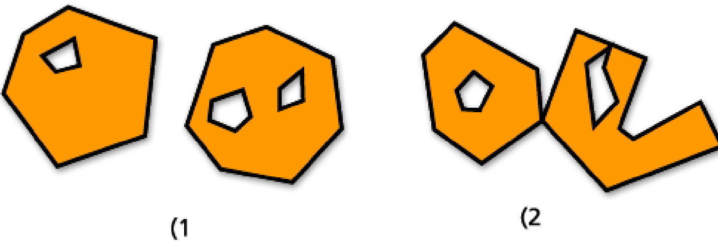
Entre las funciones que crean polígonos se incluyen las siguientes:

- ST_Polygon
- ST_PolyFromText (solo Oracle y SQLite)
- ST_PolyFromWKB
- ST_Surface (solo Oracle y SQLite)

ST_MultiPolygon

El límite de un ST_MultiPolygon es la longitud acumulativa del anillo exterior e interior de sus elementos. El interior de un tipo ST_MultiPolygon se define como los interiores acumulados de su elemento ST_Polygons. El límite de los elementos de un tipo ST_MultiPolygon solo se puede intersecar en un punto tangente. Además de las demás propiedades inherentes de la ST_Geometry de superclase, ST_MultiPolygons tiene área. ST_MultiPolygons define las entidades, tales como un estrato de un bosque o una parcela no contigua de tierra, como una cadena de islas del Pacífico.

El gráfico siguiente proporciona ejemplos de ST_MultiPolygon: 1 es un ST_MultiPolygon con dos elementos ST_Polygon. El límite se define mediante los dos anillos exteriores y los tres anillos interiores. 2 también es un ST_MultiPolygon con dos elementos ST_Polygon, pero el límite está definido por los dos anillos exteriores y los dos anillos interiores, y los dos elementos ST_Polygon se intersecan en un punto tangente.



Las siguientes funciones crean multipolígonos:

- ST_MultiPolygon
- ST_MPolyFromText (solo Oracle y SQLite)

Copyright © 1995-2023 Esri. All rights reserved.

- ST_MPolyFromWKB
- ST_MultiSurface (solo Oracle)

Construir geometrías a partir de geometrías existentes

Aunque no sean estrictamente funciones del constructor, las siguientes funciones devuelven una nueva geometría tomando las geometrías existentes como entrada y realizando análisis en ellas:

- ST_Aggr_ConvexHull (solo Oracle y SQLite)
- ST_Aggr_Intersection (solo Oracle y SQLite)
- ST_Aggr_Union
- ST_Boundary
- ST_Buffer
- ST_Centroid
- ST_ConvexHull
- ST_Difference
- ST_Envelope
- ST_ExteriorRing
- ST_Intersection
- ST_SymmetricDiff
- ST_Transform
- ST_Union

Copyright © 1995-2023 Esri. All rights reserved.

Funciones del descriptor de acceso espacial para ST_Geometry

Las funciones del descriptor de acceso devuelven la propiedad de una geometría. Existen funciones del descriptor de acceso para determinar las siguientes propiedades de una entidad ST_Geometry:

Dimensionalidad

Las dimensiones de una geometría son las coordenadas mínimas (ninguna, x, y) requeridas para definir la extensión espacial de la geometría.

Una geometría puede tener una dimensión de 0, 1 ó 2.

Las dimensiones son las siguientes:

- 0: no tiene ni longitud ni área
- 1: tiene longitud (x o y)
- 2: contiene área (x, y)

Los subtipos punto y multipunto tienen una dimensión de 0. Los puntos representan entidades de dimensión cero que pueden moldearse con una sola coordenada, mientras que los multipuntos representan datos que deben moldearse con un clúster de coordenadas desconectadas.

Los subtipos cadena de líneas y cadena de líneas multilínea tienen una dimensión de 1. Almacenan entidades como segmentos de carreteras, ramificación de sistemas de ríos y cualquier otra entidad que sea lineal por naturaleza.

Los subtipos polígono y multipolígono tienen una dimensión de 2. Las masas forestales, parcelas, cuerpos de agua y otras entidades que tienen perímetros que encierran un área definida pueden renderizarse mediante el tipo de datos de polígono o multipolígono.

La dimensión es importante no sólo como propiedad del subtipo sino también para determinar la relación espacial de dos entidades. La dimensión de la entidad o entidades resultantes determina si la operación se realizó de manera correcta o no. Las funciones del descriptor de acceso espacial examinan las dimensiones de las entidades para determinar cómo deben compararse.

Para evaluar la dimensión de una geometría, utilice la función ST_Dimension, la cual toma una entidad ST_Geometry y devuelve su dimensión como un número entero.

Las coordenadas de una geometría también tienen dimensiones. Si una geometría tiene solamente coordenadas x e y, la dimensión de la coordenada es 2. Si una geometría tiene coordenadas x, y, y z, la dimensión de la coordenada es 3. Si una geometría tiene coordenadas x, y, z y m, la dimensión de la coordenada es 4.

Puede utilizar la función ST_CoordDim para determinar las dimensiones de coordenadas presentes en una geometría.

Coordenadas z

Algunas geometrías tienen una altitud o profundidad asociada: una tercera dimensión. Cada uno de los puntos que forman la geometría de una entidad pueden incluir una coordenada z opcional que representa una altitud o profundidad relativa a la superficie de la tierra.

La función de predicado ST_Is3d toma un ST_Geometry como entrada, y devuelve true si la función tiene coordenadas z o false si no las tiene.

Copyright © 1995-2023 Esri. All rights reserved.

Puede determinar la coordenada z de un punto con la función ST_Z.

La función ST_MaxZ devuelve la coordenada z máxima, y la función ST_MinZ devuelve la coordenada z mínima de una geometría.

Medidas

Las medidas son valores asignados a cada coordenada. Se usan para aplicaciones de referenciación lineal y segmentación dinámica. Por ejemplo, las ubicaciones de hitos a lo largo de una carretera pueden contener mediciones que indican su posición. El valor representa todo lo que se puede almacenar como un número de doble precisión.

La función de predicado ST_IsMeasured toma una geometría, y devuelve true si contiene medidas y false si no las tiene. Esta función solo se utiliza con las implementaciones de Oracle y SQLite de ST_Geometry.

Puede averiguar el valor de medición de un punto con la función ST_M.

La función ST_MaxZ devuelve la coordenada m máxima, y la función ST_MinZ devuelve la coordenada m mínima de una geometría.

Tipo de geometría

El tipo de geometría se refiere al tipo de entidad geométrica. Estas incluyen lo siguiente:

- Puntos y multipuntos
- Líneas y multilíneas
- Polígonos y multipolígonos

ST_Geometry es una superclase que puede almacenar varios subtipos. Para determinar a qué subtipo pertenece una geometría, utilice la función ST_GeometryType o ST_Entity (solo Oracle y SQLite).

Colección de puntos (vértice) y cantidad de puntos

Una geometría puede tener cero o más puntos. Una geometría se considera vacía si tiene cero puntos. El subtipo punto es la única geometría que está restringida a cero o un punto; todos los demás subtipos pueden tener cero o más.

ST_Point

Un ST_Point es una geometría de cero dimensión que ocupa una sola ubicación en un espacio de coordenadas. Un ST_Point que tiene un solo valor de coordenada x,y siempre es simple y tiene un límite NULL. ST_Point puede utilizarse para definir entidades, como pozos petroleros, hitos y sitios de recolección de muestras de agua.

Las funciones que operan solamente sobre los tipos de datos ST_Point incluyen lo siguiente:

- ST_X: devuelve el valor de una coordenada x del tipo de dato de punto como un número de doble precisión
- ST_Y: devuelve el valor de una coordenada y del tipo de dato de punto como un número de doble precisión
- ST_Z: devuelve el valor de una coordenada z del tipo de dato de punto como un número de doble precisión
- ST_M: devuelve el valor de una coordenada m del tipo de dato de punto como un número de doble precisión

ST_MultiPoint

Un ST_MultiPoint es un conjunto de ST_Points y, como sus elementos, tiene una dimensión 0. El tipo ST_MultiPoint

Copyright © 1995-2023 Esri. All rights reserved.

es simple si ninguno de sus elementos ocupa el mismo espacio de coordenadas. El límite de un ST_MultiPoint es NULL. ST_MultiPoints pueden definir cosas como patrones de transmisión aérea e incidentes del brote de una enfermedad.

Puede utilizar la función ST_NumGeometries para determinar la cantidad de puntos de una geometría multipunto.

Longitud, área y perímetro

La longitud, área y perímetro son características medibles de geometrías. Las cadenas de texto de líneas y los elementos de las cadena de texto multilínea tienen una sola dimensión y poseen la característica de la longitud. Los polígonos y los elementos de los multipolígonos son superficies de dos dimensiones y, por consiguiente, tienen un área y perímetros que pueden medirse. Puede utilizar las funciones ST_Length, ST_Area y ST_Perimeter para determinar estas propiedades. Las unidades de medida varían según cómo se almacenan los datos.

ST_LineString

Un ST_LineString es un objeto unidimensional almacenado como una secuencia de puntos que define una ruta interpolada lineal. El tipo ST_LineString es simple si no se interseca con el interior. Los extremos (el límite) de un tipo ST_LineString cerrado ocupan el mismo punto en el espacio. Un tipo ST_LineString es un anillo si es cerrado y simple. Al igual que otras propiedades inherentes de ST_Geometry de superclase, ST_LineStrings tiene longitud. ST_LineStrings a menudo se utilizan para definir entidades lineales como carreteras, ríos y líneas de alimentación.

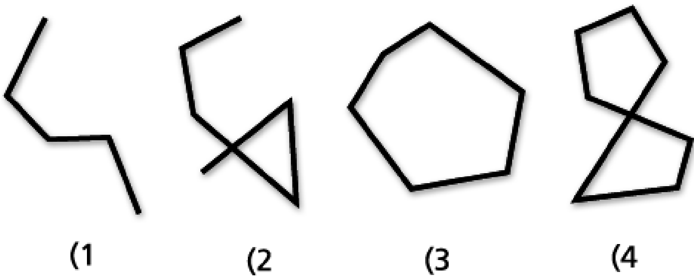
Los extremos normalmente forman el límite de un tipo ST_LineString a menos que éste sea cerrado, en cuyo caso el límite es NULL. El interior de ST_LineString es una ruta conectada que está entre los extremos, a menos que esté cerrada y, en tal caso, el interior es continuo.

Las funciones que operan en ST_LineStrings incluyen lo siguiente:

- ST_StartPoint: devuelve el primer punto de un ST_LineString especificado
- ST_EndPoint: devuelve el último punto de un ST_LineString
- ST_IsClosed: una función de predicado que devuelve true si el ST_LineString especificado está cerrado (el punto de inicio y el extremo de la cadena de líneas se intersecan) y false si no lo está
- ST_IsRing: una función de predicado que devuelve true si el ST_LineString especificado es un anillo y false si no lo es
- ST_Length: devuelve la longitud de un ST_LineString como un número de doble precisión
- ST_NumPoints: evalúa un ST_LineString y devuelve la cantidad de puntos en su secuencia como un número entero
- ST_PointN: toma un ST_LineString y un índice en un punto n y devuelve ese punto

El gráfico siguiente muestra ejemplos de objetos ST_LineString: (1) es un ST_LineString simple y no cerrado; (2) es un ST_LineString que no es simple y no está cerrado; (3) es un ST_LineString cerrado y simple y, por tanto, un anillo; y (4) es un ST_LineString cerrado y que no es simple, pero no es un anillo.

Copyright © 1995-2023 Esri. All rights reserved.

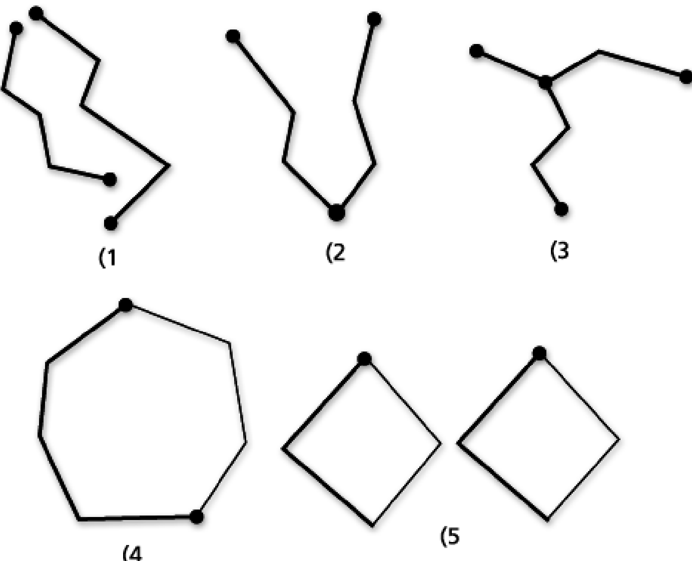


ST_MultiLineString

Un ST_MultiLineString es una colección de ST_LineStrings. Los tipos ST_MultiLineStrings son simples si solo se intersecan en los extremos de los elementos de ST_LineString. Los tipos ST_MultiLineStrings no son simples si los interiores de los elementos de ST_LineString se intersecan.

El límite de un ST_MultiLineString son los extremos no intersecados de los elementos de ST_LineString. El límite de un ST_MultiLineString es NULL si todos los extremos de todos los elementos se intersecan. Además de las demás propiedades inherentes de ST_Geometry de superclase, ST_MultiLineStrings tiene longitud. ST_MultiLineStrings se usa para definir entidades lineales no contiguas, como arroyos o redes de carreteras.

El siguiente gráfico proporciona ejemplos de ST_MultiLineStrings: (1) es un ST_MultiLineString simple para el cual el límite son los cuatro extremos de sus dos elementos ST_LineString. (2) es un ST_MultiLineString simple porque solo se intersecan los extremos de los elementos ST_LineString. El límite son dos extremos no intersecados. (3) es un ST_MultiLineString no simple porque se interseca el interior de uno de sus elementos ST_LineString. El límite de este ST_MultiLineString son los tres extremos no intersecados. (4) es un ST_MultiLineString simple y no cerrado. No está cerrado porque su elemento ST_LineStrings no está cerrado. Es simple porque ninguno de los interiores de ninguno de los elementos ST_LineStrings se intersecan. (5) es un ST_MultiLineString único, simple y cerrado. Está cerrado porque todos sus elementos están cerrados. Es simple porque ninguno de sus elementos se interseca en los interiores.



Las funciones que operan en ST_MultiLineStrings son las siguientes:

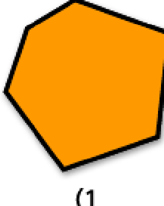
Copyright © 1995-2023 Esri. All rights reserved.

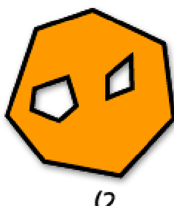
- ST_IsClosed: esta función de predicado devuelve un valor que indica true si el ST_MultiLineString especificado está cerrado, y un valor de false si no lo está.
- ST_Length: esta función evalúa un ST_MultiLineString y devuelve la longitud acumulada de todos sus elementos ST_LineString como un número de doble precisión.
- ST_NumGeometries: esta función devuelve el número de líneas de una cadena de líneas multilínea.

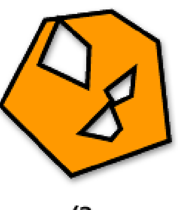
ST_Polygon

Un ST_Polygon es una superficie de dos dimensiones almacenada como una secuencia de puntos que define el anillo de delimitación exterior y 0 o más anillos interiores. Los ST_Polygons son siempre simples. Los ST_Polygons definen entidades que tienen una extensión espacial, como parcelas de tierra, cuerpos de agua y áreas de jurisdicción.

Este gráfico muestra ejemplos de objetos ST_Polygon: 1 es un ST_Polygon cuyo límite viene definido por un anillo exterior. 2 es un ST_Polygon con un límite definido por un anillo exterior y dos anillos interiores. El área dentro de los anillos interiores forma parte del exterior del ST_Polygon. 3 es un ST_Polygon legal porque los anillos se intersecan en un único punto tangente.







La parte exterior y los anillos interiores definen el límite de un tipo ST_Polygon y el espacio cerrado entre los anillos define el interior de ST_Polygon. Los anillos de un tipo ST_Polygon pueden intersecarse en un punto tangente pero nunca cruzarse. Además de las demás propiedades inherentes de la ST_Geometry de superclase, ST_Polygons tiene área.

Las funciones que operan en un ST_Polygon incluyen lo siguiente:

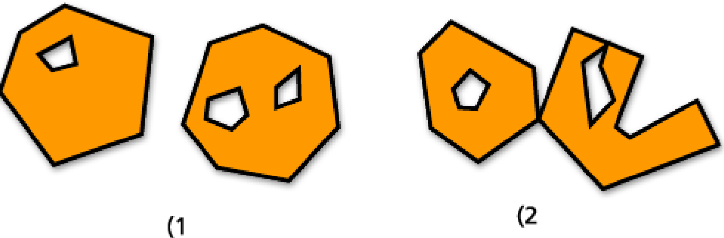
- ST_Area: devuelve el área de un ST_Polygon como un número de precisión doble
- ST_Centroid: devuelve un ST_Point que representa el centro del contorno del ST_Polygon
- ST_ExteriorRing: devuelve el anillo exterior de un ST_Polygon como ST_LineString
- ST_InteriorRingN: evalúa un ST_Polygon y un índice, y devuelve el anillo interior n como un ST_LineString
- ST_NumInteriorRing: devuelve un recuento de anillos interiores que contiene un ST_Polygon
- ST_PointOnSurface: devuelve un ST_Point que se garantiza que está en la superficie del ST_Polygon especificado

ST_MultiPolygon

El límite de un ST_MultiPolygon es la longitud acumulativa del anillo exterior e interior de sus elementos. El interior de un tipo ST_MultiPolygon se define como los interiores acumulados de su elemento ST_Polygons. El límite de los elementos de un tipo ST_MultiPolygon solo se puede intersecar en un punto tangente. Además de las demás propiedades inherentes de la ST_Geometry de superclase, ST_MultiPolygons tiene área. ST_MultiPolygons define las entidades, tales como un estrato de un bosque o una parcela no contigua de tierra, como una cadena de islas del Pacífico.

Copyright © 1995-2023 Esri. All rights reserved.

El gráfico siguiente proporciona ejemplos de ST_MultiPolygon: 1 es un ST_MultiPolygon con dos elementos ST_Polygon. El límite se define mediante los dos anillos exteriores y los tres anillos interiores. 2 también es un ST_MultiPolygon con dos elementos ST_Polygon, pero el límite está definido por los dos anillos exteriores y los dos anillos interiores, y los dos elementos ST_Polygon se intersecan en un punto tangente.



Las funciones que operan en ST_MultiPolygons son las siguientes:

- ST_Area: devuelve un número de doble precisión que representa el ST_Area acumulado de los elementos ST_Polygon de un ST_MultiPolygon.
- ST_Centroid: devuelve un ST_Point que es el centro del sobre de un ST_MultiPolygon.
- ST_NumGeometries: devuelve un recuento del número de polígonos de un multipolígono.
- ST_PointOnSurface: evalúa un ST_MultiPolygon y devuelve un ST_Point que se garantiza que es normal para la superficie de uno de sus elementos ST_Polygon.

Geometrías simples en una geometría multiparte

Las geometrías multiparte están compuestas de geometrías simples individuales.

Puede que quiera determinar cuántas geometrías individuales hay en una geometría multiparte, como un ST_MultiPoint, ST_MultiLineString y ST_MultiPolygon. Para hacer esto, utilice la función de predicado ST_NumGeometries. Esta función devuelve la cuenta de los elementos individuales en un conjunto de geometrías.

Al usar la función ST_GeometryN, puede determinar qué geometría en la geometría multiparte existe en la posición N, siendo N un número que usted proporciona con la función. Por ejemplo, si desea devolver el tercer punto de una geometría multipunto, debería incluir 3 cuando ejecute la función.

Para devolver las geometrías individuales y su posición desde una geometría multiparte en PostgreSQL, use la función ST_GeomFromCollection.

Interior, límite, exterior

Todas las geometrías ocupan una posición en un espacio definido por sus interiores, límites y exteriores. El exterior de una geometría es todo espacio no ocupado por la geometría. El interior es el espacio ocupado por la geometría. El límite de una geometría es la ubicación entre el interior y el exterior. El subtipo hereda las propiedades interiores y exteriores directamente; sin embargo, la propiedad del límite es diferente para cada uno.

Utilice la función ST_Boundary para determinar el límite del ST_Geometry de origen.

Simple o no simple

Algunos subtipos de ST_Geometry son siempre simples, como ST_Points o ST_Polygons. Sin embargo, los subtipos ST_LineStrings, ST_MultiPoints y ST_MultiLineStrings pueden ser simples y no simples. Son simples si obedecen todas las reglas topológicas que se les impone, y no simples si no lo hacen.

Entre las reglas topológicas se encuentran las siguientes:

Copyright © 1995-2023 Esri. All rights reserved.

- El tipo ST_LineString es simple si no se interseca con el interior, y no simple si lo hace.
- Un ST_MultiPoint es simple si no hay dos de sus elementos ocupando el mismo espacio de coordenada (tiene las mismas coordenadas x,y) y es no-simple si lo hacen.
- Un ST_MultiLineString es simple si ninguno de los interiores de sus elementos se intersecan con su propio interior y es no-simple si alguno de los interiores de los elementos se intersecan.

La función de predicado ST_IsSimple se usa para determinar si un ST_LineString, ST_MultiPoint o ST_MultiLineString es simple o no simple. ST_IsSimple toma un ST_Geometry y devuelve true si la geometría es simple, y false si no lo es.

Vacío o no vacío

Una geometría está vacía si no tiene ningún punto. Una geometría vacía tiene sobre, límite, interior y exterior nulos. Una geometría vacía es siempre simple. Las cadenas de líneas y las multicadenas de líneas tienen longitud 0. Los polígonos y los multipolígonos tienen área 0.

La función de predicado ST_IsEmpty puede usarse para determinar si una geometría está vacía. Analiza un ST_Geometry y devuelve true si la geometría está vacía, y false si no lo está.

IsClosed e IsRing

Las geometrías de cadena de líneas pueden ser cerradas o ser anillos. Las cadenas de texto de líneas pueden ser cerradas sin ser anillos. Puede determinar si una cadena de líneas es cerrada mediante la función de predicado ST_IsClosed; devuelve true si el punto de inicio y el extremo de la cadena de líneas se intersecan. Los anillos son cadenas de texto de línea que están cerradas y son simples. La función de predicado ST_IsRing puede usarse si una cadena de líneas es verdaderamente un anillo; devuelve true si la cadena de líneas está cerrada y es simple.

Contorno

Cada geometría tiene un sobre. El sobre de una geometría es la geometría de delimitación formada por las coordenadas x,y mínima y máxima. Para geometrías de punto, dado que las coordenadas x,y máximas y mínimas son las mismas, se crea un rectángulo, o sobre, alrededor de estas coordenadas. Para geometrías de línea, los extremos de la línea representan dos lados del sobre y los otros dos lados se crean justo encima y justo debajo de la línea.

La función ST_Envelope toma un ST_Geometry y devuelve un ST_Geometry que representa el sobre del ST_Geometry de origen.

Para encontrar las coordenadas x,y mínimas y máximas individuales de una geometría, utilice las funciones ST_MinX, ST_MinY, ST_MaxX y ST_MaxY.

Sistema de referencia espacial

El sistema de referencia espacial identifica la matriz de transformación de coordenadas para cada geometría. Está compuesto por un sistema de coordenadas, una resolución y una tolerancia.

Todos los sistemas de referencia espacial conocidos para la geodatabase se almacenan en una tabla del sistema de geodatabase.

Las siguientes funciones obtienen información sobre los sistemas de referencia espacial de las geometrías:

- ST_SRID: toma un ST_Geometry y devuelve su identificador de referencia espacial (SRID) como un número entero.
- ST_Equalsrs: determina si los sistemas de referencia espacial de dos clases de entidades diferentes son idénticos

Copyright © 1995-2023 Esri. All rights reserved.

(true) o no (false).

Tamaño de las entidades (solo PostgreSQL)

Las entidades (registros espaciales en una tabla) toman una cierta cantidad de espacio de almacenamiento en bytes. Puede utilizar la función ST_GeoSize para determinar el tamaño de cada entidad en una tabla.

Definiciones de texto y binarias de una geometría

Para obtener la definición de texto conocido o la definición de binario conocida de la geometría en una fila específica de la tabla espacial, utilice las funciones ST_AsText y ST_AsBinary, respectivamente.

Copyright © 1995-2023 Esri. All rights reserved.

Relaciones espaciales

Una función principal de un SIG es determinar las relaciones espaciales entre entidades: ¿se superponen? ¿Hay una contenida en la otra? ¿Una cruza la otra?

Las geometrías pueden estar relacionadas espacialmente de diferentes maneras. A continuación, se muestran ejemplos de cómo una geometría puede estar relacionada espacialmente con otra:

- La geometría A atraviesa a la geometría B.
- La geometría A está completamente contenida en la geometría B.
- La geometría A contiene completamente la geometría B.
- Las geometrías no se intersecan ni se tocan.
- Las geometrías son completamente coincidentes.
- Las geometrías se superponen.
- Las geometrías se tocan en un punto.

Para determinar si estas relaciones existen o no, utilice funciones de relación espacial. Estas funciones comparan las siguientes propiedades de las geometrías que especifique en una consulta:

- El exterior (E) de las geometrías: el exterior es todo el espacio no ocupado por una geometría.
- El interior (I) de las geometrías: el interior es el espacio ocupado por una geometría.
- El límite (B) de las geometrías: el límite es la interfaz entre el interior de una geometría y su exterior.

Al construir una consulta de relación espacial, especifique el tipo de relación espacial que está buscando y las geometrías que desea comparar. Las consultas devuelven los valores true o false. Es decir, si las geometrías participan entre sí en la relación espacial especificada o no. En la mayoría de los casos, se utiliza una consulta de relación espacial para filtrar un conjunto de resultados colocándolo en la cláusula `WHERE`.

Por ejemplo, si tiene una tabla que almacena las ubicaciones de los emplazamientos de desarrollo propuestos y otra tabla que almacena la ubicación de los yacimientos de importancia arqueológica, puede emitir una consulta para garantizar que ninguno de los emplazamientos de desarrollo se interseque con los yacimientos arqueológicos y, si alguno lo hiciera, devolver el Id. de esos desarrollos propuestos. En este ejemplo, se utiliza la función ST_Disjoint en PostgreSQL.

```
SELECT d.projname,a.siteid
 FROM dev d, archsites a
 WHERE sde.st_disjoint(d.shape,a.shape)= 'f'

projname        siteid

bow wow chow    A1009
```

Esta consulta devuelve el nombre del desarrollo (projname) y el Id. del yacimiento arqueológico (siteid) que no son inconexos, es decir, los emplazamientos que se intersecan. Devuelve un proyecto de desarrollo, Bow Wow Chow, que se interseca con el yacimiento arqueológico A1009.

Copyright © 1995-2023 Esri. All rights reserved.

Funciones relacionales para ST_Geometry

Las funciones relacionales utilizan predicados para probar diferentes tipos de relaciones espaciales. Para ello, las pruebas comparan las relaciones entre lo siguiente:

- El exterior (E) de las geometrías: el exterior es todo el espacio no ocupado por una geometría.
- El interior (I) de las geometrías: el interior es el espacio ocupado por una geometría.
- El límite (B) de las geometrías: el límite es la interfaz entre el interior de una geometría y su exterior.

Los predicados prueban las relaciones. Devuelven 1 (Oracle y SQLite) o t (PostgreSQL) si una comparación cumple con los criterios de la función; de lo contrario, devuelven 0 (Oracle y SQLite) o f (PostgreSQL). Los predicados que prueban una relación espacial comparan pares de geometría que pueden ser de diferente tipo o dimensión.

Los predicados comparan las coordenadas x e y de las geometrías enviadas. Las coordenadas z y los valores de medición, si existen, se omiten. Las geometrías que tienen medidas o coordenadas z se pueden comparar con aquellas que no las tienen.

El Modelo de 9 intersecciones dimensionalmente extendido (DE-9IM) desarrollado por Clementini, et al., amplía de manera dimensional el Modelo de 9 intersecciones de Egenhofer y Herring. El DE-9IM es un enfoque matemático que define la relación espacial de pares entre geometrías de distintos tipos y dimensiones. Este modelo expresa las relaciones espaciales entre todos los tipos de geometría como intersecciones de pares de su interior, límite y exterior, teniendo en cuenta la dimensión de las intersecciones resultantes.

Dadas las geometrías a y b, I(a), B(a) y E(a) representan el interior, el límite y el exterior de a, e I(b), B(b) y E(b) representan el interior, el límite y el exterior de b. Las intersecciones de I(a), B(a) y E(a) con I(b), B(b) y E(b) producen una matriz de tres por tres. Cada intersección puede resultar en geometrías de diferentes dimensiones. Por ejemplo, la intersección de los límites de dos polígonos podría consistir en un punto y una cadena de líneas, en cuyo caso la función dim (dimensión) devolvería la dimensión máxima de 1.

La función dim devuelve un valor de -1, 0, 1 o 2. El -1 corresponde al conjunto nulo que se devuelve cuando no se encuentra ninguna intersección o dim(Ãƒ˜).

	Interior	**Límite**	**Exterior**
Interior	dim(I(a) interseca I(b))	dim(I(a) interseca B(b))	dim(I(a) interseca E(b))
Límite	dim(B(a) interseca I(b))	dim(B(a) interseca B(b))	dim(B(a) interseca E(b))
Exterior	dim(E(a) interseca I(b))	dim(E(a) interseca B(b))	dim(E(a) interseca E(b))

Ejemplo de intersección de un predicado

Los resultados de los predicados de relación espacial se pueden entender o verificar comparando los resultados del predicado con una matriz de patrón que representa los valores aceptables para el DE-9IM.

La matriz de patrón contiene los valores aceptables para cada una de las celdas de la matriz de intersección. Los valores posibles del patrón son los siguientes:

T: debe existir una intersección; dim = 0, 1 o 2

F: no debe existir ninguna intersección; dim = -1

*: no importa si existe una intersección o no; dim = -1, 0, 1 o 2

0: debe existir una intersección y su dimensión máxima debe ser 0; dim = 0

1: debe existir una intersección y su dimensión máxima debe ser 1; dim = 1

Copyright © 1995-2023 Esri. All rights reserved.

2: debe existir una intersección y su dimensión máxima debe ser 2; dim = 2

Cada predicado tiene al menos una matriz de patrón, pero algunos requieren más de una para describir las relaciones de varias combinaciones de tipos de geometría.

La matriz de patrón del predicado ST_Within para las combinaciones de geometría tiene la siguiente forma:

			Geometría b	
		Interior	**Límite**	**Exterior**
Geometría a	**Interior**	T	*	F
	Límite	*	*	F
	Exterior	*	*	*

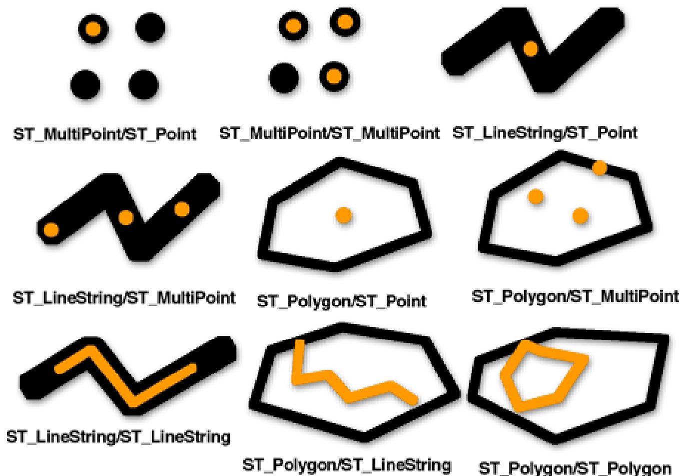
Ejemplo de matriz de patrón

El predicado ST_Within tiene el valor true cuando los interiores de ambas geometrías se intersecan, y el interior y el límite de la geometría a no se intersecan con el exterior de la geometría b. El resto de condiciones no son relevantes.

Las siguientes secciones describen diferentes predicados utilizados para relaciones espaciales. En los diagramas de estas secciones, la primera geometría de entrada se muestra en negro y la segunda se representa en naranja.

ST_Contains

ST_Contains devuelve 1 o t (true) si la segunda geometría está completamente contenida en la primera geometría. El predicado ST_Contains devuelve el resultado exactamente opuesto al predicado ST_Within.



La matriz de patrón del predicado ST_Contains establece que los interiores de ambas geometrías se deben intersecar y que el interior y el límite de la secundaria (geometría b) no deben intersecar el exterior de la principal (geometría a).

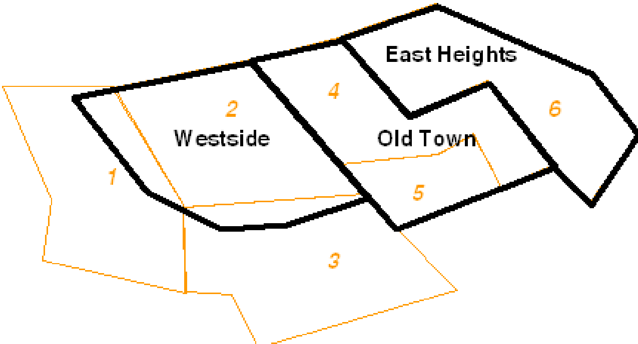
			Geometría b	
		Interior	**Límite**	**Exterior**
Geometría a	**Interior**	T	*	*

Matriz de ST_Contains

Copyright © 1995-2023 Esri. All rights reserved.

	Límite	*	*	*
	Exterior	F	F	*

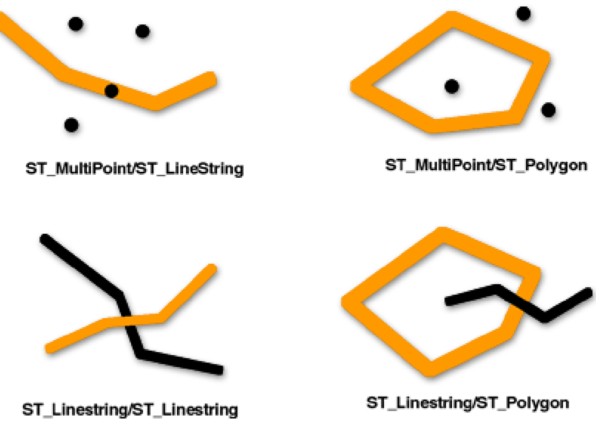
Las funciones ST_Within y ST_Contains identifican solo aquellas geometrías que están completamente dentro de otra geometría. Esto ayuda a eliminar de la selección entidades que podrían distorsionar los resultados. En el siguiente ejemplo, un vendedor ambulante de helados desea determinar qué vecindades tienen la mayor cantidad de niños (clientes potenciales) y restringir su ruta a esas áreas. El vendedor compara los polígonos de las vecindades designadas con los distritos censales, que poseen un atributo de un número total de niños menores de 16.



A menos que todos los niños que viven en el distrito censal 1 y el distrito censal 3 vivan en los falsos polígonos que quedan dentro del Westside, incluir estos distritos en la selección podría elevar erróneamente el recuento de niños en la vecindad del Westside. Al especificar que solo se incluyan los distritos censales que estén totalmente comprendidos en las vecindades (ST_Within = 1), el vendedor de helados ahorra potencialmente tiempo y dinero al no aventurarse a esas partes del Westside.

ST_Crosses

ST_Crosses devuelve 1 o t (true) si la intersección resulta en una geometría que tiene una dimensión que es uno menos que la dimensión máxima de las dos geometrías de origen y el conjunto de intersección es interior en ambas geometrías de origen. ST_Crosses devuelve 1 o t (true) solo para las comparaciones ST_MultiPoint/ST_Polygon, ST_MultiPoint/ST_LineString, ST_Linestring/ST_LineString, ST_LineString/ST_Polygon y ST_LineString/ST_MultiPolygon.



La siguiente matriz de patrón del predicado ST_Crosses se aplica a ST_MultiPoint/ST_LineString, ST_MultiPoint/ST_MultiLineString, ST_MultiPoint/ST_Polygon, ST_MultiPoint/ST_MultiPolygon, ST_LineString/ST_Polygon y

Copyright © 1995-2023 Esri. All rights reserved.

ST_LineString/ST_MultiPolygon. La matriz establece que los interiores se deben intersecar y que al menos el interior de la principal (geometría a) debe intersecar el exterior de la secundaria (geometría b).

			Geometría b	
		Interior	**Límite**	**Exterior**
Geometría a	**Interior**	T	*	T
	Límite	*	*	*
	Exterior	*	*	*

Matriz 1 de ST_Crosses

La siguiente matriz del predicado ST_Crosses se aplica a ST_LineString/ST_LineString, ST_LineString/ ST_MultiLineString y ST_MultiLineString/ST_MultiLineString. La matriz establece que la dimensión de la intersección de los interiores debe ser 0 (interseca en un punto). Si la dimensión de esta intersección fuera 1 (intersecara en una cadena de líneas), el predicado ST_Crosses devolvería false, pero el predicado ST_Overlaps devolvería true.

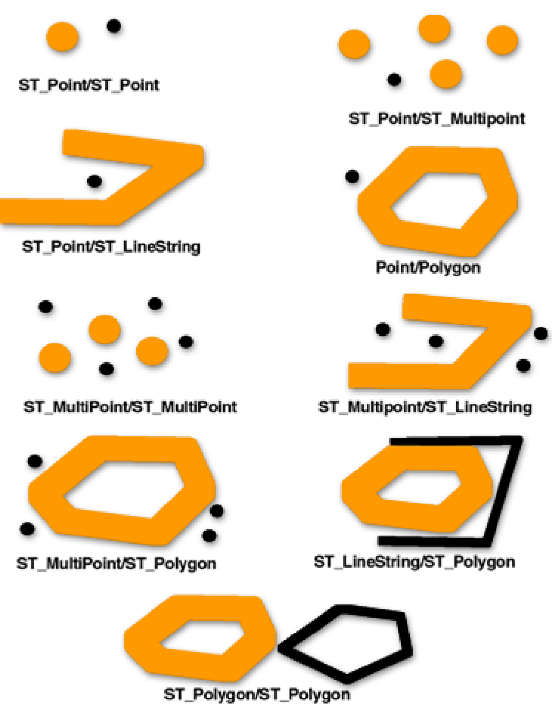
			Geometría b	
		Interior	**Límite**	**Exterior**
Geometría a	**Interior**	0	*	*
	Límite	*	*	*
	Exterior	*	*	*

Matriz 2 de ST_Crosses

ST_Disjoint

ST_Disjoint devuelve 1 o t (true) si la intersección de las dos geometrías es un conjunto vacío. En otras palabras, las geometrías son inconexas si no se intersecan entre sí.

Copyright © 1995-2023 Esri. All rights reserved.



La matriz de patrón del predicado ST_Disjoint establece que no se intersecan ni los interiores ni los límites de ninguna de las dos geometrías.

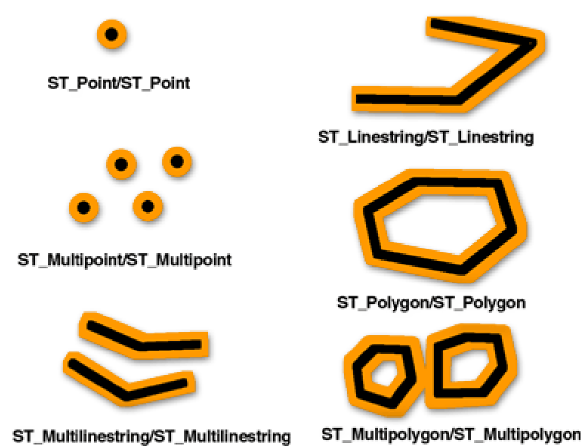
			Geometría b	
		Interior	**Límite**	**Exterior**
Geometría a	**Interior**	**F**	**F**	*
	Límite	**F**	**F**	*
	Exterior	*	*	*

Matriz de ST_Disjoint

ST_Equals

ST_Equals devuelve 1 o t (true) si dos geometrías del mismo tipo tienen valores de coordenadas x,y idénticos. La primera y segunda planta de un edificio de oficinas podrían tener coordenadas x,y idénticas y, por lo tanto, ser iguales. ST_Equals también puede identificar si dos entidades se colocaron por error una encima de la otra.

Copyright © 1995-2023 Esri. All rights reserved.



La matriz de patrón DE-9IM de igualdad garantiza que los interiores se intersecan y que ninguna parte del interior o el límite de cualquiera de las dos geometrías interseca el exterior de la otra.

			Geometría b	
		Interior	**Límite**	**Exterior**
Geometría a	**Interior**	T	*	F
	Límite	*	*	F
	Exterior	F	F	*

Matriz de ST_Equals

ST_Intersects

ST_Intersects devuelve 1 o t (true) si la intersección no resulta en un conjunto vacío. ST_Intersects devuelve el resultado exactamente opuesto a ST_Disjoint.

El predicado ST_Intersects devuelve true si las condiciones de alguna de las siguientes matrices de patrón devuelven true.

El predicado ST_Intersects devuelve true si los interiores de ambas geometrías se intersecan.

			Geometría b	
		Interior	**Límite**	**Exterior**
Geometría a	**Interior**	T	*	*
	Límite	*	*	*
	Exterior	*	*	*

Matriz 1 de ST_Intersects

El predicado ST_Intersects devuelve true si el interior de la primera geometría interseca el límite de la segunda geometría.

			Geometría b	
		Interior	**Límite**	**Exterior**

Matriz 2 de ST_Intersects

Copyright © 1995-2023 Esri. All rights reserved.

Geometría a	**Interior**	*	T	*
	Límite	*	*	*
	Exterior	*	*	*

El predicado ST_Intersects devuelve true si el límite de la primera geometría interseca el interior de la segunda.

			Geometría b	
		Interior	**Límite**	**Exterior**
Geometría a	**Interior**	*	*	*
	Límite	T	*	*
	Exterior	*	*	*

Matriz 3 de ST_Intersects

El predicado ST_Intersects devuelve true si los límites de cualquiera de las dos geometrías se intersecan.

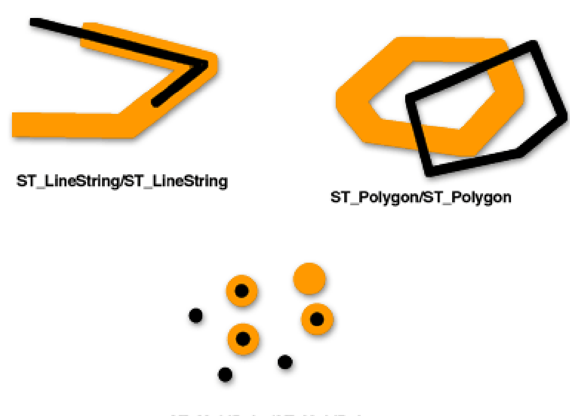
			Geometría b	
		Interior	**Límite**	**Exterior**
Geometría a	**Interior**	*	*	*
	Límite	*	T	*
	Exterior	*	*	*

Matriz 4 de ST_Intersects

ST_Overlaps

ST_Overlaps compara dos geometrías de la misma dimensión y devuelve 1 o t (true) si su conjunto de intersección resulta en una geometría distinta a las dos geometrías de entrada, pero es de la misma dimensión.

ST_Overlaps devuelve 1 o t (true) solo para geometrías de la misma dimensión y solo cuando su conjunto de intersección resulta en una geometría de la misma dimensión. Es decir, si la intersección de dos ST_Polygons resulta en un ST_Polygon, la superposición devuelve 1 o t (true).



Esta matriz de patrón se aplica a superposiciones ST_Polygon/ST_Polygon, ST_MultiPoint/ST_MultiPoint y

Copyright © 1995-2023 Esri. All rights reserved.

ST_MultiPolygon/ST_MultiPolygon. Para estas combinaciones, el predicado de superposición devuelve true si el interior de ambas geometrías interseca el interior y el exterior de la otra.

			Geometría b	
		Interior	**Límite**	**Exterior**
Geometría a	**Interior**	T	*	T
	Límite	*	*	*
	Exterior	T	*	*

Matriz 1 de ST_Overlaps

La siguiente matriz de patrón se aplica a superposiciones ST_LineString/ST_LineString y ST_MultiLineString/ST_MultiLineString. En este caso, la intersección de las geometrías debe dar como resultado una geometría que tenga una dimensión de 1 (otro ST_LineString o ST_MultiLineString). Si la dimensión de la intersección de los interiores diera como resultado 0 (un punto), el predicado ST_Overlaps devolvería false. Sin embargo, el predicado ST_Crosses devolvería true.

			Geometría b	
		Interior	**Límite**	**Exterior**
Geometría a	**Interior**	1	*	T
	Límite	*	*	*
	Exterior	T	*	*

Matriz 2 de ST_Overlaps

ST_Relate

ST_Relate devuelve un valor de 1 o t (true) si la relación espacial especificada por la matriz de patrón es válida. Un valor de 1 o t (true) indica que existe algún tipo de relación espacial entre las geometrías.

Si los interiores o límites de las geometrías a y b están relacionados de alguna manera, ST_Relate es true. No importa si los exteriores de una geometría intersecan el interior o el límite de la otra.

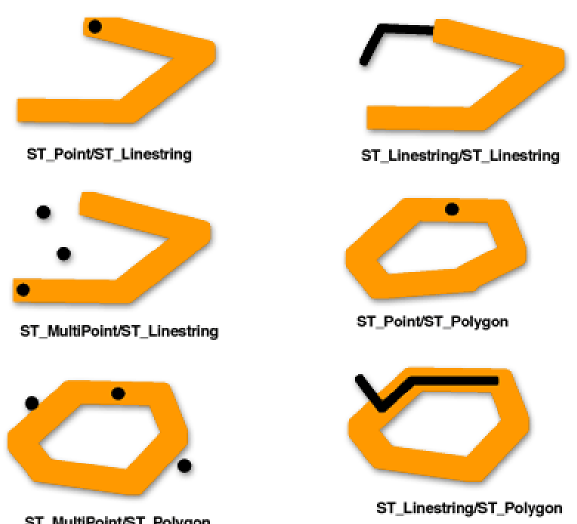
			Geometría b	
		Interior	**Límite**	**Exterior**
Geometría a	**Interior**	T	T	*
	Límite	T	T	*
	Exterior	*	*	*

Matriz de ST_Relate

ST_Touches

ST_Touches devuelve 1 o t (true) si ninguno de los puntos comunes a ambas geometrías se interseca con los interiores de ambas geometrías. Al menos una geometría debe ser ST_LineString, ST_Polygon, ST_MultiLineString o ST_MultiPolygon.

Copyright © 1995-2023 Esri. All rights reserved.



Las matrices de patrón muestran que el predicado ST_Touches devuelve el valor true cuando los interiores de la geometría no se intersecan, y el límite de cualquiera de las dos geometrías interseca el interior o el límite de la otra.

El predicado ST_Touches devuelve true si el límite de la geometría b interseca el interior de la geometría a, pero los interiores no se intersecan.

			Geometría b	
		Interior	**Límite**	**Exterior**
Geometría a	**Interior**	F	T	*
	Límite	*	*	*
	Exterior	*	*	*

Matriz 1 de ST_Touches

El predicado ST_Touches devuelve true si el límite de la geometría a interseca el interior de la geometría b, pero los interiores no se intersecan.

			Geometría b	
		Interior	**Límite**	**Exterior**
Geometría a	**Interior**	F	*	*
	Límite	T	*	*
	Exterior	*	*	*

Matriz 2 de ST_Touches

El predicado ST_Touches devuelve true si los límites de ambas geometrías se intersecan, pero los interiores no.

			Geometría b	
		Interior	**Límite**	**Exterior**
Geometría a	**Interior**	F	*	*
	Límite	*	T	*

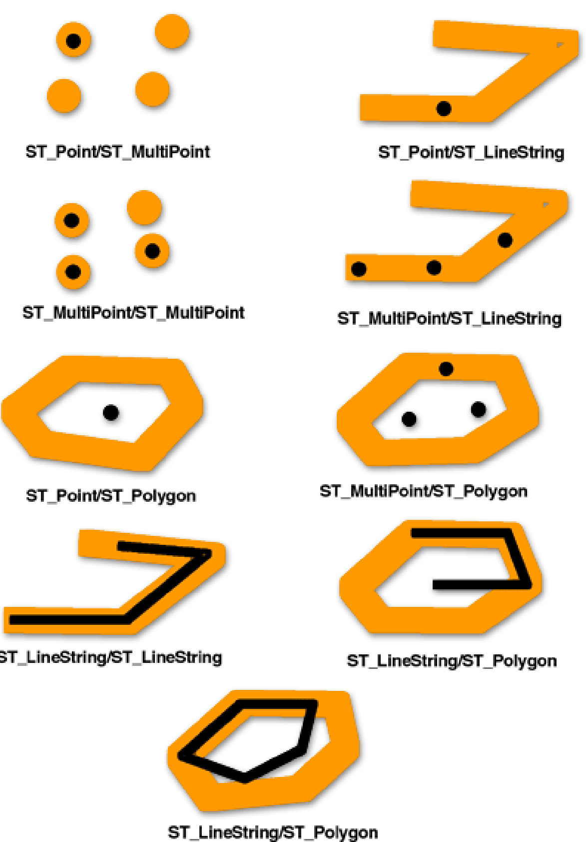
Matriz 3 de ST_Touches

Copyright © 1995-2023 Esri. All rights reserved.

	Exterior	*	*	*

ST_Within

ST_Within devuelve 1 o t (true) si la primera geometría está completamente dentro de la segunda geometría. ST_Within prueba el resultado exactamente opuesto de ST_Contains.



La matriz de patrón del predicado ST_Within establece que los interiores de ambas geometrías se deben intersecar y que el interior y el límite de la geometría principal (geometría a) no deben intersecar el exterior de la geometría secundaria (geometría b).

			Geometría b	
		Interior	**Límite**	**Exterior**
Geometría a	**Interior**	T	*	F
	Límite	*	*	F
	Exterior	*	*	*

Matriz de ST_Within

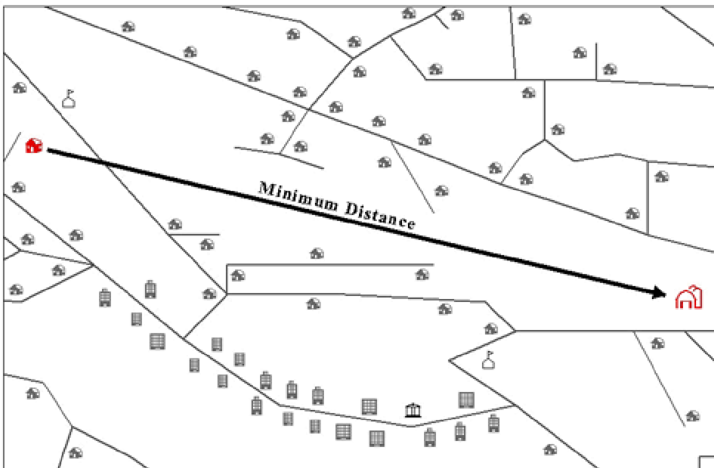
Otras relaciones espaciales

Las funciones siguientes comparan la relación espacial entre geometrías, pero comparan algo más que solo el interior, el límite y los exteriores de las geometrías.

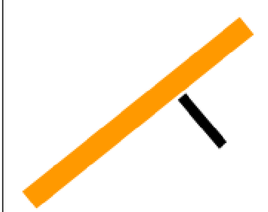
- ST_Distance: esta función toma dos geometrías inconexas como entrada y devuelve la distancia mínima entre

Copyright © 1995-2023 Esri. All rights reserved.

ambas. Si las geometrías no son inconexas (es decir, son coincidentes), la función reporta una distancia mínima de cero.
La distancia mínima que separa las entidades representa la distancia más corta entre dos ubicaciones. Por ejemplo, esta no es la distancia que recorrería si condujera de una ubicación a otra, sino la distancia que calcularía si trazara una línea recta entre dos ubicaciones en un mapa.



- ST_DWithin: usted proporciona un valor de distancia junto con las geometrías que se van a comparar. ST_DWithin devuelve true si las geometrías se encuentran dentro de la distancia especificada entre sí.
- ST_EnvIntersects: esta función evalúa si se intersecan los sobres espaciales de las geometrías especificadas, mientras que ST_Intersects evalúa si se intersecan las propias geometrías.
  En el siguiente ejemplo, los sobres de las dos líneas se intersecan, pero las líneas en sí no:



- ST_OrderingEquals: esta función amplía la comparación realizada por ST_Equals para comparar también que las coordenadas de las geometrías están definidas en el mismo orden (x,y frente a y,x). Aunque las geometrías ocupen el mismo espacio, si sus coordenadas x e y no están definidas en el mismo orden, ST_OrderingEquals devuelve false.

Copyright © 1995-2023 Esri. All rights reserved.

Operaciones espaciales

Las operaciones espaciales utilizan funciones de geometría para tomar datos espaciales como entrada, los analizan y, después, producen datos de salida que derivan del análisis realizado en los datos de entrada.

Entre los datos derivados que puede obtener de una operación espacial se incluyen los siguientes:

- Un polígono que es una zona de influencia alrededor de una entidad de entrada
- Una entidad única que es el resultado del análisis realizado en un conjunto de geometrías
- Una entidad única que es el resultado de una comparación para determinar la parte de una entidad que no ocupa el mismo espacio físico que otra entidad
- Una entidad única que es el resultado de una comparación para buscar las partes de una entidad que intersecan el espacio físico de otra entidad
- Una entidad multiparte que está compuesta por las partes de ambas entidades de entrada que no ocupan el mismo espacio físico que la otra
- Una entidad que es la combinación de dos geometrías

El análisis realizado en los datos de entrada devuelve las coordenadas o la representación textual de las geometrías resultantes. Puede utilizar esa información como parte de una consulta más grande para realizar más análisis o puede utilizar los resultados como entrada de otra tabla.

Por ejemplo, puede incluir una operación de zona de influencia en la cláusula `WHERE` de una consulta de intersección para determinar si la geometría especificada interseca un área de tamaño especificado alrededor de otra geometría.

Nota:
Los siguientes ejemplos utilizan funciones ST_Geometry. Para consultar las funciones de geometría y sintaxis específicas utilizadas para otra base de datos y tipo de datos espaciales, lea la documentación específica de esa base de datos y tipo de datos.

En este ejemplo, se deben enviar notificaciones a todos los propietarios en un radio de 1.000 pies de una calle cortada al tráfico. La cláusula `WHERE` genera una zona de influencia de 1.000 pies alrededor de la calle que se cortará al tráfico. A continuación, la zona de influencia se compara con las propiedades del área para ver cuáles son intersecadas por la zona de influencia.

```
SELECT p.owner,p.address,s.stname
FROM parcels p, streets s
WHERE s.stname = 'Main'
AND sde.st_intersects (p.shape, sde.st_buffer (s.shape, 1000)) = 't';
```

En este ejemplo, se elige una calle específica (Main) en la cláusula `WHERE` y, a continuación, se crea una zona de influencia alrededor de la calle y se compara con las entidades de la tabla de parcelas para determinar si se intersecan o no.* Para todas las parcelas intersecadas por la zona de influencia en Main Street, se devuelven el nombre y la dirección del propietario de la parcela.

Copyright © 1995-2023 Esri. All rights reserved.

Nota:
*El orden en el que se ejecutan las partes de la cláusula `WHERE` depende del optimizador de la base de datos.

A continuación, se proporciona un ejemplo donde se toman los resultados de una operación espacial (combinación) realizada en tablas que contienen áreas de vecindad y distrito escolar, y se insertan las entidades resultantes en otra tabla:

```
INSERT INTO combo c (shape)
VALUES (
(SELECT sde.st_union (n.shape,d.shape)
FROM neighborhoods n, school_districts d),5);
```

Para obtener más información sobre el uso de operadores espaciales con ST_Geometry, consulte Funciones de operación espacial para ST_Geometry.

Copyright © 1995-2023 Esri. All rights reserved.

