Referenz für ST_Geometry-SQL-Funktionen

THE SCIENCE OF WHERE

Inhaltsverzeichnis

Mit ST_Geometry verwendete SQL-Funktionen . . . 6
SQL und Esri ST_Geometry . . . 12
Laden der SQLite-Bibliothek ST_Geometry . . . 15
Konstruktorfunktionen für ST_Geometry . . . 16
Räumliche Accessor-Funktionen . . . 20
Räumliche Beziehungen . . . 28
Funktionen für räumliche Beziehungen . . . 30
Räumliche Operationen . . . 42
Funktionen für räumliche Operationen . . . 44
Parametrische Kreise, Ellipsen und Keile . . . 50
ST_Aggr_ConvexHull . . . 53
ST_Aggr_Intersection . . . 55
ST_Aggr_Union . . . 58
ST_Area . . . 61
ST_AsBinary . . . 64
ST_AsText . . . 66
ST_Boundary . . . 68
ST_Buffer . . . 72
ST_Centroid . . . 76
ST_Contains . . . 79
ST_ConvexHull . . . 83
ST_CoordDim . . . 87
ST_Crosses . . . 92
ST_Curve . . . 96
ST_Difference . . . 98
ST_Dimension . . . 102
ST_Disjoint . . . 106
ST_Distance . . . 110
ST_DWithin . . . 114
ST_EndPoint . . . 120
ST_Entity . . . 123
ST_Envelope . . . 126

ST_EnvIntersects . . . 132
ST_Equals . . . 135
ST_Equalsrs . . . 138
ST_ExteriorRing . . . 139
ST_GeomCollection . . . 142
ST_GeomCollFromWKB . . . 145
ST_Geometry . . . 147
ST_GeometryN . . . 154
ST_GeometryType . . . 156
ST_GeomFromCollection . . . 160
ST_GeomFromText . . . 162
ST_GeomFromWKB . . . 165
ST_GeoSize . . . 168
ST_InteriorRingN . . . 169
ST_Intersection . . . 172
ST_Intersects . . . 177
ST_Is3d . . . 181
ST_IsClosed . . . 185
ST_IsEmpty . . . 190
ST_IsMeasured . . . 194
ST_IsRing . . . 198
ST_IsSimple . . . 201
ST_Length . . . 204
ST_LineFromText . . . 207
ST_LineFromWKB . . . 209
ST_LineString . . . 212
ST_M . . . 214
ST_MaxM . . . 217
ST_MaxX . . . 220
ST_MaxY . . . 223
ST_MaxZ . . . 226
ST_MinM . . . 229
ST_MinX . . . 232
ST_MinY . . . 235

ST_MinZ . . . 238
ST_MLineFromText . . . 241
ST_MLineFromWKB . . . 243
ST_MPointFromText . . . 246
ST_MPointFromWKB . . . 248
ST_MPolyFromText . . . 251
ST_MPolyFromWKB . . . 253
ST_MultiCurve . . . 256
ST_MultiLineString . . . 257
ST_MultiPoint . . . 259
ST_MultiPolygon . . . 261
ST_MultiSurface . . . 263
ST_NumGeometries . . . 264
ST_NumInteriorRing . . . 267
ST_NumPoints . . . 270
ST_OrderingEquals . . . 273
ST_Overlaps . . . 275
ST_Perimeter . . . 279
ST_Point . . . 284
ST_PointFromText . . . 286
ST_PointFromWKB . . . 288
ST_PointN . . . 291
ST_PointOnSurface . . . 294
ST_PolyFromText . . . 297
ST_PolyFromWKB . . . 299
ST_Polygon . . . 302
ST_Relate . . . 304
ST_SRID . . . 309
ST_StartPoint . . . 311
ST_Surface . . . 314
ST_SymmetricDiff . . . 316
ST_Touches . . . 320
ST_Transform . . . 324
ST_Union . . . 331

ST_Within 335
ST_X 339
ST_Y 342
ST_Z 345

Mit ST_Geometry verwendete SQL-Funktionen

In diesem Referenzdokument werden die Funktionen aufgelistet und beschrieben, die für die Verwendung mit dem räumlichen Esri Datentyp "ST_Geometry" in Oracle, PostgreSQL und SQLite verfügbar sind.

Esri ST_Geometry-SQL-Funktionen und -Typen werden erstellt, wenn Sie einen der folgenden Schritte ausführen:

- Erstellen einer Geodatabase in einer Oracle-Datenbank
- Verwenden von "ST_Geometry" beim Erstellen einer Geodatabase in einer PostgreSQL-Datenbank
- Installieren des räumlichen Datentyps "ST_Geometry" in einer Oracle- oder PostgreSQL-Datenbank
- Erstellen einer SQLite-Datenbank, die den räumlichen Datentyp "ST_Geometry" enthält, mithilfe des Geoverarbeitungswerkzeugs SQLite-Datenbank erstellen bzw. der entsprechenden ArcPy-Funktion und Laden von ST_Geometry-Funktionen zur Verwendung mit der Datenbank
- Laden der ST_Geometry-Funktionen zur Verwendung mit einer mobilen Geodatabase

In Oracle- und PostgreSQL-Datenbanken werden der ST_Geometry-Typ und die zugehörigen Funktionen in einem Schema mit dem Namen "SDE" erstellt. In SQLite werden der Typ und die Funktionen in einer Bibliothek gespeichert, die Sie laden müssen, bevor Sie SQL für die SQLite-Datenbank oder mobile Geodatabase ausführen.

Tipp:

Informationen zum Esri ST_Geometry-Typ finden Sie auf den folgenden ArcGIS Pro-Hilfeseiten:

- "ST_Geometry" in PostgreSQL
- "ST_Geometry" in Oracle
- Datenbanken und ST_Geometry
- Laden der SQLite-Bibliothek "ST_Geometry"
- Laden von "ST_Geometry" in eine mobile Geodatabase für den SQL-Zugriff

Format der SQL-Funktionsseiten

Die Funktionsseiten in diesem Dokument sind wie folgt aufgebaut:

- Definition: Eine kurze Erläuterung des Verhaltens der Funktion
- Syntax: Die SQL-Syntax für die Funktion

  Hinweis:
  Bei relationalen Operatoren spielt die Reihenfolge, in der die Parameter angegeben werden, eine wichtige Rolle: Der erste Parameter bezeichnet die Tabelle, aus der die Auswahl getroffen wird, und der zweite Parameter bezeichnet die Tabelle, die als Filter verwendet wird.

- Rückgabetyp: Der Datentyp, der von der Funktion zurückgegeben wird
- Beispiel: Beispiele, in denen die bestimmte Funktion zum Einsatz kommt

Liste der SQL-Funktionen

Klicken Sie auf die Links unten, um zu den Funktionen zu wechseln, die Sie mit dem ST_Geometry-Typ in Oracle,

PostgreSQL und SQLite verwenden können.

Beim Verwenden von "ST_Geometry"-Funktionen in Oracle müssen Sie das Präfix `sde` zusammen mit den Funktionen und Operatoren angeben. Beispiel: Bei "ST_Buffer" müssen Sie "sde.ST_Buffer" angeben. Am Präfix `sde` erkennt die Software, dass die Funktion im Schema des Benutzers "sde" gespeichert ist. Für PostgreSQL ist die Angabe zwar optional, wird jedoch empfohlen. Beziehen Sie bei Verwendung der Funktionen mit SQLite die Qualifikation nicht mit ein, da SQLite-Datenbanken kein SDE-Schema enthalten.

Wenn Sie Well-Known Text-Zeichenfolgen als Eingabe mit einer ST_Geometry-SQL-Funktion bereitstellen, können Sie zur Angabe sehr großer oder sehr kleiner Werte die wissenschaftliche Schreibweise verwenden. Wenn Sie beispielsweise Koordinaten bei der Konstruktion eines Features mithilfe von Well-Known Text angeben und eine der Koordinaten 0.00002350000181650102 6 lautet, können Sie stattdessen `2.3500001816501026e-005` eingeben.

Tipp:

Informationen zu anderen räumlichen Datentypen – wie den PostGIS-Typen, Oracle SDO_Geometry, räumlichen Microsoft SQL Server-Typen, IBM Db2 ST_Geometry oder SAP HANA ST_Geometry – und den von ihnen verwendeten Funktionen finden Sie in der Herstellerdokumentation des jeweiligen Datenbankmanagementsystem.

Die folgenden ST_Geometry-SQL-Funktionen von Esri werden nach Verwendung gruppiert.

Konstruktorfunktionen

Konstruktorfunktionen verwenden einen Geometrietyp oder eine Textbeschreibung der Geometrie und erstellen eine Geometrie. In der folgenden Tabelle werden die Konstruktorfunktionen aufgelistet und sie enthält Angaben dazu, welche ST_Geometry-Implementierungen sich jeweils unterstützen.

Konstruktorfunktionen

Funktion	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
ST_MPointFromWKB	X	X	X

Funktion	Oracle	PostgreSQL	SQLite
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

Accessor-Funktionen

Es gibt eine Vielzahl von Funktionen, die eine oder mehrere Geometrien als Eingabe verwenden und bestimmte Informationen über sie zurückgeben.

Einige dieser Accessor-Funktionen überprüfen, ob ein oder mehrere Features bestimmte Kriterien erfüllen. Wenn die Geometrie die Kriterien erfüllt, gibt die Funktion den Wert "1" (Oracle und SQLite) oder "t" (PostgreSQL) für "true" (Wahr) zurück. Wenn die Geometrie die Kriterien nicht erfüllt, gibt die Funktion den Wert "0" (Oracle und SQLite) oder "f" (PostgreSQL) für "false" (Falsch) zurück.

Wenn nicht anders angegeben, gelten diese Funktionen für alle Implementierungen.

Accessor-Funktionen

ST_Area
ST_AsBinary
ST_AsText
ST_CoordDim
ST_Dimension
ST_EndPoint
ST_Entity
ST_Equalsrs (nur PostgreSQL)
ST_ExteriorRing
ST_GeomFromCollection (nur PostgreSQL)

ST_GeometryType
ST_GeoSize (nur 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

Relationale Funktionen

Relationale Funktionen verwenden Geometrien als Eingabe und ermitteln, ob eine räumliche Beziehung zwischen den Geometrien besteht. Wenn die Bedingungen einer räumlichen Beziehung erfüllt sind, geben diese Funktionen den Wert "1" (Oracle und SQLite) oder ein "t" (PostgreSQL) für "true" (Wahr) aus. Wenn die Bedingungen nicht erfüllt sind (es besteht keine Beziehung), geben diese Funktionen den Wert "0" (Oracle und SQLite) oder ein "f" (PostgreSQL) für "false" (Falsch) aus.

Wenn nicht anders angegeben, gelten diese Funktionen für alle Implementierungen.

Relationale Funktionen

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

Funktionen für Geometrie-Operationen

Mit diesen Funktionen werden für räumliche Daten räumliche Operationen durchgeführt und eine Geometrie zurückgeben.

Wenn nicht anders angegeben, gelten diese Funktionen für alle Implementierungen.

Funktionen für Geometrie-Operationen

ST_Aggr_ConvexHull (nur Oracle und SQLite)
ST_Aggr_Intersection (nur Oracle und 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 und Esri ST_Geometry

Wenn Sie mit den in einer Geodatabase oder Datenbank gespeicherten Daten arbeiten möchten, in der der ST_Geometry-Typ installiert ist, können Sie die SQL-Funktionen (Structured Query Language), Datentypen und Tabellenformate des Datenbankmanagementsystems verwenden. SQL ist eine Datenbanksprache, die Datendefinitionen und Befehle zur Datenbearbeitung unterstützt.

Erfolgt der Zugriff auf die Daten über SQL, können externe Anwendungen mit den Tabellendaten arbeiten, die von der Geodatabase oder Datenbank verwaltet werden. Bei diesen externen Anwendungen kann es sich um nichträumliche Datenbankanwendungen oder benutzerdefinierte räumliche Anwendungen handeln.

Wenn Sie Daten mit SQL in eine Geodatabase oder Datenbank einfügen oder Daten in einer Geodatabase oder Datenbank mit SQL bearbeiten, führen Sie nach dem Ausführen der SQL-Anweisung eine `COMMIT`- oder eine `ROLLBACK`-Anweisung aus, um die Änderungen zu bestätigen oder rückgängig zu machen. Dadurch werden Sperren für die Zeilen, Seiten oder Tabellen verhindert, die Sie bearbeiten.

Einfügen von ST_Geometry-Daten mit SQL

Sie können SQL verwenden, um räumliche Daten in eine Datenbank oder Geodatabase-Tabelle mit einer ST_Geometry-Spalte einzufügen. Verwenden Sie zum Einfügen bestimmter Geometrietypen die ST_Geometry-Konstruktorfunktionen. Darüber hinaus können Sie auch festlegen, dass die Ausgabe von bestimmten Funktionen für räumliche Operationen in einer vorhandenen Tabelle erfolgen soll.

Beachten Sie beim Einfügen einer Geometrie in eine Tabelle mit SQL Folgendes:

- Geben Sie eine gültige Raumbezugs-ID (SRID) an.
- Alle Geometrien in einer Spalte müssen dieselbe SRID aufweisen.
- Wenn Sie die Tabelle weiterhin mit ArcGIS verwenden möchten, darf das als Objekt-ID verwendete Feld nicht NULL sein.

Raumbezugs-IDs

Die SRID, die Sie beim Einfügen einer Geometrie in eine Tabelle in Oracle angeben und die den räumlichen Typ "ST_Geometry" verwendet, muss sich in der Tabelle "ST_SPATIAL_REFERENCES" befinden. Zudem muss es für diese SRID in der Tabelle "SDE.SPATIAL_REFERENCES" einen entsprechenden Datensatz geben. Die SRID, die Sie beim Einfügen einer Geometrie in eine Tabelle in PostgreSQL angeben und die den räumlichen Typ "ST_Geometry" verwendet, muss sich in der Tabelle "public.sde_spatial_references" befinden. Diese Tabellen werden automatisch mit Raumbezügen und SRIDs gefüllt.

Die SRID, die Sie beim Einfügen einer Geometrie in eine Tabelle in SQLite angeben und die den räumlichen Typ "ST_Geometry" verwendet, muss sich in der Tabelle "st_spatial_reference_systems" befinden.

Wenn Sie einen benutzerdefinierten Raumbezug benötigen, der nicht in der Tabelle vorhanden ist, laden oder erstellen Sie am besten eine Feature-Class, die die gewünschten Raumbezugswerte enthält. Achten Sie darauf, dass die erstellte Feature-Class einen ST_Geometry-Speicher verwendet. Dadurch wird in Oracle in den Tabellen SDE.SPATIAL_REFERENCES und ST_SPATIAL_REFERENCES, in PostgreSQL in "public.sde_spatial_references" bzw. in SQLite in "st_aux_spatial_reference_systems_table" ein Datensatz erstellt.

In Geodatabases können Sie eine Abfrage für die Tabelle LAYERS (Oracle) oder "sde_layers" (PostgreSQL) durchführen, um die SRID zu ermitteln, die der räumlichen Tabelle zugewiesen ist. Anschließend können Sie diese SRID zum Erstellen von räumlichen Tabellen sowie zum Einfügen von Daten mithilfe von SQL verwenden.

Hinweis:
Zur Verwendung der Beispiele in diesem Dokument wurde in die Tabellen "ST_SPATIAL_REFERENCES" und "sde_spatial_references" ein Datensatz zur Kennzeichnung eines unbekannten Raumbezugs eingebunden. Dieser Datensatz hat die SRID 0. Sie können diese SRID für die Beispiele in diesem Dokument verwenden. Hierbei handelt es sich jedoch nicht um eine offizielle SRID. Sie wird lediglich zum Ausführen von SQL-Beispielcode bereitgestellt. Diese SRID sollten Sie nicht für Ihre Produktionsdaten verwenden.

Objekt-IDs

Damit ArcGIS Daten abfragen kann, muss die Tabelle ein Feld mit einer eindeutigen Objekt-ID enthalten.

Feature-Classes, die mit ArcGIS erstellt werden, enthalten immer ein Objekt-ID-Feld, das als Kennungsfeld verwendet wird. Beim Einfügen von Datensätzen in die Feature-Class mithilfe von ArcGIS wird in das Objekt-ID-Feld immer ein Einzelwert eingefügt. Das Objekt-ID-Feld in einer Geodatabase-Tabelle wird von ArcGIS verwaltet. Das von ArcGIS erstellte Objekt-ID-Feld in einer Datenbanktabelle wird vom Datenbankmanagementsystem verwaltet.

Wenn Sie Datensätze in eine Geodatabase-Tabelle mithilfe von SQL einfügen, müssen Sie für die Objekt-ID einen gültigen Einzelwert einfügen.

Datenbanktabellen, die Sie außerhalb von ArcGIS erstellen, müssen ein Feld (oder mehrere Felder) enthalten, die von ArcGIS als Objekt-ID verwendet werden können. Wenn Sie den datenbanknativen Datentyp für automatisches Inkrementieren für das ID-Feld in der Tabelle verwenden, wird dieses Feld von der Datenbank ausgefüllt, wenn Sie einen Datensatz mithilfe von SQL hinzufügen. Achten Sie beim Bearbeiten der Tabelle mit SQL darauf, dass Sie für die ID einen Einzelwert angeben, wenn Sie die Werte im Feld mit der eindeutigen Kennung manuell verwalten.

Hinweis:
Daten aus Tabellen mit einem Feld für eindeutige Kennungen, die nicht von ArcGIS oder dem Datenbankmanagementsystem verwaltet werden, können nicht veröffentlicht werden.

Bearbeiten von ST_Geometry-Daten mit SQL

In SQL an vorhandenen Datensätzen vorgenommene Änderungen wirken sich häufig auf die in der Tabelle gespeicherten nicht räumlichen Attribute aus. Die Daten in der ST_Geometry-Spalte können jedoch mithilfe der Konstruktorfunktionen in SQL-`UPDATE` Anweisungen bearbeitet werden.

Wenn die Daten in einer Geodatabase gespeichert sind, müssen Sie bei der Bearbeitung mit SQL weitere Richtlinien berücksichtigen:

- Datensätze dürfen nicht mit SQL aktualisiert werden, wenn die Daten als versioniert registriert oder für die Geodatabase-Archivierung aktiviert wurden.
- Attribute, die sich auf andere Objekte in der Datenbank auswirken, die sich wiederum auf das Verhalten der Geodatabase auswirken, dürfen nicht geändert werden. Hierzu zählen beispielsweise Beziehungsklassen, Feature-bezogene Annotationen, Topologie, Attributregeln und Netzwerke.
- SQL darf nicht zum Ändern von Tabellenschemas verwendet werden.

⚠ Vorsicht:

Wenn Sie SQL für den Zugriff auf die Geodatabase verwenden, stehen einige Geodatabase-Funktionen wie etwa Versionierung, Topologie, Netzwerke, Terrains, Feature-bezogene Annotationen oder andere Klassen- bzw. Workspace-Erweiterungen nicht zur Verfügung. Funktionen des Datenbankmanagementsystems wie Trigger und gespeicherte Prozeduren können u. U. zur Verwaltung von Beziehungen zwischen Tabellen verwendet werden, die für bestimmte Geodatabase-Funktionen erforderlich sind. Wenn Sie SQL-Befehle für die Geodatabase ohne diese zusätzliche Funktionen ausführen (d. h., wenn Sie beispielsweise mit `INSERT`-Anweisungen einer Tabelle mit aktivierter Geodatabase-Archivierung Datensätze oder einer vorhandenen Feature-Class eine Spalte hinzufügen), stehen die Geodatabase-Funktionen nicht zur Verfügung und die Beziehungen zwischen den Daten in der Geodatabase werden möglicherweise beschädigt.

Laden der SQLite-Bibliothek ST_Geometry

Führen Sie folgende Schritte aus, bevor Sie SQL-Befehle mit ST_Geometry-Funktionen für eine SQLite-Datenbank ausführen:

1. Laden Sie die ZIP-Datei mit den "ST_Geometry"-Bibliotheken für ArcGIS Pro (SQLite) von My Esri herunter, und entpacken Sie sie.
2. Installieren Sie einen SQL-Editor auf demselben Computer wie die Datenbank.
3. Speichern Sie die Datei "ST_Geometry" an einem Ort, auf den die SQLite-Datenbank und der SQL-Editor, über den Sie "ST_Geometry" laden, zugreifen können.
   Wenn sich die SQLite-Datenbank auf einem Microsoft Windows-Computer befindet, ist die Datei `stgeometry_sqlite.dll` zu verwenden. Wenn sich die SQLite-Datenbank auf einem Linux-Computer befindet, ist die Datei `libstgeometry_sqlite.so` zu verwenden.
4. Öffnen Sie den SQL-Editor, und stellen Sie eine Verbindung mit der SQLite-Datenbank her.
5. Laden Sie die Bibliothek "ST_Geometry".
   Im ersten Beispiel wird die Bibliothek für eine SQLite-Datenbank auf einem Windows-Computer geladen. Im zweiten Beispiel wird die Bibliothek für eine SQLite-Datenbank auf einem Linux-Computer geladen.

```
--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'
);
```

Sie können nun SQL-Befehle mit ST_Geometry-Funktionen für die SQLite-Datenbank ausführen.

Konstruktorfunktionen für ST_Geometry

Mit Konstruktorfunktionen wird aus einer Well-Known Text-Beschreibung oder einer Well-known Binary eine Geometrie erstellt.

Wenn Sie zum Konstruieren einer Geometrie eine bekannte Textbeschreibung bereitstellen, muss die Messkoordinate zuletzt angegeben werden. Wenn der Text beispielsweise Koordinaten für Y, Y, Z und M enthält, müssen sie in dieser Reihenfolge angegeben werden.

Eine Geometrie kann null oder mehr Punkte aufweisen. Eine Geometrie gilt als leer, wenn sie null Punkte enthält. Der Subtype "Point" ist die einzige Geometrie, die auf null oder einen Punkt beschränkt ist. Alle anderen Subtypes können null oder mehr Punkte besitzen.

In den folgenden Abschnitten werden die übergeordnete Objektklasse der Geometrie und die Subclass-Geometrien beschrieben. Außerdem werden die Funktionen aufgeführt, die jeweils erstellt werden können.

Geometrien können auch als Ergebnis aus einer räumlichen Operation, die für vorhandene Geometrien ausgeführt wird, erstellt werden.

Übergeordnete Objektklasse der Geometrie

Die übergeordnete Objektklasse "ST_Geometry" kann nicht instanziiert werden. Auch wenn Sie eine Spalte als Datentyp "ST_Geometry" festlegen können, werden die tatsächlich eingefügten Daten als Punkt-, Linestring-, Polygon-, Multipoint-, Multilinestring- oder Multipolygon-Entitäten festgelegt.

Die folgenden Funktionen können zum Erstellen einer übergeordneten Objektklasse verwendet werden, um die oben genannten Entitätstypen zu speichern.

- ST_Geometry
- ST_GeomFromText (nur Oracle und SQLite)
- ST_GeomFromWKB

Subclasses

Sie können ein Feature als bestimmte Subclass festlegen. In diesem Fall kann nur der für diese Subclass zulässige Entitätstyp eingefügt werden. Beispielsweise können mit "ST_PointFromWKB" nur Punktentitäten erstellt werden.

ST_Point

Ein "ST_Point" ist eine nulldimensionale Geometrie, die eine einzelne Position in einem Koordinatenbereich einnimmt. Ein "ST_Point" besitzt einen einzigen XY-Koordinatenwert, ist immer eine einfache Geometrie und weist eine NULL-Grenze auf. "ST_Point" kann zum Definieren von Features wie Ölquellen, Landmarks und Sammelstellen für Wasserproben verwendet werden.

Mit den folgenden Funktionen können Punkte erstellt werden:

- ST_Point
- ST_PointFromText (nur Oracle und SQLite)
- ST_PointFromWKB

ST_MultiPoint

Ein "ST_Multipoint" ist eine Sammlung von "ST_Points" und besitzt wie seine Elemente die Dimension 0. Ein "ST_MultiPoint" ist einfach, wenn keines seiner Elemente den gleichen Koordinatenbereich einnimmt. Die Grenze eines "ST_MultiPoint" beträgt NULL. "ST_MultiPoints" können Dinge wie Übertragungsmuster und Orte von Krankheitsausbrüchen definieren.

Mit den folgenden Funktionen können Multipoint-Geometrien erstellt werden:

- ST_MultiPoint
- ST_MPointFromText (nur Oracle)
- ST_MPointFromWKB

ST_LineString

Ein "ST_LineString" ist ein eindimensionales Objekt, das als Abfolge von Punkten gespeichert ist und einen linear interpolierten Pfad definiert. Der "ST_LineString" besitzt eine einfache Geometrie, wenn er seinen Innenbereich nicht schneidet. Die Endpunkte (Grenze) eines geschlossenen "ST_LineString" nehmen denselben Punkt im Raum ein. Ein "ST_LineString" ist ein Ring, wenn er geschlossen ist und eine einfache Geometrie aufweist. Wie die anderen Eigenschaften, die von der übergeordneten Objektklasse "ST_Geometry" übernommen werden, besitzen "ST_LineStrings" eine Länge. Mit "ST_LineStrings" werden häufig lineare Features wie Straßen, Flüsse und Stromleitungen definiert.

Die Endpunkte bilden im Normalfall die Grenze eines "ST_LineString". Wenn der "ST_LineString" geschlossen ist, ist die Grenze jedoch NULL. Der Innenbereich eines "ST_LineString" ist der verbundene Pfad, der zwischen den Endpunkten liegt. Wenn er geschlossen ist, gibt es einen durchgehenden Innenbereich.

Mit den folgenden Funktionen können Linestrings erstellt werden:

- ST_LineString
- ST_LineFromText (nur Oracle und SQLite)
- ST_LineFromWKB
- ST_Curve (nur Oracle und SQLite)

ST_MultiLineString

Ein ST_MultiLineString ist eine Sammlung von ST_LineStrings.

Die Grenze eines "ST_MultiLineString" bilden die Endpunkte der "ST_LineString"-Elemente, die nicht geschnitten werden. Die Grenze eines "ST_MultiLineString" ist NULL, wenn alle Endpunkte aller Elemente geschnitten werden. Zusätzlich zu den anderen Eigenschaften, die von der übergeordneten Objektklasse "ST_Geometry" übernommen werden, besitzen "ST_MultiLineStrings" eine Länge. Mit "ST_MultiLineStrings" werden nicht zusammenhängende lineare Features wie Flüsse oder Straßennetze definiert.

Mit den folgenden Funktionen können Multilinestrings erstellt werden:

- ST_MultiLineString
- ST_MLineFromText (nur Oracle und SQLite)
- ST_MLineFromWKB
- ST_MultiCurve (nur Oracle)

ST_Polygon

Ein ST_Polygon ist eine zweidimensionale Oberfläche, die als Abfolge von Punkten gespeichert ist und den umfassenden äußeren Ring sowie 0 oder mehr innere Ringe definiert. "ST_Polygons" besitzen stets eine einfache Geometrie. "ST_Polygons" definieren Features mit einer räumlichen Ausdehnung wie Flurstücke, Gewässer und Gerichtsbezirke.

In der folgenden Abbildung sind Beispiele für "ST_Polygon"-Objekte dargestellt: 1 ist ein "ST_Polygon", dessen Grenze durch einen äußeren Ring definiert ist. 2 ist ein "ST_Polygon", dessen Grenze durch einen äußeren Ring und zwei innere Ringe definiert ist. Die Fläche innerhalb der inneren Ringe ist Teil des "ST_Polygon"-Außenbereichs. 3 ist ein zulässiges "ST_Polygon", weil sich die Ringe an einem einzelnen Tangentenpunkt schneiden.

Der äußere und alle inneren Ringe definieren die Grenze eines "ST_Polygon". Der Raum, der zwischen den Ringen eingeschlossen ist, definiert den Innenbereich des "ST_Polygon". Die Ringe eines "ST_Polygon" dürfen sich an einem Tangentenpunkt schneiden, jedoch nicht kreuzen. Zusätzlich zu den anderen Eigenschaften, die von der übergeordneten Objektklasse "ST_Geometry" übernommen werden, besitzen "ST_Polygons" eine Fläche.

