esri

Справка по функциям SQL ST_Geometry

THE SCIENCE OF WHERE

Таблица содержания

Функции SQL, используемые совместно с ST_Geometry

В документе справки приводится список и описание функций, доступных для использования с типом пространственных данных Esri ST_Geometry в Oracle, PostgreSQL и SQLite.

Функции и типы Esri ST_Geometry SQL создаются, когда вы:

- Создаете базу геоданных в базе данных Oracle.
- Используете ST_Geometry при создании базы геоданных в базе данных PostgreSQL.
- Устанавливаете тип пространственных данных ST_Geometry в базе данных Oracle или PostgreSQL.
- Создаете базу данных SQLite, которая включает тип пространственных данных ST_Geometry, используя инструмент геообработки Создать базу данных SQLite или функцию ArcPy, и загрузите функции ST_Geometry для использования их в базе данных.
- Загружаете функции ST_Geometry для использования в мобильной базе геоданных.

В базах данных Oracle или PostgreSQL тип ST_Geometry и его функции создаются в схеме пользователя sde. В SQLite тип и функции хранятся в библиотеке, которую необходимо загрузить до запуска SQL-запроса к базе данных SQLite или мобильной базе геоданных.

Подсказка:

Для получения информации о типе Esri ST_Geometry см. следующие страницы справки ArcGIS Pro:

- ST_Geometry в PostgreSQL
- ST_Geometry в Oracle
- Базы данных и ST_Geometry
- Загрузка библиотеки ST_Geometry SQLite
- Загрузка ST_Geometry в мобильную базу геоданных для доступа к SQL

Страницы с Форматом функции SQL

Страницы, посвященные функциям в этом документе, структурированы так:

- Определение - короткое выражение выполняемых функцией действий
- Синтаксис - синтаксис SQL для использования функции

Примечание:

В отношении реляционных операторов важен порядок, в котором указываются параметры: первый параметр должен относиться к таблице, из которой берется выборка, а второй – к таблице, которая будет использоваться в качестве фильтра.

- Тип возврата - тип возвращаемых данных при использовании функции
- Пример - примеры, в которых используется указанная функция

Перечень SQL-функций

Щелкните ссылку, приведенную ниже, чтобы перейти к функциям, которые вы можете использовать с типом

ST_Geometry в Oracle, PostgreSQL и SQLite.

При использовании функций ST_Geometry в Oracle вам необходимо указывать функции и операторы с использованием префикса `sde`. Например, ST_Buffer будет выглядеть как sde.ST_Buffer. Добавление префикса `sde.` указывает программному обеспечению, что эта функция хранится в схеме пользователя sde. Для PostgreSQL использование префикса необязательно, но хорошей практикой считается его добавление. Не включайте добавление префикса при использовании функций с SQLite, поскольку схема sde в базах данных SQLite отсутствует.

Когда вы задаете текстовую строку в формате WKT в качестве входной для функции ST_Geometry SQL, можно использовать научные нотации для задания очень больших или очень малых значений. Например, если вы задаете координаты, используя WKT, при создании объекта, и одна из координат имеет значение 0.00002350000181650102 6, можно ввести `2.3500001816501026e-005`.

Подсказка:

По поводу пространственных типов – таких как типы PostGIS, Oracle, пространственные типы Microsoft SQL Server, IBM Db2 SDO_Geometry или SAP HANA ST_Geometry – обратитесь к документации, предоставляемой поставщиком СУБД, чтобы узнать об используемых ими функциях.

Функции Esri ST_Geometry SQL ниже сгруппированы по виду их использования.

Функции конструктора

Функции конструктора получают один тип геометрии или текстовое описание геометрии и создают геометрию. В следующей таблице перечислены функции построения и указана их поддержка различными реализациями ST_Geometry.

Функции конструктора

Функция	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

Функция	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

Функции метода доступа

Ниже приводится несколько функций, которые используют в качестве входного параметра геометрию или несколько геометрий и возвращают определенную информацию о них.

Некоторые из этих функций доступа проверяют, удовлетворяет ли объект или несколько объектов заданному критерию. Если геометрия удовлетворяет критерию, функция возвращает значение 1(Oracle и SQLite) или t (PostgreSQL). Если геометрия не удовлетворяет критерию, функция возвращает значение 0 (Oracle и SQLite) или f (PostgreSQL).

Эти функции применяются ко всем реализациям, если не указано обратное.

Функции метода доступа

ST_Area
ST_AsBinary
ST_AsText
ST_CoordDim
ST_Dimension
ST_EndPoint
ST_Entity
ST_Equalsrs (только для PostgreSQL)
ST_ExteriorRing

ST_GeomFromCollection (только для PostgreSQL)
ST_GeometryType
ST_GeoSize (только для 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

Реляционные функции

Реляционные функции получают в качестве входного параметра геометрию объектов и определяют, существует ли пространственное отношение между ними. Если выполнены условия пространственного отношения, эти функции возвращают 1 (Oracle и SQLite) или t (PostgreSQL). Если условия не выполнены (не существует никакого отношения), эти функции возвращают 0 (Oracle и SQLite) или f (PostgreSQL).

Эти функции применяются ко всем реализациям, если не указано обратное.

Реляционные функции

ST_Contains
ST_Crosses
ST_Disjoint
ST_Distance
ST_DWithin
ST_EnvIntersects (только для Oracle и SQLite)
ST_Equals
ST_Intersects
ST_OrderingEquals (только для Oracle и PostgreSQL)
ST_Overlaps
ST_Relate
ST_Touches
ST_Within

Функции операций с геометрией

Эти функции получают пространственные данные, выполняют с ними пространственные операции и возвращают геометрию.

Эти функции применяются ко всем реализациям, если не указано обратное.

Функции операций с геометрией

ST_Aggr_ConvexHull (только для Oracle и SQLite)
ST_Aggr_Intersection (только для Oracle и 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
ST_Union

SQL и ST_Geometry Esri

Вы можете использовать Язык структурированных запросов (SQL), типы данных и форматы таблиц системы управления базами данных для работы с информацией, хранящейся в базе геоданных или базе данных, в которой установлен тип ST_Geometry. SQL - это язык базы данных, который поддерживает команды определения данных и обработки данных.

Доступ к данным через SQL позволяет внешним приложениям работать с табличными данными, управляемыми базой геоданных или базой данных. Эти внешние приложения могут быть непространственными приложениями базы данных или пользовательскими пространственными приложениями.

При вставке данных или редактировании данных в базе геоданных или базе данных с помощью SQL введите выражение `COMMIT` или `ROLLBACK` после запуска выражения SQL, чтобы убедиться, что изменения либо зафиксированы в базе данных, либо отменены. Это помогает предотвратить блокировку строк, страниц или таблиц, которые вы редактируете.

Вставка данных ST_Geometry с помощью SQL

Вы можете использовать SQL для вставки пространственных данных в базу данных или таблицу базы геоданных, содержащую столбец ST_Geometry. Вы используете функции конструктора ST_Geometry для вставки определенных типов геометрии. Вы также можете указать, что выходные данные некоторых функций пространственных операций должны выводиться в существующую таблицу.

Когда вы вставляете геометрию в таблицу с помощью SQL, имейте в виду следующее:

- Необходимо указать допустимый ID пространственной привязки (SRID).
- Все геометрии в одном столбце должны иметь одинаковый SRID.
- Чтобы продолжить использование таблицы с ArcGIS, поле, используемое в качестве ID объекта, не может быть иметь значения null или содержать неуникальные значения.

ID пространственной привязки

SRID, который вы указываете при вставке геометрии в таблицу в Oracle, в которой используется пространственный тип ST_Geometry, должен находиться в таблице ST_SPATIAL_REFERENCES и иметь соответствующую запись в таблице SDE.SPATIAL_REFERENCES. SRID, который вы указываете при вставке геометрии в таблицу в PostgreSQL, в которой используется пространственный тип ST_Geometry, должен находиться в таблице public.sde_spatial_references. Эти таблицы предварительно заполнены пространственными привязками и идентификаторами SRID.

SRID, указываемый при вставке геометрии в таблицу в SQLite, в которой используется пространственный тип ST_Geometry, должен находиться в таблице st_spatial_reference_systems.

Если вам нужно использовать пользовательскую пространственную привязку, которой нет в таблице, проще всего это сделать с помощью загрузки или создания класса пространственных объектов с нужными вам значениями пространственной привязки. Убедитесь, что создаваемый вами класс пространственных объектов использует хранилище ST_Geometry. Это создает запись в таблицах SDE.SPATIAL_REFERENCES и ST_SPATIAL_REFERENCES в Oracle, запись в таблице public.sde_spatial_references в PostgreSQL, или запись в таблице st_aux_spatial_reference_systems_table в SQLite.

В базах геоданных вы можете запросить таблицу LAYERS (Oracle) или sde_layers (PostgreSQL), чтобы

обнаружить SRID, присвоенный пространственной таблице. Затем вы можете использовать этот SRID при создании пространственных таблиц и вставке данных с помощью SQL.

Примечание:
Для использования примеров в этом документе, в таблицы ST_SPATIAL_REFERENCES и sde_spatial_references добавлена запись для обозначения неизвестной пространственной привязки. Эта запись имеет SRID 0. Вы можете использовать этот SRID для примеров в этом документе. Однако это не официальный SRID - он предоставляется для выполнения примера кода SQL. Не рекомендуется использовать этот SRID для производственных данных.

ID объектов

Чтобы ArcGIS запрашивал данные, необходимо, чтобы таблица содержала поле уникального идентификатора объекта.

Классы пространственных объектов, созданные с помощью ArcGIS, всегда имеют поле ID объекта, которое используется в качестве поля идентификатора. При вставке записей в класс пространственных объектов с помощью ArcGIS уникальное значение всегда вставляется в поле ID объекта. Поле ID объекта в таблице базы геоданных поддерживается ArcGIS. Поле ID объекта в таблице базы данных, созданной из ArcGIS, поддерживается системой управления базой данных.

Когда вы вставляете записи в таблицу базы геоданных с помощью SQL, вы должны вставить действительное уникальное значение ID объекта.

Таблицы базы данных, создаваемые вне ArcGIS, должны содержать поле (или набор полей), которое ArcGIS может использовать в качестве ID объекта. Если вы используете собственный автоматически увеличивающийся тип данных базы данных для поля ID в таблице, это поле будет заполняться базой данных при вставке записи с помощью SQL. Если вы вручную поддерживаете значения в поле уникального идентификатора, обязательно укажите уникальное значение для ID при редактировании таблицы из SQL.

Примечание:
Вы не можете публиковать данные из таблиц, имеющих поле уникального идентификатора, которое не поддерживается ArcGIS или системой управления базами данных.

Редактирование данных ST_Geometry с помощью SQL

Изменения SQL в существующих записях часто влияют на непространственные атрибуты, хранящиеся в таблице; однако вы можете редактировать данные в столбце ST_Geometry, используя функции-конструкторы внутри выражений `UPDATE` SQL.

Если данные хранятся в базе геоданных, существуют дополнительные рекомендации, которым необходимо следовать при редактировании с помощью SQL:

- Не обновляйте записи с помощью SQL, если данные были зарегистрированы как версионные или включены для архивирования базы геоданных.
- Не изменяйте никакие атрибуты, влияющие на другие объекты в базе данных, участвующие в поведении базы геоданных, такие как классы отношений, объектно-связанные аннотации, топология, правила атрибутов или сети.
- Не используйте SQL для изменения схем таблиц.

⚠ Внимание:

Использование SQL для доступа к базе геоданных обходит функциональные возможности базы геоданных, такие как управление версиями, топология, сети, terrains, объектно-связанные аннотации или другие расширения классов или рабочих областей. Можно использовать функции системы управления базами данных, такие как триггеры и хранимые процедуры, для поддержания отношений между таблицами, необходимых для определенных функций базы геоданных. Однако выполнение команд SQL для базы геоданных без учета этой дополнительной функциональности - например, выдача выражений `INSERT` для добавления записей в таблицу, для которой включено архивирование базы геоданных, или добавление столбца в существующий класс пространственных объектов - приведет к обходу функциональности базы геоданных и, возможно, испортит отношения между данными в базе геоданных.

Загрузите библиотеку SQLite ST_Geometry

Перед запуском команд SQL, которые содержат функции ST_Geometry для базы данных SQLite, выполните следующее:

1. Загрузите zip-файл ArcGIS Pro библиотеки ST_Geometry (SQLite) из My Esri и распакуйте его.
2. Установите редактор SQL на той же машине, на которой находится база данных.
3. Поместите файл ST_Geometry в местоположение, доступное для базы данных SQLite и редактора SQL, из которого вы загрузите ST_Geometry.
   Если база данных SQLite находится на машине Microsoft Windows, используйте файл `stgeometry_sqlite.dll`. Если база данных SQLite находится на машине Linux, используйте файл `libstgeometry_sqlite.so`.
4. Откройте редактор SQL и подключитесь к базе данных SQLite.
5. Загрузите библиотеку ST_Geometry.
   В первом примере ниже библиотека загружена для базы данных SQLite на машине Windows. Во втором примере библиотека загружается в базу данных SQLite на машине 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'
);
```

Теперь вы можете запускать команды SQL, содержащие функции ST_Geometry, для базы данных SQLite.

Функции конструктора для ST_Geometry

Функции конструктора создают геометрию из стандартного текстового описания или WKB.

Когда вы даёте известное текстовое описание для построения геометрии, последними должны указываться координаты измерения. Например, если ваш текст содержит координаты x, y, z и m, то они должны указываться в том же порядке.

Геометрия может иметь несколько точек (или вообще их не иметь). Геометрия считается пустой, если в ней нет точек. Точечный подтип (point subtype) – это единственная геометрия, для которой установлено ограничение в одну или ноль точек; остальные подтипы могут иметь ноль или более точек.

В следующих разделах описывают геометрии суперклассов и подклассов для выбранной геометрии, а также перечисляют функции, которые могут создавать одна другую.

Вы также можете построить геометрии как результат пространственной операции, выполняемой с существующими геометрическими объектами.

Суперкласс Геометрия

Суперкласс ST_Geometry не может быть установлен; однако, вы можете определить столбец в формате ST_Geometry, и добавляемые актуальные данные определятся, как объекты точка (point), линия (linestring), полигон (polygon), мультиточка (multipoint), мультилиния (multilinestring) или мультиполигон (multipolygon).

Следующие функции могут быть использованы для создания суперкласса, который будет хранить любые вышеперечисленные типы элементов.

- ST_Geometry
- ST_GeomFromText (только Oracle и SQLite)
- ST_GeomFromWKB

Подклассы

Вы можете определить объект, как специальный подкласс; в этом случае будут добавляться только те типы элементов, которые допустимы для этого подкласса. Например, ST_PointFromWKB может создавать только точечные элементы.

ST_Point

ST_Point – это геометрия нулевой размерности, которая занимает отдельное местоположение в координатном пространстве. ST_Point имеет одиночное значение координат x,y, всегда является простым и не имеет (NULL) границу. ST_Point может быть использован для определения объектов, таких как нефтяные скважины, достопримечательности и места сбора проб воды.

Следующие функции создают точку

- ST_Point
- ST_PointFromText (только Oracle и SQLite)
- ST_PointFromWKB

ST_MultiPoint

ST_MultiPoint это – набор элементов ST_Points, размер которых равен 0. ST_MultiPoint является простым, если

ни один из его элементов не занимает одно и то же координатное пространство. Граница у ST_MultiPoint отсутствует (NULL). ST_MultiPoints может определять такие закономерности, как воздушные схемы трансляции и случаи распространения заболеваний.

Следующие функции создают геометрию мультиточки:

- ST_MultiPoint
- ST_MPointFromText (только Oracle)
- ST_MPointFromWKB

ST_LineString

ST_LineString – это объект с размерностью 1, хранящийся, как последовательность точек, определяющая линейно-интерполированный путь. ST_LineString является простым, если он не имеет самопересечений. Конечные точки (граница) замкнутого ST_LineString занимают одну и ту же точку в пространстве. ST_LineString является кольцом, если он одновременно замкнутый и простой. В качестве свойства, наследованного от их суперкласса ST_Geometry, объекты ST_LineString имеют длину. ST_LineString часто используется для определения линейных объектов, таких как дороги, реки и линии электропередач.

Конечные точки обычно формируют границу ST_LineString, кроме того случая, когда ST_LineString замкнут, в этом случае граница отсутствует (NULL). Внутренняя часть ST_LineString – это связанный путь, который лежит между конечными точками, кроме того случая, когда ST_LineString замкнут; в этом случае внутренняя часть является непрерывной.

Функции, которые создают линии, перечислены ниже:

- ST_LineString
- ST_LineFromText (только Oracle и SQLite)
- ST_LineFromWKB
- ST_Curve (только Oracle и SQLite)

ST_MultiLineString

ST_MultiLineString – это набор из объектов ST_LineString.

Границей ST_MultiLineString являются непересекающиеся конечные точки элементов ST_LineString. Граница ST_MultiLineString отсутствует (NULL), если все конечные точки всех элементов пересекаются. В дополнение к другим свойствам, наследованным от суперкласса ST_Geometry, объекты ST_MultiLineString имеют длину. ST_MultiLineString часто используется для разрозненных (несмежных) линейных объектов, таких как водотоки или дорожные сети.

Функции, которые создают мультилинии, перечислены ниже:

- ST_MultiLineString
- ST_MLineFromText (только Oracle и SQLite)
- ST_MLineFromWKB
- ST_MultiCurve (только Oracle)

ST_Polygon

ST_Polygon – это двумерная поверхность, хранящаяся как последовательность точек, определяющих ее внешнее ограничивающее кольцо и ноль или более внутренних колец. Объекты ST_Polygon всегда являются простыми. Элементы ST_Polygon определяют объекты, которые имеют пространственный экстент, такие как участки земли, водные поверхности и области юрисдикции.

Данный рисунок показывает примеры объектов ST_Polygon: 1 – ST_Polygon, граница которого задана внешним кольцом. (2) – ST_Polygon, граница которого задана внешним кольцом и двумя внутренними кольцами. Область, заключенная между внутренних колец, относится к внешней части ST_Polygon. 3 – корректный ST_Polygon, поскольку кольца сходятся в одной касательной точке.

Внешние и любые внутренние кольца определяют границу ST_Polygon, а пространство, заключенное между кольцами, определяет внутреннюю часть ST_Polygon. Кольца объекта ST_Polygon могут пересекаться в касательной точке, но не могут пересекаться (перекрещиваться). В дополнение к другим свойствам, наследованным от суперкласса ST_Geometry, у объектов ST_Polygon есть площадь.

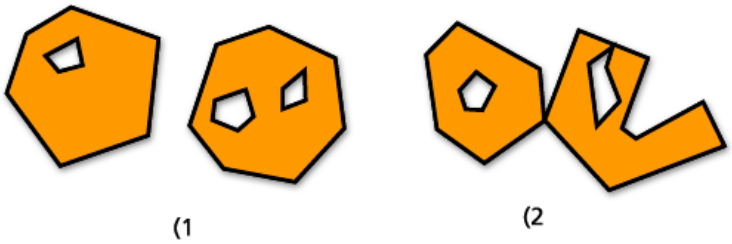
Функции, которые создают полигоны, перечислены ниже:

- ST_Polygon
- ST_PolyFromText (только Oracle и SQLite)
- ST_PolyFromWKB
- ST_Surface (только Oracle и SQLite)

ST_MultiPolygon

Границей ST_MultiPolygon является суммированная длина всех внутренних и внешних колец его элементов. Внутренняя часть ST_MultiPolygon определяется, как суммарные внутренние части всех его элементов ST_Polygon. Граница элементов объекта ST_MultiPolygon может пересекаться только в касательной точке. В дополнение к другим свойствам, наследованным от суперкласса ST_Geometry, объекты ST_MultiPolygon имеют площадь. Элементы ST_MultiPolygon определяют объекты, такие как лесной массив или несмежный участок земли, например, цепь тихоокеанских островов.

На рисунке ниже показаны примеры объектов ST_MultiPolygon: 1 - ST_MultiPolygon с двумя элементами ST_Polygon. Граница задана двумя внешними и тремя внутренними кольцами. 2 - тоже ST_MultiPolygon с двумя элементами ST_Polygon, но его граница задается двумя внешними и двумя внутренними кольцами. Два элемента ST_Polygon сходятся в касательной точке.



Следующие функции создают мультиполигоны:

- ST_MultiPolygon
- ST_MPolyFromText (только Oracle и SQLite)
- ST_MPolyFromWKB
- ST_MultiSurface (только Oracle)

Построение новых геометрических объектов из существующих

Не являясь, строго говоря, функциями-конструкторами, следующие функции возвращают новую геометрию, используя в качестве входных данных существующие геометрические элементы и выполняя их анализ:

- ST_Aggr_ConvexHull (только для Oracle и SQLite)
- ST_Aggr_Intersection (только для Oracle и 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

Функции пространственного доступа для ST_Geometry

Функции пространственного доступа возвращают свойство геометрии. Приведенные функции доступа определяют следующие свойства объекта ST_Geometry:

Размерность

Размерность геометрии представляет собой минимальные координаты (нет, x, y), необходимые для определения пространственного экстента геометрии.

Геометрия может иметь размерность, равную 0, 1 или 2.

Значения размерности соответствует следующему:

- 0 – нет ни длины, ни площади
- 1 – Имеется длина (x или y)
- 2 – Имеется площадь (x и y)

Подтипы Point и multipoint имеют размерность 0. Точки представляют объекты нулевой размерности, которые могут быть смоделированы всего одной координатой, в то время как мультиточки представляют данные, которые могут быть смоделированы кластером, состоящим из несвязанных координат.

Подтипы Linestring и Multilinestring имеют размерность 1. Они хранят объекты, такие как сегменты дорог, разветвленные системы рек, а также любые другие объекты, являющиеся линейными по природе.

Подтипы полигона и мультиполигона имеют размерность 2. Лесные массивы, участки, водная поверхность и другие объекты, которые имеют периметры, замыкающиеся в определяемой области, могут быть представлены полигональным или мультиполигональным типом данных.

Размерность важна не только, как свойство подтипа, но также для определения пространственных отношений между двумя объектами. Размерность результирующего объекта или объектов определяет, была ли операция успешной, или нет. Пространственные функции доступа проверяют размерность объектов, чтобы определить, как их следует сравнивать.

Чтобы оценить размерность геометрии, используйте функцию ST_Dimension, которая получает объект ST_Geometry и возвращает значение его размерности как целое число.

Координаты геометрии также имеют размерность. Если геометрия имеет только координаты x и y, размерность координат равна 2. Если геометрия имеет координаты x, y и z, размерность координат равна 3. Если геометрия имеет координаты x, y, z и m, размерность координат равна 4.

Вы можете использовать функцию ST_CoordDim для определения размерности координат, присутствующих в геометрии.

Z-координаты

С некоторыми геометриями связано третье измерение – высота или глубина. Каждая из точек, образующих геометрию объекта, может включать дополнительную координату z, обозначающую высоту или глубину объекта по отношению к поверхности земли.

Функция предиката ST_Is3d принимает в качестве входных данных ST_Geometry и возвращает значение true, если функция имеет z-координаты, или значение false, если это не так.

Вы можете определить z-координату точки, используя функцию ST_Z.

Функция ST_MaxZ возвращает максимальное значение координаты z, а функция ST_MinZ возвращает минимальное значение координаты z для геометрии.

Измерения

Измерения – это значения, присвоенные каждой координате. Они используются для систем линейных координат и динамической сегментации. Например, местоположения километровых столбов вдоль шоссе могут содержать измерения, указывающие их положение. Это значение может показывать что угодно, что можно показать с помощью числа двойной точности.

Функция предиката ST_IsMeasured получает геометрию и возвращает значение true, если данная геометрия содержит измерения, и значение false, если нет. Эта функция используется только с реализациями ST_Geometry Oracle и SQLite.

Вы можете определить значение измерения точки с помощью функцию ST_M.

Функция ST_MaxM возвращает максимальное значение m-координаты, а функция ST_MinM возвращает минимальное значение m-координаты для геометрии.

Тип геометрии

Тип геометрии связан с типом геометрического объекта. Это следующие требования:

- Точки и мультиточки
- Линии и мультилинии
- Полигоны и мультиполигоны

ST_Geometry – это суперкласс, который может хранить различные подтипы. Чтобы определить, какой подтип имеет геометрия, используйте функцию ST_GeometryType или ST_Entity (только Oracle и SQLite).

Набор точек (вершин) и число точек

Геометрия может иметь несколько точек (или вообще их не иметь). Геометрия считается пустой, если в ней нет точек. Точечный подтип (point subtype) – это единственная геометрия, для которой установлено ограничение в одну или ноль точек; остальные подтипы могут иметь ноль или более точек.

ST_Point

ST_Point – это геометрия нулевой размерности, которая занимает отдельное местоположение в координатном пространстве. ST_Point имеет одиночное значение координат x,y, всегда является простым и не имеет (NULL) границу. ST_Point может быть использован для определения объектов, таких как нефтяные скважины, достопримечательности и места сбора проб воды.

Функции, которые работают только с типом данных ST_Point, включают следующие:

- ST_X – Возвращает значение x-координаты точечного типа данных как число двойной точности.
- ST_Y – Возвращает значение y-координаты точечного типа данных как число двойной точности.
- ST_Z – Возвращает значение z-координаты точечного типа данных как число двойной точности.
- ST_M – Возвращает значение m-координаты точечного типа данных как число двойной точности.

ST_MultiPoint

ST_MultiPoint это – набор элементов ST_Points, размер которых равен 0. ST_MultiPoint является простым, если

ни один из его элементов не занимает одно и то же координатное пространство. Граница у ST_MultiPoint отсутствует (NULL). ST_MultiPoints может определять такие закономерности, как воздушные схемы трансляции и случаи распространения заболеваний.

Вы можете использовать функцию ST_NumGeometries для определения количества точек в мультиточечной геометрии.

Длина, площадь и периметр

Длина, площадь и площадь – это измеряемые характеристики геометрии. Ломаные и элементы мультиломаных являются одномерными и имеют характеристику длины. Полигоны и элементы мультиполигонов являются двухмерными поверхностями и, соответственно, имеют площадь и периметр, которую можно измерить. Для определения этих свойств вы можете использовать функции ST_Length, ST_Area и ST_Perimeter. Единицы измерения могут меняться, в зависимости от того, как хранятся данные.

ST_LineString

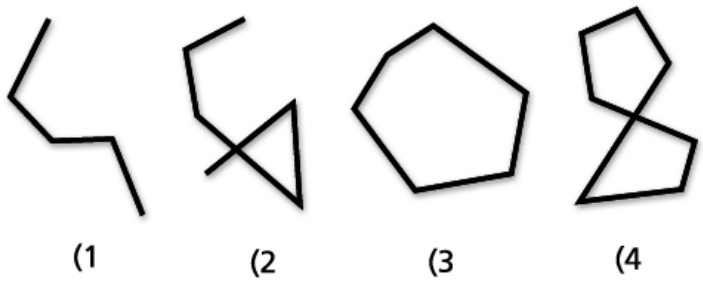
ST_LineString – это объект с размерностью 1, хранящийся, как последовательность точек, определяющая линейно-интерполированный путь. ST_LineString является простым, если он не имеет самопересечений. Конечные точки (граница) замкнутого ST_LineString занимают одну и ту же точку в пространстве. ST_LineString является кольцом, если он одновременно замкнутый и простой. В качестве свойства, наследованного от их суперкласса ST_Geometry, объекты ST_LineString имеют длину. ST_LineString часто используется для определения линейных объектов, таких как дороги, реки и линии электропередач.

Конечные точки обычно формируют границу ST_LineString, кроме того случая, когда ST_LineString замкнут, в этом случае граница отсутствует (NULL). Внутренняя часть ST_LineString – это связанный путь, который лежит между конечными точками, кроме того случая, когда ST_LineString замкнут; в этом случае внутренняя часть является непрерывной.

Функции, которые работают с объектами ST_LineString, перечислены ниже:

- ST_StartPoint – Возвращает первую точку выбранного ST_LineString
- ST_EndPoint – Возвращает последнюю точку ST_LineString
- ST_IsClosed - функция предиката, возвращающая true, если указанная ST_LineString замкнута (начальная и конечная точки строки пересекаются), и false, если она не замкнута
- ST_IsRing - функция предиката, которая возвращает true, если указанная ST_LineString является кольцом, и false, если это не так
- ST_Length – Возвращает длину ST_LineString как число двойной точности
- ST_NumPoints – Оценивает ST_LineString и возвращает число точек в его последовательности как целое число
- ST_PointN - получает ST_LineString и индекс n-ой точки и возвращает эту точку

На рисунке ниже показаны примеры объектов ST_LineString: (1 — простая, незамкнутая ST_LineString; (2 — непростая, незамкнутая ST_LineString; (3 — замкнутая, простая ST_LineString и, следовательно, кольцо; и (4) — замкнутая, непростая ST_LineString, но это не кольцо.

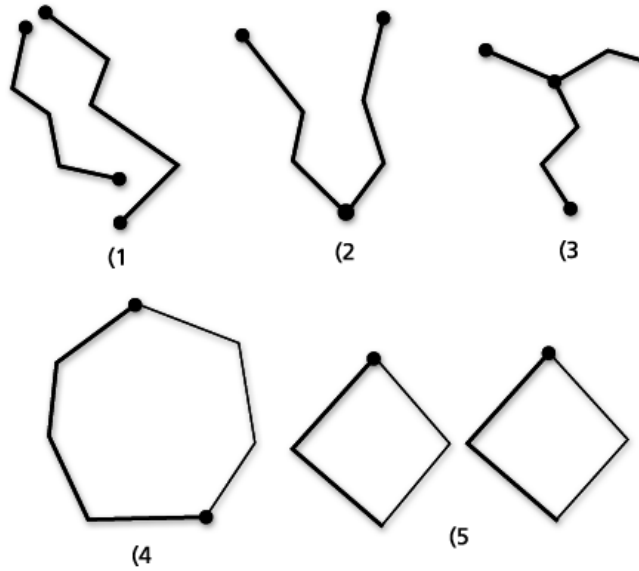


ST_MultiLineString

ST_MultiLineString – это набор из объектов ST_LineString. Объекты ST_MultiLineString являются простыми, если они пересекаются только в конечных точках элементов ST_LineString. Объекты ST_MultiLineString не являются простыми, если внутренние части элементов ST_LineString пересекаются.

Границей ST_MultiLineString являются непересекающиеся конечные точки элементов ST_LineString. Граница ST_MultiLineString отсутствует (NULL), если все конечные точки всех элементов пересекаются. В дополнение к другим свойствам, наследованным от суперкласса ST_Geometry, объекты ST_MultiLineString имеют длину. ST_MultiLineString часто используется для разрозненных (несмежных) линейных объектов, таких как водотоки или дорожные сети.

На следующем рисунке представлены примеры ST_MultiLineStrings: (1 - это простая ST_MultiLineString, для которой граница - это четыре конечные точки двух ее элементов ST_LineString. (2 - это простая ST_MultiLineString, поскольку пересекаются только конечные точки элементов ST_LineString. Граница представляет собой две непересекающиеся конечные точки. (3 - это непростая ST_MultiLineString, потому что внутренняя часть одного из ее элементов ST_LineString пересекается. Границей этой ST_MultiLineString являются три непересекающиеся конечные точки. (4 — простая незамкнутая ST_MultiLineString. Он не является замкнутым, поскольку его элементы ST_LineString не являются замкнутыми. Она простая, потому что ни одна из внутренних частей любого из элементов ST_LineStrings не пересекается. (5 - одиночная простая закрытая ST_MultiLineString. Он является замкнутым, поскольку все его элементы замкнуты. Он является простым, поскольку все внутренние части его элементов не пересекаются.



Функции, которые работают с ST_MultiLineStrings, включают следующее:

- ST_IsClosed - Это функция предиката возвращает значение, указывающее true, если указанная ST_MultiLineString замкнута, и значение false, если она не замкнута.
- ST_Length - Эта функция оценивает ST_MultiLineString и возвращает совокупную длину всех ее элементов ST_LineString в виде числа с двойной точностью.
- ST_NumGeometries - Эта функция возвращает количество строк в многострочной строке.

ST_Polygon

ST_Polygon – это двумерная поверхность, хранящаяся как последовательность точек, определяющих ее внешнее ограничивающее кольцо и ноль или более внутренних колец. Объекты ST_Polygon всегда являются простыми. Элементы ST_Polygon определяют объекты, которые имеют пространственный экстент, такие как участки земли, водные поверхности и области юрисдикции.

Данный рисунок показывает примеры объектов ST_Polygon: 1 – ST_Polygon, граница которого задана внешним кольцом. (2) – ST_Polygon, граница которого задана внешним кольцом и двумя внутренними кольцами. Область, заключенная между внутренних колец, относится к внешней части ST_Polygon. 3 – корректный ST_Polygon, поскольку кольца сходятся в одной касательной точке.

Внешние и любые внутренние кольца определяют границу ST_Polygon, а пространство, заключенное между кольцами, определяет внутреннюю часть ST_Polygon. Кольца объекта ST_Polygon могут пересекаться в касательной точке, но не могут пересекаться (перекрещиваться). В дополнение к другим свойствам, наследованным от суперкласса ST_Geometry, у объектов ST_Polygon есть площадь.

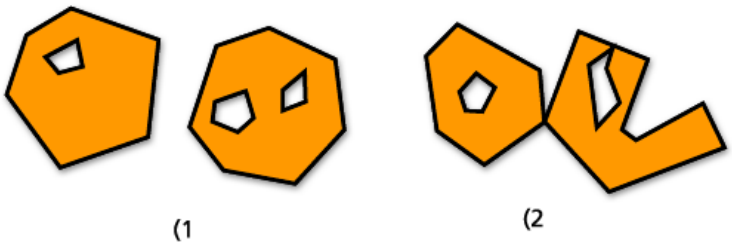
Функции, которые работают с объектами ST_Polygon, перечислены ниже:

- ST_Area – Возвращает площадь ST_Polygon как число двойной точности
- ST_Centroid – Возвращает объект ST_Point, который представляет центр конверта объекта ST_Polygon
- ST_ExteriorRing – Возвращает внешнее кольцо объекта ST_Polygon, как ST_LineString
- ST_InteriorRingN – Оценивает объект ST_Polygon и индекс, и возвращает n-ное внешнее кольцо как ST_LineString
- ST_NumInteriorRing – Возвращает количество внешних колец, которые содержит объект ST_Polygon
- ST_PointOnSurface – Возвращает объект ST_Point, который точно находится на поверхности заданного объекта ST_Polygon

ST_MultiPolygon

Границей ST_MultiPolygon является суммированная длина всех внутренних и внешних колец его элементов. Внутренняя часть ST_MultiPolygon определяется, как суммарные внутренние части всех его элементов ST_Polygon. Граница элементов объекта ST_MultiPolygon может пересекаться только в касательной точке. В дополнение к другим свойствам, наследованным от суперкласса ST_Geometry, объекты ST_MultiPolygon имеют площадь. Элементы ST_MultiPolygon определяют объекты, такие как лесной массив или несмежный участок земли, например, цепь тихоокеанских островов.

На рисунке ниже показаны примеры объектов ST_MultiPolygon: 1 - ST_MultiPolygon с двумя элементами ST_Polygon. Граница задана двумя внешними и тремя внутренними кольцами. 2 - тоже ST_MultiPolygon с двумя элементами ST_Polygon, но его граница задается двумя внешними и двумя внутренними кольцами. Два элемента ST_Polygon сходятся в касательной точке.



Функции, которые работают с ST_MultiPolygons, включают следующее:

- ST_Area - Возвращает число двойной точности, представляющее суммарное значение ST_Area для элементов ST_Polygon объекта ST_MultiPolygon.
- ST_Centroid - Возвращает точку ST_Point, которая является центром оболочки ST_MultiPolygon.
- ST_NumGeometries - Возвращает количество полигонов в мультиполигоне.
- ST_PointOnSurface - Оценивает объект ST_MultiPolygon и возвращает объект ST_Point, который гарантированно находится на поверхности одного из его элементов ST_Polygon.

Простые геометрические формы в составной геометрии

Составная геометрия состоит из отдельных простых форм.

Вам может понадобиться определить, сколько отдельных геометрий содержится в составной геометрии, такой как ST_MultiPoint, ST_MultiLineString и ST_MultiPolygon. Чтобы сделать это, используйте функцию

предиката ST_NumGeometries. Эта функция возвращает количество индивидуальных элементов в наборе геометрий.

Используя функцию ST_GeometryN, вы можете определить, какая геометрия из составной геометрии существует на N-ой позиции; N – это номер, который вы задаете функции. Например, если вы хотите получить третью точку из геометрии мультиточки, вам нужно указать 3 при запуске функции.

Чтобы вернуться к отдельным геометрическим формам и их расположению из составной геометрии в PostgreSQL, используйте функцию ST_GeomFromCollection.

Внутренняя часть, граница, внешняя часть

Все геометрии занимают положение в пространстве, определенное их содержанием, границами и внешним окружением. Внешнее окружение геометрии – это все пространство, не занятое геометрией. Содержание геометрии – это все пространство, занятое этой геометрией. Граница геометрии – это раздел между ее содержанием и внешним окружением. Подтип наследует внутренние и внешние свойства напрямую, однако свойства границ различаются для каждого случая.

Для определения границы исходного объекта ST_Geometry используйте функцию ST_Boundary.

Простой или непростой

Некоторые подтипы ST_Geometry всегда являются простыми, например ST_Points или ST_Polygons. Однако, подтипы ST_LineString, ST_MultiPoint и ST_MultiLineString могут быть как простыми, так и непростыми. Они являются простыми, если они подчиняются всем топологическим правилам, которые на них накладываются, и являются непростыми в противном случае.

Топологические правила включают следующее:

- ST_LineString является простым, если он не имеет самопересечений, и является непростым, если имеет самопересечения.
- ST_MultiPoint является простым, если никакие два из его элементов не занимают одно и то же координатное пространство (имеют одинаковые координаты x,y), и непростым в противном случае.
- Объект ST_MultiLineString является простым, если ни одна из внутренних частей всех его элементов не пересекается собственной внутренней частью, и является непростым, если любые внутренние части его элементов пересекаются.

Функция предиката ST_IsSimple используется для определения, является ли объект ST_LineString, ST_MultiPoint или ST_MultiLineString простым или нет. ST_IsSimple получает объект ST_Geometry и возвращает true, если ST_Geometry является простой, и false в противном случае.

Пустой (empty) или не пустой

При отсутствии точек геометрия является пустой. Пустая геометрия имеет нулевые значения конверта, границы, содержания и внешнего окружения. Пустая геометрия всегда является простой. Пустые строки линий и строки мультилиний имеют длину 0. Пустые полигоны и мультиполигоны имеют площадь 0.

Функция предиката ST_IsEmpty может быть использована для определения того, является ли геометрия пустой. Она анализирует объект ST_Geometry и возвращает true, если ST_Geometry является пустой геометрией, и false в противном случае.

IsClosed (замкнуты) и IsRing (кольцо)

Геометрия Linestring может быть замкнутой или кольцевой. Ломаные могут быть замкнутыми и не быть

кольцами. Вы можете определить, является ли линия действительно замкнутой, используя функцию предиката ST_IsClosed; она возвращает true, если начальная точка и конечная точка линии пересекаются. Кольца – это ломаные, которые являются замкнутыми и простыми. ST_IsRing – это функция предиката, которая может быть использована для проверки, является ли линия действительно кольцом; возвращает true, если линия замкнутая и простая.

Конверт

Каждая геометрия имеет конверт. Конверт геометрии представляет собой граничную геометрию, образуемую минимальными и максимальными координатами x,y. Для точечных геометрий, где минимальные и максимальные координаты x,y совпадают, вокруг этих координат создается прямоугольник или конверт. В линейной геометрии конечные точки линии представляют собой две стороны конверта, а другие две стороны создаются непосредственно над линией и под линией.

Функция ST_Envelope получает ST_Geometry и возвращает объект ST_Geometry, представляющий конверт исходного ST_Geometry.

Чтобы найти отдельные минимальные и максимальные значение координаты x и y для геометрии, используйте функции ST_MinX, ST_MinY, ST_MaxX и ST_MaxY.

Система пространственной привязки

Система пространственной привязки определяет матрицу трансформации координат для каждого типа геометрии. Состоит из системы координат, разрешения и допуска.

Все системы пространственной привязки, известные базе геоданных, хранятся в системной таблице базы геоданных.

Следующие функции получают информацию о системах пространственной привязки геометрий:

- ST_SRID - получает ST_Geometry и возвращает его идентификатор пространственной привязки (SRID) как целое число.
- ST_Equalsrs - Определяет, идентичны ли системы пространственной привязки двух разных классов пространственных объектов (true) или нет (false).

Размер объектов (только PostgreSQL)

Объекты (пространственные записи в таблице) требуют определенного размера пространства для хранения в байтах. Для определения того, насколько большим является каждый объект в таблице, вы можете использовать функцию ST_GeoSize.

Текстовые и бинарные определения геометрии

Чтобы получить стандартное текстовое определение или стандартное двоичное определение геометрии в определенной строке пространственной таблицы, используйте функции ST_AsText и ST_AsBinary соответственно.

Пространственные отношения

Основная функция ГИС заключается в определении пространственных отношений между объектами: Перекрываются ли они? Содержится ли один в другом? Пересекаются ли они?

Геометрии могут быть пространственно связаны по-разному. Ниже приведены примеры того, как одна геометрия может быть пространственно связана с другой:

- Геометрия A проходит через геометрию B.
- Геометрия A полностью содержится в геометрии B.
- Геометрия A полностью содержит геометрию B.
- Геометрии не пересекаются и не касаются друг друга.
- Геометрии полностью совпадают.
- Геометрии накладываются друг на друга.
- Геометрии соприкасаются в одной точке.

Чтобы определить, существуют ли эти отношения или нет, используйте функции пространственных отношений. Эти функции сравнивают следующие свойства геометрий, указанных в запросе:

- Внешняя часть (E) геометрии. Внешняя часть — это все пространство, не занятое геометрией.
- Внутренняя часть (I) геометрии. Внутренняя часть - пространство, занимаемое геометрией.
- Граница (B) геометрии. Граница - это линия между внутренней и внешней частями геометрии.

При построении запроса пространственного отношения укажите тип пространственного отношения, которое вы ищете, и геометрии, которые вы хотите сравнить. Запросы возвращаются как true или false. Другими словами, геометрии либо участвуют друг с другом в заданных пространственных отношениях, либо нет. В большинстве случаев вы используете запрос пространственного отношения для фильтрации набора результатов, помещая его в условие `WHERE`.

Например, если у вас есть таблица, где хранятся местоположения предполагаемых участков застройки, и другая таблица, в которой хранятся местоположения археологически значимых участков, вы можете выполнить запрос, чтобы убедиться, что ни один из участков застройки не пересекается с археологическими участками, и, если они пересекаются, вернуть ID этих предлагаемых застроек. В этом примере функция ST_Disjoint используется в 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
```

Этот запрос возвращает название застройки (projname) и ID места археологических раскопок (siteid), которые не являются раздельными, или другими словами - пересекаются друг с другом. Он возвращает один проект развития, Bow Wow Chow, который пересекает археологические раскопки A1009.

Реляционные функции для ST_Geometry

реляционные функции используют предикаты для тестирования разных типов пространственных отношений. Для этого тесты сравнивают отношения между следующими элементами:

- Внешняя часть (E) геометрии. Внешняя часть — это все пространство, не занятое геометрией.
- Внутренняя часть (I) геометрии. Внутренняя часть - пространство, занимаемое геометрией.
- Граница (B) геометрии. Граница - это линия между внутренней и внешней частями геометрии.

Предикаты проверяют отношения. Они возвращают 1 (Oracle и SQLite) или t (PostgreSQL), если сравнение соответствует критериям функции, и 0 (Oracle и SQLite) или f (PostgreSQL), если не соответствует. Предикат, проверяющий пространственное отношение, сравнивает пары геометрических элементов, у которых могут быть разные типы или измерения.

Предикаты сравнивают координаты x и y отправленных геометрий. Z-координаты и значения измерений, при их наличии, игнорируются. Геометрии, у которых есть координаты z или измерения, можно сравнивать с геометриями, у которых они отсутствуют.

Модель Dimensionally Extended 9 Intersection Model (DE-9IM), разработанная Клементини и др., пространственно расширяет модель 9 Intersection Model Эгенхофера и Герринга. DE-9IM — это математический подход, определяющий попарные пространственные отношения между геометрическими элементами различных типов и измерений. Эта модель выражает пространственные отношения между всеми типами геометрии в виде попарных пересечений их внутренней и внешней частей и границы с учетом измерения полученных пересечений.

При заданных геометриях a и b I(a), B(a) и E(a) - внутрення часть, граница и внешняя часть a, a I(b), B(b) и E(b) - внутренняя часть, граница и внешняя часть b. При пересечении I(a), B(a) и E(a) с I(b), B(b) и E(b) получается матрица три на три. Каждое пересечение может быть представлено геометрическими элементами разных измерений. Например, пересечение границ двух полигонов может состоять из точки и линии, и в этом случае функция dim (измерение) вернет в качестве максимального измерения 1.

Функция dim возвращает значения -1, 0, 1 и 2. -1 соответствует пустому набору, который возвращается, если пересечение не найдено или dim(Ã*ƒ˜*).

	Внутренняя	**Границы**	**Внешняя часть**
Внутренняя	dim(I(a) intersects I(b))	dim(I(a) intersects B(b))	dim(I(a) intersects E(b))
Границы	dim(B(a) intersects I(b))	dim(B(a) intersects B(b))	dim(B(a) intersects E(b))
Внешняя часть	dim(E(a) intersects I(b))	dim(E(a) intersects B(b))	dim(E(a) intersects E(b))

Пример пересечения предиката

Результаты предикатов пространственного отношения можно изучить и проверить, сравнив результаты предиката с матрицей образца, которая содержит допустимые для DE-9IM значения.

Матрица образца содержит допустимые значения для каждой ячейки матрицы пересечения. Возможные значения образца приведены ниже:

T—Должно существовать пересечение; dim = 0, 1 или 2

F—Должно отсутствовать пересечение; dim = -1

*—Не важно, есть ли пересечение; dim = -1, 0, 1, или 2

0—Должно существовать пересечение и его максимальное измерение должно быть равно 0; dim = 0

1—Должно существовать пересечение и его максимальное измерение должно быть равно 1; dim = 1

2—Должно существовать пересечение и его максимальное измерение должно быть равно 2; dim = 2

У каждого предиката есть по крайней мере одна матрицу образца, но некоторым для описания взаимосвязей различных комбинаций типов геометрии требуется несколько матриц образцов.

Матрица образца предиката ST_Within для комбинаций геометрии выглядит следующий образом:

			Геометрия b	
		Внутренняя	**Границы**	**Внешняя часть**
Геометрия a	**Внутренняя**	T	*	F
	Границы	*	*	F
	Внешняя часть	*	*	*

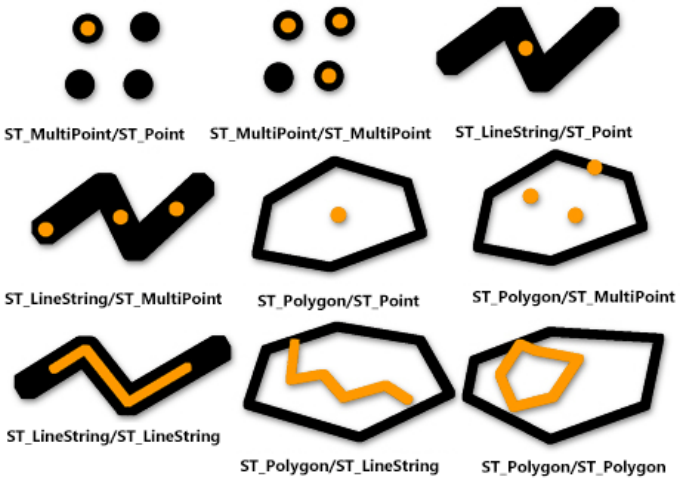
Пример матрицы образца

Предикат ST_Within равен true, если пересекаются внутренние части двух геометрий, а внутренняя часть и граница геометрии a не пересекают внешнюю часть геометрии b. Все остальные условия не имеют значения.