Funciones de operación espacial para ST_Geometry

Las operaciones espaciales utilizan funciones de geometría para tomar datos espaciales como entrada, los analizan y, después, producen datos de salida que derivan del análisis realizado en los datos de entrada.

Puede realizar las operaciones que se describen en las siguientes secciones para crear entidades a partir de entidades de entrada.

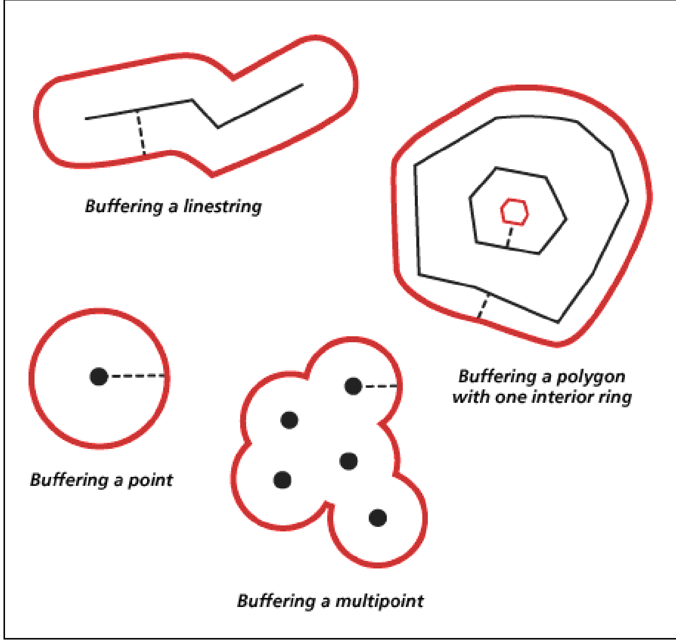
Geometría de creación de zonas de influencia

La función ST_Buffer genera una geometría rodeando la geometría especificada a la distancia que especifique. Se genera un solo polígono cuando se crea una zona de influencia de una geometría principal o cuando los polígonos de zona de influencia de un conjunto están lo suficientemente cerca como para superponerse. Cuando existe suficiente separación entre los elementos de un conjunto con zona de influencia, los ST_Polygons de zona de influencia individuales dan como resultado un ST_MultiPolygon.

La función ST_Buffer acepta distancias positivas y negativas, pero puede aplicar distancias negativas solo a geometrías de dos dimensiones (ST_Polygon y ST_MultiPolygon). ST_Buffer utiliza el valor absoluto de la distancia de zona de influencia cuando la geometría de origen tiene menos de dos dimensiones, es decir, todas las geometrías que no son ni ST_Polygon ni ST_MultiPolygon. Las distancias de zona de influencia positivas generan anillos de polígono que se alejan del centro de la geometría de origen y, en el caso del anillo exterior de un ST_Polygon o ST_MultiPolygon, se acercan al centro cuando la distancia es negativa. En el caso de los anillos interiores de un ST_Polygon o ST_MultiPolygon, el anillo de zona de influencia se acerca al centro cuando la distancia de zona de influencia es positiva y se aleja del centro cuando es negativa.

El proceso de creación de zonas de influencia fusiona los polígonos de zona de influencia que se superponen. Las distancias negativas mayores que la mitad del ancho máximo interior de un polígono dan como resultado una geometría vacía.

En el siguiente diagrama, las zonas de influencia están dibujadas en rojo.



Copyright © 1995-2023 Esri. All rights reserved.

ConvexHull

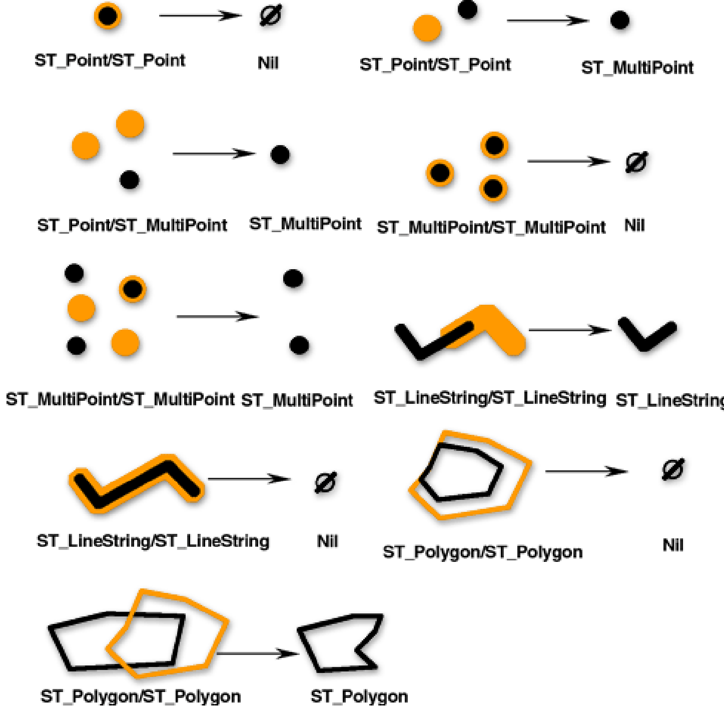
La función ST_ConvexHull devuelve el polígono de envoltura convexa de cualquier geometría que tenga al menos tres vértices que formen un convexo. Si los vértices de la geometría no forman un convexo, ST_ConvexHull devuelve un valor nulo. Por ejemplo, si utiliza ST_ConvexHull en una línea compuesta por dos vértices, se devolverá un valor nulo. Del mismo modo, si utiliza la operación ST_ConvexHull en una entidad de punto, se devolverá un valor nulo. Crear una envoltura convexa suele ser el primer paso al teselar un conjunto de puntos para crear una red irregular de triángulos (TIN).

Diferencia de geometrías

La función ST_Difference devuelve la porción de la geometría principal que no está intersecada por la geometría secundaria. Este es el valor lógico `AND NOT` del espacio.

La función ST_Difference opera solamente en geometrías de dimensión similar y devuelve un conjunto que tiene la misma dimensión que las geometrías de origen. Si las geometrías de origen son iguales, se devuelve una geometría vacía.

En el siguiente diagrama, las primeras geometrías de entrada son de color negro y las segundas geometrías de entrada son de color naranja.



Intersección de geometrías

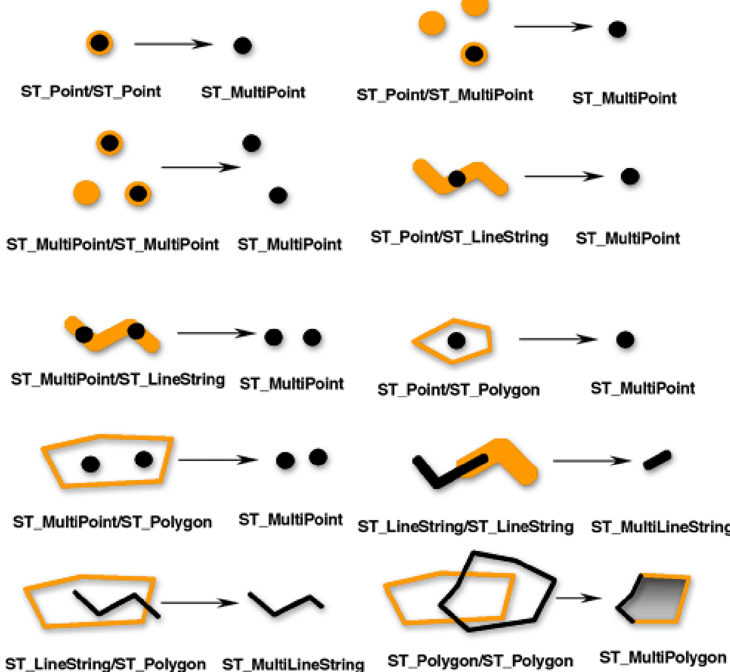
La función ST_Intersection devuelve el conjunto de intersección de dos geometrías. El conjunto de intersección siempre se devuelve como un conjunto que es la dimensión mínima de las geometrías de origen.

Por ejemplo, para un ST_LineString que interseca un ST_Polygon, la función ST_Intersection devuelve la porción de ST_LineString común al interior y el límite del ST_Polygon como un ST_MultiLineString. El ST_MultiLineString contiene más de un ST_LineString si el ST_LineString de origen intersecaba el ST_Polygon con dos o más segmentos

Copyright © 1995-2023 Esri. All rights reserved.

discontinuos. Si las geometrías no se intersecan o si la intersección da como resultado una dimensión menor que ambas geometrías de origen, se devuelve una geometría vacía.

La siguiente figura ilustra ejemplos de la función ST_Intersection. Las primeras geometrías de entrada son de color negro y las segundas geometrías de entrada son de color naranja.



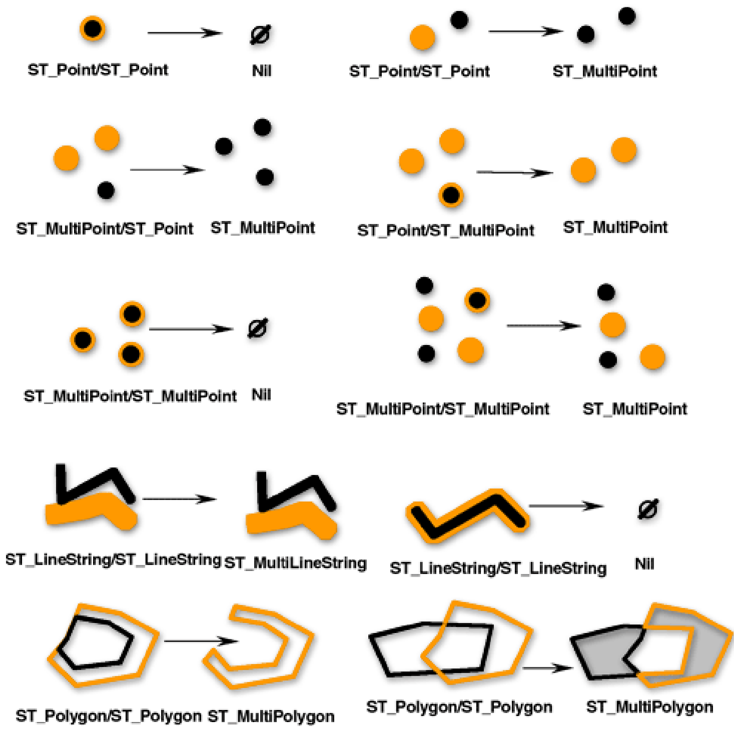
Diferencia simétrica de geometrías

La función ST_SymmetricDiff devuelve las partes de las geometrías de origen que no son parte del conjunto de intersección. Este es el valor lógico XOR del espacio.

Las geometrías de origen deben tener la misma dimensión. Si las geometrías son iguales, la función ST_SymmetricDiff devuelve una geometría vacía; de lo contrario, la función devuelve el resultado como un conjunto.

En el siguiente diagrama, las primeras geometrías de entrada son de color negro y las segundas geometrías de entrada son de color naranja.

Copyright © 1995-2023 Esri. All rights reserved.

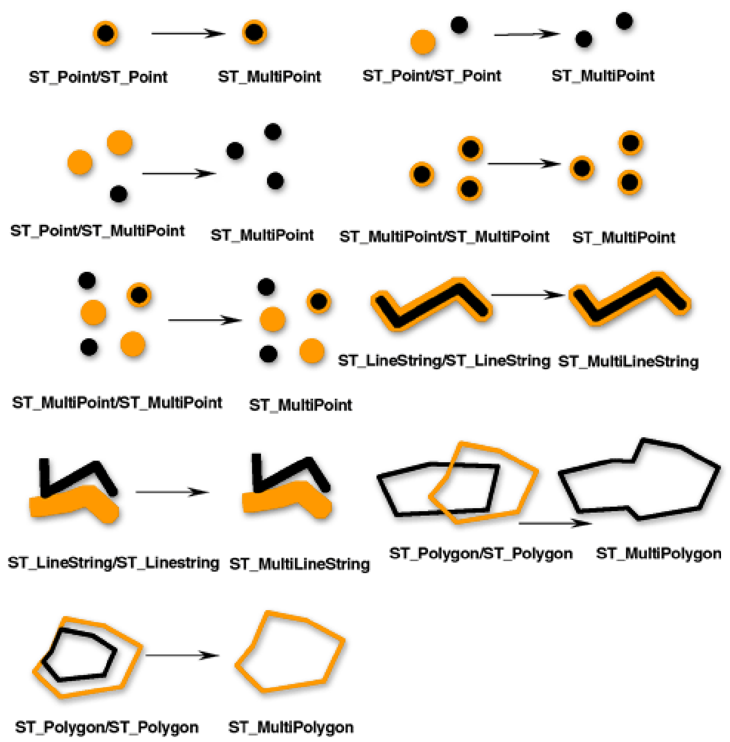


Combinación de geometrías

La función ST_Union devuelve el conjunto de combinación de dos geometrías. Este es el valor lógico booleano OR del espacio. Las geometrías de origen deben tener la misma dimensión. ST_Union siempre devuelve el resultado como un conjunto.

En el siguiente diagrama, las primeras geometrías de entrada son de color negro y las segundas geometrías de entrada son de color naranja.

Copyright © 1995-2023 Esri. All rights reserved.



Agregaciones

Las operaciones de agregación devuelven una sola geometría como resultado del análisis realizado en un conjunto de geometrías. La función ST_Aggr_ConvexHull devuelve el multipolígono compuesto por los polígonos de envoltura convexa de cada una de las geometrías de entrada. Cualquier geometría de entrada con menos de tres vértices no tendrá una envoltura convexa. Si todas las geometrías de entrada tienen menos de tres vértices, ST_Aggr_ConvexHull devuelve un valor nulo.

La función ST_Aggr_Intersection devuelve una sola geometría que es una agregación de las intersecciones de todas las geometrías de entrada.

ST_Aggr_Intersection busca la intersección de varias geometrías, mientras que ST_Intersection solo busca la intersección entre dos geometrías. Por ejemplo, si quiere buscar una propiedad que tenga cobertura para distintos servicios específicos (como un distrito escolar, un servicio telefónico y un proveedor de Internet de alta velocidad específicos) y que esté representada por un concejal o concejala concretos, debe buscar la intersección de todas esas áreas. Si busca la intersección de solo dos de esas áreas, no obtendrá toda la información que necesita, por lo que debe usar la función ST_Aggr_Intersection para que todas las áreas se puedan evaluar en la misma consulta.

Como ejemplo adicional, si busca las intersecciones de las líneas y puntos de dos clases de entidad, cada función devuelve lo siguiente:

- ST_Intersection: se devuelve una geometría ST_Point para cada intersección.
- ST_Aggr_Intersection: se devuelve una geometría ST_MultiPoint compuesta por todos los puntos de intersección. (Sin embargo, si solo se intersecan una entidad de punto y una entidad de línea, obtendrá una geometría ST_Point).

La función ST_Aggr_Union devuelve una geometría que es la combinación de todas las geometrías proporcionadas.

Las geometrías de entrada deben ser del mismo tipo; por ejemplo, puede combinar ST_LineStrings con

Copyright © 1995-2023 Esri. All rights reserved.

ST_LineStrings o puede combinar ST_Polygons con ST_Polygons, pero no puede combinar una clase de entidad ST_LineString con una clase de entidad ST_Polygon.

La geometría que resulta de la combinación agregada es normalmente un conjunto. Por ejemplo, si desea la combinación agregada de todas las parcelas vacantes de menos de medio acre, la geometría devuelta será un multipolígono, a menos que todas las parcelas que cumplan esos criterios sean contiguas, en cuyo caso se devolverá un polígono.

Distancia mínima

Las funciones anteriores devolvieron geometrías nuevas. La función ST_Distance realiza una operación espacial (evalúa la distancia mínima entre dos geometrías), pero no devuelve una geometría nueva.

Copyright © 1995-2023 Esri. All rights reserved.

Círculos, elipses y porciones de círculos paramétricos

Puede crear y consultar círculos, elipses y porciones de círculos paramétricos en columnas de ST_Geometry con la función ST_Geometry.

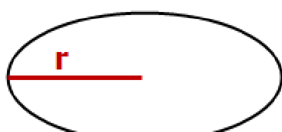
Los círculos, las elipses y las porciones de círculos paramétricos son polígonos definidos por parámetros específicos, tales como valores de coordenadas, ángulos y radios. Las bases de datos almacenan estos parámetros en lugar de vértices y líneas específicas. Al almacenar los parámetros que definen la forma, las formas paramétricas pueden ser más precisos y utilizar menos espacio de almacenamiento que si se almacenan como representaciones de polígonos con varios lados. El uso de formas paramétricas además permite la inclusión de parámetros de coordenada z y valor m de medida.

Se esperan siete parámetros cuando crea un círculo:

- Un valor de coordenada x del punto central del círculo
- Un valor de coordenada y del punto central del círculo
- Un valor de coordenada z del punto central del círculo
- Un valor m
- Radio del círculo que se va a crear
- Número de puntos utilizados para definir el círculo
  La cantidad mínima de puntos que puede especificar es 9. Si no especifica una cantidad de puntos, se utilizarán 50 puntos por defecto. Esos puntos no se almacenarán con la forma pero se generarán cuando se genere el círculo para validar la forma.
- Id. de referencia espacial (SRID) utilizado para ubicar el círculo en el espacio

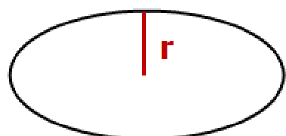
Se esperan nueve parámetros cuando se crea una elipse:

- Un valor de coordenada x del punto central de la elipse
- Un valor de coordenada y del punto central de la elipse
- Un valor de coordenada z del punto central de la elipse
- Un valor m
- El semieje mayor de la elipse
  El semieje mayor es el radio más largo de una elipse. El valor especificado para el semieje mayor debe ser mayor que el semieje menor.



- El semieje menor de la elipse
  El semieje menor es el radio más corto de una elipse. El valor especificado para el semieje menor debe ser mayor que 0,0.

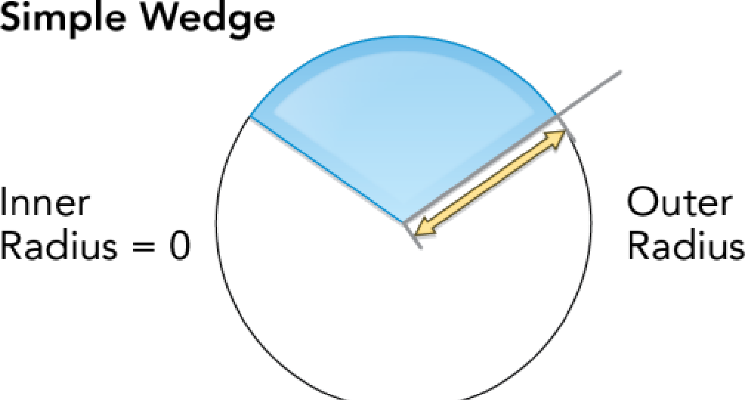
Copyright © 1995-2023 Esri. All rights reserved.



- El ángulo de rotación de la elipse
  El valor especificado para el ángulo de rotación se especifica en grados y debe ser mayor que 0,0, pero menor que 360. La rotación se realiza en el sentido de la agujas del reloj.
- Número de puntos utilizados para definir la elipse
  La cantidad mínima de puntos que puede especificar es 9. Si no especifica una cantidad de puntos, se utilizarán 50 puntos por defecto. Esos puntos no se almacenarán con la forma pero se generarán cuando se genere la elipse para validar la forma.
- SRID utilizado para ubicar la elipse en el espacio

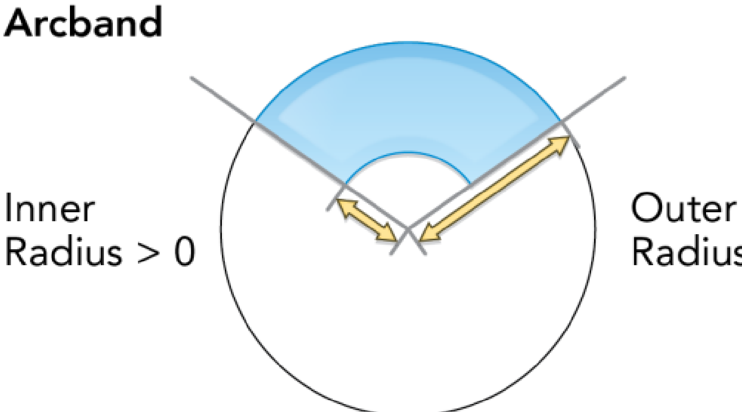
Se esperan diez parámetros cuando crea una porción de círculo:

- Un valor de coordenada x del punto central del círculo que define la porción de círculo
- Un valor de coordenada y del punto central del círculo que define la porción de círculo
- Un valor de coordenada z del punto central del círculo que define la porción de círculo
- Un valor m
- El ángulo de inicio de la porción del círculo
  El ángulo de inicio define el inicio de la porción del círculo debido a que la cantidad de grados se miden en el sentido contrario a las agujas del reloj desde 0 grados.
- El ángulo de finalización de la porción del círculo
  El ángulo de finalización define el final de la porción del círculo debido a que la cantidad de grados se miden en el sentido contrario a las agujas del reloj desde 0 grados.
- El radio externo
  El radio externo define la distancia del centro de un círculo al punto extremo de la porción del círculo.
- El radio interno
  El radio interno define la distancia del centro de un círculo al punto más recóndito de la porción del círculo definiendo así el inicio de la porción del círculo. Si el radio interno es 0, la forma es una porción de círculo simple.



Si el radio interno es mayor que 0, la porción de círculo es técnicamente una banda de arco.

Copyright © 1995-2023 Esri. All rights reserved.



- Número de puntos utilizados para definir la porción de círculo
  La cantidad mínima de puntos que puede especificar es 9. Si no especifica una cantidad de puntos, se utilizarán 80 puntos por defecto. Esos puntos no se almacenarán con la forma pero se generarán cuando se genere la porción de círculo para validar la forma.
- SRID utilizado para ubicar la porción de círculo en el espacio

Todos los radios, incluidos los semiejes mayor y menor y los radios externo e interno, se definen en las unidades determinadas por la referencia de coordenadas especificada con el SRID.

Consulte la función ST_Geometry para obtener información sobre la sintaxis y ver ejemplos de la creación de un círculo, una elipse o una porción de círculo paramétricos.

Copyright © 1995-2023 Esri. All rights reserved.

ST_Aggr_ConvexHull

Nota:
Solo Oracle y SQLite

Definición

ST_Aggr_ConvexHull crea una geometría única que es una envoltura convexa de una geometría derivada de una combinación de todas las geometrías de entrada. De hecho, ST_Aggr_ConvexHull equivale a ST_ConvexHull(ST_Aggr_Union(geometries)).

Sintaxis

Oracle

```
sde.st_aggr_convexhull (geometry sde.st_geometry)
```

SQLite

```
st_aggr_convexhull (geometry st_geometry)
```

Tipo de devolución

Oracle

ST_Geometry

SQLite

Geometryblob

Ejemplo

El ejemplo crea una tabla service_territories y ejecuta una sentencia `SELECT` que agrega todas las geometrías, lo que genera una geometría única que representa la envoltura convexa de la combinación de todas las formas.

Oracle

```
CREATE TABLE service_territories
 (ID integer not null, UNITS number, SHAPE sde.st_geometry);

INSERT INTO service_territories (id, units, shape) VALUES (
 1,
 1250,
 sde.st_polygon ('polygon ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO service_territories (id, units, shape) VALUES (
 2,
 875,
 sde.st_polygon ('polygon ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
```

Copyright © 1995-2023 Esri. All rights reserved.

```
);

INSERT INTO service_territories (id, units, shape) VALUES (
 3,
 1700,
 sde.st_polygon ('polygon ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
);

SELECT sde.st_astext(sde.st_aggr_convexhull(shape)) CONVEX_HULL
 FROM service_territories
 WHERE units >= 1000;

CONVEX_HULL

POLYGON  (( 20.00000000 40.00000000, 20.00000000 30.00000000, 30.00000000 30.00000000,
60.00000000 40.00000000, 60.00000000 60.00000000, 40.00000000 60.00000000, 20.00000000
40.00000000))
```

SQLite

```
CREATE TABLE service_territories (
  ID integer primary key autoincrement not null,
  UNITS numeric
);

SELECT AddGeometryColumn(
 NULL,
 'service_territories',
 'shape',
 4326,
 'polygon',
 'xy',
 'null'
);

INSERT INTO service_territories (units, shape) VALUES (
 1250,
 st_polygon ('polygon ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO service_territories (units, shape) VALUES (
 875,
 st_polygon ('polygon ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
);

INSERT INTO service_territories (units, shape) VALUES (
 1700,
 st_polygon ('polygon ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
);

SELECT st_astext(st_aggr_convexhull(shape)) AS "CONVEX HULL"
 FROM service_territories
 WHERE units >= 1000;

CONVEX HULL

POLYGON  (( 20.00000000 40.00000000, 20.00000000 30.00000000, 30.00000000 30.00000000,
60.00000000 40.00000000, 60.00000000 60.00000000, 40.00000000 60.00000000, 20.00000000
40.00000000))
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Aggr_Intersection

Nota:
Solo Oracle y SQLite

Definición

ST_Aggr_Intersection devuelve una sola geometría que es una combinación de la intersección de todas las geometrías de entrada.

Sintaxis

Oracle

```
sde.st_aggr_intersection (geometry1 sde.st_geometry)
```

SQLite

```
st_aggr_intersection (geometry1 geometryblob)
```

Tipo de devolución

Oracle

ST_Geometry

SQLite

Geometryblob

Ejemplo

En este ejemplo, un biólogo está intentando encontrar la intersección de tres hábitats naturales.

Oracle

Primero, cree la tabla que almacena los hábitats.

```
CREATE TABLE habitats (
 id integer not null,
 shape sde.st_geometry
);
```

A continuación, inserte los tres polígonos en la tabla.

```
INSERT INTO habitats (id, shape) VALUES (
 1,
 sde.st_polygon ('polygon ((5 5, 12 5, 12 10, 5 10, 5 5))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO habitats (id, shape) VALUES (
 2,
 sde.st_polygon ('polygon ((10 8, 14 8, 14 15, 10 15, 10 8))', 4326)
);

INSERT INTO habitats (id, shape) VALUES (
 3,
 sde.st_polygon ('polygon ((6 8, 20 8, 20 20, 6 20, 6 8))', 4326)
);
```

Por último, seleccione la intersección de los hábitats.

```
SELECT sde.st_astext(sde.st_aggr_intersection(shape)) AGGR_SHAPES
 FROM habitats;

AGGR_SHAPES

POLYGON  (( 10.00000000 8.00000000, 12.00000000 8.00000000, 12.00000000 10.00000000,
10.00000000 10.00000000, 10.00000000 8.00000000))
```

SQLite

Primero, cree la tabla que almacena los hábitats.

```
CREATE TABLE habitats (
  id integer primary key autoincrement not null
);

SELECT AddGeometryColumn(
 NULL,
 'habitats',
 'shape',
 4326,
 'polygon',
 'xy',
 'null'
);
```

A continuación, inserte los tres polígonos en la tabla.

```
INSERT INTO habitats (shape) VALUES (
 st_polygon ('polygon ((5 5, 12 5, 12 10, 5 10, 5 5))', 4326)
);

INSERT INTO habitats (shape) VALUES (
 st_polygon ('polygon ((10 8, 14 8, 14 15, 10 15, 10 8))', 4326)
);

INSERT INTO habitats (shape) VALUES (
 st_polygon ('polygon ((6 8, 20 8, 20 20, 6 20, 6 8))', 4326)
);
```

Por último, seleccione la intersección de los hábitats.

```
SELECT st_astext(st_aggr_intersection(shape))
```

Copyright © 1995-2023 Esri. All rights reserved.

```
  AS "AGGR_SHAPES"
  FROM habitats;

AGGR_SHAPES

POLYGON  (( 10.00000000 8.00000000, 12.00000000 8.00000000, 12.00000000 10.00000000,
10.00000000 10.00000000, 10.00000000 8.00000000))
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Aggr_Union

Definición

ST_Aggr_Union devuelve una sola geometría que es la combinación de todas las geometrías de entrada.

Sintaxis

Oracle y PostgreSQL

```
sde.st_aggr_union(geometry sde.st_geometry)
```

SQLite

```
st_aggr_union(geometry geometryblob)
```

Tipo de devolución

Oracle y PostgreSQL

ST_Geometry

SQLite

Geometryblob

Ejemplo

Un analista de comercialización necesita crear una geometría única de todas las áreas de servicio para el que las ventas excedieron 1,000 unidades. En este ejemplo creará una tabla service_territories1 y la rellenará con números de valores de venta. A continuación usará st_aggr_union en una declaración SELECT para devolver el multipolígono que es la combinación de todas las geometrías para las cuales los números de ventas son iguales o superiores a 1000 unidades.

Oracle y PostgreSQL

```
--Create and populate tables.
CREATE TABLE service_territories1 (
 ID integer not null,
 UNITS number,
 SHAPE sde.st_geometry);
INSERT INTO service_territories1 (id, units, shape) VALUES (
 1,
 1250,
 sde.st_polygon ('polygon ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO service_territories1 (id, units, shape) VALUES (
 2,
 875,
 sde.st_polygon ('polygon ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO service_territories1 (id, units, shape) VALUES (
 3,
 1700,
 sde.st_polygon ('polygon ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
);
```

```
--Union of all geometries for which sales numbers are >= 1,000 units.
SELECT sde.st_astext(sde.st_aggr_union(shape)) UNION_SHAPE
  FROM service_territories1
  WHERE units >= 1000;
UNION_SHAPE
MULTIPOLYGON  ((( 20.00000000 30.00000000, 30.00000000 30.00000000, 30.00000000
40.00000000, 20.00000000 40.00000000, 20.00000000 30.00000000)),(( 40.00000000
40.00000000,
60.00000000 40.00000000, 60.00000000 60.00000000, 40.00000000 60.00000000,
40.00000000 40.00000000)))
```

SQLite

```
--Create table, add geometry column to it, and populate table.
CREATE TABLE service_territories1 (
 id integer primary key autoincrement not null,
 units number
);
SELECT AddGeometryColumn(
 NULL,
 'service_territories1',
 'shape',
 4326,
 'polygon',
 'xy',
 'null'
);
INSERT INTO service_territories1 (units, shape) VALUES (
 1250,
 st_polygon ('polygon ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO service_territories1 (units, shape) VALUES (
 875,
 st_polygon ('polygon ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
);

INSERT INTO service_territories1 (units, shape) VALUES (
 1700,
 st_polygon ('polygon ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
);
```

```
--Union of all geometries for which sales numbers are >= 1,000 units.
SELECT st_astext(st_aggr_union(shape))
 AS "UNION_SHAPE"
 FROM service_territories1
 WHERE units >= 1000;
UNION_SHAPE
MULTIPOLYGON ((( 40.00000000 40.00000000, 60.00000000 40.00000000, 60.00000000 6
0.00000000, 40.00000000 60.00000000, 40.00000000 40.00000000)),(( 20.00000000 30
.00000000, 30.00000000 30.00000000, 30.00000000 40.00000000, 20.00000000 40.0000
```

Copyright © 1995-2023 Esri. All rights reserved.

```
0000, 20.00000000 30.00000000)))
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Area

Definición

ST_Area devuelve el área de un polígono o multipolígono.

Sintaxis

Oracle y PostgreSQL

```
sde.st_area (polygon sde.st_geometry)
sde.st_area (multipolygon sde.st_geometry)
```

SQLite

```
st_area (polygon st_geometry)
st_area (polygon st_geometry, unit_name)
```

Tipo de devolución

Precisión doble

Ejemplo

El ingeniero de la ciudad necesita una lista de áreas de construcción. Para crear la lista, un técnico SIG selecciona el Id. de la construcción y el área de cada huella del edificio.

Las huellas de los edificios se almacenan en la tabla bfp.

Para cumplir con la solicitud del ingeniero de la ciudad, el técnico selecciona la clave única, el building_id y el área de cada huella de edificio desde la tabla bfp.

Copyright © 1995-2023 Esri. All rights reserved.

Oracle

```
--Create and populate table.
CREATE TABLE bfp (
 building_id integer not null,
 footprint sde.st_geometry);

INSERT INTO BFP (building_id, footprint) VALUES (
 1,
 sde.st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);

INSERT INTO BFP (building_id, footprint) VALUES (
 2,
 sde.st_polygon ('polygon ((20 0, 30 20, 40 0, 20 0))', 4326)
);

INSERT INTO BFP (building_id, footprint) VALUES (
 3,
 sde.st_polygon ('polygon ((20 30, 25 35, 30 30, 20 30))', 4326)
);
```

```
--Get area of geometries.
SELECT building_id, sde.st_area (footprint) Area
 FROM BFP;

BUILDING_ID       Area

          1        100
          2        200
          3         25
```

PostgreSQL

```
--Create and populate table.
CREATE TABLE bfp (
 building_id serial,
 footprint sde.st_geometry);

INSERT INTO bfp (footprint) VALUES (
 sde.st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);

INSERT INTO bfp (footprint) VALUES (
 sde.st_polygon ('polygon ((20 0, 30 20, 40 0, 20 0))', 4326)
);

INSERT INTO bfp (footprint) VALUES (
 sde.st_polygon ('polygon ((20 30, 25 35, 30 30, 20 30))', 4326)
);
```

```
--Get area of geometries.
SELECT building_id, sde.st_area (footprint)
 AS Area
 FROM bfp;

building_id       area
```

Copyright © 1995-2023 Esri. All rights reserved.

```
          1        100
          2        200
          3         25
```

SQLite

```
--Create table, add geometry column to it, and populate the table.
CREATE TABLE bfp (
 building_id integer primary key autoincrement not null
);

SELECT AddGeometryColumn(
 NULL,
 'bfp',
 'footprint',
 4326,
 'polygon',
 'xy',
 'null'
);

INSERT INTO bfp (footprint) VALUES (
 st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);

INSERT INTO bfp (footprint) VALUES (
 st_polygon ('polygon ((20 0, 30 20, 40 0, 20 0))', 4326)
);

INSERT INTO bfp (footprint) VALUES (
 st_polygon ('polygon ((20 30, 25 35, 30 30, 20 30))', 4326)
);
```

```
--Get area of geometries.
SELECT building_id, st_area (footprint)
 AS "area"
 FROM bfp;

building_id       area

          1        100.0
          2        200.0
          3         25.0
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_AsBinary

Definición

ST_AsBinary toma un objeto de geometría y devuelve su representación binaria conocida.

Sintaxis

Oracle y PostgreSQL

```
sde.st_asbinary (geometry sde.st_geometry)
```

SQLite

```
st_asbinary (geometry geometryblob)
```

Tipo de devolución

Oracle y PostgreSQL

ST_Geometry

SQLite

Geometryblob

Ejemplo

En este ejemplo se llena la columna WKB del registro 1111 con el contenido de la columna GEOMETRY del registro 1100.

Oracle

```
CREATE TABLE sample_points (
 id integer not null,
 geometry sde.st_geometry,
 wkb blob
);

INSERT INTO SAMPLE_POINTS (id, geometry) VALUES (
 1100,
 sde.st_geometry ('point (10 20)', 4326)
);

INSERT INTO SAMPLE_POINTS (id, wkb) VALUES (
 1111,
 (SELECT sde.st_asbinary (geometry) FROM sample_points WHERE id = 1100)
);

SELECT id, sde.st_astext (sde.st_geomfromwkb (wkb, 4326))
 FROM SAMPLE_POINTS
 WHERE id = 1111;

ID              Point
```

Copyright © 1995-2023 Esri. All rights reserved.

```
1111     POINT (10.00000000 20.00000000)
```

PostgreSQL

```
CREATE TABLE sample_points (
 id serial,
 geometry sde.st_geometry,
 wkb bytea);

INSERT INTO sample_points (geometry) VALUES (
 sde.st_point (10, 20, 4326)
);

INSERT INTO sample_points (wkb) VALUES (
 (SELECT sde.st_asbinary (geometry) FROM sample_points WHERE id = 1100)
);

SELECT id, sde.st_astext (sde.st_geomfromwkb (wkb, 4326))
 FROM sample_points
 WHERE id = 1111;

ID              st_astext
1111     POINT (10 20)
```

SQLite

```
CREATE TABLE sample_points (
 id integer primary key autoincrement not null,
 wkb blob
);

SELECT AddGeometryColumn(
 NULL,
 'sample_points',
 'geometry',
 4326,
 'point',
 'xy',
 'null'
);

INSERT INTO sample_points (geometry) VALUES (
 st_point (10, 20, 4326)
);

INSERT INTO sample_points (wkb) VALUES (
 (SELECT st_asbinary (geometry) FROM sample_points WHERE id = 1)
);

SELECT id, st_astext (st_geomfromwkb (wkb, 4326))
 FROM sample_points
 WHERE id = 2;

ID              st_astext
2        POINT (10.00000000 20.00000000)
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_AsText

Definición

ST_AsText toma una geometría y devuelve su representación de texto conocida.

Sintaxis

Oracle y PostgreSQL

```
sde.st_astext (geometry sde.st_geometry)
```

SQLite

```
st_astext (geometry geometryblob)
```

Tipo de devolución

Oracle

CLOB

PostgreSQL y SQLite

Texto

Ejemplo

La función ST_AsText convierte el punto de ubicación hazardous_sites en su descripción de texto.

Oracle

```
CREATE TABLE hazardous_sites (
 site_id integer not null,
 name varchar(40),
 loc sde.st_geometry);

INSERT INTO HAZARDOUS_SITES (site_id, name, loc) VALUES (
 102,
 'W. H. KleenareChemical Repository',
 sde.st_geometry ('point (1020.12 324.02)', 4326)
);

SELECT site_id, name, sde.st_astext (loc) Location
 FROM HAZARDOUS_SITES;

SITE_ID      NAME                              Location

   102    W. H. KleenareChemical Repository   POINT (1020.12000000 324.02000000)
```

Copyright © 1995-2023 Esri. All rights reserved.

PostgreSQL

```
CREATE TABLE hazardous_sites (
 site_id serial,
 name varchar(40),
 loc sde.st_geometry);

INSERT INTO hazardous_sites (name, loc) VALUES (
 'W. H. KleenareChemical Repository',
 sde.st_point ('point (1020.12 324.02)', 4326)
);

SELECT site_id, name, sde.st_astext (loc)
 AS location
 FROM hazardous_sites;

site_id      name                             location

   102    W. H. KleenareChemical Repository   POINT (1020.12000001 324.01999999)
```

SQLite

```
CREATE TABLE hazardous_sites (
 site_id integer primary key autoincrement not null,
 name varchar(40)
);

SELECT AddGeometryColumn(
 NULL,
 'hazardous_sites',
 'loc',
 4326,
 'point',
 'xy',
 'null'
);

INSERT INTO hazardous_sites (name, loc) VALUES (
 'W. H. KleenareChemical Repository',
 st_point ('point (1020.12 324.02)', 4326)
);

SELECT site_id, name, st_astext (loc)
 FROM hazardous_sites;

   1    W. H. KleenareChemical Repository   POINT (1020.12000000 324.02000000)
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Boundary

Definición

ST_Boundary toma una geometría y devuelve su límite combinado como objeto de geometría.

Sintaxis

Oracle y PostgreSQL

```
sde.st_boundary (geometry sde.st_geometry)
```

SQLite

```
st_boundary (geometry geometryblob)
```

Tipo de devolución

Oracle y PostgreSQL

ST_Geometry

SQLite

Geometryblob

Ejemplo

En este ejemplo, la tabla de límites se crea con dos columnas: tipo y geometría. Las sentencias INSERT posteriores agregan un registro para cada una de las geometrías de la subclase. La función ST_Boundary recupera el límite de cada subclase almacenada en la columna de geometría. Tenga en cuenta que la dimensión de la geometría resultante siempre es uno menos que la geometría de entrada. Los puntos y multipuntos siempre dan como resultado un límite que es una geometría vacía, dimensión –1. Las cadenas de texto de líneas y los elementos de las cadenas de texto multilínea devuelven un límite multipunto, dimensión 0. Un polígono o un multipolígono siempre devuelven un límite multilínea, dimensión 1.

Oracle

```
CREATE TABLE boundaries (
 geotype varchar(20),
 geometry sde.st_geometry
);

INSERT INTO BOUNDARIES VALUES (
 'Point',
 sde.st_pointfromtext ('point (10.02 20.01)', 4326)
);

INSERT INTO BOUNDARIES VALUES (
 'Linestring',
 sde.st_linefromtext ('linestring (10.02 20.01, 10.32 23.98, 11.92 25.64)', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO BOUNDARIES VALUES (
 'Polygon',
 sde.st_polyfromtext ('polygon ((10.02 20.01, 11.92 35.64, 25.02 34.15, 19.15 33.94,
10.02 20.01))', 4326)
);

INSERT INTO BOUNDARIES VALUES (
 'Multipoint',
 sde.st_mpointfromtext ('multipoint ((10.02 20.01), (10.32 23.98), (11.92 25.64))',
4326)
);

INSERT INTO BOUNDARIES VALUES (
 'Multilinestring',
 sde.st_mlinefromtext ('multilinestring ((10.02 20.01, 10.32 23.98, 11.92 25.64), (9.55
23.75, 15.36 30.11))', 0)
);

INSERT INTO BOUNDARIES VALUES (
 'Multipolygon',
 sde.st_mpolyfromtext ('multipolygon (((10.02 20.01, 11.92 35.64, 25.02 34.15, 19.15
33.94, 10.02 20.01), (51.71 21.73,73.36 27.04,71.52 32.87, 52.43 31.90, 51.71
21.73)))', 4326)
);

SELECT geotype, sde.st_astext (sde.st_boundary (geometry)) "The boundary"
 FROM BOUNDARIES;

GEOTYPE            The boundary

Point              POINT EMPTY
Linestring         MULTIPOINT((10.02000000 20.01000000), (11.92000000 25.64000000))
Polygon            MULTILINESTRING ((10.02000000 20.01000000, 19.15000000
33.94000000,25.02000000 34.15000000, 11.92000000 35.64000000, 10.02000000 20.01000000))
Multipoint         POINT EMPTY
Multilinestring    MULTIPOINT ((9.55000000 23.75000000), (10.02000000 20.01000000),
(11.92000000 25.64000000), (15.36000000 30.11000000))
Multipolygon       MULTILINESTRING((51.71000000 21.73000000, 73.36000000 27.04000000,
71.52000000 32.87000000, 52.43000000 31.90000000, 51.71000000 21.73000000),
(10.02000000 20.01000000, 19.15000000 33.94000000, 25.02000000 34.15000000, 11.92000000
35.64000000, 10.02000000 20.01000000))
```

PostgreSQL

```
CREATE TABLE boundaries (
 geotype varchar(20),
 geometry st_geometry
);

INSERT INTO boundaries VALUES (
 'Point',
 st_point ('point (10.02 20.01)', 4326)
);

INSERT INTO boundaries VALUES (
 'Linestring',
 st_linestring ('linestring (10.02 20.01, 10.32 23.98, 11.92 25.64)', 4326)
);

INSERT INTO boundaries VALUES (
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 'Polygon',
 st_polygon ('polygon ((10.02 20.01, 11.92 35.64, 25.02 34.15, 19.15 33.94, 10.02 
20.01))', 4326)
);

INSERT INTO boundaries VALUES (
'Multipoint',
st_multipoint ('multipoint (10.02 20.01, 10.32 23.98, 11.92 25.64)', 0)
);

INSERT INTO boundaries VALUES (
 'Multilinestring',
 st_multilinestring ('multilinestring ((10.02 20.01, 10.32 23.98, 11.92 25.64), (9.55 
23.75, 15.36 30.11))', 4326)
);

INSERT INTO boundaries VALUES (
 'Multipolygon',
 st_multipolygon ('multipolygon (((10.02 20.01, 11.92 35.64, 25.02 34.15, 19.15 33.94, 
10.02 20.01),
(51.71 21.73, 73.36 27.04, 71.52 32.87, 52.43 31.90, 51.71 21.73)))', 4326)
);

SELECT geotype, st_astext (st_boundary (geometry))
 AS "The boundary"
 FROM boundaries;

geotype            The boundary

Point              EMPTY
Linestring         MULTIPOINT(10.02000000 20.01000000, 11.92000000 25.64000000)
Polygon            LINESTRING ((10.02000000 20.01000000, 19.15000000 
33.94000000,25.02000000 34.15000000, 11.92000000 35.64000000, 10.02000000 
20.01000000))
Multipoint         EMPTY
Multilinestring    MULTIPOINT (9.55000000 23.75000000, 10.02000000 20.01000000, 
11.92000000 25.64000000, 15.36000000 30.11000000)
Multipolygon       MULTILINESTRING((51.71000000 21.73000000, 73.36000000 27.04000000, 
71.52000000 32.87000000, 52.43000000 31.90000000, 
51.71000000 21.73000000), (10.02000000 20.01000000, 19.15000000 33.94000000, 
25.02000000 34.15000000, 11.92000000 35.64000000, 
10.02000000 20.01000000))
```

SQLite

```
CREATE TABLE boundaries (
 geotype varchar(20)
);

SELECT AddGeometryColumn (
 NULL,
 'boundaries',
 'geometry',
 4326,
 'geometry',
 'xy',
 'null'
);

INSERT INTO boundaries VALUES (
 'Point',
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 st_point ('point (10.02 20.01)', 4326)
);

INSERT INTO boundaries VALUES (
 'Linestring',
 st_linestring ('linestring (10.02 20.01, 10.32 23.98, 11.92 25.64)', 4326)
);

INSERT INTO boundaries VALUES (
 'Polygon',
 st_polygon ('polygon ((10.02 20.01, 11.92 35.64, 25.02 34.15, 19.15 33.94, 10.02
20.01))', 4326)
);

INSERT INTO boundaries VALUES (
 'Multipoint',
 st_multipoint ('multipoint ((10.02 20.01), (10.32 23.98), (11.92 25.64))', 4326)
);

INSERT INTO boundaries VALUES (
 'Multilinestring',
 st_multilinestring ('multilinestring ((10.02 20.01, 10.32 23.98, 11.92 25.64), (9.55
23.75, 15.36 30.11))', 4326)
);

INSERT INTO boundaries VALUES (
 'Multipolygon',
 st_multipolygon ('multipolygon (((10.02 20.01, 11.92 35.64, 25.02 34.15, 19.15 33.94,
10.02 20.01),
(51.71 21.73, 73.36 27.04, 71.52 32.87, 52.43 31.90, 51.71 21.73)))', 4326)
);

SELECT geotype, st_astext (st_boundary (geometry))
 FROM boundaries;

Point              EMPTY
Linestring         MULTIPOINT((10.02000000 20.01000000), (11.92000000 25.64000000))
Polygon            LINESTRING ((10.02000000 20.01000000, 19.15000000
33.94000000,25.02000000 34.15000000, 11.92000000 35.64000000, 10.02000000
20.01000000))
Multipoint         EMPTY
Multilinestring    MULTIPOINT ((9.55000000 23.75000000), (10.02000000 20.01000000),
(11.92000000 25.64000000), (15.36000000 30.11000000))
Multipolygon       MULTILINESTRING((51.71000000 21.73000000, 73.36000000 27.04000000,
71.52000000 32.87000000, 52.43000000 31.90000000,
51.71000000 21.73000000), (10.02000000 20.01000000, 19.15000000 33.94000000,
25.02000000 34.15000000, 11.92000000 35.64000000,
10.02000000 20.01000000))
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Buffer

Definición

ST_Buffer toma un objeto de geometría y distancia, y devuelve un objeto de geometría que es la zona de influencia en torno a objeto de origen.

Sintaxis

`Unit_name` es la unidad de medida de la distancia de zona de influencia (por ejemplo, `metros`, `kilómetros`, `pies` o `millas`). Consulte la primera tabla de `Projected coordinate system tables.pdf`, a la que se puede acceder desde Sistemas de coordenadas, proyecciones y transformaciones.

Oracle

```
sde.st_buffer (geometry sde.st_geometry, distance double_precision)
sde.st_buffer (geometry sde.st_geometry, distance double, varchar2 unit_name)
```

PostgreSQL

```
sde.st_buffer (geometry sde.st_geometry, distance double_precision)
sde.st_buffer (geometry sde.st_geometry, distance double, text unit_name)
```

SQLite

```
st_buffer (geometry geometryblob, distance double_precision)
st_buffer (geometry geometryblob, distance double, text unit_name)
```

Tipo de devolución

Oracle y PostgreSQL

ST_Geometry

SQLite

Geometryblob

Ejemplo

En este ejemplo se crean dos tablas, sensitive_areas y hazardous_sites; rellena las tablas; utiliza ST_Buffer para generar una zona de influencia alrededor de los polígonos en la tabla hazardous_sites; y se encuentra donde las zonas de influencia superpongan los polígonos sensitive_areas.

Oracle

```
CREATE TABLE sensitive_areas (
 id integer,
 zone sde.st_geometry
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
CREATE TABLE hazardous_sites (
 site_id integer,
 name varchar(40),
 location sde.st_geometry
);

INSERT INTO SENSITIVE_AREAS VALUES (
 1,
 sde.st_polygon ('polygon ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO SENSITIVE_AREAS VALUES (
 2,
 sde.st_polygon ('polygon ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
);

INSERT INTO SENSITIVE_AREAS VALUES (
3,
sde.st_polygon ('polygon ((40 40, 40 60, 60 60, 60 40, 40 40))', 0)
);

INSERT INTO HAZARDOUS_SITES VALUES (
 102,
 'W. H. KleenareChemical Repository',
 sde.st_pointfromtext ('point (60 60)', 4326)
);

SELECT sa.id "Sensitive Areas", hs.name "Hazardous Sites"
 FROM SENSITIVE_AREAS sa, HAZARDOUS_SITES hs
 WHERE sde.st_overlaps (sa.zone, sde.st_buffer (hs.location, .01)) = 1;
```

PostgreSQL

```
CREATE TABLE sensitive_areas (
 id serial,
 zone sde.st_geometry
);

CREATE TABLE hazardous_sites (
 site_id serial,
 name varchar(40),
 location sde.st_geometry
);

INSERT INTO sensitive_areas (zone) VALUES (
 sde.st_polygon ('polygon ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO sensitive_areas (zone) VALUES (
 sde.st_polygon ('polygon ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
);

INSERT INTO sensitive_areas (zone) VALUES (
 sde.st_polygon ('polygon ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
);

INSERT INTO hazardous_sites (name, location) VALUES (
 'W. H. KleenareChemical Repository',
 sde.st_point ('point (60 60)', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
SELECT sa.id AS "Sensitive Areas", hs.name AS "Hazardous Sites"
 FROM sensitive_areas sa, hazardous_sites hs
 WHERE sde.st_overlaps (sa.zone, sde.st_buffer (hs.location, .01)) = 't';

Sensitive Areas                Hazardous Sites
             3               W.H. KleenareChemical Repository
```

SQLite

```
CREATE TABLE sensitive_areas (
 id integer primary key autoincrement not null
);

SELECT AddGeometryColumn (
 NULL,
 'sensitive_areas',
 'zone',
 4326,
 'polygon',
 'xy',
 'null'
);

CREATE TABLE hazardous_sites (
 site_id integer primary key autoincrement not null,
 name varchar(40)
);

SELECT AddGeometryColumn (
 NULL,
 'hazardous_sites',
 'location',
 4326,
 'point',
 'xy',
 'null'
);

INSERT INTO sensitive_areas (zone) VALUES (
 st_polygon ('polygon ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO sensitive_areas (zone) VALUES (
 st_polygon ('polygon ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
);

INSERT INTO sensitive_areas (zone) VALUES (
 st_polygon ('polygon ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
);

INSERT INTO hazardous_sites (name, location) VALUES (
 'W. H. KleenareChemical Repository',
 st_point ('point (60 60)', 4326)
);

SELECT sa.id AS "Sensitive Areas", hs.name AS "Hazardous Sites"
 FROM sensitive_areas sa, hazardous_sites hs
 WHERE st_overlaps (sa.zone, st_buffer (hs.location, .01)) = 1;

Sensitive Areas                Hazardous Sites
```

Copyright © 1995-2023 Esri. All rights reserved.

```
            3                W.H. KleenareChemical Repository
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Centroid

Definición

ST_Centroid toma un polígono, un multipolígono o una cadena de texto multilínea y devuelve el punto que está en el centro del sobre de geometría. Esto significa que el punto centroide está en medio entre las extensiones x e y mínima y máxima de la geometría.