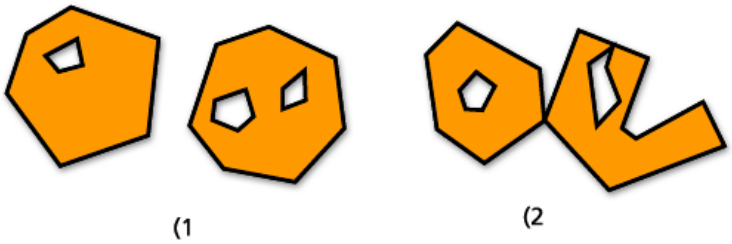
Mit den folgenden Funktionen können Polygone erstellt werden:

- ST_Polygon
- ST_PolyFromText (nur Oracle und SQLite)
- ST_PolyFromWKB
- ST_Surface (nur Oracle und SQLite)

ST_MultiPolygon

Die Grenze eines ST_MultiPolygon ist die kumulative Länge der äußeren und inneren Ringe seiner Elemente. Der Innenbereich eines "ST_MultiPolygon" ist als die kumulativen Innenbereiche der zugehörigen "ST_Polygon"-Elemente definiert. Die Grenzen der Elemente eines "ST_MultiPolygon" dürfen sich nur an einem Tangentenpunkt schneiden. Zusätzlich zu den anderen Eigenschaften, die von der übergeordneten Objektklasse "ST_Geometry" übernommen werden, besitzen "ST_MultiPolygons" eine Fläche. Mit "ST_MultiPolygons" werden Features wie Waldstrata und nicht durchgängige Flurstücke (z. B. Inselketten im Pazifik) definiert.

Die folgende Grafik zeigt Beispiele von "ST_MultiPolygons": 1 ist ein "ST_MultiPolygon" mit zwei "ST_Polygon"-Elementen. Die Grenze wird durch die zwei äußeren Ringe und die drei inneren Ringe definiert. 2 auch ist ein "ST_MultiPolygon" mit zwei "ST_Polygon"-Elementen, wobei die Grenze jedoch durch die zwei äußeren Ringe und die zwei inneren Ringe definiert ist und die zwei "ST_Polygon"-Elemente sich an einem Tangentenpunkt schneiden.



Mit den folgenden Funktionen können Multipolygone erstellt werden:

- ST_MultiPolygon
- ST_MPolyFromText (nur Oracle und SQLite)
- ST_MPolyFromWKB
- ST_MultiSurface (nur Oracle)

Erstellen von Geometrien aus vorhandenen Geometrien

Die folgenden Funktionen sind zwar keine reinen Konstruktorfunktionen, geben jedoch eine neue Geometrie zurück, indem sie vorhandene Geometrien als Eingabe verwenden und dafür Analysen durchführen:

- ST_Aggr_ConvexHull (nur Oracle und SQLite)
- ST_Aggr_Intersection (nur Oracle und 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

Räumliche Accessor-Funktionen für "ST_Geometry"

Räumliche Accessor-Funktionen geben die Eigenschaft einer Geometrie zurück. Es gibt Accessor-Funktionen zum Bestimmen der folgenden Eigenschaften eines ST_Geometry-Features:

Dimensionalität

Die Dimensionen einer Geometrie sind die erforderlichen Mindestkoordinaten (keine, X, Y), um die räumliche Ausdehnung der Geometrie zu definieren.

Eine Geometrie kann die Dimension 0, 1 oder 2 aufweisen.

Die Dimensionen im Einzelnen sind:

- 0 – Weist weder eine Länge noch eine Fläche auf
- 1 – Hat eine Länge (X oder Y)
- 2 – Hat eine Fläche (X und Y)

Die Subtypes "Point" und "Multipoint" besitzen die Dimension 0. "Points" stellen nulldimensionale Features dar, die mit einer einzigen Koordinate modelliert werden können, während "Multipoints" Daten darstellen, die mit einem Cluster nicht verbundener Koordinaten modelliert werden müssen.

Die Subtypes "Linestring" und "Multilinestring" besitzen die Dimension 1. Sie speichern Features wie Straßenabschnitte, verzweigte Flusssysteme und andere Features, die im Normalfall linear sind.

Die Subtypes "Polygon" und "Multipolygon" besitzen die Dimension 2. Waldbestände, Flurstücke, Gewässer und andere Features, deren Umfang eine definierbare Fläche umschließt, können vom Datentyp "Polygon" oder "Multipolygon" gerendert werden.

Die Dimension ist nicht nur als Eigenschaft des Subtypes wichtig, sondern auch bei der Bestimmung der räumlichen Beziehung von zwei Features. Die Dimension des bzw. der entstehenden Features legt fest, ob der Vorgang erfolgreich war. Räumliche Accessor-Funktionen untersuchen die Dimensionen der Features, damit die Vergleichsmethode festgelegt werden kann.

Verwenden Sie die Funktion ST_Dimension, um die Dimension einer Geometrie zu bewerten. Diese Funktion gibt die Dimension eines Features "ST_Geometry" als Ganzzahl zurück.

Die Koordinaten einer Geometrie weisen ebenfalls Dimensionen auf. Wenn eine Geometrie lediglich X- und Y-Koordinaten besitzt, lautet die Koordinatendimension 2. Wenn X-, Y- und Z-Koordinaten vorhanden sind, lautet die Koordinatendimension 3. Wenn eine Geometrie X-, Y-, Z- und M-Koordinaten besitzt, lautet die Dimension 4.

Mit der Funktion ST_CoordDim können Sie die in einer Geometrie vorhandenen Koordinatendimensionen bestimmen.

Z-Koordinaten

Manchen Geometrien ist eine Höhe oder Tiefe zugewiesen – eine dritte Dimension. Jeder der Punkte, die die Geometrie eines Features bilden, kann eine optionale Z-Koordinate besitzen, die eine Höhe oder Tiefe relativ zur Erdoberfläche darstellt.

Die Prädikatfunktion ST_Is3d verwendet ST_Geometry als Eingabe und gibt "true" zurück, wenn die Funktion Z-Koordinaten besitzt. Andernfalls gibt sie "false" zurück.

Mit der Funktion ST_Z können Sie die Z-Koordinate eines Punktes bestimmen.

Die Funktion ST_MaxZ gibt die maximale Z-Koordinate und die Funktion ST_MinZ die minimale Z-Koordinate einer Geometrie zurück.

Messwerte

Messwerte sind Werte, die den einzelnen Koordinaten zugewiesen sind. Sie werden in Anwendungen zur linearen Referenzierung und dynamischen Segmentierung verwendet. Die Kilometerschilder an einer Autobahn können beispielsweise Messwerte enthalten, die ihre Position angeben. Der Wert kann für alles stehen, was als Zahl mit doppelter Genauigkeit gespeichert werden kann.

Die Prädikatfunktion ST_IsMeasured gibt für eine Geometrie den Wert "true" zurück, wenn Messwerte enthalten sind, und den Wert "false", wenn keine Messwerte enthalten sind. Diese Funktion wird nur mit den Oracle- und SQLite-Implementierungen von ST_Geometry verwendet.

Mit der Funktion ST_M können Sie den Messwert eines Punktes ermitteln.

Die Funktion ST_MaxM gibt die maximale Z-Koordinate und die Funktion ST_MinM die minimale Z-Koordinate einer Geometrie zurück.

Geometrie-Typ

Der Geometrietyp verweist auf den Typ der geometrischen Entität. Zu diesen zählen folgende:

- Punkte und Multipoints
- Linien und Multilines
- Polygone und Multipolygone

ST_Geometry ist eine übergeordnete Objektklasse, die verschiedene Subtypes speichern kann. Verwenden Sie die Funktion ST_GeometryType oder ST_Entity (nur Oracle und SQLite), um den Subtype einer Geometrie zu bestimmen.

Punktsammlung (Stützpunkte) und Anzahl der Punkte

Eine Geometrie kann null oder mehr Punkte aufweisen. Eine Geometrie gilt als leer, wenn sie null Punkte enthält. Der Subtype "Point" ist die einzige Geometrie, die auf null oder einen Punkt beschränkt ist. Alle anderen Subtypes können null oder mehr Punkte besitzen.

ST_Point

Ein "ST_Point" ist eine nulldimensionale Geometrie, die eine einzelne Position in einem Koordinatenbereich einnimmt. Ein "ST_Point" besitzt einen einzigen XY-Koordinatenwert, ist immer eine einfache Geometrie und weist eine NULL-Grenze auf. "ST_Point" kann zum Definieren von Features wie Ölquellen, Landmarks und Sammelstellen für Wasserproben verwendet werden.

Nachfolgend sind einige Funktionen genannt, die ausschließlich für den Datentyp "ST_Point" verwendet werden können:

- ST_X: Gibt den X-Koordinatenwert eines "Point"-Datentyps als Zahl mit doppelter Genauigkeit zurück.
- ST_Y: Gibt den Y-Koordinatenwert eines "Point"-Datentyps als Zahl mit doppelter Genauigkeit zurück.
- ST_Z: Gibt den Z-Koordinatenwert eines "Point"-Datentyps als Zahl mit doppelter Genauigkeit zurück.
- ST_M: Gibt den M-Koordinatenwert eines "Point"-Datentyps als Zahl mit doppelter Genauigkeit zurück.

ST_MultiPoint

Ein "ST_Multipoint" ist eine Sammlung von "ST_Points" und besitzt wie seine Elemente die Dimension 0. Ein "ST_MultiPoint" ist einfach, wenn keines seiner Elemente den gleichen Koordinatenbereich einnimmt. Die Grenze eines "ST_MultiPoint" beträgt NULL. "ST_MultiPoints" können Dinge wie Übertragungsmuster und Orte von Krankheitsausbrüchen definieren.

Mit der Funktion ST_NumGeometries können Sie die Anzahl der Punkte in einer Multipoint-Geometrie ermitteln.

Länge, Fläche und Umfang

Länge, Fläche und Umfang sind messbare Eigenschaften von Geometrien. Linestrings und die Elemente von Multilinestrings sind eindimensional und besitzen die Eigenschaft einer Länge. Polygone und die Elemente von Multipolygonen sind zweidimensionale Oberflächen und besitzen demnach eine messbare Fläche und einen messbaren Umfang. Sie können die Funktionen ST_Length, ST_Area und ST_Perimeter verwenden, um diese Eigenschaften zu bestimmen. Maßeinheiten variieren abhängig davon, wie die Daten gespeichert sind.

ST_LineString

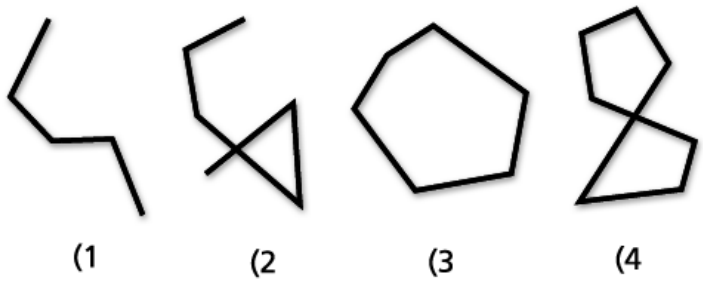
Ein "ST_LineString" ist ein eindimensionales Objekt, das als Abfolge von Punkten gespeichert ist und einen linear interpolierten Pfad definiert. Der "ST_LineString" besitzt eine einfache Geometrie, wenn er seinen Innenbereich nicht schneidet. Die Endpunkte (Grenze) eines geschlossenen "ST_LineString" nehmen denselben Punkt im Raum ein. Ein "ST_LineString" ist ein Ring, wenn er geschlossen ist und eine einfache Geometrie aufweist. Wie die anderen Eigenschaften, die von der übergeordneten Objektklasse "ST_Geometry" übernommen werden, besitzen "ST_LineStrings" eine Länge. Mit "ST_LineStrings" werden häufig lineare Features wie Straßen, Flüsse und Stromleitungen definiert.

Die Endpunkte bilden im Normalfall die Grenze eines "ST_LineString". Wenn der "ST_LineString" geschlossen ist, ist die Grenze jedoch NULL. Der Innenbereich eines "ST_LineString" ist der verbundene Pfad, der zwischen den Endpunkten liegt. Wenn er geschlossen ist, gibt es einen durchgehenden Innenbereich.

Nachfolgend sind einige Funktionen genannt, die für "ST_LineStrings" verwendet werden können:

- ST_StartPoint: Gibt den ersten Punkt für den angegebenen "ST_LineString" zurück.
- ST_EndPoint: Gibt den letzten Punkt für einen "ST_LineString" zurück.
- ST_IsClosed: Eine Prädikatfunktion, die "true" zurückgibt, wenn der angegebene ST_LineString geschlossen ist (Startpunkt und Endpunkt des Linestring schneiden sich), und "false", wenn dies nicht der Fall ist
- ST_IsRing: Eine Prädikatfunktion, die "true" zurückgibt, wenn der angegebene ST_LineString ein Ring ist, und "false", wenn dies nicht der Fall ist
- ST_Length: Gibt die Länge eines "ST_LineString" als Zahl mit doppelter Genauigkeit zurück.
- ST_NumPoints: Bewertet einen "ST_LineString" und gibt die Anzahl der Punkte in seiner Folge als Ganzzahl zurück.
- ST_PointN: Verwendet einen "ST_LineString" und einen Index zum n-ten Punkt und gibt diesen Punkt zurück.

Die folgende Grafik zeigt Beispiele verschiedener "ST_LineString"-Objekte: (1 ist ein einfacher, nicht geschlossener "ST_LineString"; (2 ist ein komplexer, nicht geschlossener "ST_LineString"; (3 ist ein geschlossener, einfacher "ST_LineString", also ein Ring; (4 ist ein geschlossener, komplexer "ST_LineString", aber kein Ring.

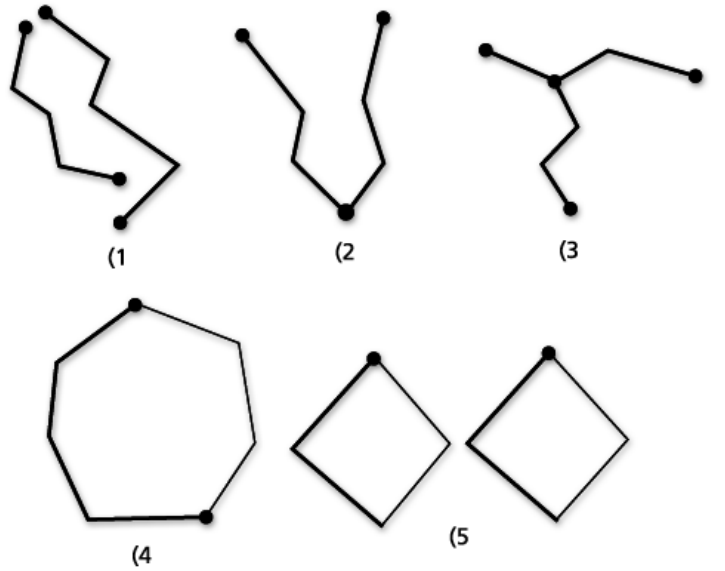


ST_MultiLineString

Ein ST_MultiLineString ist eine Sammlung von ST_LineStrings. "ST_MultiLineStrings" sind einfach, wenn sich nur die Endpunkte der "ST_LineString"-Elemente schneiden. "ST_MultiLineStrings" sind komplex, wenn sich die Innenbereiche der "ST_LineString"-Elemente schneiden.

Die Grenze eines "ST_MultiLineString" bilden die Endpunkte der "ST_LineString"-Elemente, die nicht geschnitten werden. Die Grenze eines "ST_MultiLineString" ist NULL, wenn alle Endpunkte aller Elemente geschnitten werden. Zusätzlich zu den anderen Eigenschaften, die von der übergeordneten Objektklasse "ST_Geometry" übernommen werden, besitzen "ST_MultiLineStrings" eine Länge. Mit "ST_MultiLineStrings" werden nicht zusammenhängende lineare Features wie Flüsse oder Straßennetze definiert.

Die folgende Grafik zeigt Beispiele für "ST_MultiLineStrings": (1 ist ein einfacher "ST_MultiLineString", dessen Grenze die vier Endpunkte seiner zwei "ST_LineString"-Elemente bilden. (2 ist ein einfacher "ST_MultiLineString", weil nur die Endpunkte der "ST_LineString"-Elemente geschnitten werden. Die Grenze bilden zwei Endpunkte, die sich nicht schneiden. (3 ist ein komplexer "ST_MultiLineString", weil der Innenbereich eines der "ST_LineString"-Elemente geschnitten wird. Die Grenze dieses "ST_MultiLineString" bilden die drei nicht geschnittenen Endpunkte. (4) ist ein einfacher, nicht geschlossener "ST_MultiLineString". Er ist nicht geschlossen, weil seine "ST_LineString"-Elemente nicht geschlossen sind. Es handelt sich um eine einfache Geometrie, weil der Innenbereich keines der "ST_LineString"-Elemente geschnitten wird. (5) ist ein einzelner, einfacher, geschlossener "ST_MultiLineString". Er ist geschlossen, weil alle zugehörigen Elemente geschlossen sind. Es handelt sich um eine einfache Geometrie, weil der Innenbereich keines der zugehörigen Elemente geschnitten wird.



Nachfolgend sind einige Funktionen genannt, die für "ST_MultiLineStrings" verwendet werden können:

- ST_IsClosed: Diese Prädikatfunktion gibt "true" zurück, wenn der angegebene "ST_MultiLineString" geschlossen ist, und "false", wenn nicht.
- ST_Length: Diese Funktion wertet einen "ST_MultiLineString" aus und gibt die kumulative Länge aller zugehörigen "ST_LineString"-Elemente als Zahl mit doppelter Genauigkeit aus.
- ST_NumGeometries: Diese Funktion gibt die Anzahl der Linien in einem Multilinestring zurück

ST_Polygon

Ein ST_Polygon ist eine zweidimensionale Oberfläche, die als Abfolge von Punkten gespeichert ist und den umfassenden äußeren Ring sowie 0 oder mehr innere Ringe definiert. "ST_Polygons" besitzen stets eine einfache Geometrie. "ST_Polygons" definieren Features mit einer räumlichen Ausdehnung wie Flurstücke, Gewässer und Gerichtsbezirke.

In der folgenden Abbildung sind Beispiele für "ST_Polygon"-Objekte dargestellt: 1 ist ein "ST_Polygon", dessen Grenze durch einen äußeren Ring definiert ist. 2 ist ein "ST_Polygon", dessen Grenze durch einen äußeren Ring und zwei innere Ringe definiert ist. Die Fläche innerhalb der inneren Ringe ist Teil des "ST_Polygon"-Außenbereichs. 3 ist ein zulässiges "ST_Polygon", weil sich die Ringe an einem einzelnen Tangentenpunkt schneiden.

Der äußere und alle inneren Ringe definieren die Grenze eines "ST_Polygon". Der Raum, der zwischen den Ringen eingeschlossen ist, definiert den Innenbereich des "ST_Polygon". Die Ringe eines "ST_Polygon" dürfen sich an einem Tangentenpunkt schneiden, jedoch nicht kreuzen. Zusätzlich zu den anderen Eigenschaften, die von der übergeordneten Objektklasse "ST_Geometry" übernommen werden, besitzen "ST_Polygons" eine Fläche.

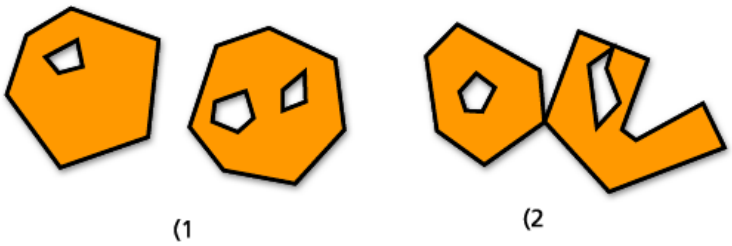
Nachfolgend sind einige Funktionen genannt, die für ein "ST_Polygon" verwendet werden können:

- ST_Area: Gibt die Fläche eines "ST_Polygon" als Zahl mit doppelter Genauigkeit zurück.
- ST_Centroid: Gibt einen "ST_Point" zurück, der den Mittelpunkt des "ST_Polygon"-Envelopes darstellt.
- ST_ExteriorRing: Gibt den äußeren Ring eines "ST_Polygon" als "ST_LineString" zurück.
- ST_InteriorRingN: Wertet ein "ST_Polygon" und einen Index aus und gibt den n-ten inneren Ring als "ST_LineString" zurück.
- ST_NumInteriorRing: Gibt eine Anzahl der inneren Ringe eines "ST_Polygon" zurück.
- ST_PointOnSurface: Gibt einen "ST_Point" zurück, der sicher auf der Oberfläche des angegebenen "ST_Polygon" liegt.

ST_MultiPolygon

Die Grenze eines ST_MultiPolygon ist die kumulative Länge der äußeren und inneren Ringe seiner Elemente. Der Innenbereich eines "ST_MultiPolygon" ist als die kumulativen Innenbereiche der zugehörigen "ST_Polygon"-Elemente definiert. Die Grenzen der Elemente eines "ST_MultiPolygon" dürfen sich nur an einem Tangentenpunkt schneiden. Zusätzlich zu den anderen Eigenschaften, die von der übergeordneten Objektklasse "ST_Geometry" übernommen werden, besitzen "ST_MultiPolygons" eine Fläche. Mit "ST_MultiPolygons" werden Features wie Waldstrata und nicht durchgängige Flurstücke (z. B. Inselketten im Pazifik) definiert.

Die folgende Grafik zeigt Beispiele von "ST_MultiPolygons": 1 ist ein "ST_MultiPolygon" mit zwei "ST_Polygon"-Elementen. Die Grenze wird durch die zwei äußeren Ringe und die drei inneren Ringe definiert. 2 auch ist ein "ST_MultiPolygon" mit zwei "ST_Polygon"-Elementen, wobei die Grenze jedoch durch die zwei äußeren Ringe und die zwei inneren Ringe definiert ist und die zwei "ST_Polygon"-Elemente sich an einem Tangentenpunkt schneiden.



Nachfolgend sind einige Funktionen genannt, die für "ST_MultiPolygons" verwendet werden können:

- ST_Area: Gibt eine Zahl mit doppelter Genauigkeit zurück, die den kumulativen "ST_Area"-Werten der "ST_Polygon"-Elemente eines "ST_MultiPolygon" entspricht.
- ST_Centroid: Gibt einen "ST_Point" zurück, der den Mittelpunkt eines "ST_MultiPolygon"-Envelopes darstellt.
- ST_NumGeometries: Gibt die Anzahl der Polygone in einem Multipolygon zurück.
- ST_PointOnSurface: Wertet ein "ST_MultiPolygon" aus und gibt einen "ST_Point" zurück, der sicher auf der Oberfläche eines der zugehörigen "ST_Polygon"-Elemente liegt.

Einfache Geometrien in einer Multipart-Geometrie

Multipart-Geometrien setzen sich aus einzelnen, einfachen Geometrien zusammen.

Sie können ermitteln, wie viele einzelne Geometrien in einer Multipart-Geometrie wie "ST_MultiPoint", "ST_MultiLineString" und "ST_MultiPolygon" enthalten sind. Verwenden Sie dazu die Prädikatfunktion

ST_NumGeometries. Diese Funktion gibt die Anzahl der einzelnen Elemente in einer Sammlung von Geometrien zurück.

Mit der Funktion ST_GeometryN können Sie ermitteln, welche Geometrie in der Multipart-Geometrie an Position N vorhanden ist. Dabei ist N eine Zahl, die Sie mit der Funktion angeben. Beispiel: Wenn Sie den dritten Punkt einer Multipoint-Geometrie zurückgeben möchten, geben Sie bei der Ausführung der Funktion den Wert "3" an.

Um die einzelnen Geometrien und ihre Position aus einer Multipart-Geometrie in PostgreSQL zurückzugeben, verwenden Sie die Funktion ST_GeomFromCollection.

Innenbereich, Außenbereich oder Grenze

Alle Geometrien belegen eine Position im Raum, der durch ihren Innenbereich, ihre Grenze und ihren Außenbereich definiert ist. Der Außenbereich einer Geometrie ist der Raum, der nicht von der Geometrie in Anspruch genommen wird. Der Innenbereich ist der Raum, der von der Geometrie in Anspruch genommen wird. Die Grenze einer Geometrie ist der Bereich zwischen dem Innen- und Außenbereich. Die Eigenschaften für den Innen- und Außenbereich werden vom Subtype direkt übernommen, die Eigenschaft für die Grenze variiert jedoch.

Mit der Funktion ST_Boundary ermitteln Sie die Grenze des "ST_Geometry"-Quellobjekts.

Einfach oder komplex

Einige Subtypes von "ST_Geometry" sind grundsätzlich einfach, wie "ST_Points" oder "ST_Polygons". Die Subtypes "ST_LineStrings", "ST_MultiPoints" und "ST_MultiLineStrings" können jedoch einfach oder komplex sein. Sie sind einfach, wenn sie allen ihnen auferlegten Topologieregeln folgen, bzw. komplex, wenn sie ihnen nicht folgen.

Nachfolgend werden einige Topologieregeln beschrieben:

- Ein "ST_LineString" besitzt eine einfache Geometrie, wenn er seinen Innenbereich nicht schneidet. Andernfalls ist die Geometrie komplex.
- Ein "ST_MultiPoint" besitzt eine einfache Geometrie, wenn zwei der Elemente nicht denselben Koordinatenbereich einnehmen(d. h. dieselben XY-Koordinaten besitzen). Andernfalls ist die Geometrie komplex.
- Ein "ST_MultiLineString" besitzt eine einfache Geometrie, wenn sich die Innenbereiche seiner Elemente nicht selbst schneiden. Wenn sich die Innenbereiche der Elemente schneiden, ist die Geometrie komplex.

Mit der Prädikatfunktion ST_IsSimple wird ermittelt, ob ein "ST_LineString", "ST_MultiPoint" oder "ST_MultiLineString" einfach oder komplex ist. "ST_IsSimple" gibt für eine "ST_Geometry" den Wert "true" zurück, wenn die Geometrie einfach ist. Andernfalls wird der Wert "false" zurückgegeben.

Leer oder nicht leer

Eine Geometrie ist leer, wenn sie keine Punkte besitzt. Bei einer leeren Geometrie sind Envelope, Grenze, Innenbereich und Außenbereich null. Eine leere Geometrie ist grundsätzlich einfach. Leere Linestrings und Multilinestrings besitzen die Länge 0. Leere Polygone und Multipolygone besitzen eine 0-Fläche.

Mit der Prädikatfunktion ST_IsEmpty kann ermittelt werden, ob eine Geometrie leer ist. Diese Funktion analysiert eine "ST_Geometry" und gibt den Wert "true" zurück, wenn die "ST_Geometry" leer ist. Andernfalls wird der Wert "false" zurückgegeben.

"IsClosed" und "IsRing"

Linestring-Geometrien können geschlossen oder Ringe sein. Linestrings können geschlossen sein, ohne gleichzeitig Ringe zu sein. Mit der Prädikatfunktion ST_IsClosed können Sie ermitteln, ob ein Linestring geschlossen ist. "True"

wird zurückgegeben, wenn sich Start- und Endpunkt des Linestrings schneiden. Ringe sind geschlossene Linestrings mit einfacher Geometrie. Mit der Prädikatfunktion ST_IsRing kann getestet werden, ob ein Linestring wirklich ein Ring ist. "True" wird zurückgegeben, wenn der Linestring geschlossen ist und eine einfache Geometrie besitzt.

Envelope

Jede Geometrie besitzt einen Envelope. Der Envelope einer Geometrie ist die umgebende Geometrie, die durch die minimalen und maximalen XY-Koordinaten gebildet wird. Da bei Punktgeometrien die minimalen und maximalen XY-Koordinaten identisch sind, wird ein Rechteck oder Envelope um diese Koordinaten gebildet. Bei Liniengeometrien stellen die Endpunkte der Linie zwei Seiten des Envelopes dar. Die anderen zwei Seiten befinden sich unmittelbar über und unter der Linie.

Die Funktion ST_Envelope gibt für eine "ST_Geometry" eine "ST_Geometry" zurück, die den Envelope des "ST_Geometry"-Quellobjekts darstellt.

Um die einzelnen minimalen und maximalen XY-Koordinaten einer Geometrie zu ermitteln, verwenden Sie die Funktionen ST_MinX, ST_MinY, ST_MaxX, und ST_MaxY.

Raumbezugssystem

Das Raumbezugssystem gibt die Koordinatentransformationsmatrix für jede Geometrie an. Diese besteht aus Koordinatensystem, Auflösung und Toleranz.

Alle Raumbezugssysteme für die Geodatabase werden in einer Geodatabase-Systemtabelle gespeichert.