В следующих разделах описываются разные предикаты, которые используются для пространственных отношений. Здесь на схемах первая входная геометрия показана черным, а вторая - оранжевым.

ST_Contains

ST_Contains возвращает 1 или t (true), если вторая геометрия полностью содержится в первой. Предикат ST_Contains возвращает результат, противоположный результату предиката ST_Within.



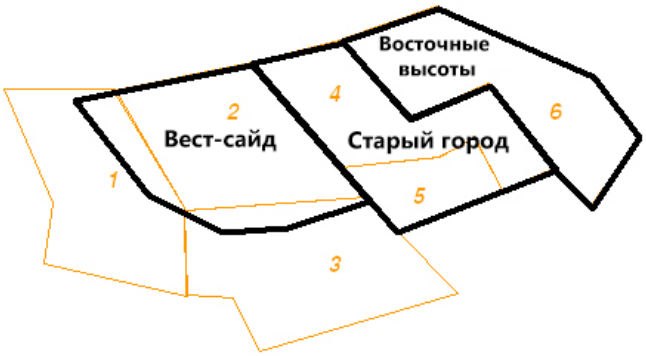
Матрица образца предиката ST_Contains указывает на то, что внутренние части обеих геометрий должны пересекаться, и внутренние части и границы второй геометрии (b) не должны пересекаться с внешней частью первой (геометрии a).

			Геометрия b	

Матрица ST_Contains

		Внутренняя	Границы	Внешняя часть
Геометрия a	**Внутренняя**	T	*	*
	Границы	*	*	*
	Внешняя часть	F	F	*

Функции ST_Within и ST_Contains находят только те геометрические элементы, которые полностью находятся внутрь других элементов. Это поможет удалить из выборки объекты, которые могут исказить результаты. В приведенном ниже примере продавец мороженого хочет определить, в каких районах проживает наибольшее количество детей (потенциальных клиентов), и ограничить свой маршрут этими районами. Поставщик сравнивает полигоны обозначенных районов с переписными участками, у которых есть атрибут общего числа детей в возрасте до 16 лет.



Если только все дети, проживающие в переписных участках 1 и 3, не проживают на осколочных участках земли, расположенных в пределах Вестсайда, включение этих участков в выборку может привести к ошибочному увеличению числа детей в районе Вестсайд. Указав, что будут включаться только переписные районы, полностью находящиеся в пределах кварталов (ST_Within = 1), продавец мороженого может сэкономить время и деньги и не заходить в эти части Вестсайда.

ST_Crosses

ST_Crosses возвращает 1 или t (true), если результатом пересечения является геометрический элемент, измерение которого на единицу меньше максимального значения измерений двух исходных геометрических элементов, а набор пересечений находится внутри обеих исходных геометрий. ST_Crosses возвращает 1 или t (true) только для сравнений ST_MultiPoint/ST_Polygon, ST_MultiPoint/ST_LineString, ST_Linestring/ST_LineString, ST_LineString/ST_Polygon и ST_LineString/ST_MultiPolygon.

ST_MultiPoint/ST_LineString

ST_MultiPoint/ST_Polygon

ST_Linestring/ST_Linestring

ST_Linestring/ST_Polygon

Следующий предикат ST_Crosses матрицы образца применим к ST_MultiPoint/ST_LineString, ST_MultiPoint/ST_MultiLineString, ST_MultiPoint/ST_Polygon, ST_MultiPoint/ST_MultiPolygon, ST_LineString/ST_Polygon и ST_LineString/ST_MultiPolygon. Матрица показывает, что внутренние области должны пересекаться и что, по крайней мере, внутренняя часть первого элемента (геометрии a) должна пересекаться с внешней областью второго (геометрии b).

			Геометрия b	
		Внутренняя	**Границы**	**Внешняя часть**
Геометрия a	**Внутренняя**	T	*	T
	Границы	*	*	*
	Внешняя часть	*	*	*

Матрица 1 ST_Crosses

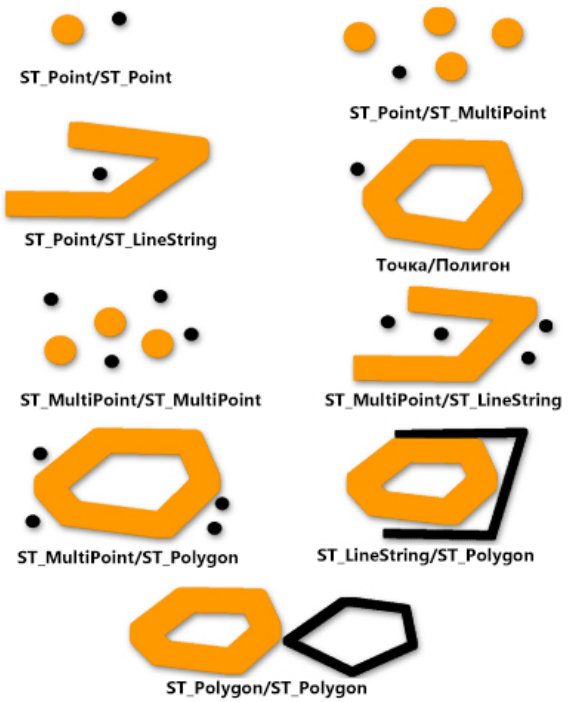
Следующая матрица предиката применяется для ST_LineString/ST_LineString, ST_LineString/ST_MultiLineString и ST_MultiLineString/ST_MultiLineString. Матрица показывает, что измерение пересечения внутренних частей должно быть равно 0 (пересечение в точке). Если измерение этого пересечения равно 1 (пересечение по линии), предикат ST_Crosses может возвратит false, а предикат ST_Overlaps - true.

			Геометрия b	
		Внутренняя	**Границы**	**Внешняя часть**
Геометрия a	**Внутренняя**	0	*	*
	Границы	*	*	*
	Внешняя часть	*	*	*

Матрица 2 ST_Crosses

ST_Disjoint

ST_Disjoint возвратит 1 или t (true), если пересечение двух геометрических элементов - пустой набор. Другими словами, геометрии считаются не пересекающимися (disjoint), если они не пересекаются друг с другом.



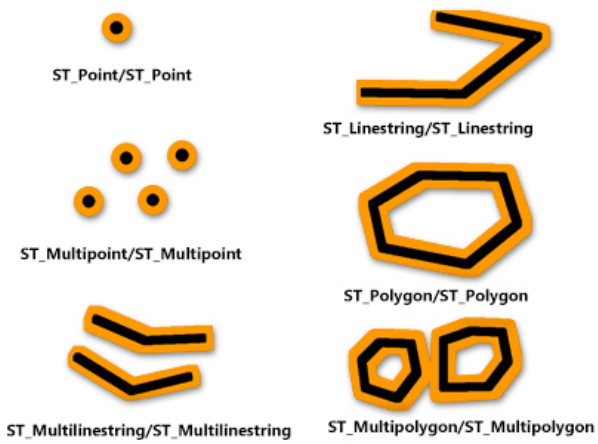
Матрица образца предиката ST_Disjoint показывает, что ни внутренние части, ни границы геометрических элементов не должны пересекаться.

			Геометрия b	
		Внутренняя	**Границы**	**Внешняя часть**
Геометрия a	**Внутренняя**	**F**	**F**	*****
	Границы	**F**	**F**	*****
	Внешняя часть	*	*	*

Матрица ST_Disjoint

ST_Equals

ST_Equals возвращает 1 или t (true), если у двух геометрических объектов одного типа одинаковые значения координат x и y. Первый и второй этажи офисного здания могут иметь одинаковые координаты x,y и, следовательно, считаться одинаковыми. Функция ST_Equals также может определить, не были ли два объекта ошибочно размещены друг над другом.



Матрица образца DE-9IM для равенства (equality) гарантирует, что внутренние части пересекаются и что ни одна из частей внутренней части и ни одна граница любого из элементов геометрии не пересекается с внешней частью другого элемента.

			Геометрия b	
		Внутренняя	**Границы**	**Внешняя часть**
Геометрия a	**Внутренняя**	T	*	F
	Границы	*	*	F
	Внешняя часть	F	F	*

Матрица ST_Equals

ST_Intersects

ST_Intersects возвращает 1 или t (true), если пересечение не возвратит пустой набор. ST_Intersects возвращает результат, противоположный результату выполнения функции ST_Disjoint.

Предикат ST_Intersects возвращает true, если условие любой из следующих матриц образцов возвращает true.

Предикат ST_Intersects возвращает true, если пересекаются внутренние части элементов геометрии.

			Геометрия b	
		Внутренняя	**Границы**	**Внешняя часть**
Геометрия a	**Внутренняя**	T	*	*
	Границы	*	*	*
	Внешняя часть	*	*	*

Матрица 1 ST_Intersects

Предикат ST_Intersects возвращает true, если внутренняя часть первого геометрического объекта пересекает границу второго.

			Геометрия b	

Матрица 2 ST_Intersects

		Внутренняя	Границы	Внешняя часть
Геометрия a	**Внутренняя**	*	T	*
	Границы	*	*	*
	Внешняя часть	*	*	*

Предикат ST_Intersects возвращает true, если граница первого геометрического объекта пересекает внутреннюю часть второго.

			Геометрия b	
		Внутренняя	**Границы**	**Внешняя часть**
Геометрия a	**Внутренняя**	*	*	*
	Границы	T	*	*
	Внешняя часть	*	*	*

Матрица 3 ST_Intersects

Предикат ST_Intersects возвращает true, если пересекаются границы геометрических объектов.

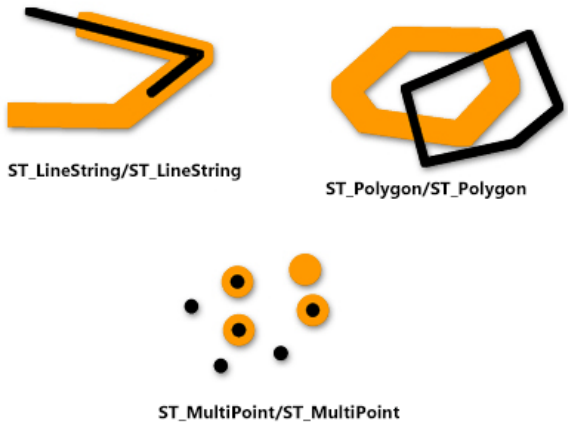
			Геометрия b	
		Внутренняя	**Границы**	**Внешняя часть**
Геометрия a	**Внутренняя**	*	*	*
	Границы	*	T	*
	Внешняя часть	*	*	*

Матрица 4 ST_Intersects

ST_Overlaps

ST_Overlaps сравнивает две геометрии одинакового измерения и возвращает 1 или t (истина), если их набор пересечений приводит к получению геометрии, отличной от обеих входных геометрий, но имеющей одинаковое измерение.

ST_Overlaps возвращает 1 или t (истина) только для геометрических элементов одинакового измерения и только тогда, когда их набор пересечений приводит к получению геометрии одинакового измерения. Другими словами, если в результате пересечения двух ST_Polygon получается ST_Polygon, наложение (overlap) возвратит 1 или t (true).



Эта матрица образца применима к наложениям ST_Polygon/ST_Polygon, ST_MultiPoint/ST_MultiPoint и ST_MultiPolygon/ST_MultiPolygon. Для таких комбинаций предикат перекрытия возвращает значение true, если внутренняя часть обоих геометрических объектов пересекает внутреннюю и внешнюю части другой.

			Геометрия b	
		Внутренняя	**Границы**	**Внешняя часть**
Геометрия a	**Внутренняя**	T	*	T
	Границы	*	*	*
	Внешняя часть	T	*	*

Матрица 1 ST_Overlaps

Следующая матрица образца применяется к наложениям ST_LineString/ST_LineString и ST_MultiLineString/ ST_MultiLineString. В этом случае пересечение геометрий должно привести к получению геометрии с размерностью 1 (еще одной ST_LineString или ST_MultiLineString). Если измерение пересечения внутренних частей равно 0 (точка), предикат ST_Overlaps возвратит false. Однако предикат ST_Crosses возвратит true.

			Геометрия b	
		Внутренняя	**Границы**	**Внешняя часть**
Геометрия a	**Внутренняя**	1	*	T
	Границы	*	*	*
	Внешняя часть	T	*	*

Матрица 2 ST_Overlaps

ST_Relate

ST_Relate возвращает значение 1 или t (true), если заданное матрицей образца пространственное отношение корректно. Значение 1 или t (true) означает, что между геометрическими объектами есть какие-то пространственные отношения.

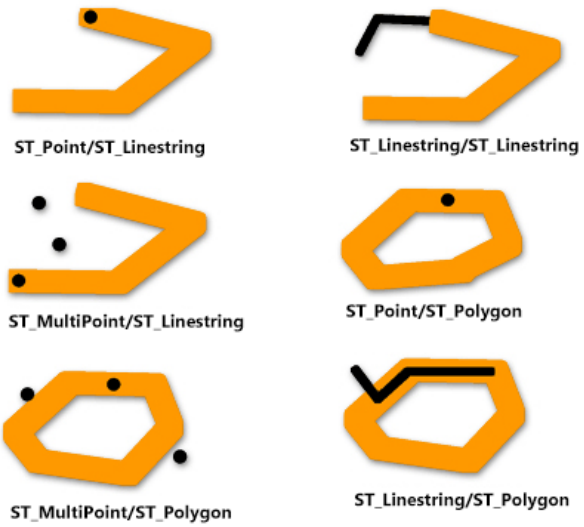
Если между внутренними частями или границами геометрических объектов a и b существуют какие-то отношения, ST_Relate будет равно true. Не имеет значения, пересекаются ли внешние части одной геометрии с внутренней частью или границей другой.

			Геометрия b	
		Внутренняя	**Границы**	**Внешняя часть**
Геометрия a	**Внутренняя**	T	T	*
	Границы	T	T	*
	Внешняя часть	*	*	*

Матрица ST_Relate

ST_Touches

ST_Touches возвращает 1 или t (истина), если ни одна из точек, общих для обеих геометрических объектов, не пересекает их внутреннюю часть. Хотя бы один геометрический объект должен быть ST_LineString, ST_Polygon, ST_MultiLineString или ST_MultiPolygon.



Матрицы образцов показывают, что предикат ST_Touches возвращает true, если внутренние части геометрических объектов не пересекаются, а граница одного объекта пересекает внутреннюю часть или границу другого.

Предикат ST_Touches возвращает значение true, если граница геометрии b пересекает внутреннюю часть геометрии a, но их внутренние части не пересекаются.

			Геометрия b	
		Внутренняя	**Границы**	**Внешняя часть**
Геометрия a	**Внутренняя**	F	T	*
	Границы	*	*	*
	Внешняя часть	*	*	*

Матрица 1 ST_Touches

Предикат ST_Touches возвращает значение true, если граница геометрии a пересекает внутреннюю часть геометрии b, но их внутренние части не пересекаются.

			Геометрия b	
		Внутренняя	**Границы**	**Внешняя часть**
Геометрия a	**Внутренняя**	F	*	*
	Границы	T	*	*
	Внешняя часть	*	*	*

Матрица 2 ST_Touches

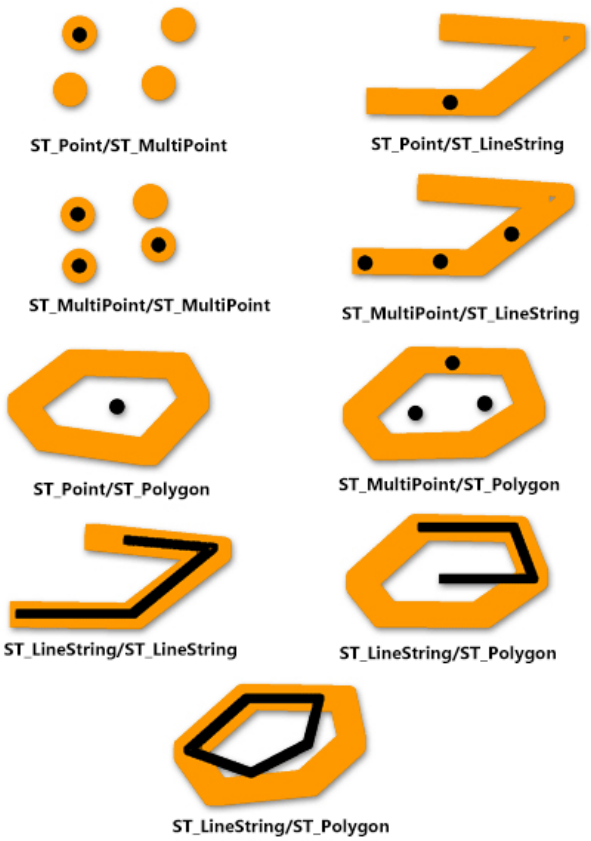
Предикат ST_Touches возвращает значение true, если границы обеих геометрий пересекаются, а их внутренние части — нет.

			Геометрия b	
		Внутренняя	**Границы**	**Внешняя часть**
Геометрия a	**Внутренняя**	F	*	*
	Границы	*	T	*
	Внешняя часть	*	*	*

Матрица 3 ST_Touches

ST_Within

ST_Within возвращает 1или t (true), если первая геометрия полностью находится внутри второй. ST_Within получает результат, прямо противоположный ST_Contains.



Матрица образца предиката ST_Within указывает на то, что внутренние части обеих геометрий должны пересекаться, и внутренние части и границы первой геометрии (a) не должны пересекаться с внешней частью второй (геометрии b).

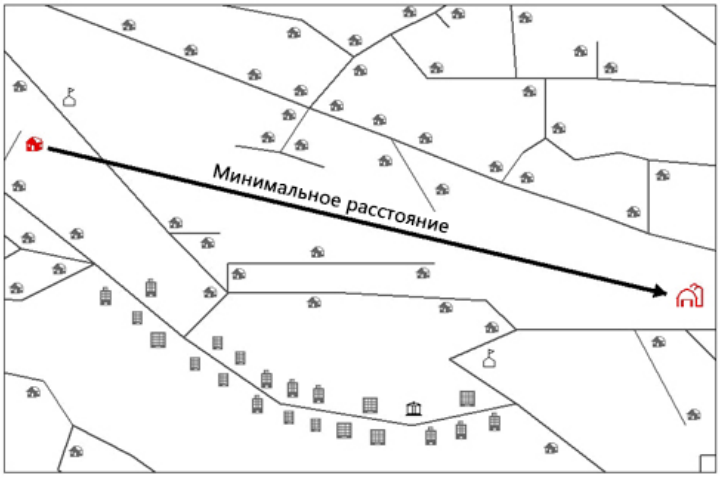
			Геометрия b	
		Внутренняя	**Границы**	**Внешняя часть**
Геометрия a	**Внутренняя**	T	*	F
	Границы	*	*	F
	Внешняя часть	*	*	*

Матрица ST_Within

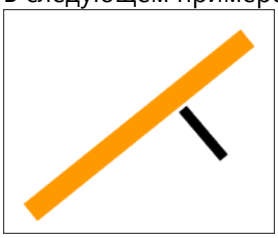
Другие пространственные отношения

Следующие функции сравнивают пространственные отношения между геометриями, но они сравнивают не только внутренние части, границы и внешние части геометрических объектов.

- ST_Distance - эта функция берет два непересекающихся геометрических объекта и возвращает минимальное расстояние между ними. Если геометрии пересекаются, функция сообщает о нулевом минимальном расстоянии.
  Минимальное расстояние, разделяющее объекты, представляет собой кратчайшее расстояние между двумя местоположениями. Например, это не расстояние, которое вы бы преодолели, если бы проехали из одного места в другое, а расстояние, вычисленное при измерении прямой линии, соединяющей две точки на карте.



- ST_DWithin - вы указываете значение расстояния, а также сравниваемые геометрические объекты. ST_DWithin возвращает значение true, если геометрии расположены в пределах заданного расстояния друг от друга.
- ST_EnvIntersects - эта функция оценивает, пересекаются ли пространственные конверты указанных геометрических объектов, тогда как ST_Intersects определяет, пересекаются ли сами эти объекты. В следующем примере конверты двух линий пересекаются, а сами линии - нет:



- ST_OrderingEquals - эта функция расширяет сравнение, выполняемое ST_Equals, позволяя также проверить, заданы ли координаты геометрии в том же порядке (x, y против y, x). Даже если геометрические объекты занимают одно и то же пространство, но при этом их координаты x и y не определены в одном и том же порядке, ST_OrderingEquals возвратит false.

Пространственные операции

Пространственные операции используют функции геометрии для получения пространственных данных в качестве входных, анализа данных, а затем создания выходных данных, которые являются производными анализа, выполненного на входных данных.

Производные данные, которые вы можете получить из пространственной операции, включают следующее:

- Полигон, являющийся буфером вокруг входного объекта
- Отдельный объект, являющийся результатом анализа, выполненного над набором геометрий
- Отдельный объект, являющийся результатом сравнения для определения части объекта, которая не находится в том же физическом пространстве, что и другой объект
- Отдельный объект, являющийся результатом сравнения для поиска частей объекта, которые пересекают физическое пространство другого объекта
- Составной объект, состоящий из частей обоих входных объектов, которые не находятся в одном физическом пространстве друг с другом
- Объект, являющийся объединением двух геометрий

Анализ, выполненный на входных данных, возвращает координаты или текстовое представление результирующей геометрии. Вы можете использовать эту информацию как часть более крупного запроса для дальнейшего анализа или использовать результаты в качестве входных данных для другой таблицы.

Например, вы можете включить буферную операцию в условие `WHERE` запроса на пересечение, чтобы определить, пересекает ли указанная геометрия область заданного размера вокруг другой геометрии.

Примечание:
В следующих примерах используются функции ST_Geometry. Чтобы узнать о конкретных геометрических функциях и синтаксисе, используемых для другой базы данных и типа пространственных данных, прочитайте документацию по этой базе данных и типу данных.

В этом примере уведомления должны быть отправлены всем владельцам собственности в пределах 1000 футов от перекрытой улицы. Условие `WHERE` создает 1000-футовый буфер вокруг улицы, которая будет закрыта. Затем этот буфер сравнивается с объектами недвижимости в этой области, чтобы увидеть, какие из них пересекаются буфером.

```
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';
```

В этом примере в условии `WHERE` выбирается одна конкретная улица (Main), затем вокруг улицы создается буфер, который сравнивается с объектами в таблице участков, чтобы определить, пересекаются ли они.* Для всех участков, пересекаемых буфером на Main Street, возвращается имя и адрес владельца участка.

Примечание:

* Порядок, в котором выполняются части условия WHERE, зависит от оптимизатора базы данных.

Ниже приведен пример использования результатов пространственной операции (объединения), выполненной над таблицами, содержащими области соседства и школьного округа, и вставки полученных объектов в другую таблицу:

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

Дополнительные сведения об использовании пространственных операторов с ST_Geometry см. в разделе Функции пространственных операций для ST_Geometry.

Функции пространственных операций для ST_Geometry

Пространственные операции используют функции геометрии для получения пространственных данных в качестве входных, анализа данных, а затем создания выходных данных, которые являются производными анализа, выполненного на входных данных.

Вы можете выполнять операции, описанные в следующих разделах, для создания объектов из входных объектов.