Sintaxis

Oracle y PostgreSQL

```
sde.st_centroid (polygon sde.st_geometry)
sde.st_centroid (multipolygon sde.st_geometry)
sde.st_centroid (multilinestring sde.st_geometry)
```

SQLite

```
st_centroid (polygon geometryblob)
st_centroid (multipolygon geometryblob)
st_centroid (multilinestring geometryblob)
```

Tipo de devolución

ST_Point

Ejemplo

El técnico SIG de la ciudad desea visualizar los multipolígonos de la huella de edificio como puntos únicos en un gráfico de densidad del edificio. Las huellas de edificios se almacenan en la tabla bfp creada y completada con las declaraciones que se muestran para cada base de datos

Oracle 11g

```
--Create and populate table
CREATE TABLE bfp (
 building_id integer,
 footprint sde.st_geometry);
INSERT INTO bfp VALUES (
 1,
 sde.st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);
INSERT INTO bfp VALUES (
 2,
 sde.st_polygon ('polygon ((20 0, 30 20, 40 0, 20 0))', 4326)
);
INSERT INTO bfp VALUES (
 3,
 sde.st_polygon ('polygon ((20 30, 25 35, 30 30, 20 30))', 4326)
);
```

```
--The ST_Centroid function returns the centroid of each building footprint
```

Copyright © 1995-2023 Esri. All rights reserved.

```
multipolygon.
--The ST_AsText function converts each centroid point into a text representation
recognized by the application.
SELECT building_id,
 sde.st_astext (sde.st_centroid (footprint)) Centroid
 FROM bfp;
BUILDING_ID                   Centroid
   1                     POINT  (5.00000000 5.00000000)
   2                     POINT  (30.00000000 10.00000000)
   3                     POINT  (25.00000000 32.50000000)
```

PostgreSQL

```
--Create and populate table
CREATE TABLE bfp (
 building_id serial,
 footprint sde.st_geometry);
INSERT INTO bfp (footprint) VALUES (
 sde.st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);
INSERT INTO bfp (footprint) VALUES (
 sde.st_polygon ('polygon ((20 0, 30 20, 40 0, 20 0))', 4326)
);
INSERT INTO bfp (footprint) VALUES (
 sde.st_polygon ('polygon ((20 30, 25 35, 30 30, 20 30))', 4326)
);
```

```
--The ST_Centroid function returns the centroid of each building footprint
multipolygon.
--The ST_AsText function converts each centroid point into a text representation
recognized by the application.
SELECT building_id, sde.st_astext (sde.st_centroid (footprint))
 AS centroid
 FROM bfp;
building_id                  centroid
   1                     POINT  (5 5)
   2                     POINT  (30 10)
   3                     POINT  (25 33)
```

SQLite

```
--Create table, add geometry column, and populate table
CREATE TABLE bfp (
 building_id integer primary key autoincrement not null
);
SELECT AddGeometryColumn (
 NULL,
 'bfp',
 'footprint',
 4326,
 'polygon',
 'xy',
 'null'
);
INSERT INTO bfp (footprint) VALUES (
 st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO bfp (footprint) VALUES (
 st_polygon ('polygon ((20 0, 30 20, 40 0, 20 0))', 4326)
);
INSERT INTO bfp (footprint) VALUES (
 st_polygon ('polygon ((20 30, 25 35, 30 30, 20 30))', 4326)
);
```

```
--The ST_Centroid function returns the centroid of each building footprint
multipolygon.
--The ST_AsText function converts each centroid point into a text representation
recognized by the application.
SELECT building_id, st_astext (st_centroid (footprint))
 AS "centroid"
 FROM bfp;
building_id              centroid
   1                     POINT  (5.00000000 5.00000000)
   2                     POINT  (30.00000000 10.0000000)
   3                     POINT  (25.00000000 32.5000000)
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Contains

Definición

ST_Contains toma dos objetos de geometría y devuelve 1 (Oracle y SQLite) o t (PostgreSQL) si el primer objeto contiene completamente al segundo; de lo contrario, devuelve 0 (Oracle y SQLite) o f (PostgreSQL).

Sintaxis

Oracle y PostgreSQL

```
sde.st_contains (geometry1 sde.st_geometry, geometry2 sde.st_geometry)
```

SQLite

```
st_contains (geometry1 geometryblob, geometry2 geometryblob)
```

Tipo de devolución

Booleano

Ejemplo

En los ejemplos siguientes, se crean dos tablas. Una, buildingfootprints, contiene las huellas de edificios de la ciudad, mientras que la otra, lots, contiene las parcelas. El ingeniero de la ciudad desea garantizar que todas las huellas de los edificios están completamente inscritas en sus parcelas.

El ingeniero de la ciudad usa ST_Intersects y ST_Contains para seleccionar los edificios que no están completamente inscritos en una parcela.

Oracle

```
--Create tables and insert values.
CREATE TABLE bfp (
 building_id integer,
 footprint sde.st_geometry
);

CREATE TABLE lots (
 lot_id integer,
 lot sde.st_geometry
);

INSERT INTO BFP (building_id, footprint) VALUES (
 1,
 sde.st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);

INSERT INTO BFP (building_id, footprint) VALUES (
 2,
 sde.st_polygon ('polygon ((20 0, 20 10, 30 10, 30 0, 20 0))', 4326)
);

INSERT INTO BFP (building_id, footprint) VALUES (
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 3,
 sde.st_polygon ('polygon ((40 0, 40 10, 50 10, 50 0, 40 0))', 4326)
);

INSERT INTO LOTS (lot_id, lot) VALUES (
 1,
 sde.st_polygon ('polygon ((-1 -1, -1 11, 11 11, 11 -1, -1 -1))', 4326)
);

INSERT INTO LOTS (lot_id, lot) VALUES (
 2,
 sde.st_polygon ('polygon ((19 -1, 19 11, 29 9, 31 -1, 19 -1))', 4326)
);

INSERT INTO LOTS (lot_id, lot) VALUES (
 3,
 sde.st_polygon ('polygon ((39 -1, 39 11, 51 11, 51 -1, 39 -1))', 4326)
);
```

```
--Select the buildings that are not completely contained within one lot.
SELECT UNIQUE (building_id)
 FROM BFP, LOTS
 WHERE sde.st_intersects (lot, footprint) = 1
 AND sde.st_contains (lot, footprint) = 0;

BUILDING_ID

          2
```

PostgreSQL

```
--Create tables and insert values.
CREATE TABLE bfp (
 building_id serial,
 footprint st_geometry);

CREATE TABLE lots
 (lot_id serial,
 lot st_geometry);

INSERT INTO bfp (footprint) VALUES (
 st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);

INSERT INTO bfp (footprint) VALUES (
 st_polygon ('polygon ((20 0, 20 10, 30 10, 30 0, 20 0))', 4326)
);

INSERT INTO bfp (footprint) VALUES (
 st_polygon ('polygon ((40 0, 40 10, 50 10, 50 0, 40 0))', 4326)
);

INSERT INTO lots (lot) VALUES (
 st_polygon ('polygon ((-1 -1, -1 11, 11 11, 11 -1, -1 -1))', 4326)
);

INSERT INTO lots (lot) VALUES (
 st_polygon ('polygon ((19 -1, 19 11, 29 9, 31 -1, 19 -1))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO lots (lot) VALUES (
 st_polygon ('polygon ((39 -1, 39 11, 51 11, 51 -1, 39 -1))', 4326)
);
```

```
--Select the buildings that are not completely contained within one lot.
SELECT DISTINCT (building_id)
 FROM bfp, lots
 WHERE st_intersects (lot, footprint) = 't'
 AND st_contains (lot, footprint) = 'f';

building_id

          2
```

SQLite

```
--Create tables, add geometry columns, and insert values.
CREATE TABLE bfp (
 building_id integer primary key autoincrement not null
);

SELECT AddGeometryColumn (
 NULL,
 'bfp',
 'footprint',
 4326,
 'polygon',
 'xy',
 'null'
);

CREATE TABLE lots
 (lot_id integer primary key autoincrement not null
);

SELECT AddGeometryColumn (
 NULL,
 'lots',
 'lot',
 4326,
 'polygon',
 'xy',
 'null'
);

INSERT INTO bfp (footprint) VALUES (
 st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);

INSERT INTO bfp (footprint) VALUES (
 st_polygon ('polygon ((20 0, 20 10, 30 10, 30 0, 20 0))', 4326)
);

INSERT INTO bfp (footprint) VALUES (
 st_polygon ('polygon ((40 0, 40 10, 50 10, 50 0, 40 0))', 4326)
);

INSERT INTO lots (lot) VALUES (
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 st_polygon ('polygon ((-1 -1, -1 11, 11 11, 11 -1, -1 -1))', 4326)
);

INSERT INTO lots (lot) VALUES (
 st_polygon ('polygon ((19 -1, 19 11, 29 9, 31 -1, 19 -1))', 4326)
);

INSERT INTO lots (lot) VALUES (
 st_polygon ('polygon ((39 -1, 39 11, 51 11, 51 -1, 39 -1))', 4326)
);
```

```
--Select the buildings that are not completely contained within one lot.
SELECT DISTINCT (building_id)
 FROM bfp, lots
 WHERE st_intersects (lot, footprint) = 1
 AND st_contains (lot, footprint) = 0;

building_id

2
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_ConvexHull

Definición

ST_ConvexHull devuelve la envoltura convexa de un objeto ST_Geometry.

Sintaxis

Oracle y PostgreSQL

```
sde.st_convexhull (geometry1 sde.st_geometry)
```

SQLite

```
st_convexhull (geometry1 geometryblob)
```

Tipo de devolución

Oracle y PostgreSQL

ST_Geometry

SQLite

Geometryblob

Ejemplo

Estos ejemplos crean la tabla sample_geometries con tres columnas: id, spatial_type y geometry. El campo spatial_type almacena el tipo de geometría que se crea en la columna geometry. Se insertan tres entidades en la tabla: una cadena de líneas, un polígono y un multipunto.

La sentencia `SELECT` que incluye la función ST_ConvexHull devuelve la envoltura convexa de cada geometría.

Oracle

```
--Create table and insert three sample geometries.
CREATE TABLE sample_geometries (
 id integer,
 spatial_type varchar(18),
 geometry sde.st_geometry
);

INSERT INTO sample_geometries (id, spatial_type, geometry) VALUES (
 1,
 'ST_LineString',
 sde.st_geometry ('linestring (20 20, 30 30, 20 40, 30 50)', 4326)
);

INSERT INTO sample_geometries (id, spatial_type, geometry) VALUES (
 2,
 'ST_Polygon',
 sde.st_geometry ('polygon ((30 30, 25 35, 15 50, 35 80, 40 85, 80 90, 70 75, 65 70, 55
50, 75 40, 60 30, 30 30))', 4326)
```

Copyright © 1995-2023 Esri. All rights reserved.

```
);

INSERT INTO sample_geometries (id, spatial_type, geometry) VALUES (
 3,
 'ST_MultiPoint',
 sde.st_geometry ('multipoint ((20 20), (30 30), (20 40), (30 50))', 4326)
);
```

```
--Find the convex hull of each geometry subtype.
SELECT id, spatial_type, sde.st_astext (sde.st_convexhull (geometry)) CONVEXHULL
 FROM  SAMPLE_GEOMETRIES;

ID    SPATIAL_TYPE       CONVEXHULL

 1    ST_LineString      POLYGON ((20.00000000 40.00000000,
                           20.00000000 20.00000000, 30.00000000
                           30.00000000, 30.00000000 50.00000000,
                           20.00000000 40.00000000))

 2    ST_Polygon         POLYGON ((15.00000000 50.00000000,
                           25.00000000 35.00000000, 30.00000000
                           30.00000000, 60.00000000 30.00000000,
                           75.00000000 40.00000000, 80.00000000
                           90.00000000, 40.00000000 85.00000000,
                           35.00000000 80.00000000, 15.00000000
                           50.00000000))

 3    ST_MultiPoint      POLYGON ((20.00000000 40.00000000,
                           20.00000000 20.00000000, 30.00000000
                           30.00000000, 30.00000000 50.00000000,
                           20.00000000 40.00000000))
```

Copyright © 1995-2023 Esri. All rights reserved.

PostgreSQL

```
--Create table and insert three sample geometries.
CREATE TABLE sample_geometries (
 id integer,
 spatial_type varchar(18),
 geometry sde.st_geometry
);

INSERT INTO sample_geometries (id, spatial_type, geometry) VALUES (
 1,
 'ST_LineString',
 sde.st_geometry ('linestring (20 20, 30 30, 20 40, 30 50)', 4326)
);

INSERT INTO sample_geometries (id, spatial_type, geometry) VALUES (
 2,
 'ST_Polygon',
 sde.st_geometry ('polygon ((30 30, 25 35, 15 50, 35 80, 40 85, 80 90, 70 75, 65 70, 55
50, 75 40, 60 30, 30 30))', 4326)
);

INSERT INTO sample_geometries (id, spatial_type, geometry) VALUES (
 3,
 'ST_MultiPoint',
 sde.st_geometry ('multipoint (20 20, 30 30, 20 40, 30 50)', 4326)
);
```

```
--Find the convex hull of each geometry subtype.
SELECT id, spatial_type, st_astext (sde.st_convexhull (geometry))
 AS CONVEXHULL
 FROM sample_geometries;

id    spatial_type       convexhull

 1    ST_LineString      POLYGON (( 20 40, 20 20, 30 30, 30 50,
                           20 40))

 2    ST_Polygon         POLYGON (( 15 50, 25 35, 30 30, 60 30,
                           75 40, 80 90, 40 85, 35 80, 15 50))

 3    ST_MultiPoint      POLYGON (( 20 40, 20 20, 30 30, 30 50,
                           20 40))
```

SQLite

```
--Create table and insert three sample geometries.
CREATE TABLE sample_geometries (
 id integer primary key autoincrement not null,
 spatial_type varchar(18)
);

SELECT AddGeometryColumn(
 NULL,
 'sample_geometries',
 'geometry',
 4326,
 'geometry',
 'xy',
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 'null'
);

INSERT INTO sample_geometries (spatial_type, geometry) VALUES (
 'ST_LineString',
 st_geometry ('linestring (20 20, 30 30, 20 40, 30 50)', 4326)
);

INSERT INTO sample_geometries (spatial_type, geometry) VALUES (
 'ST_Polygon',
 st_geometry ('polygon ((30 30, 25 35, 15 50, 35 80, 40 85, 80 90, 70 75, 65 70, 55 50,
75 40, 60 30, 30 30))', 4326)
);

INSERT INTO sample_geometries (spatial_type, geometry) VALUES (
 'ST_MultiPoint',
 st_geometry ('multipoint ((20 20), (30 30), (20 40), (30 50))', 4326)
);
```

```
--Find the convex hull of each geometry subtype.
SELECT id, spatial_type, st_astext (st_convexhull (geometry))
 AS CONVEXHULL
 FROM sample_geometries;

id    spatial_type       CONVEXHULL

 1    ST_LineString      POLYGON ((20.00000000 40.00000000,
                           20.00000000 20.00000000, 30.00000000
                           30.00000000, 30.00000000 50.00000000,
                           20.00000000 40.00000000))

 2    ST_Polygon         POLYGON ((15.00000000 50.00000000,
                           25.00000000 35.00000000, 30.00000000
                           30.00000000, 60.00000000 30.00000000,
                           75.00000000 40.00000000, 80.00000000
                           90.00000000, 40.00000000 85.00000000,
                           35.00000000 80.00000000, 15.00000000
                           50.00000000))

 3    ST_MultiPoint      POLYGON ((20.00000000 40.00000000,
                           20.00000000 20.00000000, 30.00000000
                           30.00000000, 30.00000000 50.00000000,
                           20.00000000 40.00000000))
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_CoordDim

Definición

ST_CoordDim devuelve las dimensiones de los valores de coordenadas para una columna de geometría.

Sintaxis

Oracle y PostgreSQL

```
sde.st_coorddim (geometry1 sde.st_geometry)
```

SQLite

```
st_coorddim (geometry1 geometryblob)
```

Tipo de devolución

Entero

2 = coordenadas x, y

3 = coordenadas x, y, z o x, y, m

4 = coordenadas x, y, z, m

Ejemplo

En estos ejemplos, la tabla coorddim_test se crea con las columnas geotype y g1. La columna geotype almacena el nombre de la subclase de geometría y la dimensión almacenada en la columna de geometría g1.

La declaración SELECT enumera el nombre de subclase almacenado en la columna geotype con la dimensión de las coordenadas de la geometría.

Oracle

```
--Create test table.
CREATE TABLE coorddim_test (
 geotype varchar(20),
 g1 sde.st_geometry
);
```

```
--Insert values to the test table.
INSERT INTO COORDDIM_TEST VALUES (
 'Point',
 sde.st_geometry ('point (60.567222 -140.404)', 4326)
);

INSERT INTO COORDDIM_TEST VALUES (
 'Point Z',
 sde.st_geometry ('point Z (60.567222 -140.404 5959)', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO COORDDIM_TEST VALUES (
 'Point M',
 sde.st_geometry ('point M (60.567222 -140.404 5250)', 4326)
);

INSERT INTO COORDDIM_TEST VALUES (
 'Point ZM',
 sde.st_geometry ('point ZM (60.567222 -140.404 5959 5250)', 4326)
);
```

```
--Determine the dimensionality of each feature.
SELECT geotype, sde.st_coorddim (g1) coordinate_dimension
 FROM COORDDIM_TEST;

GEOTYPE              coordinate_dimension

Point                          2
Point Z                        3
Point M                        3
Point ZM                       4
```

PostgreSQL

```
--Create test table.
CREATE TABLE coorddim_test (
 geotype varchar(20),
 g1 sde.st_geometry
);
```

```
--Insert values to the test table.
INSERT INTO coorddim_test VALUES (
 'Point',
 st_point ('point (60.567222 -140.404)', 4326)
);

INSERT INTO coorddim_test VALUES (
 'Point Z',
 st_point ('point z (60.567222 -140.404 5959)', 4326)
);

INSERT INTO coorddim_test VALUES (
 'Point M',
 st_point ('point m (60.567222 -140.404 5250)', 4326)
);

INSERT INTO coorddim_test VALUES (
 'Point ZM',
 st_point ('point zm (60.567222 -140.404 5959 5250)', 4326)
);
```

```
--Determine the dimensionality of each feature.
SELECT geotype, st_coorddim (g1)
 AS coordinate_dimension
 FROM coorddim_test;
```

Copyright © 1995-2023 Esri. All rights reserved.

```
geotype              coordinate_dimension

Point                          2
Point Z                        3
Point M                        3
Point ZM                       4
```

SQLite

```
--Create test tables and add geometry columns.
CREATE TABLE coorddim_test (
 geotype varchar(20)
);

SELECT AddGeometryColumn(
 NULL,
 'coorddim_test',
 'g1',
 4326,
 'pointzm',
 'xyzm',
 'null'
);

CREATE TABLE coorddim_test2 (
 geotype varchar(20)
);

SELECT AddGeometryColumn(
 NULL,
 'coorddim_test2',
 'g1',
 4326,
 'pointz',
 'xyz',
 'null'
);

CREATE TABLE coorddim_test3 (
 geotype varchar(20)
);

SELECT AddGeometryColumn(
 NULL,
 'coorddim_test3',
 'g1',
 4326,
 'pointm',
 'xym',
 'null'
);

CREATE TABLE coorddim_test4 (
 geotype varchar(20)
);

SELECT AddGeometryColumn(
 NULL,
 'coorddim_test4',
 'g1',
 4326,
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 'point',
 'xy',
 'null'
);
```

```
--Insert values to the test table.
INSERT INTO coorddim_test4 VALUES (
 'Point',
 st_point ('point (60.567222 -140.404)', 4326)
);

INSERT INTO coorddim_test2 VALUES (
 'Point Z',
 st_point ('point z (60.567222 -140.404 5959)', 4326)
);

INSERT INTO coorddim_test3 VALUES (
 'Point M',
 st_point ('point m (60.567222 -140.404 5250)', 4326)
);

INSERT INTO coorddim_test VALUES (
 'Point ZM',
 st_point ('point zm (60.567222 -140.404 5959 5250)', 4326)
);
```

```
--Determine the dimensionality of features in each table.
SELECT geotype, st_coorddim (g1)
 AS coordinate_dimension
 FROM coorddim_test;

geotype              coordinate_dimension

Point ZM                       4


SELECT geotype, st_coorddim (g1)
 AS coordinate_dimension
 FROM coorddim_test2;

geotype              coordinate_dimension

Point Z                        3


SELECT geotype, st_coorddim (g1)
 AS coordinate_dimension
 FROM coorddim_test3;

geotype              coordinate_dimension

Point M                        3


SELECT geotype, st_coorddim (g1)
 AS coordinate_dimension
 FROM coorddim_test4;

geotype              coordinate_dimension
```

Copyright © 1995-2023 Esri. All rights reserved.

```
Point                        2
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Crosses

Definición

ST_Crosses toma dos objetos de ST_Geometry y devuelve 1 (Oracle y SQLite) o t (PostgreSQL) si su intersección genera un objeto de geometría cuya dimensión es un número menor que la dimensión máxima de los objetos de origen. El objeto de intersección debe contener puntos interiores en ambas geometrías de origen y no están a la altura de cualquiera de los objetos de origen. De lo contrario, devuelve 0 (Oracle y SQLite) o f (PostgreSQL).

Sintaxis

```
sde.st_crosses (geometry1 sde.st_geometry, geometry2 sde.st_geometry)
```

Oracle y PostgreSQL

```
sde.st_crosses (geometry1 sde.st_geometry, geometry2 sde.st_geometry)
```

SQLite

```
st_crosses (geometry1 geometryblob, geometry2 geometryblob)
```

Tipo de devolución

Booleano

Ejemplo

El gobierno del condado está considerando un nuevo reglamento que establece que todas las instalaciones de almacenamiento de residuos peligrosos en el condado pueden no estar en un radio específico de alguna vía navegable. El administrador de SIG de condado tiene una representación precisa de los ríos y arroyos almacenados como cadenas de texto de líneas en la tabla de vías navegables, pero solo tiene una ubicación de punto único para cada una de las instalaciones de almacenamiento de residuos peligrosos.

Para determinar si debe alertar al supervisor de condado de cualquier instalación existente que violaría el reglamento propuesto, el administrador de SIG tendrá que crear zonas de influencia para las ubicaciones hazardous_sites con el fin de ver si algún río o arroyo cruza los polígonos de zona de influencia. El predicado de cruzada compara los puntos hazardous_sites en zona de influencia con las vías navegables, devolviendo solo los registros en los que las vías navegables cruzan sobre el radio regulado propuesto del país.

Oracle

```
--Define tables and insert values.
CREATE TABLE waterways (
 id integer,
 name varchar(128),
 water sde.st_geometry
);

CREATE TABLE hazardous_sites (
 site_id integer,
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 name varchar(40),
 location sde.st_geometry
);

INSERT INTO waterways VALUES (
 2,
 'Zanja',
 sde.st_geometry ('linestring (40 50, 50 40)', 4326)
);

INSERT INTO waterways VALUES (
 3,
 'Keshequa',
 sde.st_geometry ('linestring (20 20, 60 60)', 4326)
);

INSERT INTO hazardous_sites VALUES (
 4,
 'StorIt',
 sde.st_point ('point (60 60)', 4326)
);

INSERT INTO hazardous_sites VALUES (
 5,
 'Glowing Pools',
 sde.st_point ('point (30 30)', 4326)
);
```

```
--Buffer hazardous waste sites and find if any buffers cross a waterway.
SELECT UNIQUE (ww.name) "River or stream", hs.name "Hazardous sites"
 FROM WATERWAYS ww, HAZARDOUS_SITES hs
 WHERE sde.st_crosses (sde.st_buffer (hs.location, .01), ww.water) = 1;

River or stream                              Hazardous sites

Keshequa                                                    StorIt
Keshequa           Glowing Pools
```

PostgreSQL

```
--Define tables and insert values.
CREATE TABLE waterways (
 id serial,
 name varchar(128),
 water sde.st_geometry
);

CREATE TABLE hazardous_sites (
 site_id integer,
 name varchar(40),
 location sde.st_geometry
);

INSERT INTO waterways (name, water) VALUES (
 'Zanja',
 sde.st_geometry ('linestring (40 50, 50 40)', 4326)
);

INSERT INTO waterways (name, water) VALUES (
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 'Keshequa',
 sde.st_geometry ('linestring (20 20, 60 60)', 4326)
);

INSERT INTO hazardous_sites (name, location) VALUES (
 'StorIt',
 sde.st_point ('point (60 60)', 4326)
);

INSERT INTO hazardous_sites (name, location) VALUES (
 'Glowing Pools',
 sde.st_point ('point (30 30)', 4326)
);
```

```
--Buffer hazardous waste sites and find if any buffers cross a waterway.
SELECT DISTINCT (ww.name) AS "River or stream", hs.name AS "Hazardous sites"
 FROM waterways ww, hazardous_sites hs
 WHERE sde.st_crosses (sde.st_buffer (hs.location, .01), ww.water) = 't';

River or stream                            Hazardous sites

Keshequa                                                  StorIt
Keshequa             Glowing Pools
```

SQLite

```
--Define tables and insert values.
CREATE TABLE waterways (
 id integer primary key autoincrement not null,
 name varchar(128)
);

SELECT AddGeometryColumn(
 NULL,
 'waterways',
 'water',
 4326,
 'linestring',
 'xy',
 'null'
);

CREATE TABLE hazardous_sites (
 site_id integer primary key autoincrement not null,
 name varchar(40)
);

SELECT AddGeometryColumn(
 NULL,
 'hazardous_sites',
 'location',
 4326,
 'point',
 'xy',
 'null'
);

INSERT INTO waterways (name, water) VALUES (
 'Zanja',
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 st_geometry ('linestring (40 50, 50 40)', 4326)
);

INSERT INTO waterways (name, water) VALUES (
 'Keshequa',
 st_geometry ('linestring (20 20, 60 60)', 4326)
);

INSERT INTO hazardous_sites (name, location) VALUES (
 'StorIt',
 st_point ('point (60 60)', 4326)
);

INSERT INTO hazardous_sites (name, location) VALUES (
 'Glowing Pools',
 st_point ('point (30 30)', 4326)
);
```

```
--Buffer hazardous waste sites and find if any buffers cross a waterway.
SELECT DISTINCT (ww.name) AS "River or stream", hs.name AS "Hazardous sites"
 FROM waterways ww, hazardous_sites hs
 WHERE st_crosses (st_buffer (hs.location, .01), ww.water) = 1;

River or stream                           Hazardous sites

Keshequa                                                StorIt
Keshequa             Glowing Pools
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Curve

Nota:
Solo Oracle y SQLite

Definición

ST_Curve construye una entidad de curva a partir de una representación de texto conocido.

Sintaxis

Oracle

```
sde.st_curve (wkt clob, srid integer)
```

SQLite

```
st_curve (wkt text, srid int32)
```

Tipo de devolución

ST_LineString

Ejemplo

Este ejemplo crea una tabla con una geometría de curva, inserta valores en ella y selecciona una entidad de la tabla.

Oracle

```
CREATE TABLE curve_test (
 id integer,
 geometry sde.st_curve
);

INSERT INTO CURVE_TEST VALUES (
 1910,
 sde.st_curve ('linestring (33 2, 34 3, 35 6)', 4326)
);

SELECT id, sde.st_astext (geometry) CURVE
 FROM CURVE_TEST;

ID     CURVE

1110   LINESTRING (33.00000000 2.00000000, 34.00000000 3.00000000,
              35.00000000 6.00000000)
```

SQLite

```
CREATE TABLE curve_test (
 id integer primary key autoincrement not null
```

Copyright © 1995-2023 Esri. All rights reserved.

```
);

SELECT AddGeometryColumn(
 NULL,
 'curve_test',
 'geometry',
 4326,
 'linestring',
 'xy',
 'null'
);

INSERT INTO CURVE_TEST (geometry) VALUES (
 st_curve ('linestring (33 2, 34 3, 35 6)', 4326)
);

SELECT id, st_astext (geometry)
 AS curve
 FROM curve_test;

id     curve

1   LINESTRING (33.00000000 2.00000000, 34.00000000 3.00000000,
       35.00000000 6.00000000)
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Difference

Definición

ST_Difference toma dos objetos de geometría y devuelve un objeto de geometría que es la diferencia de los objetos de origen.

Sintaxis

Oracle y PostgreSQL

```
sde.st_difference (geometry1 sde.st_geometry, geometry2 sde.st_geometry)
```

SQLite

```
st_difference (geometry1 geometryblob, geometry2 geometryblob)
```

Tipo de devolución

Oracle y PostgreSQL

ST_Geometry

SQLite

Geometryblob

Ejemplo

En los ejemplos siguientes, el ingeniero de la ciudad necesita conocer el área total de las parcelas de la ciudad que no está cubierta por edificios; por tanto, desea obtener la suma del área de parcelas una vez eliminada el área de los edificios.

El ingeniero de la ciudad así mismo une la tabla de huellas y lotes en el lot_id y toma la suma del área de la diferencia de los lotes menos las huellas.

Oracle

```
--Create tables and insert values
CREATE TABLE footprints (
 building_id integer,
 footprint sde.st_geometry
);

CREATE TABLE lots (
 lot_id integer,
 lot sde.st_geometry
);

INSERT INTO footprints (building_id, footprint) VALUES (
 1,
 sde.st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO footprints (building_id, footprint) VALUES (
 2,
 sde.st_polygon ('polygon ((20 0, 20 10, 30 10, 30 0, 20 0))', 4326)
);

INSERT INTO footprints (building_id, footprint) VALUES (
 3,
 sde.st_polygon ('polygon ((40 0, 40 10, 50 10, 50 0, 40 0))', 4326)
);

INSERT INTO lots (lot_id, lot) VALUES (
 1,
 sde.st_polygon ('polygon ((-1 -1, -1 11, 11 11, 11 -1, -1 -1))', 4326)
);

INSERT INTO lots (lot_id, lot) VALUES (
 2,
 sde.st_polygon ('polygon ((19 -1, 19 11, 29 9, 31 -1, 19 -1))', 4326)
);

INSERT INTO lots (lot_id, lot) VALUES (
 3,
 sde.st_polygon ('polygon ((39 -1, 39 11, 51 11, 51 -1, 39 -1))', 4326)
);
```

```
SELECT SUM (sde.st_area (sde.st_difference (lot, footprint)))
  FROM FOOTPRINTS bf, LOTS
  WHERE bf.building_id = lots.lot_id;

SUM(ST_AREA(ST_DIFFERENCE(LOT,FOOTPRINT)))

114
```

PostgreSQL

```
--Create tables and insert values
CREATE TABLE footprints (
 building_id integer,
 footprint sde.st_geometry
);

CREATE TABLE lots (
 lot_id integer,
 lot sde.st_geometry
);

INSERT INTO footprints (building_id, footprint) VALUES (
 1,
 sde.st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);

INSERT INTO footprints (building_id, footprint) VALUES (
 2,
 sde.st_polygon ('polygon ((20 0, 20 10, 30 10, 30 0, 20 0))', 4326)
);

INSERT INTO footprints (building_id, footprint) VALUES (
 3,
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 sde.st_polygon ('polygon ((40 0, 40 10, 50 10, 50 0, 40 0))', 4326)
);

INSERT INTO lots (lot_id, lot) VALUES (
 1,
 sde.st_polygon ('polygon ((-1 -1, -1 11, 11 11, 11 -1, -1 -1))', 4326)
);

INSERT INTO lots (lot_id, lot) VALUES (
 2,
 sde.st_polygon ('polygon ((19 -1, 19 11, 29 9, 31 -1, 19 -1))', 4326)
);

INSERT INTO lots (lot_id, lot) VALUES (
 3,
 sde.st_polygon ('polygon ((39 -1, 39 11, 51 11, 51 -1, 39 -1))', 4326)
);
```

```
SELECT SUM (sde.st_area (sde.st_difference (lot, footprint)))
 FROM footprints bf, lots
 WHERE bf.building_id = lots.lot_id;

sum

114
```

SQLite

```
--Create tables, add geometry columns, and insert values
CREATE TABLE footprints (
 building_id integer primary key autoincrement not null
);

SELECT AddGeometryColumn (
 NULL,
 'footprints',
 'footprint',
 4326,
 'polygon',
 'xy',
 'null'
);

CREATE TABLE lots (
 lot_id integer primary key autoincrement not null
);

SELECT AddGeometryColumn (
 NULL,
 'lots',
 'lot',
 4326,
 'polygon',
 'xy',
 'null'
);

INSERT INTO footprints (footprint) VALUES (
 st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
```

Copyright © 1995-2023 Esri. All rights reserved.

```
);

INSERT INTO footprints (footprint) VALUES (
 st_polygon ('polygon ((20 0, 20 10, 30 10, 30 0, 20 0))', 4326)
);

INSERT INTO footprints (footprint) VALUES (
 st_polygon ('polygon ((40 0, 40 10, 50 10, 50 0, 40 0))', 4326)
);

INSERT INTO lots (lot) VALUES (
 st_polygon ('polygon ((-1 -1, -1 11, 11 11, 11 -1, -1 -1))', 4326)
);

INSERT INTO lots (lot) VALUES (
 st_polygon ('polygon ((19 -1, 19 11, 29 9, 31 -1, 19 -1))', 4326)
);

INSERT INTO lots (lot) VALUES (
 st_polygon ('polygon ((39 -1, 39 11, 51 11, 51 -1, 39 -1))', 4326)
);
```

```
SELECT SUM (st_area (st_difference (lot, footprint)))
 FROM footprints bf, lots
 WHERE bf.building_id = lots.lot_id;

sum

114.0
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Dimension

Definición

ST_Dimension devuelve la dimensión de un objeto de geometría. En este caso, la dimensión se refiere al largo y al ancho. Por ejemplo, un punto no tiene largo ni ancho, por lo que su dimensión es 0; mientras que una línea tiene largo, pero no ancho, por lo que su dimensión es 1.

Sintaxis

Oracle y PostgreSQL

```
sde.st_dimension (geometry1 sde.st_geometry)
```

SQLite

```
st_dimension (geometry1 geometryblob)
```

Tipo de devolución

Entero

Ejemplo

La tabla dimension_test se crea con las columnas geotype y g1. La columna de geotipo almacena el nombre de la subclase almacenada en la columna de geometría g1.

La sentencia SELECT enumera el nombre de la subclase almacenado en la columna de geotipo con la dimensión de ese geotipo.

Oracle

```
CREATE TABLE dimension_test (
 geotype varchar(20),
 g1 sde.st_geometry
);

INSERT INTO DIMENSION_TEST VALUES (
 'Point',
 sde.st_pointfromtext ('point (10.02 20.01)', 4326)
);

INSERT INTO DIMENSION_TEST VALUES (
 'Linestring',
 sde.st_linefromtext ('linestring (10.02 20.01, 10.32 23.98, 11.92 25.64)', 4326)
);

INSERT INTO DIMENSION_TEST VALUES (
 'Polygon',
 sde.st_polyfromtext ('polygon ((10.02 20.01, 11.92 35.64, 25.02 34.15, 19.15 33.94,
10.02 20.01))', 4326)
);

INSERT INTO DIMENSION_TEST VALUES (
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 'Multipoint',
 sde.st_mpointfromtext ('multipoint ((10.02 20.01), (10.32 23.98), (11.92 25.64))',
4326)
);

INSERT INTO DIMENSION_TEST VALUES (
 'Multilinestring',
 sde.st_mlinefromtext ('multilinestring ((10.02 20.01, 10.32 23.98, 11.92 25.64), (9.55
23.75, 15.36 30.11))', 4326)
);

INSERT INTO DIMENSION_TEST VALUES (
 'Multipolygon',
 sde.st_mpolyfromtext ('multipolygon (((10.02 20.01, 11.92 35.64, 25.02 34.15,
19.15 33.94, 10.02 20.01), (51.71 21.73, 73.36 27.04, 71.52 32.87,
52.43 31.90, 51.71 21.73)))', 4326)
);
```

```
SELECT geotype, sde.st_dimension (g1) Dimension
 FROM DIMENSION_TEST;

GEOTYPE                    Dimension

Point                       0
Linestring                  1
Polygon                     2
Multipoint                  0
Multilinestring             1
Multipolygon                2
```

PostgreSQL

```
CREATE TABLE dimension_test (
 geotype varchar(20),
 g1 sde.st_geometry
);

INSERT INTO dimension_test VALUES (
 'Point',
 sde.st_point ('point (10.02 20.01)', 4326)
);

INSERT INTO dimension_test VALUES (
 'Linestring',
 sde.st_linestring ('linestring (10.02 20.01, 10.32 23.98, 11.92 25.64)', 4326)
);

INSERT INTO dimension_test VALUES (
 'Polygon',
 sde.st_polygon ('polygon ((10.02 20.01, 11.92 35.64, 25.02 34.15, 19.15 33.94, 10.02
20.01))', 4326)
);

INSERT INTO dimension_test VALUES (
 'Multipoint',
 sde.st_multipoint ('multipoint (10.02 20.01, 10.32 23.98, 11.92 25.64)', 4326)
);

INSERT INTO dimension_test VALUES (
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 'Multilinestring',
 sde.st_multilinestring ('multilinestring ((10.02 20.01, 10.32 23.98, 11.92 25.64),
(9.55 23.75, 15.36 30.11))', 4326)
);

INSERT INTO dimension_test VALUES (
 'Multipolygon',
 sde.st_multipolygon ('multipolygon (((10.02 20.01, 11.92 35.64, 25.02 34.15,
19.15 33.94, 10.02 20.01), (51.71 21.73, 73.36 27.04, 71.52 32.87,
52.43 31.90, 51.71 21.73)))', 4326)
);
```

```
SELECT geotype, sde.st_dimension (g1)
 AS Dimension
 FROM dimension_test;

geotype                  dimension

Point                       0
Linestring                  1
Polygon                     2
Multipoint                  0
Multilinestring             1
Multipolygon                2
```

SQLite

```
CREATE TABLE dimension_test (
 geotype varchar(20)
);

SELECT AddGeometryColumn (
 NULL,
 'dimension_test',
 'g1',
 4326,
 'geometry',
 'xy',
 'null'
);

INSERT INTO dimension_test VALUES (
 'Point',
 st_point ('point (10.02 20.01)', 4326)
);

INSERT INTO dimension_test VALUES (
 'Linestring',
 st_linestring ('linestring (10.02 20.01, 10.32 23.98, 11.92 25.64)', 4326)
);

INSERT INTO dimension_test VALUES (
 'Polygon',
 st_polygon ('polygon ((10.02 20.01, 11.92 35.64, 25.02 34.15, 19.15 33.94, 10.02
20.01))', 4326)
);

INSERT INTO dimension_test VALUES (
 'Multipoint',
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 st_multipoint ('multipoint ((10.02 20.01), (10.32 23.98), (11.92 25.64))', 4326)
);

INSERT INTO dimension_test VALUES (
 'Multilinestring',
 st_multilinestring ('multilinestring ((10.02 20.01, 10.32 23.98, 11.92 25.64), (9.55
23.75, 15.36 30.11))', 4326)
);

INSERT INTO dimension_test VALUES (
 'Multipolygon',
 st_multipolygon ('multipolygon (((10.02 20.01, 11.92 35.64, 25.02 34.15,
19.15 33.94, 10.02 20.01), (51.71 21.73, 73.36 27.04, 71.52 32.87,
52.43 31.90, 51.71 21.73)))', 4326)
);
```

```
SELECT geotype, st_dimension (g1)
 AS "Dimension"
 FROM dimension_test;

geotype                 Dimension

Point                      0
Linestring                 1
Polygon                    2
Multipoint                 0
Multilines                 1
Multipolyg                 2
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Disjoint

Definición

ST_Disjoint toma dos geometrías y devuelve 1 (Oracle y SQLite) o t (PostgreSQL) si la intersección de las dos geometrías produce un conjunto vacío; de lo contrario, devuelve 0 (Oracle y SQLite) o f (PostgreSQL).

Sintaxis

Oracle y PostgreSQL

```
sde.st_disjoint (geometry1 sde.st_geometry, geometry2 sde.st_geometry)
```

SQLite

```
st_disjoint (geometry1 geometryblob, geometry2 geometryblob)
```

Tipo de devolución

Booleano

Ejemplo

En este ejemplo, se crean dos tablas (distribution_areas y factories) y se insertan valores en cada una de ellas. A continuación, se crea una zona de influencia en torno a las fábricas y se usa st_disjoint para determinar qué zonas de influencia de fábricas no atraviesan las áreas de distribución.

Sugerencia:

Puede utilizar la función de ST_Intersects en su lugar en esta consulta al equiparar el resultado de la función a 0, porque ST_Intersects y ST_Disjoint devuelven resultados contrarios. La función de ST_Intersects utiliza el índice espacial al evaluar la consulta, mientras que la función ST_Disjoint no.

Oracle

```
--Create tables and insert values.
CREATE TABLE distribution_areas (
 id integer,
 areas sde.st_geometry
);

CREATE TABLE factories (
 id integer,
 loc sde.st_geometry
);

INSERT INTO distribution_areas (id, areas) VALUES (
 1,
 sde.st_geometry ('polygon ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO distribution_areas (id, areas) VALUES (
 2,
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 sde.st_geometry ('polygon ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
);

INSERT INTO distribution_areas (id, areas) VALUES (
 3,
 sde.st_geometry ('polygon ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
);

INSERT INTO factories (id,loc) VALUES (
 4,
 sde.st_geometry ('point (60 60)', 4326)
);

INSERT INTO factories (id,loc) VALUES (
 5,
 sde.st_geometry ('point (30 30)', 4326)
);
```

```
--Buffer factories and find which buffers are separate from distribution areas.
SELECT da.id
 FROM DISTRIBUTION_AREAS da, FACTORIES f
 WHERE sde.st_disjoint ((sde.st_buffer (f.loc, .001)), da.areas) = 1;
```

Copyright © 1995-2023 Esri. All rights reserved.

PostgreSQL

```
--Create tables and insert values.
CREATE TABLE distribution_areas (
 id serial,
 areas sde.st_geometry
);

CREATE TABLE factories (
 id serial,
 loc sde.st_geometry
);

INSERT INTO distribution_areas (areas) VALUES (
 sde.st_geometry ('polygon ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO distribution_areas (areas) VALUES (
 sde.st_geometry ('polygon ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
);

INSERT INTO distribution_areas (areas) VALUES (
 sde.st_geometry ('polygon ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
);

INSERT INTO factories (loc) VALUES (
 sde.st_geometry ('point (60 60)', 4326)
);

INSERT INTO factories (loc) VALUES (
 sde.st_geometry ('point (30 30)', 4326)
);
```

```
--Buffer factories and find which buffers are separate from distribution areas.
SELECT da.id
 FROM distribution_areas da, factories f
 WHERE sde.st_disjoint ((sde.st_buffer (f.loc, .001)), da.areas) = 't';
```

SQLite

```
--Create tables and insert values.
CREATE TABLE distribution_areas (
 id integer primary key autoincrement not null
);

SELECT AddGeometryColumn (
 NULL,
 'distribution_areas',
 'areas',
 4326,
 'polygon',
 'xy',
 'null'
);

CREATE TABLE factories (
 id integer primary key autoincrement not null
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
SELECT AddGeometryColumn (
 NULL,
 'factories',
 'loc',
 4326,
 'point',
 'xy',
 'null'
);

INSERT INTO distribution_areas (areas) VALUES (
 st_geometry ('polygon ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO distribution_areas (areas) VALUES (
 st_geometry ('polygon ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
);

INSERT INTO distribution_areas (areas) VALUES (
 st_geometry ('polygon ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
);

INSERT INTO factories (loc) VALUES (
 st_geometry ('point (60 60)', 4326)
);

INSERT INTO factories (loc) VALUES (
 st_geometry ('point (30 30)', 4326)
);
```

```
--Buffer factories and find which buffers are separate from distribution areas.
SELECT da.id
 FROM distribution_areas da, factories f
 WHERE st_disjoint ((st_buffer (f.loc, .001)), da.areas) = 1;

id

1
2
3
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Distance

Definición

ST_Distance devuelve la distancia entre dos geometrías. La distancia se mide desde el vértice más cercano de las dos geometrías.

Sintaxis

Oracle y PostgreSQL

```
sde.st_distance (geometry1 sde.st_geometry, geometry2 sde.st_geometry)
```

```
sde.st_distance (geometry1 sde.st_geometry, geometry2 sde.st_geometry, unit_name text)
```

SQLite

```
st_distance (geometry1 geometryblob, geometry2 geometryblob)
```

```
st_distance (geometry1 geometryblob, geometry2 geometryblob, unit_name text)
```

Los nombres de unidad válidos son los siguientes:

Milímetro	Pulgada	Yarda	Vínculo
Centímetro	Inch_US	Yard_US	Link_US
Decímetro	Pie	Yard_Clarke	Link_Clarke
Metro	Foot_US	Yard_Sears	Link_Sears
Meter_German	Foot_Clarke	Yard_Sears_1922_Truncated	Link_Sears_1922_Truncated
Kilómetro	Foot_Sears	Yard_Benoit_1895_A	Link_Benoit_1895_B
50_Kilometers	Foot_Sears_1922_Truncated	Yard_Indian	Cadena
150_Kilometers	Foot_Benoit_1895_A	Yard_Indian_1937	Chain_US
Vara_US	Foot_1865	Yard_Indian_1962	Chain_Clarke
Smoot	Foot_Indian	Yard_Indian_1975	Chain_Sears
	Foot_Indian_1937	Braza inglesa	Chain_Sears_1922_Truncated
	Foot_Indian_1962	Mile_US	Chain_Benoit_1895_A
	Foot_Indian_1975	Statute_Mile	Rod
	Foot_Gold_Coast	Nautical_Mile	Rod_US
	Foot_British_1936	Nautical_Mile_US	
		Nautical_Mile_UK	

Copyright © 1995-2023 Esri. All rights reserved.

Tipo de devolución

Precisión doble

Ejemplo

Se crean y se completan dos tablas, study1 y zones. A continuación, la función ST_Distance se usa para determinar la distancia entre el límite de cada subárea y los polígonos de la tabla de área study1 que tienen el código de uso 400. Puesto que hay tres zonas con esta forma, se deben devolver tres registros.

Si no especifica unidades, ST_Distance utiliza las unidades del sistema de proyección de los datos. En el primer ejemplo, se usan grados decimales. En los últimos dos ejemplos, se especifica el kilómetro; por tanto, la distancia se devuelve en kilómetros.

Oracle y PostgreSQL

```
--Create tables and insert values.
CREATE TABLE zones (
 sa_id integer,
 usecode integer,
 shape sde.st_geometry
);
CREATE TABLE study1 (
 code integer unique,
 shape sde.st_geometry
);
INSERT INTO zones (sa_id, usecode, shape) VALUES (
 1,
 400,
 sde.st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);
INSERT INTO zones (sa_id, usecode, shape) VALUES (
 2,
 400,
 sde.st_polygon ('polygon ((12 3, 12 6, 15 6, 15 3, 12 3))', 4326)
);
INSERT INTO zones (sa_id, usecode, shape) VALUES (
 3,
 400,
 sde.st_polygon ('polygon ((20 0, 20 10, 30 10, 30 0, 20 0))', 4326)
);
INSERT INTO zones (sa_id, usecode, shape) VALUES (
 4,
 402,
 sde.st_polygon ('polygon ((40 0, 40 10, 50 10, 50 0, 40 0))', 4326)
);
INSERT INTO study1 (code, shape) VALUES (
 400,
 sde.st_polygon ('polygon ((-1 -1, -1 11, 11 11, 19 11, 31 11, 31 -1, 19 -1, 11 -1, -1
-1))', 4326)
);
INSERT INTO study1 (code, shape) VALUES (
 402,
 sde.st_polygon ('polygon ((39 -1, 39 11, 51 11, 51 -1, 39 -1))', 4326)
);
```

```
--Oracle SELECT statement without units
SELECT UNIQUE s.code, z.sa_id, sde.st_distance(z.shape, sde.st_boundary(s.shape))
```

Copyright © 1995-2023 Esri. All rights reserved.

```
DISTANCE
 FROM zones z, study1 s
 WHERE z.usecode = s.code AND s.code = 400
 ORDER BY DISTANCE;
CODE       SA_ID      DISTANCE
---------- ---------- ----------
400                    1                                            1
400                           3                                            1
400                           3                                            4
--PostgreSQL SELECT statement without units
SELECT DISTINCT s.code, z.sa_id, sde.st_distance(z.shape, sde.st_boundary(s.shape))
 AS Distance
 FROM zones z, study1 s
 WHERE z.usecode = s.code AND s.code = 400
 ORDER BY Distance;
code             sa_id                distance
400           1                          1
400           3                          1
400                 2                          4
--Oracle SELECT statement with values returned in kilometers
SELECT UNIQUE s.code, z.sa_id, sde.st_distance(z.shape, sde.st_boundary(s.shape),
'kilometer') DISTANCE
 FROM zones z, study1 s
 WHERE z.usecode = s.code AND s.code = 400
 ORDER BY DISTANCE;
      CODE      SA_ID   DISTANCE
---------- ---------- ----------
       400          1 109.639196
       400          3 109.639196
       400          2 442.300258
--PostgreSQL SELECT statement with values returned in kilometers
SELECT DISTINCT s.code, z.sa_id, sde.st_distance(z.shape, sde.st_boundary(s.shape),
'kilometer')
 AS Distance
 FROM zones z, study1 s
 WHERE z.usecode = s.code AND s.code = 400
 ORDER BY Distance;
code             sa_id                distance
400       1      109.63919620267
400       3      109.63919620267
400       2      442.300258454087
```

SQLite

```
--Create tables, add geometry columns, and insert values.
CREATE TABLE zones (
 sa_id integer primary key autoincrement not null,
 usecode integer
);
SELECT AddGeometryColumn (
 NULL,
 'zones',
 'shape',
 4326,
 'polygon',
 'xy',
 'null'
);
CREATE TABLE study1 (
 code integer unique
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
SELECT AddGeometryColumn (
 NULL,
 'study1',
 'shape',
 4326,
 'polygon',
 'xy',
 'null'
);
INSERT INTO zones (usecode, shape) VALUES (
 400,
 st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);
INSERT INTO zones (usecode, shape) VALUES (
 400,
 st_polygon ('polygon ((12 3, 12 6, 15 6, 15 3, 12 3))', 4326)
);
INSERT INTO zones (usecode, shape) VALUES (
 400,
 st_polygon ('polygon ((20 0, 20 10, 30 10, 30 0, 20 0))', 4326)
);
INSERT INTO zones (usecode, shape) VALUES (
 402,
 st_polygon ('polygon ((40 0, 40 10, 50 10, 50 0, 40 0))', 4326)
);
INSERT INTO study1 (code, shape) VALUES (
 400,
 st_polygon ('polygon ((-1 -1, -1 11, 11 11, 19 11, 31 11, 31 -1, 19 -1, 11 -1, -1
-1))', 4326)
);
INSERT INTO study1 (code, shape) VALUES (
 402,
 st_polygon ('polygon ((39 -1, 39 11, 51 11, 51 -1, 39 -1))', 4326)
);
```

```
--SQLite SELECT statement without units
SELECT DISTINCT s.code, z.sa_id, st_distance(z.shape, st_boundary(s.shape))
 AS "Distance(km)"
 FROM zones z, study1 s
 WHERE z.usecode = s.code AND s.code = 400
 ORDER BY "Distance(km)";
code             sa_id                distance
400           1                                1
400           3                                1
400                  2                                4
--SQLite SELECT statement with units
SELECT DISTINCT s.code, z.sa_id, st_distance(z.shape, st_boundary(s.shape),
"kilometer")
 AS "Distance(km)"
 FROM zones z, study1 s
 WHERE z.usecode = s.code AND s.code = 400
 ORDER BY "Distance(km)";
code                 sa_id      Distance(km)
400                      1                                         109.63919620267
400                            3
109.63919620267
400                            2
442.30025845408
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_DWithin

Definición

ST_DWithin acepta dos geometrías como entrada y devuelve true si la distancia que separa a las geometrías es inferior o igual a la distancia especificada; de lo contrario, devuelve false. El sistema de referencia espacial de las geometrías determina qué unidad de medida se aplica a la distancia especificada. Por tanto, las dos geometrías que proporcione a ST_DWithin deben usar la misma proyección de coordenadas y el mismo Id. de referencia espacial (SRID).

Sintaxis

Oracle y PostgreSQL

```
sde.st_dwithin (st_geometry geometry1, st_geometry geometry2, double_precision distance);
```

SQLite

```
st_dwithin (geometryblob geometry1, geometryblob geometry2, double_precision distance);
```

Tipo de devolución

Booleano

Ejemplos

En los siguientes ejemplos se crean dos tablas y se insertan entidades en ellas. A continuación, se utiliza la función ST_DWithin en dos declaraciones SELECT diferentes: una para determinar si un punto de la primera tabla está a menos de 100 metros de un polígono de la segunda tabla y otra para determinar qué entidades están a menos de 300 metros entre sí.

Oracle

```
--Create table to store points.
CREATE TABLE dwithin_test_pt (id INT, geom sde.st_geometry);

--Create table to store polygons.
CREATE TABLE dwithin_test_poly (id INT, geom sde.st_geometry);

--Insert features into each table.

INSERT INTO dwithin_test_pt
 VALUES
 (
  1,
  sde.st_geometry('point (1 2)', 4326)
 )
;

INSERT INTO dwithin_test_pt
 VALUES
 (
  2,
  sde.st_geometry('point (10.02 20.01)', 4326)
 )
;
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO dwithin_test_poly
 VALUES
 (
  1,
  sde.st_geometry('polygon ((10.02 20.01, 11.92 35.64, 25.02 34.15, 19.15 33.94, 10.02
20.01))', 4326)
 )
;

INSERT INTO dwithin_test_poly
 VALUES
 (
  2,
  sde.st_geometry('polygon ((101.02 200.01, 111.92 350.64, 250.02 340.15, 190.15
330.94, 101.02 200.01))', 4326)
 )
;
```

A continuación, se utiliza ST_DWithin para determinar qué entidades de cada tabla están a menos de 100 metros de distancia entre sí y cuáles están más lejos. La función ST_Distance se incluye en esta sentencia para mostrar la distancia real entre las entidades.

```
--Determine which features in the point table are within 100 meters of the features in
the polygon table.
SELECT pt.id, poly.id, sde.st_distance(pt.geom, poly.geom) distance_meters,
sde.st_dwithin(pt.geom, poly.geom, 100) DWithin
 FROM dwithin_test_pt pt, dwithin_test_poly poly;
```

La sentencia devuelve lo siguiente:

```
ID     ID   DISTANCE_METERS   DWITHIN
1      1         20.1425048         1
1      2          221.83769         0
2      1                  0         1
2      2         201.695315         0
```

En el siguiente ejemplo, se utiliza ST_DWithin para determinar qué entidades están a menos de 300 metros de distancia entre sí:

```
--Determine which features in the point table are within 300 meters of the features in
the polygon table.
SELECT pt.id, poly.id, sde.st_distance(pt.geom, poly.geom) distance_meters,
sde.st_dwithin(pt.geom, poly.geom, 300) DWithin
 FROM dwithin_test_pt pt, dwithin_test_poly poly;
```

La segunda sentencia Select devuelve lo siguiente al ejecutarse en datos en Oracle:

```
ID     ID   DISTANCE_METERS    DWITHIN
1      1         20.1425048          1
1      2          221.83769          1
2      1                  0          1
2      2         201.695315          1
```

Copyright © 1995-2023 Esri. All rights reserved.

PostgreSQL

```
--Create table to store points.
CREATE TABLE dwithin_test_pt (id INT, geom sde.st_geometry);

--Create table to store polygons.
CREATE TABLE dwithin_test_poly (id INT, geom sde.st_geometry);

--Insert features into each table.

INSERT INTO dwithin_test_pt
 VALUES
 (
  1,
  sde.st_geometry('point (1 2)', 4326)
 )
;

INSERT INTO dwithin_test_pt
 VALUES
 (
  2,
  sde.st_geometry('point (10.02 20.01)', 4326)
 )
;

INSERT INTO dwithin_test_poly
 VALUES
 (
  1,
  sde.st_geometry('polygon ((10.02 20.01, 11.92 35.64, 25.02 34.15, 19.15 33.94, 10.02
20.01))', 4326)
 )
;

INSERT INTO dwithin_test_poly
 VALUES
 (
  2,
  sde.st_geometry('polygon ((101.02 200.01, 111.92 350.64, 250.02 340.15, 190.15
330.94, 101.02 200.01))', 4326)
 )
;
```

A continuación, se utiliza ST_DWithin para determinar qué entidades de cada tabla están a menos de 100 metros de distancia entre sí y cuáles están más lejos. La función ST_Distance se incluye en esta sentencia para mostrar la distancia real entre las entidades.

```
--Determine which features in the point table are within 100 meters of the features in
the polygon table.
SELECT pt.id, poly.id, sde.st_distance(pt.geom, poly.geom) distance_meters,
sde.st_dwithin(pt.geom, poly.geom, 100) DWithin
 FROM dwithin_test_pt pt, dwithin_test_poly poly;
```

La sentencia devuelve lo siguiente:

```
id     id     distance_meters        dwithin
1      1      20.1425048094819       t
```

Copyright © 1995-2023 Esri. All rights reserved.

```
1      2      221.837689538996       f
2      1      0                      t
2      2      201.69531476958        f
```

En el siguiente ejemplo, se utiliza ST_DWithin para determinar qué entidades están a menos de 300 metros de distancia entre sí:

```
--Determine which features in the point table are within 300 meters of the features in
the polygon table.
SELECT pt.id, poly.id, sde.st_distance(pt.geom, poly.geom) distance_meters,
sde.st_dwithin(pt.geom, poly.geom, 300) DWithin
 FROM dwithin_test_pt pt, dwithin_test_poly poly;
```

Esta segunda sentencia Select devuelve lo siguiente:

```
id     id     distance_meters        dwithin
1      1      20.1425048094819       t
1      2      221.837689538996       t
2      1      0                      t
2      2      201.69531476958        t
```

SQLite

```
--Create table to store points.
CREATE TABLE dwithin_test_pt (
 id integer not null
);

SELECT AddGeometryColumn(
 NULL,
 'dwithin_test_pt',
 'geom',
 4326,
 'point',
 'xy',
 'null'
);

--Create table to store polygons.
CREATE TABLE dwithin_test_poly (
 id integer not null
);

SELECT AddGeometryColumn(
 NULL,
 'dwithin_test_poly',
 'geom',
 4326,
 'polygon',
 'xy',
 'null'
);

--Insert features into each table.

INSERT INTO dwithin_test_pt
 VALUES
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 (
  1,
  st_geometry('point (1 2)', 4326)
 )
;

INSERT INTO dwithin_test_pt
 VALUES
 (
  2,
  st_geometry('point (10.02 20.01)', 4326)
 )
;

INSERT INTO dwithin_test_poly
 VALUES
 (
  1,
  st_geometry('polygon ((10.02 20.01, 11.92 35.64, 25.02 34.15, 19.15 33.94, 10.02
20.01))', 4326)
 )
;

INSERT INTO dwithin_test_poly
 VALUES
 (
  2,
  st_geometry('polygon ((101.02 200.01, 111.92 350.64, 250.02 340.15, 190.15 330.94,
101.02 200.01))', 4326)
 )
;
```

A continuación, se utiliza ST_DWithin para determinar qué entidades de cada tabla están a menos de 100 metros de distancia entre sí y cuáles están más lejos. La función ST_Distance se incluye en esta sentencia para mostrar la distancia real entre las entidades.

```
--Determine which features in the point table are within 100 meters of the features in
the polygon table.
SELECT pt.id, poly.id, st_distance(pt.geom, poly.geom) distance_meters,
st_dwithin(pt.geom, poly.geom, 100) DWithin
 FROM dwithin_test_pt pt, dwithin_test_poly poly;
```

La sentencia devuelve lo siguiente:

```
1|1|20.1425048094819|1
1|2|221.837689538996|0
2|1|0.0|1
2|2|201.69531476958|0
```

En el siguiente ejemplo, se utiliza ST_DWithin para determinar qué entidades están a menos de 300 metros de distancia entre sí:

```
--Determine which features in the point table are within 300 meters of the features in
the polygon table.
SELECT pt.id, poly.id, st_distance(pt.geom, poly.geom) distance_meters,
st_dwithin(pt.geom, poly.geom, 300) DWithin
```

Copyright © 1995-2023 Esri. All rights reserved.

```
FROM dwithin_test_pt pt, dwithin_test_poly poly;
```

Esta segunda sentencia Select devuelve lo siguiente:

```
1|1|20.1425048094819|1
1|2|221.837689538996|1
2|1|0.0|1
2|2|201.69531476958|1
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_EndPoint

Definición

ST_EndPoint devuelve el último punto de una cadena de texto.

Sintaxis

Oracle y PostgreSQL

```
sde.st_endpoint (line1 sde.st_geometry)
```

SQLite

```
st_endpoint (line1 geometryblob)
```

Tipo de devolución

ST_Point

Ejemplo

La tabla endpoint_test gid almacena la columna de enteros gid, que identifica exclusivamente cada fila, y la columna ST_LineString ln1, que almacena cadenas de texto de líneas.

Las declaraciones INSERT insertar cadenas de texto de líneas en la tabla endpoint_test. La primera cadena de texto de líneas no tienen coordenadas z o medidas, mientras que la segunda sí.

La consulta enumera la columna gid y la geometría ST_Point generada por la función ST_EndPoint.

Oracle

```
--Create table and insert values.
CREATE TABLE endpoint_test (
 gid integer,
 ln1 sde.st_geometry
);

INSERT INTO ENDPOINT_TEST VALUES (
 1,
 sde.st_linefromtext ('linestring (10.02 20.01, 23.73 21.92, 30.10 40.23)', 4326)
);

INSERT INTO ENDPOINT_TEST VALUES (
 2,
 sde.st_linefromtext ('linestring zm(10.02 20.01 5.0 7.0, 23.73 21.92 6.5 7.1,30.10
40.23 6.9 7.2)', 4326)
);
```

```
--Find the end point of each line.
SELECT gid, sde.st_astext (sde.st_endpoint (ln1)) Endpoint
 FROM ENDPOINT_TEST;
```

Copyright © 1995-2023 Esri. All rights reserved.