Mit den folgenden Funktionen werden Informationen zu Raumbezugssystemen von Geometrien abgerufen:

- ST_SRID: Gibt für "ST_Geometry" die Raumbezugskennung (SRID, Spatial Reference IDentifier) als Ganzzahl zurück.
- ST_Equalsrs: Ermittelt, ob die Raumbezugssysteme von zwei verschiedenen Feature-Classes identisch (true) sind oder nicht (false).

Größe der Features (nur PostgreSQL)

Die Features (räumliche Datensätze in einer Tabelle) beanspruchen eine bestimmte Menge an Speicherplatz in Byte. Mit der Funktion ST_GeoSize können Sie ermitteln, wie groß die einzelnen Features in einer Tabelle sind.

Text- und Binärdefinition einer Geometrie

Verwenden Sie die Funktion ST_AsText bzw. ST_AsBinary, um die WKT-Definition (Well-Known Text) bzw. die WKB-Definition (Well-Known Binary) in einer bestimmten Zeile in der räumlichen Tabelle abzurufen.

Räumliche Beziehungen

Eine der Hauptaufgaben eines GIS besteht darin, die räumlichen Beziehungen zwischen Features zu ermitteln: Überlappen sie? Ist eines im anderen enthalten? Überschneidet sich das eine mit dem anderen?

Zwischen Geometrien können unterschiedliche Arten von Beziehungen bestehen. In den folgenden Beispielen wird beschrieben, welche räumlichen Beziehungen zwischen zwei Geometrien bestehen können:

- Geometrie A verläuft durch Geometrie B.
- Geometrie A ist vollständig in Geometrie B enthalten.
- Geometrie B ist vollständig in Geometrie A enthalten.
- Die Geometrien überschneiden oder berühren sich nicht.
- Die Geometrien sind vollständig lagegleich.
- Die Geometrien überlappen einander.
- Die Geometrien berühren sich an einem Punkt.

Wenn Sie ermitteln möchten, ob diese Beziehungen bestehen, verwenden Sie Funktionen für räumliche Beziehungen. Mit diesen Funktionen werden die folgenden Eigenschaften der von Ihnen in einer Abfrage angegebenen Geometrien verglichen:

- Dem Außenbereich (Exterior - E) der Geometrien: Der Außenbereich ist der gesamte Raum, der nicht von einer Geometrie eingenommen wird.
- Dem Innenbereich (Interior - I) der Geometrien: Der Innenbereich ist der von einer Geometrie eingenommene Raum.
- Der Grenze (Boundary - B) der Geometrien: Die Grenze ist die Schnittstelle zwischen dem Innenbereich und dem Außenbereich einer Geometrie.

Beim Erstellen einer Abfrage für räumliche Beziehungen geben Sie die Art der gesuchten räumlichen Beziehung und die Geometrien an, die verglichen werden sollen: Die Abfragen geben entweder "true" oder "false" zurück. Das bedeutet, dass die Geometrien entweder in der angegebenen räumlichen Beziehung zueinander stehen oder eben nicht. Eine Abfrage für räumliche Beziehungen wird meist verwendet, um Ergebnisse durch Einfügen in eine `WHERE`-Klausel zu filtern.

Wenn Sie beispielsweise über eine Tabelle verfügen, in der die Standorte von vorgeschlagenen Erschließungsgebieten gespeichert sind, und eine weitere Tabelle, in der die Standorte archäologisch bedeutender Fundstätten gespeichert sind, können Sie eine Abfrage durchführen, mit der Sie sicherstellen können, dass keines der Erschließungsgebiete die archäologischen Fundstatten überschneidet, und wenn doch, dass dann die ID dieser vorgeschlagenen Erschließungsgebiete zurückgegeben wird. In diesem Beispiel wurde die ST_Disjoint-Funktion in PostgreSQL verwendet.

```
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
```

Diese Abfrage gibt den Namen des Erschließungsgebiets (projname) und die ID der archäologischen Fundstätte (siteid) zurück, die nicht getrennt sind, der Standorte, die einander überschneiden. Sie gibt ein Erschließungsprojekt (Bow Wow Chow) zurück, das die archäologische Fundstätte A1009 überschneidet.

Relationale Funktionen für ST_Geometry

Relationale Funktionen verwenden Prädikate, um zu prüfen, ob verschiedene Arten von räumlichen Beziehungen vorliegen. Mit den Prüfungen wird dieses Ziel erreicht, indem die Beziehungen zwischen Folgendem verglichen werden:

- Dem Außenbereich (Exterior - E) der Geometrien: Der Außenbereich ist der gesamte Raum, der nicht von einer Geometrie eingenommen wird.
- Dem Innenbereich (Interior - I) der Geometrien: Der Innenbereich ist der von einer Geometrie eingenommene Raum.
- Der Grenze (Boundary - B) der Geometrien: Die Grenze ist die Schnittstelle zwischen dem Innenbereich und dem Außenbereich einer Geometrie.

Beziehungen werden mit Prädikaten geprüft. Dabei wird 1 (Oracle und SQLite) oder t (PostgreSQL) zurückgegeben, wenn bei einem Vergleich die Kriterien der Funktion erfüllt sind. Andernfalls wird 0 (Oracle und SQLite) oder f (PostgreSQL) zurückgegeben. Mit Prädikaten, mit denen geprüft wird, ob eine räumliche Beziehung vorliegt, werden Geometriepaare verglichen, bei denen es sich um unterschiedliche Typen oder Dimensionen handelt.

Mit Prädikaten werden die X- und Y-Koordinaten der übermittelten Geometrien verglichen. Die Z-Koordinaten und Messwerte werden, falls vorhanden, ignoriert. Geometrien mit Z-Koordinaten oder Messwerten können mit Geometrien ohne Z-Koordinaten oder Messwerte verglichen werden.

Das von Clementini, et al. entwickelte Dimensionally Extended 9 Intersection Model (DE-9IM) ist eine dimensionale Erweiterung des 9 Intersection Model von Egenhofer und Herring. DE-9IM ist ein mathematischer Ansatz, mit dem die paarweisen räumlichen Beziehungen zwischen Geometrien unterschiedlichen Typs und unterschiedlicher Dimension definiert werden. Bei diesem Modell werden räumliche Beziehungen zwischen allen Geometrietypen als paarweise Schnittmenge ihres Innenbereichs, ihrer Grenze und ihres Außenbereichs ausgedrückt, wobei die Dimension der resultierenden Schnittmengen berücksichtigt wird.

Bei gegebenen Geometrien a und b stellen I(a), B(a) und E(a) den Innenbereich, die Grenze und den Außenbereich von a dar und I(b), B(b) und E(b) den Innenbereich, die Grenze und den Außenbereich von b. Die Schnittmengen von I(a), B(a) und E(a) mit I(b), B(b) und E(b) ergeben eine Matrix von 3 x 3. Jede Schnittmenge kann Geometrien unterschiedlicher Dimensionen ergeben. So kann beispielsweise die Schnittmenge der Grenzen zweier Polygone aus einem Punkt und einem Linestring bestehen. Ist dies der Fall, gibt die dim-Funktion (Dimension) die maximale Dimension 1 zurück.

Die dim-Funktion gibt den Wert –1, 0, 1 oder 2 zurück. Dabei entspricht –1 der Nullmenge, die zurückgegeben wird, wenn keine Schnittmenge gefunden wird, oder dim(Ã*ƒ*˜).

	Innenbereich	**Grenze**	**Außenbereich**
Innenbereich	dim(I(a) schneidet I(b))	dim(I(a) schneidet B(b))	dim(I(a) schneidet E(b))
Grenze	dim(B(a) schneidet I(b))	dim(B(a) schneidet B(b))	dim(B(a) schneidet E(b))
Außenbereich	dim(E(a) schneidet I(b))	dim(E(a) schneidet B(b))	dim(E(a) schneidet E(b))

Beispiel für eine Schnittmenge eines Prädikats

Die Ergebnisse der Prädikate für räumliche Beziehungen können durch den Vergleich der Ergebnisse des Prädikats mit einer Mustermatrix, die die zulässigen Werte für das DE-9IM darstellt, verstanden oder überprüft werden.

Die Mustermatrix enthält die zulässigen Werte für die einzelnen Matrixzellen der Schnittmenge. Folgende

Wertemuster sind möglich:

T: Es muss eine Schnittmenge vorhanden sein; dim = 0, 1 oder 2

F: Es darf keine Schnittmenge vorhanden sein; dim = –1

*: Es spielt keine Rolle, ob eine Schnittmenge vorhanden ist; dim = –1, 0, 1 oder 2

0: Es muss eine Schnittmenge vorhanden sein, und die entsprechende maximale Dimension muss 0 betragen; dim = 0

1: Es muss eine Schnittmenge vorhanden sein, und die entsprechende maximale Dimension muss 1 betragen; dim = 1

2: Es muss eine Schnittmenge vorhanden sein, und die entsprechende maximale Dimension muss 2 betragen; dim = 2

Für jedes Prädikat gibt es mindestens eine Mustermatrix. Für einige sind jedoch mehrere Mustermatrizen erforderlich, um die Beziehungen zwischen verschiedenen Geometrietypkombinationen zu beschreiben.

Die Mustermatrix des ST_Within-Prädikats für Geometriekombinationen weist das folgende Format auf:

			Geometrie b	
		Innenbereich	**Grenze**	**Außenbereich**
Geometrie a	**Innenbereich**	T	*	F
	Grenze	*	*	F
	Außenbereich	*	*	*

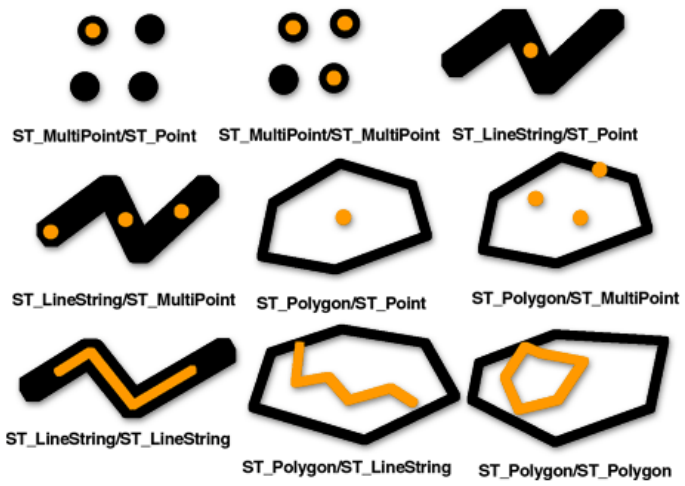
Beispiel für eine Mustermatrix

Das ST_Within-Prädikat gibt "true", wenn sich die Innenbereiche der beiden Geometrien schneiden und der Innenbereich und die Grenze der Geometrie a den Außenbereich der Geometrie b nicht schneiden. Alle anderen Bedingungen spielen keine Rolle.

In den folgenden Abschnitten werden die für räumliche Beziehungen verwendeten Prädikate beschrieben. In den Abbildungen in diesen Abschnitten ist die erste Eingabe-Geometrie in Schwarz und die zweite in Orange dargestellt.

ST_Contains

ST_Contains gibt 1 oder t (true) zurück, wenn die zweite Geometrie vollständig in der ersten Geometrie enthalten ist. Das ST_Contains-Prädikat gibt genau das gegenteilige Ergebnis des ST_Within-Prädikats zurück.

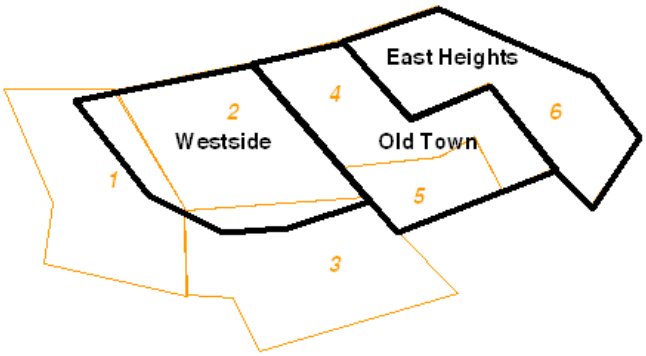


Die Mustermatrix des ST_Contains-Prädikats besagt, dass sich die Innenbereiche der beiden Geometrien schneiden müssen und dass der Innenbereich und die Grenze der zweiten Geometrie (Geometrie b) den Außenbereich der ersten Geometrie (Geometrie a) nicht schneiden dürfen.

			Geometrie b	
		Innenbereich	**Grenze**	**Außenbereich**
Geometrie a	**Innenbereich**	T	*	*
	Grenze	*	*	*
	Außenbereich	F	F	*

ST_Contains-Matrix

Mit den Funktionen ST_Within und ST_Contains werden nur die Geometrien ermittelt, die sich vollständig in einer anderen Geometrien befinden. So können Features aus der Auswahl entfernt werden, durch die die Ergebnisse verfälscht werden könnten. Im folgenden Beispiel möchte ein mobiler Eisverkäufer herausfinden, in welchen Stadtteilen sich die meisten Kinder (potenzielle Kunden) befinden, und er möchte seine Route auf diese Gebiete beschränken. Der Verkäufer vergleicht Polygone der ausgewiesenen Stadtteile mit Zählbezirken, die ein Attribut für die Gesamtzahl der Kinder unter 16 Jahren aufweisen.

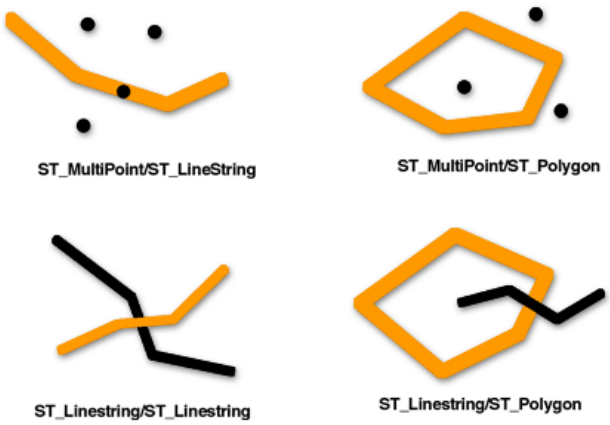


Wenn nicht alle Kinder, die in den Zählbezirken 1 und 3 leben, in den Landsplittern leben, die in Westside liegen, könnte sich die Anzahl der Kinder im Stadtteil Westside durch Einbeziehen dieser Zählbezirke in die Auswahl fälschlicherweise erhöhen. Wenn angegeben wird, dass nur Zählbezirke, die vollständig innerhalb der Stadtteile liegen, einbezogen werden sollen (ST_Within = 1), spart der Eisverkäufer möglicherweise Zeit und Geld, indem er

sich nicht in diese Teile von Westside begibt.

ST_Crosses

ST_Crosses gibt 1 oder t (true) zurück, wenn die Schnittmenge eine Geometrie ergibt, deren Dimension um eins kleiner ist als die maximale Dimension der beiden Quellgeometrien und die Schnittmenge innerhalb der beiden Quellgeometrien liegt. ST_Crosses gibt 1 oder t (true) nur bei ST_MultiPoint/ST_Polygon-, ST_MultiPoint/ST_LineString-, ST_Linestring/ST_LineString-, ST_LineString/ST_Polygon- und ST_LineString/ST_MultiPolygon-Vergleichen zurück.



Die folgende Mustermatrix des ST_Crosses-Prädikats gilt für ST_MultiPoint/ST_LineString, ST_MultiPoint/ST_MultiLineString, ST_MultiPoint/ST_Polygon, ST_MultiPoint/ST_MultiPolygon, ST_LineString/ST_Polygon und ST_LineString/ST_MultiPolygon. Die Matrix besagt, dass sich die Innenbereiche schneiden müssen und dass zumindest der Innenbereich der ersten Geometrie (Geometrie a) den Außenbereich der zweiten Geometrie (Geometrie b) schneiden muss.

			Geometrie b	
		Innenbereich	**Grenze**	**Außenbereich**
Geometrie a	**Innenbereich**	T	*	T
	Grenze	*	*	*
	Außenbereich	*	*	*

ST_Crosses-Matrix 1

Das folgende ST_Crosses-Prädikat gilt für ST_LineString/ST_LineString, ST_LineString/ST_MultiLineString und ST_MultiLineString/ST_MultiLineString. Die Matrix besagt, dass die Dimension der Schnittmenge der Innenbereiche 0 (Schnittmenge an einem Punkt) sein muss. Wenn die Dimension dieser Schnittmenge 1 (Schnittmenge an einem Linestring) beträgt, gibt das ST_Crosses-Prädikat "false" zurück. Das ST_Overlaps-Prädikat gibt dagegen "true" zurück.

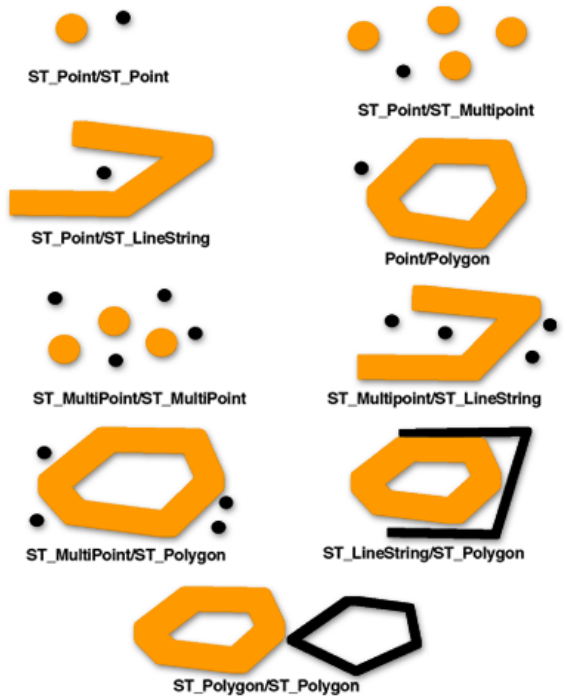
			Geometrie b	
		Innenbereich	**Grenze**	**Außenbereich**
Geometrie a	**Innenbereich**	0	*	*
	Grenze	*	*	*

ST_Crosses-Matrix 2

	Außenbereich	*	*	*

ST_Disjoint

ST_Disjoint gibt 1 oder t (true) zurück, wenn die Schnittmenge der beiden Geometrien leer ist. Das bedeutet, dass Geometrien getrennt sind, wenn sie einander nicht schneiden.



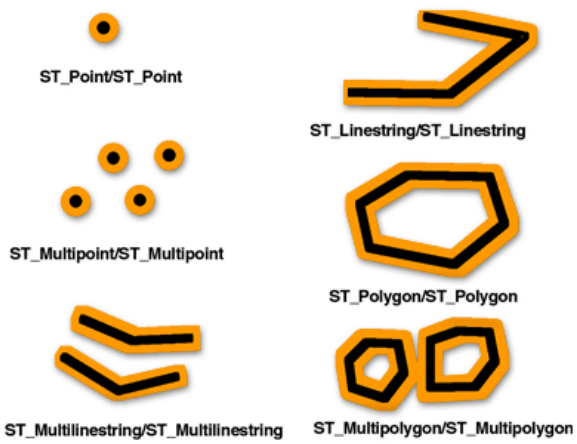
Die Mustermatrix des ST_Disjoint-Prädikats besagt, dass sich weder die Innenbereiche noch die Grenzen der beiden Geometrien schneiden.

			Geometrie b	
		Innenbereich	**Grenze**	**Außenbereich**
Geometrie a	**Innenbereich**	**F**	**F**	*****
	Grenze	**F**	**F**	*****
	Außenbereich	*	*	*

ST_Disjoint-Matrix

ST_Equals

ST_Equals gibt 1 oder t (true) zurück, wenn zwei Geometrien desselben Typs identische XY-Koordinatenwerte aufweisen. Die erste und zweite Etage eines Bürogebäudes können beispielsweise identische XY-Koordinaten aufweisen und daher gleich sein. Mit ST_Equals lässt sich auch feststellen, ob zwei Features fälschlicherweise aufeinander gelegt wurden.



Mit der DE-9IM-Mustermatrix für Gleichheit wird sichergestellt, dass sich die Innenbereiche schneiden und dass kein Teil des Innenbereichs oder der Grenze der einen Geometrie den Außenbereich der anderen schneidet.

			Geometrie b	
		Innenbereich	**Grenze**	**Außenbereich**
Geometrie a	**Innenbereich**	T	*	F
	Grenze	*	*	F
	Außenbereich	F	F	*

ST_Equals-Matrix

ST_Intersects

ST_Intersects gibt 1 oder t (true) zurück, wenn die Schnittmenge nicht leer ist. ST_Intersects gibt genau das gegenteilige Ergebnis von ST_Disjoint zurück.

Das ST_Intersects-Prädikat gibt "true" zurück, wenn aufgrund der Bedingungen einer der folgenden Mustermatrizen "true" zurückgegeben wird.

Das ST_Intersects-Prädikat gibt "true" zurück, wenn sich die Innenbereiche beider Geometrien schneiden.

			Geometrie b	
		Innenbereich	**Grenze**	**Außenbereich**
Geometrie a	**Innenbereich**	T	*	*
	Grenze	*	*	*
	Außenbereich	*	*	*

ST_Intersects-Matrix 1

Das ST_Intersects-Prädikat gibt "true" zurück, wenn der Innenbereich der ersten Geometrie die Grenze der zweiten Geometrie schneidet.

			Geometrie b	
		Innenbereich	**Grenze**	**Außenbereich**

ST_Intersects-Matrix 2

Geometrie a	**Innenbereich**	*	T	*
	Grenze	*	*	*
	Außenbereich	*	*	*

Das ST_Intersects-Prädikat gibt "true" zurück, wenn die Grenze der ersten Geometrie den Innenbereich der zweiten Geometrie schneidet.

			Geometrie b	
		Innenbereich	**Grenze**	**Außenbereich**
Geometrie a	**Innenbereich**	*	*	*
	Grenze	T	*	*
	Außenbereich	*	*	*

ST_Intersects-Matrix 3

Das ST_Intersects-Prädikat gibt "true" zurück, wenn sich die Grenzen der beiden Geometrien schneiden.

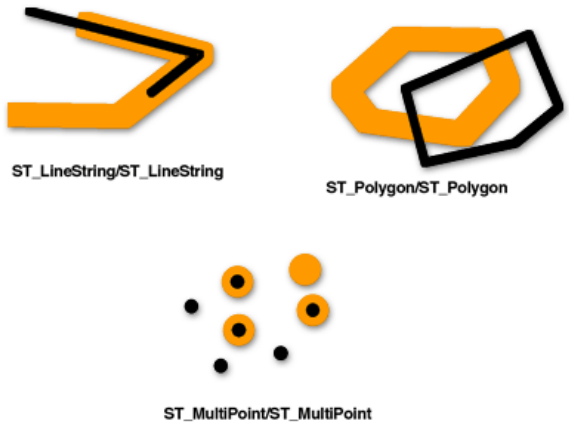
			Geometrie b	
		Innenbereich	**Grenze**	**Außenbereich**
Geometrie a	**Innenbereich**	*	*	*
	Grenze	*	T	*
	Außenbereich	*	*	*

ST_Intersects-Matrix 4

ST_Overlaps

ST_Overlaps vergleicht zwei Geometrien derselben Dimension und gibt 1 oder t (true) zurück, wenn deren Schnittmenge eine Geometrie ergibt, die sich von den beiden Eingabe-Geometrien unterscheidet, jedoch dieselbe Dimension aufweist.

ST_Overlaps gibt 1 oder t (true) nur für Geometrien derselben Dimension und nur dann zurück, wenn deren Schnittmenge eine Geometrie derselben Dimension ergibt. Das bedeutet, dass ST_Overlaps 1 oder t (true) zurückgibt, wenn die Schnittmenge von zwei ST_Polygons ein ST_Polygon ergibt.



Diese Mustermatrix gilt für ST_Polygon/ST_Polygon-, ST_MultiPoint/ST_MultiPoint- und ST_MultiPolygon/ST_MultiPolygon-Überlagerungen. Bei diesen Kombinationen gibt das ST_Overlaps-Prädikat "true" zurück, wenn der Innenbereich beider Geometrien den Innenbereich und den Außenbereich der jeweils anderen Geometrie schneidet.

			Geometrie b	
		Innenbereich	**Grenze**	**Außenbereich**
Geometrie a	**Innenbereich**	T	*	T
	Grenze	*	*	*
	Außenbereich	T	*	*

ST_Overlaps-Matrix 1

Die folgende Mustermatrix gilt für ST_LineString/ST_LineString- und ST_MultiLineString/ST_MultiLineString-Überlappungen. In diesem Fall muss die Schnittmenge der Geometrien eine Geometrie ergeben, die eine Dimension von 1 aufweist (ein weiteres ST_LineString- bzw. ST_MultiLineString-Prädikat). Wenn die Dimension der Schnittmenge der Innenbereiche 0 (einen Punkt) ergibt, gibt das ST_Overlaps-Prädikat "false" zurück. Das ST_Crosses-Prädikat gibt dagegen "true" zurück.

			Geometrie b	
		Innenbereich	**Grenze**	**Außenbereich**
Geometrie a	**Innenbereich**	1	*	T
	Grenze	*	*	*
	Außenbereich	T	*	*

ST_Overlaps-Matrix 2

ST_Relate

ST_Relate gibt 1 oder t (true) zurück, wenn die durch die Mustermatrix angegebene räumliche Beziehung gültig ist. Der Wert 1 oder t (true) gibt an, dass zwischen den Geometrien eine räumliche Beziehung besteht.

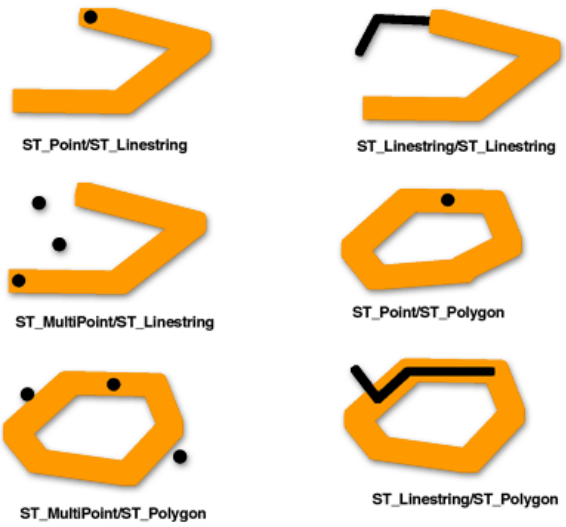
Wenn zwischen den Innenbereichen oder Grenzen der Geometrien a und b eine Beziehung besteht, gibt ST_Relate "true" zurück. Ob die Außenbereiche einer Geometrie den Innenbereich oder die Grenze der anderen Geometrie schneiden, spielt keine Rolle.

			Geometrie b	
		Innenbereich	**Grenze**	**Außenbereich**
Geometrie a	**Innenbereich**	T	T	*
	Grenze	T	T	*
	Außenbereich	*	*	*

ST_Relate-Matrix

ST_Touches

ST_Touches gibt 1 oder t (true) zurück, wenn keiner der in beiden Geometrien enthaltenen Punkte die Innenbereiche der beiden Geometrien schneidet. Mindestens eine Geometrie muss ein ST_LineString, ST_Polygon, ST_MultiLineString oder ST_MultiPolygon sein.



Die Mustermatrix besagt, dass das ST_Touches-Prädikat gibt "true" zurück, wenn sich die Innenbereiche der Geometrie nicht schneiden und die Grenze einer der beiden Geometrien den Innenbereich oder die Grenze der jeweils anderen Geometrie schneidet.

Das ST_Touches-Prädikat gibt "true" zurück, wenn die Grenze von Geometrie b den Innenbereich von Geometrie a schneidet, die Innenbereiche sich jedoch nicht schneiden.

			Geometrie b	
		Innenbereich	**Grenze**	**Außenbereich**
Geometrie a	**Innenbereich**	F	T	*
	Grenze	*	*	*
	Außenbereich	*	*	*

ST_Touches-Matrix 1

Das ST_Touches-Prädikat gibt "true" zurück, wenn die Grenze von Geometrie a den Innenbereich von Geometrie b schneidet, die Innenbereiche sich jedoch nicht schneiden.