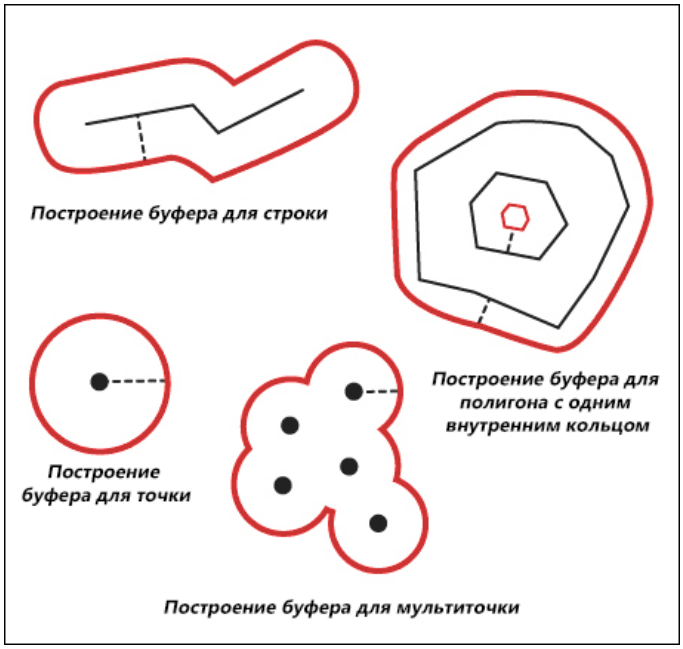
Буферизация геометрии

Функция ST_Buffer создает геометрию, окружая указанную вами геометрию на указанном вами расстоянии. Один полигон получается, когда первичная геометрия буферизуется или когда буферные полигоны коллекции расположены достаточно близко, чтобы перекрываться. Когда между элементами буферизованной коллекции существует достаточное разделение, отдельные буферные ST_Polygons приводят к ST_MultiPolygon.

Функция ST_Buffer принимает как положительные, так и отрицательные расстояния, но вы можете применять отрицательные расстояния только к двухмерным геометриям (ST_Polygon и ST_MultiPolygon). ST_Buffer использует абсолютное значение буферного расстояния, когда исходная геометрия имеет менее двух измерений, другими словами, все геометрии, отличные от ST_Polygon или ST_MultiPolygon. Положительные буферные расстояния генерируют полигональные кольца, которые находятся вдали от центра исходной геометрии и - для внешнего кольца ST_Polygon или ST_MultiPolygon - ближе к центру, если расстояние отрицательное. Для внутренних колец ST_Polygon или ST_MultiPolygon буферное кольцо направлено к центру, когда буферное расстояние положительное, и от центра, когда оно отрицательное.

Процесс буферизации объединяет перекрывающиеся полигоны буфера. Отрицательные расстояния, превышающие половину максимальной внутренней ширины полигона, приводят к пустой геометрии.

На следующей диаграмме буферы показаны красным цветом.



ConvexHull

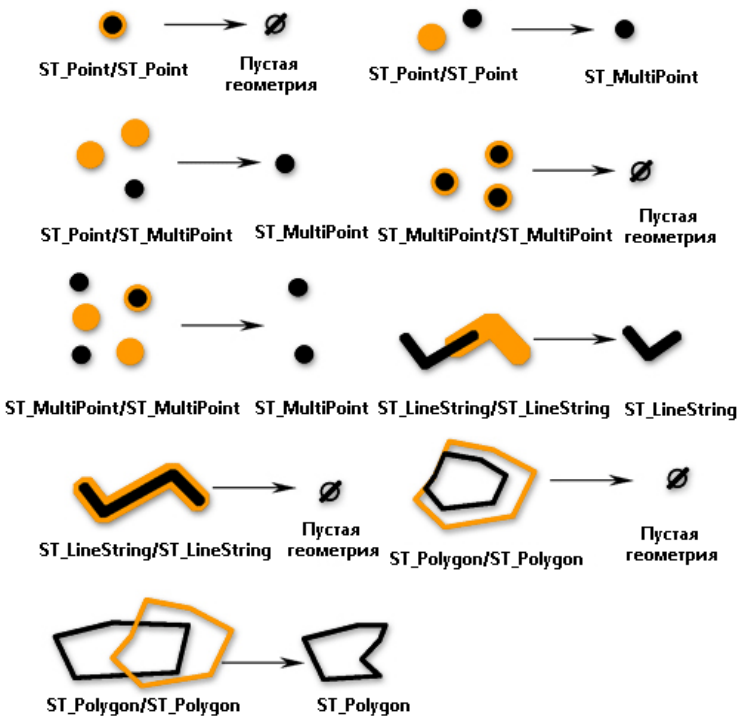
Функция ST_ConvexHull возвращает полигон выпуклой оболочки любой геометрии, которая имеет по крайней мере три вершины, образующие выпуклость. Если вершины геометрии не образуют выпуклость, ST_ConvexHull возвращает значение null. Например, использование ST_ConvexHull на линии, состоящей из двух вершин, вернет null. Точно так же использование операции ST_ConvexHull для точечного объекта вернет значение null. Создание выпуклой оболочки часто является первым шагом при замощении набора точек для создания нерегулярной триангуляционной сети (TIN).

Разница геометрий

Функция ST_Difference возвращает часть первичной геометрии, не пересекающуюся со вторичной геометрией. Это логика пространства `AND NOT`.

Функция ST_Difference работает только с геометриями одинаковой размерности и возвращает коллекцию с той же размерностью, что и исходная геометрия. Если исходные геометрии совпадают, возвращается пустая геометрия.

На диаграмме ниже первая входная геометрия выделена черным цветом, а вторая входная геометрия — оранжевым.



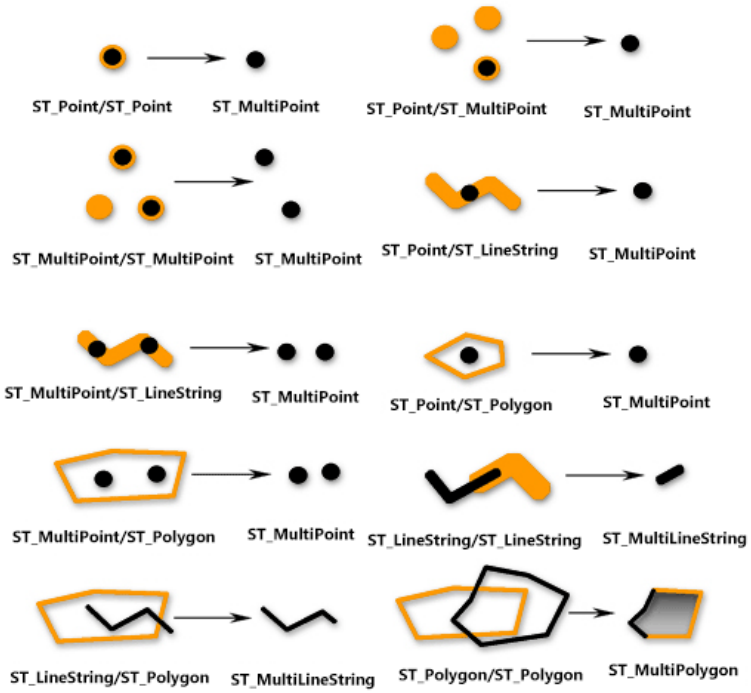
Пересечения геометрий

Функция ST_Intersection возвращает набор пересечений двух геометрий. Набор пересечений всегда возвращается как коллекция, которая является минимальным измерением исходных геометрий.

Например, для ST_LineString, пересекающей ST_Polygon, функция ST_Intersection возвращает ту часть ST_LineString, которая является общей для внутренней части и границы ST_Polygon, как ST_MultiLineString. ST_MultiLineString содержит более одной ST_LineString, если исходная ST_LineString пересекает ST_Polygon

двумя или более прерывистыми сегментами. Если геометрии не пересекаются или в результате пересечения размер меньше, чем у обеих исходных геометрий, возвращается пустая геометрия.

На следующем рисунке показаны примеры функции ST_Intersection. Первые входные геометрии черные, а вторые входные геометрии оранжевые.

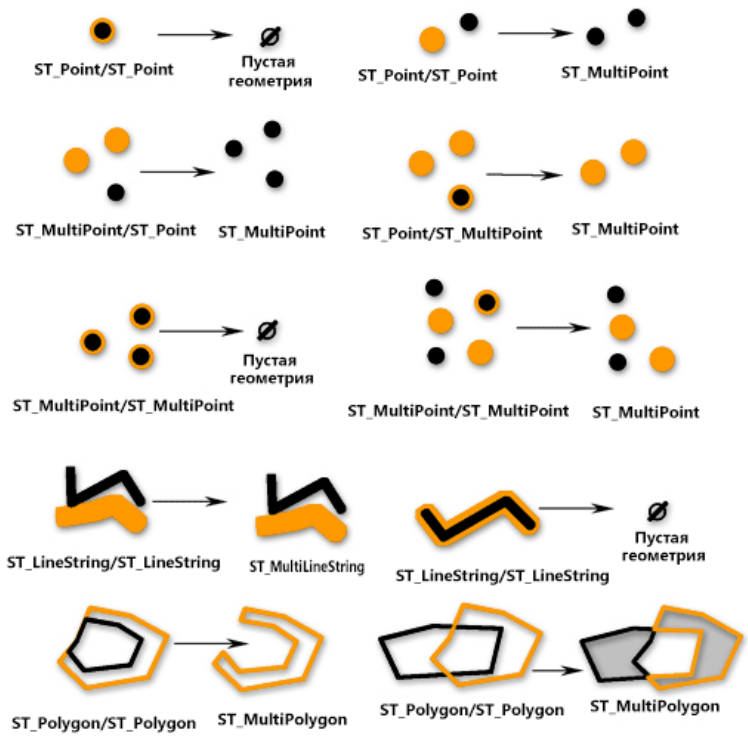


Симметричная разность геометрий

Функция ST_SymmetricDiff возвращает части исходной геометрии, которые не являются частью набора пересечений. Это логика пространства XOR.

Исходная геометрия должна быть в одной размерности. Если геометрии равны, функция ST_SymmetricDiff возвращает пустую геометрию; в противном случае функция возвращает результат в виде коллекции.

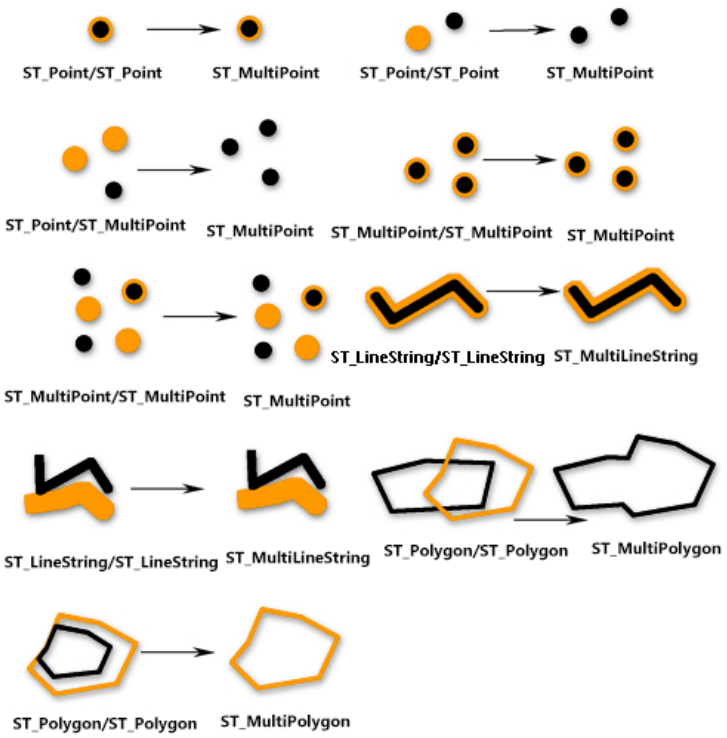
На диаграмме ниже первая входная геометрия выделена черным цветом, а вторая входная геометрия — оранжевым.



Объединение геометрий

Функция ST_Union возвращает набор объединения двух геометрий. Это булева логика пространства OR. Исходные геометрии должны иметь одну размерность. ST_Union всегда возвращает результат в виде коллекции.

На диаграмме ниже первая входная геометрия выделена черным цветом, а вторая входная геометрия — оранжевым.



Агрегации

Операции агрегации возвращают одну геометрию в результате анализа набора геометрий. Функция ST_Aggr_ConvexHull возвращает мультиполигон, состоящий из полигонов выпуклой оболочки каждой из входных геометрий. Любая входная геометрия с менее чем тремя вершинами не будет иметь выпуклой оболочки. Если все входные геометрии имеют менее трех вершин, ST_Aggr_ConvexHull возвращает значение null.

Функция ST_Aggr_Intersection возвращает единую геометрию, представляющую собой совокупность пересечений всех входных геометрий.

ST_Aggr_Intersection находит пересечение нескольких геометрий, тогда как ST_Intersection находит пересечение только двух геометрий. Например, если вы хотите найти недвижимость, на которую распространяются различные конкретные услуги, такие как конкретный школьный округ, телефонная связь и поставщик высокоскоростного Интернета, и которая представлена конкретным членом совета, вам нужно найти пересечение всех областей этих услуг. Поиск пересечения только двух из этих областей не вернет всю необходимую информацию, поэтому вы должны использовать функцию ST_Aggr_Intersection, чтобы все области можно было оценить в одном запросе.

В качестве еще одного примера, при поиске пересечений линий и точек в двух классах пространственных объектов каждая функция будет возвращать следующее:

- ST_Intersection — геометрия ST_Point возвращается для каждого пересечения.
- ST_Aggr_Intersection — возвращается одна геометрия ST_MultiPoint, состоящая из всех точек пересечения. (Однако, если пересекаются только один точечный объект и один линейный объект, вы получите геометрию ST_Point.)

Функция ST_Aggr_Union возвращает одну геометрию, которая представляет собой объединение всех

предоставленных геометрий.

Входные геометрии должны быть одного типа; например, вы можете объединить ST_LineStrings с ST_LineStrings или вы можете объединить ST_Polygons с ST_Polygons, но вы не можете объединить класс объектов ST_LineString с классом объектов ST_Polygon.

Геометрия, полученная в результате агрегатного объединения, обычно является коллекцией. Например, если вы хотите объединить все свободные участки меньше половины акра, возвращаемая геометрия будет мультиполигоном, если только все участки, соответствующие критериям, не являются смежными, тогда будет возвращен один полигон.

Минимальное расстояние

Предыдущие функции возвращали новую геометрию. Функция ST_Distance выполняет пространственную операцию — оценивает минимальное расстояние между двумя геометриями — но не возвращает новую геометрию.

Параметрические окружности, эллипсы и клинья

Вы можете создавать параметрические окружности, эллипсы и клинья в столбцах ST_Geometry с использованием функции ST_Geometry.

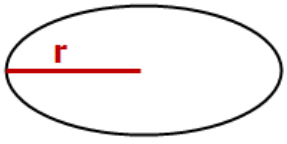
Параметрические окружности, эллипсы и клинья являются полигонами, определяемыми специфическими параметрами, такими как значения координат, углы и радиусы. База данных хранит эти параметры вместо вершин и линий. Из-за хранения определяющих кривую параметров параметрические кривые могут иметь более высокую точность и занимать меньше места, по сравнению с хранением их в виде полигонов-многоугольников. Использование параметрических кривых также позволяет включить значение координат Z и измерений M.

При создании окружности требуются семь параметров:

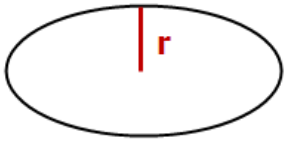
- Значение координаты X центра окружности
- Значение координаты Y центра окружности
- Значение координаты Z центра окружности
- Значение M
- Радиус создаваемой окружности
- Число точек, использующихся для определения формы окружности
  Минимально возможное число точек – 9. Если вы не укажете число точек, будет использовано значение по умолчанию, равное 50. Эти точки не сохраняются с геометрией, а создаются, когда создается окружность для проверки геометрии.
- Идентификатор пространственной привязки (SRID), использующийся для размещения окружности в пространстве

Девять параметров используются при создании эллипса:

- Координата X центра эллипса
- Координата Y центра эллипса
- Координата Z центра эллипса
- Значение M
- Большая полуось эллипса
  Большая полуось – это самый большой радиус эллипса. Длина большой полуоси должна быть больше длины малой полуоси.



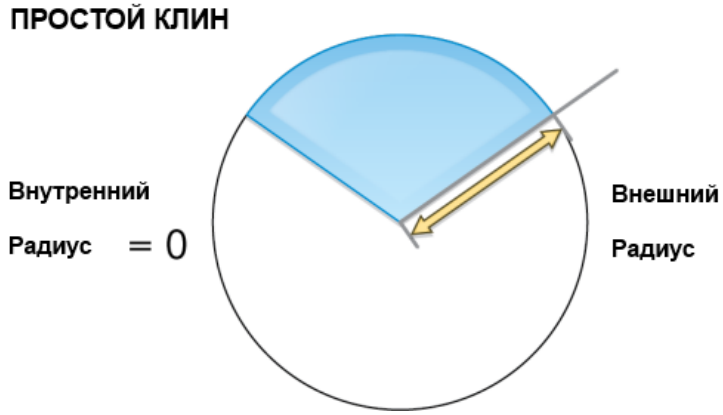
- Малая полуось эллипса
  Малая полуось – это наименьший радиус эллипса. Длина малой полуоси должна быть положительной.



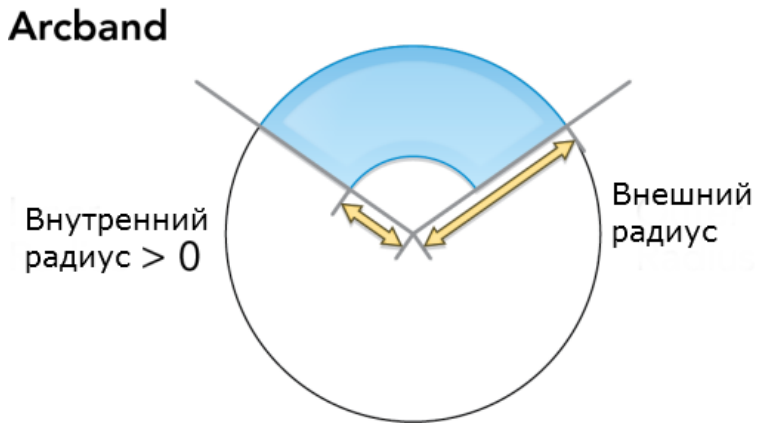
- Угол вращения эллипса
  Угол вращения указывается в градусах и должен быть положительным числом, меньшим 360. Угол вращения отсчитывается по часовой стрелке.
- Количество точек, использующихся для определения эллипса.
  Минимально возможное число точек – 9. Если вы не укажете число точек, будет использовано значение по умолчанию, равное 50. Эти точки не сохраняются с геометрией, а создаются, когда создается эллипс для проверки геометрии.
- SRID используется для размещения эллипса в пространстве.

При создании клина используются 10 параметров:

- Координата X центральной точки окружности, определяющей клин
- Координата Y центральной точки окружности, определяющей клин
- Координата Z центральной точки окружности, определяющей клин
- Значение M
- Начальный угол клина
  Начальный угол определяет начало клина в градусах, отсчитанных от нулевого значения против часовой стрелки.
- Конечный угол клина
  Конечный угол определяет конец клина в градусах, отсчитанных от нулевого значения против часовой стрелки.
- Внешний радиус
  Внешний радиус определяет расстояние из центра окружности до самой удаленной точки клина.
- Внутренний радиус
  Внутренний радиус определяет расстояние от центра окружности до ближайшей точки клина, т.е. определяет начало клина. Если внутренний радиус равен нулю, это будет форма простого клина.



Если внутренний радиус больше нуля, это будет форма arcband.



- Число точек, определяющих клин
  Минимально возможное число точек – 9. Если вы не укажете число точек, будет использовано значение по умолчанию, равное 80. Эти точки не сохраняются с геометрией, а создаются, когда создается клин для проверки геометрии.
- SRID используется для размещения клина в пространстве

Все радиусы, в том числе большая и малая полуоси, внутренние и внешние радиусы, определяются в единицах измерения, указанных в координатной привязке SRID.

Чтобы познакомиться с синтаксисом и примерами создания параметрической окружности, эллипса или клина, обратитесь к функции ST_Geometry.

ST_Aggr_ConvexHull

Примечание:
Только Oracle и SQLite

Описание

ST_Aggr_ConvexHull создает единую геометрию, которая является выпуклой оболочкой геометрии, полученной в результате объединения всех входных геометрий. По сути, ST_Aggr_ConvexHull эквивалентна ST_ConvexHull(ST_Aggr_Union(геометрия)).

Синтаксис

Oracle

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

SQLite

```
st_aggr_convexhull (geometry st_geometry)
```

Тип возвращаемого значения

Oracle

ST_Geometry

SQLite

Geometryblob

Пример

В примере создается таблица service_territories и запускается выражение SELECT, которое объединяет все геометрические элементы, тем самым создавая единую геометрическую форму, представляющую выпуклую оболочку объединения всех фигур.

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)
);

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))
```

ST_Aggr_Intersection

Примечание:
Только Oracle и SQLite

Определение

ST_Aggr_Intersection возвращает единую геометрию, объединяющую пересечения всех входных геометрий.

Синтаксис

Oracle

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

SQLite

```
st_aggr_intersection (geometry1 geometryblob)
```

Тип возврата

Oracle

ST_Geometry

SQLite

Geometryblob

Пример:

В этом примере биолог пытается найти пересечение местообитаний трех диких животных.

Oracle

Сначала создайте таблицу для хранения местообитаний.

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

Затем вставьте в таблицу три полигона.

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

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)
);
```

Выберите пересечение местообитаний.

```
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

Сначала создайте таблицу для хранения местообитаний.

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

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

Затем вставьте в таблицу три полигона.

```
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)
);
```

Выберите пересечение местообитаний.

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

ST_Aggr_Union

Определение

ST_Aggr_Union возвращает геометрию, представляющую собой объединение всех входных геометрий.

Синтаксис

Oracle и PostgreSQL

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

SQLite

```
st_aggr_union(geometry geometryblob)
```

Тип возвращаемого значения

Oracle и PostgreSQL

ST_Geometry

SQLite

Geometryblob

Пример:

Аналитик рынка должен создать одну геометрию всех областей обслуживания, в которых продажи превысили 1000 штук. В этом примере вы создадите таблицу service_territories1 и заполните ее значениями показателей продаж. Затем вы используете параметр st_aggr_union в инструкции SELECT, чтобы возвратить мультиполигон, являющийся объединением всех геометрий, для которых показатель объема продаж больше или равен 1000 штукам.

Oracle и 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)
);
```

```
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
```

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

ST_Area

Определение

ST_Area возвращает площадь полигона или мультиполигона.

Синтаксис

Oracle и 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)
```

Тип возврата

Двойная точность

Пример:

Городскому инженеру нужен список площадей зданий. Для создания списка ГИС-техник выбирает ID здания и площадь контура каждого здания.

Контуры зданий хранятся в таблице bfp.

Для выполнения запроса инженера ГИС-техник выбирает уникальный ключ, building_id, и площадь контура каждого здания из таблицы bfp.

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
```

```
          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
```

ST_AsBinary

Определение

Функция ST_AsBinary принимает объект геометрии и возвращает его WKB-представление.