```
GID         Endpoint

 1    POINT (30.10 40.23)
 2    POINT ZM (30.10 40.23 6.9 7.2)
```

PostgreSQL

```
--Create table and insert values.
CREATE TABLE endpoint_test (
 gid integer,
 ln1 sde.st_geometry
);

INSERT INTO endpoint_test VALUES (
 1,
st_linestring ('linestring (10.02 20.01, 23.73 21.92, 30.10 40.23)', 4326)
);

INSERT INTO endpoint_test VALUES (
 2,
 st_linestring ('linestring zm(10.02 20.01 5.0 7.0, 23.73 21.92 6.5 7.1,30.10 40.23 6.9
7.2)', 4326)
);
```

```
--Find the end point of each line.
SELECT gid, st_astext (st_endpoint (ln1))
 AS endpoint
 FROM endpoint_test;

gid          endpoint

 1    POINT (30.10 40.23)
 2    POINT ZM (30.10 40.23 6.9 7.2)
```

SQLite

```
--Create table, add geometry column, and insert values.
CREATE TABLE endpoint_test (
 gid integer primary key autoincrement not null
);

SELECT AddGeometryColumn (
 NULL,
 'endpoint_test',
 'ln1',
 4326,
 'linestringzm',
 'xyzm',
 'null'
);

INSERT INTO endpoint_test (ln1) VALUES (
 st_linestring ('linestring (10.02 20.01, 23.73 21.92, 30.10 40.23)', 4326)
);

INSERT INTO endpoint_test (ln1) VALUES (
 st_linestring ('linestring zm(10.02 20.01 5.0 7.0, 23.73 21.92 6.5 7.1,30.10 40.23 6.9
```

Copyright © 1995-2023 Esri. All rights reserved.

```
7.2)', 4326)
);
```

```
--Find the end point of each line.
SELECT gid, st_astext (st_endpoint (ln1))
 AS "endpoint"
 FROM endpoint_test;

gid          endpoint

 1    POINT (30.10000000 40.23000000)
 2    POINT ZM (30.10000000 40.23000000 6.90000000 7.20000000)
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Entity

Definición

ST_Entity devuelve el tipo de entidad espacial de un objeto de geometría. El tipo de entidad espacial es el valor almacenado en el campo de miembro de la entidad del objeto de geometría.

Sintaxis

Oracle y PostgreSQL

```
sde.st_entity (geometry1 sde.st_geometry)
```

SQLite

```
st_entity (geometry1 geometryblob)
```

Tipo de devolución

Se devuelve un número (Oracle) o entero (SQLite y PostgreSQL) que representa los siguientes tipos de entidad:

0	forma nula
1	punto
2	línea (incluidas las líneas espagueti)
4	cadena de líneas
8	área
257	multipunto
258	multilínea (incluidas las líneas espagueti)
260	cadena de texto multilínea
264	multiárea

Ejemplo

En los ejemplos siguientes se crea una tabla y se insertan diferentes geometrías en la tabla. Se ejecuta ST_Entity en la tabla para devolver el subtipo de geometría de cada registro de la tabla.

Oracle

```
CREATE TABLE sample_geos (
 id integer,
 geometry sde.st_geometry
);
INSERT INTO sample_geos (id, geometry) VALUES (
 1901,
 sde.st_geometry ('point (1 2)', 4326)
);
INSERT INTO sample_geos (id, geometry) VALUES (
 1902,
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 sde.st_geometry ('linestring (33 2, 34 3, 35 6)', 4326)
);
INSERT INTO sample_geos (id, geometry) VALUES (
 1903,
 sde.st_geometry ('polygon ((3 3, 4 6, 5 3, 3 3))', 4326)
);
SELECT sde.st_entity (geometry) entity, UPPER (sde.st_geometrytype (geometry)) TYPE
 FROM sample_geos;
```

La sentencia SELECT devuelve los siguientes valores:

```
 ENTITY        TYPE
    1    ST_POINT
    4    ST_LINESTRING
    8    ST_POLYGON
```

PostgreSQL

```
CREATE TABLE sample_geos (
 id integer,
 geometry sde.st_geometry
);
INSERT INTO sample_geos (id, geometry) VALUES (
 1900,
 sde.st_geometry ('Point Empty', 4326)
);
INSERT INTO sample_geos (id, geometry) VALUES (
 1901,
 sde.st_geometry ('point (1 2)', 4326)
);
INSERT INTO sample_geos (id, geometry) VALUES (
 1902,
 sde.st_geometry ('linestring (33 2, 34 3, 35 6)', 4326)
);
INSERT INTO sample_geos (id, geometry) VALUES (
 1903,
 sde.st_geometry ('polygon ((3 3, 4 6, 5 3, 3 3))', 4326)
);
INSERT INTO sde.entity_test (id, geometry) VALUES (
 1904,
 sde.st_geometry ('multipoint (10.01 20.03, 10.52 40.11, 30.29 41.56, 31.78 10.74)',
4326)
);
INSERT INTO sde.entity_test (id, geometry) VALUES (
 1905,
 sde.st_geometry ('multilinestring ((10.01 20.03, 10.52 40.11, 30.29 41.56,31.78
10.74), (20.93 20.81, 21.52 40.10))', 4326)
);
INSERT INTO sde.entity_test (id, geometry) VALUES (
 1906,
 sde.st_geometry ('multipolygon (((3 3, 4 6, 5 3, 3 3), (8 24, 9 25, 1 28, 8 24), (13
33, 7 36, 1 40, 10 43, 13 33)))', 4326)
);
SELECT id AS "id",
 sde.st_entity (geometry) AS "entity",
 sde.st_geometrytype (geometry) AS "geom_type"
 FROM sample_geos;
```

Copyright © 1995-2023 Esri. All rights reserved.

La sentencia SELECT devuelve los siguientes valores:

```
 id       entity     geom_type
    1900     0          "ST_GEOMETRY"
    1901     1          "ST_POINT"
    1902     4          "ST_LINESTRING"
    1903     8          "ST_POLYGON"
    1904     257        "ST_MULTIPOINT"
    1905     260        "ST_MULTILINESTRING"
    1906     264        "ST_MULTIPOLYGON"
```

SQLite

```
CREATE TABLE sample_geos (
 id integer primary key autoincrement not null
);
SELECT AddGeometryColumn (
 NULL,
 'sample_geos',
 'geometry',
 4326,
 'geometry',
 'xy',
 'null'
);
INSERT INTO sample_geos (geometry) VALUES (
 st_geometry ('point (1 2)', 4326)
);
INSERT INTO sample_geos (geometry) VALUES (
 st_geometry ('linestring (33 2, 34 3, 35 6)', 4326)
);
INSERT INTO sample_geos (geometry) VALUES (
 st_geometry ('polygon ((3 3, 4 6, 5 3, 3 3))', 4326)
);
SELECT st_entity (geometry) AS "entity",
 st_geometrytype (geometry) AS "type"
 FROM sample_geos;
```

La sentencia SELECT devuelve los siguientes valores:

```
entity    type
1         ST_POINT
4         ST_LINESTRING
8         ST_POLYGON
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Envelope

Definición

ST_Envelope devuelve el cuadro mínimo de delimitación de un objeto de geometría como un polígono.

Explorar:
Esta función cumple la especificación de Entidades Simples de Open Geospatial Consortium (OGC) que establece que ST_Envelope devuelve un polígono. Para trabajar con casos especiales de geometrías de puntos o líneas horizontales o verticales, la función ST_Envelope devuelve un polígono alrededor de estas formas que es una tolerancia de contorno pequeña calculada en función del factor de escala XY para el sistema de referencia espacial de la geometría. Esta tolerancia se resta de las coordenadas x e y mínimas y se suma a las coordenadas x e y máximas para devolver el polígono alrededor de estas formas.

Sintaxis

Oracle y PostgreSQL

```
sde.st_envelope (geometry1 sde.st_geometry)
```

SQLite

```
st_envelope (geometry1 geometryblob)
```

Tipo de devolución

Oracle y PostgreSQL

ST_Geometry

SQLite

Geometryblob

Ejemplo

La columna de geotipo de la tabla envelope_test guarda el nombre de la subclase de geometría guardada en la columna g1. Las declaraciones sentencias INSERT insertan cada subclase de geometría en la tabla envelope_test.

A continuación, se ejecuta la función ST_Envelope para devolver el contorno del polígono alrededor de cada geometría.

Oracle

```
--Create table and insert values.
CREATE TABLE envelope_test (
 geotype varchar(20),
 g1 sde.st_geometry
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO ENVELOPE_TEST VALUES (
'Point',
sde.st_geometry ('point (-1509734.232 -36684.757)', 102004)
);

INSERT INTO ENVELOPE_TEST VALUES (
'Linestring',
sde.st_geometry ('linestring (-1511144.181 -37680.015, -1509734.232 -38841.149,
-1508656.036 -39753.469)', 102004)
);

INSERT INTO ENVELOPE_TEST VALUES (
'Polygon',
sde.st_geometry ('polygon ((-1506333.768 -36435.943, -1504343.252 -36767.695,
-1502684.489 -35357.747, -1506333.768 -36435.943))', 102004)
);

INSERT INTO ENVELOPE_TEST VALUES (
'Multipoint',
sde.st_geometry ('multipoint ((-1493229.539 -40665.789), (-1494141.859 -40831.665),
(-1495800.622 -42739.242))', 102004)
);

INSERT INTO ENVELOPE_TEST VALUES (
'Multilinestring',
sde.st_geometry ('multilinestring ((-1504757.943 -33201.355, -1507411.964 -35606.561),
(-1502518.613 -38094.706, -1499781.653 -37099.448, -1498952.272 -34694.241))', 102004)
);

INSERT INTO ENVELOPE_TEST VALUES (
'Multipolygon',
sde.st_geometry ('multipolygon (((-1492068.405 -47300.841, -1492814.848 -45725.016,
-1493975.983 -46471.459,
-1493478.354 -47798.47, -1492068.405 -47300.841), (-1497874.076 -48047.284,
-1498537.581 -50618.367, -1497210.571 -50037.8,
-1497874.076 -48047.284)))', 102004)
);
```

```
--Return the polygon envelope around each geometry in well-known text.
SELECT geotype geometry_type,
 sde.st_astext (sde.st_envelope (g1)) envelope
 FROM ENVELOPE_TEST;

GEOMETRY_TYPE    ENVELOPE

Point           |POLYGON (( -1509734.23220000 -36684.75720000, -1509734.23180000
-36684.75720000,
-1509734.23180000 -36684.75680000, -1509734.23220000 -36684.75680000, -1509734.23220000
-36684.75720000))

Linestring      |POLYGON (( -1511144.18100000 -39753.46900000, -1508656.03600000
-39753.46900000,
-1508656.03600000 -37680.01500000, -1511144.18100000 -37680.01500000, -1511144.18100000
-39753.46900000))

Polygon         |POLYGON (( -1506333.76800000 -36767.69500000, -1502684.48900000
-36767.69500000,
-1502684.48900000 -35357.74700000, -1506333.76800000 -35357.74700000, -1506333.76800000
-36767.69500000))

Multipoint      |POLYGON (( -1495800.62200000 -42739.24200000, -1493229.53900000
```

Copyright © 1995-2023 Esri. All rights reserved.

```
-42739.24200000,
-1493229.53900000 -40665.78900000, -1495800.62200000 -40665.78900000, -1495800.62200000
-42739.24200000))

Multilinestring |POLYGON (( -1507411.96400000 -38094.70600000, -1498952.27200000
-38094.70600000,
-1498952.27200000 -33201.35500000, -1507411.96400000 -33201.35500000, -1507411.96400000
-38094.70600000))

Multipolygon    |POLYGON (( -1498537.58100000 -50618.36700000, -1492068.40500000
-50618.36700000,
-1492068.40500000 -45725.01600000, -1498537.58100000 -45725.01600000, -1498537.58100000
-50618.36700000))
```

Copyright © 1995-2023 Esri. All rights reserved.

PostgreSQL

```
--Create table and insert values.
CREATE TABLE envelope_test (
 geotype varchar(20),
 g1 sde.st_geometry
);

INSERT INTO ENVELOPE_TEST VALUES (
'Point',
sde.st_geometry ('point (-1509734.232 -36684.757)', 102004)
);

INSERT INTO ENVELOPE_TEST VALUES (
'Linestring',
sde.st_geometry ('linestring (-1511144.181 -37680.015, -1509734.232 -38841.149,
-1508656.036 -39753.469)', 102004)
);

INSERT INTO ENVELOPE_TEST VALUES (
'Polygon',
sde.st_geometry ('polygon ((-1506333.768 -36435.943, -1504343.252 -36767.695,
-1502684.489 -35357.747, -1506333.768 -36435.943))', 102004)
);

INSERT INTO ENVELOPE_TEST VALUES (
'Multipoint',
sde.st_geometry ('multipoint (-1493229.539 -40665.789, -1494141.859 -40831.665,
-1495800.622 -42739.242)', 102004)
);

INSERT INTO ENVELOPE_TEST VALUES (
'Multilinestring',
sde.st_geometry ('multilinestring ((-1504757.943 -33201.355, -1507411.964 -35606.561),
(-1502518.613 -38094.706, -1499781.653 -37099.448, -1498952.272 -34694.241))', 102004)
);

INSERT INTO ENVELOPE_TEST VALUES (
'Multipolygon',
sde.st_geometry ('multipolygon (((-1492068.405 -47300.841, -1492814.848 -45725.016,
-1493975.983 -46471.459,
-1493478.354 -47798.47, -1492068.405 -47300.841), (-1497874.076 -48047.284,
-1498537.581 -50618.367, -1497210.571 -50037.8,
-1497874.076 -48047.284)))', 102004)
);
```

```
--Return the polygon envelope around each geometry in well-known text.
SELECT geotype AS geometry_type,
 sde.st_astext (sde.st_envelope (g1)) AS Envelope
 FROM envelope_test;

geometry_type     envelope

"Point"           |"POLYGON (( -1509734.23220000 -36684.75720000, -1509734.23180000
-36684.75720000,
-1509734.23180000 -36684.75680000, -1509734.23220000 -36684.75680000, -1509734.23220000
-36684.75720000))"

"Linestring"      |"POLYGON (( -1511144.18100000 -39753.46900000, -1508656.03600000
-39753.46900000,
-1508656.03600000 -37680.01500000, -1511144.18100000 -37680.01500000, -1511144.18100000
```

Copyright © 1995-2023 Esri. All rights reserved.

```
-39753.46900000))"

"Polygon"         |"POLYGON (( -1506333.76800000 -36767.69500000, -1502684.48900000
-36767.69500000,
-1502684.48900000 -35357.74700000, -1506333.76800000 -35357.74700000, -1506333.76800000
-36767.69500000))"

"Multipoint"      |"POLYGON (( -1495800.62200000 -42739.24200000, -1493229.53900000
-42739.24200000,
-1493229.53900000 -40665.78900000, -1495800.62200000 -40665.78900000, -1495800.62200000
-42739.24200000))"

"Multilinestring" |"POLYGON (( -1507411.96400000 -38094.70600000, -1498952.27200000
-38094.70600000,
-1498952.27200000 -33201.35500000, -1507411.96400000 -33201.35500000, -1507411.96400000
-38094.70600000))"

"Multipolygon"    |"POLYGON (( -1498537.58100000 -50618.36700000, -1492068.40500000
-50618.36700000,
-1492068.40500000 -45725.01600000, -1498537.58100000 -45725.01600000, -1498537.58100000
-50618.36700000))"
```

SQLite

```
--Create table and insert values.
CREATE TABLE envelope_test (
 geotype varchar(20)
);

SELECT AddGeometryColumn (
 NULL,
 'envelope_test',
 'g1',
 4326,
 'geometry',
 'xy',
 'null'
);

INSERT INTO ENVELOPE_TEST VALUES (
 'Point',
 st_geometry ('point (-1509734.232 -36684.757)', 102004)
);

INSERT INTO ENVELOPE_TEST VALUES (
 'Linestring',
 st_geometry ('linestring (-1511144.181 -37680.015, -1509734.232 -38841.149,
-1508656.036 -39753.469)', 102004)
);

INSERT INTO ENVELOPE_TEST VALUES (
 'Polygon',
 st_geometry ('polygon ((-1506333.768 -36435.943, -1504343.252 -36767.695, -1502684.489
-35357.747, -1506333.768 -36435.943))', 102004)
);

INSERT INTO ENVELOPE_TEST VALUES (
 'Multipoint',
 st_geometry ('multipoint ((-1493229.539 -40665.789), (-1494141.859 -40831.665),
(-1495800.622 -42739.242))', 102004)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO ENVELOPE_TEST VALUES (
 'Multilinestring',
 st_geometry ('multilinestring ((-1504757.943 -33201.355, -1507411.964 -35606.561),
(-1502518.613 -38094.706, -1499781.653 -37099.448, -1498952.272 -34694.241))', 102004)
);

INSERT INTO ENVELOPE_TEST VALUES (
 'Multipolygon',
 st_geometry ('multipolygon (((-1492068.405 -47300.841, -1492814.848 -45725.016,
-1493975.983 -46471.459,
-1493478.354 -47798.47, -1492068.405 -47300.841), (-1497874.076 -48047.284,
-1498537.581 -50618.367, -1497210.571 -50037.8,
-1497874.076 -48047.284)))', 102004)
);
```

```
--Return the polygon envelope around each geometry in well-known text.
SELECT geotype AS geometry_type,
 st_astext (st_envelope (g1)) AS "Envelope"
 FROM envelope_test;

geometry_type   Envelope

Point           POLYGON (( -1509734.23220000 -36684.75720000, -1509734.23180000
-36684.75720000,
-1509734.23180000 -36684.75680000, -1509734.23220000 -36684.75680000, -1509734.23220000
-36684.75720000))

Linestring      POLYGON (( -1511144.18100000 -39753.46900000, -1508656.03600000
-39753.46900000,
-1508656.03600000 -37680.01500000, -1511144.18100000 -37680.01500000, -1511144.18100000
-39753.46900000))

Polygon         POLYGON (( -1506333.76800000 -36767.69500000, -1502684.48900000
-36767.69500000,
-1502684.48900000 -35357.74700000, -1506333.76800000 -35357.74700000, -1506333.76800000
-36767.69500000))

Multipoint      POLYGON (( -1495800.62200000 -42739.24200000, -1493229.53900000
-42739.24200000,
-1493229.53900000 -40665.78900000, -1495800.62200000 -40665.78900000, -1495800.62200000
-42739.24200000))

Multilinestring POLYGON (( -1507411.96400000 -38094.70600000, -1498952.27200000
-38094.70600000,
-1498952.27200000 -33201.35500000, -1507411.96400000 -33201.35500000, -1507411.96400000
-38094.70600000))

Multipolygon    POLYGON (( -1498537.58100000 -50618.36700000, -1492068.40500000
-50618.36700000,
-1492068.40500000 -45725.01600000, -1498537.58100000 -45725.01600000, -1498537.58100000
-50618.36700000))
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_EnvIntersects

Nota:
Solo Oracle y SQLite

Definición

ST_EnvIntersects devuelve 1 (verdadero) si se intersecan los sobres de las dos geometrías; de lo contrario, devuelve 0 (falso).

Sintaxis

Oracle

```
sde.st_envintersects (geometry1 sde.st_geometry, geometry2 sde.st_geometry)
sde.st_envintersects (geometry1 sde.st_geometry, minx number, miny number, maxx number,
maxy number)
```

SQLite

```
st_envintersects (geometry1 geometryblob, geometry2 geoemtryblob)
st_envintersects (geometry1 geoemtryblob, minx float64, miny float64, maxx float64,
maxy float64)
```

Tipo de devolución

Booleano

Ejemplo

En este ejemplo se busca una geometría que tiene un sobre que se interseca con el polígono definido.

La primera declaración SELECT compara los sobres de las dos geometrías y las propias geometrías para ver si las entidades o los sobres se intersecan.

La segunda declaración SELECT usa un sobre para detectar las entidades, si las hubiera, que se encuentran dentro del sobre que se transmite con la cláusula WHERE de la declaración SELECT.

Oracle

```
--Define and populate the table.
CREATE TABLE sample_geoms (
 id integer,
 geometry sde.st_geometry);

INSERT INTO SAMPLE_GEOMS (id, geometry) VALUES (
 1,
 sde.st_geometry ('linestring (10 10, 50 50)', 4326)
);

INSERT INTO SAMPLE_GEOMS (id, geometry) VALUES (
 2,
 sde.st_geometry ('linestring (10 20, 50 60)', 4326)
```

Copyright © 1995-2023 Esri. All rights reserved.

```
);
```

```
--Find the intersection of the geometries and the geometries' envelopes.
SELECT a.id, b.id, sde.st_intersects (a.geometry, b.geometry) Intersects,
sde.st_envintersects (a.geometry, b.geometry) Envelope_Intersects
 FROM SAMPLE_GEOMS a, SAMPLE_GEOMS b
 WHERE a.id = 1 AND b.id=2;

ID      ID      INTERSECTS   ENVELOPE_INTERSECTS

1       2       0            1
```

```
--Find the geometries whose envelopes intersect the specified envelope.
SELECT id
 FROM SAMPLE_GEOMS
 WHERE sde.st_envintersects(geometry, 5, 5, 60, 65) = 1;

ID

1
2
```

SQLite

```
--Define and populate the table.
CREATE TABLE sample_geoms (
 id integer primary key autoincrement not null
);

SELECT AddGeometryColumn (
 NULL,
 'sample_geoms',
 'geometry',
 4326,
 'linestring',
 'xy',
 'null'
);

INSERT INTO SAMPLE_GEOMS (geometry) VALUES (
 st_geometry ('linestring (10 10, 50 50)', 4326)
);

INSERT INTO SAMPLE_GEOMS (geometry) VALUES (
 st_geometry ('linestring (10 20, 50 60)', 4326)
);
```

```
--Find the intersection of the geometries and the geometries' envelopes.
SELECT a.id AS aid, b.id AS bid, st_intersects (a.geometry, b.geometry) AS "Intersects",
 st_envintersects (a.geometry, b.geometry) AS "Envelope_Intersects"
 FROM SAMPLE_GEOMS a, SAMPLE_GEOMS b
 WHERE a.id = 1 AND b.id = 2;

aid     bid     Intersects   Envelope_Intersects
```

Copyright © 1995-2023 Esri. All rights reserved.

```
1       2       0            1
```

```
--Find the geometries whose envelopes intersect the specified envelope.
SELECT id
 FROM SAMPLE_GEOMS
 WHERE st_envintersects(geometry, 5, 5, 60, 65) = 1;

ID

1
2
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Equals

Definición

ST_Equals compara dos geometrías y devuelve 1 (Oracle y SQLite) o t (PostgreSQL) si las geometrías son idénticas; de lo contrario, devuelve 0 (Oracle y SQLite) o f (PostgreSQL).

Sintaxis

Oracle y PostgreSQL

```
sde.st_equals (geometry1 sde.st_geometry, geometry2 sde.st_geometry)
```

SQLite

```
st_equals (geometry1 geometryblob, geometry2 geometryblob)
```

Tipo de devolución

Booleano

Ejemplo

El técnico SIG sospecha que algunos de los datos de la tabla studies están duplicados. Para despejar sus dudas, consulta la tabla con el fin de determinar si algunos de los multipolígonos de forma son iguales.

La tabla studies se crea y se completa con las siguientes declaraciones. La columna Id. identifica de forma única los edificios, y el campo forma almacena la geometría del área.

A continuación, la tabla studies se une espacialmente a sí misma mediante el predicado de igualdad, que devuelve 1 (Oracle y SQLite) o t (PostgreSQL) siempre que encuentra dos multipolígonos iguales. La condición s1.id<>s2.id elimina la comparación de una geometría consigo misma.

Oracle

```
CREATE TABLE studies (
 id integer unique,
 shape sde.st_geometry
);

INSERT INTO studies (id, shape) VALUES (
 1,
 sde.st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);

INSERT INTO studies (id, shape) VALUES (
 2,
 sde.st_polygon ('polygon ((20 0, 20 10, 30 10, 30 0, 20 0))', 4326)
);

INSERT INTO studies (id, shape) VALUES (
 3,
 sde.st_polygon ('polygon ((40 0, 40 10, 50 10, 50 0, 40 0))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO studies (id, shape) VALUES (
 4,
 sde.st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);
```

```
SELECT UNIQUE (s1.id), s2.id
 FROM STUDIES s1, STUDIES s2
 WHERE sde.st_equals (s1.shape, s2.shape) = 1
 AND s1.id <> s2.id;

ID     ID

 4       1
 1       4
```

PostgreSQL

```
CREATE TABLE studies (
 id serial,
 shape st_geometry
);

INSERT INTO studies (shape) VALUES (
 st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);

INSERT INTO studies (shape) VALUES (
 st_polygon ('polygon ((20 0, 20 10, 30 10, 30 0, 20 0))', 4326)
);

INSERT INTO studies (shape) VALUES (
 st_polygon ('polygon ((40 0, 40 10, 50 10, 50 0, 40 0))', 4326)
);

INSERT INTO studies (shape) VALUES (
 st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);
```

```
SELECT DISTINCT (s1.id), s2.id
 FROM studies s1, studies s2
 WHERE st_equals (s1.shape, s2.shape) = 't'
 AND s1.id <> s2.id;

id    id

 1      4
 4      1
```

SQLite

```
CREATE TABLE studies (
 id integer primary key autoincrement not null
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
SELECT AddGeometryColumn (
 NULL,
 'studies',
 'shape',
 4326,
 'polygon',
 'xy',
 'null'
);

INSERT INTO studies (shape) VALUES (
 st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);

INSERT INTO studies (shape) VALUES (
 st_polygon ('polygon ((20 0, 20 10, 30 10, 30 0, 20 0))', 4326)
);

INSERT INTO studies (shape) VALUES (
 st_polygon ('polygon ((40 0, 40 10, 50 10, 50 0, 40 0))', 4326)
);

INSERT INTO studies (shape) VALUES (
 st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);
```

```
SELECT DISTINCT (s1.id), s2.id
 FROM studies s1, studies s2
 WHERE st_equals (s1.shape, s2.shape) = 1
 AND s1.id <> s2.id;

id    id

 1      4
 4      1
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Equalsrs

Nota:
Solo PostgreSQL

Definición

ST_Equalsrs comprueba si dos sistemas de referencia espacial de dos clases de entidad diferentes son idénticos. Si los sistemas de referencia espacial son idénticos, devuelve t (true). Si los sistemas de referencia espacial no son idénticos, ST_Equalsrs devuelve f (false).

Sintaxis

```
sde.st_equalsrs (srid1 integer, srid2 integer)
```

Tipo de devolución

Booleano

Ejemplo

En este ejemplo, se averiguan los Id. de referencia espacial (SRID) de diferentes clases de entidad y, a continuación, se utiliza ST_Equalsrs para ver si los SRID representan el mismo sistema de referencia espacial.

```
SELECT srid, table_name
FROM sde_layers
WHERE table_name = 'transmains' OR table_name = 'streets';
```

srid	table_name
2	streets
6	transmains

Resultados de la consulta de sde_layers

Ahora, utilice ST_Equalsrs para determinar si los sistemas de referencia espacial identificados por estos dos SRID son iguales.

```
SELECT sde.st_equalsrs(2,6) ;

 st_equalsrs

-------------

f

(1 row)
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_ExteriorRing

Definición

ST_ExteriorRing devuelve el anillo exterior de un polígono como una cadena de texto de líneas.

Sintaxis

```
sde.st_exteriorring (polygon1 sde.st_geometry)
```

Oracle y PostgreSQL

```
sde.st_exteriorring (polygon1 sde.st_geometry)
```

SQLite

```
st_exteriorring (polygon1 geometryblob)
```

Tipo de devolución

ST_LineString

Ejemplo

Un ornitólogo, desea estudiar la población de aves en varias islas, sabe que la zona de alimentación de las especies de aves en la que está interesado está restringida a la línea costera. Como parte de su cálculo de la capacidad de carga de las islas, el ornitólogo requiere los perímetros de las islas. Algunas de las islas son tan grandes que tienen varios lagos en ellas. Sin embargo, la costa de los lagos está habitado únicamente por otras especies de aves más agresivas. Por lo tanto, la ornitólogo requiere el perímetro de solo el anillo exterior de las islas.

Las columnas de Id. y de nombre de la tabla de islas identifica de cada isla, mientras que la columna del polígono de tierra almacena la geometría de la isla.

La función ST_ExteriorRing extrae el anillo exterior de cada polígono de isla como una cadena de texto de líneas. La longitud de la cadena de texto de líneas se calcula según la función ST_Length. Las longitudes de la cadena de texto de líneas se resume según la función SUM.

Los anillos exteriores de las islas representan la interfaz ecológica que cada isla comparte con el mar.

Oracle

```
--Create the table and insert two polygons.
CREATE TABLE islands (
 id integer,
 name varchar(32),
 land sde.st_geometry
);

INSERT INTO islands VALUES (
1,
'Bear',
sde.st_polygon ('polygon ((40 120, 90 120, 90 150, 40 150, 40 120),(50 130, 60 130, 60
```

Copyright © 1995-2023 Esri. All rights reserved.

```
140, 50 140, 50 130),
(70 130, 80 130, 80 140, 70 140, 70 130))', 4326)
);

INSERT INTO islands VALUES (
 2,
 'Johnson',
sde.st_polygon ('polygon ((10 10, 50 10, 10 30, 10 10))', 4326)
);
```

```
--Extract the exterior ring from each island and find its length.
SELECT SUM (sde.st_length (sde.st_exteriorring (land)))
 FROM ISLANDS;

SUM(ST_LENGTH(ST_EXTERIORRING(LAND)))

                            264.72136
```

PostgreSQL

```
--Create the table and insert two polygons.
CREATE TABLE islands (
 id serial,
 name varchar(32),
 land sde.st_geometry
);

INSERT INTO islands (name, land) VALUES (
 'Bear',
 sde.st_polygon ('polygon ((40 120, 90 120, 90 150, 40 150, 40 120),(50 130, 60 130, 60
140, 50 140, 50 130),
(70 130, 80 130, 80 140, 70 140, 70 130))', 4326)
);

INSERT INTO islands (name, land) VALUES (
 'Johnson',
 sde.st_polygon ('polygon ((10 10, 50 10, 10 30, 10 10))', 4326)
);
```

```
--Extract the exterior ring from each island and find its length.
SELECT SUM (sde.st_length (sde.st_exteriorring (land)))
 FROM islands;

sum

264.721359549996
```

SQLite

```
--Create the table and insert two polygons.
CREATE TABLE islands (
 id integer primary key autoincrement not null,
 name varchar(32)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
SELECT AddGeometryColumn (
 NULL,
 'islands',
 'land',
 4326,
 'polygon',
 'xy',
 'null'
);

INSERT INTO islands (name, land) VALUES (
 'Bear',
 st_polygon ('polygon ((40 120, 90 120, 90 150, 40 150, 40 120),(50 130, 60 130, 60
140, 50 140, 50 130),
(70 130, 80 130, 80 140, 70 140, 70 130))', 4326)
);

INSERT INTO islands (name, land) VALUES (
 'Johnson',
 st_polygon ('polygon ((10 10, 50 10, 10 30, 10 10))', 4326)
);
```

```
--Extract the exterior ring from each island and find its length.
SELECT SUM (st_length (st_exteriorring (land)))
 FROM islands;

sum

264.721359549996
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_GeomCollection

Nota:
Solo Oracle y PostgreSQL

Definición

ST_GeomCollection construye un conjunto de geometría a partir de una representación de texto conocido.

Sintaxis

Oracle

```
sde.st_multilinestring (wkt clob, srid integer)
sde.st_multipoint (wkt clob, srid integer)
sde.st_multipolygon (wkt clob, srid integer)
```

PostgreSQL

```
sde.st_multilinestring (wkt, srid integer)
sde.st_multilinestring (esri_shape bytea, srid integer)
sde.st_multipoint (wkt, srid integer)
sde.st_multipoint (esri_shape bytea, srid integer)
sde.st_multipolygon (wkt, srid integer)
sde.st_multipolygon (esri_shape bytea, srid integer)
```

Tipo de devolución

ST_GeomCollection

Ejemplo

Oracle

Cree una tabla, geomcoll_test, e inserte geometrías en ella.

```
CREATE TABLE geomcoll_test (id integer, geometry sde.st_geometry);

INSERT INTO geomcoll_test (id, geometry) VALUES (
1901,
sde.st_multipoint ('multipoint ((1 2), (4 3), (5 6))', 0)
);

INSERT INTO geomcoll_test (id, geometry) VALUES (
1902,
sde.st_multilinestring ('multilinestring ((33 2, 34 3, 35 6),
(28 4, 29 5, 31 8, 43 12), (39 3, 37 4, 36 7))', 0)
);

INSERT INTO geomcoll_test (id, geometry) VALUES (
1903,
sde.st_multipolygon ('multipolygon (((3 3, 4 6, 5 3, 3 3),
(8 24, 9 25, 1 28, 8 24), (13 33, 7 36, 1 40, 10 43, 13 33)))', 0)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

Seleccione el conjunto de geometría en la tabla geomcoll_test.

```
SELECT id, sde.st_astext (geometry) Geomcollection
FROM GEOMCOLL_TEST;

        ID     GEOMCOLLECTION

      1901     MULTIPOINT ((1.00000000 2.00000000), (4.00000000 3.00000000),
                      (5.00000000 6.00000000))

      1902     MULTILINESTRING ((33.00000000 2.00000000, 34.00000000
                  3.00000000, 35.00000000 6.00000000),(28.00000000 4.00000000,
                  29.00000000 5.00000000, 31.00000000 8.00000000, 43.00000000
                   12.00000000),(39.00000000 3.00000000, 37.00000000
                   4.00000000, 36.00000000 7.00000000))

      1903     MULTIPOLYGON (((13.00000000 33.00000000, 10.00000000
                  43.00000000, 1.00000000 40.00000000, 7.00000000 36.00000000,
                   13.00000000 33.00000000)),((8.00000000 24.00000000,
                   9.00000000 25.00000000, 1.00000000 28.00000000, 8.00000000
                   24.00000000)), ((3.00000000 3.00000000,5.00000000
                   3.00000000, 4.00000000 6.00000000,3.00000000 3.00000000)))
```

PostgreSQL

Cree una tabla, geomcoll_test, e inserte geometrías en ella.

```
CREATE TABLE geomcoll_test (id integer, geometry sde.st_geometry);

INSERT INTO geomcoll_test (id, geometry) VALUES (
1901,
sde.st_multipoint ('multipoint (1 2, 4 3, 5 6)', 0)
);

INSERT INTO geomcoll_test (id, geometry) VALUES (
1902,
sde.st_multilinestring ('multilinestring ((33 2, 34 3, 35 6),
(28 4, 29 5, 31 8, 43 12), (39 3, 37 4, 36 7))', 0)
);

INSERT INTO geomcoll_test (id, geometry) VALUES (
1903,
sde.st_multipolygon ('multipolygon (((3 3, 4 6, 5 3, 3 3),
(8 24, 9 25, 1 28, 8 24), (13 33, 7 36, 1 40, 10 43, 13 33)))', 0)
);
```

Seleccione el conjunto de geometría en la tabla geomcoll_test.

```
SELECT id, sde.st_astext (geometry)
AS geomcollection
FROM geomcoll_test;

        id     geomcollection

1901     MULTIPOINT (1 2, 4 3, 5 6)

1902     MULTILINESTRING ((33 2, 34        3, 35 6),(28 4,
29 5, 31 8, 43 12),(39 3, 37 4, 36 7))
```

Copyright © 1995-2023 Esri. All rights reserved.

```
1903     MULTIPOLYGON (((13 33, 10 43, 1 40, 7 36,
13 33)),((8 24, 9 25, 1 28, 8 24)), 3 3, 5 3, 4 6, 3 3)))
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_GeomCollFromWKB

Nota:
Solo PostgreSQL

Definición

ST_GeomCollFromWKB construye una colección de geometría desde una representación binaria conocida.

Sintaxis

```
sde.st_geomcollfromwkb (wkb bytea, srid integer)
```

Tipo de devolución

ST_GeomCollection

Ejemplo

Nota:
Se han introducido saltos de carro para ofrecer mayor legibilidad. Elimínelos si copia las declaraciones.

Cree una tabla, gcoll_test.

```
CREATE TABLE gcoll_test (pkey integer, shape sde.st_geomcollection );
```

Introduzca valores en la tabla.

```
INSERT INTO gcoll_test VALUES
(1,
st_geomcollfromwkb (sde.st_asbinary(sde.st_geomcollection
('multipoint(20 20, 30 30, 20 40, 30 50)', 0)), 0));

INSERT INTO gcoll_test VALUES
(2,
sde.st_geomcollfromwkb (sde.st_asbinary(sde.st_geomcollection
('multilinestring ((10.02 20.01, 10.32 23.98, 11.92 25.64),
(9.55 23.75,15.36 30.11),(10 10,20 20,30 30,40 40, 90 90))', 0)), 0));

INSERT INTO gcoll_test VALUES
(3,
sde.st_geomcollfromwkb (sde.st_asbinary(sde.st_geomcollection
('multipolygon(((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)),
((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)))', 0)), 0));
```

Seleccione la geometría de la tabla gcoll_test.

```
SELECT pkey, sde.st_astext(shape) from gcoll_test;

 pkey   st_astext
```

Copyright © 1995-2023 Esri. All rights reserved.

```
   1          MULTIPOINT ( 20 20, 30 30, 20 40, 30 50)

   3           MULTIPOLYGON ((( 0 0, 4 0, 4 4, 0 4, 0 0), (1 1, 1 2,
2 2, 2 1, 1 1)), ((-1 -1, -2 -1, -2 -2, -1 -2, -1 -1))
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Geometry

Definición

ST_Geometry construye una geometría a partir de una representación de texto conocido.

Nota:
Cuando se crean tablas espaciales que se utilizarán con ArcGIS, es mejor crear la columna como supertipo de geometría (por ejemplo, ST_Geometry) en lugar de especificar un subtipo ST_Geometry.

Sintaxis

Oracle

- Para cadenas de líneas, polígonos y puntos

```
sde.st_geometry (wkt clob, srid integer)
```

- Para puntos optimizados (que no lanzan un agente extproc y, por lo tanto, procesan la consulta más rápidamente)

```
sde.st_geometry (x, y, z, m, srid)
```

  Utilice la construcción de puntos optimizados cuando realice inserciones por lotes de grandes cantidades de datos de puntos.

- Para círculos paramétricos

```
sde.st_geometry (x, y, z, m, radius, number_of_points, srid)
```

- Para elipses paramétricas

```
sde.st_geometry (x, y, z, m, semi_major_axis, semi_minor_axis, angle,
number_of_points, srid)
```

- Para cuñas paramétricas

```
sde.st_geometry (x, y, z, m, startangle, endangle, outerradius, innerradius,
number_of_points, srid)
```

PostgreSQL

- Para cadenas de líneas, polígonos y puntos

```
sde.st_geometry (wkt, srid integer)
sde.st_geometry (esri_shape bytea, srid integer)
```

Copyright © 1995-2023 Esri. All rights reserved.

- Para círculos paramétricos

```
sde.st_geometry (x, y, z, m, radius, number_of_points, srid)
```

- Para elipses paramétricas

```
sde.st_geometry (x, y, z, m, semi_major_axis, semi_minor_axis, angle,
number_of_points, srid)
```

- Para cuñas paramétricas

```
sde.st_geometry (x, y, z, m, startangle, endangle, outerradius, innerradius,
number_of_points, srid)
```

SQLite

- Para cadenas de líneas, polígonos y puntos

```
st_geometry (text  WKT_string,int32 srid)
```

- Para círculos paramétricos

```
st_geometry (x, y, z, m, radius, number_of_points, srid)
```

- Para elipses paramétricas

```
st_geometry (x, y, z, m, semi_major_axis, semi_minor_axis, angle_of_rotation,
number_of_points, srid)
```

- Para cuñas paramétricas

```
st_geometry (x, y, z, m, start_angle, end_angle, outer_radius, inner_radius,
number_of_points, srid)
```

Tipo de devolución

Oracle y PostgreSQL

ST_Geometry

SQLite

Geometryblob

Ejemplos

Crear y consultar entidades de punto, de cadena de líneas y poligonales

Estos ejemplos crean una tabla (geoms) e insertan en ella valores de punto, de cadena de líneas y poligonales.

Copyright © 1995-2023 Esri. All rights reserved.

Oracle

```
CREATE TABLE geoms (
 id integer,
 geometry sde.st_geometry
);
```

```
INSERT INTO GEOMS (id, geometry) VALUES (
 1901,
 sde.st_geometry ('point (1 2)', 4326)
);

--To insert the same point using optimized point construction:
INSERT INTO GEOMS (id, geometry) VALUES (
 1901,
 sde.st_geometry (1,2,null,null,4326)
);

INSERT INTO GEOMS (id, geometry) VALUES (
 1902,
 sde.st_geometry ('linestring (33 2, 34 3, 35 6)', 4326)
);

INSERT INTO GEOMS (id, geometry) VALUES (
 1903,
 sde.st_geometry ('polygon ((3 3, 4 6, 5 3, 3 3))', 4326)
);
```

PostgreSQL

```
CREATE TABLE geoms (
 id serial,
 geometry sde.st_geometry
);
```

```
INSERT INTO geoms (geometry) VALUES (
 sde.st_geometry ('point (1 2)', 4326)
);

INSERT INTO geoms (geometry) VALUES (
 sde.st_geometry ('linestring (33 2, 34 3, 35 6)', 4326)
);

INSERT INTO geoms (geometry) VALUES (
 sde.st_geometry ('polygon ((3 3, 4 6, 5 3, 3 3))', 4326)
);
```

SQLite

```
CREATE TABLE geoms (
 id integer primary key autoincrement not null
);

SELECT AddGeometryColumn (
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 NULL,
 'geoms',
 'geometry',
 4326,
 'geometry',
 'xy',
 'null'
);
```

```
INSERT INTO geoms (geometry) VALUES (
 st_geometry ('point (1 2)', 4326)
);

INSERT INTO geoms (geometry) VALUES (
 st_geometry ('linestring (33 2, 34 3, 35 6)', 4326)
);

INSERT INTO geoms (geometry) VALUES (
 st_geometry ('polygon ((3 3, 4 6, 5 3, 3 3))', 4326)
);
```

Crear y consultar círculos paramétricos

Cree una tabla, radii, e inserte círculos en ella.

Oracle

```
CREATE TABLE radii (
 id integer,
 geometry sde.st_geometry
);
```

```
INSERT INTO RADII (id, geometry) VALUES (
 1904,
 sde.st_geometry (10,10,NULL,NULL,25,50,4326)
);

INSERT INTO RADII (id, geometry) VALUES (
 1905,
 sde.st_geometry (5,15,NULL,NULL,10,20,4326)
);
```

PostgreSQL

```
CREATE TABLE radii (
 id serial,
 geometry sde.st_geometry
);
```

```
INSERT INTO radii (geometry) VALUES (
 sde.st_geometry (10,10,NULL,NULL,25,50,4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO radii (geometry) VALUES (
 sde.st_geometry (5,15,NULL,20,10,30,4326)
);
```

SQLite

```
CREATE TABLE radii (
 id integer primary key autoincrement not null
);

SELECT AddGeometryColumn (
 NULL,
 'radii',
 'geometry',
 4326,
 'geometry',
 'xy',
 'null'
);
```

```
INSERT INTO radii (geometry) VALUES (
 st_geometry (10,10,NULL,NULL,25,50,4326)
);

INSERT INTO radii (geometry) VALUES (
 st_geometry (5,15,NULL,20,10,30,4326)
);
```

Crear y consultar elipses paramétricas

Cree una tabla, track, e inserte elipses en ella.

Oracle

```
CREATE TABLE track (
 id integer,
 geometry sde.st_geometry
);
```

```
INSERT INTO TRACK (id, geometry) VALUES (
 1907,
 sde.st_geometry (0,0,NULL,NULL,10,5,0,50,4326)
);

INSERT INTO TRACK (id, geometry) VALUES (
 1908,
 sde.st_geometry (4,19,10,20,10,5,0,40,4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

PostgreSQL

```
CREATE TABLE track (
 id serial,
 geometry sde.st_geometry
);
```

```
INSERT INTO track (geometry) VALUES (
 sde.st_geometry (0,0,NULL,NULL,10,5,0,50,4326)
);

INSERT INTO track (geometry) VALUES (
 sde.st_geometry (4,19,10,20,10,5,0,40,4326)
);
```

SQLite

```
CREATE TABLE track (
 id integer primary key autoincrement not null
);

SELECT AddGeometryColumn (
 NULL,
 'track',
 'geometry',
 4326,
 'geometry',
 'xy',
 'null'
);
```

```
INSERT INTO track (geometry) VALUES (
 st_geometry (0,0,NULL,NULL,10,5,0,50,4326)
);

INSERT INTO track (geometry) VALUES (
 st_geometry (4,19,10,20,10,5,0,40,4326)
);
```

Crear y consultar cuñas paramétricas

Cree una tabla, pwedge, e inserte una cuña en ella.

Oracle

```
CREATE TABLE pwedge (
 id integer,
 label varchar2(8),
 shape sde.st_geometry
);
```

```
INSERT INTO PWEDGE (id, label, shape) VALUES (
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 1,
 'Wedge1',
 sde.st_geometry (10,30,NULL,NULL,45,145,5,2,60,4326)
);
```

PostgreSQL

```
CREATE TABLE pwedge (
 id serial,
 label varchar(8),
 shape sde.st_geometry
);
```

```
INSERT INTO pwedge (label, shape) VALUES (
 'Wedge',
 sde.st_geometry(10,30,NULL,NULL,45,145,5,2,60,4326)
);
```

SQLite

```
CREATE TABLE pwedge (
 id integer primary key autoincrement not null,
 label varchar(8)
);

SELECT AddGeometryColumn (
 NULL,
 'pwedge',
 'shape',
 4326,
 'geometry',
 'xy',
 'null'
);
```

```
INSERT INTO pwedge (label, shape) VALUES (
 'Wedge',
 st_geometry(10,30,NULL,NULL,45,145,5,2,60,4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_GeometryN

Definición

ST_GeometryN toma un conjunto y un índice de enteros y devuelve el enésimo objeto ST_Geometry en la colección.

Sintaxis

Oracle y PostgreSQL

```
sde.st_geometryn (mpt1 sde.st_multipoint, index integer)
sde.st_geometryn (mln1 sde.st_multilinestring, index integer)
sde.st_geometryn (mpl1 sde.st_multipolygon, index integer)
```

SQLite

```
st_geometryn (mpt1 st_multipoint, index integer)
st_geometryn (mln1 st_multilinestring, index integer)
st_geometryn (mpl1 st_multipolygon, index integer)
```

Tipo de devolución

Oracle y PostgreSQL

ST_Geometry

SQLite

Geometryblob

Ejemplo

En este ejemplo, se crea un multipolígono. Después se utiliza ST_GeometryN para enumerar el segundo elemento del multipolígono.

Oracle

```
CREATE TABLE districts (
 dist_id integer,
 shape sde.st_multipolygon
);

INSERT INTO DISTRICTS (dist_id, shape) VALUES (
 1,
 sde.st_multipolygon ('multipolygon (((-1 -1, -1 11, 11 11, 11 -1, -1 -1),
(19 -1, 19 11, 29 9, 31 -1, 19 -1), (39 -1, 39 11, 51 11, 51 -1, 39 -1)))', 4326)
);

SELECT sde.st_astext (sde.st_geometryn (shape, 2)) Second_Element
 FROM DISTRICTS;

Second_Element

POLYGON  ((-1.00000000 -1.00000000, 11.00000000 -1.00000000, 11.0000000 0 11.000
```

Copyright © 1995-2023 Esri. All rights reserved.

PostgreSQL

```
CREATE TABLE districts (
 dist_id serial,
 shape sde.st_geometry
);

INSERT INTO districts (shape) VALUES (
 sde.st_multipolygon ('multipolygon (((-1 -1, -1 11, 11 11, 11 -1, -1 -1),
(19 -1, 19 11, 29 9, 31 -1, 19 -1), (39 -1, 39 11, 51 11, 51 -1, 39 -1)))', 4326)
);

SELECT sde.st_astext (sde.st_geometryn (shape, 2)) AS Second_Element
 FROM districts;

second_element

POLYGON  ((39 -1, 51 -1, 51 11, 39 11, 39 -1))
```

SQLite

```
CREATE TABLE districts (
 dist_id integer primary key autoincrement not null
);

SELECT AddGeometryColumn (
 NULL,
 'districts',
 'shape',
 4326,
 'multipolygon',
 'xy',
 'null'
);

INSERT INTO districts (shape) VALUES (
 st_multipolygon ('multipolygon (((-1 -1, -1 11, 11 11, 11 -1, -1 -1),
(19 -1, 19 11, 29 9, 31 -1, 19 -1), (39 -1, 39 11, 51 11, 51 -1, 39 -1)))', 4326)
);

SELECT st_astext (st_geometryn (shape, 2))
 AS "Second_Element"
 FROM districts;

Second_Element

POLYGON  ((39.00000000 -1.00000000, 51.00000000 -1.00000000, 51.00000000 11.00000000,
39.00000000 11.00000000, 39.00000000 -1.00000000))
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_GeometryType

Definición

ST_GeometryType toma un objeto de geometría y devuelve su tipo de geometría (por ejemplo, punto, línea, polígono, multipunto) como cadena.

Sintaxis

Oracle y PostgreSQL

```
sde.st_geometrytype (g1 sde.st_geometry)
```

SQLite

```
st_geometrytype (g1 geometryblob)
```

Tipo de devolución

Varchar(32) (Oracle y PostgreSQL) o texto (SQLite) que contenga uno de los siguientes:

- ST_Point
- ST_LineString
- ST_Polygon
- ST_MultiPoint
- ST_MultiLineString
- ST_MultiPolygon

Ejemplo

La tabla geometrictype_test contiene la columna de geometría g1.

Las sentencias INSERT insertan cada subclase de geometría en la columna g1.

La consulta SELECT enumera el tipo de geometría de cada subclase almacenada en la columna de geometría g1.

Oracle

```
CREATE TABLE geometrytype_test (g1 sde.st_geometry);

INSERT INTO geometrytype_test VALUES (
 sde.st_geometry ('point (10.02 20.01)', 4326)
);

INSERT INTO geometrytype_test VALUES (
 sde.st_geometry ('linestring (10.01 20.01, 10.01 30.01, 10.01 40.01)', 4326)
);

INSERT INTO geometrytype_test VALUES (
 sde.st_geometry ('polygon ((10.02 20.01, 11.92 35.64, 25.02 34.15,
19.15 33.94, 10.02 20.01))', 4326)
```

Copyright © 1995-2023 Esri. All rights reserved.

```
);

INSERT INTO geometrytype_test VALUES (
 sde.st_geometry ('multipoint ((10.02 20.01), (10.32 23.98), (11.92 25.64))', 4326)
);

INSERT INTO geometrytype_test VALUES (
 sde.st_geometry ('multilinestring ((10.02 20.01, 10.32 23.98,
11.92 25.64), (9.55 23.75, 15.36 30.11))', 4326)
);

INSERT INTO geometrytype_test VALUES (
 sde.st_geometry ('multipolygon (((10.02 20.01, 11.92 35.64, 25.02 34.15,
19.15 33.94, 10.02 20.01)), ((51.71 21.73, 73.36 27.04, 71.52 32.87,
52.43 31.90,51.71 21.73)))', 4326)
);
```

```
SELECT UPPER (sde.st_geometrytype (g1)) Geometry_type
FROM GEOMETRYTYPE_TEST;

Geometry_type

ST_POINT
ST_LINESTRING
ST_POLYGON
ST_MULTIPOINT
ST_MULTILINESTRING
ST_MULTIPOLYGON
```

PostgreSQL

```
CREATE TABLE geometrytype_test (g1 sde.st_geometry);

INSERT INTO geometrytype_test VALUES (
 sde.st_geometry ('point (10.02 20.01)', 4326)
);

INSERT INTO geometrytype_test VALUES (
 sde.st_geometry ('linestring (10.01 20.01, 10.01 30.01, 10.01 40.01)', 4326)
);

INSERT INTO geometrytype_test VALUES (
 sde.st_geometry ('polygon ((10.02 20.01, 11.92 35.64, 25.02 34.15,
19.15 33.94, 10.02 20.01))', 4326)
);

INSERT INTO geometrytype_test VALUES (
 sde.st_geometry ('multipoint (10.02 20.01, 10.32 23.98, 11.92 25.64)', 4326)
);

INSERT INTO geometrytype_test VALUES (
 sde.st_geometry ('multilinestring ((10.02 20.01, 10.32 23.98,
11.92 25.64), (9.55 23.75, 15.36 30.11))', 4326)
);

INSERT INTO geometrytype_test VALUES (
 sde.st_geometry ('multipolygon (((10.02 20.01, 11.92 35.64, 25.02 34.15,
19.15 33.94, 10.02 20.01)), ((51.71 21.73, 73.36 27.04, 71.52 32.87,
52.43 31.90,51.71 21.73)))', 4326)
```

Copyright © 1995-2023 Esri. All rights reserved.

```
);
```

```
SELECT (sde.st_geometrytype (g1))
 AS Geometry_type
 FROM geometrytype_test;

Geometry_type

ST_POINT
ST_LINESTRING
ST_POLYGON
ST_MULTIPOINT
ST_MULTILINESTRING
ST_MULTIPOLYGON
```

Copyright © 1995-2023 Esri. All rights reserved.