			Geometrie b	
		Innenbereich	**Grenze**	**Außenbereich**
Geometrie a	**Innenbereich**	F	*	*
	Grenze	T	*	*
	Außenbereich	*	*	*

ST_Touches-Matrix 2

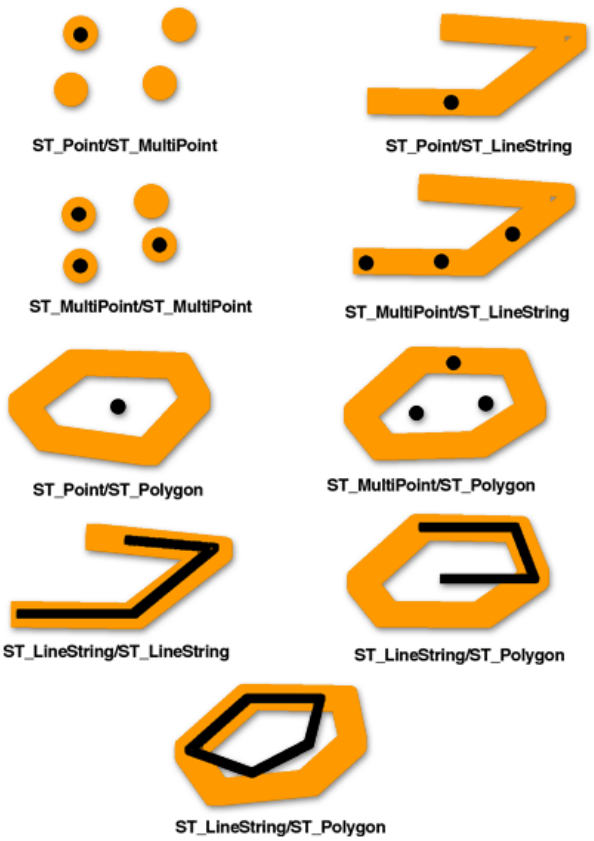
Das ST_Touches-Prädikat gibt "true" zurück, wenn sich die Grenzen der beiden Geometrien schneiden, die Innenbereiche jedoch nicht.

			Geometrie b	
		Innenbereich	**Grenze**	**Außenbereich**
Geometrie a	**Innenbereich**	F	*	*
	Grenze	*	T	*
	Außenbereich	*	*	*

ST_Touches-Matrix 3

ST_Within

ST_Within gibt 1 oder t (true) zurück, wenn die erste Geometrie vollständig in der zweiten Geometrie enthalten ist. ST_Within ergibt das genaue Gegenteil von ST_Contains.



Die Mustermatrix des ST_Within-Prädikats besagt, dass sich die Innenbereiche der beiden Geometrien schneiden müssen und dass der Innenbereich und die Grenze der ersten Geometrie (Geometrie a) den Außenbereich der zweiten Geometrie (Geometrie b) nicht schneiden dürfen.

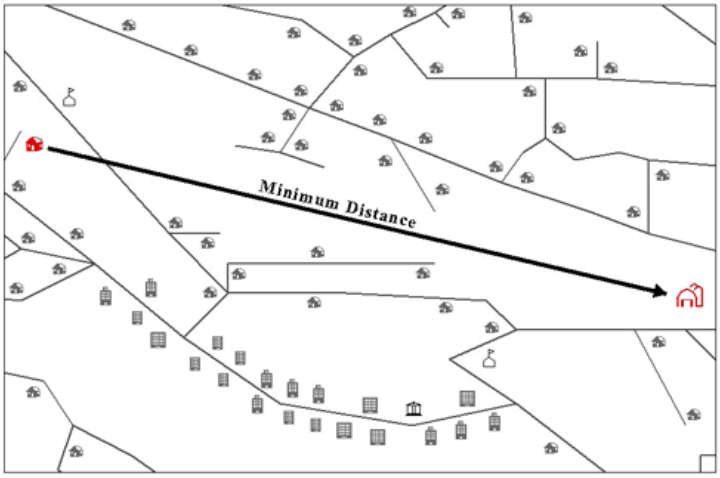
			Geometrie b	
		Innenbereich	**Grenze**	**Außenbereich**
Geometrie a	**Innenbereich**	T	*	F
	Grenze	*	*	F
	Außenbereich	*	*	*

ST_Within-Matrix

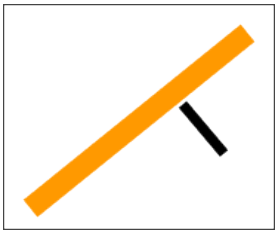
Andere räumliche Beziehungen

Die folgenden Funktionen vergleichen die räumliche Beziehung zwischen Geometrien. Dabei vergleichen sie jedoch nicht nur den Innenbereich, die Grenze und die Außenbereiche der Geometrien.

- ST_Distance: Diese Funktion verwendet zwei getrennte Geometrien als Eingabe und gibt den kleinsten Abstand zwischen den beiden zurück. Wenn die Geometrien nicht getrennt sind (d. h., wenn sie lagegleich sind), gibt die Funktion für den kleinsten Abstand 0 zurück.
  Der kleinste Abstand zwischen Features entspricht der kürzesten Entfernung zwischen zwei Positionen. Dies ist nicht die Entfernung, die Sie zwischen zwei Orten zurücklegen würden, sondern die Entfernung, die Sie berechnen würden, wenn Sie zwei Punkte auf einer Karte mit einer geraden Linie verbinden.



- ST_DWithin: Sie geben einen Entfernungswert sowie die miteinander zu vergleichenden Geometrien an. ST_DWithin gibt "true" zurück, wenn sich die Geometrien innerhalb der angegebenen Entfernung voneinander befinden.
- ST_EnvIntersects: Diese Funktion prüft, ob sich die räumlichen Envelopes der angegebenen Geometrien schneiden. ST_Intersects prüft dagegen, ob sich die Geometrien selbst schneiden.
  Im folgenden Beispiel schneiden sich die Envelopes der beiden Linien, die Linien selbst schneiden sich jedoch nicht:



- ST_OrderingEquals: Diese Funktion weitet den von ST_Equals durchgeführten Vergleich aus und prüft auch, ob die Koordinaten der Geometrien in der gleichen Reihenfolge definiert sind (XY bzw. YX). Selbst wenn die Geometrien den gleichen Bereich belegen, gibt ST_OrderingEquals den "false" zurück, wenn die X- und Y-Koordinaten nicht in derselben Reihenfolge definiert sind.

Räumliche Operationen

Bei räumlichen Operationen werden Geometriefunktionen verwendet, um als Eingabedaten verwendete räumliche Daten zu analysieren und anschließend Ausgabedaten zu generieren, bei denen es sich um Ableitungen der für die Eingabedaten durchgeführten Analyse handelt.

Folgende Daten können durch eine räumliche Operation abgeleitet werden:

- Ein Polygon, bei dem es sich um einen Puffer um ein Eingabe-Feature handelt
- Eine einzelne Geometrie, die das Ergebnis aus einer Analyse ist, die für eine Sammlung von Geometrien durchgeführt wurde
- Ein einzelnes Feature, das sich bei einem Vergleich ergibt, mit dem der Teil eines Features ermittelt wird, der nicht denselben physischen Raum wie ein anderes Feature einnimmt
- Ein einzelnes Feature, das sich bei einem Vergleich ergibt, mit dem die Teile eines Features ermittelt werden, die den physischen Raum eines anderen Features schneiden
- Ein Multipart-Feature, das aus den Teilen beider Eingabe-Features besteht, die nicht denselben physischen Raum wie ein anderes Feature
- Ein Feature, bei dem es sich um die Vereinigung von zwei Geometrien handelt

Die Analyse der Eingabedaten liefert die Koordinaten oder Textrepräsentation der resultierenden Geometrien. Sie können diese Informationen im Rahmen einer größeren Abfrage für weitere Analysen oder die Ergebnisse als Eingabe für eine andere Tabelle verwenden.

So können Sie beispielsweise eine Pufferoperation in die `WHERE`-Klausel einer Überschneidungsabfrage einbinden, wenn Sie feststellen möchten, ob die angegebene Geometrie einen Bereich mit einer bestimmten Größe um eine andere Geometrie herum schneidet.

Hinweis:
In den folgenden Beispielen werden ST_Geometry-Funktionen verwendet. Informationen zu bestimmten Geometriefunktionen sowie zur entsprechenden Syntax, die für andere Datenbanken und räumliche Datentypen verwendet werden, finden Sie in der Dokumentation zur jeweiligen Datenbank bzw. zum jeweiligen Datentype.

In diesem Beispiel müssen Benachrichtigungen an alle Grundstückseigentümer im Umkreis von 1.000 Fuß um eine Straßensperrung versendet werden. Die `WHERE`-Klausel generiert einen Puffer von 1.000 Fuß um die Straße herum, die gesperrt werden soll. Anschließend wird dieser Puffer mit den Grundstücken im Gebiet verglichen, um zu ermitteln, welche Grundstücke vom Puffer geschnitten werden.

```
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';
```

In diesem Beispiel wird eine bestimmte Straße (Main) in der `WHERE`-Klausel ausgewählt. Anschließend wird ein Puffer um die Straße herum erstellt und mit den Features in der Flurstückstabelle verglichen, um zu ermitteln, ob sich Puffer und Flurstücke überschneiden.* Für alle Flurstücke an der Main Street, die vom Puffer überschnitten werden, werden Name und Adresse des Flurstückbesitzers zurückgegeben.

Hinweis:
*Die Reihenfolge, in der die Teile der `WHERE`-Klausel ausgeführt werden, hängt vom Datenbank-Optimierer ab.

Im folgenden Beispiel werden die aus einer räumlichen Operation (Vereinigung) – die für Tabellen mit Stadtteilen und Schulbezirken durchgeführt wurde – resultierenden Features in eine andere Tabelle eingefügt:

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

Weitere Informationen zur Verwendung von räumlichen Operatoren mit ST_Geometry finden Sie unter Funktionen für räumliche Operationen für ST_Geometry.

Funktionen für räumliche Operationen für ST_Geometry

Bei räumlichen Operationen werden Geometriefunktionen verwendet, um als Eingabedaten verwendete räumliche Daten zu analysieren und anschließend Ausgabedaten zu generieren, bei denen es sich um Ableitungen der für die Eingabedaten durchgeführten Analyse handelt.

Sie können die in den folgenden Abschnitten beschriebenen Operationen durchführen, um aus Eingabe-Features Features zu erstellen.

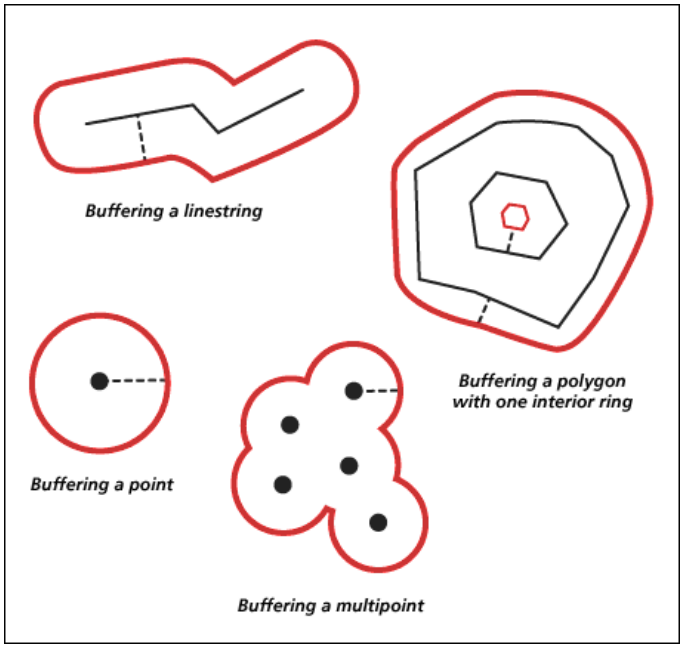
Pufferungsgeometrie

Die Funktion ST_Buffer generiert eine Geometrie, indem sie die von Ihnen angegebene Geometrie mit dem von Ihnen angegebenen Abstand einkreist. Ein einzelnes Polygon entsteht, wenn eine primäre Geometrie gepuffert wird oder wenn die Pufferpolygone einer Sammlung so nahe beieinander liegen, dass sich überlappen. Wenn die Elemente einer gepufferten Sammlung weit genug voneinander entfernt sind, ergibt sich aus einzelnen gepufferten ST_Polygons ein ST_MultiPolygon.

Die Funktion ST_Buffer akzeptiert sowohl positive als auch negative Abstände. Negative Abstände können jedoch nur auf zweidimensionale Geometrien (ST_Polygon und ST_MultiPolygon) angewendet werden. ST_Buffer verwendet den absoluten Wert des Pufferabstands, wenn die Quellgeometrie weniger als zwei Dimensionen hat, d. h. alle Geometrien außer ST_Polygon und ST_MultiPolygon. Bei positiven Pufferabständen entstehen Polygonringe, die vom Mittelpunkt der Quellgeometrie weg verlaufen, und – für den äußeren Ring eines ST_Polygon oder ST_MultiPolygon – bei negativen Abständen zum Mittelpunkt hin verlaufen. Bei inneren Ringen eines ST_Polygon oder ST_MultiPolygon verläuft der Pufferring zum Mittelpunkt hin, wenn der Pufferabstand positiv ist, und vom Mittelpunkt weg, wenn er negativ ist.

Bei der Pufferung werden überlappende Pufferpolygone zusammengeführt. Negative Abstände, die größer als die Hälfte der maximalen inneren Breite eines Polygons sind, ergeben eine leere Geometrie.

In der folgenden Abbildung sind Puffer in Rot dargestellt.



ConvexHull

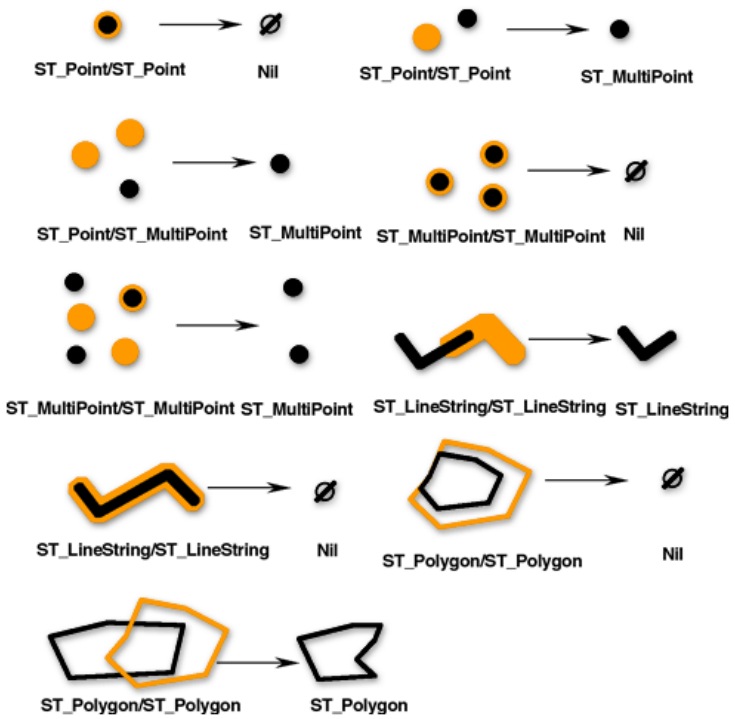
Die Funktion ST_ConvexHull gibt das Konvexe-Hülle-Polygon einer Geometrie zurück, bei der mindestens drei Stützpunkte eine konvexe Hülle bilden. Wenn die Stützpunkte der Geometrie keine konvexe Hülle bilden, gibt ST_ConvexHull NULL zurück. Wenn ST_ConvexHull beispielsweise für eine Linie mit zwei Stützpunkten verwendet wird, wird NULL zurückgegeben. Ebenso wird NULL zurückgegeben, wenn die ST_ConvexHull-Operation für ein Punkt-Feature verwendet wird. Die Erstellung einer konvexen Hülle ist häufig der erste Schritt beim Mosaikieren einer Gruppe von Punkten zum Erstellen eines TIN (Triangulated Irregular Network, unregelmäßigen Dreiecksnetzes).

Differenz bei Geometrien

Die Funktion ST_Difference gibt den Teil der primären Geometrie zurück, der von der sekundären Geometrie nicht geschnitten wird. Dies ist der logische `AND NOT`-Operator des Raums.

Die Funktion ST_Difference wird nur für Geometrien ähnlicher Dimension verwendet und gibt eine Sammlung zurück, die dieselbe Dimension wie die Quellgeometrien aufweist. Wenn die Quellgeometrien gleich sind, wird eine leere Geometrie zurückgegeben.

In der folgenden Abbildung sind die ersten Eingabe-Geometrien in Schwarz und die zweiten Eingabe-Geometrien in Orange dargestellt.



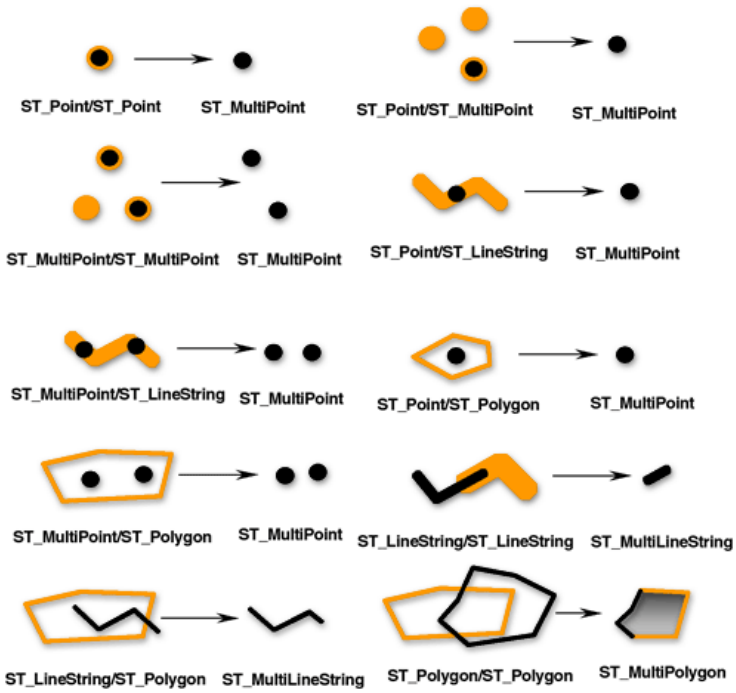
Schnittmenge von Geometrien

Die Funktion ST_Intersection gibt die Schnittmenge von zwei Geometrien zurück. Die Schnittmenge wird immer als Sammlung zurückgegeben, die der mindestens erforderlichen Dimension der Quellgeometrien entspricht.

Beispielsweise gibt die Funktion ST_Intersection für einen ST_LineString, der ein ST_Polygon schneidet, den Teil des ST_LineString, den der Innenbereich und die Grenze des ST_Polygon gemeinsam haben, als ST_MultiLineString

zurück. Wenn der Quell-ST_LineString das ST_Polygon mit mindestens zwei nicht kontinuierlichen Segmenten schneidet, enthält der ST_MultiLineString mehrere ST_LineStrings. Wenn sich die Geometrien nicht schneiden oder wenn die Schnittmenge eine Dimension ergibt, die kleiner ist als beide Quellgeometrien, wird eine leere Geometrie zurückgegeben.

In der folgenden Abbildung sind Beispiele für die ST_Intersection-Funktion dargestellt. Die ersten Eingabe-Geometrien sind in Schwarz und die zweiten Eingabe-Geometrien in Orange dargestellt.

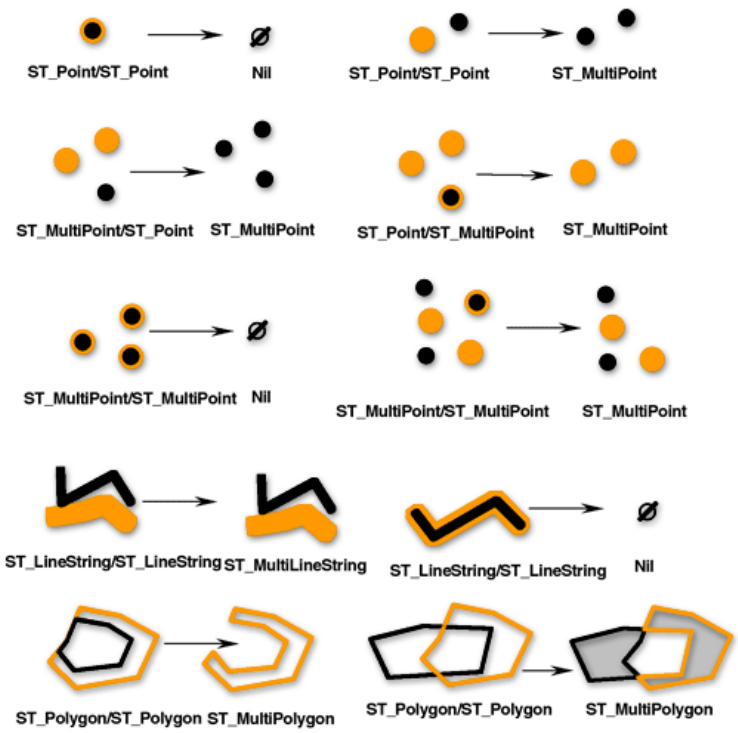


Symmetrische Differenz bei Geometrien

Die Funktion ST_SymmetricDiff gibt die Teile der Quellgeometrien zurück, die nicht Teil der Schnittmenge sind. Dies ist der logische XOR-Operator des Raums.

Die Quellgeometrien müssen dieselbe Dimension aufweisen. Wenn die Geometrien gleich sind, gibt die Funktion ST_SymmetricDiff eine leere Geometrie zurück. Ist dies nicht der Fall, gibt die Funktion das Ergebnis als Sammlung zurück.

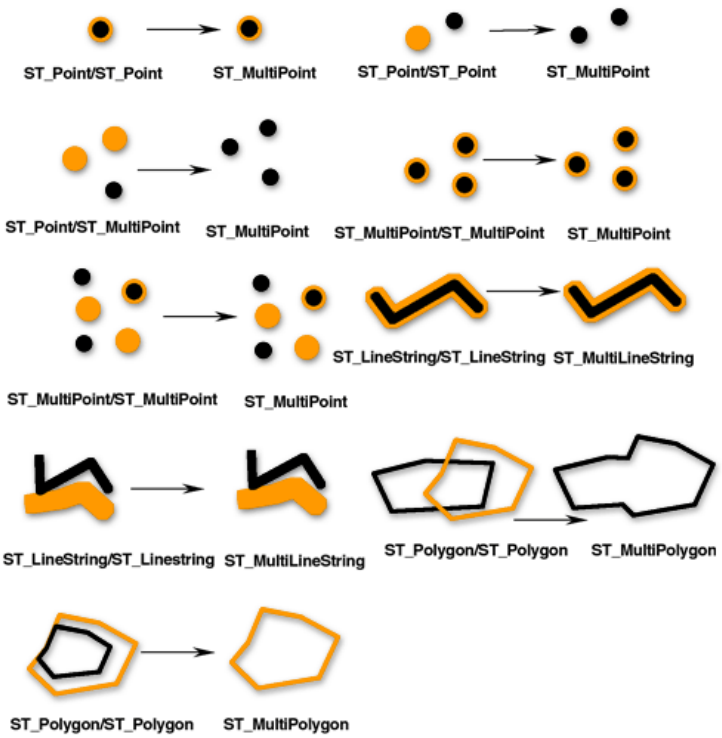
In der folgenden Abbildung sind die ersten Eingabe-Geometrien in Schwarz und die zweiten Eingabe-Geometrien in Orange dargestellt.



Vereinigung bei Geometrien

Die Funktion ST_Union gibt die Vereinigungsmenge von zwei Geometrien zurück. Dies ist der boolesche logische `OR`-Operator des Raums. Die Quellgeometrien müssen dieselbe Dimension aufweisen. ST_Union gibt das Ergebnis immer als Sammlung zurück.

In der folgenden Abbildung sind die ersten Eingabe-Geometrien in Schwarz und die zweiten Eingabe-Geometrien in Orange dargestellt.



Aggregationen

Aggregationsoperationen geben eine einzelne Geometrie als Ergebnis aus einer Analyse zurück, die für eine Sammlung von Geometrien durchgeführt wurde. Die Funktion ST_Aggr_ConvexHull gibt das Multipolygon zurück, das aus den Konvexe-Hülle-Polygonen der einzelnen Eingabe-Geometrien besteht. Eine Eingabe-Geometrie mit weniger als drei Stützpunkten hat keine konvexe Hülle. Wenn alle Eingabe-Geometrien weniger als drei Stützpunkte haben, gibt ST_Aggr_ConvexHull NULL zurück.

Die Funktion ST_Aggr_Intersection gibt eine einzelne Geometrie zurück, die eine Aggregation der Schnittmengen aller Eingabe-Geometrien ist.

ST_Aggr_Intersection findet die Schnittmenge mehrerer Geometrien, während ST_Intersection nur die Schnittmenge zweier Geometrien findet. Wenn Sie beispielsweise nach einer Immobilie in einem Gebiet suchen, in dem bestimmte Dienstleistungen angeboten werden – z. B. ein bestimmter Telefon-Service oder schnelles Internet oder eine Immobilie in einem bestimmten Schulbezirk oder in einem Gebiet, das von einem bestimmten Stadtrat vertreten wird – müssen Sie die Schnittmenge all dieser Gebiete suchen. Die Suche nach nur zwei dieser Gebiete liefert nicht alle erforderlichen Informationen. Daher müssen Sie die Funktion ST_Aggr_Intersection verwenden, sodass alle Gebiete mit einer Abfrage ausgewertet werden können.

Ein weiteres Beispiel: Bei der Suche nach der Schnittmenge von Linien und Punkten in zwei Feature-Classes wird von den Funktionen jeweils Folgendes zurückgegeben:

- ST_Intersection: Für jeden Schnittpunkt wird eine ST_Point-Geometrie zurückgegeben.
- ST_Aggr_Intersection: Es wird eine ST_MultiPoint-Geometrie zurückgegeben, die aus allen Punkten der Schnittmenge besteht. (Wenn sich jedoch nur ein Punkt-Feature und ein Linien-Feature schneiden, wird nur eine ST_Point-Geometrie zurückgegeben.)

Die Funktion ST_Aggr_Union gibt eine Geometrie zurück, die die Vereinigungsmenge aller angegebenen

Geometrien ist.

Die Eingabe-Geometrien müssen denselben Typ aufweisen. So können Sie beispielsweise ST_LineStrings mit ST_LineStrings oder ST_Polygons mit ST_Polygons vereinigen. Eine ST_LineString-Feature-Class kann jedoch nicht mit einer ST_Polygon-Feature-Class vereinigt werden.

Eine Aggregatvereinigung ergibt in der Regel die Geometrie einer Sammlung. Wenn Sie beispielsweise die Aggregatvereinigung von allen unbebauten Flurstücken ermitteln möchten, die kleiner sind als ein halber Acre, wird als Geometrie ein Multipolygon zurückgegeben, es sei denn, alle Flurstücke, die den Kriterien entsprechen, sind zusammenhängend. In diesem Fall wird ein Polygon zurückgegeben.

Minimale Entfernung

Die bisherigen Funktionen haben neue Geometrien zurückgegeben. Die Funktion ST_Distance führt eine räumliche Operation durch – sie ermittelt den kleinsten Abstand zwischen zwei Geometrien –, gibt aber keine neue Geometrie zurück.

Parametrische Kreise, Ellipsen und Keile

Sie können parametrische Kreise, Ellipsen und Keile in ST_Geometry-Spalten mithilfe der Funktion "ST_Geometry" erstellen und abfragen.

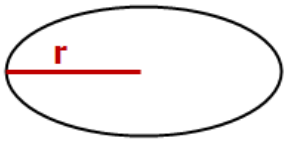
Parametrische Kreise, Ellipsen und Keile sind Polygone, die von bestimmten Parametern definiert werden, z. B. von Koordinatenwerten, Winkeln und Radien. In der Datenbank werden diese Parameter anstelle von bestimmten Stützpunkten und Linien gespeichert. Durch Speichern der Parameter, die das Shape definieren, können parametrische Shapes genauer sein. Sie benötigen auch weniger Speicherplatz als vielseitige Polygondarstellungen. Die Verwendung parametrischer Shapes ermöglicht auch das Einschließen von Z-Koordinaten- und Messwertparametern (M-Wert).