Синтаксис

Oracle и PostgreSQL

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

SQLite

```
st_asbinary (geometry geometryblob)
```

Тип возврата

Oracle и PostgreSQL

ST_Geometry

SQLite

Geometryblob

Пример:

В этом примере столбец WKB записи 1111 заполняется содержимым столбца GEOMETRY записи 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
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)
```

ST_AsText

Определение

ST_AsText берет геометрию и возвращает ее стандартное текстовое представление.

Синтаксис

Oracle и PostgreSQL

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

SQLite

```
st_astext (geometry geometryblob)
```

Тип возврата

Oracle

CLOB

PostgreSQL и SQLite

Текст

Пример:

Функция ST_AsText преобразует точку hazardous_sites в текстовое описание.

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)
```

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)
```

ST_Boundary

Описание

ST_Boundary принимает геометрию и возвращает ее комбинированную границу в виде объекта геометрии.

Синтаксис

Oracle и PostgreSQL

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

SQLite

```
st_boundary (geometry geometryblob)
```

Тип возвращаемого значения

Oracle и PostgreSQL

ST_Geometry

SQLite

Geometryblob

Пример

В этом примере таблица границ создана с двумя столбцами: тип и геометрия. Последующие выражения INSERT добавляют по одной записи для каждой из геометрий подкласса. Функция ST_Boundary извлекает границу каждого подкласса, хранящегося в столбце геометрии. Обратите внимание, что размер итоговой геометрии всегда на единицу меньше, чем размер входной геометрии. Точки и мультиточки всегда приводят к границе, которая представляет собой пустую геометрию, измерение –1. Линии и мультилинии возвращают многоточечную границу, измерение 0. Полигон или мультиполигон всегда возвращает мультилинейную границу, измерение 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)
);
```

```
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 (
```

```
 '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',
```

```
 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))
```

ST_Buffer

Описание

ST_Buffer принимает объект геометрии и расстояние и возвращает объект геометрии, который представляет буфер вокруг объекта.

Синтаксис

`Unit_name` - это единица измерения буферного расстояния (например, `метры`, `километры`, `футы` или `мили`). См. первую таблицу в `Projected coordinate system tables.pdf`, доступ к которой можно получить из раздела Системы координат, проекции и преобразования.

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)
```

Тип возвращаемого значения

Oracle и PostgreSQL

ST_Geometry

SQLite

Geometryblob

Пример

В этом примере создаются две таблицы, sensitive_areas и hazardous_sites; таблицы заполняются, ST_Buffer используется для создания буфера вокруг полигонов в таблице hazardous_sites и выполняется поиск пересечения этих буферов с полигонами sensitive_areas.

Oracle

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

```
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)
);
```

```
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
```

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

ST_Centroid

Определение

ST_Centroid принимает объект полигона, мультиполигона или мультилинии и возвращает точку, являющуюся центром огибающей геометрии. Это означает, что точка центроида находится посередине между минимальным и максимальным экстентом x и y геометрии.

Синтаксис

Oracle и 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)
```

Тип возвращаемого значения

ST_Point

Пример

Городской ГИС-техник хочет отобразить мультиполигоны контуров зданий как точки на графике плотности зданий. Контуры зданий хранятся в таблице bfp, созданной и заполненной выражениями, показанными для каждой базы данных.

Oracle

```
--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
```

```
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)
);
```

```
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)
```

ST_Contains

Определение

ST_Contains берет два объекта геометрии, а возвращает 1 (Oracle и SQLite) либо t (PostgreSQL), если первый объект целиком содержит второй. В противном случае возвращается значение 0 (Oracle и SQLite) либо f (PostgreSQL).

Синтаксис

Oracle и PostgreSQL

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

SQLite

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

Тип возврата

Логический

Пример:

В примерах ниже создаются две таблицы. Первая, buildingfootprints, содержит контуры зданий города, а другая, lots, содержит участки. Городской инженер хочет убедиться, что все контуры зданий находятся полностью в пределах своих участков.

Городской инженер использует ST_Intersects и ST_Contains для выбора зданий, которые не лежат полностью в пределах одного участка.

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 (
 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)
```

```
);

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 (
 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
```

ST_ConvexHull

Описание

ST_ConvexHull возвращает выпуклую оболочку объекта ST_Geometry.

Синтаксис

Oracle и PostgreSQL

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

SQLite

```
st_convexhull (geometry1 geometryblob)
```

Тип возвращаемого значения

Oracle и PostgreSQL

ST_Geometry

SQLite

Geometryblob

Пример

В этих примерах создается таблица sample_geometries с тремя столбцами: id, spatial_type и geometry. Поле spatial_type хранит тип геометрии, созданной в столбце geometry. В таблицу вставляются три объекта — linestring, полигон и мультиточка.

Выражение SELECT, включающее функцию ST_ConvexHull, возвращает выпуклую оболочку каждой геометрии.

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)
);

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))
```

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',
 '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))
```

ST_CoordDim

Определение

ST_CoordDim возвращает размерность значений координат для столбца геометрии.

Синтаксис

Oracle и PostgreSQL

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

SQLite

```
st_coorddim (geometry1 geometryblob)
```

Тип возврата

Целочисленное (Integer)

2 = координаты x,y

3 = координаты x,y,z или x,y,m

4 = координаты x,y,z,m

Пример:

В этих примерах создается таблица coorddim_test со столбцами geotype и g1. В столбце geotype содержится имя подкласса геометрии и размер, хранимые в столбце g1 геометрии.

Инструкция SELECT указывает имя подкласса, хранимого в столбце geotype с измерением координат этой геометрии.

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)
);
```

```
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;
```

```
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,
```

```
 '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
```

```
Point                          2
```

ST_Crosses

Определение

ST_Crosses берет два объекта ST_Geometry и возвращает значение 1 (Oracle и SQLite) либо t (PostgreSQL), если в результате их пересечения возникает объект геометрии, размерность которого на единицу меньше, чем максимальная размерность исходных объектов. Объект пересечения должен содержать точки, являющиеся внутренними для исходных геометрий и не равные какому-либо из исходных объектов. В противном случае возвращается значение 0 (Oracle и SQLite) или f (PostgreSQL).

Синтаксис

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

Oracle и PostgreSQL

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

SQLite

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

Тип возврата

Логический

Пример:

Окружное управление рассматривает новый закон, говорящий о том, что все объекты хранения токсичных отходов в округе могут располагаться на определенном расстоянии от водоемов. У ГИС-менеджера округа есть точное представление рек и ручьев, хранимых как строки linestring в таблице waterways, но у него есть только точечное расположение для каждого объекта хранения токсичных отходов.

Чтобы определить, нужно ли уведомить главу округа о том, что какой-то из объектов нарушает предлагаемый закон, ГИС-менеджер должен буферизовать расположения hazardous_sites, чтобы посмотреть, пересекают ли реки или ручьи полигоны буфера. Предикат cross сравнивает буферизованные точки hazardous_sites с таблицей waterways, возвращая только те записи, в которых водоем пересекает предложенный округом радиус.

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,
```

```
 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 (
```

```
 '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',
```

```
 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
```

ST_Curve

Примечание:
Только Oracle и SQLite

Описание

ST_Curve создает объект кривой из стандартного текстового представления (WKT).

Синтаксис

Oracle

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

SQLite

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

Тип возвращаемого значения

ST_LineString

Пример

В этом примере создается таблица с геометрией кривой, вставляются в нее значения и выбирается один объект из таблицы.

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
);

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)
```

ST_Difference

Определение

ST_Difference берет два объекта геометрии, а возвращает один, который представляет собой разность исходных объектов.

Синтаксис

Oracle и PostgreSQL

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

SQLite

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

Тип возврата

Oracle и PostgreSQL

ST_Geometry

SQLite

Geometryblob

Пример:

В следующих примерах городской инженер должен знать общую площадь незастроенных участков города, поэтому вычисляется сумма площадей участков, образовавшихся после удаления застроенной территории.

Городской инженер объединяет таблицы footprints и lots в lot_id и берет разность лотов и контуров.

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)
);

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,
 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)
);

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
```

ST_Dimension

Описание

ST_Dimension возвращает измерение геометрического объекта. В этом случае измерение ссылается на длину и ширину. Например, точка не имеет ни длины, ни ширины, поэтому её измерение равно 0; тогда как линия имеет длину, но не имеет ширины, поэтому её измерение равно 1.

Синтаксис

Oracle и PostgreSQL

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

SQLite

```
st_dimension (geometry1 geometryblob)
```

Тип возвращаемого значения

Целочисленные

Пример

Таблица dimension_test создается со столбцами geotype и g1. Столбец geotype хранит имя подкласса, который находится в столбце геометрии g1.

Выражение SELECT перечисляет имя подкласса, хранящегося в столбце geotype с измерением данного геотипа.

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 (
```

```
 '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 (
```

```
 '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',
```

```
 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
```

ST_Disjoint

Определение

ST_Disjoint берет две геометрии и возвращает 1 (Oracle и SQLite) или t (PostgreSQL), если пересечение двух геометрий представляет собой пустое множество. В противном случае возвращается 0 (Oracle и SQLite) или f (PostgreSQL).

Синтаксис

Oracle и PostgreSQL

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

SQLite

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

Тип возврата

Логический

Пример:

В этом примере создаются две таблицы (distribution_areas и factories), и в каждую из них вставляются значения. После этого вокруг заводов создается буфер, а функция st_disjoint используется для поиска буферов завода, не пересекающих ареалы.

Подсказка:

Вы могли использовать функцию ST_Intersects в этом запросе, приравняв результат функции к 0, так как ST_Intersects и ST_Disjoint возвращают противоположные значения. Функция ST_Intersects использует пространственный индекс при вычислении запроса, а функция ST_Disjoint не использует индекс.

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,
 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;
```

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
);
```

```
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
```

ST_Distance

Определение

ST_Distance возвращает расстояние между двумя геометриями. Расстояние измеряется от наиближайших вершин до двух геометрий.

Синтаксис

Oracle и 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)
```

Допустимыми являются следующие наименования единиц измерения:

Миллиметр	Дюйм	Ярд	Линк
Сантиметр	Inch_US	Yard_US	Link_US
Дециметр	Фут	Yard_Clarke	Link_Clarke
Метр	Foot_US	Yard_Sears	Link_Sears
Meter_German	Foot_Clarke	Yard_Sears_1922_Truncated	Link_Sears_1922_Truncated
Километр	Foot_Sears	Yard_Benoit_1895_A	Link_Benoit_1895_B
50_Kilometers	Foot_Sears_1922_Truncated	Yard_Indian	Чейн
150_Kilometers	Foot_Benoit_1895_A	Yard_Indian_1937	Chain_US
Vara_US	Foot_1865	Yard_Indian_1962	Chain_Clarke
Смут	Foot_Indian	Yard_Indian_1975	Chain_Sears
	Foot_Indian_1937	Морская сажень	Chain_Sears_1922_Truncated
	Foot_Indian_1962	Mile_US	Chain_Benoit_1895_A
	Foot_Indian_1975	Statute_Mile	Род
	Foot_Gold_Coast	Nautical_Mile	Rod_US
	Foot_British_1936	Nautical_Mile_US	
		Nautical_Mile_UK	

Тип возвращаемого значения

Двойная точность

Пример

Созданы и заполнены две таблицы – study1 и zones. Функция ST_Distance используется для определения расстояния между границей каждой подзоны и полигонами таблицы study1, имеющими код 400. Поскольку там содержатся три зоны, будут возвращены три записи.

Если не указаны единицы измерения, ST_Distance использует единицы системы координат проекции данных. В первом примере применяются десятичные градусы. В последних двух указаны километры, поэтому расстояние возвращается в километрах.

Oracle и 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))
```

```
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
);
```

```
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
```

ST_DWithin

Описание

ST_DWithin получает на вход две геометрии и возвращает true, если они находятся в пределах заданного расстояния друг от друга, иначе возвращается false. Система пространственной привязки геометрий определяет используемые для расстояния единицы измерения. Поэтому, геометрии для ST_DWithin должны использовать одну и ту же проекцию координат и ID пространственной привязки (SRID).

Синтаксис

Oracle и PostgreSQL

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

SQLite

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

Тип возвращаемого значения

Boolean

Примеры

В следующих примерах создаются две таблицы, в которые помещаются объекты. Далее функция ST_DWithin используется в двух разных выражениях SELECT, чтобы определить, находится ли точка из первой таблицы в пределах 100 метров от полигона из второй таблицы, и в одном выражении, чтобы определить, какие объекты находятся в пределах 300 метров друг от друга.

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)
 )
;

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)
 )
;
```

Затем, используйте ST_DWithin для определения, какие объекты в каждой таблице находятся в пределах 100 метров друг от друга, а какие - нет. Функция ST_Distance включена в это выражения для отображения реального расстояния между объектами.

```
--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;
```

Выражение возвращает следующее:

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

В следующем примере, ST_DWithin используется для поиска объектов, которые находятся на расстоянии 300 метров друг от друга:

```
--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;
```

Второе выражение выборки возвращает следующее при работе с данными в 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
```

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)
 )
;
```

Затем, используйте ST_DWithin для определения, какие объекты в каждой таблице находятся в пределах 100 метров друг от друга, а какие - нет. Функция ST_Distance включена в это выражения для отображения реального расстояния между объектами.

```
--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;
```

Выражение возвращает следующее:

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

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

В следующем примере, ST_DWithin используется для поиска объектов, которые находятся на расстоянии 300 метров друг от друга:

```
--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;
```

Это второе выражение выборки возвращает следующее:

```
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
```

```
 (
  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)
 )
;
```

Затем, используйте ST_DWithin для определения, какие объекты в каждой таблице находятся в пределах 100 метров друг от друга, а какие - нет. Функция ST_Distance включена в это выражения для отображения реального расстояния между объектами.

```
--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;
```

Выражение возвращает следующее:

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

В следующем примере, ST_DWithin используется для поиска объектов, которые находятся на расстоянии 300 метров друг от друга:

```
--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
```

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

Это второе выражение выборки возвращает следующее:

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

ST_EndPoint

Определение

ST_EndPoint возвращает последнюю точку линии linestring.

Синтаксис

Oracle и PostgreSQL

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

SQLite

```
st_endpoint (line1 geometryblob)
```

Тип возврата

ST_Point

Пример:

В таблице endpoint_test хранится целочисленный столбец gid, который уникально определяет каждую строку, и столбец ln1 ST_LineString, в котором хранятся строки linestring.

Инструкция INSERT вставляет две строки linestring в таблицу endpoint_test. У первой строки linestring нет z-координат или измерений, а у второй есть.

Запрос возвращает столбец gid и созданную функцией ST_EndPoint геометрию ST_Point.

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;
```

```
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
```

```
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)
```

ST_Entity

Определение

ST_Entity возвращает пространственный тип объекта геометрии. Пространственный тип – это значение, которое хранится в соответствующем поле объекта геометрии.

Синтаксис

Oracle и PostgreSQL

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

SQLite

```
st_entity (geometry1 geometryblob)
```

Тип возвращаемого значения

Возвращается номер (Oracle) либо целочисленное значение (SQLite и PostgreSQL), соответствующие следующим типам объекта:

0	объект nil
1	точка
2	линия (в том числе неструктурированные линии)
4	linestring
8	область
257	мультиточечный
258	мультилиния (в том числе неструктурированные линии)
260	мультилиния
264	мультиплощадь

Пример

В данном примере создается таблица, в которую добавляются различные геометрии. ST_Entity запускается для таблицы, чтобы возвратить подтип геометрии для каждой записи в таблице.

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,
 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;
```

Выражение выборки SELECT возвращает следующие значения:

```
 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;
```

Выражение выборки SELECT возвращает следующие значения:

```
 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;
```

Выражение выборки SELECT возвращает следующие значения:

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

ST_Envelope

Описание

ST_Envelope возвращает минимальный ограничивающий прямоугольник геометрического объекта в виде полигона.

Более подробно:
Эта функция соответствует спецификации Open Geospatial Consortium (OGC) Simple Features, в которой указано, что ST_Envelope возвращает полигон. Для работы с особыми случаями точечной геометрии или горизонтальными или вертикальными линиями функция ST_Envelope возвращает полигон вокруг этих форм, который представляет собой небольшой допуск конверта, рассчитанный на основе масштабного коэффициента XY для системы пространственной привязки геометрии. Этот допуск вычитается из минимальных значений x и y и добавляется к максимальным координатам x и y, чтобы получить полигон вокруг этих форм.

Синтаксис

Oracle и PostgreSQL

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

SQLite

```
st_envelope (geometry1 geometryblob)
```

Тип возвращаемого значения

Oracle и PostgreSQL

ST_Geometry

SQLite

Geometryblob

Пример

Столбец геотипа таблицы envelope_test хранит имя подкласса геометрии, которая хранится в столбце g1. Выражения INSERT вставляют каждый подкласс геометрии в таблицу envelope_test.

Затем запускается функция ST_Envelope, чтобы вернуть конверт полигона вокруг каждой геометрии.

Oracle

```
--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 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
-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))
```

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
```

```
-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)
);
```

```
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))
```

ST_EnvIntersects

Примечание:
Только Oracle и SQLite

Определение

ST_EnvIntersects возвращает 1 (true), если ограничивающие прямоугольники двух геометрий пересекаются. В противном случае возвращается значение 0 (false).

Синтаксис

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)
```

Тип возврата

Логический

Пример:

В этом примере ищется геометрия, ограничивающий прямоугольник которой пересекается указанным полигоном.

Первое выражение SELECT сравнивает ограничивающие прямоугольники двух геометрий и сами геометрии с целью определения, пересекаются ли эти объекты (или ограничивающие их прямоугольники).

Второе выражение SELECT использует ограничивающий прямоугольник для определения того, попадают ли какие-нибудь объекты внутрь прямоугольника, с которым работает условие WHERE выражения 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)
);
```

```
--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
```

```
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
```

ST_Equals

Определение

ST_Equals сравнивает две геометрии и возвращает 1 (Oracle и SQLite) либо t (PostgreSQL), если геометрии идентичны. В противном случае возвращается значение 0 (Oracle и SQLite) либо f (PostgreSQL).

Синтаксис

Oracle и PostgreSQL

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

SQLite

```
st_equals (geometry1 geometryblob, geometry2 geometryblob)
```

Возвращаемый тип

Логический

Пример:

Городской ГИС-специалист подозревает, что какие-то данные в таблице studies были продублированы. Для подтверждения подозрений он запрашивает таблицу, чтобы определить, есть ли равные мультиполигоны объектов.

Таблица studies была создана и заполнена следующими инструкциями. Столбец id таблицы уникально определяет изучаемые территории, а в столбце shape хранится геометрия территории.

После этого таблица bldgs пространственно присоединяется к себе с помощью предиката equal, при этом функция возвращает значение 1 (Oracle и SQLite) либо t (PostgreSQL) при обнаружении двух равных мультиполигонов. Условие s1.id<>s2.id устраняет сравнение геометрии с собой.

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)
```

```
);

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
);
```

```
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
```

ST_Equalsrs

Примечание:
Только PostgreSQL

Описание

ST_Equalsrs проверяет, идентичны ли две системы пространственной привязки двух разных классов пространственных объектов. Если системы пространственной привязки идентичны, возвращается t (истина). Если системы пространственной привязки не идентичны, ST_Equalsrs возвращает f (ложь).

Синтаксис

```
sde.st_equalsrs (srid1 integer, srid2 integer)
```

Тип возвращаемого значения

Boolean

Пример

В этом примере обнаруживаются ID пространственной привязки (SRID) различных классов пространственных объектов, затем используется ST_Equalsrs, чтобы увидеть, представляют ли SRID одну и ту же систему пространственной привязки.

```
SELECT srid, table_name
FROM sde_layers
WHERE table_name = 'transmains' OR table_name = 'streets';
```

srid	table_name
2	streets
6	transmains

Результаты запроса sde_layers

Теперь используйте ST_Equalsrs, чтобы определить, совпадают ли системы пространственной привязки, идентифицированные этими двумя SRID.

```
SELECT sde.st_equalsrs(2,6) ;

 st_equalsrs

-------------

f

(1 row)
```

ST_ExteriorRing

Определение

ST_ExteriorRing возвращает внешнее кольцо полигона в виде строки linestring.

Синтаксис

```
sde.st_exteriorring (polygon1 sde.st_geometry)
```

Oracle и PostgreSQL

```
sde.st_exteriorring (polygon1 sde.st_geometry)
```

SQLite

```
st_exteriorring (polygon1 geometryblob)
```

Тип возврата

ST_LineString

Пример:

Орнитолог, которая хочет изучить популяцию птиц на нескольких островах, знает, что зона питания интересующего ее вида птиц ограничена прибрежной полосой. При вычислении совокупной емкости популяции островов орнитолог должна знать периметр островов. Некоторые острова такие большие, что содержат несколько озер. Однако береговая линия озер населена только другим, более агрессивным видом птиц. Поэтому орнитолог должна знать периметр только внешнего кольца островов.

Столбцы ID и name таблицы islands определяют каждый остров, а в столбце land polygon хранится геометрия острова.

Функция ST_ExteriorRing извлекает внешнее кольцо из каждого полигона острова в виде строки linestring. Функция ST_Length вычисляет длину строки linestring. Длины строк linestring суммируются функцией SUM.

Внешние кольца островов представляют экологическую зону каждого острова, общую с морем.

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
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)
);

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
```

ST_GeomCollection

Примечание:
Только Oracle и PostgreSQL

Описание

ST_GeomCollection создает коллекцию геометрии из формата WKT.

Синтаксис

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)
```

Тип возвращаемого значения

ST_GeomCollection

Пример

Oracle

Создайте таблицу geomcoll_test и вставьте в нее геометрию.

```
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)
);
```

Выберите коллекцию геометрии из таблицы 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

Создайте таблицу geomcoll_test и вставьте в нее геометрию.

```
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)
);
```

Выберите коллекцию геометрии из таблицы 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))
```

```
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)))
```

ST_GeomCollFromWKB

Примечание:
Только PostgreSQL

Определение

ST_GeomCollFromWKB создает совокупность геометрии из стандартного двоичного представления.

Синтаксис

```
sde.st_geomcollfromwkb (wkb bytea, srid integer)
```

Возвращаемый тип

ST_GeomCollection

Пример:

Примечание:
Твердые переносы вставлены для удобочитаемости. При копировании выражений удалите их.

Создайте таблицу, gcoll_test.

```
CREATE TABLE gcoll_test (pkey integer, shape sde.st_geomcollection );
```

Вставьте в таблицу значения.

```
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));
```

Выберите геометрию из таблицы gcoll_test.

```
SELECT pkey, sde.st_astext(shape) from gcoll_test;

 pkey   st_astext
```

```
   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))
```

ST_Geometry

Описание

ST_Geometry создает коллекцию геометрии из формата WKT.

Примечание:
При создании пространственных таблиц, которые будут использоваться в ArcGIS, лучше всего создать столбец как супертип геометрии (например, ST_Geometry), а не указывать подтип ST_Geometry.

Синтаксис

Oracle

- Для линий, полигонов и точек

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

- Для оптимизированных точек (которые не запускают extproc-агент и, следовательно, быстрее обрабатывают запрос)

```
sde.st_geometry (x, y, z, m, srid)
```

  Используйте оптимизированное построение точек при пакетной вставке большого количества точечных данных.

- Для параметрических окружностей

```
sde.st_geometry (x, y, z, m, radius, number_of_points, srid)
```

- Для параметрических эллипсов

```
sde.st_geometry (x, y, z, m, semi_major_axis, semi_minor_axis, angle,
number_of_points, srid)
```

- Для параметрических секторов

```
sde.st_geometry (x, y, z, m, startangle, endangle, outerradius, innerradius,
number_of_points, srid)
```

PostgreSQL

- Для линий, полигонов и точек

```
sde.st_geometry (wkt, srid integer)
sde.st_geometry (esri_shape bytea, srid integer)
```

- Для параметрических окружностей

```
sde.st_geometry (x, y, z, m, radius, number_of_points, srid)
```

- Для параметрических эллипсов

```
sde.st_geometry (x, y, z, m, semi_major_axis, semi_minor_axis, angle,
number_of_points, srid)
```

- Для параметрических секторов

```
sde.st_geometry (x, y, z, m, startangle, endangle, outerradius, innerradius,
number_of_points, srid)
```

SQLite

- Для линий, полигонов и точек

```
st_geometry (text  WKT_string,int32 srid)
```

- Для параметрических окружностей

```
st_geometry (x, y, z, m, radius, number_of_points, srid)
```

- Для параметрических эллипсов

```
st_geometry (x, y, z, m, semi_major_axis, semi_minor_axis, angle_of_rotation,
number_of_points, srid)
```

- Для параметрических секторов

```
st_geometry (x, y, z, m, start_angle, end_angle, outer_radius, inner_radius,
number_of_points, srid)
```

Тип возвращаемого значения

Oracle и PostgreSQL

ST_Geometry

SQLite

Geometryblob

Примеры

Создание и запрос точечных, линейных и полигональных объектов

В этих примерах создается таблица (geoms) и вставляются в нее значения точек, линий и полигонов.