SQLite

```
CREATE TABLE geometrytype_test (id integer primary key autoincrement not null);

SELECT AddGeometryColumn (
 NULL,
 'geometrytype_test',
 'g1',
 4326,
 'geometry',
 'xy',
 'null'
);

INSERT INTO geometrytype_test (g1) VALUES (
 st_geometry ('point (10.02 20.01)', 4326)
);

INSERT INTO geometrytype_test (g1) VALUES (
 st_geometry ('linestring (10.01 20.01, 10.01 30.01, 10.01 40.01)', 4326)
);

INSERT INTO geometrytype_test (g1) VALUES (
 st_geometry ('polygon ((10.02 20.01, 11.92 35.64, 25.02 34.15,
19.15 33.94, 10.02 20.01))', 4326)
);

INSERT INTO geometrytype_test (g1) VALUES (
 st_geometry ('multipoint ((10.02 20.01), (10.32 23.98), (11.92 25.64))', 4326)
);

INSERT INTO geometrytype_test (g1) VALUES (
 st_geometry ('multilinestring ((10.02 20.01, 10.32 23.98,
11.92 25.64), (9.55 23.75, 15.36 30.11))', 4326)
);

INSERT INTO geometrytype_test (g1) VALUES (
 st_geometry ('multipolygon (((10.02 20.01, 11.92 35.64, 25.02 34.15,
19.15 33.94, 10.02 20.01)), ((51.71 21.73, 73.36 27.04, 71.52 32.87,
52.43 31.90,51.71 21.73)))', 4326)
);
```

```
SELECT (st_geometrytype (g1))
 AS "Geometry_type"
 FROM geometrytype_test;

Geometry_type

ST_POINT
ST_LINESTRING
ST_POLYGON
ST_MULTIPOINT
ST_MULTILINESTRING
ST_MULTIPOLYGON
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_GeomFromCollection

Nota:
Solo PostgreSQL

Definición

ST_GeomFromCollection devuelve un conjunto de filas de st_geometry. Cada fila consta de una geometría y un entero. El entero representa la posición de la geometría en el conjunto.

Use la función ST_GeomFromCollection para acceder a cada geometría individual de una geometría multiparte. Si la geometría de entrada es una colección o una geometría multiparte (por ejemplo, ST_MultiLineString, ST_MultiPoint, ST_MultiPolygon), ST_GeomFromCollection devuelve un registro para cada uno de los componentes de la colección y la ruta expresa la posición del componente en la colección.

Si usa ST_GeomFromCollection con una geometría simple (por ejemplo, ST_Point, ST_LineString, ST_Polygon), se devuelve un solo registro con una ruta vacía, dado que solo existe una geometría.

Sintaxis

`sde.st_geomfromcollection (shape sde.st_geometry)`

Para devolver solo la geometría, use `(sde.st_geomfromcollection (shape)).st_geo`.

Para devolver solo la posición de la geometría, use `(sde.st_geomfromcollection (shape)).path[1]`.

Tipo de devolución

Conjunto ST_Geometry

Ejemplo

En este ejemplo, cree una clase de entidad multilínea (ghanasharktracks) que contiene una sola entidad con una forma de cuatro partes.

```
--Create the feature class.
CREATE TABLE ghanasharktracks (objectid integer, shape sde.st_geometry);
--Insert a multiline with four parts using SRID 4326.
INSERT INTO ghanasharktracks VALUES
 (1,
  sde.st_geometry('MULTILINESTRING Z (( 1 1 0, 1 6 0),(1 3 0, 3 3 0),(3 1 0, 3 3 0), (4
1 0, 4 6 0))',
  4326
  )
 );
```

Para confirmar que el campo contiene datos, ejecute una consulta con la tabla. Use ST_AsText directamente en el campo de forma para ver las coordenadas de la forma como texto. Observe que se devuelve la descripción de texto de la cadena de texto multilínea.

```
--View inserted feature. SELECT gst_orig.objectid, sde.st_astext(gst_orig.shape)
shapetext FROM ghanasharktracks gst_orig;
shapetext
-------------------------------
"MULTILINESTRING Z (( 1.00000000 1.00000000 0.00000000, 1.00000000 6.00000000
```

Copyright © 1995-2023 Esri. All rights reserved.

```
0.00000000),(1.00000000 3.00000000 0.00000000, 3.00000000 3.00000000
0.00000000),(3.00000000 1.00000000 0.00000000, 3.00000000 3.00000000 0.00000000),
(4.00000000 1.00000000 0.00000000, 4.00000000 6.00000000 0.00000000))"
```

Para devolver individualmente cada geometría de la cadena de texto de línea, use la función ST_GeomFromCollection. Para ver la geometría como texto, este ejemplo usa la función ST_AsText con la función ST_GeomFromCollection.

```
--Return each linestring in the multilinestring
SELECT sde.st_astext((sde.st_geomfromcollection(gst.shape)).st_geo) shapetext,
((sde.st_geomfromcollection(gst.shape)).path[1]) path FROM ghanasharktracks gst;
shapetext
      path
-------------------------------------------------------------------------------------
-------------------
"LINESTRING Z ( 1.00000000 1.00000000 0.00000000, 1.00000000 6.00000000
0.00000000)"          1
"LINESTRING Z ( 1.00000000 3.00000000 0.00000000, 3.00000000 3.00000000
0.00000000)"          2
"LINESTRING Z ( 3.00000000 1.00000000 0.00000000, 3.00000000 3.00000000
0.00000000)"          3
"LINESTRING Z ( 4.00000000 1.00000000 0.00000000, 4.00000000 6.00000000
0.00000000)"          4
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_GeomFromText

Nota:
Solo se utiliza en Oracle y SQLite; para PostgreSQL, utilice ST_Geometry.

Definición

ST_GeomFromText toma una representación de texto conocido y un Id. de referencia espacial y devuelve un objeto de geometría.

Sintaxis

Oracle

```
sde.st_geomfromtext (wkt clob, srid integer)
```

```
sde.st_geomfromtext (wkt clob)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

SQLite

```
st_geomfromtext (wkt text, srid int32)
```

```
st_geomfromtext (wkt text)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

Tipo de devolución

Oracle

ST_Geometry

SQLite

Geometryblob

Ejemplo

La tabla geometry_test contiene la columna gid de entero, que identifica unívocamente cada fila, y la columna g1, que almacena la geometría.

Las declaraciones INSERT introducen datos en las columnas gid y g1 de la tabla geometry_test. La función ST_GeomFromText convierte la representación de texto de cada geometría en su subclase representable correspondiente. La declaración SELECT al final se realiza para garantizar que los datos se introducen en la columna g1.

Copyright © 1995-2023 Esri. All rights reserved.

Oracle

```
CREATE TABLE geometry_test (
 gid smallint unique,
 g1 sde.st_geometry
);
```

```
INSERT INTO GEOMETRY_TEST VALUES (
 1,
 sde.st_geomfromtext ('point (10.02 20.01)', 4326)
);

INSERT INTO GEOMETRY_TEST VALUES (
 2,
 sde.st_geomfromtext('linestring (10.01 20.01, 10.01 30.01, 10.01 40.01)', 4326)
);

INSERT INTO GEOMETRY_TEST VALUES (
 3,
 sde.st_geomfromtext('polygon ((10.02 20.01, 11.92 35.64, 25.02 34.15,
19.15 33.94, 10.02 20.01))', 4326)
);

INSERT INTO GEOMETRY_TEST VALUES (
 4,
 sde.st_geomfromtext('multipoint ((10.02 20.01), (10.32 23.98), (11.92 25.64))', 4326)
);

INSERT INTO GEOMETRY_TEST VALUES (
 5,
 sde.st_geomfromtext ('multilinestring ((10.02 20.01, 10.32 23.98,
11.92 25.64), (9.55 23.75, 15.36 30.11))', 4326)
);

INSERT INTO GEOMETRY_TEST VALUES (
 6,
 sde.st_geomfromtext ('multipolygon (((10.02 20.01, 11.92 35.64,
25.02 34.15, 19.15 33.94, 10.02 20.01), (51.71 21.73, 73.36 27.04,
71.52 32.87, 52.43 31.90, 51.71 21.73)))', 4326)
);

SELECT sde.st_astext(g1)
 FROM GEOMETRY_TEST;

POINT ( 10.02000000 20.01000000)
LINESTRING ( 10.01000000 20.01000000, 10.01000000 30.01000000, 10.01000000 40.01000000)
POLYGON (( 10.02000000 20.01000000, 19.15000000 33.94000000, 25.02000000 34.15000000,
11.92000000 35.64000000, 10.02000000 20.01000000))
MULTIPOINT (( 10.02000000 20.01000000), (10.32000000 23.98000000), (11.92000000
25.64000000))
MULTILINESTRING (( 10.02000000 20.01000000, 10.32000000 23.98000000, 11.92000000
25.64000000),( 9.55000000 23.75000000, 15.36000000 30.11000000))
MULTIPOLYGON ((( 51.71000000 21.73000000, 73.36000000 27.04000000, 71.52000000
32.87000000, 52.43000000 31.90000000, 51.71000000 21.73000000)),(( 10.02000000
20.01000000, 19.15000000 33.94000000, 25.02000000 34.15000000, 11.92000000 35.64000000,
10.02000000 20.01000000)))
```

Copyright © 1995-2023 Esri. All rights reserved.

SQLite

```
CREATE TABLE geometry_test (
 gid integer primary key autoincrement not null
);

SELECT AddGeometryColumn (
 NULL,
 'geometry_test',
 'g1',
 4326,
 'geometry',
 'xy',
 'null'
);
```

```
INSERT INTO GEOMETRY_TEST (g1) VALUES (
 st_geomfromtext ('point (10.02 20.01)', 4326)
);

INSERT INTO GEOMETRY_TEST (g1) VALUES (
 st_geomfromtext('linestring (10.01 20.01, 10.01 30.01, 10.01 40.01)', 4326)
);

INSERT INTO GEOMETRY_TEST (g1) VALUES (
 st_geomfromtext('polygon ((10.02 20.01, 11.92 35.64, 25.02 34.15,
19.15 33.94, 10.02 20.01))', 4326)
);

INSERT INTO GEOMETRY_TEST (g1) VALUES (
 st_geomfromtext('multipoint ((10.02 20.01), (10.32 23.98), (11.92 25.64))', 4326)
);

INSERT INTO GEOMETRY_TEST (g1) VALUES (
 st_geomfromtext ('multilinestring ((10.02 20.01, 10.32 23.98,
11.92 25.64), (9.55 23.75, 15.36 30.11))', 4326)
);

INSERT INTO GEOMETRY_TEST (g1) VALUES (
 st_geomfromtext ('multipolygon (((10.02 20.01, 11.92 35.64,
25.02 34.15, 19.15 33.94, 10.02 20.01), (51.71 21.73, 73.36 27.04,
71.52 32.87, 52.43 31.90, 51.71 21.73)))', 4326)
);

SELECT st_astext(g1)
 FROM geometry_test;

POINT (10.02000000 20.01000000)
LINESTRING (10.01000000 20.01000000, 10.01000000 30.01000000, 10.01000000 40.01000000)
POLYGON ((10.02000000 20.01000000, 19.15000000 33.94000000, 25.02000000 34.15000000,
11.92000000 35.64000000, 10.02000000 20.01000000))
MULTIPOINT ((10.02000000 20.01000000), (10.32000000 23.98000000), (11.92000000
25.64000000))
MULTILINESTRING ((10.02000000 20.01000000, 10.32000000 23.98000000, 11.92000000
25.64000000),(9.55000000 23.75000000, 15.36000000 30.11000000))
MULTIPOLYGON (((51.71000000 21.73000000, 73.36000000 27.04000000, 71.52000000
32.87000000, 52.43000000 31.90000000, 51.71000000 21.73000000)),((10.02000000
20.01000000, 19.15000000 33.94000000, 25.02000000 34.15000000, 11.92000000 35.64000000,
10.02000000 20.01000000)))
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_GeomFromWKB

Definición

ST_GeomFromWKB toma una representación binaria conocida (WKB) y un Id. de referencia espacial para devolver un objeto de geometría.

Sintaxis

Oracle

```
sde.st_geomfromwkb (wkb blob, srid integer)
```

```
sde.st_geomfromwkb (wkb blob)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

PostgreSQL

```
sde.st_geomfromwkb (wkb, srid integer)
```

```
sde.st_geomfromwkb (esri_shape bytea, srid integer)
```

SQLite

```
st_geomfromwkb (wkb blob, srid int32)
```

```
st_geomfromwkb (wkb blob)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

Tipo de devolución

Oracle y PostgreSQL

ST_Geometry

SQLite

Geometryblob

Ejemplo

En el siguiente ejemplo, las líneas de los resultados se han reformateado para su legibilidad. El espaciado en sus resultados variará según su visualización en línea. El siguiente código ilustra cómo la función ST_GeomFromWKB se

Copyright © 1995-2023 Esri. All rights reserved.

puede utilizar para crear e insertar una línea desde una representación de línea WKB. En el siguiente ejemplo, se introduce un registro en la tabla sample_gs con un Id. y una geometría en el sistema de referencia espacial 4326 en una representación WKB.

Oracle

```
CREATE TABLE sample_gs (
 id integer,
 geometry sde.st_geometry,
 wkb blob
);
INSERT INTO sample_gs (id, geometry) VALUES (
 1901,
 sde.st_geomfromtext ('point (1 2)', 4326)
);
INSERT INTO sample_gs (id, geometry) VALUES (
 1902,
 sde.st_geomfromtext ('linestring (33 2, 34 3, 35 6)', 4326)
);
INSERT INTO sample_gs (id, geometry) VALUES (
 1903,
 sde.st_geomfromtext ('polygon ((3 3, 4 6, 5 3, 3 3))', 4326)
);

UPDATE sample_gs
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 1901;
UPDATE sample_gs
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 1902;
UPDATE sample_gs
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 1903;
SELECT id, sde.st_astext (sde.st_geomfromwkb (wkb, 4326))
 FROM sample_gs;
ID   GEOMETRY
1901 POINT (1.00000000 2.00000000)
1902 LINESTRING (33.00000000 2.00000000, 34.00000000 3.00000000, 35.00000000
6.00000000)
1903 POLYGON ((3.00000000 3.00000000, 5.00000000 3.00000000, 4.00000000 6.00000000,
3.00000000 3.00000000))
```

PostgreSQL

```
CREATE TABLE sample_gs (
 id integer,
 geometry sde.st_geometry,
 wkb bytea);
INSERT INTO sample_gs (id, geometry) VALUES (
 1901,
 sde.st_geometry ('point (1 2)', 4326)
);
INSERT INTO sample_gs (id, geometry) VALUES (
 1902,
 sde.st_geometry ('linestring (33 2, 34 3, 35 6)', 4326)
);
INSERT INTO sample_gs (id, geometry) VALUES (
 1903,
 sde.st_geometry ('polygon ((3 3, 4 6, 5 3, 3 3))', 4326)
```

Copyright © 1995-2023 Esri. All rights reserved.

```
);

UPDATE sample_gs
 SET wkb = sde.st_asshape (geometry)
 WHERE id = 1901;
UPDATE sample_gs
 SET wkb = sde.st_asshape (geometry)
 WHERE id = 1902;
UPDATE sample_gs
 SET wkb = sde.st_asshape (geometry)
 WHERE id = 1903;
SELECT id, sde.st_astext (sde.st_geomfromshape (wkb, 4326))
 FROM sample_gs;
id   st_astext
1901 POINT (1 2)
1902 LINESTRING (33 2, 34 3, 35 6)
1903 POLYGON ((3 3, 5 3, 4 6, 3 3))
```

SQLite

```
CREATE TABLE sample_gs (
 id integer primary key autoincrement not null,
 wkb blob
);
SELECT AddGeometryColumn (
 NULL,
 'sample_gs',
 'geometry',
 4326,
 'geometry',
 'xy',
 'null'
);
INSERT INTO sample_gs (geometry) VALUES (
 st_geomfromtext ('point (1 2)', 4326)
);
INSERT INTO sample_gs (geometry) VALUES (
 st_geomfromtext ('linestring (33 2, 34 3, 35 6)', 4326)
);
INSERT INTO sample_gs (geometry) VALUES (
 st_geomfromtext ('polygon ((3 3, 4 6, 5 3, 3 3))', 4326)
);

--Replace IDs with actual values.
UPDATE sample_gs
 SET wkb = st_asbinary (geometry)
 WHERE id = 1;
UPDATE sample_gs
 SET wkb = st_asbinary (geometry)
 WHERE id = 2;
UPDATE sample_gs
 SET wkb = st_asbinary (geometry)
 WHERE id = 3;
SELECT id, st_astext (st_geomfromwkb (wkb, 4326))
 FROM sample_gs;
ID   GEOMETRY
1    POINT (1.00000000 2.00000000)
2    LINESTRING (33.00000000 2.00000000, 34.00000000 3.00000000, 35.00000000
6.00000000)
3    POLYGON ((3.00000000 3.00000000, 5.00000000 3.00000000, 4.00000000 6.00000000,
3.00000000 3.00000000))
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_GeoSize

Nota:
Solo PostgreSQL

Definición

ST_GeoSize toma un objeto ST_Geometry y devuelve su tamaño en bytes.

Sintaxis

```
st_geosize (st_geometry)
```

Tipo de devolución

Entero

Ejemplo

Puede descubrir el tamaño de las entidades creadas en el ejemplo ST_GeomFromWKB consultando la columna de geometría de la tabla sample_geometries.

```
SELECT st_geosize (geometry)  FROM sample_geometries;  st_geosize         512
592         616
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_InteriorRingN

Definición

ST_InteriorRingN devuelve el enésimo anillo interior de un polígono como ST_LineString.

El orden de los anillos no se puede definir previamente ya que los anillos se organizan según las reglas definidas por las rutinas de verificación de la geometría interior y no por la orientación geométrica. Si el índice supera el número de anillos interiores que contiene un polígono, se devuelve un valor nulo.

Sintaxis

Oracle

```
sde.st_interiorringn (polygon1 sde.st_polygon, INDEX integer)
```

PostgreSQL

```
sde.st_interiorringn (polygon1 sde.st_polygon, ring_number integer)
```

SQLite

```
st_interiorringn (polygon1 sde.st_polygon, ring_number int32)
```

Tipo de devolución

ST_LineString

Ejemplo

Cree una tabla, sample_polys, y agregue un registro. A continuación, seleccione el Id. y la geometría del anillo interior.

Oracle

```
CREATE TABLE sample_polys (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO sample_polys VALUES (
 1,
 sde.st_polygon ('polygon ((40 120, 90 120, 90 150, 40 150, 40 120), (50 130, 60 130,
60 140, 50 140, 50 130),
(70 130, 80 130, 80 140, 70 140, 70 130))', 4326)
);
```

```
SELECT id, sde.st_astext (sde.st_interiorringn (geometry, 2)) Interior_Ring
 FROM SAMPLE_POLYS;

ID INTERIOR_RING
```

Copyright © 1995-2023 Esri. All rights reserved.

```
1  LINESTRING (70.00000000 130.00000000, 70.00000000 140.00000000, 80.00000000
140.00000000, 80.00000000 130.00000000, 70.00000000 130.00000000)
```

PostgreSQL

```
CREATE TABLE sample_polys (
 id serial,
 geometry sde.st_geometry
);

INSERT INTO sample_polys (geometry) VALUES (
 sde.st_polygon ('polygon ((40 120, 90 120, 90 150, 40 150, 40 120), (50 130, 60 130,
60 140, 50 140, 50 130),
(70 130, 80 130, 80 140, 70 140, 70 130))', 4326)
);
```

```
SELECT id, sde.st_astext (st_interiorringn (geometry, 2))
 AS Interior_Ring
 FROM sample_polys;

id interior_ring

1  LINESTRING (70 130, 70 140, 80 140, 80 130, 70 130)
```

SQLite

```
CREATE TABLE sample_polys (
 id integer primary key autoincrement not null
);

SELECT AddGeometryColumn (
 NULL,
 'sample_polys',
 'geometry',
 4326,
 'polygon',
 'xy',
 'null'
);

INSERT INTO sample_polys (geometry) VALUES (
 st_polygon ('polygon ((40 120, 90 120, 90 150, 40 150, 40 120), (50 130, 60 130, 60
140, 50 140, 50 130),
(70 130, 80 130, 80 140, 70 140, 70 130))', 4326)
);
```

```
SELECT id, st_astext (st_interiorringn (geometry, 2))
 AS "Interior_Ring"
 FROM sample_polys;

id Interior_Ring

1  LINESTRING (70.00000000 130.00000000, 70.00000000 140.00000000, 80.00000000
140.00000000, 80.00000000 130.00000000, 70.00000000 130.00000000)
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Intersection

Definición

ST_Intersection toma dos objetos de geometría y devuelve el conjunto de intersección como un objeto de geometría bidimensional.

Sintaxis

Oracle y PostgreSQL

```
sde.st_intersection (geometry1 sde.st_geometry, geometry2 sde.st_geometry)
```

SQLite

```
st_intersection (geometry1 geometryblob, geometry2 geometryblob)
```

Tipo de devolución

Oracle y PostgreSQL

ST_Geometry

SQLite

Geometryblob

Ejemplo

El jefe de bomberos debe obtener las áreas de los hospitales, escuelas y centros de asistencia intersecados por el radio de una posible contaminación de los residuos peligrosos.

Los hospitales, las escuelas y los centros de asistencia se almacenan en la tabla population creada con la declaración CREATE TABLE que se muestra a continuación. La columna de forma, que se define como un polígono, almacena el contorno de cada una de las zonas sensibles.

Los emplazamientos peligrosos se almacenan en la tabla waste_sites creada con la declaración CREATE TABLE siguiente. La columna de sitio, que se define como un punto, almacena una ubicación que es el centro geográfico de cada sitio peligroso.

La función ST_Buffer genera una zona de influencia que rodea los sitios de residuos peligrosos. La función ST_Intersection genera polígonos desde la intersección de las áreas sensibles y sitios de residuos peligrosos de la zona de influencia.

Oracle

```
CREATE TABLE population (
 id integer,
 shape sde.st_geometry
);

CREATE TABLE waste_sites (
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 id integer,
 site sde.st_geometry
);


INSERT INTO population VALUES (
 1,
 sde.st_geometry ('polygon ((.20 .30, .30 .30, .30 .40, .20 .40, .20 .30))', 4326)
);

INSERT INTO population VALUES (
 2,
 sde.st_geometry ('polygon ((.30 .30, .30 .50, .50 .50, .50 .30, .30 .30))', 4326)
);

INSERT INTO population VALUES (
 3,
 sde.st_geometry ('polygon ((.40 .40, .40 .60, .60 .60, .60 .40, .40 .40))', 4326)
);

INSERT INTO waste_sites VALUES (
 40,
 sde.st_geometry ('point (.60 .60)', 4326)
);

INSERT INTO waste_sites VALUES (
 50,
 sde.st_geometry ('point (.30 .30)', 4326)
);
```

```
SELECT sa.id, sde.st_astext (sde.st_intersection (sde.st_buffer (hs.site, .1),
sa.shape)) Intersection
 FROM population sa, waste_sites hs
 WHERE hs.id = 50
 AND sde.st_astext (sde.st_intersection (sde.st_buffer (hs.site, .01), sa.shape))
 NOT LIKE '%EMPTY%';

  ID  INTERSECTION

1     POLYGON (( 0.29000000 0.30000000, 0.30000000 0.30000000, 0.30000000
0.31000000, 0.29934597 0.30997859, 0.29869474 0.30991445, 0.29804910 0.30980785,
 0.29741181 0.30965926, 0.29678561 0.30946930, 0.29617317 0.30923880, 0.29557711
 0.30896873, 0.29500000 0.30866025, 0.29444430 0.30831470, 0.29391239 0.30793353
, 0.29340654 0.30751840, 0.29292893 0.30707107, 0.29248160 0.30659346, 0.2920664
7 0.30608761, 0.29168530 0.30555570, 0.29133975 0.30500000, 0.29103127 0.3044228
9, 0.29076121 0.30382683, 0.29053070 0.30321440, 0.29034074 0.30258819, 0.290192
15 0.30195090, 0.29008555 0.30130526, 0.29002141 0.30065403, 0.29000000 0.300000
00))

2     POLYGON (( 0.30000000 0.30000000, 0.31000000 0.30000000, 0.30997859
0.30065403, 0.30991445 0.30130526, 0.30980785 0.30195090, 0.30965926 0.30258819,
 0.30946930 0.30321440, 0.30923880 0.30382683, 0.30896873 0.30442289, 0.30866025
 0.30500000, 0.30831470 0.30555570, 0.30793353 0.30608761, 0.30751840 0.30659346
, 0.30707107 0.30707107, 0.30659346 0.30751840, 0.30608761 0.30793353, 0.3055557
0 0.30831470, 0.30500000 0.30866025, 0.30442289 0.30896873, 0.30382683 0.3092388
0, 0.30321440 0.30946930, 0.30258819 0.30965926, 0.30195090 0.30980785, 0.301305
26 0.30991445, 0.30065403 0.30997859, 0.30000000 0.31000000, 0.30000000 0.300000
00))
```

Copyright © 1995-2023 Esri. All rights reserved.

PostgreSQL

```
CREATE TABLE population (
 id serial,
 shape sde.st_geometry
);

CREATE TABLE waste_sites (
 id serial,
 site sde.st_geometry
);


INSERT INTO population (shape) VALUES (
 sde.st_geometry ('polygon ((.20 .30, .30 .30, .30 .40, .20 .40, .20 .30))', 4326)
);

INSERT INTO population (shape) VALUES (
 sde.st_geometry ('polygon ((.30 .30, .30 .50, .50 .50, .50 .30, .30 .30))', 4326)
);

INSERT INTO population (shape) VALUES (
 sde.st_geometry ('polygon ((.40 .40, .40 .60, .60 .60, .60 .40, .40 .40))', 4326)
);

INSERT INTO waste_sites (site) VALUES (
 sde.st_geometry ('point (.60 .60)', 4326)
);

INSERT INTO waste_sites (site) VALUES (
 sde.st_geometry ('point (.30 .30)', 4326)
);
```

```
--Replace hs.id with ID value of second record in waste_sites table if not 2.
SELECT sa.id, sde.st_astext (sde.st_intersection (sde.st_buffer (hs.site, .01),
sa.shape))
 AS Intersection
 FROM population sa, waste_sites hs
 WHERE hs.id = 2
 AND sde.st_astext (sde.st_intersection (sde.st_buffer (hs.site, .1),
sa.shape))::varchar
 NOT LIKE '%EMPTY%';

  id  intersection

1      POLYGON (( 0.29000000 0.30000000, 0.30000000 0.30000000, 0.30000000
0.31000000, 0.29934597 0.30997859, 0.29869474 0.30991445, 0.29804910 0.30980785,
 0.29741181 0.30965926, 0.29678561 0.30946930, 0.29617317 0.30923880, 0.29557711
 0.30896873, 0.29500000 0.30866025, 0.29444430 0.30831470, 0.29391239 0.30793353
, 0.29340654 0.30751840, 0.29292893 0.30707107, 0.29248160 0.30659346, 0.2920664
7 0.30608761, 0.29168530 0.30555570, 0.29133975 0.30500000, 0.29103127 0.3044228
9, 0.29076121 0.30382683, 0.29053070 0.30321440, 0.29034074 0.30258819, 0.290192
15 0.30195090, 0.29008555 0.30130526, 0.29002141 0.30065403, 0.29000000 0.300000
00))

2      POLYGON (( 0.30000000 0.30000000, 0.31000000 0.30000000, 0.30997859
0.30065403, 0.30991445 0.30130526, 0.30980785 0.30195090, 0.30965926 0.30258819,
 0.30946930 0.30321440, 0.30923880 0.30382683, 0.30896873 0.30442289, 0.30866025
 0.30500000, 0.30831470 0.30555570, 0.30793353 0.30608761, 0.30751840 0.30659346
, 0.30707107 0.30707107, 0.30659346 0.30751840, 0.30608761 0.30793353, 0.3055557
0 0.30831470, 0.30500000 0.30866025, 0.30442289 0.30896873, 0.30382683 0.3092388
```

Copyright © 1995-2023 Esri. All rights reserved.

```
0, 0.30321440 0.30946930, 0.30258819 0.30965926, 0.30195090 0.30980785, 0.301305
26 0.30991445, 0.30065403 0.30997859, 0.30000000 0.31000000, 0.30000000 0.300000
00))
```

SQLite

```
CREATE TABLE population (
 id integer primary key autoincrement not null
);

SELECT AddGeometryColumn (
 NULL,
 'population',
 'shape',
 4326,
 'polygon',
 'xy',
 'null'
);

CREATE TABLE waste_sites (
 id integer primary key autoincrement not null
);

SELECT AddGeometryColumn (
 NULL,
 'waste_sites',
 'site',
 4326,
 'point',
 'xy',
 'null'
);

INSERT INTO population (shape) VALUES (
 st_geometry ('polygon ((.20 .30, .30 .30, .30 .40, .20 .40, .20 .30))', 4326)
);

INSERT INTO population (shape) VALUES (
 st_geometry ('polygon ((.30 .30, .30 .50, .50 .50, .50 .30, .30 .30))', 4326)
);

INSERT INTO population (shape) VALUES (
 st_geometry ('polygon ((.40 .40, .40 .60, .60 .60, .60 .40, .40 .40))', 4326)
);

INSERT INTO waste_sites (site) VALUES (
 st_geometry ('point (.60 .60)', 4326)
);

INSERT INTO waste_sites (site) VALUES (
 st_geometry ('point (.30 .30)', 4326)
);
```

```
--Replace hs.id with ID value of second record in waste_sites table if not 2.
SELECT sa.id, st_astext (st_intersection (st_buffer (hs.site, .01), sa.shape))
 AS "Intersection"
 FROM population sa, waste_sites hs
 WHERE hs.id = 2
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 AND st_astext (st_intersection (st_buffer (hs.site, .1), sa.shape))
 NOT LIKE '%EMPTY%';

  id  Intersection

1     POLYGON (( 0.29000000 0.30000000, 0.30000000 0.30000000, 0.30000000
0.31000000, 0.29934597 0.30997859, 0.29869474 0.30991445, 0.29804910 0.30980785,
 0.29741181 0.30965926, 0.29678561 0.30946930, 0.29617317 0.30923880, 0.29557711
 0.30896873, 0.29500000 0.30866025, 0.29444430 0.30831470, 0.29391239 0.30793353
, 0.29340654 0.30751840, 0.29292893 0.30707107, 0.29248160 0.30659346, 0.2920664
7 0.30608761, 0.29168530 0.30555570, 0.29133975 0.30500000, 0.29103127 0.3044228
9, 0.29076121 0.30382683, 0.29053070 0.30321440, 0.29034074 0.30258819, 0.290192
15 0.30195090, 0.29008555 0.30130526, 0.29002141 0.30065403, 0.29000000 0.300000
00))

2     POLYGON (( 0.30000000 0.30000000, 0.31000000 0.30000000, 0.30997859
0.30065403, 0.30991445 0.30130526, 0.30980785 0.30195090, 0.30965926 0.30258819,
 0.30946930 0.30321440, 0.30923880 0.30382683, 0.30896873 0.30442289, 0.30866025
 0.30500000, 0.30831470 0.30555570, 0.30793353 0.30608761, 0.30751840 0.30659346
, 0.30707107 0.30707107, 0.30659346 0.30751840, 0.30608761 0.30793353, 0.3055557
0 0.30831470, 0.30500000 0.30866025, 0.30442289 0.30896873, 0.30382683 0.3092388
0, 0.30321440 0.30946930, 0.30258819 0.30965926, 0.30195090 0.30980785, 0.301305
26 0.30991445, 0.30065403 0.30997859, 0.30000000 0.31000000, 0.30000000 0.300000
00))
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Intersects

Definición

ST_Intersects devuelve 1 (Oracle y SQLite) o t (PostgreSQL) si la intersección de dos geometrías no genera un conjunto vacío; de lo contrario, devuelve 0 (Oracle y SQLite) o f (PostgreSQL).

Sintaxis

Oracle y PostgreSQL

```
sde.st_intersects (geometry1 sde.st_geometry, geometry2 sde.st_geometry)
```

SQLite

```
st_intersects (geometry1 geometryblob, geometry2 geometryblob)
```

Tipo de devolución

Booleano

Ejemplo

El jefe de bomberos quiere una lista de áreas sensibles dentro de un radio de un sitio de residuos peligrosos.

Las áreas sensibles se almacenan en la tabla sensitive_areas. La columna de forma, que se define como un polígono, almacena el contorno de cada una de las zonas sensibles.

Los emplazamientos peligrosos se almacenan en la tabla hazardous_sites. La columna de sitio, que se define como un punto, almacena una ubicación que es el centro geográfico de cada sitio peligroso.

La consulta SELECT crea un radio de zona de influencia alrededor de cada emplazamiento peligroso y devuelve una lista de las áreas sensibles que se intersecan con las zonas de influencia de los emplazamientos peligrosos.

Oracle

```
--Create and populate tables.
CREATE TABLE sensitive_areas (
 id integer,
 shape sde.st_geometry
);

CREATE TABLE hazardous_sites (
 id integer,
 site sde.st_geometry
);

INSERT INTO sensitive_areas VALUES (
 1,
 sde.st_geometry ('polygon ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO sensitive_areas VALUES (
 2,
 sde.st_geometry ('polygon ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
```

Copyright © 1995-2023 Esri. All rights reserved.

```
);

INSERT INTO sensitive_areas VALUES (
 3,
 sde.st_geometry ('polygon ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
);

INSERT INTO hazardous_sites VALUES (
 4,
 sde.st_geometry ('point (60 60)', 4326)
);

INSERT INTO hazardous_sites VALUES (
 5,
 sde.st_geometry ('point (30 30)', 4326)
);
```

```
--Create a buffer around the hazardous sites, then find the hazardous site buffers that
intersect sensitive areas.
SELECT sa.id SA_ID, hs.id HS_ID
 FROM SENSITIVE_AREAS sa, HAZARDOUS_SITES hs
 WHERE sde.st_intersects (sde.st_buffer (hs.site, .1), sa.shape) = 1
 ORDER BY sa.id;

SA_ID      HS_ID

    1          5
    2          5
    3          4
```

PostgreSQL

```
--Create and populate tables.
CREATE TABLE sensitive_areas (
 id serial,
 shape sde.st_geometry
);

CREATE TABLE hazardous_sites (
 id serial,
 site sde.st_geometry
);

INSERT INTO sensitive_areas (shape) VALUES (
 sde.st_geometry ('polygon ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO sensitive_areas (shape) VALUES (
 sde.st_geometry ('polygon ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
);

INSERT INTO sensitive_areas (shape) VALUES (
 sde.st_geometry ('polygon ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
);

INSERT INTO hazardous_sites (site) VALUES (
 sde.st_geometry ('point (60 60)', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO hazardous_sites (site) VALUES (
 sde.st_geometry ('point (30 30)', 4326)
);
```

```
--Create a buffer around the hazardous sites, then find the hazardous site buffers that
intersect sensitive areas.
SELECT sa.id AS sid, hs.id AS hid
 FROM sensitive_areas sa, hazardous_sites hs
 WHERE sde.st_intersects (sde.st_buffer (hs.site, .1), sa.shape) = 't'
 ORDER BY sa.id;

sid     hid

  1       2
  2       2
  3       1
```

SQLite

```
--Create and populate tables.
CREATE TABLE sensitive_areas (
 id integer primary key autoincrement not null
);

SELECT AddGeometryColumn (
 NULL,
 'sensitive_areas',
 'shape',
 4326,
 'polygon',
 'xy',
 'null'
);

CREATE TABLE hazardous_sites (
 id integer primary key autoincrement not null
);

SELECT AddGeometryColumn (
 NULL,
 'hazardous_sites',
 'site',
 4326,
 'point',
 'xy',
 'null'
);

INSERT INTO sensitive_areas (shape) VALUES (
 st_geometry ('polygon ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO sensitive_areas (shape) VALUES (
 st_geometry ('polygon ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
);

INSERT INTO sensitive_areas (shape) VALUES (
 st_geometry ('polygon ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO hazardous_sites (site) VALUES (
 st_geometry ('point (60 60)', 4326)
);

INSERT INTO hazardous_sites (site) VALUES (
 st_geometry ('point (30 30)', 4326)
);
```

```
--Create a buffer around the hazardous sites, then find the hazardous site buffers that
intersect sensitive areas.
SELECT sa.id AS "sid", hs.id AS "hid"
 FROM sensitive_areas sa, hazardous_sites hs
 WHERE st_intersects (st_buffer (hs.site, .1), sa.shape) = 1
 ORDER BY sa.id;

sid     hid

  1       2
  2       2
  3       1
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Is3d

Definición

ST_Is3d toma una ST_Geometry como parámetro de entrada y devuelve 1 (Oracle y SQLite) o t (PostgreSQL) si la geometría dada contiene coordinadas z; de lo contrario, devuelve 0 (Oracle y SQLite) o f (PostgreSQL).

Sintaxis

Oracle y PostgreSQL

```
sde.st_is3d (geometry1 sde.st_geometry)
```

SQLite

```
st_is3d (geometry1 geometryblob)
```

Tipo de devolución

Booleano

Ejemplo

Este ejemplo crea una tabla, is3d_test, y la rellena con registros.

A continuación, usando ST_Is3d, compruebe si alguno de los registros tiene una coordenada z.

Copyright © 1995-2023 Esri. All rights reserved.

Oracle

```
CREATE TABLE is3d_test (
 id integer,
 geo sde.st_geometry
);

INSERT INTO IS3D_TEST VALUES (
 1902,
 sde.st_geometry ('polygon ((40 120, 90 120, 90 150, 40 150, 40 120))', 4326)
);

INSERT INTO IS3D_TEST VALUES (
 1903,
 sde.st_geometry ('multipoint m((10 10 5), (50 10 6), (10 30 8))' , 4326)
);

INSERT INTO IS3D_TEST VALUES (
 1904,
 sde.st_geometry ('linestring z(10 10 166, 20 10 168)', 4326)
);

INSERT INTO IS3D_TEST VALUES (
 1905,
 sde.st_geometry ('point zm(10 10 16 30)', 4326)
);
```

```
SELECT id, sde.st_is3d (geo) Is_3D
 FROM IS3D_TEST;

        ID      IS_3D

      1902          0
      1903          0
      1904          1
      1905          1
```

PostgreSQL

```
CREATE TABLE is3d_test (
 id integer,
 geo sde.st_geometry
);

INSERT INTO IS3D_TEST VALUES (
 1902,
 sde.st_geometry ('polygon ((40 120, 90 120, 90 150, 40 150, 40 120))', 4326)
);

INSERT INTO IS3D_TEST VALUES (
 1903,
 sde.st_geometry ('multipoint m(10 10 5, 50 10 6, 10 30 8)' , 4326)
);

INSERT INTO IS3D_TEST VALUES (
 1904,
 sde.st_geometry ('linestring z(10 10 166, 20 10 168)', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO IS3D_TEST VALUES (
 1905,
 sde.st_geometry ('point zm(10 10 16 30)', 4326)
);
```

```
SELECT id, sde.st_is3d (geo)
 AS Is_3D
 FROM is3d_test;

        id      is_3d

      1902          f
      1903          f
      1904          t
      1905          t
```

SQLite

```
CREATE TABLE is3d_test (
 id integer
);

SELECT AddGeometryColumn (
 NULL,
 'is3d_test',
 'geo',
 4326,
 'geometryzm',
 'xyzm',
 'null'
);

INSERT INTO is3d_test VALUES (
 1902,
 st_geometry ('polygon ((40 120, 90 120, 90 150, 40 150, 40 120))', 4326)
);

INSERT INTO is3d_test VALUES (
 1903,
 st_geometry ('multipoint m((10 10 5), (50 10 6), (10 30 8))' , 4326)
);

INSERT INTO is3d_test VALUES (
 1904,
 st_geometry ('linestring z(10 10 166, 20 10 168)', 4326)
);

INSERT INTO is3d_test VALUES (
 1905,
 st_geometry ('point zm(10 10 16 30)', 4326)
);
```

```
SELECT id, st_is3d (geo)
 AS "Is_3D"
 FROM is3d_test;

      id      Is_3D
```

Copyright © 1995-2023 Esri. All rights reserved.

```
    1902        0
    1903        0
    1904        1
    1905        1
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_IsClosed

Definición

ST_IsClosed toma un ST_LineString o ST_MultiLineString y devuelve 1 (Oracle y SQLite) o t (PostgreSQL) si está cerrado; de lo contrario, devuelve 0 (Oracle y SQLite) o f (PostgreSQL).

Sintaxis

Oracle y PostgreSQL

```
sde.st_isclosed (line1 sde.st_geometry)
sde.st_isclosed (multiline1 sde.st_geometry)
```

SQLite

```
st_isclosed (geometry1 geometryblob)
```

Tipo de devolución

Booleano

Ejemplos

Pruebas de una cadena de texto de líneas

La tabla closed_linestring se crea con una columna de cadena de texto de línea única.

Las declaraciones INSERT insertan dos registros en la tabla closed_linestring. El primer registro no es una cadena de texto de líneas cerradas, mientras que la segunda lo es.

La consulta devuelve los resultados de la función ST_IsClosed. La primera fila devuelve 0 o f porque la cadena de líneas no está cerrada, mientras que la segunda fila devuelve 1 o t porque la cadena de líneas está cerrada.

Oracle

```
CREATE TABLE closed_linestring (ln1 sde.st_geometry);
```

```
INSERT INTO CLOSED_LINESTRING VALUES (
 sde.st_linefromtext ('linestring (10.02 20.01, 10.32 23.98, 11.92 25.64)', 4326)
);

INSERT INTO CLOSED_LINESTRING VALUES (
 sde.st_linefromtext ('linestring (10.02 20.01, 11.92 35.64, 25.02 34.15, 19.15 33.94,
10.02 20.01)', 4326)
);
```

```
SELECT sde.st_isclosed (ln1) Is_it_closed
 FROM CLOSED_LINESTRING;
```

Copyright © 1995-2023 Esri. All rights reserved.

```
Is_it_closed

0
1
```

PostgreSQL

```
CREATE TABLE closed_linestring (ln1 sde.st_geometry);
```

```
INSERT INTO closed_linestring VALUES (
 sde.st_linestring ('linestring (10.02 20.01, 10.32 23.98, 11.92 25.64)', 4326)
);

INSERT INTO closed_linestring VALUES (
sde.st_linestring ('linestring (10.02 20.01, 11.92 35.64, 25.02 34.15, 19.15 33.94,
10.02 20.01)', 4326)
);
```

```
SELECT sde.st_isclosed (ln1) AS Is_it_closed
FROM closed_linestring;

is_it_closed

f
t
```

Copyright © 1995-2023 Esri. All rights reserved.

SQLite

```
CREATE TABLE closed_linestring (id integer);

SELECT AddGeometryColumn (
 NULL,
 'closed_linestring',
 'ln1',
 4326,
 'linestring',
 'xy',
 'null'
);
```

```
INSERT INTO closed_linestring VALUES (
 1,
 st_linefromtext ('linestring (10.02 20.01, 10.32 23.98, 11.92 25.64)', 4326)
);

INSERT INTO closed_linestring VALUES (
 2,
 st_linefromtext ('linestring (10.02 20.01, 11.92 35.64, 25.02 34.15, 19.15 33.94,
10.02 20.01)', 4326)
);
```

```
SELECT st_isclosed (ln1)
 AS "Is_it_closed"
 FROM closed_linestring;

Is_it_closed

0
1
```

Pruebas de una cadena de texto multilínea

La tabla closed_mlinestring se crea con una columna ST_MultiLineString única.

Las declaraciones INSERT insertan un registro ST_MultiLineString que no está cerrado y otro que si.

La consulta enumera los resultados de la función ST_IsClosed. La primera fila devuelve 0 o f porque la cadena de texto multilínea no está cerrada. La segunda fila devuelve 1 o t porque la cadena de texto multilínea almacenada en la columna ln1 está cerrada. Una cadena de texto multilínea está cerrada si todos sus elementos de cadena de texto de línea están cerrados.

Oracle

```
CREATE TABLE closed_mlinestring (mln1 sde.st_geometry);
```

```
INSERT INTO closed_mlinestring VALUES (
 sde.st_mlinefromtext ('multilinestring ((10.02 20.01, 10.32 23.98, 11.92 25.64), (9.55
23.75, 15.36 30.11))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO closed_mlinestring VALUES (
 sde.st_mlinefromtext ('multilinestring ((10.02 20.01, 11.92 35.64, 25.02 
34.15, 19.15 33.94, 10.02 20.01), (51.71 21.73, 73.36 27.04, 71.52 32.87, 52.43 31.90, 
51.71 21.73))', 4326)
);
```

```
SELECT sde.st_isclosed (mln1) Is_it_closed
 FROM CLOSED_MLINESTRING;

Is_it_closed

0
1
```

PostgreSQL

```
CREATE TABLE closed_mlinestring (mln1 sde.st_geometry);
```

```
INSERT INTO closed_mlinestring VALUES (
 sde.st_mlinefromtext ('multilinestring ((10.02 20.01, 10.32 23.98, 11.92 25.64), (9.55 
23.75, 15.36 30.11))', 4326)
);

INSERT INTO closed_mlinestring VALUES (
 sde.st_mlinefromtext ('multilinestring ((10.02 20.01, 11.92 35.64, 25.02 
34.15, 19.15 33.94, 10.02 20.01), (51.71 21.73, 73.36 27.04, 71.52 32.87, 52.43 31.90, 
51.71 21.73))', 4326)
);
```

```
SELECT st_isclosed (mln1) 
 AS Is_it_closed
 FROM closed_mlinestring;

is_it_closed

f
t
```

Copyright © 1995-2023 Esri. All rights reserved.

SQLite

```
CREATE TABLE closed_mlinestring (mln1 geometryblob);

SELECT AddGeometryColumn (
 NULL,
 'closed_mlinestring',
 'mln1',
 4326,
 'multilinestring',
 'xy',
 'null'
);
```

```
INSERT INTO closed_mlinestring VALUES (
 st_mlinefromtext ('multilinestring ((10.02 20.01, 10.32 23.98, 11.92 25.64), (9.55
23.75, 15.36 30.11))', 4326)
);

INSERT INTO closed_mlinestring VALUES (
 st_mlinefromtext ('multilinestring ((10.02 20.01, 11.92 35.64, 25.02
34.15, 19.15 33.94, 10.02 20.01), (51.71 21.73, 73.36 27.04, 71.52 32.87, 52.43 31.90,
51.71 21.73))', 4326)
);
```

```
SELECT sde.st_isclosed (mln1)
 AS "Is_it_closed"
 FROM CLOSED_MLINESTRING;

Is_it_closed

0
1
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_IsEmpty

Definición

ST_IsEmpty devuelve 1 (Oracle y SQLite) o t (PostgreSQL) si el objeto ST_Geometry está vacío; de lo contrario, devuelve 0 (Oracle y SQLite) o f (PostgreSQL).

Sintaxis

Oracle y PostgreSQL

```
sde.st_isempty (geometry1 sde.st_geometry)
```

SQLite

```
st_isempty (geometry1 geometryblob)
```

Tipo de devolución

Booleano

Ejemplo

La declaración CREATE TABLE que se muestra a continuación crea la tabla empty_test con geotipo, que almacena el tipo de datos de las subclases almacenadas en la columna g1.

Las declaraciones INSERT insertan dos registros para las subclases de geometría de punto, cadena de texto de líneas y polígono: uno que está vacío y otro que no lo está.

La consulta SELECT devuelve el tipo de geometría de la columna geotipo y los resultados de la función ST_IsEmpty.

Oracle

```
CREATE TABLE empty_test (
 geotype varchar(20),
 g1 sde.st_geometry
);

INSERT INTO EMPTY_TEST VALUES (
 'Point',
 sde.st_pointfromtext ('point (10.02 20.01)', 4326)
);

INSERT INTO EMPTY_TEST VALUES (
 'Point',
 sde.st_pointfromtext ('point empty', 4326)
);

INSERT INTO EMPTY_TEST VALUES (
 'Linestring',
 sde.st_linefromtext ('linestring (10.02 20.01, 10.32 23.98, 11.92 25.64)', 4326)
);

INSERT INTO EMPTY_TEST VALUES (
 'Linestring',
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 sde.st_linefromtext ('linestring empty', 4326)
);

INSERT INTO EMPTY_TEST VALUES (
 'Polygon',
 sde.st_polyfromtext ('polygon ((10.02 20.01, 11.92 35.64, 25.02 34.15,
19.15 33.94, 10.02 20.01))', 4326)
);

INSERT INTO EMPTY_TEST VALUES (
 'Polygon',
 sde.st_polyfromtext('polygon empty', 4326)
);
```

```
SELECT geotype, sde.st_isempty (g1) Is_it_empty
FROM EMPTY_TEST;

GEOTYPE    Is_it_empty

Point         0
Point         1
Linestring    0
Linestring    1
Polygon       0
Polygon       1
```

PostgreSQL

```
CREATE TABLE empty_test (
 geotype varchar(20),
 g1 sde.st_geometry
);

INSERT INTO empty_test VALUES (
 'Point',
 sde.st_point ('point (10.02 20.01)', 4326)
);

INSERT INTO empty_test VALUES (
 'Point',
 sde.st_point ('point empty', 4326)
);

INSERT INTO empty_test VALUES (
 'Linestring',
 sde.st_linestring ('linestring (10.02 20.01, 10.32 23.98, 11.92 25.64)', 4326)
);

INSERT INTO empty_test VALUES (
 'Linestring',
 sde.st_linestring ('linestring empty', 4326)
);

INSERT INTO empty_test VALUES (
 'Polygon',
 sde.st_polygon ('polygon ((10.02 20.01, 11.92 35.64, 25.02 34.15,
19.15 33.94, 10.02 20.01))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO empty_test VALUES (
 'Polygon',
 sde.st_polygon ('polygon empty', 4326)
);
```

```
SELECT geotype, sde.st_isempty (g1)
 AS Is_it_empty
 FROM empty_test;

geotype   is_it_empty

Point         f
Point         t
Linestring    f
Linestring    t
Polygon       f
Polygon       f
```

SQLite

```
CREATE TABLE empty_test (
 geotype text(20)
);

SELECT AddGeometryColumn (
 NULL,
 'empty_test',
 'g1',
 4326,
 'geometry',
 'xy',
 'null'
);

INSERT INTO empty_test VALUES (
 'Point',
 st_point ('point (10.02 20.01)', 4326)
);

INSERT INTO empty_test VALUES (
 'Point',
 st_point ('point empty', 4326)
);

INSERT INTO empty_test VALUES (
 'Linestring',
 st_linestring ('linestring (10.02 20.01, 10.32 23.98, 11.92 25.64)', 4326)
);

INSERT INTO empty_test VALUES (
 'Linestring',
 st_linestring ('linestring empty', 4326)
);

INSERT INTO empty_test VALUES (
 'Polygon',
 st_polygon ('polygon ((10.02 20.01, 11.92 35.64, 25.02 34.15,
19.15 33.94, 10.02 20.01))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO empty_test VALUES (
 'Polygon',
 st_polygon ('polygon empty', 4326)
);
```

```
SELECT geotype, st_isempty (g1)
 AS "Is_it_empty"
 FROM empty_test;

GEOTYPE    Is_it_empty

Point         0
Point         1
Linestring    0
Linestring    1
Polygon       0
Polygon       1
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_IsMeasured

Definición

ST_IsMeasured toma un objeto de geometría como parámetro de entrada y devuelve 1 (Oracle y SQLite) o t (PostgreSQL) si la geometría dada contiene medidas; de lo contrario, devuelve 0 (Oracle y SQLite) o f (PostgreSQL).

Sintaxis

Oracle y PostgreSQL

```
sde.st_ismeasured (geometry1 sde.st_geometry)
```

SQLite

```
st_ismeasured (geometry1 geometryblob)
```

Tipo de devolución

Booleano

Ejemplo

Cree una tabla, ism_test, inserte valores en ella y determine a continuación qué filas de la tabla ism_test contienen medidas.

Copyright © 1995-2023 Esri. All rights reserved.

Oracle

```
CREATE TABLE ism_test (
 id integer,
 geom sde.st_geometry
);

INSERT INTO ISM_TEST VALUES (
 19,
 sde.st_geometry ('polygon ((40 120, 90 120, 90 150, 40 150, 40 120))', 4326)
);

INSERT INTO ISM_TEST VALUES (
 20,
 sde.st_geometry ('multipoint m((10 10 5), (50 10 6), (10 30 8))' , 4326)
);

INSERT INTO ISM_TEST VALUES (
 21,
 sde.st_geometry ('linestring z(10 10 166, 20 10 168)', 4326)
);

INSERT INTO ISM_TEST VALUES (
 22,
 sde.st_geometry ('point zm(10 10 16 30)', 4326)
);
```

```
SELECT id, sde.st_ismeasured (geom) M_values
 FROM ISM_TEST;

      ID    M_values

      19         0
      20         1
      21         0
      22         1
```

PostgreSQL

```
CREATE TABLE ism_test (
 id integer,
 geom sde.st_geometry
);

INSERT INTO ISM_TEST VALUES (
 19,
 sde.st_geometry ('polygon ((40 120, 90 120, 90 150, 40 150, 40 120))', 4326)
);

INSERT INTO ISM_TEST VALUES (
 20,
 sde.st_geometry ('multipoint m(10 10 5, 50 10 6, 10 30 8)' , 4326)
);

INSERT INTO ISM_TEST VALUES (
 21,
 sde.st_geometry ('linestring z(10 10 166, 20 10 168)', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO ISM_TEST VALUES (
 22,
 sde.st_geometry ('point zm(10 10 16 30)', 4326)
);
```

```
SELECT id, sde.st_ismeasured (geom)
  AS has_measures
  FROM ism_test;

      id   has_measures

      19         f
      20         t
      21         f
      22         t
```

SQLite

```
CREATE TABLE ism_test (
 id integer
);

SELECT AddGeometryColumn (
 NULL,
 'ism_test',
 'geom',
 4326,
 'geometryzm',
 'xyzm',
 'null'
);

INSERT INTO ISM_TEST VALUES (
 19,
 st_geometry ('polygon ((40 120, 90 120, 90 150, 40 150, 40 120))', 4326)
);

INSERT INTO ISM_TEST VALUES (
 20,
 st_geometry ('multipoint m((10 10 5), (50 10 6), (10 30 8))' , 4326)
);

INSERT INTO ISM_TEST VALUES (
 21,
 st_geometry ('linestring z(10 10 166, 20 10 168)', 4326)
);

INSERT INTO ISM_TEST VALUES (
 22,
 st_geometry ('point zm(10 10 16 30)', 4326)
);
```

```
SELECT id, st_ismeasured (geom)
 AS "M_values"
 FROM ism_test;

        ID    M_values
```

Copyright © 1995-2023 Esri. All rights reserved.

```
     19         0
     20         1
     21         0
     22         1
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_IsRing

Definición

ST_IsRing toma un ST_LineString y devuelve 1 (Oracle y SQLite) o t (PostgreSQL) si es un anillo (por ejemplo, el tipo ST_LineString es cerrado y simple); de lo contrario, devuelve 0 (Oracle y SQLite) o f (PostgreSQL).

Sintaxis

Oracle y PostgreSQL

```
sde.st_isring (line1 sde.st_geometry)
```

SQLite

```
st_isring (line1 geometryblob)
```

Tipo de devolución

Booleano

Ejemplo

La tabla ring_linestring se crea con una columna única ST_LineString, ln1.

Las declaraciones INSERT insertan tres cadenas de texto de líneas en la columna ln1. La primera fila contiene una cadena de texto de líneas que no está cerrada y no es un anillo. La segunda fila contiene una cadena de texto de líneas simple y cerrada que es un anillo. La tercera fila contiene una cadena de texto de líneas que está cerrada, pero no es simple porque se interseca con su propio interior. Tampoco es un anillo.

La consulta SELECT devuelve los resultados de la función ST_IsRing. La primera fila devuelve 0 o f porque las cadenas de líneas no son anillos, mientras que la segunda y la tercera fila devuelven 1 o t porque son anillos.