Beim Erstellen eines Kreises werden sieben Parameter abgefragt:

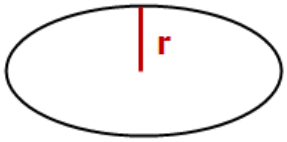
- Ein X-Koordinatenwert des Kreismittelpunktes
- Ein Y-Koordinatenwert des Kreismittelpunktes
- Ein Z-Koordinatenwert des Kreismittelpunktes
- M-Wert
- Radius des zu erstellenden Kreises
- Anzahl der Punkte, die zum Definieren des Kreises verwendet werden
  Die Mindestanzahl von Punkten, die Sie angeben können, ist 9. Wenn Sie keine Reihe von Punkten angeben, werden standardmäßig 50 verwendet. Diese Punkte werden nicht mit dem Shape gespeichert. Sie werden gleichzeitig mit dem Kreis generiert, der erstellt wird, um das Shape zu überprüfen.
- Raumbezugs-ID (SRID) zum Platzieren des Kreises im Raum

Beim Erstellen einer Ellipse werden neun Parameter abgefragt:

- Ein X-Koordinatenwert des Ellipsenmittelpunktes
- Ein Y-Koordinatenwert des Ellipsenmittelpunktes
- Ein Z-Koordinatenwert des Ellipsenmittelpunktes
- M-Wert
- Große Halbachse der Ellipse
  Die große Halbachse ist der längste Radius einer Ellipse. Der für die große Halbachse angegebene Wert muss größer als der Wert der kleinen Halbachse sein.



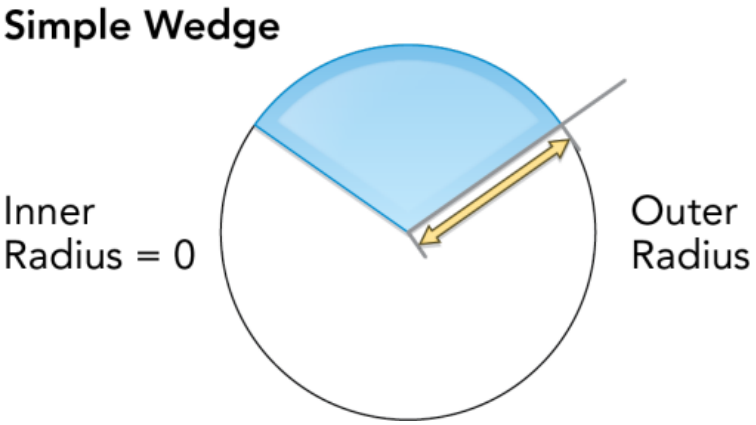
- Kleine Halbachse der Ellipse
  Die kleine Halbachse ist der kürzeste Radius einer Ellipse. Der für die kleine Halbachse angegebene Wert muss größer als 0,0 sein.



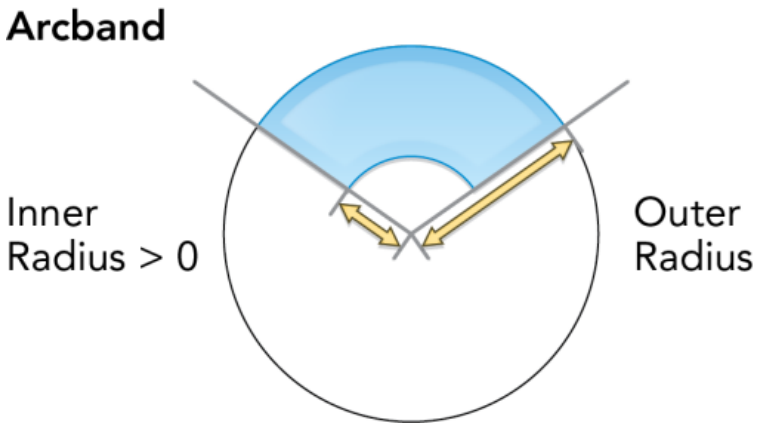
- Rotationswinkel der Ellipse
  Der Wert für den Rotationswinkel wird in Grad angegeben und muss größer als 0,0 und kleiner als 360 sein. Die Rotation erfolgt im Uhrzeigersinn.
- Anzahl der Punkte, die zum Definieren der Ellipse verwendet werden
  Die Mindestanzahl von Punkten, die Sie angeben können, ist 9. Wenn Sie keine Reihe von Punkten angeben, werden standardmäßig 50 Punkte verwendet. Diese Punkte werden nicht mit dem Shape gespeichert. Sie werden gleichzeitig mit der Ellipse generiert, die erstellt wird, um das Shape zu überprüfen.
- SRID zum Platzieren der Ellipse im Raum

Beim Erstellen eines Keiles werden zehn Parameter erwartet:

- Ein X-Koordinatenwert des Kreismittelpunktes, der den Keil definiert
- Ein Y-Koordinatenwert des Kreismittelpunktes, der den Keil definiert
- Ein Z-Koordinatenwert des Kreismittelpunktes, der den Keil definiert
- M-Wert
- Der Anfangswinkel des Keiles
  Der Anfangswinkel definiert den Anfang des Keiles als Anzahl der Grade, die gegen den Uhrzeigersinn von 0 Grad an gemessen werden.
- Der Endwinkel des Keiles
  Der Endwinkel definiert das Ende des Keiles als Anzahl der Grade, die gegen den Uhrzeigersinn von 0 Grad an gemessen werden.
- Der äußere Radius
  Der äußere Radius definiert den Abstand vom Mittelpunkt des Kreises zum äußersten Punkt des Keiles.
- Der innere Radius
  Der innere Radius definiert den Abstand vom Mittelpunkt des Kreises zum innersten Punkt des Keiles und definiert somit den Anfang des Keiles. Wenn der innere Radius 0 beträgt, ist das Shape ein einfacher Keil.



Wenn der innere Radius größer als 0 ist, ist der Keil technisch ein Kreisbogensektor.



- Die Anzahl der Punkte, die verwendet wurde, um den Keil zu definieren
  Die Mindestanzahl von Punkten, die Sie angeben können, ist 9. Wenn Sie keine Reihe von Punkten angeben, werden standardmäßig 80 Punkte verwendet. Diese Punkte werden nicht mit dem Shape gespeichert. Sie werden gleichzeitig mit dem Keil generiert, der erstellt wird, um das Shape zu überprüfen.
- Die SRID, die verwendet wurde, um den Keil im Raum zu platzieren

Alle Radien, einschließlich der großen und kleinen Halbachsen sowie der inneren und äußeren Radien, werden in den Einheiten definiert, die vom mit der SRID angegebenen Koordinatenbezug bestimmt werden.

Informationen zur Syntax und Beispiele zum Erstellen von parametrischen Kreisen, Ellipsen und Keilen finden Sie unter der Funktion ST_Geometry.

ST_Aggr_ConvexHull

Hinweis:
Nur Oracle und SQLite

Definition

ST_Aggr_ConvexHull erstellt eine Geometrie, bei der es sich um eine konvexe Hülle einer Geometrie handelt, die sich aus der Vereinigung aller Eingabe-Geometrien ergibt. ST_Aggr_ConvexHull entspricht im Prinzip ST_ConvexHull(ST_Aggr_Union(geometries)).

Syntax

Oracle

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

SQLite

```
st_aggr_convexhull (geometry st_geometry)
```

Rückgabetyp

Oracle

ST_Geometry

SQLite

Geometryblob

Beispiel

Im Beispiel wird die Tabelle "service_territories" erstellt und eine SELECT-Anweisung ausgeführt, die alle Geometrien aggregiert. Dadurch wird eine einzelne Geometrie generiert, die die konvexe Hülle der Vereinigung aller Shapes darstellt.

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

Hinweis:
Nur Oracle und SQLite

Definition

Die Funktion "ST_Aggr_Intersection" gibt eine einzelne Geometrie zurück, die eine Vereinigungsmenge der Schnittmengen aller Eingabegeometrien ist.

Syntax

Oracle

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

SQLite

```
st_aggr_intersection (geometry1 geometryblob)
```

Rückgabetyp

Oracle

ST_Geometry

SQLite

Geometryblob

Beispiel

In diesem Beispiel sucht ein Biologe die Schnittmenge dreier Lebensräume von Wildtieren.

Oracle

Erstellen Sie zunächst die Tabelle, in der die Lebensräume gespeichert sind.

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

Fügen Sie anschließend der Tabelle die drei Polygone hinzu.

```
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)
);
```

Wählen Sie schließlich die Schnittmenge für die Lebensräume aus.

```
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

Erstellen Sie zunächst die Tabelle, in der die Lebensräume gespeichert sind.

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

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

Fügen Sie anschließend der Tabelle die drei Polygone hinzu.

```
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)
);
```

Wählen Sie abschließend die Schnittmenge für die Lebensräume aus.

```
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

Definition

Die Funktion "ST_Aggr_Union" gibt eine einzelne Geometrie zurück, die die Vereinigungsmenge aller Eingabegeometrien ist.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_aggr_union(geometry geometryblob)
```

Rückgabetyp

Oracle und PostgreSQL

ST_Geometry

SQLite

Geometryblob

Beispiel

Ein Marketingfachmann muss eine Geometrie aller Einzugsgebiete erstellen, in denen der Absatz 1000 Einheiten übersteigt. Für dieses Beispiel erstellen Sie die Tabelle "service_territories1" und füllen Sie mit Absatzzahlen. Anschließend verwenden Sie "st_aggr_union" in einer SELECT-Anweisung, um das Multipolygon zurückzugeben, das die Vereinigungsmenge aller Geometrien darstellt, in denen der Absatz größer gleich 1000 Einheiten war.

Oracle und 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

Definition

ST_Area gibt die Fläche eines Polygons oder Multipolygons zurück.

Syntax

Oracle und 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)
```

Rückgabetyp

Doppelte Genauigkeit

Beispiel

Der technische Beigeordnete benötigt eine Liste mit Baugebieten. Zum Erstellen der Liste wählt ein GIS-Fachmann die Gebäude-ID und Fläche des Grundrisses jedes Gebäudes aus.

Die Gebäudegrundrisse werden in der Tabelle "bfp" gespeichert.

Zum Erfüllen der Anforderung des technischen Beigeordneten wählt der GIS-Fachmann den eindeutigen Schlüssel, die Gebäude-ID und Fläche des Grundrisses jedes Gebäudes in der Tabelle "bfp" aus.

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

Definition

ST_AsBinary wählt ein Geometrieobjekt aus und gibt dessen WKB-Darstellung (Well-Known Binary) zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_asbinary (geometry geometryblob)
```

Rückgabetyp

Oracle und PostgreSQL

ST_Geometry

SQLite

Geometryblob

Beispiel

In diesem Beispiel wird die WKB-Spalte des Datensatzes 1111 mit dem Inhalt aus der Spalte "GEOMETRY" des Datensatzes 1100 gefüllt.

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

Definition

ST_AsText wählt eine Geometrie aus und gibt dessen WKT-Darstellung (Well-Known Text) zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_astext (geometry geometryblob)
```

Rückgabetyp

Oracle

CLOB

PostgreSQL und SQLite

Text

Beispiel

Die Funktion ST_AsText wandelt den Standort "hazardous_sites" in dessen Textbeschreibung um.

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

Definition

"ST_Boundary" gibt für eine Geometrie die kombinierte Grenze als Geometrieobjekt zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_boundary (geometry geometryblob)
```

Rückgabetyp

Oracle und PostgreSQL

ST_Geometry

SQLite

Geometryblob

Beispiel

In diesem Beispiel wird die Tabelle "boundaries" mit den beiden Spalten "type" und "geometry" erstellt. Die nachfolgenden INSERT-Anweisungen fügen für jede Subclass-Geometrie einen Datensatz ein. Die Funktion "ST_Boundary" ruft die Grenze der einzelnen Subclasses, die in der Spalte "geometry" gespeichert sind, ab. Beachten Sie, dass die Dimension der resultierenden Geometrie stets um 1 kleiner ist als die der Eingabe-Geometrie. Punkte und Multipoints ergeben stets eine Grenze mit leerer Geometrie und einer Dimension von –1. Für Linestrings und Multilinestrings wird eine Multipoint-Grenze mit einer Dimension von 0 zurückgegeben. Polygone und Multipolygone ergeben stets eine Multilinestring-Grenze mit einer Dimension von 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

Definition

ST_Buffer wählt ein Geometrieobjekt und eine Entfernung aus und gibt ein Geometrieobjekt zurück, das der das Quellobjekt umgebende Puffer ist.

Syntax

`Unit_name` ist die Maßeinheit für den Pufferabstand (z. B. `Meter`, `Kilometer`, `Fuß` oder `Meile`). Weitere Informationen finden Sie in der ersten Tabelle in der Datei `Projected coordinate system tables.pdf`, die Sie über Koordinatensysteme, Projektionen und Transformationen aufrufen können.

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)
```

Rückgabetyp

Oracle und PostgreSQL

ST_Geometry

SQLite

Geometryblob

Beispiel

In diesem Beispiel werden die beiden Tabellen "sensitive_areas" und "hazardous_sites" erstellt, die Tabellen mit Daten aufgefüllt, mithilfe von ST_Buffer ein Puffer um die Polygone in der Tabelle "hazardous_sites" generiert und die Stellen ermittelt, an denen diese Puffer die Polygone in "sensitive_areas" überlappen.

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

Definition

ST_Centroid wählt ein Polygon, Multipolygon oder einen Mehrfachlinienzug aus und gibt den Punkt in der Mitte des Envelope der Geometrie zurück. Dies bedeutet, dass sich der Schwerpunkt auf halber Strecke zwischen der minimalen und maximalen X- und Y-Ausdehnung der Geometrie befindet.

Syntax

Oracle und 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)
```

Rückgabetyp

ST_Point

Beispiel

Der städtische GIS-Fachmann möchte die Multipolygone von Gebäudegrundrissen in einer Bebauungsgrafik als einzelne Punkte anzeigen. Die Gebäudegrundrisse sind in der Tabelle "bfp" gespeichert, die mit den für jede Datenbank angezeigten Anweisungen erstellt und aufgefüllt wurde.

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

Definition

ST_Contains wählt zwei Geometrieobjekte aus und gibt "1" (Oracle und SQLite) oder "t" (PostgreSQL) zurück, wenn das zweite Objekt vollständig im ersten Objekt enthalten ist. Andernfalls wird "0" (Oracle und SQLite) oder "f" (PostgreSQL) zurückgegeben.

Syntax

Oracle und PostgreSQL

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

SQLite

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

Rückgabetyp

Boolesch

Beispiel

In den folgenden Beispielen werden zwei Tabellen erstellt. Die Tabelle "buildingfootprints" enthält die Gebäudegrundrisse einer Stadt, die Tabelle "lots" Flurstücke. Der technische Beigeordnete will sicherstellen, dass sich alle Gebäudegrundrisse vollständig innerhalb ihrer Flurstücke befinden.

Der technische Beigeordnete verwendet "ST_Intersects" und "ST_Contains", um die Gebäude auszuwählen, die nicht vollständig in einer Parzelle enthalten sind.

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

Definition

ST_ConvexHull gibt die konvexe Hülle eines ST_Geometry-Objekts zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_convexhull (geometry1 geometryblob)
```

Rückgabetyp

Oracle und PostgreSQL

ST_Geometry

SQLite

Geometryblob

Beispiel

In diesen Beispielen wird die Tabelle "sample_geometries" mit den drei Spalten "id", "spatial_type" und "geometry" erstellt. Im Feld "spatial_type" wird der Typ der Geometrie gespeichert, die in der Spalte "geometry" erstellt wird. In die Tabelle werden drei Features eingefügt: ein Linestring-, ein Polygon- und ein Multipoint-Feature.

Die SELECT-Anweisung, die die ST_ConvexHull-Funktion enthält, gibt die konvexe Hülle der einzelnen Geometrien zurück.

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

Definition

ST_CoordDim gibt die Dimensionen von Koordinatenwerten für eine Geometriespalte zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_coorddim (geometry1 geometryblob)
```

Rückgabetyp

Ganzzahl

2 = X-, Y-Koordinaten

3 = X-, Y- Z- oder X-, Y-, M-Koordinaten

4 = X-, Y-, Z-, M-Koordinaten

Beispiel

In diesen Beispielen wird die Tabelle "coorddim_test" mit den Spalten "geotype" und "g1" erstellt. In der Spalte "geotype" werden der Name der Geometrie-Subclass und die Dimension gespeichert, die in der ST_Geometry-Spalte "g1" gespeichert sind.

Mit der SELECT-Anweisung werden der Subclass-Name, der in der Spalte "geotype" gespeichert ist, und die Dimension der Koordinaten dieser Geometrie aufgelistet.

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

Definition

ST_Crosses wählt zwei ST_Geometry-Objekte aus und gibt den Wert "1" (Oracle und SQLite) oder "t" (PostgreSQL) zurück, wenn deren Schnittmenge eine Geometrie ergibt, deren Dimension der höchsten Dimension der Quellobjekte minus eins entspricht. Das Schnittpunktobjekt muss Punkte enthalten, die sich innerhalb der beiden Quellgeometrien befinden und mit keinem der beiden Quellobjekte identisch sind. Andernfalls ist die Rückgabe "0" (Oracle und SQLite) oder "f" (PostgreSQL).

Syntax

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

Oracle und PostgreSQL

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

SQLite

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

Rückgabetyp

Boolesch

Beispiel

Die Kreisverwaltung erwägt eine neue Vorschrift, die besagt, dass alle Sondermülldeponien im Kreis sich nicht innerhalb eines bestimmten Radius von Wasserläufen befinden dürfen. Der GIS-Fachmann des Kreises hat eine genaue Darstellung fließender Gewässer in der Tabelle "waterways" als Linestrings gespeichert, hat aber für jede der Sondermülldeponien nur einen einzelnen Punkt zur Verfügung.

Um zu bestimmen, ob er den Kreisumweltbeauftragten über vorhandene Deponien informieren muss, die gegen die vorgeschlagene Vorschrift verstoßen, muss der GIS-Fachmann die Sondermüllstandorte mit einem Puffer versehen, um zu prüfen, ob fließende Gewässer die Pufferpolygone schneiden. Die Eigenschaft ST_Crosses vergleicht die gepufferten Sondermüllstandorte mit Wasserläufen und gibt nur die Datensätze zurück, bei denen der Wasserlauf den vorgeschlagenen vorgeschriebenen Radius des Landkreises schneidet.

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

Hinweis:
Nur Oracle und SQLite

Definition

Mit ST_Curve wird aus einer Well-known-Text-Repräsentation ein Kurven-Feature erstellt.

Syntax

Oracle

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

SQLite

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

Rückgabetyp

ST_LineString

Beispiel

In diesem Beispiel wird eine Tabelle mit einer Kurvengeometrie erstellt, Werte eingefügt und in der Tabelle ein Feature ausgewählt.

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

Definition

ST_Difference wählt zwei Geometrieobjekte aus und gibt ein Geometrieobjekt zurück, das die Differenz der Quellobjekte ist.

Syntax

Oracle und PostgreSQL

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

SQLite

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

Rückgabetyp

Oracle und PostgreSQL

ST_Geometry

SQLite

Geometryblob

Beispiel

In den folgenden Beispielen möchte der technische Beigeordnete die Gesamtgröße der städtischen Bauplatzfläche ermitteln, auf der keine Gebäude stehen.

Der technische Beigeordnete verbindet die Tabellen "footprints" und "lots" über die "lot_id" und wählt die Summe der Fläche der Differenz der Bauplätze minus die Grundrisse aus.

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

Definition

"ST_Dimension" gibt die Dimension eines Geometrieobjekts zurück. In diesem Fall bezieht sich Dimension auf Länge und Breite. Ein Punkt beispielsweise besitzt weder Länge noch Breite, sodass er die Dimension 0 hat. Eine Linie besitzt eine Länge, jedoch keine Breite, sodass sie die Dimension 1 hat.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_dimension (geometry1 geometryblob)
```

Rückgabetyp

Integer

Beispiel

Die Tabelle "dimension_test" wird mit den Spalten "geotype" und "g1" erstellt. In der Spalte "geotype" wird der Name der Subclass gespeichert, die wiederum in der Geometriespalte "g1" gespeichert ist.

Die SELECT-Anweisung listet den Subclass-Namen, der in der Spalte "geotype" gespeichert ist, zusammen mit der Dimension des jeweiligen geographischen Typs auf.

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

Definition

ST_Disjoint übernimmt zwei Geometrien und gibt 1 (Oracle und SQLite) oder t (PostgreSQL) zurück, wenn die Überschneidung der beiden Geometrien eine leere Menge ergibt. Andernfalls wird 0 (Oracle und SQLite) oder f (PostgreSQL) zurückgegeben.

Syntax

Oracle und PostgreSQL

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

SQLite

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

Rückgabetyp

Boolesch

Beispiel

In diesem Beispiel werden zwei Tabellen erstellt ("distribution_areas" und "factories"), in die jeweils Werte eingefügt werden. Als nächstes wird ein Puffer um die "factories" erstellt und "st_disjoint" wird verwendet, um zu ermitteln, welche "factory"-Puffer keine "distribution_areas" kreuzen.

Tipp:

Sie könnten in dieser Abfrage stattdessen die Funktion "ST_Intersects" verwenden, indem Sie das Ergebnis der Funktion gleich 0 setzen, weil "ST_Intersects" und "ST_Disjoint" entgegengesetzte Ergebnisse liefern. Die Funktion "ST_Intersects" verwendet den räumlichen Index zum Auswerten der Abfrage, die Funktion "ST_Disjoint " verwendet ihn nicht.

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

Definition

ST_Distance gibt die Entfernung zwischen zwei Geometrien zurück. Die Entfernung wird von den nächsten Stützpunkten der beiden Geometrien gemessen.

Syntax

Oracle und 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)
```

Die folgenden Einheitennamen sind gültig:

Millimeter	Zoll	Yard	Link
Zentimeter	Inch_US	Yard_US	Link_US
Dezimeter	Fuß	Yard_Clarke	Link_Clarke
Meter	Foot_US	Yard_Sears	Link_Sears
Meter_German	Foot_Clarke	Yard_Sears_1922_Truncated	Link_Sears_1922_Truncated
Kilometer	Foot_Sears	Yard_Benoit_1895_A	Link_Benoit_1895_B
50_Kilometers	Foot_Sears_1922_Truncated	Yard_Indian	Chain
150_Kilometers	Foot_Benoit_1895_A	Yard_Indian_1937	Chain_US
Vara_US	Foot_1865	Yard_Indian_1962	Chain_Clarke
Smoot	Foot_Indian	Yard_Indian_1975	Chain_Sears
	Foot_Indian_1937	Klafter	Chain_Sears_1922_Truncated
	Foot_Indian_1962	Mile_US	Chain_Benoit_1895_A
	Foot_Indian_1975	Statute_Mile	Rod
	Foot_Gold_Coast	Nautical_Mile	Rod_US
	Foot_British_1936	Nautical_Mile_US	
		Nautical_Mile_UK	

Rückgabetyp

Doppelte Genauigkeit

Beispiel

Es werden zwei Tabellen, "study1" und "zones", erstellt und gefüllt. Die Funktion "ST_Distance" wird anschließend verwendet, um die Entfernung zwischen der Grenze jedes Teilgebiets und den Polygonen in der Flächentabelle "study1" zu ermitteln, die einen Nutzungscode von 400 aufweisen. Da sich drei Zonen auf dem Shape befinden, sollten drei Datensätze zurückgegeben werden.

Wenn Sie keine Einheiten angeben, werden von "ST_Distance" die Einheiten des Datenprojektionssystems verwendet. Im ersten Beispiel sind dies Dezimalgrad. Da in den letzten beiden Beispielen Kilometer angegeben sind, wird die Entfernung in Kilometern zurückgegeben.

Oracle und 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

Definition

ST_DWithin wählt zwei Geometrien als Eingabe aus und gibt "true" zurück, wenn die Geometrien sich innerhalb der angegebenen Entfernung voneinander befinden. Andernfalls lautet die Rückgabe "false". Das Raumbezugssystem der Geometrien bestimmt, welche Maßeinheit für die angegebene Entfernung verwendet wird. Die an "ST_DWithin" bereitgestellten Geometrien müssen deshalb die gleiche Koordinatenprojektion und Raumbezugs-ID (SRID) verwenden.

Syntax

Oracle und PostgreSQL

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

SQLite

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

Rückgabetyp

Boolesch

Beispiele

In den folgenden Beispielen werden zwei Tabellen erstellt und mit Features versehen. Dann wird die Funktion "ST_DWithin" in zwei verschiedenen SELECT-Anweisungen verwendet: Mit der ersten wird bestimmt, ob ein Punkt in der ersten Tabelle sich innerhalb von 100 Metern Entfernung zu einem Polygon in der zweiten Tabelle befindet. Mit der zweiten wird ermittelt, welche Features sich innerhalb einer Entfernung von 300 Metern zueinander befinden.

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)
 )
;
```

Verwenden Sie ST_DWithin dann, um zu ermitteln, welche Features in den beiden Tabellen sich innerhalb einer Entfernung von 100 Metern voneinander befinden und welche nicht. Die ST_Distance-Funktion ist in dieser Anweisung enthalten, um die tatsächliche Entfernung zu den Features zu zeigen.

```
--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;
```

Die Anweisung gibt Folgendes zurück:

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

Im folgenden Beispiel wird ST_DWithin verwendet, um Features zu finden, die sich innerhalb einer Entfernung von 300 Metern zueinander befinden:

```
--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;
```

Die zweite Select-Anweisung gibt Folgendes zurück, wenn sie für Daten in Oracle ausgeführt wird:

```
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)
 )
;
```

Verwenden Sie ST_DWithin dann, um zu ermitteln, welche Features in den beiden Tabellen sich innerhalb einer Entfernung von 100 Metern voneinander befinden und welche nicht. Die ST_Distance-Funktion ist in dieser Anweisung enthalten, um die tatsächliche Entfernung zu den Features zu zeigen.

```
--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;
```

Die Anweisung gibt Folgendes zurück:

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

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

Im folgenden Beispiel wird ST_DWithin verwendet, um Features zu finden, die sich innerhalb einer Entfernung von 300 Metern zueinander befinden:

```
--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;
```

Die zweite SELECT-Anweisung gibt Folgendes zurück:

```
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)
 )
;
```

Verwenden Sie ST_DWithin dann, um zu ermitteln, welche Features in den beiden Tabellen sich innerhalb einer Entfernung von 100 Metern voneinander befinden und welche nicht. Die ST_Distance-Funktion ist in dieser Anweisung enthalten, um die tatsächliche Entfernung zu den Features zu zeigen.

```
--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;
```

Die Anweisung gibt Folgendes zurück:

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

Im folgenden Beispiel wird ST_DWithin verwendet, um Features zu finden, die sich innerhalb einer Entfernung von 300 Metern zueinander befinden:

```
--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;
```

Die zweite SELECT-Anweisung gibt Folgendes zurück:

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

ST_EndPoint

Definition

ST_EndPoint gibt den letzten Punkt für einen Linestring zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_endpoint (line1 geometryblob)
```

Rückgabetyp

ST_Point

Beispiel

Die Tabelle "endpoint_test" enthält die Ganzzahl-Spalte "gid", durch die die einzelnen Zeilen eindeutig gekennzeichnet werden, und die Spalte "ln1" vom Typ ST_LineString, in der die Linestrings gespeichert werden.

Mit der INSERT-Anweisung werden die Linestrings in die Tabelle "endpoint_test" eingefügt. Der erste Linestring enthält weder Z-Koordinaten noch Messwerte, der zweite dagegen schon.

In der Abfrage sind die Spalte "gid" und die Geometrie "ST_Point" aufgelistet, die durch die Funktion "ST_EndPoint" erstellt werden.

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

Definition

Mit ST_Entity wird der räumliche Entitätstyp eines Geometrieobjekts zurückgegeben. Der räumliche Entitätstyp ist der im Entitätsmitgliedsfeld des Geometrieobjekts gespeicherte Wert.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_entity (geometry1 geometryblob)
```

Rückgabetyp

Es wird eine Zahl (Oracle) oder ganze Zahl (SQLite und PostgreSQL) zurückgegeben, die die folgenden Entitätstypen darstellt:

0	NIL-Shape
1	Punkt
2	Linie (umfasst Spagettilinien)
4	Linestring
8	Fläche
257	Multipoint
258	Multiline (umfasst Spagettilinien)
260	Multilinestring
264	Multiarea

Beispiel

In den folgenden Beispielen wird eine Tabelle erstellt und werden verschiedene Geometrien in die Tabelle eingefügt. "ST_Entity" wird für die Tabelle ausgeführt, um den Geometrie-Subtype jedes Datensatzes in der Tabelle zurückzugeben.