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 (
```

```
 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)
);
```

Создание и запрос параметрических окружностей

Создайте таблицу, radii, и вставьте в нее окружности.

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)
);
```

```
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)
);
```

Создание и запрос параметрических эллипсов

Создайте таблицу, треки и вставьте в нее эллипсы.

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)
);
```

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)
);
```

Создание и запрос параметрических секторов

Создайте таблицу, pwedge и вставьте в нее сектор.

Oracle

```
CREATE TABLE pwedge (
 id integer,
 label varchar2(8),
 shape sde.st_geometry
);
```

```
INSERT INTO PWEDGE (id, label, shape) VALUES (
```

```
 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)
);
```

ST_GeometryN

Определение

Функция ST_GeometryN принимает совокупность и цельночисленный индекс и возвращает n-й объект ST_Geometry в совокупности.

Синтаксис

Oracle и 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)
```

Тип возврата

Oracle и PostgreSQL

ST_Geometry

SQLite

Geometryblob

Пример:

В этом примере создается мультиполигон. Затем используется функция ST_GeometryN для получения второго элемента мультиполигона.

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
```

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))
```

ST_GeometryType

Описание

ST_GeometryType принимает объект геометрии и возвращает его тип геометрии (например, точку, линию, полигон, мультиточку) в виде строки.

Синтаксис

Oracle и PostgreSQL

```
sde.st_geometrytype (g1 sde.st_geometry)
```

SQLite

```
st_geometrytype (g1 geometryblob)
```

Тип возвращаемого значения

Varchar(32) (Oracle и PostgreSQL) или текст (SQLite), содержащий одно из следующего:

- ST_Point
- ST_LineString
- ST_Polygon
- ST_MultiPoint
- ST_MultiLineString
- ST_MultiPolygon

Пример

Таблица geometrytype_test содержит столбец геометрии g1.

Выражения INSERT вставляют каждый подкласс геометрии в столбец g1.

Запрос SELECT перечисляет тип геометрии каждого подкласса, хранящегося в столбце геометрии 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)
```

```
);

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)
```

```
);
```

```
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
```

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
```

ST_GeomFromCollection

Примечание:
Только PostgreSQL

Определение

ST_GeomFromCollection возвращает набор строк st_geometry. В каждой строке находится геометрия и целочисленное значение. Значение соответствует положению геометрии в наборе.

Используйте функцию ST_GeomFromCollection для доступа к отдельным геометрическим формам в составной геометрии. Когда входная геометрия является группой или составной геометрией (например, ST_MultiLineString, ST_MultiPoint, ST_MultiPolygon), ST_GeomFromCollection возвращает строку для каждого компонента группы, а path указывает положение компонента в группе.

Если вы используете ST_GeomFromCollection с простой геометрией (например, ST_Point, ST_LineString, ST_Polygon), возвращается одна запись с пустым значением path, поскольку геометрия только одна.

Синтаксис

```
sde.st_geomfromcollection (shape sde.st_geometry)
```

Чтобы вернуть только геометрию, используйте `(sde.st_geomfromcollection (shape)).st_geo`.

Чтобы вернуть только положение геометрии, используйте `(sde.st_geomfromcollection (shape)).path[1]`.

Тип возвращаемого значения

ST_Geometry set

Пример:

In this example, create a multiline feature class (ghanasharktracks) containing a single feature with a four-part shape.

```
--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
  )
 );
```

To confirm the field contains data, query the table. Use ST_AsText directly on the shape field to see the shape coordinates as text. Note that the text description of the multilinestring is returned.

```
--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
```

```
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))"
```

To return each linestring geometry individually, use the ST_GeomFromCollection function. To see the geometry as text, this example uses the ST_AsText function with the ST_GeomFromCollection function.

```
--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
```

ST_GeomFromText

Примечание:
Используется только в Oracle и SQLite; для PostgreSQL используйте ST_Geometry.

Описание

ST_GeomFromText берет представление точечного типа и идентификатор пространственной привязки и возвращает объект геометрии.

Синтаксис

Oracle

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

```
sde.st_geomfromtext (wkt clob)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

SQLite

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

```
st_geomfromtext (wkt text)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

Тип возвращаемого значения

Oracle

ST_Geometry

SQLite

Geometryblob

Пример

В таблице geometry_test хранится целочисленный столбец gid, который уникально определяет каждую строку, и столбец g1, в котором хранится геометрия.

Инструкция INSERT вставляет данные в столбцы gid и g1 таблицы geometry_test. Функция ST_GeomFromText преобразует каждое текстовое представление геометрии в соответствующий подкласс. Выполняется инструкция SELECT, чтобы убедиться, что в столбец g1 были вставлены данные.

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)))
```

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)))
```

ST_GeomFromWKB

Определение

ST_GeomFromWKB берет стандартное бинарное представление (WKB) и ID пространственной привязки, чтобы возвратить объект геометрии.

Синтаксис

Oracle

```
sde.st_geomfromwkb (wkb blob, srid integer)
```

```
sde.st_geomfromwkb (wkb blob)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 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)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

Тип возвращаемого значения

Oracle и PostgreSQL

ST_Geometry

SQLite

Geometryblob

Пример

В следующем примере строки результатов были отформатированы для улучшения восприятия. Отступы в результатах будут зависеть от отображения на экране. В следующем коде показано, как функция

ST_GeomFromWKB используется для создания и вставки линии на основе ее WKB-представления. В следующем примере в таблицу sample_lines вставляется строка с ID и геометрией, имеющей пространственную привязку к системе 4326 в WKB-представлении.

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)
```

```
);

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))
```

ST_GeoSize

Примечание:
Только PostgreSQL

Определение

ST_GeoSize получает объект ST_Geometry и возвращает его размер в байтах.

Синтаксис

```
st_geosize (st_geometry)
```

Тип возврата

Целое

Пример

Вы можете ознакомиться с размерами объектов, созданных в примере ST_GeomFromWKB, выполнив запрос к столбцу геометрии в таблице sample_geometries.

```
SELECT st_geosize (geometry)
FROM sample_geometries;

st_geosize
        512
        592
        616
```

ST_InteriorRingN

Определение

ST_InteriorRingN возвращает n-е внутреннее кольцо полигона в виде строки ST_LineString.

Порядок колец не может быть задан заранее, так как они организуются в соответствии с правилами, определенными внутренними процедурами проверки геометрии, а не геометрической ориентацией. Если индекс превышает число внутренних колец полигона, возвращается ноль.

Синтаксис

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)
```

Тип возврата

ST_LineString

Пример:

Создайте таблицу sample_polys и добавьте запись, а затем выберите ID и геометрию внутреннего кольца.

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
```

```
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)
```

ST_Intersection

Определение

Функция ST_Intersection использует два объекта геометрии и возвращает набор пересечений в виде двухмерного объекта геометрии.

Синтаксис

Oracle и PostgreSQL

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

SQLite

```
st_intersection (geometry1 geometryblob, geometry2 geometryblob)
```

Тип возврата

Oracle и PostgreSQL

ST_Geometry

SQLite

Geometryblob

Пример:

Начальник пожарной команды должен получить площади больниц, школ и домов престарелых с пересечением области возможного заражения токсичными отходами.

Больницы, школы и дома престарелых хранятся в таблице population, созданной следующей инструкцией CREATE TABLE. В столбце shape, определенном как полигон, содержится контур каждой важной области.

Опасные для жизни участки хранятся в таблице waste_sites, которая была создана с помощью следующего выражения CREATE TABLE. Столбец site, определенный как точка, содержит расположение, являющееся географическим центром каждого опасного участка.

Функция ST_Buffer создает буфер, окружающий вредоносные участки. Функция ST_Intersection создает полигоны на основе пересечений буферизированных опасных участков и важных областей.

Oracle

```
CREATE TABLE population (
 id integer,
 shape sde.st_geometry
);

CREATE TABLE waste_sites (
 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))
```

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
```

```
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
```

```
 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))
```

ST_Intersects

Определение

ST_Intersects возвращает 1 (Oracle и SQLite) либо t (PostgreSQL), если пересечение двух геометрий не образует пустого множества. В противном случае возвращается значение 0 (Oracle и SQLite) или f (PostgreSQL).

Синтаксис

Oracle и PostgreSQL

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

SQLite

```
st_intersects (geometry1 geometryblob, geometry2 geometryblob)
```

Тип возврата

Логический

Пример:

Начальник пожарной команды хочет получить список важных областей с радиусом участков хранения токсичных отходов.

Важные области сохраняются в таблице sensitive_areas. В столбце shape, определенном как полигон, содержится контур каждой важной области.

Опасные объекты хранятся в таблице hazardous_sites. Столбец site, определенный как точка, содержит расположение, являющееся географическим центром каждого опасного участка.

Запрос SELECT создает буферную зону вокруг каждого опасного объекта и возвращает список важных областей, которые пересекают эти буферные зоны.

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)
);

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)
);

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)
);

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
```

ST_Is3d

Описание

ST_Is3d получает ST_Geometry в качестве входного параметра и возвращает 1 (Oracle и SQLite) или t (PostgreSQL), если данная геометрия имеет z-координаты; в противном случае возвращает 0 (Oracle и SQLite) или f (PostgreSQL).

Синтаксис

Oracle и PostgreSQL

```
sde.st_is3d (geometry1 sde.st_geometry)
```

SQLite

```
st_is3d (geometry1 geometryblob)
```

Тип возвращаемого значения

Boolean

Пример

В этом примере вы создаете таблицу is3d_test и заполняете её записями.

Затем с помощью ST_Is3d проверьте, содержит ли какая-либо из записей z-координату.

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)
);
```

```
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
```

```
1902          0
1903          0
1904          1
1905          1
```

ST_IsClosed

Определение

ST_IsClosed берет объект ST_LineString или ST_MultiLineString и возвращает 1 (Oracle и SQLite) либо t (PostgreSQL), если запись закрыта. В противном случае возвращается 0 (Oracle и SQLite) либо f (PostgreSQL).

Синтаксис

Oracle и PostgreSQL

```
sde.st_isclosed (line1 sde.st_geometry)
sde.st_isclosed (multiline1 sde.st_geometry)
```

SQLite

```
st_isclosed (geometry1 geometryblob)
```

Тип возврата

Логический

Примеры

Тестирование строки linestring

Таблица closed_linestring создается с одним столбцом типа linestring.

Инструкция INSERT вставляет две записи в таблицу closed_linestring. Первая запись не является закрытой строкой linestring, а вторая является.

Запрос возвращает результаты функции ST_IsClosed. Первая строка возвращает 0 или f, так как linestring не закрыта, а вторая строка возвращает 1 или t, так как строка linestring закрыта.

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;
```

```
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
```

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
```

Тестирование строки multilinestring

Таблица closed_mlinestring создается с одним столбцом ST_MultiLineString.

Инструкция INSERT вставляет закрытую и незакрытую запись ST_MultiLineString.

Запрос возвращает результаты функции ST_IsClosed. Первая строка возвращает 0 или f, так как строка multilinestring не закрыта. Вторая строка возвращает 1 или t, так как строка multilinestring в столбце ln1 закрыта. Строка multilinestring закрыта, если все элементы linestring закрыты.

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)
);

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
```

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
```

ST_IsEmpty

Определение

ST_IsEmpty возвращает значение 1 (Oracle и SQLite) или t (PostgreSQL), если объект ST_Geometry пустой. В противном случае возвращается значение 0 (Oracle и SQLite) или f (PostgreSQL).

Синтаксис

Oracle и PostgreSQL

```
sde.st_isempty (geometry1 sde.st_geometry)
```

SQLite

```
st_isempty (geometry1 geometryblob)
```

Тип возврата

Логический

Пример:

Инструкция CREATE TABLE ниже создает таблицу empty_test со столбцом geotype, в котором хранится тип данных подклассов, хранимых в столбце g1.

Инструкция INSERT вставляет две записи для подклассов геометрии point, linestring и polygon: одну пустую и другую непустую.

Запрос SELECT возвращает тип геометрии из столбца geotype и результаты функции 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',
```

```
 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)
);
```

```
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)
);
```

```
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
```

ST_IsMeasured

Описание

ST_IsMeasured получает объект геометрии в качестве входного параметра и возвращает 1 (Oracle и SQLite) или t (PostgreSQL), если данная геометрия содержит измерения, в противном случае возвращается 0 (Oracle и SQLite) или f (PostgreSQL).

Синтаксис

Oracle и PostgreSQL

```
sde.st_ismeasured (geometry1 sde.st_geometry)
```

SQLite

```
st_ismeasured (geometry1 geometryblob)
```

Тип возвращаемого значения

Boolean

Пример

Создает таблицу ism_test, вставляет в неё значения, затем определяет, какие строки в таблице ism_test содержат измерения.

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)
);
```

```
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
```

```
19          0
20          1
21          0
22          1
```

ST_IsRing

Определение

ST_IsRing берет объект ST_LineString и возвращает 1 (Oracle и SQLite) либо t (PostgreSQL), если это кольцо (например, ST_LineString закрыта и является простой). В противном случае возвращается 0 (Oracle и SQLite) или f (PostgreSQL).

Синтаксис

Oracle и PostgreSQL

```
sde.st_isring (line1 sde.st_geometry)
```

SQLite

```
st_isring (line1 geometryblob)
```

Тип возврата

Логический

Пример:

Таблица ring_linestring создается с одним столбцом типа ST_LineString, ln1.

Инструкция INSERT вставляет три строки linestring в столбец ln1. Первая строка содержит строку linestring, которая не является закрытой и не является кольцом. Вторая строка содержит закрытую простую строку linestring, которая является кольцом. Третья строка содержит закрытую, но не простую строку linestring, так как она пересекается с собственными внутренними точками. Она также не является кольцом.

Запрос SELECT возвращает результаты выполнения функции ST_IsRing. Первая строка возвращает 0 или f, так как строки linestring не являются кольцами, а вторая и третья строки возвращают 1 или t, так как они являются кольцевыми.

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
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
```

ST_IsSimple

Определение

ST_IsSimple возвращает 1 (Oracle и SQLite) или t (PostgreSQL), если объект геометрии является простым и определяется Open Geospatial Consortium (OGC). В противном случае возвращается 0 (Oracle и SQLite) или f (PostgreSQL).

Синтаксис

Oracle и PostgreSQL

```
sde.st_issimple (geometry1 sde.st_geometry)
```

SQLite

```
st_issimple (geometry1 geometryblob)
```

Тип возврата

Логический

Пример:

Создается таблица issimple_test с двумя столбцами. Тип данных столбца pid – smallint, столбец содержит уникальный идентификатор для каждой строки. В столбце g1 хранятся простой и непростой образцы геометрии.

Инструкция INSERT вставляет две записи в таблицу issimple_test. Первая строка является простой, так как она не пересекается с собственными внутренними точками. Вторая строка является непростой по определению OGC, так как она пересекается с собственными внутренними точками.

Запрос возвращает результаты функции ST_IsSimple. Первая запись возвращает 1 или t, так как строка linestring является простой, а вторая запись возвращает 0 или f, поскольку строка не является простой.

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
```

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
```

ST_Length

Определение

ST_Length возвращает длину одинарной или множественной строки.

Синтаксис

Oracle и 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)
```

Для получения списка поддерживаемых наименований единиц измерения обратитесь к разделу ST_Distance.

Тип возврата

Двойная точность

Пример:

Местный эколог, изучающий особенности миграции популяции лосося в водоемах страны, хочет получить длину всех ручьев и речных систем в стране.

Создается таблица waterways со столбцами ID и name, которые определяют все ручьи и реки, хранимые в таблице. Столбец water имеет тип multilinestring, так как ручьи и реки часто являются объединениями нескольких элементов типа linestring.

Запрос SELECT возвращает имя каждой системы и длину систему, полученную с использованием функции ST_Length. В Oracle и PostgreSQL единицы измерения соответствуют используемой вами координатной системе. В SQLite указаны километры.

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)
```

```
);
```

```
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',
```

```
  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
```

ST_LineFromText

Примечание:
Поддерживается только в Oracle и SQLite; для PostgreSQL используйте ST_LineString.

Определение

ST_LineFromText принимает WKT-представление типа ST_LineString и идентификатор пространственной привязки и возвращает ST_LineString.

Синтаксис

Oracle

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

```
sde.st_linefromtext (wkt clob)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

SQLite

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

```
st_linefromtext (wkt text)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

Тип возвращаемого значения

ST_LineString

Пример

Таблица linestring_test создается с одним столбцом ln1 типа ST_LineString.

Инструкция SELECT вставляет ST_LineString в столбец ln1 с помощью функции 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)
);
```

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)
);
```

ST_LineFromWKB

Определение

ST_LineFromWKB принимает WKB-представление типа ST_LineString и идентификатор пространственной привязки и возвращает ST_LineString.

Синтаксис

Oracle

```
sde.st_linefromwkb (wkb blob, srid integer)
```

```
sde.st_linefromwkb (wkb blob)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

PostgreSQL

```
sde.st_linefromwkb (wkb bytea, srid integer)
```

SQLite

```
st_linefromwkb (wkb blob, srid int32)
```

```
st_linefromwkb (wkb blob)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

Тип возвращаемого значения

ST_LineString

Пример

Следующие команды создают таблицу (sample_lines) и используют функцию ST_LineFromWKB для вставки линий из WKB-представления. Строка вставляется в таблицу sample_lines с ИД и линией в системе пространственной привязки 4326 в WKB-представлении.

Oracle

```
CREATE TABLE sample_lines (
 id smallint,
 geometry sde.st_linestring,
 wkb blob
);
INSERT INTO SAMPLE_LINES (id, geometry) VALUES (
```

```
 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',
```

```
 '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)
```

ST_LineString

Описание

ST_LineString - это функция доступа, которая создает строку linestring из стандартного текстового представления WKT.

Примечание:
При создании пространственных таблиц, которые будут использоваться в ArcGIS, лучше всего создать столбец как супертип геометрии (например, ST_Geometry), а не указывать подтип ST_Geometry.

Синтаксис

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)
```

Тип возвращаемого значения

ST_LineString

Пример

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)
```

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)
```

ST_M

Определение

ST_M принимает ST_Point как входной параметр и возвращает координату измерения (m).

В SQLite ST_M может также использоваться для обновления значения измерения.

Синтаксис

Oracle и PostgreSQL

```
sde.st_m (point1 sde.st_point)
```

SQLite

```
st_m (point1 geometryblob)
st_m (point1 geometryblob, new_Mvalue double)
```

Тип возврата

Oracle и PostgreSQL

Число (Number)

SQLite

Для запроса значения измерения требуется двойная точность; geometryblob – при обновлении значения измерения.

Примеры

Oracle

Создается таблица m_test, в которую вставляются три точки. Все три содержат значения измерений. Выражение SELECT запускается с функцией ST_M для возврата значения измерения для каждой точки.

```
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)
```

```
);

SELECT id, sde.st_m (geometry) M_COORD
 FROM M_TEST;

        ID    M_COORD

         1          5
         2          4
         3          7
```

PostgreSQL

Создается таблица m_test, в которую вставляются три точки. Все три содержат значения измерений. Выражение SELECT запускается с функцией ST_M для возврата значения измерения для каждой точки.

```
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

В первом примере создается таблица m_test, в которую вставляются три точки. Все три содержат значения измерений. Выражение SELECT запускается с функцией ST_M для возврата значения измерения для каждой точки.

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

SELECT AddGeometryColumn (
 NULL,
 'm_test',
 'geometry',
 4326,
```

```
 'pointzm',
 '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
```

Во втором примере значение измерения обновляется для записи 3 таблицы m_test.

```
SELECT st_m (geometry, 7.5)
 FROM m_test
 WHERE id = 3;
```

ST_MaxM

Определение

Функция ST_MaxM принимает геометрию в качестве входного параметра и возвращает максимальную координату m.

Синтаксис

Oracle и PostgreSQL

```
sde.st_maxm (geometry1 sde.st_geometry)
```

SQLite

```
st_maxm (geometry1 geometryblob)
```

Тип возврата

Oracle и PostgreSQL

Число (Number)

Если значения m отсутствуют, возвращается NULL.

SQLite

Двойная точность

Если значения m отсутствуют, возвращается NULL.

Пример:

Создается таблица maxm_test, в которую вставляются два полигона. Затем запускается ST_MaxM для определения максимального в каждом полигоне значения m.

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)
);
```

```
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 (
 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
```

ST_MaxX

Определение

Функция ST_MaxX принимает геометрию в качестве входного параметра и возвращает максимальную координату x.

Синтаксис

Oracle и PostgreSQL

```
sde.st_maxx (geometry1 sde.st_geometry)
```

SQLite

```
st_maxx (geometry1 geometryblob)
```

Тип возврата

Oracle и PostgreSQL

Число (Number)

SQLite

Двойная точность

Пример:

Создается таблица maxx_test, в которую вставляются два полигона. Затем запускается функция ST_MaxX для определения максимального в каждом полигоне значения x.

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

      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)
);
```

```
SELECT id, st_maxx (geometry)
 AS "max_x"
 FROM maxx_test;

id      max_x

1901    120.0
1902    5.00000000
```

ST_MaxY

Определение

Функция ST_MaxY принимает геометрию в качестве входного параметра и возвращает максимальную координату y.

Синтаксис

Oracle и PostgreSQL

```
sde.st_maxy (geometry1 sde.st_geometry)
```

SQLite

```
st_maxy (geometry1 geometryblob)
```

Тип возврата

Oracle и PostgreSQL

Число (Number)

SQLite

Двойная точность

Пример:

Создается таблица maxy_test, в которую вставляются два полигона. Затем запускается функция ST_MaxY для определения максимального в каждом полигоне значения y.

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

      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)
);
```

```
SELECT id, st_maxy (geometry)
 AS "max_y"
 FROM maxy_test;

id      max_y

1901    140.0
1902    4.00000000
```

ST_MaxZ

Определение

Функция ST_MaxZ принимает геометрию в качестве входного параметра и возвращает максимальную координату z.

Синтаксис

Oracle и PostgreSQL

```
sde.st_maxz (geometry1 sde.st_geometry)
```

SQLite

```
st_maxz (geometry1 geometryblob)
```

Тип возврата

Oracle и PostgreSQL

Число (Number)

Если значения z отсутствуют, возвращается NULL.

SQLite

Двойная точность

Если значения z отсутствуют, возвращается NULL.

Пример:

В следующем примере создается таблица maxz_test, в которую вставляются два полигона. Затем запускается ST_MaxZ для возврата максимального в каждом полигоне значения z.

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)
);
```

```
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 (
 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
```

ST_MinM

Определение

Функция ST_MinM принимает геометрию в качестве входного параметра и возвращает минимальную координату m.

Синтаксис

Oracle и PostgreSQL

```
sde.st_minm (geometry1 sde.st_geometry)
```

SQLite

```
st_minm (geometry1 geometryblob)
```

Тип возврата

Oracle и PostgreSQL

Число (Number)

Если значения m отсутствуют, возвращается NULL.

SQLite

Двойной точности (Double precision)

Если значения m отсутствуют, возвращается NULL.

Пример:

Создается таблица minm_test, в которую вставляются два полигона. Затем запускается ST_MinM для определения минимального в каждом полигоне значения измерения.

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)
);

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)
```

```
);

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
```

ST_MinX

Определение

Функция ST_MinX принимает геометрию в качестве входного параметра и возвращает минимальную координату x.

Синтаксис

Oracle и PostgreSQL

```
sde.st_minx (geometry1 sde.st_geometry)
```

SQLite

```
st_minx (geometry1 geometryblob)
```

Тип возврата

Oracle и PostgreSQL

Число (Number)

SQLite

Двойной точности (Double precision)

Пример:

Создается таблица minx_test, в которую вставляются два полигона. Затем запускается ST_MinX для определения минимального в каждом полигоне значения координаты x.

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

      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)
);
```

```
SELECT id AS "ID", st_minx (geometry) AS "MinX"
 FROM minx_test;

ID       MinX

1914     110.0
1915       0.0
```

ST_MinY

Определение

Функция ST_MinY принимает геометрию в качестве входного параметра и возвращает минимальную координату y.