Copyright © 1995-2023 Esri. All rights reserved.

Oracle

```
CREATE TABLE ring_linestring (ln1 sde.st_geometry);
```

```
INSERT INTO RING_LINESTRING VALUES (
 sde.st_linefromtext ('linestring (10.02 20.01, 10.32 23.98, 11.92 25.64)', 4326)
);

INSERT INTO RING_LINESTRING VALUES (
 sde.st_linefromtext ('linestring (10.02 20.01, 11.92 35.64, 25.02 34.15, 19.15 33.94, 
10.02 20.01)', 4326)
);

INSERT INTO ring_linestring (ln1) VALUES (
 sde.st_linestring ('linestring (11 31, 11.25 31.12, 21.83 44.13, 16.45 44.24, 11 31)', 
4326)
);
```

```
SELECT sde.st_isring (ln1) Is_it_a_ring
 FROM RING_LINESTRING;

Is_it_a_ring

0
1
1
```

PostgreSQL

```
CREATE TABLE ring_linestring (ln1 sde.st_geometry);
```

```
INSERT INTO ring_linestring VALUES (
 sde.st_linestring ('linestring (10.02 20.01, 10.32 23.98, 11.92 25.64)', 4326)
);

INSERT INTO ring_linestring VALUES (
 sde.st_linestring ('linestring (10.02 20.01, 11.92 35.64, 25.02 34.15, 19.15 33.94, 
10.02 20.01)', 4326)
);

INSERT INTO ring_linestring (ln1) VALUES (
 sde.st_linestring ('linestring (11 31, 11.25 31.12, 21.83 44.13, 16.45 44.24, 11 31)', 
4326)
);
```

```
SELECT sde.st_isring (ln1)
 AS Is_it_a_ring
 FROM ring_linestring;

Is_it_a_ring

f
```

Copyright © 1995-2023 Esri. All rights reserved.

```
t
t
```

SQLite

```
CREATE TABLE ring_linestring (id integer primary key autoincrement not null);

SELECT AddGeometryColumn (
 NULL,
 'ring_linestring',
 'ln1',
 4326,
 'linestring',
 'xy',
 'null'
);
```

```
INSERT INTO ring_linestring (ln1) VALUES (
 st_linestring ('linestring (10.02 20.01, 10.32 23.98, 11.92 25.64)', 4326)
);

INSERT INTO ring_linestring (ln1) VALUES (
 st_linestring ('linestring (10.02 20.01, 11.92 35.64, 25.02 34.15, 19.15 33.94, 10.02 
20.01)', 4326)
);

INSERT INTO ring_linestring (ln1) VALUES (
 st_linestring ('linestring (11 31, 11.25 31.12, 21.83 44.13, 16.45 44.24, 11 31)', 
4326)
);
```

```
SELECT st_isring (ln1)
 AS "Is it a ring?"
 FROM ring_linestring;

Is it a ring?

0
1
1
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_IsSimple

Definición

ST_IsSimple devuelve 1 (Oracle y SQLite) o t (PostgreSQL) si el objeto de geometría es simple según lo define el Consorcio Geoespacial abierto (OGC); de lo contrario, devuelve 0 (Oracle y SQLite) o f (PostgreSQL).

Sintaxis

Oracle y PostgreSQL

```
sde.st_issimple (geometry1 sde.st_geometry)
```

SQLite

```
st_issimple (geometry1 geometryblob)
```

Tipo de devolución

Booleano

Ejemplo

Se crea la tabla issimple_test con dos columnas. La columna pid es un tipo de datos smallint que contiene el identificador único para cada fila. La columna g1 almacena las muestras de geometría simples y no simples.

Las declaraciones INSERT insertan dos registros en la tabla issimple_test. La primera es una cadena de texto de líneas simples porque no se interseca con su interior. La segunda es no simple, según lo define el OGC, porque no se interseca con su interior.

La consulta devuelve los resultados de la función ST_IsSimple. El primer registro devuelve 1 o t porque la cadena de líneas es simple, mientras que el segundo registro devuelve 0 o f porque la cadena de líneas no es simple.

Copyright © 1995-2023 Esri. All rights reserved.

Oracle

```
CREATE TABLE issimple_test (
 pid smallint,
 g1 sde.st_geometry
);
```

```
INSERT INTO ISSIMPLE_TEST VALUES (
 1,
 sde.st_linefromtext ('linestring (10 10, 20 20, 30 30)', 4326)
);

INSERT INTO ISSIMPLE_TEST VALUES (
 2,
 sde.st_linefromtext ('linestring (10 10, 20 20, 20 30, 10 30, 10 20,
20 10)', 4326)
);
```

```
SELECT pid, sde.st_issimple (g1) Is_it_simple
 FROM ISSIMPLE_TEST;

PID                 Is_it_simple

1                    1
2                    0
```

PostgreSQL

```
CREATE TABLE issimple_test (
 pid smallint,
 g1 sde.st_geometry
);
```

```
INSERT INTO issimple_test VALUES (
 1,
 sde.st_linestring ('linestring (10 10, 20 20, 30 30)', 4326)
);

INSERT INTO issimple_test VALUES (
 2,
 sde.st_linestring  ('linestring (10 10, 20 20, 20 30, 10 30, 10 20, 20 10)', 4326)
);
```

```
SELECT pid, sde.st_issimple (g1)
 AS Is_it_simple
 FROM issimple_test;

pid                is_it_simple

1                    t
2                    f
```

Copyright © 1995-2023 Esri. All rights reserved.

SQLite

```
CREATE TABLE issimple_test (
 pid integer
);

SELECT AddGeometryColumn (
 NULL,
 'issimple_test',
 'g1',
 4326,
 'linestring',
 'xy',
 'null'
);
```

```
INSERT INTO issimple_test VALUES (
 1,
 st_linestring ('linestring (10 10, 20 20, 30 30)', 4326)
);

INSERT INTO issimple_test VALUES (
 2,
 st_linestring  ('linestring (10 10, 20 20, 20 30, 10 30, 10 20, 20 10)', 4326)
);
```

```
SELECT pid, st_issimple (g1)
 AS Is_it_simple
 FROM issimple_test;

PID                 Is_it_simple

1                    1
2                    0
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Length

Definición

ST_Length devuelve la longitud de una cadena de líneas o de cadena de texto multilínea.

Sintaxis

Oracle y PostgreSQL

```
sde.st_length (line1 sde.st_geometry)
sde.st_length (multiline1 sde.st_geometry)
```

SQLite

```
st_length (line1 geometryblob)
st_length (multiline1 geometryblob)
st_length (line1  geometryblob, unit_name text)
st_length (multiline1  geometryblob, unit_name text)
```

Para obtener una lista de los nombres de unidades admitidos, consulte ST_Distance.

Tipo de devolución

Precisión doble

Ejemplo

Un ecologista que estudia los patrones migratorios de la población de salmones en las vías navegables del país quiere obtener la longitud de todos los sistemas de ríos y arroyos del país.

La tabla de vías navegables se crea con las columnas de Id. y nombre, que identifican cada sistema de arroyo y ríos almacenado en la tabla. La columna del agua es una cadena de texto multilínea porque los sistemas de ríos y arroyos se agregan con frecuencia de varias cadenas de texto de líneas.

La consulta SELECT devuelve el nombre del sistema junto con la longitud del sistema generada por la función ST_Length. En Oracle y PostgreSQL, las unidades son las empleadas por el sistema de coordenadas. En SQLite, se especifican kilómetros.

Oracle

```
CREATE TABLE waterways (
 oid integer,
 name varchar(128),
 water sde.st_geometry
);
```

```
INSERT INTO waterways (oid, name, water) VALUES (
  1111,
  'Genesee',
  sde.st_multilinestring ('multilinestring ((33 2, 34 3, 35 6),
(28 4, 29 5, 31 8, 43 12), (39 3, 37 4, 36 7))', 4326)
```

Copyright © 1995-2023 Esri. All rights reserved.

```
);
```

```
SELECT name, sde.st_length (water) "Length"
 FROM WATERWAYS;

NAME        Length

Genesee        27.6437123
```

PostgreSQL

```
CREATE TABLE waterways (
 oid serial,
 name varchar(128),
 water sde.st_geometry
);
```

```
INSERT INTO waterways (name, water) VALUES (
  'Genesee',
  sde.st_multilinestring ('multilinestring ((33 2, 34 3, 35 6),
(28 4, 29 5, 31 8, 43 12), (39 3, 37 4, 36 7))', 4326)
);
```

```
SELECT name AS "Watershed Name",
 sde.st_length (water) AS "Length"
 FROM waterways;

Watershed Name         |  Length

Genesee               | 27.6437123387202
```

SQLite

```
CREATE TABLE waterways (
 oid integer primary key autoincrement not null,
 name text(128)
);

SELECT AddGeometryColumn (
 NULL,
 'waterways',
 'water',
 4326,
 'multilinestring',
 'xy',
 'null'
);
```

```
INSERT INTO waterways (name, water) VALUES (
  'Genesee',
```

Copyright © 1995-2023 Esri. All rights reserved.

```
  st_multilinestring ('multilinestring ((33 2, 34 3, 35 6),
(28 4, 29 5, 31 8, 43 12), (39 3, 37 4, 36 7))', 4326)
);
```

```
SELECT name AS "Watershed Name",
 st_length (water, 'kilometer') AS "Length"
 FROM waterways1;

Watershed Name            Length

Genesee                   3047.75515002795
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_LineFromText

Nota:
Solo se admite en Oracle y SQLite; para PostgreSQL, utilice ST_LineString.

Definición

ST_LineFromText toma una representación de texto conocida de tipo ST_LineString y un Id. de referencia espacial y devuelve un ST_LineString.

Sintaxis

Oracle

```
sde.st_linefromtext (wkt clob, srid integer)
```

```
sde.st_linefromtext (wkt clob)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

SQLite

```
st_linefromtext (wkt text, srid int32)
```

```
st_linefromtext (wkt text)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

Tipo de devolución

ST_LineString

Ejemplo

La tabla linestring_test se crea con una columna única ST_LineString ln1.

La declaración INSERT inserta un ST_LineString en la columna ln1 utilizando la función ST_LineFromText.

Oracle

```
CREATE TABLE linestring_test (ln1 sde.st_geometry);
```

```
INSERT INTO LINESTRING_TEST VALUES (
 sde.st_linefromtext ('linestring (10.01 20.03, 35.93 19.04)', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

SQLite

```
CREATE TABLE linestring_test (id integer);
SELECT AddGeometryColumn (
 NULL,
 'linestring_test',
 'ln1',
 4326,
 'linestring',
 'xy',
 'null'
);
```

```
INSERT INTO LINESTRING_TEST (id, ln1) VALUES (
 1,
 st_linefromtext ('linestring (10.01 20.03, 35.93 19.04)', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_LineFromWKB

Definición

ST_LineFromWKB toma una representación binaria conocida (WKB) de tipo ST_LineString y un Id. de referencia espacial y devuelve un ST_LineString.

Sintaxis

Oracle

```
sde.st_linefromwkb (wkb blob, srid integer)
```

```
sde.st_linefromwkb (wkb blob)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

PostgreSQL

```
sde.st_linefromwkb (wkb bytea, srid integer)
```

SQLite

```
st_linefromwkb (wkb blob, srid int32)
```

```
st_linefromwkb (wkb blob)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

Tipo de devolución

ST_LineString

Ejemplo

Los siguientes comandos crean una tabla (sample_lines) y utilizan la función ST_LineFromWKB para insertar líneas desde una representación WKB. La fila se inserta en la tabla sample_lines con un Id. y una línea en el sistema de referencia espacial 4326 en la representación WKB.

Oracle

```
CREATE TABLE sample_lines (
 id smallint,
 geometry sde.st_linestring,
 wkb blob
);
INSERT INTO SAMPLE_LINES (id, geometry) VALUES (
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 1901,
 sde.st_linestring ('linestring (850 250, 850 850)', 4326)
);
INSERT INTO SAMPLE_LINES (id, geometry) VALUES (
 1902,
 sde.st_linestring ('linestring (33 2, 34 3, 35 6)', 4326)
);
UPDATE SAMPLE_LINES
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 1901;
UPDATE SAMPLE_LINES
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 1902;
SELECT id, sde.st_astext (sde.st_linefromwkb (wkb,4326)) LINE
 FROM SAMPLE_LINES;
ID   LINE
1901 LINESTRING (850.00000000 250.00000000, 850.00000000 850.00000000)
1902 LINESTRING (33.00000000 2.00000000, 34.00000000 3.00000000, 35.00000000 6.00000000)
```

PostgreSQL

```
CREATE TABLE sample_lines (
 id serial,
 geometry sde.st_linestring,
 wkb bytea
);
INSERT INTO sample_lines (geometry) VALUES (
 sde.st_linestring ('linestring (850 250, 850 850)', 4326)
);
INSERT INTO sample_lines (geometry) VALUES (
 sde.st_linestring ('linestring (33 2, 34 3, 35 6)', 4326)
);
--Replace ID values if necessary.
UPDATE sample_lines
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 1;
UPDATE sample_lines
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 2;
SELECT id, sde.st_astext (st_linefromwkb (wkb,4326))
 AS LINE
 FROM sample_lines;
id   line
1    LINESTRING (850 250, 850 850)
2    LINESTRING (33 2, 34 3, 35 6)
```

SQLite

```
CREATE TABLE sample_lines (
 id integer primary key autoincrement not null,
 wkb blob
);
SELECT AddGeometryColumn (
 NULL,
 'sample_lines',
 'geometry',
 4326,
 'linestring',
 'xy',
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 'null'
);
INSERT INTO sample_lines (geometry) VALUES (
 st_linestring ('linestring (850 250, 850 850)', 4326)
);
INSERT INTO sample_lines (geometry) VALUES (
 st_linestring ('linestring (33 2, 34 3, 35 6)', 4326)
);
--Replace ID values if necessary.
UPDATE sample_lines
 SET wkb = st_asbinary (geometry)
 WHERE id = 1;
UPDATE sample_lines
 SET wkb = st_asbinary (geometry)
 WHERE id = 2;
SELECT id, st_astext (st_linefromwkb (wkb,4326))
 AS LINE
 FROM sample_lines;
id   LINE
1    LINESTRING (850.00000000 250.00000000, 850.00000000 850.00000000)
2    LINESTRING (33.00000000 2.00000000, 34.00000000 3.00000000, 35.00000000 6.00000000)
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_LineString

Definición

ST_LineString es una función del descriptor de acceso que construye una cadena de líneas a partir de una representación de texto conocido.

Nota:
Cuando se crean tablas espaciales que se utilizarán con ArcGIS, es mejor crear la columna como supertipo de geometría (por ejemplo, ST_Geometry) en lugar de especificar un subtipo ST_Geometry.

Sintaxis

Oracle

```
sde.st_linestring (wkt clob, srid integer)
```

PostgreSQL

```
sde.st_linestring (wkt text, srid integer)
sde.st_linestring (esri_shape bytea, srid integer)
```

SQLite

```
st_linestring (wkt text, srid int32)
```

Tipo de devolución

ST_LineString

Ejemplo

Oracle

```
CREATE TABLE lines_test (
 id smallint,
 geometry sde.st_geometry
);

INSERT INTO LINES_TEST (id, geometry) VALUES (
 1901,
 sde.st_linestring ('linestring (750 150, 750 750)', 4326)
);

SELECT id, sde.st_astext (geometry) Linestring
 FROM   LINES_TEST;

  ID  LINESTRING

1901  LINESTRING  (750.00000000 150.00000000,
750.00000000 750.00000000)
```

Copyright © 1995-2023 Esri. All rights reserved.

PostgreSQL

```
CREATE TABLE lines_test (
 id serial,
 geometry sde.st_geometry
);

INSERT INTO lines_test (geometry) VALUES (
 sde.st_linestring ('linestring (750 150, 750 750)', 4326)
);

SELECT id, sde.st_astext (geometry)
 AS Linestring
 FROM lines_test;

  id  linestring

  1  LINESTRING  (750 150, 750 750)
```

SQLite

```
CREATE TABLE lines_test (
 id integer primary key autoincrement not null
);

SELECT AddGeometryColumn (
 NULL,
 'lines_test',
 'geometry',
 4326,
 'linestring',
 'xy',
 'null'
);

INSERT INTO lines_test (geometry) VALUES (
 st_linestring ('linestring (750 150, 750 750)', 4326)
);

SELECT id, st_astext (geometry)
 AS "Linestring"
 FROM lines_test;

  id  linestring

  1  LINESTRING  (750.00000000 150.00000000, 750.00000000 750.00000000)
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_M

Definición

ST_M toma un ST_Point como un parámetro de entrada y devuelve las coordenadas de medida (m).

En SQLite, ST_M se puede utilizar también para actualizar un valor de medición.

Sintaxis

Oracle y PostgreSQL

```
sde.st_m (point1 sde.st_point)
```

SQLite

```
st_m (point1 geometryblob)
st_m (point1 geometryblob, new_Mvalue double)
```

Tipo de devolución

Oracle y PostgreSQL

Número

SQLite

Precisión doble si se consulta un valor de medición; geometryblob si se actualiza un valor de medición

Ejemplos

Oracle

Se crea la tabla m_test y se insertan en ella tres puntos. Los tres contienen valores de medición. Se ejecuta una declaración SELECT con la función ST_M para devolver el valor de medición de cada punto.

```
CREATE TABLE m_test (
 id integer,
 geometry sde.st_point);

INSERT INTO M_TEST VALUES (
 1,
 sde.st_point (2, 3, 32, 5, 4322)
);

INSERT INTO M_TEST VALUES (
 2,
 sde.st_point (4, 5, 20, 4, 4326)
);

INSERT INTO M_TEST VALUES (
 3,
 sde.st_point (3, 8, 23, 7, 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
SELECT id, sde.st_m (geometry) M_COORD
 FROM M_TEST;

        ID    M_COORD

         1          5
         2          4
         3          7
```

PostgreSQL

Se crea la tabla m_test y se insertan en ella tres puntos. Los tres contienen valores de medición. Se ejecuta una declaración SELECT con la función ST_M para devolver el valor de medición de cada punto.

```
CREATE TABLE m_test (
 id serial,
 geometry sde.st_point
);

INSERT INTO m_test (geometry) VALUES (
 sde.st_point (2, 3, 32, 5, 4326)
);

INSERT INTO m_test (geometry) VALUES (
 sde.st_point (4, 5, 20, 4, 4326)
);

INSERT INTO m_test (geometry) VALUES (
 sde.st_point (3, 8, 23, 7, 4326)
);

SELECT id, sde.st_m (geometry)
 AS M_COORD
 FROM m_test;

        id    m_coord

         1          5
         2          4
         3          7
```

SQLite

En el primer ejemplo, se crea la tabla m_test y se insertan en ella tres puntos. Los tres contienen valores de medición. Se ejecuta una declaración SELECT con la función ST_M para devolver el valor de medición de cada punto.

```
CREATE TABLE m_test (
 id integer primary key autoincrement not null
);

SELECT AddGeometryColumn (
 NULL,
 'm_test',
 'geometry',
 4326,
 'pointzm',
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 'xyzm',
 'null'
);

INSERT INTO m_test (geometry) VALUES (
 st_point (2, 3, 32, 5, 4326)
);

INSERT INTO m_test (geometry) VALUES (
 st_point (4, 5, 20, 4, 4326)
);

INSERT INTO m_test (geometry) VALUES (
 st_point (3, 8, 23, 7, 4326)
);

SELECT id, st_m (geometry)
 AS M_COORD
 FROM m_test;

id    m_coord

1     5.0
2     4.0
3     7.0
```

En este segundo ejemplo, el valor de medición se actualiza para el registro 3 de la tabla m_test.

```
SELECT st_m (geometry, 7.5)
 FROM m_test
 WHERE id = 3;
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_MaxM

Definición

ST_MaxM toma una geometría como un parámetro de entrada y devuelve la coordenada m máxima.

Sintaxis

Oracle y PostgreSQL

```
sde.st_maxm (geometry1 sde.st_geometry)
```

SQLite

```
st_maxm (geometry1 geometryblob)
```

Tipo de devolución

Oracle y PostgreSQL

Número

Si no hay valores m, se devuelve el valor NULL.

SQLite

Precisión doble

Si no hay valores m, se devuelve el valor NULL.

Ejemplo

Se crea la tabla maxm_test y se insertan en ella dos polígonos. A continuación, se ejecuta ST_MaxM para determinar el valor m máximo de cada polígono.

Oracle

```
CREATE TABLE maxm_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO MAXM_TEST VALUES (
 1901,
 sde.st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20
3))', 4326)
);

INSERT INTO MAXM_TEST VALUES (
 1902,
 sde.st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))',
4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
SELECT id, sde.st_maxm (geometry) Max_M
 FROM MAXM_TEST;

        ID            MAX_M

      1901                4
      1902               12
```

PostgreSQL

```
CREATE TABLE maxm_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO maxm_test VALUES (
 1901,
 sde.st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20
3))', 4326)
);

INSERT INTO maxm_test VALUES (
 1902,
 sde.st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))',
4326)
);

SELECT id, sde.st_maxm (geometry)
 AS Max_M
 FROM maxm_test;

        id            max_m

      1901                4
      1902               12
```

SQLite

```
CREATE TABLE maxm_test (
 id integer
);

SELECT AddGeometryColumn (
 NULL,
 'maxm_test',
 'geometry',
 4326,
 'polygonzm',
 'xyzm',
 'null'
);

INSERT INTO maxm_test VALUES (
 1901,
 st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20 3))',
4326)
);

INSERT INTO maxm_test VALUES (
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 1902,
 st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))', 4326)
);

SELECT id, st_maxm (geometry)
 AS "Max M"
 FROM maxm_test;

id             Max M

1901           4.0
1902           12.0
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_MaxX

Definición

ST_MaxX toma una geometría como un parámetro de entrada y devuelve la coordenada x máxima.

Sintaxis

Oracle y PostgreSQL

```
sde.st_maxx (geometry1 sde.st_geometry)
```

SQLite

```
st_maxx (geometry1 geometryblob)
```

Tipo de devolución

Oracle y PostgreSQL

Número

SQLite

Precisión doble

Ejemplo

Se crea la tabla maxx_test y se insertan en ella dos polígonos. A continuación, se usa la función ST_MaxX para determinar el valor x máximo de cada polígono.

Oracle

```
CREATE TABLE maxx_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO MAXX_TEST VALUES (
 1901,
 sde.st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20
3))', 4326)
);

INSERT INTO MAXX_TEST VALUES (
 1902,
 sde.st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))',
4326)
);

SELECT id, sde.st_maxx (geometry) Max_X
 FROM MAXX_TEST;

        ID      MAX_X
```

Copyright © 1995-2023 Esri. All rights reserved.

```
      1901       120
      1902         5
```

PostgreSQL

```
CREATE TABLE maxx_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO maxx_test VALUES (
 1901,
 sde.st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20
3))', 4326)
);

INSERT INTO maxx_test VALUES (
 1902,
 sde.st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))',
4326)
);

SELECT id, sde.st_maxx (geometry)
 AS Max_X
 FROM maxx_test;

        id      max_x

      1901       120
      1902         5
```

SQLite

```
CREATE TABLE maxx_test (
 id integer
);

SELECT AddGeometryColumn (
 NULL,
 'maxx_test',
 'geometry',
 4326,
 'polygonzm',
 'xyzm',
 'null'
);

INSERT INTO maxx_test VALUES (
 1901,
 st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20 3))',
4326)
);

INSERT INTO maxx_test VALUES (
 1902,
 st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
SELECT id, st_maxx (geometry)
 AS "max_x"
 FROM maxx_test;

id      max_x

1901    120.0
1902    5.00000000
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_MaxY

Definición

ST_MaxY toma una geometría como un parámetro de entrada y devuelve la coordenada y máxima.

Sintaxis

Oracle y PostgreSQL

```
sde.st_maxy (geometry1 sde.st_geometry)
```

SQLite

```
st_maxy (geometry1 geometryblob)
```

Tipo de devolución

Oracle y PostgreSQL

Número

SQLite

Precisión doble

Ejemplo

Se crea la tabla maxy_test y se insertan en ella dos polígonos. A continuación, se usa la función ST_MaxY para determinar el valor y máximo de cada polígono.

Oracle

```
CREATE TABLE maxy_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO MAXY_TEST VALUES (
 1901,
 sde.st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20
3))', 4326)
);

INSERT INTO MAXY_TEST VALUES (
 1902,
 sde.st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))',
4326)
);

SELECT id, sde.st_maxy (geometry) Max_Y
 FROM MAXY_TEST;

        ID      MAX_Y
```

Copyright © 1995-2023 Esri. All rights reserved.

```
      1901        140
      1902          4
```

PostgreSQL

```
CREATE TABLE maxy_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO maxy_test VALUES (
 1901,
 sde.st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20
3))', 4326)
);

INSERT INTO maxy_test VALUES (
 1902,
 sde.st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))',
4326)
);

SELECT id, sde.st_maxy (geometry)
 AS Max_Y
 FROM maxy_test;

        id      max_y

      1901        140
      1902          4
```

SQLite

```
CREATE TABLE maxy_test (
 id integer
);

SELECT AddGeometryColumn (
 NULL,
 'maxy_test',
 'geometry',
 4326,
 'polygonzm',
 'xyzm',
 'null'
);

INSERT INTO maxy_test VALUES (
 1901,
 st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20 3))',
4326)
);

INSERT INTO maxy_test VALUES (
 1902,
 st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
SELECT id, st_maxy (geometry)
 AS "max_y"
 FROM maxy_test;

id      max_y

1901    140.0
1902    4.00000000
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_MaxZ

Definición

ST_MaxZ toma una geometría como un parámetro de entrada y devuelve la coordenada z máxima.

Sintaxis

Oracle y PostgreSQL

```
sde.st_maxz (geometry1 sde.st_geometry)
```

SQLite

```
st_maxz (geometry1 geometryblob)
```

Tipo de devolución

Oracle y PostgreSQL

Número

Si no hay valores z, se devuelve el valor NULL.

SQLite

Precisión doble

Si no hay valores z, se devuelve el valor NULL.

Ejemplo

En el siguiente ejemplo, se crea la tabla maxz_test y se insertan en ella dos polígonos. A continuación, se ejecuta ST_MaxZ para devolver el valor z máximo de cada polígono.

Oracle

```
CREATE TABLE maxz_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO MAXZ_TEST VALUES (
 1901,
 sde.st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20
3))', 4326)
);

INSERT INTO MAXZ_TEST VALUES (
 1902,
 sde.st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))',
4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
SELECT id, sde.st_maxz (geometry) Max_Z
 FROM MAXZ_TEST;

        ID      MAX_Z

      1901         26
      1902         40
```

PostgreSQL

```
CREATE TABLE maxz_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO maxz_test VALUES (
 1901,
 sde.st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20
3))', 4326)
);

INSERT INTO maxz_test VALUES (
 1902,
 sde.st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))',
4326)
);

SELECT id, sde.st_maxz (geometry)
 AS Max_Z
 FROM maxz_test;

        id      max_z

      1901         26
      1902         40
```

SQLite

```
CREATE TABLE maxz_test (
 id integer
);

SELECT AddGeometryColumn (
 NULL,
 'maxz_test',
 'geometry',
 4326,
 'polygonzm',
 'xyzm',
 'null'
);

INSERT INTO maxz_test VALUES (
 1901,
 st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20 3))',
4326)
);

INSERT INTO maxz_test VALUES (
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 1902,
 st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))', 4326)
);

SELECT id AS "ID", st_maxz (geometry) AS "Max Z"
 FROM maxz_test;

ID      Max Z

1901    26.0
1902    40.0
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_MinM

Definición

ST_MinM toma una geometría como un parámetro de entrada y devuelve la coordenada m mínima.

Sintaxis

Oracle y PostgreSQL

```
sde.st_minm (geometry1 sde.st_geometry)
```

SQLite

```
st_minm (geometry1 geometryblob)
```

Tipo de devolución

Oracle y PostgreSQL

Número

Si no hay valores m, se devuelve el valor NULL.

SQLite

Precisión doble

Si no hay valores m, se devuelve el valor NULL.

Ejemplo

Se crea la tabla minm_test y se insertan en ella dos polígonos. A continuación, se ejecuta ST_MinM para determinar el valor de medición mínimo de cada polígono.

PostgreSQL

Oracle

```
CREATE TABLE minm_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO MINM_TEST VALUES (
 1901,
 sde.st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20
3))', 4326)
);

INSERT INTO MINM_TEST VALUES (
 1902,
 sde.st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))',
4326)
```

Copyright © 1995-2023 Esri. All rights reserved.

```
);

SELECT id, sde.st_minm (geometry) MinM
 FROM MINM_TEST;

        ID       MINM

      1901          3
      1902          5
```

PostgreSQL

```
CREATE TABLE minm_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO minm_test VALUES (
 1901,
 sde.st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20
3))', 4326)
);

INSERT INTO minm_test VALUES (
 1902,
 sde.st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))',
4326)
);

SELECT id, sde.st_minm (geometry)
 AS MinM
 FROM minm_test;

        id       minm

      1901          3
      1902          5
```

SQLite

```
CREATE TABLE minm_test (
 id integer
);

SELECT AddGeometryColumn (
 NULL,
 'minm_test',
 'geometry',
 4326,
 'polygonzm',
 'xyzm',
 'null'
);

INSERT INTO minm_test VALUES (
 1901,
 st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20 3))',
4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO minm_test VALUES (
 1902,
 st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))', 4326)
);

SELECT id, st_minm (geometry)
 AS "MinM"
 FROM minm_test;

id       MinM

1901     3.0
1902     5.0
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_MinX

Definición

ST_MinX toma una geometría como un parámetro de entrada y devuelve la coordenada x mínima.

Sintaxis

Oracle y PostgreSQL

```
sde.st_minx (geometry1 sde.st_geometry)
```

SQLite

```
st_minx (geometry1 geometryblob)
```

Tipo de devolución

Oracle y PostgreSQL

Número

SQLite

Precisión doble

Ejemplo

Se crea la tabla minx_test y se insertan en ella dos polígonos. A continuación, se ejecuta ST_MinX para determinar el valor de coordenada x mínimo de cada polígono.

Oracle

```
CREATE TABLE minx_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO MINX_TEST VALUES (
 1901,
 sde.st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20
3))', 4326)
);

INSERT INTO MINX_TEST VALUES (
 1902,
 sde.st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))',
4326)
);

SELECT id, sde.st_minx (geometry) MinX
 FROM MINX_TEST;

        ID       MINX
```

Copyright © 1995-2023 Esri. All rights reserved.

```
      1901        110
      1902          0
```

PostgreSQL

```
CREATE TABLE minx_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO minx_test VALUES (
 1901,
 sde.st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20
3))', 4326)
);

INSERT INTO minx_test VALUES (
 1902,
 sde.st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))',
4326)
);

SELECT id, sde.st_minx (geometry)
 AS MinX
 FROM minx_test;

        id       minx

      1901        110
      1902          0
```

SQLite

```
CREATE TABLE minx_test (
 id integer
);

SELECT AddGeometryColumn (
 NULL,
 'minx_test',
 'geometry',
 4326,
 'polygonzm',
 'xyzm',
 'null'
);

INSERT INTO minx_test VALUES (
 1914,
 st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20 3))',
4326)
);

INSERT INTO minx_test VALUES (
 1915,
 st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
SELECT id AS "ID", st_minx (geometry) AS "MinX"
 FROM minx_test;

ID       MinX

1914     110.0
1915       0.0
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_MinY

Definición

ST_MinY toma una geometría como un parámetro de entrada y devuelve la coordenada y mínima.

Sintaxis

Oracle y PostgreSQL

```
sde.st_miny (geometry1 sde.st_geometry)
```

SQLite

```
st_miny (geometry1 geometryblob)
```

Tipo de devolución

Oracle y PostgreSQL

Número

SQLite

Precisión doble

Ejemplo

Se crea la tabla miny_test y se insertan en ella dos polígonos. A continuación, se ejecuta ST_MinY para determinar el valor de coordenada y mínimo de cada polígono.

PostgreSQL

Oracle

```
CREATE TABLE miny_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO MINY_TEST VALUES (
 1901,
 sde.st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20
3))', 4326)
);

INSERT INTO MINY_TEST VALUES (
 1902,
 sde.st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))',
4326)
);

SELECT id, sde.st_miny (geometry) MinY
 FROM MINY_TEST;
```

Copyright © 1995-2023 Esri. All rights reserved.

```
        ID       MINY

      1901        120
      1902          0
```

PostgreSQL

```
CREATE TABLE miny_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO miny_test VALUES (
 1901,
 sde.st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20
3))', 4326)
);

INSERT INTO miny_test VALUES (
 1902,
 sde.st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))',
4326)
);

SELECT id, sde.st_miny (geometry)
 AS MinY
 FROM miny_test;

        id       miny

      1901        120
      1902          0
```

SQLite

```
CREATE TABLE miny_test (
 id integer
);

SELECT AddGeometryColumn (
 NULL,
 'miny_test',
 'geometry',
 4326,
 'polygonzm',
 'xyzm',
 'null'
);

INSERT INTO miny_test VALUES (
 101,
 st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20 3))',
4326)
);

INSERT INTO miny_test VALUES (
 102,
 st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))', 4326)
```

Copyright © 1995-2023 Esri. All rights reserved.

```
);

SELECT id, st_miny (geometry)
 AS "MinY"
 FROM miny_test;

id       MinY

101      120.0
102        0.0
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_MinZ

Definición

ST_MinZ toma una geometría como un parámetro de entrada y devuelve la coordenada z mínima.

Sintaxis

Oracle y PostgreSQL

```
sde.st_minz (geometry1 sde.st_geometry)
```

SQLite

```
st_minz (geometry1 geometryblob)
```

Tipo de devolución

Oracle y PostgreSQL

Número

Si no hay valores z, se devuelve el valor NULL.

SQLite

Precisión doble

Si no hay valores z, se devuelve el valor NULL.

Ejemplo

Se crea la tabla minz_test y se insertan en ella dos polígonos. A continuación, se ejecuta ST_MinZ para determinar el valor de coordenada z mínimo de cada polígono.

Oracle

```
CREATE TABLE minz_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO MINZ_TEST VALUES (
 1901,
 sde.st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20
3))', 4326)
);

INSERT INTO MINZ_TEST VALUES (
 1902,
 sde.st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))',
4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
SELECT id, sde.st_minz (geometry) MinZ
 FROM MINZ_TEST;

        ID       MINZ

      1901         20
      1902         31
```

PostgreSQL

```
CREATE TABLE minz_test (
 id integer,
 geometry st_geometry
);

INSERT INTO minz_test VALUES (
 1901,
 st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20 3))',
4326)
);

INSERT INTO minz_test VALUES (
 1902,
 st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))', 4326)
);

SELECT id, st_minz (geometry)
 AS MinZ
 FROM minz_test;

        id       minz

      1901         20
      1902         31
```

SQLite

```
CREATE TABLE minz_test (
 id integer
);

SELECT AddGeometryColumn (
 NULL,
 'minz_test',
 'geometry',
 4326,
 'polygonzm',
 'xyzm',
 'null'
);

INSERT INTO minz_test VALUES (
 1901,
 st_polygon ('polygon zm((110 120 20 3, 110 140 22 3, 120 130 26 4, 110 120 20 3))',
4326)
);

INSERT INTO minz_test VALUES (
 1902,
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 st_polygon ('polygon zm((0 0 40 7, 0 4 35 9, 5 4 32 12, 5 0 31 5, 0 0 40 7))', 4326)
);

SELECT id, st_minz (geometry)
 AS "MinZ"
 FROM minz_test;

id        MinZ

1901      20.0
1902      31.0
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_MLineFromText

Nota:
Solo se utiliza en Oracle y SQLite; para PostgreSQL, utilice ST_MultiLineString.

Definición

ST_MLineFromText toma una representación de texto conocido de tipo ST_MultiLineString y un Id. de referencia espacial y devuelve un ST_MultiLineString.

Sintaxis

Oracle

```
sde.st_mlinefromtext (wkt clob, srid integer)
```

```
sde.st_mlinefromtext (wkt clob)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

SQLite

```
st_mlinefromtext (wkt text, srid int32)
```

```
st_mlinefromtext (wkt text)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

Tipo de devolución

ST_MultiLineString

Ejemplo

La tabla mlinestring_test se crea con la columna gid smallint que identifica exclusivamente la fila y la columna ST_MultiLineString ml1.

La declaración INSERT inserta ST_MultiLineString con la función ST_MLineFromText.

Copyright © 1995-2023 Esri. All rights reserved.

Oracle

```
CREATE TABLE mlinestring_test (
 gid smallint,
 ml1 sde.st_geometry
);
```

```
INSERT INTO MLINESTRING_TEST VALUES (
 1,
 sde.st_mlinefromtext ('multilinestring ((10.01 20.03, 10.52 40.11, 30.29 41.56,
31.78 10.74), (20.93 20.81, 21.52 40.10))', 4326)
);
```

SQLite

```
CREATE TABLE mlinestring_test (
 gid integer
);
SELECT AddGeometryColumn (
 NULL,
 'mlinestring_test',
 'ml1',
 4326,
 'multilinestring',
 'xy',
 'null'
);
```

```
INSERT INTO MLINESTRING_TEST VALUES (
 1,
 st_mlinefromtext ('multilinestring ((10.01 20.03, 10.52 40.11, 30.29 41.56,
31.78 10.74), (20.93 20.81, 21.52 40.10))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_MLineFromWKB

Definición

ST_MLineFromWKB toma una representación binaria conocida (WKB) de tipo ST_MultiLineString y un Id. de referencia espacial y crea un ST_MultiLineString.

Sintaxis

Oracle

```
sde.st_mlinefromwkb (wkb blob, srid integer)
```

```
sde.st_mlinefromwkb (wkb blob)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

PostgreSQL

```
sde.st_mlinefromwkb (wkb bytea, srid integer)
```

SQLite

```
st_mlinefromwkb (wkb blob, srid int32)
```

```
st_mlinefromwkb (wkb blob)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

Tipo de devolución

ST_MultiLineString

Ejemplo

En este ejemplo se muestra cómo se puede utilizar ST_MLineFromWKB para crear una cadena de texto multilínea desde su representación binaria conocida. La geometría es una cadena de texto multilínea en el sistema de referencia espacial 4326. En este ejemplo, la cadena de texto multilínea se almacena con el Id. = 10 en la columna geometry de la tabla sample_mlines, y la columna wkb se actualiza con su representación binaria conocida (utilizando la función ST_AsBinary). Por último, la función ST_MLineFromWKB se utiliza para devolver la cadena de texto multilínea desde la columna wkb. La tabla sample_mlines tiene una columna de geometría, donde se almacena la cadena de texto multilínea y una columna wkb, donde la representación de WKB de cadena de texto multilínea está almacenada.

La declaración SELECT incluye la función ST_MLineFromWKB, que se utiliza para recuperar la cadena de texto multilínea desde la columna wkb.

Copyright © 1995-2023 Esri. All rights reserved.

Oracle

```
CREATE TABLE sample_mlines (
 id integer,
 geometry sde.st_geometry,
 wkb blob
);
INSERT INTO SAMPLE_MLINES (id, geometry) VALUES (
 10,
 sde.st_multilinestring ('multilinestring ((61 2, 64 3, 65 6), (58 4, 59 5, 61 8), (69
3, 67 4, 66 7, 68 9))', 4326)
);
UPDATE SAMPLE_MLINES
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 10;
```

```
SELECT id, sde.st_astext (sde.st_mlinefromwkb (wkb,0)) MULTI_LINE_STRING
 FROM SAMPLE_MLINES
 WHERE id = 10;
ID    MULTI_LINE_STRING
10    MULTILINESTRING ((61.00000000 2.00000000, 64.00000000 3.00000000, 65.00000000
6.00000000), (58.00000000 4.00000000, 59.00000000 5.00000000, 61.00000000 8.0000000),
(69.00000000 3.00000000, 67.00000000 4.00000000, 66.00000000 7.00000000, 68.00000000
9.00000000 ))
```

PostgreSQL

```
CREATE TABLE sample_mlines (
 id integer,
 geometry sde.st_geometry,
 wkb bytea);
INSERT INTO sample_mlines (id, geometry) VALUES (
 10,
 sde.st_multilinestring ('multilinestring ((61 2, 64 3, 65 6), (58 4, 59 5, 61 8), (69
3, 67 4, 66 7, 68 9))', 4326)
);
UPDATE sample_mlines
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 10;
```

```
SELECT id, sde.st_astext (sde.st_mlinefromwkb (wkb,4326))
 AS MULTI_LINE_STRING
 FROM sample_mlines
 WHERE id = 10;
id   multi_line_string
10    MULTI_LINE_STRING ((61 2, 64 3, 65 6), (58 4, 59 5,61 8), (69 3, 67 4, 66 7, 68 9
))
```

SQLite

```
CREATE TABLE sample_mlines (
 id integer,
 wkb blob);
SELECT AddGeometryColumn (
 NULL,
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 'sample_mlines',
 'geometry',
 4326,
 'multilinestring',
 'xy',
 'null'
);
INSERT INTO sample_mlines (id, geometry) VALUES (
 10,
 st_multilinestring ('multilinestring ((61 2, 64 3, 65 6), (58 4, 59 5, 61 8), (69 3,
67 4, 66 7, 68 9))', 4326)
);
UPDATE sample_mlines
 SET wkb = st_asbinary (geometry)
 WHERE id = 10;
```

```
SELECT id, st_astext (st_mlinefromwkb (wkb,4326))
 AS MULTI_LINE_STRING
 FROM sample_mlines
 WHERE id = 10;
id   multi_line_string
10    MULTI_LINE_STRING ((61.00000000 2.00000000, 64.00000000 3.00000000, 65.00000000
6.00000000),
 (58.00000000 4.00000000, 59.00000000 5.00000000, 61.00000000 8.00000000),
 (69.00000000 3.00000000, 67.00000000 4.00000000, 66.00000000 7.00000000, 68.00000000
9.00000000 ))
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_MPointFromText

Nota:
Solo Oracle y SQLite; para PostgreSQL, utilice ST_MultiPoint.

Definición

ST_MPointFromText toma una representación de texto conocido (WKT) de tipo ST_MultiPoint y un Id. de referencia espacial y crea un ST_Multipoint.

Sintaxis

Oracle

```
sde.st_mpointfromtext (wkt clob, srid integer)
```

```
sde.st_mpointfromtext (wkt clob)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

SQLite

```
st_mpointfromtext (wkt text, srid int32)
```

```
st_mpointfromtext (wkt text)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

Tipo de devolución

ST_MultiPoint

Ejemplo

La tabla multipoint_test se crea con la única columna ST_MultiPoint mpt1.

La declaración INSERT inserta un multipunto en la columna mpt1 utilizando la función ST_MpointFromText.

Oracle

```
CREATE TABLE multipoint_test (mpt1 sde.st_geometry);
```

```
INSERT INTO MULTIPOINT_TEST VALUES (
 sde.st_mpointfromtext ('multipoint ((10.01 20.03), (10.52 40.11), (30.29 41.56),
(31.78 10.74))', 4326));
```

Copyright © 1995-2023 Esri. All rights reserved.

SQLite

```
CREATE TABLE multipoint_test (id integer);

SELECT AddGeometryColumn (
 NULL,
 'multipoint_test',
 'pt1',
 4326,
 'multipoint',
 'xy',
 'null'
);
```

```
INSERT INTO MULTIPOINT_TEST VALUES (
 1,
 st_mpointfromtext ('multipoint ((10.01 20.03), (10.52 40.11), (30.29 41.56), (31.78
10.74))', 4326));
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_MPointFromWKB

Definición

ST_MPointFromText toma una representación binaria conocida (WKB) de tipo ST_MultiPoint y un Id. de referencia espacial y crea un ST_MultiPoint.

Sintaxis

Oracle

```
sde.st_mpointfromwkb (wkb blob, srid integer)
```

```
sde.st_mpointfromwkb (wkb blob)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

PostgreSQL

```
sde.st_mpointfromwkb (wkb bytea, srid integer)
```

SQLite

```
st_mpointfromwkb (wkb blob, srid int32)
```

```
st_mpointfromwkb (wkb blob)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

Tipo de devolución

ST_MultiPoint

Ejemplo

En este ejemplo se muestra cómo se puede utilizar ST_MPointFromWKB para crear un multipunto desde su representación binaria conocida. La geometría es un multipunto en el sistema de referencia espacial 4326. En este ejemplo, el multipunto se almacena con el Id. = 10 en la columna GEOMETRY de la tabla SAMPLE_MPOINTS y, a continuación, la columna WKB se actualiza con su representación binaria conocida (utilizando la función ST_AsBinary). Por último, la función ST_MPointFromWKB se utiliza para devolver el multipunto de la columna WKB. La tabla SAMPLE_MPOINTS una columna GEOMETRY, donde el multipunto se almacena, y una columna de WKB, donde la representación binaria conocida multipunto se almacena.

En la siguiente declaración SELECT, la función ST_MPointFromWKB se usa para recuperar el multipunto desde la columna WKB.

Copyright © 1995-2023 Esri. All rights reserved.

Oracle

```
CREATE TABLE sample_mpoints (
 id integer,
 geometry sde.st_geometry,
 wkb blob
);

INSERT INTO SAMPLE_MPOINTS (id, geometry) VALUES (
 10,
 sde.st_multipoint ('multipoint ((4 14), (35 16), (24 13))', 4326)
);

UPDATE SAMPLE_MPOINTS
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 10;
```

```
SELECT id, sde.st_astext (sde.st_mpointfromwkb (wkb,4326)) MULTI_POINT
 FROM SAMPLE_MPOINTS
 WHERE id = 10;


ID           MULTI_POINT

10    MULTIPOINT ((4.00000000 14.00000000), (35.00000000 16.00000000), (24.00000000
13.00000000))
```

PostgreSQL

```
CREATE TABLE sample_mpoints (
 id integer,
 geometry sde.st_geometry,
 wkb bytea
);

INSERT INTO sample_mpoints (id, geometry) VALUES (
 10,
 sde.st_multipoint ('multipoint (4 14, 35 16, 24 13)', 4326)
);

UPDATE sample_mpoints
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 10;
```

```
SELECT id, sde.st_astext (sde.st_mpointfromwkb (wkb,4326))
 AS "MULTI_POINT"
 FROM sample_mpoints
 WHERE id = 10;


id           MULTI_POINT

10     MULTIPOINT (4 14, 35 16, 24 13)
```

Copyright © 1995-2023 Esri. All rights reserved.

SQLite

```
CREATE TABLE sample_mpoints (
 id integer,
 wkb blob
);

SELECT AddGeometryColumn (
 NULL,
 'sample_mpoints',
 'geometry',
 4326,
 'multipointzm',
 'xyzm',
 'null'
);

INSERT INTO SAMPLE_MPOINTS (id, geometry) VALUES (
 10,
 st_multipoint ('multipoint ((4 14), (35 16), (24 13))', 4326)
);

UPDATE sample_mpoints
 SET wkb = st_asbinary (geometry)
 WHERE id = 10;
```

```
SELECT id AS "ID",
 st_astext (st_mpointfromwkb (wkb,4326))
 AS "MULTI_POINT"
 FROM sample_mpoints
 WHERE id = 10;


ID           MULTI_POINT

10    MULTIPOINT ((4.00000000 14.00000000), (35.00000000 16.00000000), (24.00000000
13.00000000))
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_MPolyFromText

Nota:
Solo Oracle y SQLite; para PostgreSQL, utilice ST_MultiPolygon.

Definición

ST_MPointFromText toma una representación de texto conocido (WKB) de tipo ST_MultiPolygon y un Id. de referencia espacial y devuelve un ST_MultiPolygon.

Sintaxis

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

Oracle

```
sde.st_mpolyfromtext (wkt clob, srid integer)
```

```
sde.st_mpolyfromtext (wkt clob)
```

SQLite

```
st_mpolyfromtext (wkt text, srid int32)
```

```
st_mpolyfromtext (wkt text)
```

Tipo de devolución

ST_MultiPolygon

Ejemplo

La tabla multipolygon_test se crea con una columna ST_MultiPolygon, mpl1.

La declaración INSERT inserta un ST_MultiPolygon en la columna mpl1 utilizando la función ST_MpolyFromText.

Oracle

```
CREATE TABLE mpolygon_test (mpl1 sde.st_geometry);
```

```
INSERT INTO MPOLYGON_TEST VALUES (
 sde.st_mpolyfromtext ('multipolygon (((10.01 20.03, 10.52 40.11, 30.29 41.56,
31.78 10.74, 10.01 20.03), (21.23 15.74, 21.34 35.21, 28.94 35.35,
29.02 16.83, 21.23 15.74)), ((40.91 10.92, 40.56 20.19, 50.01 21.12,
51.34 9.81, 40.91 10.92)))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

SQLite

```
CREATE TABLE mpolygon_test (id integer);

SELECT AddGeometryColumn(
 NULL,
 'mpolygon_test',
 'mpl1',
 4326,
 'multipolygon',
 'xy',
 'null'
);
```

```
INSERT INTO MPOLYGON_TEST VALUES (
 1,
 st_mpolyfromtext ('multipolygon (((10.01 20.03, 10.52 40.11, 30.29 41.56,
31.78 10.74, 10.01 20.03), (21.23 15.74, 21.34 35.21, 28.94 35.35,
29.02 16.83, 21.23 15.74)), ((40.91 10.92, 40.56 20.19, 50.01 21.12,
51.34 9.81, 40.91 10.92)))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_MPolyFromWKB

Definición

ST_MPointFromWKB toma una representación binaria conocida (WKB) de tipo ST_MultiPolygon y un Id. de referencia espacial para devolver una ST_MultiPolygon.

Sintaxis

Oracle

```
sde.st_mpolyfromwkb (wkb blob, srid integer)
```

```
sde.st_mpolyfromwkb (wkb blob)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

PostgreSQL

```
sde.st_mpolyfromwkb (wkb bytea, srid integer)
```

SQLite

```
st_mpolyfromwkb (wkb blob, srid int32)
```

```
st_mpolyfromwkb (wkb blob)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

Tipo de devolución

ST_MultiPolygon

Ejemplo

En este ejemplo se muestra cómo se puede utilizar ST_MPolyFromWKB para crear un multipolígono desde su representación binaria conocida. La geometría es un multipolígono en el sistema de referencia espacial 4326. En este ejemplo, el multipolígono se almacena con el Id. = 10 en la columna geometry de la tabla sample_mpolys y, a continuación, la columna wkb se actualiza con su representación binaria conocida (utilizando la función ST_AsBinary). Por último, la función ST_MPolyFromWKB se utiliza para devolver el multipolígono desde la columna wkb. La tabla sample_mpolys tiene una columna de geometría, donde se almacena el multipolígono, y una columna wkb, donde la representación WKB de un multipolígono está almacenada.

La declaración SELECT incluye la función ST_MPointFromWKB, que se usa para recuperar el multipolígono desde la columna WKB.

Copyright © 1995-2023 Esri. All rights reserved.

Oracle

```
CREATE TABLE sample_mpolys (
 id integer,
 geometry sde.st_geometry,
 wkb blob
);
INSERT INTO SAMPLE_MPOLYS (id, geometry) VALUES (
 10,
 sde.st_multipolygon ('multipolygon (((1 72, 4 79, 5 76, 1 72), (10 20, 10 40, 30 41,
10 20), (9 43, 7 44, 6 47, 9 43)))', 4326)
);
UPDATE SAMPLE_MPOLYS
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 10;
```

```
SELECT id, sde.st_astext (sde.st_mpolyfromwkb (wkb,4326)) MULTIPOLYGON
 FROM SAMPLE_MPOLYS
 WHERE id = 10;
ID   MULTIPOLYGON
10   MULTIPOLYGON (((10.00000000 20.00000000, 30.00000000 41.00000000, 10.00000000
40.00000000, 10.00000000 20.00000000)), (1.00000000 72.00000000, 5.00000000
76.00000000, 4.00000000 79.0000000, 1.00000000 72,00000000)), (9.00000000 43.00000000,
6.00000000 47.00000000, 7.00000000 44.00000000, 9.00000000 43.00000000 )))
```

PostgreSQL

```
CREATE TABLE sample_mpolys (
 id integer,
 geometry sde.st_geometry,
 wkb bytea
);
INSERT INTO sample_mpolys (id, geometry) VALUES (
 10,
 sde.st_multipolygon ('multipolygon (((1 72, 4 79, 5 76, 1 72), (10 20, 10 40, 30 41,
10 20), (9 43, 7 44, 6 47, 9 43)))', 4326)
);
UPDATE sample_mpolys
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 10;
```

```
SELECT id, sde.st_astext (sde.st_mpolyfromwkb (wkb,4326))
 AS MULTIPOLYGON
 FROM sample_mpolys
 WHERE id = 10;
id   multipolygon
10   MULTIPOLYGON (((10 20, 30 41, 10 40, 10 20)),
 ((1 72, 5 76, 4 79, 1 72)), ((9 43, 6 47, 7 44, 9 43)))
```

SQLite

```
CREATE TABLE sample_mpolys (
 id integer,
 wkb blob
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
SELECT AddGeometryColumn(
 NULL,
 'sample_mpolys',
 'geometry',
 4326,
 'multipolygon',
 'xy',
 'null'
);
INSERT INTO SAMPLE_MPOLYS (id, geometry) VALUES (
 10,
 st_multipolygon ('multipolygon (((1 72, 4 79, 5 76, 1 72), (10 20, 10 40, 30 41, 10
20), (9 43, 7 44, 6 47, 9 43)))', 4326)
);
UPDATE SAMPLE_MPOLYS
 SET wkb = st_asbinary (geometry)
 WHERE id = 10;
```

```
SELECT id, st_astext (st_mpolyfromwkb (wkb,4326))
 AS "Multipolygon"
 FROM sample_mpolys
 WHERE id = 10;
id   Multipolygon
10   MULTIPOLYGON ((( 10.00000000 20.00000000, 30.00000000 41.00000000, 10.00000000
40.00000000, 10.00000000 20.00000000)),
 ((1.00000000 72.00000000, 5.00000000 76.00000000, 4.00000000 79.00000000, 1.00000000
72.00000000)),
 ((9.00000000 43.00000000, 6.00000000 47.00000000, 7.00000000 44.00000000, 9.00000000
43.00000000)))
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_MultiCurve

Nota:
Solo Oracle

Definición

ST_MultiCurve construye una entidad multicurva a partir de una representación de texto conocido.

Sintaxis

```
sde.st_multicurve (wkt clob, srid integer)
```

Tipo de devolución

ST_MultiLineString

Ejemplo

```
CREATE TABLE mcurve_test (id integer, geometry sde.st_geometry);

INSERT INTO MCURVE_TEST VALUES (
1910,
sde.st_multicurve ('multilinestring ((33 2, 34 3, 35 6),
(28 4, 29 5, 31 8, 43 12), (39 3, 37 4, 36 7))', 0)
);

SELECT sde.st_astext (geometry) MCURVE
FROM MCURVE_TEST;

ID        MCURVE

1110      MULTILINESTRING ((33.00000000 2.00000000, 34.00000000
           3.00000000, 35.00000000 6.00000000), (28.00000000 4.00000000,
           29.00000000 5.00000000, 31.00000000 8.0000000, 43.00000000
           12.00000000), (39.00000000 3.00000000, 37.00000000 4.00000000,
           36.00000000 7.00000000 ))
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_MultiLineString

Definición

ST_MultiLineString construye una cadena de líneas multilínea a partir de una representación de texto conocido.

Nota:
Cuando se crean tablas espaciales que se utilizarán con ArcGIS, es mejor crear la columna como supertipo de geometría (por ejemplo, ST_Geometry) en lugar de especificar un subtipo ST_Geometry.

Sintaxis

Oracle

```
sde.st_multilinestring (wkt clob, srid integer)
```

PostgreSQL

```
sde.st_multilinestring (wkt clob, srid integer)
sde.st_multilinestring (esri_shape bytea, srid integer)
```

SQLite

```
st_multilinestring (wkt text, srid int32)
```

Tipo de devolución

ST_MultiLineString

Ejemplo

Se crea una tabla, mlines_test, y se inserta una multilínea en ella con la función ST_MultiLineString.