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;
```

Die Select-Anweisung gibt die folgenden Werte zurück:

```
 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;
```

Die Select-Anweisung gibt die folgenden Werte zurück:

```
 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;
```

Die Select-Anweisung gibt die folgenden Werte zurück:

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

ST_Envelope

Definition

"ST_Envelope" gibt das kleinste umgebende Rechteck eines Geometrieobjekts als Polygon zurück.

Detailinformationen:
Diese Funktion entspricht der Simple-Features-Spezifikation desOpen Geospatial Consortium (OGC), die besagt, dass "ST_Envelope" ein Polygon zurückgibt. Für die Arbeit mit Spezialfällen wie Punktgeometrien und horizontalen oder vertikalen Linien gibt die Funktion "ST_Envelope" ein Polygon zurück, das diese Shapes umgibt. Dabei handelt es sich um die Toleranz eines kleinen Envelope, deren Wert auf Basis des XY-Maßstabsfaktors im Raumbezugssystem der Geometrie berechnet wird. Die Toleranz wird von den minimalen X- und Y-Koordinaten subtrahiert und zu den maximalen X- und Y-Koordinaten addiert, sodass für diese Shapes das umgebende Polygon zurückgegeben wird.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_envelope (geometry1 geometryblob)
```

Rückgabetyp

Oracle und PostgreSQL

ST_Geometry

SQLite

Geometryblob

Beispiel

In der Spalte "geotype" der Tabelle "envelope_test" wird der Name der Geometrie-Subclass gespeichert, die wiederum in der Spalte "g1" gespeichert ist. Mit den INSERT-Anweisungen werden die einzelnen Geometrie-Subclasses in die Tabelle "envelope_test" eingefügt.

Als Nächstes wird die Funktion "ST_Envelope" ausgeführt, um den Polygon-Envelope um die einzelnen Geometrien zurückzugeben.

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

Hinweis:
Nur Oracle und SQLite

Definition

ST_EnvIntersects gibt 1 (true) zurück, wenn sich die Envelopes von zwei Geometrien schneiden; andernfalls wird 0 (false) zurückgegeben.

Syntax

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)
```

Rückgabetyp

Boolesch

Beispiel

In diesem Beispiel wird nach einer Geometrie mit einem Envelope gesucht, der vom definierten Polygon überschnitten wird.

Mit der ersten SELECT-Anweisung werden die Envelopes der beiden Geometrien und die Geometrien selbst verglichen, um zu ermitteln, ob sich die Features oder Envelopes überschneiden.

Die zweite SELECT-Anweisung verwendet einen Envelope, um zu ermitteln, ob Features in dem Envelope liegen, den Sie mit der WHERE-Klausel der SELECT-Anweisung übergeben.

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

Definition

ST_Equals vergleicht zwei Geometrien und gibt 1 (Oracle und SQLite) oder t (PostgreSQL) zurück, wenn die beiden Geometrien identisch sind. Andernfalls wird 0 (Oracle und SQLite) oder f (PostgreSQL) zurückgegeben.

Syntax

Oracle und PostgreSQL

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

SQLite

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

Rückgabetyp

Boolesch

Beispiel

Der GIS-Techniker der Stadt vermutet, dass einige Daten in der Tabelle "studies" dupliziert wurden. Um diese Bedenken auszuräumen, fragt er die Tabelle ab, um festzustellen, ob mehrere identische Shape-Multipolygone vorhanden sind.

Es wurde die Tabelle "studies" erstellt und mit den folgenden Anweisungen gefüllt. Die ID-Spalte identifiziert die Untersuchungsgebiete, und im Shape-Feld wird die Geometrie des Gebiets gespeichert.

Als nächstes wird die Tabelle "studies" durch das Equal-Prädikat räumlich mit sich selbst verknüpft. Dieses Prädikat gibt immer dann "1" (Oracle und SQLite) oder "t" (PostgreSQL) zurück, wenn zwei gleiche Multipolygone gefunden werden. Durch die Bedingung "s1.id<>s2.id " wird unterbunden, dass eine Geometrie mit sich selbst verglichen wird.

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

Hinweis:
Nur PostgreSQL

Definition

Mit ST_Equalsrs wird überprüft, ob zwei Raumbezugssysteme von zwei verschiedenen Feature-Classes identisch sind. Wenn die Raumbezugssysteme identisch sind, wird "t (true)" zurückgegeben. Wenn die Raumbezugssysteme nicht identisch sind, gibt ST_Equalsrs "f (false)" zurück.

Syntax

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

Rückgabetyp

Boolesch

Beispiel

In diesem Beispiel werden die Raumbezug-IDs (SRIDs) von verschiedenen Feature-Classes ermittelt. Anschließend wird mit ST_Equalsrs geprüft, ob die SRIDs dasselbe Raumbezugssystem darstellen.

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

srid	table_name
2	Straßen
6	transmains

Ergebnisse der Abfrage "sde_layers"

Anschließend wird mit ST_Equalsrs geprüft, ob die durch diese beiden SRIDs identifizierten Raumbezugssysteme identisch sind.

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

 st_equalsrs

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

f

(1 row)
```

ST_ExteriorRing

Definition

ST_ExteriorRing gibt den äußeren Ring eines Polygons als Linestring zurück.

Syntax

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

Oracle und PostgreSQL

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

SQLite

```
st_exteriorring (polygon1 geometryblob)
```

Rückgabetyp

ST_LineString

Beispiel

Eine Ornithologin möchte die Vogelpopulation auf einigen Inseln studieren und weiß, dass die Vogelarten, an denen sie interessiert ist, nur am Strand nach Nahrung suchen. Zur Berechnung der Tragfähigkeit der Inseln benötigt die Ornithologin jeweils den Inselumfang. Einige Inseln sind so groß, dass es einige Binnenseen darauf gibt. Allerdings wird der Strand dieser Seen ausschließlich von einer anderen aggressiveren Vogelart bewohnt. Daher benötigt die Ornithologin nur den Umfang des äußeren Inselrings.

Durch die Spalten "ID" und "name" der Tabelle "islands" werden die einzelnen Inseln eindeutig gekennzeichnet, und die Spalte "land" enthält die Geometrie der einzelnen Inseln.

Die Funktion ST_ExteriorRing extrahiert den äußeren Ring aus jedem Inselpolygon und gibt ihn als Linestring zurück. Die Länge des Linestring wird mit der Funktion ST_Length berechnet. Die einzelnen Linestring-Längen werden mit der Funktion SUM zusammengezählt.

Der äußere Ring der Inseln stellt die ökologische Schnittstelle dar, die jede Insel mit dem Meer gemeinsam hat.

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

Hinweis:
Nur Oracle und PostgreSQL

Definition

Mit ST_GeomCollection wird aus einer Well-known-Text-Repräsentation eine Geometriesammlung erstellt.

Syntax

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)
```

Rückgabetyp

ST_GeomCollection

Beispiel

Oracle

Erstellen Sie die Tabelle "geomcoll_test", und fügen Sie Geometrien ein.

```
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)
);
```

Wählen Sie in der Tabelle "geomcoll_test" die Geometriesammlung aus.

```
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

Erstellen Sie die Tabelle "geomcoll_test", und fügen Sie Geometrien ein.

```
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)
);
```

Wählen Sie in der Tabelle "geomcoll_test" die Geometriesammlung aus.

```
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

Hinweis:
Nur PostgreSQL

Definition

Mit ST_GeomCollFromWKB wird eine Geometrieerfassung aus einem Well-Known Binary-Format erstellt.

Syntax

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

Rückgabetyp

ST_GeomCollection

Beispiel

Hinweis:
Zur besseren Lesbarkeit werden Zeilenumbrüche eingefügt. Entfernen Sie diese, wenn Sie die Anweisungen kopieren.

Erstellen Sie die Tabelle "gcoll_test".

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

Geben Sie Werte in die Tabelle ein.

```
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));
```

Wählen Sie die Geometrie aus der Tabelle "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

Definition

Mit ST_Geometry wird aus einer Well-known-Text-Repräsentation eine Geometrie erstellt.

Hinweis:
Beim Erstellen von räumlichen Tabellen, die mit ArcGIS verwendet werden sollen, ist es am besten, die Spalte als übergeordneten Geometrietyp (z. B. ST_Geometry) zu erstellen, statt einen ST_Geometry-Subtype anzugeben.

Syntax

Oracle

- Für Linestrings, Polygone und Punkte

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

- Für optimierte Punkte (die keinen Extproc-Agenten starten und die Abfrage somit schneller verarbeiten)

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

  Verwenden Sie die Konstruktion mit optimierten Punkten, wenn Sie Batch-Einfügungen mit einer großen Anzahl von Punktdaten durchführen.

- Für parametrische Kreise

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

- Für parametrische Ellipsen

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

- Für parametrische Keile

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

PostgreSQL

- Für Linestrings, Polygone und Punkte

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

- Für parametrische Kreise

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

- Für parametrische Ellipsen

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

- Für parametrische Keile

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

SQLite

- Für Linestrings, Polygone und Punkte

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

- Für parametrische Kreise

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

- Für parametrische Ellipsen

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

- Für parametrische Keile

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

Rückgabetyp

Oracle und PostgreSQL

ST_Geometry

SQLite

Geometryblob

Beispiele

Erstellen und Abfragen von Punkt-, Linestring- und Polygon-Features

In diesen Beispielen wird eine Tabelle (geoms) erstellt, in die Punkt-, Linestring- und Polygonwerte eingefügt werden.

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)
);
```

Erstellen und Abfragen von parametrischen Kreisen

Erstellen Sie die Tabelle "radii", und fügen Sie Kreise ein.

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)
);
```

Erstellen und Abfragen von parametrischen Ellipsen

Erstellen Sie die Tabelle "track", und fügen Sie Ellipsen ein.

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)
);
```

Erstellen und Abfragen von parametrischen Keilen

Erstellen Sie die Tabelle "pwedge", und fügen Sie einen Keil ein.

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

Definition

ST_GeometryN akzeptiert eine Sammlung und einen ganzzahligen Index und gibt das n-te ST_Geometry-Objekt der Sammlung zurück.

Syntax

Oracle und 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)
```

Rückgabetyp

Oracle und PostgreSQL

ST_Geometry

SQLite

Geometryblob

Beispiel

In diesem Beispiel wird ein Multipolygon erstellt. Dann wird mithilfe von ST_GeometryN das zweite Element des Multipolygons aufgelistet.

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

Definition

"ST_GeometryType" gibt für ein Geometrieobjekt den Geometrietyp (z. B. Punkt, Linie, Polygon, Multipoint) als Zeichenfolge zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_geometrytype (g1 geometryblob)
```

Rückgabetyp

Varchar(32) (Oracle und PostgreSQL) oder Text (SQLite) mit einer der folgenden Zeichenfolgen:

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

Beispiel

Die Tabelle "geometrytype_test" enthält die Geometriespalte "g1".

Die INSERT-Anweisungen fügen die einzelnen Geometrie-Subclasses in die Spalte "g1" ein.

Die SELECT-Abfrage listet den Geometrietyp der einzelnen Subclasses, die in der Geometriespalte "g1" gespeichert sind, auf.

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

Hinweis:
nur PostgreSQL

Definition

"ST_GeomFromCollection" gibt einen Satz an "st_geometry"-Zeilen zurück. Jede Zeile besteht aus einer Geometrie und einer ganzen Zahl. Die ganze Zahl gibt die Position der Geometrie im Satz wieder.

Mit der Funktion "ST_GeomFromCollection" können Sie auf jede einzelne Geometrie in einer Multipart-Geometrie zugreifen. Wenn die Eingabe-Geometrie eine Sammlung oder eine Multipart-Geometrie ist (z. B. ST_MultiLineString, ST_MultiPoint, ST_MultiPolygon), gibt "ST_GeomFromCollection" einen Datensatz für jede Sammlungskomponente zurück, und der Pfad gibt die Position der Komponente in der Sammlung wieder.

Wenn Sie "ST_GeomFromCollection" bei einer einfachen Geometrie verwenden (z. B. ST_Point, ST_LineString, ST_Polygon), wird ein einfacher Datensatz mit einem leeren Pfad zurückgegeben, da es nur eine Geometrie gibt.

Syntax

`sde.st_geomfromcollection (shape sde.st_geometry)`

Verwenden Sie `(sde.st_geomfromcollection (shape)).st_geo`, um nur die Geometrie zurückgeben zu lassen.

Verwenden Sie `(sde.st_geomfromcollection (shape)).path[1]`, um nur die Position der Geometrie zurückgeben zu lassen.

Rückgabetyp

ST_Geometry Satz

Beispiel

Erstellen Sie in diesem Beispiel eine Multiline-Feature-Class (ghanasharktracks) mit einem Feature, das ein vierteiliges Shape umfasst.

```
--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
  )
 );
```

Um sicherzustellen, dass das Feld Daten enthält, führen Sie eine Abfrage für die Tabelle durch. Verwenden Sie "ST_AsText" direkt für das Shape-Feld, um die Shape-Koordinaten als Text anzeigen zu lassen. Dabei wird die Textbeschreibung des Multilinestrings zurückgegeben.

```
--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))"
```

Um jede Linestring-Geometrie einzeln zurückgeben zu lassen, müssen Sie die Funktion "ST_GeomFromCollection" verwenden. Um die Geometrie als Text anzuzeigen, wird in diesem Beispiel die Funktion "ST_AsText" mit der Funktion "ST_GeomFromCollection" verwendet.

```
--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

Hinweis:
Wird nur in Oracle und SQLite verwendet; verwenden Sie für PostgreSQL ST_Geometry.

Definition

Mit ST_GeomFromText wird anhand einer Repräsentation im WKT-Format (Well-Known Text) und einer Raumbezugs-ID ein Geometrieobjekt zurückgegeben.

Syntax

Oracle

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

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

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

SQLite

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

```
st_geomfromtext (wkt text)
```

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

Rückgabetyp

Oracle

ST_Geometry

SQLite

Geometryblob

Beispiel

Die Tabelle "geometry_test" enthält die Spalte "integer gid", mit der die einzelnen Zeilen und die Spalte "g1", in der die Geometrie gespeichert wird, eindeutig gekennzeichnet werden.

Mit der INSERT-Anweisung werden die Daten in die Spalten "gid" und "g1" der Tabelle "geometry_test" eingefügt. Mit der ST_GeomFromText-Funktion wird das Textformat der einzelnen Geometrien in die entsprechende instanziierbare Subclass konvertiert. Mit der SELECT-Anweisung am Ende wird sichergestellt, dass die Daten in die Spalte "g1" eingefügt wurden.

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

Definition

Mit ST_GeomFromWKB wird anhand einer Repräsentation im WKB-Format (Well-Known Binary) und einer Raumbezugs-ID ein Geometrieobjekt zurückgegeben.

Syntax

Oracle

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

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

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 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)
```

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

Rückgabetyp

Oracle und PostgreSQL

ST_Geometry

SQLite

Geometryblob

Beispiel

Im folgenden Beispiel wurden die Zeilen der Ergebnisse zur besseren Lesbarkeit neu formatiert. Die Abstände der Ihnen angezeigten Ergebnisse können abhängig von Ihrer Online-Darstellung variieren. Mit dem folgenden Code

wird verdeutlicht, wie die ST_GeomFromWKB-Funktion verwendet werden kann, um eine Linie aus einer WKB-Linien-Repräsentation zu erstellen und einzufügen. Im folgenden Beispiel wird ein Datensatz in die Tabelle "sample_gs" mit einer ID und einer Geometrie im Raumbezugssystem 4326 in einem WKB-Format eingefügt.

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

Hinweis:
Nur PostgreSQL

Definition

Mit ST_GeoSize wird anhand eines ST_Geometry-Objekts seine Größe in Byte zurückgegeben.

Syntax

```
st_geosize (st_geometry)
```

Rückgabetyp

Ganzzahl

Beispiel

Sie können die Größe der im Beispiel ST_GeomFromWKB erstellten Features ermitteln, indem Sie die Geometriespalte der Tabelle "sample_geometries" abfragen.

```
SELECT st_geosize (geometry)
FROM sample_geometries;

st_geosize
        512
        592
        616
```

ST_InteriorRingN

Definition

ST_InteriorRingN gibt den n-ten inneren Ring eines Polygons als ST_LineString-Objekt zurück.

Die Reihenfolge der Ringe kann nicht vordefiniert werden, da die Ringe nach den durch die internen Geometrieprüfroutinen definierten Regeln und nicht nach der geometrischen Ausrichtung angeordnet werden. Wenn der Index größer als die Anzahl der in einem Polygon enthaltenen inneren Ringe ist, wird ein Nullwert zurückgegeben.

Syntax

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)
```

Rückgabetyp

ST_LineString

Beispiel

Erstellen Sie eine Tabelle mit dem Namen "sample_polys", und fügen Sie einen Datensatz hinzu, und wählen Sie anschließend die ID und die Geometrie des inneren Rings aus.

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

Definition

ST_Intersection wählt zwei Geometrieobjekte aus und gibt die Schnittmenge als zweidimensionales Geometrieobjekt zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

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

Rückgabetyp

Oracle und PostgreSQL

ST_Geometry

SQLite

Geometryblob

Beispiel

Der Kreisbrandmeister muss die Flächen der Krankenhäuser, Schulen und Pflegeheime ermitteln, die sich mit dem Radius einer möglichen Kontamination durch Sondermüll überschneiden.

Die Daten zu Schulen, Krankenhäusern und Pflegeheimen werden in der Tabelle "population" gespeichert, die mit der folgenden CREATE TABLE-Anweisung erstellt wird. Die als Polygon definierte Spalte "shape" enthält den Umriss der einzelnen empfindlichen Bereiche.

Die Sondermülldeponien werden in der Tabelle "waste_sites" gespeichert, die mit der folgenden CREATE TABLE-Anweisung erstellt wird. Die als Punkte definierte Spalte "site" enthält eine Position, die den geographischen Mittelpunkt der einzelnen Sondermülldeponien darstellt.

Die Funktion ST_Buffer generiert einen Puffer um die Sondermülldeponien. Die Funktion ST_Intersection erzeugt Polygone von der Überschneidung der gepufferten Sondermülldeponien mit den empfindlichen Bereichen.

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

Definition

ST_Intersects gibt 1 (Oracle und SQLite) oder t (PostgreSQL) zurück, wenn die Überschneidung von zwei Geometrien keine leere Menge ergibt. Andernfalls wird 0 (Oracle und SQLite) oder f (PostgreSQL) zurückgegeben.

Syntax

Oracle und PostgreSQL

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

SQLite

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

Rückgabetyp

Boolesch

Beispiel

Der Kreisbrandmeister möchte eine Liste der empfindlichen Bereiche, die innerhalb des Radius einer Sondermülldeponie liegen.

Die empfindlichen Bereiche werden in der Tabelle "sensitive_areas" gespeichert. Die als Polygon definierte Spalte "shape" enthält den Umriss der einzelnen empfindlichen Bereiche.

Die Sondermülldeponien werden in der Tabelle "hazardous_sites" gespeichert. Die als Punkte definierte Spalte "site" enthält eine Position, die den geographischen Mittelpunkt der einzelnen Sondermülldeponien darstellt.

Die SELECT-Abfrage erstellt einen Pufferradius um jede Sondermülldeponie und gibt eine Liste der empfindlichen Bereiche zurück, die den Pufferradius der Sondermülldeponien schneiden.

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

Definition

"ST_Is3d" akzeptiert als Eingabeparameter ein ST_Geometry-Objekt. Wenn die angegebene Geometrie Z-Koordinaten aufweist, gibt die Funktion 1 (Oracle und SQLite) oder t (PostgreSQL) zurück, andernfalls gibt sie 0 (Oracle und SQLite) oder f (PostgreSQL) zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_is3d (geometry1 geometryblob)
```

Rückgabetyp

Boolesch

Beispiel

In diesem Beispiel wird die Tabelle "is3d_test" erstellt und mit Datensätzen befüllt.

Als Nächstes wird mit "ST_Is3d" überprüft, ob einer der Datensätze eine Z-Koordinate aufweist.

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

Definition

ST_IsClosed wählt ein ST_LineString- oder ST_MultiLineString-Objekt aus und gibt 1 (Oracle und SQLite) bzw. t (PostgreSQL) zurück, wenn es geschlossen ist. Andernfalls wird 0 (Oracle und SQLite) oder f (PostgreSQL) zurückgegeben.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_isclosed (geometry1 geometryblob)
```

Rückgabetyp

Boolesch

Beispiele

Testen eines Linestring

Die Tabelle "closed_linestring" wird mit der Spalte "linestring" erstellt.

Mit der INSERT-Anweisung werden zwei Datensätze in die Tabelle "closed_linestring" eingefügt. Der erste Datensatz enthält keinen geschlossenen Linestring, der zweite Datensatz dagegen schon.

Die Abfrage gibt die Ergebnisse der Funktion ST_IsClosed zurück. In der ersten Zeile wird 0 oder f zurückgegeben, weil der Linestring nicht geschlossen ist. In der zweiten Zeile wird dagegen 1 oder t zurückgegeben, weil der Linestring geschlossen ist.

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
```

Testen eines Multilinestring

Die Tabelle "closed_mlinestring" wird mit einer ST_MultiLineString-Spalte erstellt.

Mit der INSERT-Anweisung werden ein Datensatz für einen nicht geschlossenen Multilinestring und ein Datensatz für einen geschlossenen Multilinestring in die ST_MultiLineString-Spalte eingefügt.

Die Abfrage listet die Ergebnisse der Funktion "ST_IsClosed" auf. In der ersten Zeile wird 0 oder f zurückgegeben, weil der Multilinestring nicht geschlossen ist. In der zweiten Zeile wird 1 oder t zurückgegeben, weil der in der Spalte "ln1" gespeicherte Multilinestring geschlossen ist. Ein Multilinestring ist geschlossen, wenn alle seine Linestring-Elemente geschlossen sind.

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

Definition

ST_IsEmpty gibt 1 (Oracle und SQLite) oder t (PostgreSQL) zurück, wenn das ST_Geometry-Objekt leer ist. Andernfalls wird 0 (Oracle und SQLite) oder f (PostgreSQL) zurückgegeben.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_isempty (geometry1 geometryblob)
```

Rückgabetyp

Boolesch

Beispiel

Mit der folgenden CREATE TABLE-Anweisung wird die Tabelle "empty_test" mit der Spalte "geotype" erstellt. In der Spalte "geotype" ist der Datentyp der Subclasses gespeichert ist, die in der Spalte "g1" gespeichert sind.

Mit der INSERT-Anweisung werden zwei Datensätze für die Geometrie-Subclasses "point", "linestring" und "polygon" eingefügt: Ein Datensatz ist leer, der andere nicht.

Die SELECT-Abfrage gibt den Geometrietyp aus der Spalte "geotype" und die Ergebnisse der Funktion "ST_IsEmpty" zurück.

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

Definition

"ST_IsMeasured" akzeptiert als Eingabeparameter ein Geometrieobjekt. Wenn die angegebene Geometrie Messwerte aufweist, gibt die Funktion 1 (Oracle und SQLite) oder t (PostgreSQL) zurück, andernfalls gibt sie 0 (Oracle und SQLite) oder f (PostgreSQL) zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_ismeasured (geometry1 geometryblob)
```

Rückgabetyp

Boolesch

Beispiel

Erstellt die Tabelle "ism_test", fügt Werte darin ein und ermittelt dann, welche Zeilen der Tabelle "ism_test" Messwerte enthalten.

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

Definition

ST_IsRing wählt ein ST_LineString-Objekt aus und gibt 1 (Oracle und SQLite) bzw. t (PostgreSQL) zurück, wenn es ein Ring ist (beispielsweise wenn das ST_LineString-Objekt geschlossen oder einfach ist). Andernfalls wird 0 (Oracle und SQLite) oder f (PostgreSQL) zurückgegeben.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_isring (line1 geometryblob)
```

Rückgabetyp

Boolesch

Beispiel

Die Tabelle "ring_linestring" wird mit der Spalte "ln1" vom Typ ST_LineString erstellt.

Mit der INSERT-Anweisung werden drei Linestrings in die Spalte "ln1" eingefügt. Die erste Zeile enthält einen Linestring, der nicht geschlossen und kein Ring ist. Die zweite Zeile enthält einen geschlossenen, einfachen Linestring, der ein Ring ist. Die dritte Zeile enthält einen Linestring, der geschlossen, aber nicht einfach ist, weil er seinen eigenen Innenbereich schneidet. Es ist also kein Ring.

Die SELECT-Abfrage gibt die Ergebnisse der Funktion "ST_IsRing" zurück. In der ersten Zeile wird 0 oder f zurückgegeben, weil die Linestrings keine Ringe sind. In der zweiten und dritten Zeile wird dagegen 1 oder t zurückgegeben, weil es Ringe sind.

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

Definition

ST_IsSimple gibt 1 (Oracle und SQLite) oder t (PostgreSQL) zurück, wenn das Geometrieobjekt nach der Definition des Open Geospatial Consortium (OGC) einfach ist. Andernfalls wird 0 (Oracle und SQLite) oder f (PostgreSQL) zurückgegeben.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_issimple (geometry1 geometryblob)
```

Rückgabetyp

Boolesch

Beispiel

Die Tabelle "issimple_test" wird mit zwei Spalten erstellt. Die Spalte "pid" ist vom Datentyp smallint und enthält die eindeutige Kennung der einzelnen Zeilen. In der Spalte "g1" werden die einfachen und nicht einfachen Geometriebeispiele gespeichert.

Mit der INSERT-Anweisung werden zwei Datensätze in die Tabelle "issimple_test" eingefügt. Der erste Datensatz enthält einen einfachen Linestring, weil er seinen Innenbereich nicht schneidet. Der zweite Datensatz ist nach der OGC-Definition nicht einfach, weil er seinen Innenbereich schneidet.

Die Abfrage gibt die Ergebnisse der Funktion ST_IsSimple zurück. Für den ersten Datensatz wird 1 oder t zurückgegeben, weil der Linestring einfach ist. Für den zweiten Datensatz wird dagegen 0 oder f zurückgegeben, weil der Linestring nicht einfach ist.

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

Definition

ST_Length gibt die Länge eines Linestring oder Multilinestring zurück.

Syntax

Oracle und 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)
```

Eine Liste der unterstützten Einheitennamen finden Sie unter ST_Distance.

Rückgabetyp

Doppelte Genauigkeit

Beispiel

Ein Ökologe, der die Migrationsmuster der Lachspopulation in den Gewässern des Landes studiert, möchte die Länge aller Strom- und Flusssysteme des Landes ermitteln.

Die Tabelle "waterways" wird mit den Spalten "ID" und "name" erstellt, welche die einzelnen in der Tabelle gespeicherten Strom- und Flusssysteme eindeutig kennzeichnen. Die Spalte "water" enthält Multilinestrings, weil die Strom- und Flusssysteme häufig Aggregate von mehreren Linestrings sind.

Die SELECT-Abfrage gibt den Namen des Systems sowie die von der Funktion ST_Length generierte Länge des jeweiligen Systems aus. In Oracle und PostgreSQL werden die Einheiten des Koordinatensystems verwendet. In SQLite werden Kilometereinheiten angegeben.

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

Hinweis:
Wird nur in Oracle und SQLite unterstützt; verwenden Sie für PostgreSQL ST_LineString.

Definition

Mit ST_LineFromText wird anhand einer Repräsentation im WKT-Format (Well-Known Text) vom Typ ST_LineString und einer Raumbezugs-ID ein ST_LineString-Objekt zurückgegeben.

Syntax

Oracle

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

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

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

SQLite

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

```
st_linefromtext (wkt text)
```

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

Rückgabetyp

ST_LineString

Beispiel

Die Tabelle "linestring_test" wird mit der Spalte "ln1" vom Typ ST_LineString erstellt.

Mit der INSERT-Anweisung wird mit der Funktion ST_LineFromText ein ST_LineString-Objekt in die Spalte "ln1" eingefügt.

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

Definition

Mit ST_LineFromWKB wird anhand einer Repräsentation im WKB-Format (Well-Known Binary) vom Typ ST_LineString und einer Raumbezugs-ID ein ST_LineString-Objekt zurückgegeben.

Syntax

Oracle

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

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

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

PostgreSQL

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

SQLite

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

```
st_linefromwkb (wkb blob)
```

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

Rückgabetyp

ST_LineString

Beispiel

Die folgenden Befehle erstellen eine Tabelle (samplel_lines) und verwenden die Funktion "ST_LineFromWKB", um Zeilen aus einer WKB-Repräsentation einzufügen. Die Zeile wird in die Tabelle "sample_lines" mit einer ID und einer Linie im Raumbezugssystem 4326 im WKB-Format eingefügt.

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

Definition

ST_LineString ist eine Accessor-Funktion, mit der aus einer Well-known-Text-Repräsentation ein Linestring erstellt wird.

Hinweis:
Beim Erstellen von räumlichen Tabellen, die mit ArcGIS verwendet werden sollen, ist es am besten, die Spalte als übergeordneten Geometrietyp (z. B. ST_Geometry) zu erstellen, statt einen ST_Geometry-Subtype anzugeben.

Syntax

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)
```

Rückgabetyp

ST_LineString

Beispiel

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

Definition

ST_M akzeptiert einen Eingabeparameter vom Typ ST_Point und gibt dessen Messwert-Koordinate (M) zurück.