Синтаксис

Oracle и PostgreSQL

```
sde.st_miny (geometry1 sde.st_geometry)
```

SQLite

```
st_miny (geometry1 geometryblob)
```

Тип возврата

Oracle и PostgreSQL

Число (Number)

SQLite

Двойная точность

Пример:

Создается таблица miny_test, в которую вставляются два полигона. Затем запускается ST_MinY для определения минимального в каждом полигоне значения координаты y.

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;

        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)
);

SELECT id, st_miny (geometry)
 AS "MinY"
 FROM miny_test;

id       MinY

101      120.0
102        0.0
```

ST_MinZ

Определение

Функция ST_MinZ принимает геометрию в качестве входного параметра и возвращает минимальную координату z.

Синтаксис

Oracle и PostgreSQL

```
sde.st_minz (geometry1 sde.st_geometry)
```

SQLite

```
st_minz (geometry1 geometryblob)
```

Тип возврата

Oracle и PostgreSQL

Число (Number)

Если значения z отсутствуют, возвращается NULL.

SQLite

Двойная точность

Если значения z отсутствуют, возвращается NULL.

Пример:

Создается таблица minz_test, в которую вставляются два полигона. Затем запускается ST_MinZ для определения минимального в каждом полигоне значения координаты z.

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)
);
```

```
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,
 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
```

ST_MLineFromText

Примечание:
Используется только в Oracle и SQLite; для PostgreSQL используйте ST_MultiLineString.

Определение

ST_MLineFromText принимает WKT-представление типа ST_MultiLineString и идентификатор пространственной привязки и возвращает ST_MultiLineString.

Синтаксис

Oracle

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

```
sde.st_mlinefromtext (wkt clob)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

SQLite

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

```
st_mlinefromtext (wkt text)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

Тип возвращаемого значения

ST_MultiLineString

Пример

Таблица mlinestring_test создается со столбцом gid типа smallint, который уникально определяет строку, и столбцом ml1 ST_MultiLineString.

Инструкция SELECT вставляет ST_MultiLineString с помощью функции ST_MLineFromText.

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)
);
```

ST_MLineFromWKB

Определение

ST_MLineFromWKB принимает WKB-представление типа ST_MultiLineString и идентификатор пространственной привязки и создает ST_MultiLineString.

Синтаксис

Oracle

```
sde.st_mlinefromwkb (wkb blob, srid integer)
```

```
sde.st_mlinefromwkb (wkb blob)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

PostgreSQL

```
sde.st_mlinefromwkb (wkb bytea, srid integer)
```

SQLite

```
st_mlinefromwkb (wkb blob, srid int32)
```

```
st_mlinefromwkb (wkb blob)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

Тип возвращаемого значения

ST_MultiLineString

Пример

В этом примере показано, как можно использовать функцию ST_MLineFromWKB для создания строки multilinestring на основе его WKB-представления. В системе пространственной привязки 4326 геометрия представлена строкой multilinestring . В этом примере multilinestring сохраняется с идентификатором ID = 10 в столбце geometry таблицы sample_mlines, затем столбец wkb обновляется с использованием WKB-представления (с помощью функции ST_AsBinary). Наконец, функция ST_MLineFromWKB используется для возврата строки multilinestring из столбца wkb. В таблице sample_mlines есть столбец geometry, в которой сохраняется multilinestring, а также столбец wkb, в котором хранится WKB-представление строки multilinestring.

В следующем выражении SELECT функция ST_MLineFromWKB используется для получения строки multilinestring из столбца wkb.

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,
```

```
 '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 ))
```

ST_MPointFromText

Примечание:
Только Oracle и SQLite; для PostgreSQL используют ST_MultiPoint.

Описание

ST_MPointFromText принимает WKT-представление типа ST_MultiPoint и идентификатор пространственной привязки и создает ST_MultiPoint.

Синтаксис

Oracle

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

```
sde.st_mpointfromtext (wkt clob)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

SQLite

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

```
st_mpointfromtext (wkt text)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

Тип возвращаемого значения

ST_MultiPoint

Пример

Таблица multipoint_test создается с одним столбцом mpt1 типа ST_MultiPoint.

Инструкция SELECT вставляет мультиточку в столбец mpt1 с помощью функции 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));
```

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));
```

ST_MPointFromWKB

Описание

ST_MPointFromText принимает WKB-представление типа ST_MultiPoint и идентификатор пространственной привязки и создает ST_MultiPoint.

Синтаксис

Oracle

```
sde.st_mpointfromwkb (wkb blob, srid integer)
```

```
sde.st_mpointfromwkb (wkb blob)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

PostgreSQL

```
sde.st_mpointfromwkb (wkb bytea, srid integer)
```

SQLite

```
st_mpointfromwkb (wkb blob, srid int32)
```

```
st_mpointfromwkb (wkb blob)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

Тип возвращаемого значения

ST_MultiPoint

Пример

В этом примере показано, как можно использовать функцию ST_MPointFromWKB для создания мультиточки на основе его WKB-представления. В системе пространственной привязки 4326 геометрия представлена строкой multipoint. В этом примере multipoint сохраняется с идентификатором ID = 10 в столбце GEOMETRY таблицы SAMPLE_MPOINTS, затем столбец WKB обновляется с использованием бинарного WKB-представления (с помощью функции ST_AsBinary). Наконец, функция ST_MPointFromWKB используется для возврата мультиточки из столбца wkb. В таблице SAMPLE_MPOINTS есть столбец GEOMETRY, в котором сохраняется мультиточка, а также столбец WKB, в котором хранится WKB-представление.

В следующем выражении SELECT функция ST_MPointFromWKB используется для получения мультиточки из столбца wkb.

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)
```

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))
```

ST_MPolyFromText

Примечание:
Только Oracle и SQLite; для PostgreSQL используют ST_MultiPolygon.

Описание

ST_MPointFromText принимает WKT-представление типа ST_MultiPolygon и идентификатор пространственной привязки и возвращает ST_MultiPolygon.

Синтаксис

Если вы не указали SRID, пространственная привязка по умолчанию будет 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)
```

Тип возвращаемого значения

ST_MultiPolygon

Пример

Таблица multipolygon_test создается со столбцом ST_MultiPolygon, mpl1.

Инструкция SELECT вставляет ST_MultiPolygon в столбец mpl1 с помощью функции 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)
);
```

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)
);
```

ST_MPolyFromWKB

Определение

ST_MPointFromWKB принимает WKB-представление типа ST_MultiPolygon и идентификатор пространственной привязки и возвращает ST_MultiPolygon.

Синтаксис

Oracle

```
sde.st_mpolyfromwkb (wkb blob, srid integer)
```

```
sde.st_mpolyfromwkb (wkb blob)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

PostgreSQL

```
sde.st_mpolyfromwkb (wkb bytea, srid integer)
```

SQLite

```
st_mpolyfromwkb (wkb blob, srid int32)
```

```
st_mpolyfromwkb (wkb blob)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

Тип возвращаемого значения

ST_MultiPolygon

Пример

В этом примере показано, как можно использовать функцию ST_MPolyFromWKB для создания мультиполигона на основе его WKB-представления. В системе пространственной привязки 4326 геометрия представлена строкой multipolygon. В этом примере multipolygon сохраняется с идентификатором ID = 10 в столбце geometry таблицы sample_mpolys, затем столбец wkb обновляется с использованием wkb-представления (с помощью функции ST_AsBinary). Наконец, функция ST_MPolyFromWKB используется для возврата мультиполигона из столбца wkb. В таблице sample_mpolys есть столбец geometry, в которой сохраняется мультиполигон, а также столбец wkb, в котором хранится WKB-представление мультиполигона.

В следующем выражении SELECT функция ST_MPolyFromWKB используется для получения мультиполигона из столбца wkb.

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
);
```

```
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)))
```

ST_MultiCurve

Примечание:
Только Oracle

Описание

ST_MultiCurve строит объект мультикривой из стандартного текстового представления WKT.

Синтаксис

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

Тип возвращаемого значения

ST_MultiLinestring

Пример

```
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 ))
```

ST_MultiLineString

Описание

ST_MultiLineString строит мультилинию из стандартного текстового представления WKT.

Примечание:
При создании пространственных таблиц, которые будут использоваться в ArcGIS, лучше всего создать столбец как супертип геометрии (например, ST_Geometry), а не указывать подтип ST_Geometry.

Синтаксис

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)
```

Тип возвращаемого значения

ST_MultiLineString

Пример

Создается таблица mlines_test, и в нее вставляется одна мультилиния с помощью функции 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 (
```

```
 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)
);
```

ST_MultiPoint

Описание

ST_MultiPoint создает объект-мультиточку из формата WKT.

Примечание:
При создании пространственных таблиц, которые будут использоваться в ArcGIS, лучше всего создать столбец как супертип геометрии (например, ST_Geometry), а не указывать подтип ST_Geometry.

Синтаксис

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)
```

Тип возвращаемого значения

ST_MultiPoint

Пример

Создается таблица mpoint_test, и в нее вставляется одна мультиточка с помощью функции 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,
```

```
 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)
);
```

ST_MultiPolygon

Описание

ST_MultiPolygon создает объект мультиполигона из формата WKT.

Примечание:
При создании пространственных таблиц, которые будут использоваться в ArcGIS, лучше всего создать столбец как супертип геометрии (например, ST_Geometry), а не указывать подтип ST_Geometry.

Синтаксис

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)
```

Тип возвращаемого значения

ST_MultiPolygon

Пример

Создается таблица mpoly_test, и один мультиполигон вставляется в нее с помощью функции 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 (
```

```
 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)
);
```

ST_MultiSurface

Примечание:
Только Oracle

Описание

ST_MultiSurface строит многоповерхностный объект из стандартного текстового представления (WKT).

Синтаксис

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

Тип возвращаемого значения

ST_MultiSurface

Пример

```
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)))
```

ST_NumGeometries

Описание

ST_NumGeometries получает коллекцию геометрии и возвращает число геометрий в коллекции.

Синтаксис

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)
```

Тип возвращаемого значения

Целочисленные

Пример

В следующем примере создается таблица с именем sample_numgeom. В неё добавлены один мультиполигон и одна мультиточка В выражении SELECT используется функция ST_NumGeometries, чтобы определить число геометрий (или объектов) в каждой геометрии.

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;
```

```
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)
);
```

```
SELECT id, st_numgeometries (geometry)
 AS "number of geometries"
 FROM sample_numgeom;

id          number of geometries

  1                 3
  2                 5
```

ST_NumInteriorRing

Определение

ST_NumInteriorRing берет ST_Polygon и возвращает число внутренних колец.

Синтаксис

Oracle и PostgreSQL

```
sde.st_numinteriorring (polygon1 sde.st_geometry)
```

SQLite

```
st_numinteriorring (polygon1 geometryblob)
```

Тип возврата

Целочисленное (Integer)

Пример:

Орнитолог хочет изучить популяцию птиц на нескольких южных морских островах. Она хочет определить, на каких островах есть одно или несколько озер, так как интересующий ее вид птиц питается только в пресноводных озерах.

Столбцы ID и name таблицы islands определяют каждый остров, а в столбце land ST_Polygon хранится геометрия островов.

Так как внутренние кольца отображают озера, выражение SELECT, включающее функцию ST_NumInteriorRing, указывает только те острова, у которых есть по крайней мере одно внутреннее кольцо.

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
```

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
```

ST_NumPoints

Определение

ST_NumPoints возвращает число точек (вершин) в геометрии.

Для полигонов учитываются начальные и конечные вершины, даже если они занимают одну точку.

Учтите, что это число отличается от значения атрибута NUMPTS типа ST_Geometry. Атрибут NUMPTS содержит число вершин во всех частях геометрии, в том числе разделители частей. Между каждой частью есть один разделитель. Например, у строки linestring с тремя частями два разделителя. В атрибуте NUMPTS каждый разделитель считается одной вершиной. И наоборот, функция ST_NumPoints не включает разделители в число вершин.

Синтаксис

Oracle и PostgreSQL

```
sde.st_numpoints (geometry1 sde.st_geometry)
```

SQLite

```
st_numpoints (geometry1 geometryblob)
```

Тип возврата

Целочисленное (Integer)

Пример:

Таблица numpoints_test создается со столбцом geotype, который содержит тип геометрии, хранимый в столбце g1 geometry.

Инструкция INSERT вставляет точку, строку linestring и полигон.

Запрос SELECT использует функцию ST_NumPoints для определения числа точек в каждом объекте для каждого типа объектов.

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)
);
```

```
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)
);
```

```
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
```

ST_OrderingEquals

Примечание:
Только Oracle и PostgreSQL

Описание

ST_OrderingEquals сравнивает два объекта ST_Geometry и возвращает 1 (Oracle) или t (PostgreSQL), если геометрии идентичны; в противном случае возвращается значение 0 (Oracle) или f (PostgreSQL).

Синтаксис

```
sde.st_orderingequals (g1 sde.st_geometry, g2 sde.st_geometry)
```

Тип возвращаемого значения

Boolean

Пример

Oracle

Следующее выражение `CREATE TABLE` создает таблицу LINESTRING_TEST, которая имеет два столбца строк, ln1 и ln2.

```
CREATE TABLE linestring_test (
lid integer,
ln1 sde.st_geometry,
ln2 sde.st_geometry);
```

Следующее выражение `INSERT` вставляет два значения ST_LineString в ln1 и ln2, которые равны и имеют одинаковый порядок координат.

```
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)
);
```

Следующее выражение `SELECT` и соответствующий набор результатов показывают, как функция ST_Equals возвращает 1 (true) независимо от порядка координат. Функция ST_OrderingEquals возвращает значение 0 (false), если геометрии не равны и у них одинаковый порядок координат.

```
SELECT lid, sde.st_equals (ln1, ln2) Equals, sde.st_orderingequals (ln1, ln2)
OrderingEquals
FROM LINESTRING_TEST;

lid Equals     OrderingEquals

1   1          0
```

PostgreSQL

Следующее выражение `CREATE TABLE` создает таблицу LINESTRING_TEST, которая имеет два столбца строк, ln1 и ln2.

```
CREATE TABLE linestring_test (
lid integer,
ln1 sde.st_geometry,
ln2 sde.st_geometry);
```

Следующее выражение `INSERT` вставляет два значения ST_LineString в ln1 и ln2, которые равны и имеют одинаковый порядок координат.

```
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)
);
```

Следующее выражение `SELECT` и соответствующий набор результатов показывают, как функция ST_Equals возвращает t (true) независимо от порядка координат. Функция ST_OrderingEquals возвращает значение f (false), если геометрии не равны и у них одинаковый порядок координат.

```
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
```

ST_Overlaps

Определение

ST_Overlaps берет два объекта геометрии и возвращает значение 1 (Oracle и SQLite) либо t (PostgreSQL), если пересечение объектов приводит к получению объекта геометрии той же размерности, но не равного исходному объекту. В противном случае возвращается значение 0 (Oracle и SQLite) или f (PostgreSQL).

Синтаксис

Oracle и PostgreSQL

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

SQLite

```
st_overlaps (geometry1 geometryblob, geometry2 geometryblob)
```

Тип возврата

Логический

Пример:

Главе округа нужен список важных областей, которые пересекаются с буферным радиусом участков хранения токсичных отходов. Таблица sensitive_areas содержит несколько столбцов, описывающих учреждения под угрозой, в дополнение к столбцу shape, который хранит геометрии ST_Polygon.

В таблице hazardous_sites в столбце id хранятся идентификаторы участков, а фактическое географическое расположение каждого участка хранится в столбце point.

Таблицы sensitive_areas и hazardous_sites объединяются функцией ST_Overlaps, которая возвращает идентификатор всех строк sensitive_areas с полигонами, пересекающими буферный радиус точек 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)
);
```

```
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 (
```

```
 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)
);
```

```
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
```

ST_Perimeter

Определение

ST_Perimeter возвращает длину непрерывной линии, которая формирует границу замкнутого полигона или мультиполигонального объекта.

Это новая функция, появившееся в 10.7.1.

Синтаксис

Первые две опции в каждом разделе возвращают периметр в единицах системы координат, определенной для объекта. Вторые две опции позволяют задать линейные единицы измерения. Для получения списка поддерживаемых значений для `linear_unit_name`, обратитесь к разделу ST_Distance.

Oracle и 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)
```

Тип возвращаемого значения

Двойная точность

Примеры

Oracle

В следующем примере эколог, изучающий поголовье птиц гнездящихся вдоль береговой линии, хочет определить длину береговой линии озер в определенной местности. Озера представлены полигонами в таблице waterbodies. Выражение `SELECT`, использующее функцию ST_Perimeter, возвращает периметр каждого озера (объекта) в таблице 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
```

```
SELECT sde.ST_Perimeter (waterbody)
  FROM waterbodies;
```

Выражение SELECT возвращает следующее:

```
ID PERIMETER
1 +1.8000000
```

В следующем примере вы создадите таблицу с именем bfp, включающую три объекта, и вычислите периметр каждого объекта в линейных единицах измерения:

```
--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;
```

Выражение SELECT возвращает периметр каждого объекта в трех измерениях:

```
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

В следующем примере эколог, изучающий поголовье птиц гнездящихся вдоль береговой линии, хочет определить длину береговой линии озер в определенной местности. Озера представлены полигонами в таблице waterbodies. Выражение SELECT, использующее функцию ST_Perimeter, возвращает периметр каждого озера (объекта) в таблице 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
SELECT sde.ST_Perimeter (waterbody)
  FROM waterbodies;
```

Выражение SELECT возвращает следующее:

```
ID PERIMETER
1 +1.8000000
```

В следующем примере вы создадите таблицу с именем bfp, включающую три объекта, и вычислите периметр каждого объекта в линейных единицах измерения:

```
--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;
```

Выражение SELECT возвращает периметр каждого объекта в трех измерениях:

```
 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

В следующем примере эколог, изучающий поголовье птиц гнездящихся вдоль береговой линии, хочет определить длину береговой линии озер в определенной местности. Озера представлены полигонами в таблице waterbodies. Выражение SELECT, использующее функцию ST_Perimeter, возвращает периметр каждого озера (объекта) в таблице waterbodies.

```
--Create table named waterbodies and add a spatial column (waterbody) to it
```

```
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;
```

Выражение SELECT возвращает следующее:

```
ID PERIMETER
1 +1.8000000
```

В следующем примере вы создадите таблицу с именем bfp, включающую три объекта, и вычислите периметр каждого объекта в линейных единицах измерения:

```
--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;
```

Выражение SELECT возвращает периметр каждого объекта в трех измерениях:

```
 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
```

ST_Point

Описание

ST_Point принимает объект WKT или координаты и ID пространственной привязки и возвращает ST_Point.

Примечание:
При создании пространственных таблиц, которые будут использоваться в ArcGIS, лучше всего создать столбец как супертип геометрии (например, ST_Geometry), а не указывать подтип ST_Geometry.

Синтаксис

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)
```

Тип возвращаемого значения

ST_Point

Пример

Следующее выражение `CREATE TABLE` создает таблицу point_test, которая имеет один столбец точек, PT1.

Функция ST_Point преобразует координаты точки в геометрию ST_Point, прежде чем она будет вставлена в столбец pt1.

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)
);
```

ST_PointFromText

Примечание:
Используется только в Oracle и SQLite; для PostgreSQL используйте ST_Point.

Определение

ST_PointFromText принимает WKT-представление точечного типа и идентификатор пространственной привязки и возвращает точку.

Синтаксис

Oracle

```
sde.st_pointfromtext (wkt varchar2, srid integer)
```

```
sde.st_pointfromtext (wkt varchar2)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

SQLite

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

```
st_pointfromtext (wkt text)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

Тип возвращаемого значения

ST_Point

Пример

Таблица point_test создается с одним столбцом pt1 типа ST_Point.

Функция ST_PointFromText преобразует текстовые координаты точки в формат точки перед тем, как инструкция INSERT вставляет ее в столбец pt1.

Oracle

```
CREATE TABLE point_test (pt1 sde.st_geometry);
```

```
INSERT INTO POINT_TEST VALUES (
 sde.st_pointfromtext ('point (10.01 20.03)', 4326)
);
```

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)
);
```

ST_PointFromWKB

Определение

ST_PointFromWKB принимает WKB-представление и идентификатор пространственной привязки и возвращает ST_Point.

Синтаксис

Oracle

```
sde.st_pointfromwkb (wkb blob, srid integer)
```

```
sde.st_pointfromwkb (wkb blob)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

PostgreSQL

```
sde.st_pointfromwkb (wkb bytea, srid integer)
```

SQLite

```
st_pointfromwkb (wkb blob, srid int32)
```

```
st_pointfromwkb (wkb blob)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

Тип возвращаемого значения

ST_Point

Пример

В этом примере показано, как можно использовать функцию ST_PointFromWKB для создания точки на основе его WKB-представления. В системе пространственной привязки 4326 геометрия представлена точками. В этом примере точки хранятся в столбце geometry таблицы sample_points, затем столбец wkb обновляется с использованием бинарного wkb-представления (с помощью функции ST_AsBinary). Наконец, функция ST_PointFromWKB используется для возврата точек из столбца wkb. В таблице sample-points есть столбец geometry, в котором хранятся точки, а также столбец wkb, в котором хранится WKB-представление точек.

В выражении SELECT функция ST_PointFromWKB используется для получения точек из столбца WKB.

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)
```

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)
```

ST_PointN

Определение

Функция ST_PointN принимает ST_LineString и целочисленный индекс и возвращает точку, являющуюся n-й вершиной в пути ST_LineString.

Синтаксис

Oracle и PostgreSQL

```
sde.st_pointn (line1 sde.st_linestring, index integer)
```

SQLite

```
st_pointn (line1 st_linestring, index int32)
```

Тип возврата

ST_Point

Пример:

Таблица pointn_test создается со столбцом gid, который уникально определяет каждую строку, и столбцом ln1 ST_LineString, в котором хранятся строки linestring. Инструкция INSERT вставляет два значения linestring. У первой строки linestring нет z-координат или измерений, а у второй есть и то, и другое.

Запрос SELECT использует функции ST_PointN и ST_AsText для возврата стандартного текста для второй вершины каждой строки linestring.

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;
```

```
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)
```

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)
```

ST_PointOnSurface

Определение

ST_PointOnSurface принимает ST_Polygon или ST_MultiPolygon и возвращает ST_Point, которая гарантированно лежит на поверхности входного объекта.

Синтаксис

Oracle и PostgreSQL

```
sde.st_pointonsurface (polygon1 sde.st_geometry)
sde.st_pointonsurface (multipolygon1 sde.st_geometry)
```

SQLite

```
st_pointonsurface (polygon1 geometryblob)
st_pointonsurface (multipolygon1 geometryblob)
```

Возвращаемый тип

ST_Point

Пример:

Городской инженер хочет создать точку метки для каждого контура исторического здания. Контуры исторических зданий хранятся в таблице buildings, созданной с помощью следующей инструкции CREATE TABLE:

Функция ST_PointOnSurface создает точку, которая гарантированно будет располагаться на поверхности контуров зданий. Функция ST_PointOnSurface возвращает точку, которую функция ST_AsText преобразует в текстовое представление, поддерживаемое приложением.

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
```

```
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)
```

ST_PolyFromText

Примечание:
Только Oracle и SQLite

Определение

ST_PolyFromText принимает WKT-представление и идентификатор пространственной привязки и возвращает ST_Polygon.

Синтаксис

Oracle

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

```
sde.st_polyfromtext (wkt clob)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

SQLite

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

```
st_polyfromtext (wkt text)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

Тип возвращаемого значения

ST_Polygon

Пример

Таблица polygon_test создается с одним столбцом polygon.

Инструкция SELECT вставляет полигон в столбец polygon с помощью функции 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)
);
```

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)
);
```

ST_PolyFromWKB

Определение

ST_PolyFromWKB принимает WKB-представление и идентификатор пространственной привязки и возвращает ST_Polygon.

Синтаксис

Oracle

```
sde.st_polyfromwkb (wkb blob, srid integer)
```

```
sde.st_polyfromwkb (wkb blob)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

PostgreSQL

```
sde.st_polyfromwkb (wkb bytea, srid integer)
```

SQLite

```
st_polyfromwkb (wkb blob, srid int32)
```

```
st_polyfromwkb (wkb blob)
```

Если вы не указали SRID, пространственная привязка по умолчанию будет 4326.