Oracle

```
CREATE TABLE mlines_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO MLINES_TEST VALUES (
 1910,
 sde.st_multilinestring ('multilinestring ((33 2, 34 3, 35 6), (28 4, 29 5, 31 8, 43
12), (39 3, 37 4, 36 7))', 4326)
);
```

PostgreSQL

```
CREATE TABLE mlines_test (
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 id integer,
 geometry sde.st_geometry
);

INSERT INTO mlines_test VALUES (
 1910,
 sde.st_multilinestring ('multilinestring ((33 2, 34 3, 35 6), (28 4, 29 5, 31 8, 43
12), (39 3, 37 4, 36 7))', 4326)
);
```

SQLite

```
CREATE TABLE mlines_test (
 id integer
);

SELECT AddGeometryColumn(
 NULL,
 'mlines_test',
 'geometry',
 4326,
 'multilinestring',
 'xy',
 'null'
);

INSERT INTO mlines_test VALUES (
 1910,
 st_multilinestring ('multilinestring ((33 2, 34 3, 35 6), (28 4, 29 5, 31 8, 43 12),
(39 3, 37 4, 36 7))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_MultiPoint

Definición

ST_MultiPoint construye una entidad multipunto a partir de una representación de texto conocido.

Nota:
Cuando se crean tablas espaciales que se utilizarán con ArcGIS, es mejor crear la columna como supertipo de geometría (por ejemplo, ST_Geometry) en lugar de especificar un subtipo ST_Geometry.

Sintaxis

Oracle

```
sde.st_multipoint (wkt clob, srid integer)
```

PostgreSQL

```
sde.st_multipoint (wkt clob, srid integer)
sde.st_multipoint (esri_shape bytea, srid integer)
```

SQLite

```
st_multipoint (wkt text, srid int32)
```

Tipo de devolución

ST_MultiPoint

Ejemplo

Se crea una tabla, mpoint_test, y se inserta un multipunto en ella con la función ST_MultiPoint.

Oracle

```
CREATE TABLE mpoint_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO MPOINT_TEST VALUES (
 1110,
 sde.st_multipoint ('multipoint ((1 2), (3 4), (5 6))', 4326)
);
```

PostgreSQL

```
CREATE TABLE mpoint_test (
 id integer,
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 geometry sde.st_geometry
);

INSERT INTO mpoint_test VALUES (
 1110,
 sde.st_multipoint ('multipoint (1 2, 3 4, 5 6)', 4326)
);
```

SQLite

```
CREATE TABLE mpoint_test (
 id integer
);

SELECT AddGeometryColumn(
 NULL,
 'mpoint_test',
 'geometry',
 4326,
 'multipoint',
 'xy',
 'null'
);

INSERT INTO mpoint_test VALUES (
 1110,
 st_multipoint ('multipoint ((1 2), (3 4), (5 6))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_MultiPolygon

Definición

ST_MultiPolygon construye una entidad de multipolígono de una representación de texto conocido.

Nota:
Cuando se crean tablas espaciales que se utilizarán con ArcGIS, es mejor crear la columna como supertipo de geometría (por ejemplo, ST_Geometry) en lugar de especificar un subtipo ST_Geometry.

Sintaxis

Oracle

```
sde.st_multipolygon (wkt clob, srid integer)
```

PostgreSQL

```
sde.st_multipolygon (wkt clob, srid integer)
sde.st_multipolygon (esri_shape bytea, srid integer)
```

SQLite

```
st_multipolygon (wkt text, srid int32)
```

Tipo de devolución

ST_MultiPolygon

Ejemplo

Se crea una tabla, mpoly_test, y se inserta un multipolígono en ella con la función ST_MultiPolygon.

Oracle

```
CREATE TABLE mpoly_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO MPOLY_TEST VALUES (
 1110,
 sde.st_multipolygon ('multipolygon (((3 3, 4 6, 5 3, 3 3),(8 24, 9 25, 1 28, 8 24),
(13 33, 7 36, 1 40, 10 43, 13 33)))', 4326)
);
```

PostgreSQL

```
CREATE TABLE mpoly_test (
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 id integer,
 geometry sde.st_geometry
);

INSERT INTO mpoly_test VALUES (
 1110,
 sde.st_multipolygon ('multipolygon (((3 3, 4 6, 5 3, 3 3), (8 24, 9 25, 1 28, 8 24),
(13 33, 7 36, 1 40, 10 43, 13 33)))', 4326)
);
```

SQLite

```
CREATE TABLE mpoly_test (
 id integer
);

SELECT AddGeometryColumn(
 NULL,
 'mpoly_test',
 'geometry',
 4326,
 'multipolygon',
 'xy',
 'null'
);

INSERT INTO mpoly_test VALUES (
 1110,
 st_multipolygon ('multipolygon (((3 3, 4 6, 5 3, 3 3), (8 24, 9 25, 1 28, 8 24), (13
33, 7 36, 1 40, 10 43, 13 33)))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_MultiSurface

Nota:
Solo Oracle

Definición

ST_MultiSurface construye una entidad multisuperficie a partir de una representación de texto conocido.

Sintaxis

```
sde.st_multisurface (wkt clob, srid integer)
```

Tipo de devolución

ST_MultiSurface

Ejemplo

```
CREATE TABLE msurf_test (id integer, geometry sde.st_geometry);

INSERT INTO MSURF_TEST VALUES (
1110,
sde.st_multisurface ('multipolygon (((3 3, 4 6, 5 3, 3 3),(8 24, 9 25, 1 28, 8 24), (13
33, 7 36, 1 40, 10 43, 13 33)))', 0)
);

SELECT id, sde.st_astext (geometry) MULTI_SURFACE
FROM MSURF_TEST
WHERE id = 1110;

     ID    MULTI_SURFACE

   1110    MULTIPOLYGON (((13.00000000 33.00000000, 10.00000000
           43.00000000, 1.00000000 40.00000000, 7.00000000 36.00000000,
           13.00000000 33.00000000)), ((8.00000000 24.00000000, 9.00000000
           25.00000000, 1.00000000 28.0000000, 8.00000000 24.00000000)),
           ((3.00000000 3.00000000, 5.00000000 3.00000000,
           4.00000000 6.00000000, 3.00000000 3.00000000)))
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_NumGeometries

Definición

ST_NumGeometries toma una colección de geometría y devuelve el número de geometrías de la colección.

Sintaxis

Oracle

```
sde.st_numgeometries (multipoint1 sde.st_geometry)
sde.st_numgeometries (multiline1 sde.st_geometry)
sde.st_numgeometries (multipolygon1 sde.st_geometry)
```

PostgreSQL

```
sde.st_numgeometries (geometry1 sde.st_geomcollection)
```

SQLite

```
st_numgeometries (geometry1 geometryblob)
```

Tipo de devolución

Entero

Ejemplo

En el siguiente ejemplo se crea una tabla denominada sample_numgeom. Se le insertan un multipolígono y un multipunto. En la declaración SELECT, la función ST_NumGeometries se usa para determinar el número de geometrías (o entidades) de cada geometría.

Oracle

```
CREATE TABLE sample_numgeom (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO SAMPLE_NUMGEOM VALUES (
 1,
 sde.st_multipolygon ('multipolygon (((3 3, 4 6, 5 3, 3 3), (8 24, 9 25, 1 28, 8 24),
(13 33, 7 36, 1 40, 10 43, 13 33)))', 4326)
);

INSERT INTO SAMPLE_NUMGEOM VALUES (
 2,
 sde.st_multipoint ('multipoint ((1 2), (4 3), (5 6), (7 6), (8 8))', 4326)
);

SELECT id, sde.st_numgeometries (geometry) NUM_GEOMS_IN_COLL
 FROM SAMPLE_NUMGEOM;
```

Copyright © 1995-2023 Esri. All rights reserved.

```
ID          NUM_GEOMS_IN_COLL

  1                3
  2                5
```

PostgreSQL

```
CREATE TABLE sample_numgeom (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO sample_numgeom VALUES (
 1,
 sde.st_multipolygon ('multipolygon (((3 3, 4 6, 5 3, 3 3), (8 24, 9 25, 1 28, 8 24),
(13 33, 7 36, 1 40, 10 43, 13 33)))', 4326)
);

INSERT INTO sample_numgeom VALUES (
 2,
 sde.st_multipoint ('multipoint (1 2, 4 3, 5 6, 7 6, 8 8)', 4326)
);

SELECT id, sde.st_numgeometries (geometry)
 AS "number of geometries"
 FROM sample_numgeom;

id          number of geometries

  1                3
  2                5
```

SQLite

```
CREATE TABLE sample_numgeom (
 id integer
);

SELECT AddGeometryColumn(
 NULL,
 'sample_numgeom',
 'geometry',
 4326,
 'geometry',
 'xy',
 'null'
);

INSERT INTO sample_numgeom VALUES (
 1,
 st_multipolygon ('multipolygon (((3 3, 4 6, 5 3, 3 3), (8 24, 9 25, 1 28, 8 24), (13
33, 7 36, 1 40, 10 43, 13 33)))', 4326)
);

INSERT INTO sample_numgeom VALUES (
 2,
 st_multipoint ('multipoint ((1 2), (4 3), (5 6), (7 6), (8 8))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
SELECT id, st_numgeometries (geometry)
 AS "number of geometries"
 FROM sample_numgeom;

id          number of geometries

  1                 3
  2                 5
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_NumInteriorRing

Definición

ST_NumInteriorRing toma un ST_Polygon y devuelve el número de sus anillos interiores.

Sintaxis

Oracle y PostgreSQL

```
sde.st_numinteriorring (polygon1 sde.st_geometry)
```

SQLite

```
st_numinteriorring (polygon1 geometryblob)
```

Tipo de devolución

Entero

Ejemplo

Un ornitólogo quiere estudiar una población de aves en varias islas del mar del sur. Quiere identificar qué islas contienen uno o más lagos, ya que las especies de aves en las que está interesada se alimentan solo en lagos de agua dulce.

Las columnas de Id. y nombre de la tabla de islas identifica cada isla, mientras que la columna ST_Polygon de tierras almacena la geometría de las islas.

Ya que los anillos interiores representan los lagos, la declaración SELECT que incluye la función ST_NumInteriorRing enumera solo aquellas islas que tienen al menos un anillo interior.

Copyright © 1995-2023 Esri. All rights reserved.

Oracle

```
CREATE TABLE islands (
 id integer,
 name varchar(32),
 land sde.st_geometry
);

INSERT INTO islands VALUES (
 1,
 'Bear',
 sde.st_polygon ('polygon ((40 120, 90 120, 90 150, 40 150, 40 120),(50 130, 60 130, 60 
140, 50 140, 50 130),(70 130, 80 130, 80 140, 70 140, 70 130))', 4326)
);

INSERT INTO islands VALUES (
 2,
 'Johnson',
 sde.st_polygon ('polygon ((10 10, 50 10, 10 30, 10 10))', 4326)
);
```

```
SELECT name
 FROM ISLANDS
 WHERE sde.st_numinteriorring (land)> 0;

NAME

Bear
```

PostgreSQL

```
CREATE TABLE islands (
 id integer,
 name varchar(32),
 land sde.st_geometry
);

INSERT INTO islands VALUES (
 1,
 'Bear',
 sde.st_polygon ('polygon ((40 120, 90 120, 90 150, 40 150, 40 120),(50 130, 60 130, 60 
140, 50 140, 50 130),(70 130, 80 130, 80 140, 70 140, 70 130))', 4326)
);

INSERT INTO islands VALUES (
 2,
 'Johnson',
 sde.st_polygon ('polygon ((10 10, 50 10, 10 30, 10 10))', 4326)
);
```

```
SELECT name
 FROM islands
 WHERE sde.st_numinteriorring (land)> 0;

name

Bear
```

Copyright © 1995-2023 Esri. All rights reserved.

SQLite

```
CREATE TABLE islands (
 id integer,
 name varchar(32)
);

SELECT AddGeometryColumn(
 NULL,
 'islands',
 'land',
 4326,
 'polygon',
 'xy',
 'null'
);

INSERT INTO islands VALUES (
 1,
 'Bear',
 st_polygon ('polygon ((40 120, 90 120, 90 150, 40 150, 40 120),(50 130, 60 130, 60
140, 50 140, 50 130),(70 130, 80 130, 80 140, 70 140, 70 130))', 4326)
);

INSERT INTO islands VALUES (
 2,
 'Johnson',
 st_polygon ('polygon ((10 10, 50 10, 10 30, 10 10))', 4326)
);
```

```
SELECT name
 FROM islands
 WHERE st_numinteriorring (land)> 0;

name

Bear
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_NumPoints

Definición

ST_NumPoints devuelve el número de puntos (vértices) de una geometría.

Para los polígonos, los vértices de inicio y de fin se cuentan, aunque ocupan la misma ubicación.

Tenga en cuenta que este número es diferente que el atributo NUMPTS del tipo ST_Geometry. El atributo NUMPTS contiene un conteo de vértices en todas las partes de la geometría incluso los separadores que tienen lugar entre las partes. Hay un separador entre cada parte. Por ejemplo, una cadena de texto de líneas multiparte con tres partes tiene dos separadores. En el atributo NUMPTS, cada separador se cuenta como un vértice. Por el contrario, la función ST_NumPoints no incluye los separadores en el conteo de vértices.

Sintaxis

Oracle y PostgreSQL

```
sde.st_numpoints (geometry1 sde.st_geometry)
```

SQLite

```
st_numpoints (geometry1 geometryblob)
```

Tipo de devolución

Entero

Ejemplo

La tabla numpoints_test se crea con la columna geotype, que contiene el tipo de geometría almacenado en la columna g1.

Las declaraciones INSERT para insertar un punto, una cadena de texto y un polígono.

La consulta SELECT utiliza la función ST_NumPoints para obtener el número de puntos de cada entidad para cada tipo de entidad.

Oracle

```
CREATE TABLE numpoints_test (
 geotype varchar(12),
 g1 sde.st_geometry
);

INSERT INTO NUMPOINTS_TEST VALUES (
 'point',
 sde.st_pointfromtext ('point (10.02 20.01)', 4326)
);

INSERT INTO NUMPOINTS_TEST VALUES (
 'linestring',
 sde.st_linefromtext ('linestring (10.02 20.01, 23.73 21.92)', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO NUMPOINTS_TEST VALUES (
 'polygon',
 sde.st_polyfromtext ('polygon ((10.02 20.01, 23.73 21.92, 24.51 12.98, 11.64 13.42,
10.02 20.01))', 4326)
);
```

```
SELECT geotype, sde.st_numpoints (g1) Number_of_points
 FROM NUMPOINTS_TEST;

GEOTYPE     Number_of_points

point                       1
linestring                  2
polygon                     5
```

PostgreSQL

```
CREATE TABLE numpoints_test (
 geotype varchar(12),
 g1 sde.st_geometry
);

INSERT INTO numpoints_test VALUES (
 'point',
 sde.st_point ('point (10.02 20.01)', 4326)
);

INSERT INTO numpoints_test VALUES (
 'linestring',
 sde.st_linestring ('linestring (10.02 20.01, 23.73 21.92)', 4326)
);

INSERT INTO numpoints_test VALUES (
 'polygon',
 sde.st_polygon ('polygon ((10.02 20.01, 23.73 21.92, 24.51 12.98, 11.64 13.42, 10.02
20.01))', 4326)
);
```

```
SELECT geotype, sde.st_numpoints (g1)
 AS Number_of_points
 FROM numpoints_test;

geotype     number_of_points

point                       1
linestring                  2
polygon                     5
```

SQLite

```
CREATE TABLE numpoints_test (
 geotype text(12)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
SELECT AddGeometryColumn(
 NULL,
 'numpoints_test',
 'g1',
 4326,
 'geometry',
 'xy',
 'null'
);

INSERT INTO numpoints_test VALUES (
 'point',
 st_point ('point (10.02 20.01)', 4326)
);

INSERT INTO numpoints_test VALUES (
 'linestring',
 st_linestring ('linestring (10.02 20.01, 23.73 21.92)', 4326)
);

INSERT INTO numpoints_test VALUES (
 'polygon',
 st_polygon ('polygon ((10.02 20.01, 23.73 21.92, 24.51 12.98, 11.64 13.42, 10.02
20.01))', 4326)
);
```

```
SELECT geotype AS "Type of geometry", st_numpoints (g1) AS "Number of points"
 FROM numpoints_test;

Type of geometry     Number of points

point                      1
linestring                 2
polygon                    5
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_OrderingEquals

Nota:
Solo Oracle y PostgreSQL

Definición

ST_OrderingEquals compara dos ST_Geometries y devuelve 1 (Oracle) o t (PostgreSQL) si las geometrías son iguales y las coordenadas están en el mismo orden; de lo contrario, devuelve 0 (Oracle) o f (PostgreSQL).

Sintaxis

```
sde.st_orderingequals (g1 sde.st_geometry, g2 sde.st_geometry)
```

Tipo de devolución

Booleano

Ejemplo

Oracle

La siguiente sentencia `CREATE TABLE` crea la tabla LINESTRING_TEST, que tiene dos columnas de cadena de líneas, ln1 y ln2.

```
CREATE TABLE linestring_test (
lid integer,
ln1 sde.st_geometry,
ln2 sde.st_geometry);
```

La siguiente sentencia `INSERT` inserta dos valores ST_LineString en ln1 y ln2 que son iguales y tienen el mismo orden de coordenadas.

```
INSERT INTO LINESTRING_TEST VALUES (
1,
sde.st_geometry ('linestring (10.01 20.02, 21.50 12.10)', 0),
sde.st_geometry ('linestring (21.50 12.10, 10.01 20.02)', 0)
);
```

La siguiente sentencia `SELECT` y el resultado correspondiente establecido muestra cómo la función ST_Equals devuelve 1 (true) sin importar el orden de las coordenadas. La función ST_OrderingEquals devuelve 0 (false) si las geometrías no son iguales ni tienen el mismo orden de coordenadas.

```
SELECT lid, sde.st_equals (ln1, ln2) Equals, sde.st_orderingequals (ln1, ln2)
OrderingEquals
FROM LINESTRING_TEST;

lid Equals     OrderingEquals

1   1          0
```

Copyright © 1995-2023 Esri. All rights reserved.

PostgreSQL

La siguiente sentencia `CREATE TABLE` crea la tabla LINESTRING_TEST, que tiene dos columnas de cadena de líneas, ln1 y ln2.

```
CREATE TABLE linestring_test (
lid integer,
ln1 sde.st_geometry,
ln2 sde.st_geometry);
```

La siguiente sentencia `INSERT` inserta dos valores ST_LineString en ln1 y ln2 que son iguales y tienen el mismo orden de coordenadas.

```
INSERT INTO linestring_test VALUES (
1,
sde.st_linestring ('linestring (10.01 20.02, 21.50 12.10)', 0),
sde.st_linestring ('linestring (21.50 12.10, 10.01 20.02)', 0)
);
```

La siguiente sentencia `SELECT` y el correspondiente conjunto de resultados muestran cómo la función ST_Equals devuelve t (true) independientemente del orden de las coordenadas. La función ST_OrderingEquals devuelve f (false) si las geometrías no son iguales ni tienen el mismo orden de coordenadas.

```
SELECT lid, sde.st_equals (ln1, ln2) AS Equals, sde.st_orderingequals (ln1, ln2)
AS OrderingEquals
FROM linestring_test;

lid equals     orderingequals

1   t          f
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Overlaps

Definición

ST_Overlaps toma dos objetos de geometría y devuelve 1 (Oracle y SQLite) o t (PostgreSQL) si la intersección de los objetos genera un objeto de geometría de la misma dimensión pero distinto de los dos objetos de origen; de lo contrario, devuelve 0 (Oracle y SQLite) o f (PostgreSQL).

Sintaxis

Oracle y PostgreSQL

```
sde.st_overlaps (geometry1 sde.st_geometry, geometry2 sde.st_geometry)
```

SQLite

```
st_overlaps (geometry1 geometryblob, geometry2 geometryblob)
```

Tipo de devolución

Booleano

Ejemplo

El supervisor del condado necesita una lista de áreas sensibles que se superponen al radio de la zona de influencia de sitios de residuos peligrosos. La tabla sensitive_areas contiene varias columnas que describen las instituciones amenazadas además de la columna de forma, que almacena las geometrías de ST_Polygon.

La tabla hazardous_sites almacena la identidad de los sitios en la id de la columna, mientras que la ubicación geográfica de cada sitio se almacena en la columna de punto de sitio.

Las tablas sensitive_areas y hazardous_sites están unidas por la función ST_Overlaps, que devuelve la Id. para todas las filas sensitive_areas que contienen polígonos que se superponen con el radio de la zona de influencia de los puntos hazardous_sites.

Oracle

```
CREATE TABLE sensitive_areas (
 id integer,
 shape sde.st_geometry
);

CREATE TABLE hazardous_sites (
 id integer,
 site sde.st_geometry
);


INSERT INTO sensitive_areas VALUES (
 1,
 sde.st_geometry ('polygon ((.20 .30, .30 .30, .30 .40, .20 .40, .20 .30))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO sensitive_areas VALUES (
 2,
 sde.st_geometry ('polygon ((.30 .30, .30 .50, .50 .50, .50 .30, .30 .30))', 4326)
);

INSERT INTO sensitive_areas VALUES (
 3,
 sde.st_geometry ('polygon ((.40 .40, .40 .60, .60 .60, .60 .40, .40 .40))', 4326)
);

INSERT INTO hazardous_sites VALUES (
 4,
 sde.st_geometry ('point (.60 .60)', 4326)
);

INSERT INTO hazardous_sites VALUES (
 5,
 sde.st_geometry ('point (.30 .30)', 4326)
);
```

```
SELECT UNIQUE (hs.id)
 FROM HAZARDOUS_SITES hs, SENSITIVE_AREAS sa
 WHERE sde.st_overlaps (sde.st_buffer (hs.site, .001), sa.shape) = 1;

ID

4
5
```

PostgreSQL

```
CREATE TABLE sensitive_areas (
 id serial,
 shape sde.st_geometry
);

CREATE TABLE hazardous_sites (
 id serial,
 site sde.st_geometry
);


INSERT INTO sensitive_areas (shape) VALUES (
 sde.st_geometry ('polygon ((.20 .30, .30 .30, .30 .40, .20 .40, .20 .30))', 4326)
);

INSERT INTO sensitive_areas (shape) VALUES (
 sde.st_geometry ('polygon ((.30 .30, .30 .50, .50 .50, .50 .30, .30 .30))', 4326)
);

INSERT INTO sensitive_areas (shape) VALUES (
 sde.st_geometry ('polygon ((.40 .40, .40 .60, .60 .60, .60 .40, .40 .40))', 4326)
);

INSERT INTO hazardous_sites (site) VALUES (
 sde.st_geometry ('point (.60 .60)', 4326)
);

INSERT INTO hazardous_sites (site) VALUES (
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 sde.st_geometry ('point (.30 .30)', 4326)
);
```

```
SELECT DISTINCT (hs.id) AS "Hazardous Site ID"
 FROM hazardous_sites hs, sensitive_areas sa
 WHERE sde.st_overlaps (sde.st_buffer (hs.site, .001), sa.shape) = 't';

id

1
2
```

SQLite

```
CREATE TABLE sensitive_areas (
 id integer primary key autoincrement not null
);

SELECT AddGeometryColumn(
 NULL,
 'sensitive_areas',
 'shape',
 4326,
 'polygon',
 'xy',
 'null'
);

CREATE TABLE hazardous_sites (
 id integer primary key autoincrement not null,
 site_name varchar(30)
);

SELECT AddGeometryColumn(
 NULL,
 'hazardous_sites',
 'site',
 4326,
 'point',
 'xy',
 'null'
);

INSERT INTO sensitive_areas (shape) VALUES (
 st_geometry ('polygon ((.20 .30, .30 .30, .30 .40, .20 .40, .20 .30))', 4326)
);

INSERT INTO sensitive_areas (shape) VALUES (
 st_geometry ('polygon ((.30 .30, .30 .50, .50 .50, .50 .30, .30 .30))', 4326)
);

INSERT INTO sensitive_areas (shape) VALUES (
 st_geometry ('polygon ((.40 .40, .40 .60, .60 .60, .60 .40, .40 .40))', 4326)
);

INSERT INTO hazardous_sites (site_name, site) VALUES (
 'Kemlabs',
 st_geometry ('point (.60 .60)', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO hazardous_sites (site_name, site) VALUES (
 'Medi-Waste',
 st_geometry ('point (.30 .30)', 4326)
);
```

```
SELECT DISTINCT (hs.site_name) AS "Hazardous Site"
 FROM hazardous_sites hs, sensitive_areas sa
 WHERE st_overlaps (st_buffer (hs.site, .001), sa.shape) = 1;

Hazardous Site

Kemlabs
Medi-Waste
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Perimeter

Definición

ST_Perimeter devuelve la longitud de la línea continua que forma el límite de un polígono cerrado o una entidad multipoligonal.

Esta función es nueva en 10.8.1.

Sintaxis

Las dos primeras opciones de cada sección devuelven el perímetro en las unidades del sistema de coordenadas definido para la entidad. Las dos opciones siguientes permiten especificar la unidad de medida lineal. Para obtener una lista de valores admitidos para `linear_unit_name`, consulte ST_Distance.

Oracle y PostgreSQL

```
sde.st_perimeter (polygon sde.st_geometry)
```

```
sde.st_perimeter (multipolygon sde.st_geometry)
```

```
sde.st_perimeter (polygon sde.st_geometry, linear_unit_name text)
```

```
sde.st_perimeter (multipolygon sde.st_geometry, linear_unit_name text)
```

SQLite

```
st_perimeter (polygon sde.st_geometry)
```

```
st_perimeter (multipolygon sde.st_geometry)
```

```
st_perimeter (polygon sde.st_geometry, linear_unit_name text)
```

```
st_perimeter (multipolygon sde.st_geometry, linear_unit_name text)
```

Tipo de devolución

Precisión doble

Ejemplos

Oracle

En el ejemplo que aparece a continuación, un naturalista que está estudiando las aves costeras necesita determinar la longitud de la orilla de los lagos de una zona determinada. Los lagos están almacenados como polígonos en la tabla waterbodies. Se usa una sentencia `SELECT` con la función ST_Perimeter para devolver el perímetro de cada lago (entidad) en la tabla waterbodies.

```
--Create table named waterbodies
CREATE TABLE waterbodies (wbid INTEGER not null, waterbody sde.st_geometry);
--Insert a polygon feature to the waterbodies table
INSERT INTO waterbodies VALUES (
  1,
  sde.ST_Polygon ('polygon ((0 0, 0 4, 5 4, 5 0, 0 0))', 1)
);
--Find the perimeter of the polygon
```

Copyright © 1995-2023 Esri. All rights reserved.

```
SELECT sde.ST_Perimeter (waterbody)
  FROM waterbodies;
```

La sentencia SELECT devuelve lo siguiente:

```
ID PERIMETER
1 +1.8000000
```

En el siguiente ejemplo, creará una tabla llamada bfp, insertará tres entidades y calculará el perímetro de cada entidad en unidades de medida lineal:

```
--Create table named bfp
CREATE TABLE bfp (
 building_id integer not null,
 footprint sde.st_geometry);
--Insert polygon features to the bfp table
INSERT INTO BFP (building_id, footprint) VALUES (
 1,
 sde.st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);
INSERT INTO BFP (building_id, footprint) VALUES (
 2,
 sde.st_polygon ('polygon ((20 0, 30 20, 40 0, 20 0))', 4326)
);
INSERT INTO BFP (building_id, footprint) VALUES (
 3,
 sde.st_polygon ('polygon ((20 30, 25 35, 30 30, 20 30))', 4326)
);
--Find the perimeter of each polygon
SELECT sde.ST_Perimeter(footprint)
            ,sde.ST_Perimeter(footprint, 'meter') as Meter
            ,sde.ST_Perimeter(footprint, 'km') as KM
            ,sde.ST_Perimeter(footprint, 'yard') As Yard
FROM bfp;
```

La sentencia SELECT devuelve el perímetro de cada entidad en tres unidades:

```
st_perimeter    |       meter       |         km         |        yard
------------------+-------------------+--------------------+--------------------
 40.00000000000001 | 4421256.128972424 |  4421.256128972425 |   4835144.49800134
  64.7213595499958 | 7159231.951087892 | 7159.2319510878915 |  7829431.267593933
 24.14213562373095 | 2417672.365575198 |  2417.672365575198 | 2643998.6500166208
```

PostgreSQL

En el ejemplo que aparece a continuación, un naturalista que está estudiando las aves costeras necesita determinar la longitud de la orilla de los lagos de una zona determinada. Los lagos están almacenados como polígonos en la tabla waterbodies. Se usa una sentencia SELECT con la función ST_Perimeter para devolver el perímetro de cada lago (entidad) en la tabla waterbodies.

```
--Create table named waterbodies
CREATE TABLE waterbodies (wbid INTEGER not null, waterbody sde.st_geometry);
--Insert a polygon feature to the waterbodies table
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO waterbodies VALUES (
  1,
  sde.ST_Polygon ('polygon ((0 0, 0 4, 5 4, 5 0, 0 0))', 1)
);
--Find the perimeter of the polygon
SELECT sde.ST_Perimeter (waterbody)
  FROM waterbodies;
```

La sentencia SELECT devuelve lo siguiente:

```
ID PERIMETER
1 +1.8000000
```

En el siguiente ejemplo, creará una tabla llamada bfp, insertará tres entidades y calculará el perímetro de cada entidad en unidades de medida lineal:

```
--Create table named bfp
CREATE TABLE bfp (
 building_id serial,
 footprint sde.st_geometry);
--Insert polygon features to the bfp table
INSERT INTO bfp (footprint) VALUES (
 sde.st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);
INSERT INTO bfp (footprint) VALUES (
 sde.st_polygon ('polygon ((20 0, 30 20, 40 0, 20 0))', 4326)
);
INSERT INTO bfp (footprint) VALUES (
 sde.st_polygon ('polygon ((20 30, 25 35, 30 30, 20 30))', 4326)
);
--Find the perimeter of each polygon
SELECT sde.st_perimeter(footprint)
            ,sde.st_perimeter(footprint, 'meter') as Meter
            ,sde.st_perimeter(footprint, 'km') as KM
            ,sde.st_perimeter(footprint, 'yard') As Yard
FROM bfp;
```

La sentencia SELECT devuelve el perímetro de cada entidad en tres unidades:

```
 st_perimeter    |       meter        |         km          |         yard
-----------------+--------------------+---------------------+--------------------
 40.00000000000001 | 4421256.128972424 |  4421.256128972425 |   4835144.49800134
  64.7213595499958 | 7159231.951087892 | 7159.2319510878915 |  7829431.267593933
 24.14213562373095 | 2417672.365575198 |  2417.672365575198 | 2643998.6500166208
```

SQLite

En el ejemplo que aparece a continuación, un naturalista que está estudiando las aves costeras necesita determinar la longitud de la orilla de los lagos de una zona determinada. Los lagos están almacenados como polígonos en la tabla waterbodies. Se usa una sentencia SELECT con la función ST_Perimeter para devolver el perímetro de cada lago (entidad) en la tabla waterbodies.

```
--Create table named waterbodies and add a spatial column (waterbody) to it
```

Copyright © 1995-2023 Esri. All rights reserved.

```
CREATE TABLE waterbodies (wbid integer primary key autoincrement not null
);
SELECT AddGeometryColumn(
 NULL,
 'waterbodies',
 'waterbody',
 4326,
 'polygon',
 'xy',
 'null'
);
--Insert a polygon feature to the waterbodies table
INSERT INTO waterbodies VALUES (
  1,
  ST_Polygon ('polygon ((0 0, 0 4, 5 4, 5 0, 0 0))', 1)
);
--Find the perimeter of the polygon
SELECT ST_Perimeter (waterbody)
  FROM waterbodies;
```

La sentencia `SELECT` devuelve lo siguiente:

```
ID PERIMETER
1 +1.8000000
```

En el siguiente ejemplo, creará una tabla llamada bfp, insertará tres entidades y calculará el perímetro de cada entidad en unidades de medida lineal:

```
--Create table named bfp and add a spatial column (footprints) to it
CREATE TABLE bfp (
 building_id integer primary key autoincrement not null
);
SELECT AddGeometryColumn(
 NULL,
 'bfp',
 'footprint',
 4326,
 'polygon',
 'xy',
 'null'
);
--Insert polygon features to the bfp table
INSERT INTO bfp (footprint) VALUES (
 st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);
INSERT INTO bfp (footprint) VALUES (
 st_polygon ('polygon ((20 0, 30 20, 40 0, 20 0))', 4326)
);
INSERT INTO bfp (footprint) VALUES (
 st_polygon ('polygon ((20 30, 25 35, 30 30, 20 30))', 4326)
);
--Find the perimeter of each polygon
SELECT ST_Perimeter(footprint)
            ,ST_Perimeter(footprint, 'meter') as Meter
            ,ST_Perimeter(footprint, 'km') as KM
            ,ST_Perimeter(footprint, 'yard') As Yard
FROM bfp;
```

Copyright © 1995-2023 Esri. All rights reserved.

La sentencia SELECT devuelve el perímetro de cada entidad en tres unidades:

```
 st_perimeter    |       meter       |         km         |        yard
-----------------+-------------------+--------------------+--------------------
 40.00000000000001 | 4421256.128972424 |  4421.256128972425 |   4835144.49800134
  64.7213595499958 | 7159231.951087892 | 7159.2319510878915 |  7829431.267593933
 24.14213562373095 | 2417672.365575198 |  2417.672365575198 | 2643998.6500166208
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Point

Definición

ST_Point toma un objeto de texto conocido o unas coordenadas y un Id. de referencia espacial, y devuelve un ST_Point.

Nota:
Cuando se crean tablas espaciales que se utilizarán con ArcGIS, es mejor crear la columna como supertipo de geometría (por ejemplo, ST_Geometry) en lugar de especificar un subtipo ST_Geometry.

Sintaxis

Oracle

```
sde.st_point (wkt clob, srid integer)
sde.st_point (x number, y number, srid integer)
sde.st_point (x number, y number, m number, srid integer)
sde.st_point (x number, y number, z number, srid integer)
sde.st_point (x number, y number, z number, m number, srid integer)
```

PostgreSQL

```
sde.st_point (wkt clob, srid integer)
sde.st_point (esri_shape bytea, srid integer)sde.
sde.st_point (x double precision, y double precision, srid integer)
sde.st_point (x double precision, y double precision, m double precision, srid integer)
sde.st_point (x double precision, y double precision, z double precision, srid integer)
sde.st_point (x double precision, y double precision, z double precision, m double
precision, srid integer)
```

SQLite

```
st_point (wkt text, srid int32)
st_point (x float64, y float64, srid int32)
st_point (x float64, y float64, z float64, m float64, srid int32)
```

Tipo de devolución

ST_Point

Ejemplo

La siguiente sentencia `CREATE TABLE` crea la tabla point_test, que tiene una sola columna de punto, PT1.

La función ST_Point convierte las coordenadas de punto en una geometría ST_Point antes de insertarla en la columna pt1.

Copyright © 1995-2023 Esri. All rights reserved.

Oracle

```
CREATE TABLE point_test (pt1 sde.st_geometry);
```

```
INSERT INTO point_test VALUES (
 sde.st_point (10.01, 20.03, 4326)
);
```

PostgreSQL

```
CREATE TABLE point_test (pt1 sde.st_geometry);
```

```
INSERT INTO point_test VALUES (
 sde.st_point (10.01, 20.03, 4326)
);
```

SQLite

```
CREATE TABLE point_test (id integer);

SELECT AddGeometryColumn(
 NULL,
 'point_test',
 'pt1',
 4326,
 'point',
 'xy',
 'null'
);
```

```
INSERT INTO point_test VALUES (
 1,
 st_point (10.01, 20.03, 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_PointFromText

Nota:
Solo se utiliza en Oracle y SQLite; para PostgreSQL, utilice ST_Point.

Definición

ST_PointFromText toma una representación de texto conocido de tipo de punto y una Id. de referencia espacial y devuelve un punto.

Sintaxis

Oracle

```
sde.st_pointfromtext (wkt varchar2, srid integer)
```

```
sde.st_pointfromtext (wkt varchar2)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

SQLite

```
st_pointfromtext (wkt text, srid int32)
```

```
st_pointfromtext (wkt text)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

Tipo de devolución

ST_Point

Ejemplo

La tabla point_test se crea con la única columna pt1 de ST_Point.

La función ST_PointFromText convierte las coordenadas de texto de punto al formato de punto antes de que la declaración INSERT inserte el punto en la columna pt1.

Oracle

```
CREATE TABLE point_test (pt1 sde.st_geometry);
```

```
INSERT INTO POINT_TEST VALUES (
 sde.st_pointfromtext ('point (10.01 20.03)', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

SQLite

```
CREATE TABLE pt_test (id integer);
SELECT AddGeometryColumn(
 NULL,
 'pt_test',
 'pt1',
 4326,
 'point',
 'xy',
 'null'
);
```

```
INSERT INTO pt_test VALUES (
 1,
 st_pointfromtext ('point (10.01 20.03)', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_PointFromWKB

Definición

ST_PointFromWKB toma una representación binaria conocida (WKB) y una Id. de referencia espacial para devolver un ST_Point.

Sintaxis

Oracle

```
sde.st_pointfromwkb (wkb blob, srid integer)
```

```
sde.st_pointfromwkb (wkb blob)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

PostgreSQL

```
sde.st_pointfromwkb (wkb bytea, srid integer)
```

SQLite

```
st_pointfromwkb (wkb blob, srid int32)
```

```
st_pointfromwkb (wkb blob)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

Tipo de devolución

ST_Point

Ejemplo

En este ejemplo se muestra cómo ST_PointFromWKB se puede utilizar para crear un punto de su representación binaria conocida. Las geometrías son puntos en el sistema de referencia espacial 4326. En este ejemplo, los puntos se almacenan en la columna geometry de la tabla sample_points y, a continuación, la columna wkb se actualiza con sus representaciones binarias conocidas (utilizando la función ST_AsBinary). Por último, la función ST_PointFromWKB se utiliza para devolver los puntos de la columna WKB. La tabla de puntos de muestra tiene una columna de geometría, en donde los puntos se almacenan, y una columna wkb, en donde las representaciones binarias conocidas de puntos se almacenan.

En la declaración SELECT, la función ST_PointFromWKB se usa para recuperar los puntos de la columna WKB.

Copyright © 1995-2023 Esri. All rights reserved.

Oracle

```
CREATE TABLE sample_points (
 id integer,
 geometry sde.st_point,
 wkb blob
);
INSERT INTO SAMPLE_POINTS (id, geometry) VALUES (
 10,
 sde.st_point ('point (44 14)', 4326)
);
INSERT INTO SAMPLE_POINTS (id, geometry) VALUES (
 11,
 sde.st_point ('point (24 13)', 4326)
);
UPDATE SAMPLE_POINTS
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 10;
UPDATE SAMPLE_POINTS
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 11;
```

```
SELECT id, sde.st_astext (sde.st_pointfromwkb(wkb, 4326)) POINTS
 FROM SAMPLE_POINTS;
ID POINTS
10 POINT (44.00000000 14.00000000)
11 POINT (24.00000000 13.00000000)
```

PostgreSQL

```
CREATE TABLE sample_points (
 id integer,
 geometry sde.st_point,
 wkb bytea
);
INSERT INTO sample_points (id, geometry) VALUES (
 10,
 sde.st_point ('point (44 14)', 4326)
);
INSERT INTO sample_points (id, geometry) VALUES (
 11,
 sde.st_point ('point (24 13)', 4326)
);
UPDATE sample_points
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 10;
UPDATE sample_points
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 11;
```

```
SELECT id, sde.st_astext (sde.st_pointfromwkb(wkb, 4326))
 AS points
 FROM sample_points;
id points
10 POINT (44 14)
11 POINT (24 13)
```

Copyright © 1995-2023 Esri. All rights reserved.

SQLite

```
CREATE TABLE sample_pts (
 id integer,
 wkb blob
);
SELECT AddGeometryColumn(
 NULL,
 'sample_pts',
 'geometry',
 4326,
 'point',
 'xy',
 'null'
);
INSERT INTO sample_pts (id, geometry) VALUES (
 10,
 st_point ('point (44 14)', 4326)
);
INSERT INTO sample_pts (id, geometry) VALUES (
 11,
 st_point ('point (24 13)', 4326)
);
UPDATE sample_pts
 SET wkb = st_asbinary (geometry)
 WHERE id = 10;
UPDATE sample_pts
 SET wkb = st_asbinary (geometry)
 WHERE id = 11;
```

```
SELECT id, st_astext (st_pointfromwkb(wkb, 4326))
 AS "points"
 FROM sample_pts;
id points
10 POINT (44.00000000 14.00000000)
11 POINT (24.00000000 13.00000000)
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_PointN

Definición

ST_PointN toma un ST_LineString y un índice de enteros y devuelve un punto que es el enésimo vértice en la ruta de ST_LineString.

Sintaxis

Oracle y PostgreSQL

```
sde.st_pointn (line1 sde.st_linestring, index integer)
```

SQLite

```
st_pointn (line1 st_linestring, index int32)
```

Tipo de devolución

ST_Point

Ejemplo

La tabla pointn_test se crea con la columna gid, que identifica exclusivamente cada fila, y la columna de ST_LineString ln1. Las declaraciones INSERT insertan dos valores de cadena de texto de líneas. La primera cadena de texto de líneas no tienen coordenadas z o medidas, mientras que la segunda cadena de texto de líneas tiene ambas.

La consulta SELECT utiliza las funciones ST_PointN y ST_AsText para devolver el texto conocido del segundo vértice de cada cadena de líneas.

Oracle

```
CREATE TABLE pointn_test (
 gid integer,
 ln1 sde.st_geometry
);


INSERT INTO POINTN_TEST VALUES (
 1,
 sde.st_linefromtext ('linestring (10.02 20.01, 23.73 21.92, 30.10 40.23)', 4326)
);

INSERT INTO POINTN_TEST VALUES (
 2,
 sde.st_linefromtext ('linestring zm(10.02 20.01 5.0 7.0, 23.73 21.92 6.5 7.1, 30.10
40.23 6.9 7.2)', 4326)
);
```

```
SELECT gid, sde.st_astext (sde.st_pointn (ln1, 2)) The_2ndvertex
 FROM POINTN_TEST;
```

Copyright © 1995-2023 Esri. All rights reserved.

```
GID The_2ndvertex

1   POINT (23.73 21.92)
2   POINT ZM (23.73 21.92 6.5 7.1)
```

PostgreSQL

```
CREATE TABLE pointn_test (
 gid serial,
 ln1 sde.st_geometry
);


INSERT INTO pointn_test (ln1) VALUES (
 sde.st_linestring ('linestring (10.02 20.01, 23.73 21.92, 30.10 40.23)', 4326)
);

INSERT INTO pointn_test (ln1) VALUES (
 sde.st_linestring ('linestring zm(10.02 20.01 5.0 7.0, 23.73 21.92 6.5 7.1, 30.10
40.23 6.9 7.2)', 4326)
);
```

```
SELECT gid, sde.st_astext (sde.st_pointn (ln1, 2))
 AS The_2ndvertex
 FROM pointn_test;

gid the_2ndvertex

1   POINT (23.73 21.92)
2   POINT ZM (23.73 21.92 6.5 7.1)
```

Copyright © 1995-2023 Esri. All rights reserved.

SQLite

```
CREATE TABLE pointn_test (
 gid integer primary key autoincrement not null
);

SELECT AddGeometryColumn(
 NULL,
 'pointn_test',
 'ln1',
 4326,
 'linestringzm',
 'xyzm',
 'null'
);

INSERT INTO pointn_test (ln1) VALUES (
 st_linestring ('linestring (10.02 20.01, 23.73 21.92, 30.10 40.23)', 4326)
);

INSERT INTO pointn_test (ln1) VALUES (
 st_linestring ('linestring zm(10.02 20.01 5.0 7.0, 23.73 21.92 6.5 7.1, 30.10 40.23
6.9 7.2)', 4326)
);
```

```
SELECT gid, st_astext (st_pointn (ln1, 2))
 AS "Second Vertex"
 FROM pointn_test;

gid  Second Vertex

1   POINT ( 23.73000000 21.92000000)
2   POINT ZM ( 23.73000000 21.92000000 6.50000000 7.10000000)
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_PointOnSurface

Definición

ST_PointOnSurface toma un ST_Polygon o ST_MultiPolygon y devuelve un ST_Point garantizado para permanecer en su superficie.

Sintaxis

Oracle y PostgreSQL

```
sde.st_pointonsurface (polygon1 sde.st_geometry)
sde.st_pointonsurface (multipolygon1 sde.st_geometry)
```

SQLite

```
st_pointonsurface (polygon1 geometryblob)
st_pointonsurface (multipolygon1 geometryblob)
```

Tipo de devolución

ST_Point

Ejemplo

El ingeniero de la ciudad quiere crear un punto de etiqueta para la huella de cada edificio histórico. Las huellas de los edificios históricos se almacenan en la tabla hbuildings que se creó con la siguiente declaración CREATE TABLE:

La función ST_PointOnSurface genera un punto que se asegura que debe estar en la superficie de las huellas de edificios. La función ST_PointOnSurface devuelve un punto que la función ST_AsText convierte a texto para que use la aplicación.

Copyright © 1995-2023 Esri. All rights reserved.

Oracle

```
CREATE TABLE hbuildings (
 hbld_id integer,
 hbld_name varchar(40),
 footprint sde.st_geometry
);
```

```
INSERT INTO hbuildings (hbld_id, hbld_name, footprint) VALUES (
 1,
 'First National Bank',
 sde.st_polygon ('polygon ((0 0, 0 .010, .010 .010, .010 0, 0 0))', 4326)
);

INSERT INTO hbuildings (hbld_id, hbld_name, footprint) VALUES (
 2,
 'Courthouse',
 sde.st_polygon ('polygon ((.020 0, .020 .010, .030 .010, .030 0, .020 0))', 4326)
);
```

```
SELECT sde.st_astext (sde.st_pointonsurface (footprint)) Historic_Site
 FROM HBUILDINGS;

HISTORIC_SITE

POINT  (0.00500000 0.00500000)
POINT  (0.02500000 0.00500000)
```

PostgreSQL

```
CREATE TABLE hbuildings (
 hbld_id serial,
 hbld_name varchar(40),
 footprint sde.st_geometry
);
```

```
INSERT INTO hbuildings (hbld_name, footprint) VALUES (
 'First National Bank',
 sde.st_polygon ('polygon ((0 0, 0 .010, .010 .010, .010 0, 0 0))', 4326)
);

INSERT INTO hbuildings (hbld_name, footprint) VALUES (
 'Courthouse',
 sde.st_polygon ('polygon ((.020 0, .020 .010, .030 .010, .030 0, .020 0))', 4326)
);
```

```
SELECT sde.st_astext (sde.st_pointonsurface (footprint))
 AS "Historic Site"
 FROM hbuildings;

Historic Site
```

Copyright © 1995-2023 Esri. All rights reserved.

```
POINT  (0.00500000 0.00500000)
POINT  (0.02500000 0.00500000)
```

SQLite

```
CREATE TABLE hbuildings (
 hbld_id integer primary key autoincrement not null,
 hbld_name text(40)
);

SELECT AddGeometryColumn(
 NULL,
 'hbuildings',
 'footprint',
 4326,
 'polygon',
 'xy',
 'null'
);
```

```
INSERT INTO hbuildings (hbld_name, footprint) VALUES (
 'First National Bank',
 st_polygon ('polygon ((0 0, 0 .010, .010 .010, .010 0, 0 0))', 4326)
);

INSERT INTO hbuildings (hbld_name, footprint) VALUES (
 'Courthouse',
 st_polygon ('polygon ((.020 0, .020 .010, .030 .010, .030 0, .020 0))', 4326)
);
```

```
SELECT st_astext (st_pointonsurface (footprint))
 AS "Historic Site"
 FROM hbuildings;

Historic Site

POINT  (0.00500000 0.00500000)
POINT  (0.02500000 0.00500000)
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_PolyFromText

Nota:
Solo Oracle y SQLite

Definición

ST_GeomFromText toma una representación de texto conocida y una Id. de referencia espacial y devuelve un objeto ST_Geometry.

Sintaxis

Oracle

```
sde.st_polyfromtext (wkt clob, srid integer)
```

```
sde.st_polyfromtext (wkt clob)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

SQLite

```
st_polyfromtext (wkt text, srid int32)
```

```
st_polyfromtext (wkt text)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

Tipo de devolución

ST_Polygon

Ejemplo

La tabla polygon_test se crea con el única columna de polígono.

La declaración INSERT inserta un polígono en la columna de polígono utilizando la función ST_PolyFromText.

Oracle

```
CREATE TABLE polygon_test (pl1 sde.st_geometry);
```

```
INSERT INTO polygon_test VALUES (
 sde.st_polyfromtext ('polygon ((10.01 20.03, 10.52 40.11, 30.29 41.56, 31.78 10.74,
10.01 20.03))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

SQLite

```
CREATE TABLE polygon_test (id integer);
SELECT AddGeometryColumn(
 NULL,
 'polygon_test',
 'pl1',
 4326,
 'polygon',
 'xy',
 'null'
);
```

```
INSERT INTO polygon_test VALUES (
 1,
 st_polyfromtext ('polygon ((10.01 20.03, 10.52 40.11, 30.29 41.56, 31.78 10.74, 10.01
20.03))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_PolyFromWKB

Definición

ST_PolyFromWKB toma una representación binaria conocida (WKB) y una Id. de referencia espacial y devuelve un ST_Polygon.

Sintaxis

Oracle

```
sde.st_polyfromwkb (wkb blob, srid integer)
```

```
sde.st_polyfromwkb (wkb blob)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

PostgreSQL

```
sde.st_polyfromwkb (wkb bytea, srid integer)
```

SQLite

```
st_polyfromwkb (wkb blob, srid int32)
```

```
st_polyfromwkb (wkb blob)
```

Si no especifica ningún SRID, la referencia espacial tiene de forma predeterminada el valor 4326.

Tipo de devolución

ST_Polygon

Ejemplo

En este ejemplo se muestra cómo ST_PolyFromWKB se puede utilizar para crear un polígono desde su representación binaria conocida. La geometría es un polígono en el sistema de referencia espacial 4326. En este ejemplo, el polígono se almacena con el Id. = 1115 en la columna geometry de la tabla sample_polys y, a continuación, la columna wkb se actualiza con su representación WKB (utilizando la función ST_AsBinary). Por último, la función ST_PointFromWKB se utiliza para devolver el multipolígono de la columna WKB. La tabla sample_polys tiene una columna de geometría, en donde el polígono se almacena, y una columna wkb, en donde la representación WKB del polígono se almacena.

En la declaración SELECT, la función ST_PointFromWKB se usa para recuperar los puntos de la columna WKB.

Copyright © 1995-2023 Esri. All rights reserved.

Oracle

```
CREATE TABLE sample_polys (
 id integer,
 geometry sde.st_geometry,
 wkb blob
);
INSERT INTO SAMPLE_POLYS (id, geometry) VALUES (
 1115,
 sde.st_polyfromtext ('polygon ((10.01 20.03, 10.52 40.11, 30.29 41.56, 31.78 10.74,
10.01 20.03))', 4326)
);
UPDATE SAMPLE_POLYS
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 1115;
```

```
SELECT id, sde.st_astext (sde.st_polyfromwkb (wkb, 4326)) POLYS
 FROM SAMPLE_POLYS;
ID            POLYS
1115    POLYGON (10.01000000 20.03000000, 31.78000000 10.74000000, 30.29000000
41.56000000, 10.52000000 40.11000000, 10.01000000 20.03000000)
```

PostgreSQL

```
CREATE TABLE sample_polys (
 id integer,
 geometry sde.st_geometry,
 wkb bytea
);
INSERT INTO sample_polys (id, geometry) VALUES (
 1115,
 sde.st_polygon ('polygon  ((10.01 20.03, 10.52 40.11, 30.29 41.56, 31.78 10.74, 10.01
20.03))', 4326)
);
UPDATE sample_polys
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 1115;
```

```
SELECT id, sde.st_astext (sde.st_polyfromwkb (wkb, 4326))
 AS POLYS
 FROM sample_polys;
id    polys
1115    POLYGON (10.01000000 20.03000000, 31.78000000 10.74000000, 30.29000000
41.56000000, 10.52000000 40.11000000, 10.01000000 20.03000000)
```

SQLite

```
CREATE TABLE sample_polys(
 id integer,
 wkb blob
);
SELECT AddGeometryColumn(
 NULL,
 'sample_polys',
 'geometry',
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 4326,
 'polygon',
 'xy',
 'null'
);
INSERT INTO sample_polys (id, geometry) VALUES (
 1115,
 st_polyfromtext ('polygon ((10.01 20.03, 10.52 40.11, 30.29 41.56, 31.78 10.74, 10.01 
20.03))', 4326)
);
UPDATE sample_polys
 SET wkb = st_asbinary (geometry)
 WHERE id = 1115;
```

```
SELECT id, st_astext (st_polyfromwkb (wkb, 4326))
 AS "polygons"
 FROM sample_polys;
id    polygons
1115    POLYGON (10.01000000 20.03000000, 31.78000000 10.74000000, 30.29000000 
41.56000000, 10.52000000 40.11000000, 10.01000000 20.03000000)
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Polygon

Definición

La función del descriptor de acceso ST_Polygon toma una representación de texto conocido (WKT) y un Id. de referencia espacial (SRID) y genera un ST_Polygon.

Nota:
Cuando se crean tablas espaciales que se utilizarán con ArcGIS, es mejor crear la columna como supertipo de geometría (por ejemplo, ST_Geometry) en lugar de especificar un subtipo ST_Geometry.

Sintaxis

Oracle

```
sde.st_polygon (wkt clob, srid integer)
```

PostgreSQL

```
sde.st_polygon (wkt clob, srid integer)
sde.st_polygon (esri_shape bytea, srid integer)
```

SQLite

```
st_polygon (wkt text, srid int32)
```

Tipo de devolución

ST_Polygon

Ejemplo

La siguiente sentencia `CREATE TABLE` crea las tablas polygon_test, que tienen una sola columna, p1. La sentencia `INSERT` subsiguiente convierte un anillo (un polígono cerrado y simple) en un ST_Polygon y lo inserta en la columna p1 utilizando la función ST_Polygon.