In SQLite kann ST_M auch zum Aktualisieren von Messwerten verwendet werden.

Syntax

Oracle und PostgreSQL

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

SQLite

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

Rückgabetyp

Oracle und PostgreSQL

Zahl

SQLite

Doppelte Genauigkeit beim Abfragen eines Messwertes, geometryblob beim Aktualisieren eines Messwertes.

Beispiele

Oracle

Die Tabelle "m_test" wird erstellt und es werden drei Punkte in sie eingefügt. Alle drei Punkte enthalten Messwerte. Eine SELECT-Anweisung wird mit der ST_M-Funktion ausgeführt, um die Messwerte für jeden Punkt zurückzugeben.

```
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

Die Tabelle "m_test" wird erstellt und es werden drei Punkte in sie eingefügt. Alle drei Punkte enthalten Messwerte. Eine SELECT-Anweisung wird mit der ST_M-Funktion ausgeführt, um die Messwerte für jeden Punkt zurückzugeben.

```
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

Im ersten Beispiel wird die Tabelle "m_test" erstellt, in die drei Punkte eingefügt werden. Alle drei Punkte enthalten Messwerte. Eine SELECT-Anweisung wird mit der ST_M-Funktion ausgeführt, um die Messwerte für jeden Punkt zurückzugeben.

```
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
```

Im zweiten Beispiel wird der Messwert für Datensatz 3 in der Tabelle "m_test" aktualisiert.

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

ST_MaxM

Definition

ST_MaxM akzeptiert eine Geometrie als Eingabeparameter und gibt dessen maximale M-Koordinate zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_maxm (geometry1 geometryblob)
```

Rückgabetyp

Oracle und PostgreSQL

Zahl

Wenn keine M-Werte vorhanden sind, wird NULL zurückgegeben.

SQLite

Doppelte Genauigkeit

Wenn keine M-Werte vorhanden sind, wird NULL zurückgegeben.

Beispiel

Die Tabelle "maxm_test" wird erstellt und es werden zwei Polygone in sie eingefügt. Anschließend wird "ST_MaxM" ausgeführt, um den maximalen M-Wert in jedem Polygon zu bestimmen.

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

Definition

ST_MaxX akzeptiert eine Geometrie als Eingabeparameter und gibt dessen maximale X-Koordinate zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_maxx (geometry1 geometryblob)
```

Rückgabetyp

Oracle und PostgreSQL

Zahl

SQLite

Doppelte Genauigkeit

Beispiel

Die Tabelle "maxx_test" wird erstellt und es werden zwei Polygone in sie eingefügt. Anschließend wird mit der Funktion "ST_MaxX" der maximale X-Wert in jedem Polygon bestimmt.

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

Definition

ST_MaxY akzeptiert eine Geometrie als Eingabeparameter und gibt dessen maximale Y-Koordinate zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_maxy (geometry1 geometryblob)
```

Rückgabetyp

Oracle und PostgreSQL

Zahl

SQLite

Doppelte Genauigkeit

Beispiel

Die Tabelle "maxy_test" wird erstellt und es werden zwei Polygone in sie eingefügt. Anschließend wird mit der Funktion "ST_MaxY" der maximale Y-Wert in jedem Polygon bestimmt.

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

Definition

ST_MaxZ akzeptiert eine Geometrie als Eingabeparameter und gibt dessen maximale Z-Koordinate zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_maxz (geometry1 geometryblob)
```

Rückgabetyp

Oracle und PostgreSQL

Zahl

Wenn keine Z-Werte vorhanden sind, wird NULL zurückgegeben.

SQLite

Doppelte Genauigkeit

Wenn keine Z-Werte vorhanden sind, wird NULL zurückgegeben.

Beispiel

Im folgenden Beispiel wird die Tabelle "maxz_test" erstellt, in die zwei Polygone eingefügt werden. Anschließend wird "ST_MaxZ" ausgeführt, um den maximalen Z-Wert für jedes Polygon zu bestimmen.

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

Definition

ST_MinM akzeptiert eine Geometrie als Eingabeparameter und gibt dessen minimale M-Koordinate zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_minm (geometry1 geometryblob)
```

Rückgabetyp

Oracle und PostgreSQL

Zahl

Wenn keine M-Werte vorhanden sind, wird NULL zurückgegeben.

SQLite

Doppelte Genauigkeit

Wenn keine M-Werte vorhanden sind, wird NULL zurückgegeben.

Beispiel

Die Tabelle "minm_test" wird erstellt und es werden zwei Polygone in sie eingefügt. Anschließend wird "ST_MinM" ausgeführt, um den minimalen Messwert in jedem Polygon zu bestimmen.

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

Definition

ST_MinX akzeptiert eine Geometrie als Eingabeparameter und gibt dessen minimale X-Koordinate zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_minx (geometry1 geometryblob)
```

Rückgabetyp

Oracle und PostgreSQL

Zahl

SQLite

Doppelte Genauigkeit

Beispiel

Die Tabelle "minx_test" wird erstellt und es werden zwei Polygone in sie eingefügt. Anschließend wird "ST_MinX" ausgeführt, um den kleinsten X-Koordinatenwert in jedem Polygon zu bestimmen.

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

Definition

ST_MinY akzeptiert eine Geometrie als Eingabeparameter und gibt dessen minimale Y-Koordinate zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_miny (geometry1 geometryblob)
```

Rückgabetyp

Oracle und PostgreSQL

Zahl

SQLite

Doppelte Genauigkeit

Beispiel

Die Tabelle "miny_test" wird erstellt und es werden zwei Polygone in sie eingefügt. Anschließend wird "ST_MinY" ausgeführt, um den kleinsten Y-Koordinatenwert in jedem Polygon zu bestimmen.

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

Definition

ST_MinZ akzeptiert eine Geometrie als Eingabeparameter und gibt dessen minimale Z-Koordinate zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_minz (geometry1 geometryblob)
```

Rückgabetyp

Oracle und PostgreSQL

Zahl

Wenn keine Z-Werte vorhanden sind, wird NULL zurückgegeben.

SQLite

Doppelte Genauigkeit

Wenn keine Z-Werte vorhanden sind, wird NULL zurückgegeben.

Beispiel

Die Tabelle "minz_test" wird erstellt und es werden zwei Polygone in sie eingefügt. Anschließend wird "ST_MinZ" ausgeführt, um den kleinsten Z-Koordinatenwert in jedem Polygon zu bestimmen.

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

Hinweis:
Wird nur in Oracle und SQLite verwendet; verwenden Sie für PostgreSQL ST_MultiLineString.

Definition

Mit ST_MLineFromText wird anhand einer Repräsentation im WKT-Format (Well-Known Text) vom Typ ST_MultiLineString und einer Raumbezugs-ID ein ST_MultiLineString-Objekt zurückgegeben.

Syntax

Oracle

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

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

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

SQLite

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

```
st_mlinefromtext (wkt text)
```

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

Rückgabetyp

ST_MultiLineString

Beispiel

Die Tabelle "mlinestring_test" wird mit der Spalte "gid", durch die die einzelnen Zeilen eindeutig gekennzeichnet werden, und die Spalte "ml1" vom Typ "ST_MultiLineString" erstellt.

Mit der INSERT-Anweisung wird mit der Funktion ST_MLineFromText ein ST_MultiLineString-Objekt eingefügt.

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

Definition

Mit ST_MLineFromWKB wird anhand einer Repräsentation im WKB-Format (Well-Known Binary) vom Typ ST_MultiLineString und einer Raumbezugs-ID ein ST_MultiLineString-Objekt zurückgegeben.

Syntax

Oracle

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

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

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

PostgreSQL

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

SQLite

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

```
st_mlinefromwkb (wkb blob)
```

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

Rückgabetyp

ST_MultiLineString

Beispiel

In diesem Beispiel wird veranschaulicht, wie ST_MLineFromWKB zum Erstellen eines Mehrfachlinienzugs aus seiner WKB-Repräsentation verwendet werden kann. Bei der Geometrie handelt es sich um einen Mehrfachlinienzug im Raumbezugssystem 4326. In diesem Beispiel wird der Mehrfachlinienzug mit der ID = 10 in der Spalte "geometry" der Tabelle "sample_mlines" gespeichert. Anschließend wird die Spalte "wkb" mit der Repräsentation im WKB-Format (Well-Known Binary) (mithilfe der Funktion "ST_AsBinary") aktualisiert. Zum Schluss wird mit der ST_MLineFromWKB-Funktion der Multilinestring aus der Spalte "wbk" zurückgegeben. Die Tabelle "sample_mlines" verfügt über die Spalte "geometry", in der der Multilinestring gespeichert wird, sowie über die Spalte "wbk", in der die WBK-Repräsentation des Multilinestrings gespeichert wird.

Die SELECT-Anweisung enthält die ST_MLineFromWKB-Funktion, mit der Multilinestring aus der Spalte "WBK" abgerufen wird.

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

Hinweis:
Nur Oracle und SQLite; für PostgreSQL wird ST_MultiPoint verwendet.

Definition

Mit ST_MPointFromText wird anhand einer Repräsentation im WKT-Format (Well-Known Text) vom Typ ST_MultiPoint und einer Raumbezugs-ID ein ST_Multipoint-Objekt erstellt.

Syntax

Oracle

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

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

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

SQLite

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

```
st_mpointfromtext (wkt text)
```

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

Rückgabetyp

ST_MultiPoint

Beispiel

Die Tabelle "multipoint_test" wird mit der Spalte "mpt1" vom Typ ST_MultiPoint erstellt.

Mit der INSERT-Anweisung wird mit der Funktion ST_MpointFromText ein Multipoint in die Spalte "mpt1" eingefügt.

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

Definition

Mit "ST_MPointFromText" wird anhand einer Repräsentation im WKB-Format (Well-Known Binary) vom Typ "ST_MultiPoint" und einer Raumbezugs-ID ein ST_MultiPoint-Objekt erstellt.

Syntax

Oracle

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

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

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

PostgreSQL

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

SQLite

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

```
st_mpointfromwkb (wkb blob)
```

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

Rückgabetyp

ST_MultiPoint

Beispiel

In diesem Beispiel wird veranschaulicht, wie ST_MPointFromWKB zum Erstellen eines Multipoints aus seiner WKB-Repräsentation (Well-Known Binary) verwendet werden kann. Bei der Geometrie handelt es sich um einen Multipoint im Raumbezugssystem 4326. In diesem Beispiel wird der Multipoint mit der ID = 10 in der Spalte "GEOMETRY" der Tabelle "SAMPLE_MPOINTS" gespeichert. Anschließend wird die Spalte "WKB" mit der Repräsentation im WKB-Format (Well-Known Binary) (mithilfe der Funktion "ST_AsBinary") aktualisiert. Zum Schluss wird mit der ST_MPointFromWKB-Funktion der Multipoint aus der Spalte "WBK" zurückgegeben. Die Tabelle "SAMPLE_MPOINTS" verfügt über die Spalte "GEOMETRY", in der der Multipoint gespeichert wird, sowie über die Spalte "WBK", in der die Repräsentation im WKB-Format (Well-Known Binary) des Multipoints gespeichert wird.

In der folgenden SELECT-Anweisung wird mit der ST_MPointFromWKB-Funktion der Multipoint aus der Spalte "WBK" abgerufen.

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

Hinweis:
Nur Oracle und SQLite; für PostgreSQL wird ST_MultiPolygon verwendet.

Definition

Mit ST_MPolyFromText wird anhand einer Repräsentation im WKT-Format (Well-Known Text) vom Typ ST_MultiPolygon und einer Raumbezugs-ID ein ST_MultiPolygon-Objekt zurückgegeben.

Syntax

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 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)
```

Rückgabetyp

ST_MultiPolygon

Beispiel

Die Tabelle "multipolygon_test" wird mit der ST_MultiPolygon-Spalte "mpt1" erstellt.

Mit der INSERT-Anweisung wird mithilfe der Funktion "ST_MpolyFromText" ein ST_MultiPolygon in die Spalte "mpt1" eingefügt.

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

Definition

Mit ST_MPointFromWKB wird anhand einer Repräsentation im WKB-Format (Well-Known Binary) vom Typ ST_MultiPolygon und einer Raumbezugs-ID ein ST_MultiPolygon-Objekt zurückgegeben.

Syntax

Oracle

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

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

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

PostgreSQL

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

SQLite

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

```
st_mpolyfromwkb (wkb blob)
```

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

Rückgabetyp

ST_MultiPolygon

Beispiel

In diesem Beispiel wird veranschaulicht, wie ST_MPolyFromWKB zum Erstellen eines Multipolygons aus seiner WKB-Repräsentation (Well-Known Binary) verwendet werden kann. Bei der Geometrie handelt es sich um ein Multipolygon im Raumbezugssystem 4326. In diesem Beispiel wird das Multipolygon mit der ID = 10 in der Spalte "geometry" der Tabelle "sample_mpolys" gespeichert. Anschließend wird die Spalte "wkb" mit der Repräsentation im WKB-Format (Well-Known Binary) (mithilfe der Funktion "ST_AsBinary") aktualisiert. Zum Schluss wird mit der ST_MPolyFromWKB-Funktion das Multipolygon aus der Spalte "wbk" zurückgegeben. Die Tabelle "sample_mpolys" verfügt über die Spalte "geometry", in der das Multipolygon gespeichert wird, sowie über die Spalte "wbk", in der die WBK-Repräsentation des Multipolygons gespeichert wird.

Die SELECT-Anweisung enthält die ST_MPolyFromWKB-Funktion, mit der das Multipolygon aus der Spalte "WBK" abgerufen wird.

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

Hinweis:
Nur Oracle

Definition

Mit ST_MultiCurve wird aus einer Well-known-Text-Repräsentation ein Multicurve-Feature erstellt.

Syntax

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

Rückgabetyp

ST_MultiLinestring

Beispiel

```
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

Definition

Mit ST_MultiLineString wird aus einer Well-known-Text-Repräsentation ein Multilinestring erstellt.

Hinweis:
Beim Erstellen von räumlichen Tabellen, die mit ArcGIS verwendet werden sollen, ist es am besten, die Spalte als übergeordneten Geometrietyp (z. B. ST_Geometry) zu erstellen, statt einen ST_Geometry-Subtype anzugeben.

Syntax

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)
```

Rückgabetyp

ST_MultiLineString

Beispiel

Es wird die Tabelle "mlines_test" erstellt, in die mithilfe der Funktion "ST_MultiLineString" eine Multiline eingefügt wird.

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

Definition

Mit ST_MultiPolygon wird aus einer Well-known-Text-Repräsentation ein Multipoint-Feature erstellt.

Hinweis:
Beim Erstellen von räumlichen Tabellen, die mit ArcGIS verwendet werden sollen, ist es am besten, die Spalte als übergeordneten Geometrietyp (z. B. ST_Geometry) zu erstellen, statt einen ST_Geometry-Subtype anzugeben.

Syntax

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)
```

Rückgabetyp

ST_MultiPoint

Beispiel

Es wird die Tabelle "mpoint_test" erstellt, in die mithilfe der Funktion "ST_MultiPoint" ein Multipoint eingefügt wird.

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

Definition

Mit ST_MultiPolygon wird aus einer Well-known-Text-Repräsentation ein Multipolygon-Feature erstellt.

Hinweis:
Beim Erstellen von räumlichen Tabellen, die mit ArcGIS verwendet werden sollen, ist es am besten, die Spalte als übergeordneten Geometrietyp (z. B. ST_Geometry) zu erstellen, statt einen ST_Geometry-Subtype anzugeben.

Syntax

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)
```

Rückgabetyp

ST_MultiPolygon

Beispiel

Es wird die Tabelle "mpoly_test" erstellt, in die mithilfe der Funktion "ST_MultiPolygon" ein Multipolygon eingefügt wird.

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

Hinweis:
Nur Oracle

Definition

Mit ST_MultiSurface wird aus einer Well-known-Text-Repräsentation ein Multisurface-Feature erstellt.

Syntax

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

Rückgabetyp

ST_MultiSurface

Beispiel

```
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

Definition

"ST_NumGeometries" gibt für eine Geometriesammlung die Anzahl an Geometrien in der Sammlung zurück.

Syntax

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)
```

Rückgabetyp

Integer

Beispiel

Im folgenden Beispiel wird eine Tabelle namens "sample_numgeom" erstellt. Es werden ein Multipolygon und ein Multipoint eingefügt. In der SELECT-Anweisung wird mit der Funktion "ST_NumGeometries" die Anzahl der Geometrien (Features) in den einzelnen Geometrien ermittelt.

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

Definition

ST_NumInteriorRing wählt ein ST_Polygon aus und gibt die Anzahl der darin enthaltenen inneren Ringe zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_numinteriorring (polygon1 geometryblob)
```

Rückgabetyp

Ganzzahl

Beispiel

Eine Ornithologin möchte die Vogelpopulation auf mehreren Südseeinseln studieren. Sie möchte herausfinden, welche Inseln einen oder mehrere Binnenseen enthalten, weil die Vogelart, an der sie interessiert ist, nur an Süßwasserseen Nahrung sucht.

Durch die Spalten "ID" und "name" der Tabelle "islands" werden die einzelnen Inseln eindeutig gekennzeichnet, und die ST_Polygon-Spalte "land" enthält die Geometrie der einzelnen Inseln.

Weil die inneren Ringe die Seen darstellen, listet die SELECT-Anweisung, die die ST_NumInteriorRing-Funktion enthält, nur diejenigen Inseln auf, die über mindestens einen inneren Ring verfügen.

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

Definition

ST_NumPoints gibt die Anzahl von Punkten (Stützpunkten) in einer Geometrie zurück.

Bei Polygonen werden sowohl die Anfangs- und Endstützpunkte gezählt, obwohl sie sich an derselben Position befinden.

Beachten Sie, dass sich diese Anzahl vom Attribut NUMPTS des Typs ST_Geometry unterscheidet. Das Attribut NUMPTS enthält die Anzahl der Stützpunkte in allen Teilen der Geometrie, einschließlich der Trennzeichen zwischen Teilen. Es gibt einen Trennzeichen zwischen jedem Teil. Beispielsweise verfügt ein mehrteiliger Linestring mit drei Teilen über zwei Trennzeichen. Im Attribut NUMPTS wird jedes Trennzeichen als Stützpunkt gezählt. Die Funktion ST_NumPoints berücksichtigt dagegen die Trennzeichen nicht in der Anzahl der Stützpunkte.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_numpoints (geometry1 geometryblob)
```

Rückgabetyp

Ganzzahl

Beispiel

Die Tabelle "numpoints_test" wird mit der Spalte "geotype" erstellt, die den in der g1-Spalte gespeicherten Geometrietyp enthält.

Mit der INSERT-Anweisung werden ein Punkt, ein Linestrings und ein Polygon in die Tabelle eingefügt.

Die SELECT-Abfrage verwendet die Funktion "ST_NumPoints", um die Anzahl der Punkte in jedem Feature für jeden Feature-Typ abzurufen.

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

Hinweis:
Nur Oracle und PostgreSQL

Definition

ST_OrderingEquals vergleicht zwei ST_Geometry-Objekte und gibt 1 (Oracle) oder t (PostgreSQL) zurück, wenn die beiden Geometrien identisch sind und die Koordinaten in der gleichen Reihenfolge vorliegen. Andernfalls wird 0 (Oracle) oder f (PostgreSQL) zurückgegeben.

Syntax

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

Rückgabetyp

Boolesch

Beispiel

Oracle

Mit der folgenden `CREATE TABLE`-Anweisung wird die Tabelle "LINESTRING_TEST" erstellt, die über die beiden Linestring-Spalten "ln1" und "ln2" verfügt.

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

Mit der folgenden `INSERT`-Anweisung werden zwei ST_LineString-Werte in "ln1" und" ln2" eingefügt, die gleich sind und über die gleiche Koordinatenreihenfolge verfügen.

```
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)
);
```

Die folgende `SELECT`-Anweisung und der zugehörige Ergebnissatz zeigen, dass die Funktion ST_Equals unabhängig von der Koordinatenreihenfolge 1 (true) zurückgibt. Die Funktion ST_OrderingEquals gibt 0 (false) zurück, wenn die Geometrien nicht gleich sind und nicht über dieselbe Koordinatenreihenfolge verfügen.

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

lid Equals     OrderingEquals
```

```
1   1          0
```

PostgreSQL

Mit der folgenden `CREATE TABLE`-Anweisung wird die Tabelle "LINESTRING_TEST" erstellt, die über die beiden Linestring-Spalten "ln1" und "ln2" verfügt.

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

Mit der folgenden `INSERT`-Anweisung werden zwei ST_LineString-Werte in "ln1" und" ln2" eingefügt, die gleich sind und über die gleiche Koordinatenreihenfolge verfügen.

```
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)
);
```

Die folgende `SELECT`-Anweisung und der zugehörige Ergebnissatz zeigen, dass die Funktion "ST_Equals" unabhängig von der Koordinatenreihenfolge "t" (true) zurückgibt. Die Funktion "ST_OrderingEquals" gibt "f" (false) zurück, wenn die Geometrien nicht gleich sind und nicht über dieselbe Koordinatenreihenfolge verfügen.

```
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

Definition

ST_Overlaps wählt zwei Geometrieobjekte aus und gibt den Wert 1 (Oracle und SQLite) oder t (PostgreSQL) zurück, wenn die Schnittmenge dieser Objekte ein Geometrieobjekt ergibt, das sich von den beiden Quellobjekten unterscheidet, jedoch dieselbe Dimension hat. Andernfalls wird 0 (Oracle und SQLite) bzw. f (PostgreSQL) zurückgegeben.

Syntax

Oracle und PostgreSQL

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

SQLite

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

Rückgabetyp

Boolesch

Beispiel

Der Feuerwehrchef eines Landkreises benötigt eine Liste aller empfindlichen Bereiche, die den Pufferradius von Sondermülldeponien überlappen. Die Tabelle "sensitive_areas" enthält einige Spalten, die die bedrohten Einrichtungen beschreiben, sowie die Spalte "shape" mit den ST_Polygon-Geometrien der Einrichtungen.

In der Tabelle "hazardous_sites" werden die Kennungen der Deponien in der Spalte "id" und die geographische Lage der einzelnen Deponien in der Spalte "site point" gespeichert.

Die Tabellen "sensitive_areas" und "hazardous_sites" werden durch die Funktion ST_Overlaps miteinander verknüpft. Diese Funktion gibt die ID aller Datensätze der Tabelle "sensitive_areas" zurück, die Polygone enthalten, die den Pufferradius der Sondermülldeponien überlappen.

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

Definition

"ST_Perimeter" gibt die Länge der kontinuierlichen Linie zurück, die die Grenze eines geschlossenen Polygon- oder Multipolygon-Features bildet.

Diese Funktion ist erstmals in Version 10.8.1 verfügbar.

Syntax

Mit den ersten beiden Optionen in jedem Abschnitt wird den Umfang in den Einheiten des für das Feature definierten Koordinatensystems zurückgegeben. Mit den beiden zweiten Optionen können Sie die lineare Messeinheit angeben. Eine Liste der unterstützten Werte für `linear_unit_name` finden Sie unter ST_Distance.

Oracle und 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)
```

Rückgabetyp

Doppelte Genauigkeit

Beispiele

Oracle

Im folgenden Beispiel muss ein Ökologe im Rahmen einer Untersuchung zu Küstenvögeln die Uferlänge für die Seen in einem bestimmten Gebiet ermitteln. Die Seen sind als Polygone in der Tabelle waterbodies gespeichert. Eine `SELECT`-Anweisung unter Verwendung der Funktion "ST_Perimeter" wird verwendet, um den Umfang der einzelnen Seen (Features) in der Tabelle waterbodies zurückzugeben.

```
--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;
```

Die SELECT-Anweisung gibt Folgendes zurück:

```
ID PERIMETER
1 +1.8000000
```

Im nächsten Beispiel erstellen Sie eine Tabelle namens "bfp", fügen drei Features ein und berechnen den Umfang jedes Features in linearen Messeinheiten:

```
--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;
```

Die SELECT-Anweisung gibt den Umfang jedes Features in drei Einheiten zurück:

```
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

Im folgenden Beispiel muss ein Ökologe im Rahmen einer Untersuchung zu Küstenvögeln die Uferlänge für die Seen in einem bestimmten Gebiet ermitteln. Die Seen sind als Polygone in der Tabelle waterbodies gespeichert. Eine SELECT-Anweisung unter Verwendung der Funktion "ST_Perimeter" wird verwendet, um den Umfang der einzelnen Seen (Features) in der Tabelle waterbodies zurückzugeben.

```
--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;
```

Die SELECT-Anweisung gibt Folgendes zurück:

```
ID PERIMETER
1 +1.8000000
```

Im nächsten Beispiel erstellen Sie eine Tabelle namens "bfp", fügen drei Features ein und berechnen den Umfang jedes Features in linearen Messeinheiten:

```
--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;
```

Die SELECT-Anweisung gibt den Umfang jedes Features in drei Einheiten zurück:

```
 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

Im folgenden Beispiel muss ein Ökologe im Rahmen einer Untersuchung zu Küstenvögeln die Uferlänge für die Seen in einem bestimmten Gebiet ermitteln. Die Seen sind als Polygone in der Tabelle waterbodies gespeichert. Eine SELECT-Anweisung unter Verwendung der Funktion "ST_Perimeter" wird verwendet, um den Umfang der einzelnen Seen (Features) in der Tabelle waterbodies zurückzugeben.

```
--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;
```

Die SELECT-Anweisung gibt Folgendes zurück:

```
ID PERIMETER
1 +1.8000000
```

Im nächsten Beispiel erstellen Sie eine Tabelle namens "bfp", fügen drei Features ein und berechnen den Umfang jedes Features in linearen Messeinheiten:

```
--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;
```

Die SELECT-Anweisung gibt den Umfang jedes Features in drei Einheiten zurück:

```
 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

Definition

Mit ST_Point wird anhand eines Well-known-Text-Objekts oder anhand von Koordinaten und einer Raumbezugs-ID ein ST_Point-Objekt zurückgegeben.

Hinweis:
Beim Erstellen von räumlichen Tabellen, die mit ArcGIS verwendet werden sollen, ist es am besten, die Spalte als übergeordneten Geometrietyp (z. B. ST_Geometry) zu erstellen, statt einen ST_Geometry-Subtype anzugeben.

Syntax

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)
```

Rückgabetyp

ST_Point

Beispiel

Mit der folgenden `CREATE TABLE`-Anweisung wird die Tabelle "point_test ", die nur über die Spalte PT1 verfügt.

Die Funktion "ST_Point" konvertiert die Punktkoordinaten in eine ST_Point-Geometrie, bevor sie in die Spalte "pt1" eingefügt wird.

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

Hinweis:
Wird nur in Oracle und SQLite verwendet; verwenden Sie für PostgreSQL ST_Point.

Definition

Mit ST_PointFromText wird anhand einer Repräsentation im WKT-Format (Well-Known Text) vom Typ ST_Point und einer Raumbezugs-ID ein Punkt zurückgegeben.

Syntax

Oracle

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

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

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

SQLite

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

```
st_pointfromtext (wkt text)
```

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

Rückgabetyp

ST_Point

Beispiel

Die Tabelle "point_test" wird mit der Spalte "pt1" vom Typ ST_Point erstellt.

Die Funktion ST_PointFromText konvertiert die Punktkoordinaten in das Punktformat, bevor diese mit der INSERT-Anweisung in die Spalte pt1 eingefügt wird.

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

Definition

Mit ST_PointFromWKB wird anhand einer Repräsentation im WKB-Format (Well-Known Binary) und einer Raumbezugs-ID ein ST_Point-Objekt zurückgegeben.