Тип возвращаемого значения

ST_Polygon

Пример

В этом примере показано, как можно использовать функцию ST_PolyFromWKB для создания полигона на основе его WKB-представления. В системе пространственной привязки 4326 геометрия представлена полигоном. В этом примере полигон сохраняется с идентификатором ID = 1115 в столбце geometry таблицы sample_polys, затем столбец wkb обновляется с использованием WKB-представления (с помощью функции ST_AsBinary). Наконец, функция ST_PolyFromWKB используется для возврата мультиполигона из столбца wkb. В таблице sample_polys есть столбец geometry, в которой сохраняется полигон, а также столбец wkb, в котором хранится WKB-представление полигона.

В выражении SELECT функция ST_PointFromWKB используется для получения точек из столбца WKB.

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',
```

```
 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)
```

ST_Polygon

Описание

Функция доступа ST_Polygon принимает стандартное текстовое представление (WKT) и ID пространственной привязки (SRID) и генерирует ST_Polygon.

Примечание:
При создании пространственных таблиц, которые будут использоваться в ArcGIS, лучше всего создать столбец как супертип геометрии (например, ST_Geometry), а не указывать подтип ST_Geometry.

Синтаксис

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)
```

Тип возвращаемого значения

ST_Polygon

Пример

Следующее выражение `CREATE TABLE` создает таблицы polygon_test с одним столбцом p1. Следующее выражение `INSERT` преобразует кольцо (одновременно замкнутый и простой полигон) в ST_Polygon и вставляет его в столбец p1 с помощью функции 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);
```

```
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)
);
```

ST_Relate

Описание

ST_Relate сравнивает две геометрии и возвращает значение 1 (Oracle и SQLite) либо t (PostgreSQL), если геометрии соответствуют условиям, указанным строкой шаблонной матрицы DE-9IM; в противном случае возвращается 0 (Oracle и SQLite) или F (PostgreSQL).

Существует второй вариант при использовании ST_Relate в SQLite и Oracle: вы можете сравнить две геометрии, чтобы получить строку, представляющую шаблонную матрицу DE-9IM, которая определяет взаимосвязь геометрий друг с другом.

Синтаксис

Oracle

Опция 1

```
sde.st_relate (geometry1 sde.st_geometry, geometry2 sde.st_geometry, patternMatrix
string)
```

Опция 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

Опция 1

```
st_relate (geometry1 geometryblob, geometry2 geometryblob, patternMatrix string)
```

Опция 2

```
st_relate (geometry1 geometryblob, geometry2 geometryblob)
```

Тип возвращаемого значения

Булево значение возвращается для PostgreSQL.

Опция 1 для SQLite и Oracle возвращает целое число.

Опция 2 для SQLite и Oracle возвращает строку.

Примеры

Шаблонная матрица DE-9IM – это устройство для сравнения геометрий. Существует несколько типов таких матриц. Например, вы можете использовать функцию ST_Relate и матрицу шаблона равенства (T*F**FFF*), чтобы определить, равны ли какие-либо две геометрии, но вы также можете указать шаблон DE-9IM (1*F**FFF*).). С последним шаблоном ST_Relate сообщит вам, равны ли две геометрии с первой позицией, которая указывает, является ли внутренняя часть пересечения обеих геометрий линией (размерность 1).

В приведенных ниже примерах создается таблица related_test с тремя пространственными столбцами, в каждый из которых вставляются точечные объекты. В выражении SELECT функция ST_Relate используется для определения того, одинаковы ли точки.

Если вы хотите определить, равны ли геометрии, и вам не нужно находить размерность отношения, используйте вместо этого функцию ST_Equals.

Oracle

Первый пример показывает первую опцию ST_Relate, сравнивающую геометрии на основе шаблонной матрицы DE-9IM и возвращающая 1, если геометрии удовлетворяют требованиям, определенным в матрице, и 0 - если не удовлетворяют.

```
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;
```

Она возвращает следующее:

```
g1=g2     g1=g3      g2=g3

1               0               0
```

В этом примере показана вторая опция. Она сравнивает две геометрии и возвращает матрицу образца DE-9IM.

```
SELECT sde.st_relate (relate_test.g1,relate_test2.g2) AS "g1 rel g2"
 FROM relate_test, relate_test2;
```

Она возвращает следующее:

```
g1 rel g2
0FFFFFFF2
```

PostgreSQL

В примере сравниваются геометрии на основе матрицы шаблонов DE-9IM, чтобы вернуть t, если геометрии соответствуют требованиям, определенным в матрице, или f, если геометрии не соответствуют.

```
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;
```

Она возвращает следующее:

```
g1=g2     g1=g3     g2=g3

t          f         f
```

SQLite

Первый пример показывает первую опцию ST_Relate, сравнивающую геометрии на основе шаблонной

матрицы DE-9IM и возвращающая 1, если геометрии удовлетворяют требованиям, определенным в матрице, и 0 - если не удовлетворяют.

```
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;
```

Она возвращает следующее:

```
g1=g2     g1=g3     g2=g3

1          0         0
```

В этом примере показана вторая опция. Она сравнивает две геометрии и возвращает матрицу образца DE-9IM.

```
SELECT st_relate (relate_test.g1,relate_test2.g2) AS "g1 rel g2"
 FROM relate_test, relate_test2;
```

Она возвращает следующее:

```
g1 rel g2
0FFFFFFF2
```

ST_SRID

Определение

Функция ST_SRID берет объект геометрии и возвращает ID его пространственной привязки.

Синтаксис

Oracle и PostgreSQL

```
sde.st_srid (geometry1 sde.st_geometry)
```

SQLite

```
st_srid (geometry1 geometryblob)
```

Тип возврата

Целочисленное (Integer)

Примеры

Создается следующая таблица:

В следующей инструкции геометрия точки с координатами (10.01, 50.76) вставляется в столбец геометрии g1. При создании геометрии точки ей назначается SRID со значением 4326.

Функция ST_SRID возвращает идентификатор пространственной привязки введенной геометрии.

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
```

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
```

ST_StartPoint

Определение

ST_StartPoint возвращает первую точку линии linestring.

Синтаксис

Oracle и PostgreSQL

```
sde.st_startpoint (ln1 sde.st_geometry)
```

SQLite

```
st_startpoint (ln1 geometryblob)
```

Тип возврата

ST_Point

Примеры

Таблица startpoint_test создается с целочисленным столбцом gid, который уникально определяет строки таблицы, и столбцом ln1 ST_LineString, в котором хранятся строки linestring.

Инструкция INSERT вставляет три строки ST_LineStrings в столбец ln1. У первой строки ST_LineString нет z-координат или измерений, а у второй есть и то, и другое.

Функция ST_StartPoint получает первую точку каждой строки ST_LineString. У первой точки в списке нет z-координаты или измерения, а у второй есть и то, и другое, так как у исходной строки linestring есть z-координаты и измерения.

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)
```

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)
```

ST_Surface

Примечание:
Только Oracle и SQLite

Описание

ST_Surface создает объект поверхности из стандартного текстового представления (WKT). Поверхности похожи на полигоны, но у них есть значения в каждой точке их экстента.

Синтаксис

Oracle

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

SQLite

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

Тип возвращаемого значения

ST_Polygon

Пример

Создается таблица surf_test, и в нее вставляется геометрия поверхности.

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,
```

```
 'polygon',
 'xy',
 'null'
);

INSERT INTO SURF_TEST VALUES (
 1110,
 st_surface ('polygon ((110 120, 110 140, 120 130, 110 120))', 4326)
);
```

ST_SymmetricDiff

Определение

ST_SymmetricDiff берет два объекта геометрии и возвращает объект геометрии, который состоит из частей исходных объектов, не являющихся общими для обоих.

Синтаксис

Oracle и PostgreSQL

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

SQLite

```
st_symmetricdiff (geometry1 geometryblob, geometry2 geometryblob)
```

Возвращаемый тип

Oracle и PostgreSQL

ST_Geometry

SQLite

Geometryblob

Пример:

Для специального отчета глава округа должен определить водоразделы и опасные области, которые не пересекаются.

Таблица watershed содержит столбец id, столбец для хранения имени водораздела (wname) и столбец shape, в котором хранится геометрия территории водораздела.

В таблице plumes в столбце id хранятся идентификаторы участков, а фактическое географическое расположение каждого участка хранится в столбце point.

Функция ST_Buffer создает буфер, окружающий точки вредоносных участков. Функция ST_SymmetricDiff возвращает полигоны на основе буферов мест захоронения опасных отходов и водоразделов, которые не пересекаются.

Симметричная разность мест захоронения опасных отходов и водоразделов позволяет отбросить пересекающиеся области.

Oracle

```
CREATE TABLE watershed (
 id integer,
 wname varchar(40),
 shape sde.st_geometry
);
```

```
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
```

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,
```

```
 '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
```

ST_Touches

Определение

ST_Touches возвращает значение 1 (Oracle и SQLite) либо t (PostgreSQL), если никакие из общих точек геометрий не пересекают их внутренних частей. В противном случае возвращается значение 0 (Oracle) либо f (PostgreSQL). По крайней мере, одна из геометрий должна иметь тип ST_LineString, ST_Polygon, ST_MultiLineString или ST_MultiPolygon.

Синтаксис

Oracle и PostgreSQL

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

SQLite

```
st_touches (geometry1 geometryblob, geometry2 geometryblob)
```

Тип возврата

Логический

Пример:

Начальник попросил ГИС-специалиста предоставить список всех линий канализации с конечными точками, пересекающими другие линии канализации.

Создается таблица sewerlines с тремя столбцами. Первый столбец sewer_id уникально определяет каждую линию канализации. Целочисленный столбец class определяет тип линии канализации, в общем случае связанный с емкостью линии. Столбец sewer содержит геометрию линии канализации.

Запрос SELECT использует функцию ST_Touches для получения списка объектов канализации, которые пересекаются друг с другом.

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)
);

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

         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
```

ST_Transform

Описание

ST_Transform берет в качестве входных данные ST_Geometry и возвращает значения, конвертированные в пространственную привязку, указанную вами через ID пространственной привязки (SRID).

⚠ Внимание:
Если вы зарегистрировали пространственный столбец в базе данных PostgreSQL, используя функцию st_register_spatial_column, то SRID во время регистрации прописывается в таблицу sde_geometry_columns. Если вы создали пространственный индекс в пространственном столбце в базе данных Oracle, то во время создания пространственного индекса SRID прописывается в таблицу st_geometry_columns. Когда вы используете ST_Transform для изменения SRID в ST_Geometry, SRID в таблицах sde_geometry_columns или st_geometry_columns не обновляется.

Если географические системы координат отличаются, ST_Transform выполняет географическое преобразование. Географическое преобразование конвертирует две географические системы координат. Географическое преобразование определяется в конкретном направлении, например, из NAD 1927 в NAD 1983, но функция ST_Transform правильно применит преобразование независимо от исходной и целевой систем координат.

Методы географического преобразования можно разделить на два типа: математические и файловые. Математические методы являются самодостаточными, и им не требуется внешняя информация. Файловые методы используют файлы на диске для расчета значений смещения. Обычно они более точны, чем математические методы. Файловые методы используются в Австралии, Канаде, Германии, Новой Зеландии, Испании и США. Эти файлы (кроме канадских) можно получить из папки установки ArcGIS Pro или напрямую от различных национальных картографических агентств.

Для поддержки преобразований на основе файлов вы должны разместить файлы на сервере, где установлена база данных, в той же относительной структуре папок, что и папка pedata в директории установки ArcGIS Pro.

Например, существует папка с названием `pedata` в папке `Resources` директории установки ArcGIS Pro. Эта папка `pedata` включает в себя несколько папок, а три папки, которые содержат поддерживаемые файловые методы, называются `harn`, `nadcon` и `ntv2`. Либо скопируйте папку `pedata` и ее содержание из директории установки ArcGIS на сервер базы данных, или создайте директорию на сервере базы данных, которая включает в себя поддиректории и файлы поддерживаемого файлового метода преобразования. После того, как файлы окажутся на сервере базы данных, задайте переменную среды операционной системы с именем PEDATAHOME на том же сервере. Установите переменную PEDATAHOME в местоположение директории, которая содержит поддиректории и файлы; например, если папка pedata скопирована в `C:\pedata` на сервере Microsoft Windows, то установите для переменной среды PEDATAHOME значение `C:\pedata`.

Сведения об установке переменных среды см. в документации по операционной системе.

После установки переменной PEDATAHOME необходимо инициировать новый сеанс SQL перед использованием функции ST_Transform; перезагружать сервер не нужно.

Использование ST_Transform в PostgreSQL

В PostgreSQL можно выполнять конвертацию между пространственными привязками с одинаковыми или разными географическими системами координат.

Если данные хранятся в базе данных (а не в базе геоданных), выполните следующие шаги для изменения пространственной привязки данных ST_Geometry в случае одинаковой географической системы координат:

1. Создайте резервную копию таблицы.
2. Создайте второй (целевой) столбец ST_Geometry в таблице.
3. Зарегистрируйте целевой столбец ST_Geometry, указывая новый SRID.
   Указание пространственной привязки для столбца выполняется размещением записи в системной таблице sde_geometry_columns.
4. Запустите функцию ST_Transform и укажите, что преобразованные данные должны выводиться в целевой столбец ST_Geometry.
5. Отмените регистрацию первого столбца (исходного) ST_Geometry.

Если данные хранятся в базе геоданных, вы должны использовать инструменты ArcGIS, чтобы перепроецировать данные в новый класс пространственных объектов. Запуск ST_Transform у класса пространственных объектов базы геоданных позволяет обойти использование функциональности для обновления системных таблиц базы геоданных с новым SRID.

Использование ST_Transform в Oracle

В Oracle можно выполнять конвертацию между пространственными привязками с одинаковыми или разными географическими системами координат.

Если данные хранятся в базе данных (а не в базе геоданных), и пространственный индекс в пространственном столбце не определен, то вы можете добавить второй столбец ST_Geometry и направить преобразованные данные в него. Вы можете оставить оба столбца, исходный (источника) ST_Geometry и целевой ST_Geometry в таблице. Однако вы можете отображать только один столбец в ArcGIS, используя представление или изменяя определение слоя запроса для таблицы.

Если данные хранятся в базе данных (а не в базе геоданных), и в пространственном столбце имеется определенной пространственный индекс, то вы не сможете защитить исходный столбец ST_Geometry. После определения пространственного индекса в столбце ST_Geometry в таблицу метаданных st_geometry_columns записывается SRID. ST_Transform не обновляет эту таблицу.

1. Создайте резервную копию таблицы.
2. Создайте второй (целевой) столбец ST_Geometry в таблице.
3. Запустите функцию ST_Transform и укажите, что преобразованные данные должны выводиться в целевой столбец ST_Geometry.
4. Удалите пространственный индекс из исходного столбца ST_Geometry.
5. Удалите исходный столбец ST_Geometry.
6. Создайте пространственный индекс в целевом столбце ST_Geometry.

Если данные хранятся в базе геоданных, вы должны использовать инструменты ArcGIS для

перепроецирования данных в новый класс пространственных объектов. Запуск ST_Transform для класса пространственных объектов базы геоданных обходит функциональные возможности обновления системных таблиц базы геоданных новым SRID.

Использование ST_Transform в SQLite

В SQLite можно выполнять конвертацию между пространственными привязками с одинаковыми или разными географическими системами координат.

Синтаксис

Исходная и целевая пространственные привязки не имеют единой географической системы координат

Oracle и PostgreSQL

```
sde.st_transform (geometry1 sde.st_geometry, srid integer)
```

SQLite

```
st_transform (geometry1 geometryblob, srid in32)
```

Исходная и целевая пространственные привязки не имеют единой географической системы координат

Oracle

```
sde.st_transform (g1 sde.st_geometry, srid integer, geogtrans_id integer)
```

PostgreSQL

Опция 1: `sde.st_transform (g1 sde.st_geometry, srid int)`

Опция 2: `sde.st_transform (g1 sde.st_geometry, srid int, [geogtrans_id int])`

Опция 3: `sde.st_transform (g1 sde.st_geometry, srid int, [extent double] [prime meridian double] [unit conversion factor double])`

В опции 3 дополнительно можно указать экстент в виде списка координат, разделенных запятыми, в следующем порядке: левая нижняя x-координата, левая нижняя y-координата, верхняя правая x-координата, верхняя правая y-координата. Если вы не укажете экстент, ST_Transform использует больший, более общий экстент.

Если экстент задан, параметры начального меридиана и коэффициент преобразования единиц измерения являются дополнительными. Эту информацию необходимо предоставить, только если указанные значения экстента не используют Гринвичский начальный меридиан или десятичные градусы.

SQLite

```
st_transform (geometry1 geometryblob, srid int32, geogtrans_id int32)
```

Тип возвращаемого значения

Oracle и PostgreSQL

ST_Geometry

SQLite

Geometryblob

Примеры

Преобразование данных, когда географические системы координат пространственных привязок источника и назначения совпадают

В следующем примере создается таблица transform_test с двумя строковыми столбцами: ln1 и ln2. Линия вставляется в ln1 с SRID 4326. Далее в выражении UPDATE используется функция ST_Transform, которая преобразует строчное значение столбца ln1 из координатной привязки, назначенной SRID 4326, в координатную привязку, назначенную SRID 3857, и помещает его в столбец ln2.

Примечание:
Идентификаторы SRID 4326 и 3857 имеют один и тот же географический датум.

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);
```

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);
```

Преобразование данных, когда географические системы координат пространственных привязок источника и назначения не совпадают

В следующем примере создается таблица n27, содержащая столбец ID и столбец геометрии. В таблицу n27 с SRID 4267 вставляется точка. SRID 4267 использует географическую систему координат NAD 1927.

Далее создается таблица n83, и функция ST_Transform вставляет геометрию из таблицы n27 в таблицу n83, но уже с SRID равным 4269 и ID географического преобразования равным 1241. SRID 4269 использует географическую систему координат NAD 1983, а 1241 – стандартный ID для преобразования NAD_1927_To_NAD_1983_NADCON. Это файловое преобразование, и его можно использовать для 48 континентальных штатов США.

Подсказка:

Список поддерживаемых географических преобразований см. в технической статье Esri 000004829 и в ссылках, предоставленных в разделе статьи **Связанная информация**.

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.
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
);
```

Если переменная PEDATAHOME задана правильно, выражение SELECT, выполняемое для таблицы n83, возвращает следующее:

```
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,
 '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)
);
```

ST_Union

Определение

ST_Union возвращает объект геометрии, являющийся комбинацией двух исходных объектов.

Синтаксис

Oracle и PostgreSQL

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

SQLite

```
st_union (geometry1 geometryblob, geometry2 geometryblob)
```

Тип возврата

Oracle и PostgreSQL

ST_Geometry

SQLite

Geometryblob

Пример:

Таблица sensitive_areas содержит ID находящихся под угрозой учреждений в дополнение к столбцу shape, который хранит геометрии полигонов учреждений.

В таблице hazardous_sites в столбце id хранятся идентификаторы участков, а фактическое географическое расположение каждого участка хранится в столбце point.

Функция ST_Buffer создает буфер, окружающий вредоносные участки. Функция ST_Union создает полигоны на основе объединения полигонов буферизированных вредоносных участков и важных областей. Функция ST_Area возвращает площадь этих полигонов.

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)
);

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)
);
```

```
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 (
```

```
 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
```

ST_Within

Определение

ST_Within возвращает 1 (Oracle и SQLite) либо t (PostgreSQL), если первый объект ST_Geometry находится полностью во втором. В противном случае возвращается 0 (Oracle SQLite) либо f (PostgreSQL).

Синтаксис

Oracle и PostgreSQL

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

SQLite

```
st_within (geometry1 geometryblob, geometry2 geometryblob)
```

Тип возврата

Логический

Пример:

В примере ниже создаются две таблицы: zones и squares. Инструкция SELECT ищет все площади, пересекающие участок, но не находящиеся полностью в его пределах.

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)
```

```
);

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,
```

```
 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 (
```

```
 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
```

ST_X

Определение

ST_X берет ST_Point как входной параметр и возвращает ее координату x. В SQLite ST_X также может обновить координату x ST_Point.

Синтаксис

Oracle и PostgreSQL

```
sde.st_x (point1 sde.st_point)
```

SQLite

```
st_x (point1 geometryblob)
st_x (input_point geometryblob, new_Xvalue double)
```

Тип возврата

Двойная точность

Функция ST_X в SQLite также может обновить координату x точки. В этом случае возвращается geometryblob.

Примеры

Таблица x_test создается с двумя столбцами: gid, который уникально определяет каждую строку, и точечным столбцом pt1.

Инструкция INSERT вставляет две строки. Одна из них – это точка без z-координаты или измерения. В другом столбце есть z-координата или измерение.

Запрос SELECT использует функцию ST_X для получения координаты x каждого точечного объекта.

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
```

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
```

Функция ST_X в SQLite также может обновить значение координаты существующей точки. В данном примере функция ST_X используется для обновления значение координаты x первой точки в x_test.

```
UPDATE x_test
 SET pt1=st_x(
  (SELECT pt1 FROM x_test WHERE gid=1),
  10.04
  )
 WHERE gid=1;
```

ST_Y

Определение

ST_Y берет ST_Point как входной параметр и возвращает ее координату y. В SQLite ST_Y также может обновить координату y ST_Point.

Синтаксис

Oracle и PostgreSQL

```
sde.st_y (point1 sde.st_point)
```

SQLite

```
double   st_y (point1  geometryblob)
geometry st_y (input_shape geometryblob, new_Yvalue double)
```

Тип возврата

Двойная точность

Функция ST_Y в SQLite также может обновить координату y точки. В этом случае возвращается geometryblob.

Пример:

Таблица y_test создается с двумя столбцами: gid, который уникально определяет каждую строку, и точечным столбцом pt1.

Инструкция INSERT вставляет две строки. Одна из них – это точка без z-координаты или измерения. В другом столбце есть z-координата или измерение.

Запрос SELECT использует функцию ST_Y для получения координаты y каждого точечного объекта.

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
```

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
```

Функция ST_Y также может обновить значение координаты существующей точки. В данном примере функция ST_Y используется для обновления значение координаты y второй точки в y_test.

```
UPDATE y_test
 SET pt1=st_y(
  (SELECT pt1 FROM y_test WHERE gid=2),
  20.1
  )
 WHERE gid=2;
```

ST_Z

Определение

ST_Z принимает ST_Point как входной параметр и возвращает координату z (высоту). В SQLite ST_Z также может обновить координату z ST_Point.

Синтаксис

Oracle и PostgreSQL

```
sde.st_z (geometry1 sde.st_point)
```

SQLite

```
st_z (geometry  geometryblob)
st_z (input_shape geometryblob, new_Zvalue double)
```

Тип возврата

Oracle

Число (Number)

PostgreSQL

Целочисленное (Integer)

SQLite

При возврате ST_Z координаты z точки используется двойная точность. При обновлении ST_Z координаты z точки возвращается geometryblob.

Пример:

Таблица z_test создается с двумя столбцами: id, который уникально определяет каждую строку, и точечным столбцом geometry. Инструкция INSERT вставляет строку в таблицу z_test.

Инструкция SELECT указывает ИД столбца и z-координату двойной точности точки, вставленной предыдущей инструкцией.

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,
```

```
 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
```

Функция ST_Z в SQLite также может обновить значение координаты существующей точки. В данном примере функция ST_Z используется для обновления значения координаты z первой точки в z_test.

```
UPDATE z_test
 SET pt1=st_z(
  (SELECT pt1 FROM z_test where id=1), 32.04)
 WHERE id=1;
```