Oracle

```
CREATE TABLE polygon_test (p1 sde.st_geometry);

INSERT INTO polygon_test VALUES (
 sde.st_polygon ('polygon ((10.01 20.03, 20.94 21.34, 35.93 10.04, 10.01 20.03))', 4326)
);
```

PostgreSQL

```
CREATE TABLE polygon_test (p1 sde.st_geometry);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO polygon_test VALUES (
 sde.st_polygon ('polygon ((10.01 20.03, 20.94 21.34, 35.93 10.04, 10.01 20.03))', 4326)
);
```

SQLite

```
CREATE TABLE poly_test (id integerp1 geometryblob);

SELECT AddGeometryColumn(
 NULL,
 'poly_test',
 'p1',
 4326,
 'polygon',
 'xy',
 'null'
);

INSERT INTO poly_test VALUES (
 1,
 st_polygon ('polygon ((10.01 20.03, 20.94 21.34, 35.93 10.04, 10.01 20.03))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Relate

Definición

ST_Relate compara dos geometrías y devuelve 1 (Oracle y SQLite) o t (PostgreSQL) si las geometrías cumplen las condiciones especificadas por la cadena de caracteres de la matriz de patrón DE-9IM; de lo contrario, devuelve 0 (Oracle y SQLite) o f (PostgreSQL).

Existe una segunda opción al utilizar ST_Relate en SQLite y Oracle: puede comparar dos geometrías para obtener una cadena de caracteres que represente la matriz de patrón DE-9IM que defina la relación de las geometrías entre sí.

Sintaxis

Oracle

Opción 1

```
sde.st_relate (geometry1 sde.st_geometry, geometry2 sde.st_geometry, patternMatrix
string)
```

Opción 2

```
sde.st_relate (geometry1 sde.st_geometry, geometry2 sde.st_geometry)
```

PostgreSQL

```
sde.st_relate (geometry1 sde.st_geometry, geometry2 sde.st_geometry, patternMatrix
string)
```

SQLite

Opción 1

```
st_relate (geometry1 geometryblob, geometry2 geometryblob, patternMatrix string)
```

Opción 2

```
st_relate (geometry1 geometryblob, geometry2 geometryblob)
```

Tipo de devolución

Se devuelve un booleano para PostgreSQL.

La opción 1 para SQLite y Oracle devuelve un entero.

La opción 2 para SQLite y Oracle devuelve una cadena de caracteres.

Copyright © 1995-2023 Esri. All rights reserved.

Ejemplos

Una matriz de patrón DE-9IM es un dispositivo para comparar geometrías. Hay varios tipos de matrices de este tipo. Por ejemplo, puede usar la función ST_Relate y la matriz de patrón equals (T*F**FFF*) para averiguar si dos geometrías son iguales, pero también puede proporcionar el patrón DE-9IM (1*F**FFF*). Con el último patrón, ST_Relate le indicará si dos geometrías son iguales con la primera posición, que indica si los interiores de la intersección de ambas geometrías son una línea (dimensión de 1).

En estos ejemplos, se crea una tabla, relate_test, con tres columnas espaciales y se insertan entidades de punto en cada una de ellas. La función ST_Relate se usa en la sentencia `SELECT` para comprobar si los puntos son iguales.

Si desea determinar si las geometrías son iguales y no necesita buscar la dimensionalidad de la relación, utilice en su lugar la función ST_Equals.

Oracle

El primer ejemplo muestra la primera opción de ST_Relate, que compara geometrías según una matriz de patrón DE-9IM para devolver 1 si las geometrías cumplen los requisitos definidos en la matriz o 0 si no los cumplen.

```
CREATE TABLE relate_test (
 id NUMBER GENERATED ALWAYS AS IDENTITY minvalue 0,
 g1 sde.st_geometry
);

CREATE TABLE relate_test2 (
 id NUMBER GENERATED ALWAYS AS IDENTITY minvalue 0,
 g2 sde.st_geometry
);

CREATE TABLE relate_test3 (
 id NUMBER GENERATED ALWAYS AS IDENTITY minvalue 0,
 g3 sde.st_geometry
);
```

```
INSERT INTO relate_test (g1) VALUES (sde.st_geometry ('point (10.02 20.01)', 4326));

INSERT INTO relate_test2 (g2) VALUES (sde.st_geometry ('point (10.02 20.01)', 4326));

INSERT INTO relate_test3 (g3) VALUES (sde.st_geometry ('point (30.01 20.01)', 4326));
```

```
SELECT sde.st_relate (relate_test.g1, relate_test2.g2, 'T*F**FFF*') AS "g1=g2",
       sde.st_relate (relate_test.g1, relate_test3.g3, 'T*F**FFF*') AS "g1=g3",
       sde.st_relate (relate_test2.g2, relate_test3.g3, 'T*F**FFF*') AS "g2=g3"
FROM relate_test, relate_test2, relate_test3;
```

Devuelve lo siguiente:

```
g1=g2     g1=g3      g2=g3

1                0                0
```

Copyright © 1995-2023 Esri. All rights reserved.

Este ejemplo muestra la segunda opción. Compara dos geometrías y devuelve la matriz de patrón DE-9IM.

```
SELECT sde.st_relate (relate_test.g1,relate_test2.g2) AS "g1 rel g2"
 FROM relate_test, relate_test2;
```

Devuelve lo siguiente:

```
g1 rel g2
0FFFFFFF2
```

PostgreSQL

El ejemplo compara geometrías según una matriz de patrón DE-9IM para devolver t si las geometrías cumplen los requisitos definidos en la matriz o f si no los cumplen.

```
CREATE TABLE relate_test (
 id SERIAL,
 g1 sde.st_geometry
);

CREATE TABLE relate_test2 (
 id SERIAL,
 g2 sde.st_geometry
);

CREATE TABLE relate_test3 (
 id SERIAL,
 g3 sde.st_geometry
);
```

```
INSERT INTO relate_test(g1) VALUES (sde.st_geometry ('point (10.02 20.01)', 4326));

INSERT INTO relate_test2 (g2) VALUES (sde.st_geometry ('point (10.02 20.01)', 4326));

INSERT INTO relate_test3 (g3) VALUES (sde.st_geometry ('point (30.01 20.01)', 4326));
```

```
SELECT sde.st_relate (relate_test.g1, relate_test2.g2, 'T*F**FFF*') AS "g1=g2",
       sde.st_relate (relate_test.g1, relate_test3.g3, 'T*F**FFF*') AS "g1=g3",
       sde.st_relate (relate_test2.g2, relate_test3.g3, 'T*F**FFF*') AS "g2=g3"
FROM relate_test, relate_test2, relate_test3;
```

Devuelve lo siguiente:

```
g1=g2     g1=g3     g2=g3

t          f         f
```

Copyright © 1995-2023 Esri. All rights reserved.

SQLite

El primer ejemplo muestra la primera opción de ST_Relate, que compara geometrías según una matriz de patrón DE-9IM para devolver 1 si las geometrías cumplen los requisitos definidos en la matriz o 0 si no los cumplen.

```
CREATE TABLE relate_test (id integer primary key autoincrement not null);

SELECT AddGeometryColumn(
 NULL,
 'relate_test',
 'g1',
 4326,
 'point',
 'xy',
 'null'
);

CREATE TABLE relate_test2 (id integer primary key autoincrement not null);

SELECT AddGeometryColumn(
 NULL,
 'relate_test2',
 'g2',
 4326,
 'point',
 'xy',
 'null'
);

CREATE TABLE relate_test3 (id integer primary key autoincrement not null);

SELECT AddGeometryColumn(
 NULL,
 'relate_test3',
 'g3',
 4326,
 'point',
 'xy',
 'null'
);
```

```
INSERT INTO relate_test (g1) VALUES (
 st_geometry ('point (10.02 20.01)', 4326)
);

INSERT INTO relate_test2 (g2) VALUES (
 st_geometry ('point (10.02 20.01)', 4326)
);

INSERT INTO relate_test3 (g3) VALUES (
 st_geometry ('point (30.01 20.01)', 4326)
);
```

```
SELECT st_relate (relate_test.g1, relate_test2.g2, 'T*F**FFF*') AS "g1=g2",
       st_relate (relate_test.g1, relate_test3.g3, 'T*F**FFF*') AS "g1=g3",
       st_relate (relate_test2.g2, relate_test3.g3, 'T*F**FFF*') AS "g2=g3"
 FROM relate_test, relate_test2, relate_test3;
```

Copyright © 1995-2023 Esri. All rights reserved.

Devuelve lo siguiente:

```
g1=g2     g1=g3     g2=g3

1          0         0
```

Este ejemplo muestra la segunda opción. Compara dos geometrías y devuelve la matriz de patrón DE-9IM.

```
SELECT st_relate (relate_test.g1,relate_test2.g2) AS "g1 rel g2"
 FROM relate_test, relate_test2;
```

Devuelve lo siguiente:

```
g1 rel g2
0FFFFFFF2
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_SRID

Definición

ST_SRID toma un objeto de geometría y devuelve el Id. de referencia espacial.

Sintaxis

Oracle y PostgreSQL

```
sde.st_srid (geometry1 sde.st_geometry)
```

SQLite

```
st_srid (geometry1 geometryblob)
```

Tipo de devolución

Entero

Ejemplos

En la siguiente tabla se crea:

En la siguiente declaración, una geometría de punto ubicada en la coordenada (10.01, 50.76) se inserta en la columna de geometría g1. Cuando se crea la geometría de punto, se le asigna el valor SRID de 4326.

La función ST_SRID devuelve la Id. de referencia espacial de la geometría introducida recientemente.

Oracle

```
CREATE TABLE srid_test (g1 sde.st_geometry);
```

```
INSERT INTO SRID_TEST VALUES (
 sde.st_geometry ('point (10.01 50.76)', 4326)
);
```

```
SELECT sde.st_srid (g1) SRID_G1
 FROM SRID_TEST;

SRID_G1

4326
```

Copyright © 1995-2023 Esri. All rights reserved.

PostgreSQL

```
CREATE TABLE srid_test (g1 sde.st_geometry);
```

```
INSERT INTO srid_test VALUES (
 sde.st_point ('point (10.01 50.76)', 4326)
);
```

```
SELECT sde.st_srid (g1)
 AS SRID_G1
 FROM srid_test;

srid_g1

4326
```

SQLite

```
CREATE TABLE srid_test (id integer);

SELECT AddGeometryColumn(
 NULL,
 'srid_test',
 'g1',
 4326,
 'point',
 'xy',
 'null'
);
```

```
INSERT INTO srid_test VALUES (
 1,
 st_point ('point (10.01 50.76)', 4326)
);
```

```
SELECT st_srid (g1)
 AS "SRID"
 FROM srid_test;

SRID

4326
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_StartPoint

Definición

ST_StartPoint devuelve el primer punto de una cadena de texto de líneas

Sintaxis

Oracle y PostgreSQL

```
sde.st_startpoint (ln1 sde.st_geometry)
```

SQLite

```
st_startpoint (ln1 geometryblob)
```

Tipo de devolución

ST_Point

Ejemplos

La tabla startpoint_test se crea con la columna de enteros gid, que identifica exclusivamente las filas de la tabla, y la columna de ST_LineString ln1.

Las declaraciones INSERT insertan la ST_LineStrings en la columna ln1. El primer ST_LineString no tiene coordenadas z o medidas, mientras que la segunda ST_LineString tiene ambas.

La función ST_StartPoint extrae el primer punto de cada ST_LineString. El primer punto de la lista no tiene una coordenada z o de medida, mientras que el segundo punto tiene ambas, ya que la cadena de texto de líneas de origen las tiene.

Copyright © 1995-2023 Esri. All rights reserved.

Oracle

```
CREATE TABLE startpoint_test (
 gid integer,
 ln1 sde.st_geometry
);
```

```
INSERT INTO STARTPOINT_TEST VALUES (
 1,
 sde.st_linefromtext ('linestring (10.02 20.01, 23.73 21.92, 30.10 40.23)', 4326)
);

INSERT INTO STARTPOINT_TEST VALUES (
 2,
 sde.st_linefromtext ('linestring zm(10.02 20.01 5 7, 23.73 21.92 6.5 7.1, 30.10 40.23
6.9 7.2)', 4326)
);
```

```
SELECT gid, sde.st_astext (sde.st_startpoint (ln1)) Startpoint
 FROM STARTPOINT_TEST;

GID  Startpoint

1    POINT (10.02000000 20.01000000)
2    POINT ZM (10.02000000 20.01000000 5.00000000 7.00000000)
```

PostgreSQL

```
CREATE TABLE startpoint_test (
 gid serial,
 ln1 sde.st_geometry
);
```

```
INSERT INTO startpoint_test (ln1) VALUES (
 sde.st_linestring ('linestring (10.02 20.01, 23.73 21.92, 30.10 40.23)', 4326)
);

INSERT INTO startpoint_test (ln1) VALUES (
 sde.st_linestring ('linestring zm(10.02 20.01 5 7, 23.73 21.92 6.5 7.1, 30.10 40.23
6.9 7.2)', 4326)
);
```

```
SELECT gid, sde.st_astext (sde.st_startpoint (ln1))
 AS Startpoint
 FROM startpoint_test;

gid  startpoint

1    POINT (10.02000000 20.01000000)
2    POINT ZM (10.02000000 20.01000000 5.00000000 7.00000000)
```

Copyright © 1995-2023 Esri. All rights reserved.

SQLite

```
CREATE TABLE startpoint_test (
 gid integer primary key autoincrement not null
);

SELECT AddGeometryColumn(
 NULL,
 'startpoint_test',
 'ln1',
 4326,
 'linestringzm',
 'xyzm',
 'null'
);
```

```
INSERT INTO startpoint_test (ln1) VALUES (
 st_linestring ('linestring (10.02 20.01, 23.73 21.92, 30.10 40.23)', 4326)
);

INSERT INTO startpoint_test(ln1) VALUES (
 st_linestring ('linestring zm(10.02 20.01 5 7, 23.73 21.92 6.5 7.1, 30.10 40.23 6.9
7.2)', 4326)
);
```

```
SELECT gid, st_astext (st_startpoint (ln1))
 AS "Startpoint"
 FROM startpoint_test;

gid  Startpoint

1    POINT (10.02000000 20.01000000)
2    POINT ZM (10.02000000 20.01000000 5.00000000 7.00000000)
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Surface

Nota:
Solo Oracle y SQLite

Definición

ST_Surface construye una entidad de superficie a partir de una representación de texto conocido. Las superficies son similares a los polígonos, pero tienen valores en cada punto de su extensión.

Sintaxis

Oracle

```
sde.st_surface (wkt clob, srid integer)
```

SQLite

```
st_surface (wkt text, srid int32)
```

Tipo de devolución

ST_Polygon

Ejemplo

Se crea una tabla, surf_test, y se inserta en ella una geometría de superficie.

Oracle

```
CREATE TABLE surf_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO SURF_TEST VALUES (
 1110,
 sde.st_surface ('polygon ((110 120, 110 140, 120 130, 110 120))', 4326)
);
```

SQLite

```
CREATE TABLE surf_test (
 id integer
);

SELECT AddGeometryColumn(
 NULL,
 'surf_test',
 'geometry',
 4326,
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 'polygon',
 'xy',
 'null'
);

INSERT INTO SURF_TEST VALUES (
 1110,
 st_surface ('polygon ((110 120, 110 140, 120 130, 110 120))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_SymmetricDiff

Definición

ST_SymmetricDiff toma dos objetos de geometría y devuelve un objeto de geometría compuesto por las partes de los objetos de origen que no son comunes a ambos.

Sintaxis

Oracle y PostgreSQL

```
sde.st_symmetricdiff (geometry1 sde.st_geometry, geometry2 sde.st_geometry)
```

SQLite

```
st_symmetricdiff (geometry1 geometryblob, geometry2 geometryblob)
```

Tipo de devolución

Oracle y PostgreSQL

ST_Geometry

SQLite

Geometryblob

Ejemplo

Para un informe especial, el supervisor del condado debe determinar las áreas de cuencas hidrográficas y de radios con columnas de humo peligrosas que no se intersecan.

La tabla watershed contiene una columna de Id., una columna para almacenar el nombre de la cuenca hidrográfica (wname) y una columna de forma que almacena la geometría del área de la cuenca hidrográfica.

La tabla plumes almacena la identidad del sitio en columna Id., mientras que la ubicación geográfica de cada sitio se almacena en la columna de punto de sitio.

La función ST_Buffer genera una zona de influencia que rodea los puntos del sitio de residuos peligrosos. La función ST_SymmetricDiff devuelve los polígonos de las zonas de influencia de los sitios de residuos peligrosos y las cuencas hidrográficas que no se intersecan.

La diferencia simétrica de los sitios de residuos peligrosos y las cuencas hidrográficas da como resultado la sustracción de las áreas intersecadas.

Oracle

```
CREATE TABLE watershed (
 id integer,
 wname varchar(40),
 shape sde.st_geometry
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
CREATE TABLE plumes (
 id integer,
 site sde.st_geometry
);
```

```
INSERT INTO WATERSHED (ID, WNAME, SHAPE) VALUES (
 1,
 'Big River',
 sde.st_geometry ('polygon ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO WATERSHED (ID, WNAME, SHAPE) VALUES (
 2,
 'Lost Creek',
 sde.st_geometry ('polygon ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
);

INSERT INTO WATERSHED (ID, WNAME, SHAPE) VALUES (
 3,
 'Szymborska Stream',
 sde.st_geometry ('polygon ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
);

INSERT INTO PLUMES (ID, SITE) VALUES (
 20,
 sde.st_geometry ('point (60 60)', 4326)
);

INSERT INTO PLUMES (ID, SITE) VALUES (
 21,
 sde.st_geometry ('point (30 30)', 4326)
);
```

```
SELECT ws.id WS_ID,
 sde.st_area (sde.st_symmetricdiff (sde.st_buffer (p.site, .1), ws.shape)) AREA_NO_INT
 FROM PLUMES p, WATERSHED ws
 WHERE p.id = 20;

     SA_ID     AREA_NO_INT

         1     100.031393
         2     400.031393
         3     400.015697
```

Copyright © 1995-2023 Esri. All rights reserved.

PostgreSQL

```
CREATE TABLE watershed (
 id serial,
 wname varchar(40),
 shape sde.st_geometry
);

CREATE TABLE plumes (
 id serial,
 site sde.st_geometry
);
```

```
INSERT INTO watershed (wname, shape) VALUES (
 'Big River',
 sde.st_geometry ('polygon ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO watershed (wname, shape) VALUES (
 'Lost Creek',
 sde.st_geometry ('polygon ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
);

INSERT INTO watershed (wname, shape) VALUES (
 'Szymborska Stream',
 sde.st_geometry ('polygon ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
);

INSERT INTO plumes (site) VALUES (
 sde.st_geometry ('point (60 60)', 4326)
);

INSERT INTO plumes (site) VALUES (
 sde.st_geometry ('point (30 30)', 4326)
);
```

```
SELECT ws.id AS WS_ID,
 sde.st_area (sde.st_symmetricdiff (sde.st_buffer (p.site, .1), ws.shape)) AS "no
intersection"
 FROM plumes p, watershed ws
 WHERE p.id = 1;

    ws_id     no intersection

         1    100.031393502001
         2    400.031393502001
         3    400.01569751
```

SQLite

```
CREATE TABLE watershed (
 id integer primary key autoincrement not null,
 wname text(40)
);

SELECT AddGeometryColumn(
 NULL,
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 'watershed',
 'shape',
 4326,
 'polygon',
 'xy',
 'null'
);

CREATE TABLE plumes (
 id integer primary key autoincrement not null
);

SELECT AddGeometryColumn(
 NULL,
 'plumes',
 'site',
 4326,
 'point',
 'xy',
 'null'
);
```

```
INSERT INTO watershed (wname, shape) VALUES (
 'Big River',
 st_geometry ('polygon ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO watershed (wname, shape) VALUES (
 'Lost Creek',
 st_geometry ('polygon ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
);

INSERT INTO watershed (wname, shape) VALUES (
 'Szymborska Stream',
 st_geometry ('polygon ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
);

INSERT INTO plumes (site) VALUES (
 st_geometry ('point (60 60)', 4326)
);

INSERT INTO plumes (site) VALUES (
 st_geometry ('point (30 30)', 4326)
);
```

```
SELECT ws.id AS WS_ID,
 st_area (st_symmetricdiff (st_buffer (p.site, .1), ws.shape)) AS "no intersection"
 FROM plumes p, watershed ws
 WHERE p.id = 1;

    WS_ID     no intersection

         1    400.031393502001
         2    100.031393502001
         3    400.01569751
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Touches

Definición

ST_Touches devuelve 1 (Oracle y SQLite) o t (PostgreSQL) si ninguno de los puntos comunes a ambas geometrías se interseca con el interior de ambas geometrías, de lo contrario, devuelve 0 (Oracle y SQLite) o f (PostgreSQL). Al menos una geometría debe ser ST_LineString, ST_Polygon, ST_MultiLineString o ST_MultiPolygon.

Sintaxis

Oracle y PostgreSQL

```
sde.st_touches (geometry1 sde.st_geometry, geometry2 sde.st_geometry)
```

SQLite

```
st_touches (geometry1 geometryblob, geometry2 geometryblob)
```

Tipo de devolución

Booleano

Ejemplo

El jefe del técnico de SIG le solicito proporcionar una lista de todas las líneas de alcantarillado que poseen los extremos que se intersecan con otra línea de alcantarillado.

La tabla de líneas de alcantarillado se crea con tres columnas. La primera columna, sewer_id, identifica exclusivamente cada línea de alcantarillado. La columna de clase de enteros identifica el tipo de línea de alcantarillado generalmente asociada con la capacidad de la línea. La columna de alcantarillado almacena la geometría de la línea de alcantarillado.

La consulta SELECT usa la función ST_Touches para devolver una lista de las alcantarillas que se tocan entre sí.

Oracle

```
CREATE TABLE sewerlines (
 sewer_id integer,
 sewer sde.st_geometry
);

INSERT INTO SEWERLINES VALUES (
 1,
 sde.st_mlinefromtext ('multilinestring ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO SEWERLINES VALUES (
 2,
 sde.st_mlinefromtext ('multilinestring ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
);

INSERT INTO SEWERLINES VALUES (
 3,
 sde.st_mlinefromtext ('multilinestring ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
```

Copyright © 1995-2023 Esri. All rights reserved.

```
);

INSERT INTO SEWERLINES VALUES (
 4,
 sde.st_linestring ('linestring (60 60, 70 70)', 4326)
);

INSERT INTO SEWERLINES VALUES (
 5,
 sde.st_linestring ('linestring (30 30, 60 60)', 4326)
);
```

```
SELECT s1.sewer_id, s2.sewer_id
 FROM SEWERLINES s1, SEWERLINES s2
 WHERE sde.st_touches (s1.sewer, s2.sewer) = 1;

  SEWER_ID   SEWER_ID

         1          5
         3          4
         4          3
         4          5
         5          1
         5          3
         5          4
```

PostgreSQL

```
CREATE TABLE sewerlines (
 sewer_id serial,
 sewer sde.st_geometry);

INSERT INTO sewerlines (sewer) VALUES (
 sde.st_multilinestring ('multilinestring ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO sewerlines (sewer) VALUES (
 sde.st_multilinestring ('multilinestring ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
);

INSERT INTO sewerlines (sewer) VALUES (
 sde.st_multilinestring ('multilinestring ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
);

INSERT INTO sewerlines (sewer) VALUES (
 sde.st_linestring ('linestring (60 60, 70 70)', 4326)
);

INSERT INTO sewerlines (sewer) VALUES (
 sde.st_linestring ('linestring (30 30, 60 60)', 4326)
);
```

```
SELECT s1.sewer_id, s2.sewer_id
 FROM sewerlines s1, sewerlines s2
 WHERE sde.st_touches (s1.sewer, s2.sewer) = 't';

  SEWER_ID   SEWER_ID
```

Copyright © 1995-2023 Esri. All rights reserved.

```
         1          5
         3          4
         4          3
         4          5
         5          1
         5          3
         5          4
```

SQLite

```
CREATE TABLE sewerlines (
 sewer_id integer primary key autoincrement not null
);

SELECT AddGeometryColumn(
 NULL,
 'sewerlines',
 'sewer',
 4326,
 'geometry',
 'xy',
 'null'
);

INSERT INTO sewerlines (sewer) VALUES (
 st_multilinestring ('multilinestring ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO sewerlines (sewer) VALUES (
 st_multilinestring ('multilinestring ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
);

INSERT INTO sewerlines (sewer) VALUES (
 st_multilinestring ('multilinestring ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
);

INSERT INTO sewerlines (sewer) VALUES (
 st_linestring ('linestring (60 60, 70 70)', 4326)
);

INSERT INTO sewerlines (sewer) VALUES (
 st_linestring ('linestring (30 30, 60 60)', 4326)
);
```

```
SELECT s1.sewer_id, s2.sewer_id
 FROM SEWERLINES s1, SEWERLINES s2
 WHERE st_touches (s1.sewer, s2.sewer) = 1;

sewer_id   sewer_id

1          5
3          4
3          5
4          3
4          5
5          1
5          3
5          4
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Transform

Definición

ST_Transform toma como entrada datos de ST_Geometry bidimensionales y devuelve valores convertidos en la referencia espacial especificada por el ID de referencia espacial (SRID) que proporcione.

Precaución:

Si ha registrado la columna espacial en la base de datos de PostgreSQL usando la función st_register_spatial_column, el SRID en el momento del registro se escribe en la tabla sde_geometry_columns. Si ha creado un índice espacial en la columna espacial de una base de datos de Oracle, el SRID en el momento de la creación del índice espacial se escribe en la tabla st_geometry_columns. Usar ST_Transform para modificar el SRID de los datos ST_Geometry no actualiza el SRID en la tabla sde_geometry_columns o st_geometry_columns.

Cuando los sistemas de coordenadas geográficas son diferentes, ST_Transform realiza una transformación geográfica. Una transformación geográfica se convierte entre dos sistemas de coordenadas geográficas. Una transformación geográfica se define en una dirección determinada, por ejemplo de NAD 1927 a NAD 1983, pero la función ST_Transform aplicará correctamente la transformación sin importar cuáles son los sistemas de coordenadas de origen y de destino.

Los métodos de transformación geográfica se pueden dividir en dos tipos: basados en ecuación y basados en archivo. Los métodos basados en la ecuación son independientes no necesitan ninguna información externa. Los métodos basados en archivos utilizan archivos de disco para calcular los valores de desplazamiento. Son normalmente más precisos que los métodos basados en la ecuación. Los métodos basados en archivos se utilizan comúnmente en Australia, Canadá, Nueva Zelanda, Alemania, España y Estados Unidos. Los archivos (excepto los canadienses) se pueden obtener de una instalación de ArcGIS Pro o directamente de los distintos organismos cartográficos nacionales.

Para admitir las transformaciones basadas en archivos, debe colocar los archivos en el servidor donde esté instalada la base de datos, en la misma estructura de carpetas relativa que la carpeta pedata del directorio de instalación de ArcGIS Pro.

Por ejemplo, existe una carpeta llamada `pedata` en la carpeta `Resources` del directorio de instalación de ArcGIS Pro. La carpeta `pedata` contiene varias subcarpetas, pero las tres carpetas que contienen métodos compatibles basados en archivos son `harn`, `nadcon` y `ntv2`. Copie la carpeta `pedata` y su contenido desde el directorio de instalación de ArcGIS en el servidor de la base de datos o cree un directorio en el servidor de la base de datos que contenga los subdirectorios y archivos del método de transformación basado en archivos compatible. Una vez que los archivos estén en el servidor de la base de datos, establezca una variable de entorno del sistema operativo llamada PEDATAHOME en el mismo servidor. Defina la variable PEDATAHOME en la ubicación del directorio que contiene los subdirectorios y los archivos; por ejemplo, si la carpeta pedata se copia en `C:\pedata` en un servidor de Microsoft Windows, defina la variable de entorno PEDATAHOME como `C:\pedata`.

Consulte la documentación de su sistema operativo para obtener información sobre cómo establecer una variable de entorno.

Después de establecer PEDATAHOME, debe iniciar una nueva sesión de SQL para poder utilizar la función ST_Transform. Sin embargo, no es necesario reiniciar el servidor.

Copyright © 1995-2023 Esri. All rights reserved.

Usar ST_Transform con PostgreSQL

En PostgreSQL, puede realizar conversiones entre referencias espaciales que tengan el mismo sistema de coordenadas geográficas o uno diferente.

Si los datos están almacenados en una base de datos (y no en una geodatabase), haga lo siguiente para cambiar la referencia espacial de los datos ST_Geometry cuando los sistemas de coordenadas geográficas sean los mismos:

1. Cree una copia de seguridad de la tabla.
2. Cree una segunda columna (de destino) ST_Geometry en la tabla.
3. Registre la columna ST_Geometry de destino especificando el nuevo SRID.
   Esto especifica la referencia espacial de la columna mediante la inserción de un registro en la tabla de sistema sde_geometry_columns.
4. Ejecute la función ST_Transform y especifique que los datos transformados se incluyan en la columna ST_Geometry de destino.
5. Anule el registro de la primera columna ST_Geometry (origen).

Si los datos están almacenados en una geodatabase, debe usar las herramientas de ArcGIS para volver a proyectar los datos en una nueva clase de entidad. Ejecutar ST_Transform en una clase de entidad de geodatabase anula la función de actualización de las tablas de sistema de la geodatabase con el nuevo SRID.

Usar ST_Transform con Oracle

En Oracle, puede realizar conversiones entre referencias espaciales que tengan el mismo sistema de coordenadas geográficas o uno diferente.

Si los datos están almacenados en una base de datos (y no en una geodatabase) y no se ha definido ningún índice espacial en la columna espacial, puede agregar una segunda columna ST_Geometry y transferir a esa columna los datos transformados. Puede conservar tanto la columna ST_Geometry original (origen) como la columna ST_Geometry de destino en la tabla, aunque solo se puede mostrar una de ellas en ArcGIS usando una vista o modificando la definición de la capa de consulta para la tabla.

Si los datos están almacenados en una base de datos (y no en una geodatabase) y la columna espacial tiene definido un índice espacial, no puede conservar la columna ST_Geometry original. Una vez que se ha definido un índice espacial en una columna ST_Geometry, el SRID se escribe en la tabla de metadatos st_geometry_columns. ST_Transform no actualiza esa tabla.

1. Cree una copia de seguridad de la tabla.
2. Cree una segunda columna (de destino) ST_Geometry en la tabla.
3. Ejecute la función ST_Transform y especifique que los datos transformados se incluyan en la columna ST_Geometry de destino.
4. Elimine el índice espacial de la columna ST_Geometry de origen.
5. Elimine la columna ST_Geometry de origen.
6. Cree un índice espacial en la columna ST_Geometry de destino.

Si los datos están almacenados en una geodatabase, debe utilizar las herramientas de ArcGIS para reproyectar los datos en una nueva clase de entidad. Ejecutar ST_Transform en una clase de entidad de geodatabase anula la

Copyright © 1995-2023 Esri. All rights reserved.

función de actualización de las tablas de sistema de la geodatabase con el nuevo SRID.

Usar ST_Transform con SQLite

En SQLite, puede realizar conversiones entre referencias espaciales que tengan el mismo sistema de coordenadas geográficas o uno diferente.

Sintaxis

Las referencias espaciales de origen y de destino tienen el mismo sistema de coordenadas geográficas.

Oracle y PostgreSQL

```
sde.st_transform (geometry1 sde.st_geometry, srid integer)
```

SQLite

```
st_transform (geometry1 geometryblob, srid in32)
```

Las referencias espaciales de origen y de destino no tienen el mismo sistema de coordenadas geográficas.

Oracle

```
sde.st_transform (g1 sde.st_geometry, srid integer, geogtrans_id integer)
```

PostgreSQL

Opción 1: `sde.st_transform (g1 sde.st_geometry, srid int)`

Opción 2: `sde.st_transform (g1 sde.st_geometry, srid int, [geogtrans_id int])`

Opción 3: `sde.st_transform (g1 sde.st_geometry, srid int, [extent double] [prime meridian double] [unit conversion factor double])`

En la opción 3, puede especificar opcionalmente la extensión como una lista separada por comas de coordenadas en el siguiente orden: coordenada X inferior izquierda, coordenada Y inferior izquierda, coordenada X superior derecha y coordenada Y superior derecha. Si no especifica ninguna extensión, ST_Transform utiliza una extensión más grande y general.

Al especificar una extensión, los parámetros de meridiano base y de factor de conversión de unidad son opcionales. Solo tiene que proporcionar esta información si los valores de extensión que especifique no utilizan el meridiano base de Greenwich o grados decimales.

SQLite

```
st_transform (geometry1 geometryblob, srid int32, geogtrans_id int32)
```

Tipo de devolución

Copyright © 1995-2023 Esri. All rights reserved.

Oracle y PostgreSQL

ST_Geometry

SQLite

Geometryblob

Ejemplos

Transformar datos cuando las referencias espaciales de origen y de destino tienen el mismo sistema de coordenadas geográficas

El siguiente ejemplo crea una tabla, transform_test, que tiene dos columnas de cadena de líneas: ln1 y ln2. Se inserta una línea en ln1 con el SRID 4326. La función ST_Transform se utiliza a continuación en una sentencia `UPDATE` para tomar la cadena de líneas de ln1, convertirla de la referencia de coordenadas asignada al SRID 4326 a la referencia de coordenadas asignada al SRID 3857 y colocarla en la columna ln2.

Nota:
Los SRID 4326 y 3857 tienen el mismo datum geográfico.

Oracle

```
CREATE TABLE transform_test (
 ln1 sde.st_geometry,
 ln2 sde.st_geometry);

INSERT INTO transform_test (ln1) VALUES (
 sde.st_geometry ('linestring (10.01 40.03, 92.32 29.39)', 4326)
);
```

```
UPDATE transform_test
 SET ln2 = sde.st_transform (ln1, 3857);
```

PostgreSQL

```
CREATE TABLE transform_test (
 ln1 sde.st_geometry,
 ln2 sde.st_geometry);

INSERT INTO transform_test (ln1) VALUES (
 sde.st_geometry ('linestring (10.01 40.03, 92.32 29.39)', 4326)
);
```

```
UPDATE transform_test
 SET ln2 = sde.st_transform (ln1, 3857);
```

Copyright © 1995-2023 Esri. All rights reserved.

SQLite

```
CREATE TABLE transform_test (id integer);

SELECT AddGeometryColumn(
 NULL,
 'transform_test',
 'ln1',
 4326,
 'linestring',
 'xy',
 'null'
);

INSERT INTO transform_test (ln1) VALUES (
 st_geometry ('linestring (10.01 40.03, 92.32 29.39)', 4326)
);
```

```
UPDATE transform_test
 SET ln1 = st_transform (ln1, 3857);
```

Transformar datos cuando las referencias espaciales de origen y de destino no tienen el mismo sistema de coordenadas geográficas

En el siguiente ejemplo, se crea la tabla n27, que contiene una columna de Id. y una columna de geometría. Se inserta un punto en la tabla n27 con un SRID de 4267. El SRID 4267 utiliza el sistema de coordenadas geográficas NAD 1927.

A continuación, se crea la tabla n83 y la función ST_Transform se utiliza para insertar la geometría de la tabla n27 en la tabla n83 con un SRID de 4269 y un Id. de transformación geográfica de 1241. SRID 4269 utiliza el sistema de coordenadas geográficas NAD 1983 y 1241 es la Id. conocida de la transformación NAD_1927_To_NAD_1983_NADCON. Esta transformación se basa en archivos y se puede utilizar para los 48 estados contiguos de Estados Unidos.

Sugerencia:

Para obtener las listas de las transformaciones geográficas compatibles, consulte el artículo técnico de Esri 000004829 y los vínculos proporcionados en la sección **Información relacionada** del artículo.

Oracle

```
--Create table.
CREATE TABLE n27 (
 id integer,
 geometry sde.st_geometry
);

--Insert point with SRID 4267.
INSERT INTO N27 (id, geometry) VALUES (
 1,
 sde.st_geometry ('point (-123.0 49.0)', 4267)
);

--Create the n83 table as the destination table of the transformation.
```

Copyright © 1995-2023 Esri. All rights reserved.

```
CREATE TABLE n83 (
 id integer,
 geometry sde.st_geometry
);

--Run the transformation.
INSERT INTO N83 (id, geometry)(
 select c.id, sde.st_transform (c.geometry, 4269, 1241)
 from N27 c
);
```

Si se define una PEDATAHOME correctamente, la ejecución de una declaración SELECT contra la tabla n83 devolverá lo siguiente:

```
SELECT id, sde.st_astext (geometry) description
 FROM N83;

ID        DESCRIPTION
1 | POINT((-123.00130569 48.999828199))
```

PostgreSQL

```
--Option 1
--Gets geographic transformation from ST_Geometry libraries.
--Does not require you to provide a GTid.
--Performs an equation-based transformation between two geographic coordinate systems
--with different datums. (SRID 4267/DATUM NAD27 to SRID 4269/DATUM NAD 83)

--Provide point to transform.
SELECT sde.ST_AsText(sde.ST_Transform(
  sde.ST_Geometry('point (-155.7029 63.6096)',4267), 4269));

--Returns output in SRID 4269.
"POINT ( -155.70290000 63.60960000)"
```

```
--Option 2
--Example uses input point in SRID 3857(DATUM: WGS 1984)
--and geographic transformation ID (GTid) 1251.
--Transforms point to SRID 102008 (DATUM: NAD 83)

--Provide point to transform.
SELECT sde.ST_AsText(sde.ST_Transform(
  sde.ST_Geometry('point (-13244252.9404 4224702.5198)', 3857), 102008, 1251));

--Returns output in SRID 102008.
"POINT (-1957193.14740000 -297059.19680000)"
```

SQLite

```
--Create source table.
CREATE TABLE n27 (id integer);

SELECT AddGeometryColumn(
 NULL,
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 'n27',
 'geometry',
 4267,
 'point',
 'xy',
 'null'
);

--Insert point with SRID 4267.
INSERT INTO n27 (id, geometry) VALUES (
 1,
 st_geometry ('point (-123.0 49.0)', 4267)
);

--Create the n83 table as the destination table of the transformation.
CREATE TABLE n83 (id integer);

SELECT AddGeometryColumn(
 NULL,
 'n83',
 'geometry',
 4269,
 'point',
 'xy',
 'null'
);

--Run the transformation.
INSERT INTO n83 (id, geometry) VALUES (
 1,
 st_transform ((select geometry from n27 where id=1), 4269, 1241)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Union

Definición

ST_Union devuelve un objeto de geometría que es la combinación de dos objetos de origen.

Sintaxis

Oracle y PostgreSQL

```
sde.st_union (geometry1 sde.st_geometry, geometry2 sde.st_geometry)
```

SQLite

```
st_union (geometry1 geometryblob, geometry2 geometryblob)
```

Tipo de devolución

Oracle y PostgreSQL

ST_Geometry

SQLite

Geometryblob

Ejemplo

La tabla sensitive_areas almacena los Id. de las instituciones amenazadas junto con la columna de forma, que almacena las geometrías de polígono de las instituciones.

La tabla hazardous_sites almacena la identidad de los sitios en la id de la columna, mientras que la ubicación geográfica de cada sitio se almacena en la columna de punto de sitio.

La función ST_Buffer genera una zona de influencia que rodea los sitios de residuos peligrosos. La función ST_Union genera polígonos desde la unión de los sitios de residuos peligrosos de la zona de influencia y polígonos de área sensible. La función ST_Area devuelve el área de estos polígonos.

Oracle

```
CREATE TABLE sensitive_areas (
 id integer,
 shape sde.st_geometry
);

CREATE TABLE hazardous_sites (
 id integer,
 site sde.st_geometry
);


INSERT INTO SENSITIVE_AREAS VALUES (
 1,
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 sde.st_geometry ('polygon ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO SENSITIVE_AREAS VALUES (
 2,
 sde.st_geometry ('polygon ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
);

INSERT INTO SENSITIVE_AREAS VALUES (
 3,
 sde.st_geometry ('polygon ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
);

INSERT INTO HAZARDOUS_SITES VALUES (
 4,
 sde.st_geometry ('point (60 60)', 4326)
);

INSERT INTO HAZARDOUS_SITES VALUES (
 5,
 sde.st_geometry ('point (30 30)', 4326)
);
```

```
SELECT sa.id SA_ID, hs.id HS_ID,
 sde.st_area (sde.st_union (sde.st_buffer (hs.site, .01), sa.shape)) UNION_AREA
 FROM HAZARDOUS_SITES hs, SENSITIVE_AREAS sa;

     SA_ID  HS_ID     UNION_AREA

         1    4       100.000313935011
         2    4       400.000313935011
         3    4       400.000235451258
         1    5       100.000235451258
         2    5       400.000235451258
         3    5       400.000313935011
```

PostgreSQL

```
CREATE TABLE sensitive_areas (
 id integer,
 shape sde.st_geometry
);

CREATE TABLE hazardous_sites (
 id integer,
 site sde.st_geometry
);


INSERT INTO SENSITIVE_AREAS VALUES (
 1,
 sde.st_geometry ('polygon ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO SENSITIVE_AREAS VALUES (
 2,
 sde.st_geometry ('polygon ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
);
```

Copyright © 1995-2023 Esri. All rights reserved.

```
INSERT INTO SENSITIVE_AREAS VALUES (
 3,
 sde.st_geometry ('polygon ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
);

INSERT INTO HAZARDOUS_SITES VALUES (
 4,
 sde.st_geometry ('point (60 60)', 4326)
);

INSERT INTO HAZARDOUS_SITES VALUES (
 5,
 sde.st_geometry ('point (30 30)', 4326)
);
```

```
SELECT sa.id AS SA_ID, hs.id AS HS_ID,
 sde.st_area (sde.st_union (sde.st_buffer (hs.site, .01), sa.shape)) AS UNION_AREA
 FROM hazardous_sites hs, sensitive_areas sa;

     sa_id  hs_id   union_area

         1    4       100.000313935011
         2    4       400.000313935011
         3    4       400.000235451258
         1    5       100.000235451258
         2    5       400.000235451258
         3    5       400.000313935011
```

SQLite

```
CREATE TABLE sensitive_areas (
 id integer
);

SELECT AddGeometryColumn(
 NULL,
 'sensitive_areas',
 'shape',
 4326,
 'polygon',
 'xy',
 'null'
);

CREATE TABLE hazardous_sites (
 id integer
);

SELECT AddGeometryColumn(
 NULL,
 'hazardous_sites',
 'site',
 4326,
 'point',
 'xy',
 'null'
);

INSERT INTO sensitive_areas VALUES (
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 10,
 st_geometry ('polygon ((20 30, 30 30, 30 40, 20 40, 20 30))', 4326)
);

INSERT INTO sensitive_areas VALUES (
 11,
 st_geometry ('polygon ((30 30, 30 50, 50 50, 50 30, 30 30))', 4326)
);

INSERT INTO sensitive_areas VALUES (
 12,
 st_geometry ('polygon ((40 40, 40 60, 60 60, 60 40, 40 40))', 4326)
);

INSERT INTO hazardous_sites VALUES (
 40,
 st_geometry ('point (60 60)', 4326)
);

INSERT INTO hazardous_sites VALUES (
 41,
 st_geometry ('point (30 30)', 4326)
);
```

```
SELECT sa.id AS "sa_id", hs.id AS "hs_id",
 st_area (st_union (st_buffer (hs.site, .01), sa.shape)) AS "union"
 FROM hazardous_sites hs, sensitive_areas sa;

     sa_id  hs_id   union

         1    4       100.000313935011
         2    4       400.000313935011
         3    4       400.000235451258
         1    5       100.000235451258
         2    5       400.000235451258
         3    5       400.000313935011
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Within

Definición

ST_Within devuelve 1 (Oracle y SQLite) o t (PostgreSQL) si el primer objeto ST_Geometry está completamente inscrito en el segundo; de lo contrario, devuelve 0 (Oracle y SQLite) o f (PostgreSQL).

Sintaxis

Oracle y PostgreSQL

```
sde.st_within (geometry1 sde.st_geometry, geometry2 sde.st_geometry)
```

SQLite

```
st_within (geometry1 geometryblob, geometry2 geometryblob)
```

Tipo de devolución

Booleano

Ejemplo

En el siguiente ejemplo, se crean dos tablas: zones y squares. La declaración SELECT encuentra todos los cuadrados que se intersecan con una parcela pero no están totalmente contenidos en ella.

Oracle

```
CREATE TABLE squares (
 id integer,
 shape sde.st_geometry);

CREATE TABLE zones (
 id integer,
 shape sde.st_geometry);


INSERT INTO squares (id, shape) VALUES (
 1,
 sde.st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);

INSERT INTO squares (id, shape) VALUES (
 2,
 sde.st_polygon ('polygon ((20 0, 20 10, 30 10, 30 0, 20 0))', 4326)
);

INSERT INTO squares (id, shape) VALUES (
 3,
 sde.st_polygon ('polygon ((40 0, 40 10, 50 10, 50 0, 40 0))', 4326)
);

INSERT INTO zones (id, shape) VALUES (
 1,
 sde.st_polygon ('polygon ((-1 -1, -1 11, 11 11, 11 -1, -1 -1))', 4326)
```

Copyright © 1995-2023 Esri. All rights reserved.

```
);

INSERT INTO zones (id, shape) VALUES (
 2,
 sde.st_polygon ('polygon ((19 -1, 19 11, 29 9, 31 -1, 19 -1))', 4326)
);

INSERT INTO zones (id, shape) VALUES (
 3,
 sde.st_polygon ('polygon ((39 -1, 39 11, 51 11, 51 -1, 39 -1))', 4326)
);
```

```
SELECT s.id sq_id
 FROM SQUARES s, ZONES z
 WHERE sde.st_intersects (s.shape, z.shape) = 1
 AND sde.st_within (s.shape, z.shape) = 0;

SQ_ID

2
```

PostgreSQL

```
CREATE TABLE squares (
 id integer,
 shape sde.st_geometry);

CREATE TABLE zones (
 id integer,
 shape sde.st_geometry);


INSERT INTO squares (id, shape) VALUES (
 1,
 sde.st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);

INSERT INTO squares (id, shape) VALUES (
 2,
 sde.st_polygon ('polygon ((20 0, 20 10, 30 10, 30 0, 20 0))', 4326)
);

INSERT INTO squares (id, shape) VALUES (
 3,
 sde.st_polygon ('polygon ((40 0, 40 10, 50 10, 50 0, 40 0))', 4326)
);

INSERT INTO zones (id, shape) VALUES (
 1,
 sde.st_polygon ('polygon ((-1 -1, -1 11, 11 11, 11 -1, -1 -1))', 4326)
);

INSERT INTO zones (id, shape) VALUES (
 2,
 sde.st_polygon ('polygon ((19 -1, 19 11, 29 9, 31 -1, 19 -1))', 4326)
);

INSERT INTO zones (id, shape) VALUES (
 3,
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 sde.st_polygon ('polygon ((39 -1, 39 11, 51 11, 51 -1, 39 -1))', 4326)
);
```

```
SELECT s.id
 AS sq_id
 FROM squares s, zones z
 WHERE st_intersects (s.shape, z.shape) = 't'
 AND st_within (s.shape, z.shape) = 'f';

sq_id

2
```

SQLite

```
CREATE TABLE squares (
 id integer
);

SELECT AddGeometryColumn(
 NULL,
 'squares',
 'shape',
 4326,
 'polygon',
 'xy',
 'null'
);

CREATE TABLE zones (
 id integer
);

SELECT AddGeometryColumn(
 NULL,
 'zones',
 'shape',
 4326,
 'polygon',
 'xy',
 'null'
);

INSERT INTO squares (id, shape) VALUES (
 1,
 st_polygon ('polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4326)
);

INSERT INTO squares (id, shape) VALUES (
 2,
 st_polygon ('polygon ((20 0, 20 10, 30 10, 30 0, 20 0))', 4326)
);

INSERT INTO squares (id, shape) VALUES (
 3,
 st_polygon ('polygon ((40 0, 40 10, 50 10, 50 0, 40 0))', 4326)
);

INSERT INTO zones (id, shape) VALUES (
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 1,
 st_polygon ('polygon ((-1 -1, -1 11, 11 11, 11 -1, -1 -1))', 4326)
);

INSERT INTO zones (id, shape) VALUES (
 2,
 st_polygon ('polygon ((19 -1, 19 11, 29 9, 31 -1, 19 -1))', 4326)
);

INSERT INTO zones (id, shape) VALUES (
 3,
 st_polygon ('polygon ((39 -1, 39 11, 51 11, 51 -1, 39 -1))', 4326)
);
```

```
SELECT s.id
 AS "sq_id"
 FROM squares s, zones1 z
 WHERE st_intersects (s.shape, z.shape) = 1
 AND st_within (s.shape, z.shape) = 0;

sq_id

2
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_X

Definición

ST_X toma un ST_Point como parámetro de entrada y devuelve su coordenada x. En SQLite, ST_X puede actualizar también la coordenada x de un ST_Point.

Sintaxis

Oracle y PostgreSQL

```
sde.st_x (point1 sde.st_point)
```

SQLite

```
st_x (point1 geometryblob)
st_x (input_point geometryblob, new_Xvalue double)
```

Tipo de devolución

Precisión doble

La función ST_X se puede usar también con SQLite para actualizar la coordenada x de un punto. En ese caso, se devuelve un geometryblob.

Ejemplos

Se crea la tabla x_test con dos columnas: la columna gid, que identifica de forma única cada fila, y la columna de punto pt1.

Las declaraciones INSERT insertan dos filas. Uno es un punto sin una coordenada z o medida. La otra columna tiene tanto una coordenada z y medida.

La consulta SELECT usa la función ST_X para obtener la coordenada x de cada entidad de punto.

Copyright © 1995-2023 Esri. All rights reserved.

Oracle

```
CREATE TABLE x_test (
 gid integer unique,
 pt1 sde.st_point
);
```

```
INSERT INTO X_TEST VALUES (
 1,
 sde.st_pointfromtext ('point (10.02 20.01)', 4326)
);

INSERT INTO X_TEST VALUES (
 2,
 sde.st_pointfromtext ('point zm(10.1 20.01 5 7)', 4326)
);
```

```
SELECT gid, sde.st_x (pt1) "The X coordinate"
 FROM X_TEST;

       GID       The X coordinate

         1            10.02
         2            10.10
```

PostgreSQL

```
CREATE TABLE x_test (
 gid integer unique,
 pt1 sde.st_point
);
```

```
INSERT INTO x_test VALUES (
 1,
 sde.st_point ('point (10.02 20.01)', 4326)
);

INSERT INTO x_test VALUES (
 2,
 sde.st_point ('point zm(10.1 20.01 5 7)', 4326)
);
```

```
SELECT gid, sde.st_x (pt1)
 AS "The X coordinate"
 FROM x_test;

       gid       The X coordinate

         1            10.02
         2            10.10
```

Copyright © 1995-2023 Esri. All rights reserved.

SQLite

```
CREATE TABLE x_test (gid integer);

SELECT AddGeometryColumn(
 NULL,
 'x_test',
 'pt1',
 4326,
 'pointzm',
 'xyzm',
 'null'
);
```

```
INSERT INTO x_test VALUES (
 1,
 st_point ('point (10.02 20.01)', 4326)
);

INSERT INTO x_test VALUES (
 2,
 st_point ('point zm(10.1 20.01 5 7)', 4326)
);
```

```
SELECT gid, st_x (pt1)
 AS "The X coordinate"
 FROM x_test;

       gid       The X coordinate

         1            10.02
         2            10.10
```

La función ST_X se puede usar también para actualizar el valor de coordenadas de un punto existente. En este ejemplo, se usa ST_X para actualizar el valor de la coordenada x del primer punto de x_test.

```
UPDATE x_test
 SET pt1=st_x(
  (SELECT pt1 FROM x_test WHERE gid=1),
  10.04
  )
 WHERE gid=1;
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Y

Definición

ST_Y toma un ST_Point como parámetro de entrada y devuelve su coordenada y. En SQLite, ST_Y puede actualizar también la coordenada y de un ST_Point.

Sintaxis

Oracle y PostgreSQL

```
sde.st_y (point1 sde.st_point)
```

SQLite

```
double   st_y (point1  geometryblob)
geometry st_y (input_shape geometryblob, new_Yvalue double)
```

Tipo de devolución

Precisión doble

La función ST_Y se puede usar también con SQLite para actualizar el valor de la coordenada y de un punto. En ese caso, se devuelve un geometryblob.

Ejemplo

Se crea la tabla y_test con dos columnas: la columna gid, que identifica de forma única cada fila, y la columna de punto pt1.

Las declaraciones INSERT insertan dos filas. Uno es un punto sin una coordenada z o medida. El otro tanto tiene una coordenada z y una medida.

La consulta SELECT usa la función ST_Y para devolver la coordenada y de cada punto.

Copyright © 1995-2023 Esri. All rights reserved.

Oracle

```
CREATE TABLE y_test (
 gid integer unique,
 pt1 sde.st_point
);
```

```
INSERT INTO Y_TEST VALUES (
 1,
 sde.st_pointfromtext ('point (10.02 20.02)', 4326)
);

INSERT INTO Y_TEST VALUES (
 2,
 sde.st_pointfromtext ('point zm(10.1 20.01 5.0 7.0)', 4326)
);
```

```
SELECT gid, sde.st_y (pt1) "The Y coordinate"
 FROM Y_TEST;

       GID     The Y coordinate

         1          20.02
         2          20.01
```

PostgreSQL

```
CREATE TABLE y_test (
 gid integer unique,
 pt1 sde.st_point
);
```

```
INSERT INTO y_test VALUES (
 1,
 sde.st_point ('point (10.02 20.02)', 4326)
);

INSERT INTO y_test VALUES (
 2,
 sde.st_point ('point zm(10.1 20.01 5.0 7.0)', 4326)
);
```

```
SELECT gid, sde.st_y (pt1)
 AS "The Y coordinate"
 FROM y_test;

       gid    The Y coordinate

         1          20.02
         2          20.01
```

Copyright © 1995-2023 Esri. All rights reserved.

SQLite

```
CREATE TABLE y_test (gid integer);

SELECT AddGeometryColumn(
 NULL,
 'y_test',
 'pt1',
 4326,
 'pointzm',
 'xyzm',
 'null'
);
```

```
INSERT INTO y_test VALUES (
 1,
 st_point ('point (10.02 20.02)', 4326)
);

INSERT INTO y_test VALUES (
 2,
 st_point ('point zm(10.1 20.01 5.0 7.0)', 4326)
);
```

```
SELECT gid, st_y (pt1)
 AS "The Y coordinate"
 FROM y_test;

gid    The Y coordinate

1          20.02
2          20.01
```

La función ST_Y se puede usar también para actualizar el valor de coordenadas de un punto existente. En este ejemplo, se usa ST_Y para actualizar el valor de la coordenada y del segundo punto de y_test.

```
UPDATE y_test
 SET pt1=st_y(
  (SELECT pt1 FROM y_test WHERE gid=2),
  20.1
  )
 WHERE gid=2;
```

Copyright © 1995-2023 Esri. All rights reserved.

ST_Z

Definición

ST_Z toma un ST_Point como parámetro de entrada y devuelve su coordenada z (elevación). En SQLite, ST_Z puede actualizar también la coordenada z de un ST_Point.

Sintaxis

Oracle y PostgreSQL

```
sde.st_z (geometry1 sde.st_point)
```

SQLite

```
st_z (geometry  geometryblob)
st_z (input_shape geometryblob, new_Zvalue double)
```

Tipo de devolución

Oracle

Número

PostgreSQL

Entero

SQLite

Se devuelve una precisión doble cuando se usa ST_Z para devolver la coordenada z de un punto. Se devuelve un geometryblob cuando se usa ST_Z para actualizar la coordenada z de un punto.

Ejemplo

Se crea la tabla z_test con dos columnas: la columna de Id., que identifica de forma única cada fila, y la columna de punto de geometría. La declaración INSERT inserta una fila en la tabla z_test.

La declaración SELECT enumera la columna id y las coordenadas z de doble precisión del punto insertadas con la declaración anterior.

Copyright © 1995-2023 Esri. All rights reserved.

Oracle

```
CREATE TABLE z_test (
 id integer unique,
 geometry sde.st_point
);

INSERT INTO z_test (id, geometry) VALUES (
 1,
 sde.st_point (2, 3, 32, 5, 4326)
);
```

```
SELECT id, sde.st_z (geometry) Z_COORD
 FROM Z_TEST;

        ID      Z_COORD

         1        32
```

PostgreSQL

```
CREATE TABLE z_test (
 id integer unique,
 geometry sde.st_point
);

INSERT INTO z_test (id, geometry) VALUES (
 1,
 sde.st_point (2, 3, 32, 5, 4326)
);
```

```
SELECT id, sde.st_z (geometry)
 AS Z_COORD
 FROM z_test;

        id      z_coord

         1        32
```

SQLite

```
CREATE TABLE z_test (id integer);

SELECT AddGeometryColumn(
 NULL,
 'z_test',
 'pt1',
 4326,
 'pointzm',
 'xyzm',
 'null'
);

INSERT INTO z_test (id, pt1) VALUES (
 1,
```

Copyright © 1995-2023 Esri. All rights reserved.

```
 st_point (2, 3, 32, 5, 4326)
);
```

```
SELECT id, st_z (pt1)
 AS "The z coordinate"
 FROM z_test;

id      The z coordinate

1        32.0
```

La función ST_Z se puede usar también para actualizar el valor de coordenadas de un punto existente. En este ejemplo, se usa ST_Z para actualizar el valor de coordenada z del primer punto de z_test.

```
UPDATE z_test
 SET pt1=st_z(
  (SELECT pt1 FROM z_test where id=1), 32.04)
 WHERE id=1;
```

Copyright © 1995-2023 Esri. All rights reserved.