Syntax

Oracle

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

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

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

PostgreSQL

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

SQLite

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

```
st_pointfromwkb (wkb blob)
```

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

Rückgabetyp

ST_Point

Beispiel

In diesem Beispiel wird veranschaulicht, wie ST_PointFromWKB zum Erstellen eines Punktes aus seiner WKB-Repräsentation (Well-Known Binary) verwendet werden kann. Bei den Geometrien handelt es sich um Punkte im Raumbezugssystem 4326. In diesem Beispiel werden die Punkte in der Spalte "geometry " der Tabelle "sample_points" gespeichert. Anschließend wird die Spalte "wkb" mit den Repräsentationen im WKB-Format (Well-Known Binary) (mithilfe der Funktion "ST_AsBinary") aktualisiert. Zum Schluss werden die Punkte mithilfe der ST_PointFromWKB-Funktion aus der Spalte "wbk" zurückgegeben. Die Tabelle "sample-points" verfügt über die Spalte "geometry", in der die Punkte gespeichert werden, sowie über die Spalte "wbk", in der die Repräsentation im WKB-Format (Well-Known Binary) der Punkte gespeichert wird.

In der SELECT-Anweisung werden die Punkte mit der ST_PointFromWKB-Funktion aus der Spalte "WBK" abgerufen.

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

Definition

ST_PointN akzeptiert ein ST_LineString-Objekt und einen ganzzahligen Index und gibt den Punkt zurück, der den n-ten Stützpunkt im Pfad des ST_LineString-Objekts darstellt.

Syntax

Oracle und PostgreSQL

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

SQLite

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

Rückgabetyp

ST_Point

Beispiel

Die Tabelle "pointn_test" wird mit der Spalte "gid", durch die die einzelnen Zeilen eindeutig gekennzeichnet werden, und der Spalte "pt1" vom Typ ST_LineString erstellt. Mit der INSERT-Anweisung werden zwei Linestring-Werte in die Tabelle eingefügt. Der erste Linestring enthält weder Z-Koordinaten noch Messwerte, der zweite Linestring dagegen beides.

Die SELECT-Abfrage verwendet die Funktionen "ST_PointN" und "ST_AsText", um den Well-Known Text für den zweiten Stützpunkt jedes Linestrings zurückzugeben.

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

Definition

ST_PointOnSurface akzeptiert ein ST_Polygon- oder ST_MultiPolygon-Objekt und gibt ein ST_Point-Objekt zurück, das sicher auf der Oberfläche der angegebenen Geometrie liegt.

Syntax

Oracle und PostgreSQL

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

SQLite

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

Rückgabetyp

ST_Point

Beispiel

Ein Ingenieur des Bauamts möchte einen Label-Punkt für den Grundriss jedes historischen Gebäudes erstellen. Die Grundrisse historischer Gebäude werden in der Tabelle "hbuildings" gespeichert, die mit der folgenden CREATE TABLE-Anweisung erstellt wurde:

Die Funktion ST_PointOnSurface generiert einen Punkt, der sicher auf der Fläche des angegebenen Gebäudegrundrisses liegt. Die Funktion ST_PointOnSurface gibt einen Punkt zurück, der von der Funktion ST_AsText in Text konvertiert wird, der von der Anwendung verwendet wird.

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

Hinweis:
Nur Oracle und SQLite

Definition

Mit ST_PolyFromText wird anhand einer Repräsentation im WKT-Format (Well-Known Text) und einer Raumbezugs-ID ein ST_Polygon-Objekt zurückgegeben.

Syntax

Oracle

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

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

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

SQLite

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

```
st_polyfromtext (wkt text)
```

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

Rückgabetyp

ST_Polygon

Beispiel

Die Tabelle "polygon_test" wird mit der Spalte "polygon" erstellt.

Mit der INSERT-Anweisung wird mithilfe der Funktion ST_PolyFromText ein Polygon in die Spalte "polygon" eingefügt.

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

Definition

Mit ST_PolyFromWKB wird anhand einer Repräsentation im WKB-Format (Well-Known Binary) und einer Raumbezugs-ID ein ST_Polygon-Objekt zurückgegeben.

Syntax

Oracle

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

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

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

PostgreSQL

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

SQLite

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

```
st_polyfromwkb (wkb blob)
```

Wenn Sie keine SRID angeben, ist der standardmäßige Raumbezug 4326.

Rückgabetyp

ST_Polygon

Beispiel

In diesem Beispiel wird veranschaulicht, wie ST_PolyFromWKB zum Erstellen eines Polygons aus seiner WKB-Repräsentation (Well-Known Binary) verwendet werden kann. Bei der Geometrie handelt es sich um ein Polygon im Raumbezugssystem 4326. In diesem Beispiel wird das Polygon mit der ID = 1115 in der Spalte "geometry" der Tabelle "sample_polys" gespeichert. Anschließend wird die Spalte "wkb" mit dem WKB-Format (mithilfe der Funktion "ST_AsBinary") aktualisiert. Zum Schluss wird mit der ST_PolyFromWKB-Funktion das Polygon aus der Spalte "WBK" zurückgegeben. Die Tabelle "sample_polys" verfügt über die Spalte "geometry", in der das Polygon gespeichert wird, sowie über die Spalte "wbk", in der die WBK-Repräsentation des Polygons gespeichert wird.

In der SELECT-Anweisung werden die Punkte mit der ST_PointFromWKB-Funktion aus der Spalte "WBK" abgerufen.

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

Definition

Mit der Accessor-Funktion ST_Polygon wird anhand einer WKT-Repräsentation (Well-known Text) und anhand einer Raumbezugs-ID (SRID) ein ST_Polygon-Objekt erstellt.

Hinweis:
Beim Erstellen von räumlichen Tabellen, die mit ArcGIS verwendet werden sollen, ist es am besten, die Spalte als übergeordneten Geometrietyp (z. B. ST_Geometry) zu erstellen, statt einen ST_Geometry-Subtype anzugeben.

Syntax

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)
```

Rückgabetyp

ST_Polygon

Beispiel

Mit der folgenden `CREATE TABLE`-Anweisung werden die "polygon_test"-Tabellen erstellt, die nur die Spalte "p1" enthalten. Die folgende `INSERT`-Anweisung konvertiert einen Ring (ein geschlossenes und einfaches Polygon) in ein ST_Polygon-Objekt und fügt dieses mit der ST_Polygon-Funktion in die Spalte "p1" ein.

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

Definition

"ST_Relate" vergleicht zwei Geometrien und gibt 1 (Oracle und SQLite) oder t (PostgreSQL) zurück, wenn die beiden Geometrien die Bedingungen erfüllen, die in der Zeichenfolge DE-9IM-Mustermatrix festgelegt worden sind. Andernfalls wird 0 (Oracle und SQLite) oder f (PostgreSQL) zurückgegeben.

Bei Verwendung von "ST_Relate" in SQLite und Oracle gibt es eine zweite Option: Sie können zwei Geometrien vergleichen, damit sie eine Zeichenfolge zurückgeben, die die DE-9IM-Mustermatrix darstellen, mit der die Beziehung der Geometrien zueinander definiert wird.

Syntax

Oracle

Option 1

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

Option 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

Option 1

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

Option 2

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

Rückgabetyp

Bei PostgreSQL wird ein boolescher Wert zurückgegeben.

Mit Option 1 für SQLite und Oracle wird eine ganze Zahl zurückgegeben.

Mit Option 2 für SQLite und Oracle wird eine Zeichenfolge zurückgegeben.

Beispiele

Eine DE-9IM-Mustermatrix ist ein Hilfsmittel zum Vergleichen von Geometrien. Es gibt verschiedene Typen solcher Matrizen. Beispielsweise können Sie die ST_Relate-Funktion und die Mustermatrix für Gleichheit (T*F**FFF*) verwenden, um herauszufinden, ob zwei Geometrien gleich sind. Sie können jedoch auch das DE-9IM-Muster verwenden (1*F**FFF*). Beim letztgenannten Muster gibt ST_Relate an, ob zwei Geometrien an der ersten Position gleich sind, wodurch angegeben wird, ob die Innenbereiche der Schnittmenge der beiden Geometrien eine Linie ist (Dimension 1).

In den folgenden Beispielen wird die Tabelle "relate_test" mit drei räumlichen Spalten erstellt, in die jeweils Punkt-Features eingefügt werden. Die ST_Relate-Funktion wird in der `SELECT`-Anweisung verwendet, um zu testen, ob die Punkte gleich sind.

Wenn Sie feststellen möchten, ob Geometrien gleich sind, und dabei die Dimensionalität der Beziehung nicht ermitteln müssen, verwenden Sie stattdessen die Funktion ST_Equals.

Oracle

Im ersten Beispiel wird die erste ST_Relate-Option angezeigt, in der Geometrien basierend auf einer DE-9IM-Mustermatrix verglichen werden, damit sie 1 zurückgeben, wenn die Geometrien die in der Matrix definierten Anforderungen erfüllen, oder 0, wenn die Geometrien dies nicht tun.

```
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;
```

Damit wird Folgendes zurückgegeben:

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

```
1               0               0
```

In diesem Beispiel ist die zweite Option dargestellt. Es werden zwei Geometrien verglichen und die DE-9IM-Mustermatrix zurückgegeben.

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

Damit wird Folgendes zurückgegeben:

```
g1 rel g2
0FFFFFFF2
```

PostgreSQL

Im diesem Beispiel werden Geometrien basierend auf einer DE-9IM-Mustermatrix verglichen, damit sie t zurückgeben, wenn die Geometrien die in der Matrix definierten Anforderungen erfüllen, oder f, wenn die Geometrien dies nicht tun.

```
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;
```

Damit wird Folgendes zurückgegeben:

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

t          f         f
```

SQLite

In diesem ersten Beispiel wird die erste ST_Relate-Option angezeigt, in der zwei Geometrien basierend auf einer DE-9IM-Mustermatrix verglichen werden, damit sie 1 zurückgeben, wenn die Geometrien die in der Matrix definierten Anforderungen erfüllen, oder 0, wenn die Geometrien dies nicht tun.

```
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;
```

Damit wird Folgendes zurückgegeben:

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

1          0         0
```

In diesem Beispiel ist die zweite Option dargestellt. Es werden zwei Geometrien verglichen und die DE-9IM-Mustermatrix zurückgegeben.

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

Damit wird Folgendes zurückgegeben:

```
g1 rel g2
0FFFFFFF2
```

ST_SRID

Definition

ST_SRID wählt ein Geometrieobjekt aus und gibt dessen Raumbezugs-ID (SRID) zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_srid (geometry1 geometryblob)
```

Rückgabetyp

Ganzzahl

Beispiele

Es wird folgende Tabelle erstellt:

In der nächsten Anweisung wird eine Punktgeometrie mit den Koordinaten (10.01, 50.76) in die Geometriespalte "g1" eingefügt. Beim Erstellen der Punktgeometrie wird ihr der SRID-Wert 4326 Wert zugewiesen.

Die Funktion "ST_SRID" gibt die Raumbezugs-ID der soeben eingegebenen Geometrie zurück.

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

Definition

ST_StartPoint gibt den ersten Punkt eines Linestrings zurück.

Syntax

Oracle und PostgreSQL

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

SQLite

```
st_startpoint (ln1 geometryblob)
```

Rückgabetyp

ST_Point

Beispiele

Die Tabelle "startpoint_test" wird mit der ganzzahligen Spalte "gid", durch die die einzelnen Tabellenzeilen eindeutig gekennzeichnet werden, und der Spalte "pt1" vom Typ ST_LineString erstellt.

Mit der INSERT-Anweisung werden die ST_LineString-Werte in die Spalte "ln1" eingefügt. Der erste ST_LineString-Eintrag enthält weder Z-Koordinaten noch Messwerte, der zweite ST_LineString_Eintrag dagegen beides.

Funktion ST_StartPoint extrahiert den ersten Punkt jedes ST_LineString-Eintrags. Der Punkt in der Liste enthält weder Z-Koordinaten noch Messwerte, der zweite Punkt enthält dagegen beides, weil diese im Quell-Linestring enthalten sind.

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

Hinweis:
Nur Oracle und SQLite

Definition

Mit ST_Surface wird aus einer Well-known-Text-Repräsentation ein Oberflächen-Feature erstellt. Oberflächen sind mit Polygonen vergleichbar, weisen jedoch über die gesamte Ausdehnung hinweg Punktwerte auf.

Syntax

Oracle

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

SQLite

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

Rückgabetyp

ST_Polygon

Beispiel

Die Tabelle "surf_test" wird erstellt, und eine Oberflächengeometrie wird eingefügt.

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

Definition

ST_SymmetricDiff wählt zwei Geometrieobjekte aus und gibt ein Geometrieobjekt zurück, das aus den Teilen der Quellobjekte besteht, die nicht Teil der Schnittmenge der beiden Geometrien sind.

Syntax

Oracle und PostgreSQL

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

SQLite

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

Rückgabetyp

Oracle und PostgreSQL

ST_Geometry

SQLite

Geometryblob

Beispiel

Für einen Sonderbericht muss der Feuerwehrchef eines Landkreises die sich nicht schneidenden Flächen im Radius der Abflussgebiete und im Radius gefährlicher Abgase ermitteln.

Die Tabelle für Abflussgebiete enthält eine ID-Spalte, eine Spalte, in der der Name des Abflussgebiets (wname) gespeichert ist, und eine Shape-Spalte, in der die Flächengeometrie des Abflussgebiets gespeichert ist.

In der Tabelle für Abgase wird die Kennung des Standorts in der Spalte "id" und die geographische Lage der einzelnen Standorte in der Spalte "site point" gespeichert.

Die Funktion ST_Buffer generiert einen Puffer um die Sondermülldeponien. Die Funktion "ST_SymmetricDiff" gibt die Polygone des Pufferradius von Sondermülldeponien und die Abflussgebiete zurück, die sich nicht schneiden.

Die symmetrische Differenz von Sondermülldeponien und Abflussgebieten resultiert in der Subtraktion der sich überschneidenden Bereiche.

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

Definition

ST_Touches gibt den Wert 1 (Oracle und SQLite) oder t (PostgreSQL) zurück, wenn keiner der Punkte, die beiden Geometrien gemeinsam sind, den Innenbereich beider Geometrien schneidet. Andernfalls wird 0 (Oracle und SQLite) oder f (PostgreSQL) zurückgegeben. Bei mindestens einer Geometrie muss es sich um einen "ST_LineString", einen "ST_MultiLineString" oder ein "ST_MultiPolygon" handeln.

Syntax

Oracle und PostgreSQL

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

SQLite

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

Rückgabetyp

Boolesch

Beispiel

Ein GIS-Techniker wurde von seinem Vorgesetzten gebeten, eine Liste aller Abwasserkanäle zusammenzustellen, deren Endpunkte einen anderen Abwasserkanal schneiden.

Die Tabelle "sewerlines" wird mit drei Spalten erstellt. Durch die erste Spalte "sewer_id" werden die einzelnen Abwasserkanäle eindeutig identifiziert. Die ganzzahlige Spalte "class" gibt den Typ des Abwasserkanals an, der im Allgemeinen der Kapazität des Kanals zugeordnet ist. In der Spalte "sewer" wird die Geometrie des Abwasserkanals gespeichert.

Die SELECT-Abfrage verwendet die ST_Touches-Funktion, um eine Liste der Abwasserkanäle zurückzugeben, die einander berühren.

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

Definition

ST_Transform verwendet zweidimensionale ST_Geometry-Daten als Eingabe und gibt Werte zurück, die in den durch die Raumbezugs-ID (SRID) angegebenen Raumbezug konvertiert werden.

⚠ Vorsicht:
Wenn Sie die räumliche Spalte mit der Funktion "st_register_spatial_column" bei der PostgreSQL-Datenbank registriert haben, wird die SRID bei der Registrierung in die Tabelle "sde_geometry_columns" geschrieben. Wenn Sie einen räumlichen Index für die räumliche Tabelle in einer Oracle-Datenbank erstellt haben, wird die SRID zum Zeitpunkt der Erstellung des räumlichen Index in die Tabelle "st_geometry_columns" geschrieben. Wenn "ST_Transform" zum Ändern der SRID der ST_Geometry-Daten verwendet wird, wird die SRID in der Tabelle "sde_geometry_columns" bzw. "st_geometry_columns table" nicht aktualisiert.

Wenn die geographischen Koordinatensysteme unterschiedlich sind, führt ST_Transform eine geographische Transformation aus. Eine geographische Transformation ist eine Methode zum Konvertieren von Daten zwischen zwei geographischen Koordinatensystemen. Eine geographische Transformation ist in einer bestimmten Richtung definiert, beispielsweise von NAD 1927 nach NAD 1983. Unabhängig davon, welches Koordinatensystem für Quelle und Ziel verwendet wird, wendet die Funktion ST_Transform die Transformation jedoch richtig an.

Geographische Transformationsmethoden lassen sich in zwei Typen einteilen: gleichungsbasierte und dateibasierte Transformationen. Gleichungsbasierte Methoden sind in sich geschlossen und erfordern keine externen Informationen. Bei dateibasierten Methoden werden auf dem Datenträger vorhandene Dateien zum Berechnen von Versatzwerten verwendet. Diese Methoden sind in der Regel genauer als gleichungsbasierte Methoden. Dateibasierte Methoden werden häufig in Australien, Kanada, Deutschland, Neuseeland, Spanien und den USA verwendet. Die Dateien (mit Ausnahme der kanadischen Dateien) können aus einer ArcGIS Pro-Installation oder direkt von den verschiedenen nationalen Kartografiebehörden bezogen werden.

Um dateibasierte Transformationen zu unterstützen, müssen die Dateien auf dem Datenbankserver in der gleichen relativen Ordnerstruktur wie im pedata-Ordner im ArcGIS Pro-Installationsverzeichnis abgelegt werden.

Beispielsweise gibt es im Ordner `Resources` des Installationsverzeichnisses von ArcGIS Pro den Ordner `pedata`. Der Ordner `pedata` enthält mehrere Unterordner, aber die drei Ordner, die unterstützte dateibasierte Methoden enthalten, sind `harn`, `nadcon` und `ntv2`. Kopieren Sie den Ordner `pedata` und dessen Inhalt aus dem ArcGIS-Installationsverzeichnis auf den Datenbankserver, oder erstellen Sie ein Verzeichnis auf dem Datenbankserver, das die Unterverzeichnisse und Dateien der unterstützten dateibasierten Transformationsmethode enthält. Sobald sich die Dateien auf dem Datenbankserver befinden, legen Sie die Betriebssystem-Umgebungsvariable "PEDATAHOME" auf dem gleichen Server fest. Legen Sie die Variable "PEDATAHOME" auf den Speicherort des Verzeichnisses fest, das die Unterverzeichnisse und Dateien enthält. Wenn der Ordner "pedata" beispielsweise auf einem Microsoft Windows-Server nach `C:\pedata` kopiert wird, legen Sie die Umgebungsvariable "PEDATAHOME" auf `C:\pedata` fest.

Weitere Informationen zum Festlegen von Umgebungsvariablen finden Sie in der Dokumentation zu Ihrem Betriebssystem.

Nachdem Sie "PEDATAHOME" festgelegt haben, müssen Sie eine neue SQL-Sitzung starten, bevor Sie die Funktion "ST_Transform" verwenden können. Der Server muss jedoch nicht neu gestartet werden.

Verwenden von "ST_Transform" mit PostgreSQL

In PostgreSQL können auch Raumbezüge mit verschiedenen geographischen Koordinatensystemen konvertiert werden.

Wenn die geographischen Koordinatensysteme gleich sind und die Daten in einer Datenbank (statt in einer Geodatabase) gespeichert sind, führen Sie die folgenden Schritte aus, um den Raumbezug der ST_Geometry-Daten zu ändern:

1. Erstellen Sie eine Sicherungskopie der Tabelle.
2. Erstellen Sie eine zweite (Ziel-) ST_Geometry-Spalte in der Tabelle.
3. Registrieren Sie die Ziel-ST_Geometry-Spalte, indem Sie die neue SRID angeben.
   Hiermit wird der Raumbezug der Spalte angegeben, indem ein Datensatz in die Systemtabelle "sde_geometry_columns" eingefügt wird.
4. Führen Sie die ST_Transform-Funktion aus, und legen Sie fest, dass die transformierten Daten in der Ziel-ST_Geometry-Spalte ausgegeben werden.
5. Heben Sie die Registrierung der ersten (Quell-) ST_Geometry-Spalte auf.

Wenn die Daten in einer Geodatabase gespeichert sind, sollten Sie ArcGIS-Werkzeuge zum Neuprojizieren der Daten in eine neue Feature-Class verwenden. Durch die Ausführung von "ST_Transform" für eine Geodatabase-Feature-Class werden die Funktionen zum Aktualisieren von Geodatabase-Systemtabellen mit der neuen SRID umgangen.

Verwenden von "ST_Transform" mit Oracle

In Oracle können auch Raumbezüge mit verschiedenen geographischen Koordinatensystemen konvertiert werden.

Wenn die Daten in einer Datenbank (statt in einer Geodatabase) gespeichert sind und kein räumlicher Index für die räumliche Spalte definiert wurde, können Sie eine zweite ST_Geometry-Spalte hinzufügen und die transformierten Daten in dieser Spalte ausgeben. Sie können die ursprüngliche Spalte, die Quell-ST_Geometry-Spalte, und die Ziel-ST_Geometry-Spalte in der Tabelle beibehalten, obgleich Sie in ArcGIS jeweils nur eine Spalte anzeigen können. Verwenden Sie dazu eine Ansicht, oder ändern Sie die Definition des Abfrage-Layers für die Tabelle.

Wenn die Daten in einer Datenbank (statt in einer Geodatabase) gespeichert sind und ein räumlicher Index für die räumliche Spalte definiert wurde, können Sie die ursprüngliche ST_Geometry-Spalte nicht beibehalten. Nachdem ein räumlicher Index für eine ST_Geometry-Spalte definiert wurde, wird die SRID in die Metadatentabelle "st_geometry_columns" geschrieben. Diese Tabelle wird nicht mit ST_Transform aktualisiert.

1. Erstellen Sie eine Sicherungskopie der Tabelle.
2. Erstellen Sie eine zweite (Ziel-) ST_Geometry-Spalte in der Tabelle.
3. Führen Sie die ST_Transform-Funktion aus, und legen Sie fest, dass die transformierten Daten in der Ziel-ST_Geometry-Spalte ausgegeben werden.
4. Löschen Sie den räumlichen Index der Quell-ST_Geometry-Spalte.
5. Löschen Sie die Quell-ST_Geometry-Spalte.
6. Erstellen Sie einen räumlichen Index für die Ziel-ST_Geometry-Spalte.

Wenn die Daten in einer Geodatabase gespeichert sind, sollten Sie ArcGIS-Werkzeuge zum Neuprojizieren der

Daten in eine neue Feature-Class verwenden. Durch die Ausführung von "ST_Transform" für eine Geodatabase-Feature-Class werden die Funktionen zum Aktualisieren von Geodatabase-Systemtabellen mit der neuen SRID umgangen.

Verwenden von "ST_Transform" mit SQLite

In SQLite können auch Raumbezüge mit verschiedenen geographischen Koordinatensystemen konvertiert werden.

Syntax

Quell- und Zielraumbezüge weisen dasselbe geographische Koordinatensystem auf

Oracle und PostgreSQL

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

SQLite

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

Quell- und Zielraumbezüge weisen nicht dasselbe geographische Koordinatensystem auf

Oracle

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

PostgreSQL

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

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

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

In Option 3 haben Sie die Möglichkeit, die Ausdehnung als kommagetrennte Liste mit Koordinaten in der folgenden Reihenfolge anzugeben: X-Koordinate unten links, Y-Koordinate unten links, X-Koordinate oben rechts, Y-Koordinate oben rechts. Wird keine Ausdehnung angegeben, verwendet ST_Transform eine größere, allgemeinere Ausdehnung.

Bei der Angabe einer Ausdehnung können Informationen zum Nullmeridian sowie zu Einheitenumrechnungsfaktoren optional weggelassen werden. Diese Informationen werden nur dann benötigt, wenn die angegebenen Werte für die Ausdehnung weder den Nullmeridian von Greenwich noch Dezimalgradangaben verwenden.

SQLite

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

Rückgabetyp

Oracle und PostgreSQL

ST_Geometry

SQLite

Geometryblob

Beispiele

Datentransformation, wenn für Quell- und Zielraumbezüge dasselbe geographische Koordinatensystem verwendet wird.

Im folgenden Beispiel wird die Tabelle "transform_test" erstellt, die über die beiden Linestring-Spalten "ln1" und "ln2" verfügt. Eine Linie wird in die Spalte "ln1" mit der SRID 4326 eingefügt. Anschließend wird die Funktion "ST_Transform" in einer `UPDATE`-Anweisung verwendet, um den Linestring in Spalte "ln1" von dem der SRID 4326 zugeordneten Koordinatenbezug in den der SRID 3857 zugeordneten Koordinatenbezug zu konvertieren, und ihn in die Spalte "ln2" einzufügen.

Hinweis:
Die SRIDs 4326 und 3857 haben das gleiche geographische Datum.

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);
```

Datentransformation, wenn für Quell- und Zielraumbezüge nicht das gleiche geographische Koordinatensystem verwendet wird.

Im folgenden Beispiel wird die Tabelle "n27" mit einer ID- und einer Geometriespalte erstellt. Ein Punkt-Wert wird in die Tabelle "n27" mit der SRID 4267 eingefügt. Für SRID 4267 wird das geographische Koordinatensystems NAD 1927 verwendet.

Als Nächstes wird die Tabelle "n83" erstellt und mit der Funktion "ST_Transform" die Geometrie aus der Tabelle "n27" in die Tabelle "n83" eingefügt, allerdings mit der SRID 4269 und der geographischen Transformations-ID 1241. Für SRID 4269 wird das geographische Koordinatensystems NAD 1983 verwendet, und 1241 ist eine bekannte ID für die Transformation NAD_1927_To_NAD_1983_NADCON. Diese Transformation ist dateibasiert und kann für die 48 kontinentalen US-Bundesstaaten entwickelt verwendet werden.

Tipp:

Eine Liste der unterstützten geographischen Transformationen finden Sie im technischen Artikel 000012357 von Esri und unter den im Artikelabschnitt **Zugehörige Informationen** angegebenen Links.

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
);
```

Wenn PEDATAHOME richtig definiert worden ist, hat die Ausführung einer SELECT-Anweisung mit der Tabelle "n83" folgendes Ergebnis:

```
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

Definition

ST_Union gibt ein Geometrieobjekt zurück, das die Kombination der beiden Quellobjekte darstellt.

Syntax

Oracle und PostgreSQL

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

SQLite

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

Rückgabetyp

Oracle und PostgreSQL

ST_Geometry

SQLite

Geometryblob

Beispiel

Die Tabelle "sensitive_areas" enthält die IDs der bedrohten Einrichtungen sowie die Spalte "shape" mit den Polygongeometrien der Einrichtungen.

In der Tabelle "hazardous_sites" werden die Kennungen der Deponien in der Spalte "id" und die geographische Lage der einzelnen Deponien in der Spalte "site point" gespeichert.

Die Funktion ST_Buffer generiert einen Puffer um die Sondermülldeponien. Die Funktion ST_Union erzeugt Polygone aus der Vereinigung der gepufferten Sondermülldeponien mit den Polygonen der empfindlichen Bereiche. Die Elementfunktion "ST_Area" gibt die Fläche dieser Polygone zurück.

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

Definition

ST_Within gibt 1 (Oracle und SQLite) oder t (PostgreSQL) zurück, wenn das erste ST_Geometry-Objekt vollkommen innerhalb des zweiten ST_Geometry-Objekts liegt. Andernfalls wird 0 (Oracle und SQLite) oder f (PostgreSQL) zurückgegeben.

Syntax

Oracle und PostgreSQL

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

SQLite

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

Rückgabetyp

Boolesch

Beispiel

Im folgenden Beispiel werden die beiden Tabellen "zones" und "squares" erstellt. Die SELECT-Anweisung findet alle Quadrate, die sich überschneiden, jedoch nicht innerhalb einer Parzelle befinden.

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

Definition

ST_X wählt einen Eingabeparameter vom Typ "ST_Point" aus und gibt dessen X-Koordinate zurück. In SQLite kann "ST_X" auch die X-Koordinate eines ST_Point aktualisieren.

Syntax

Oracle und PostgreSQL

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

SQLite

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

Rückgabetyp

Doppelte Genauigkeit

Die Funktion "ST_X" kann mit SQLite verwendet werden, um die X-Koordinate eines Punktes zu aktualisieren. In diesem Fall wird ein geometryblob zurückgegeben.

Beispiele

Die Tabelle "x_test" wird mit zwei Spalten erstellt: der Spalte "gid", durch die die einzelnen Zeilen eindeutig gekennzeichnet werden, und der Spalte "pt1" vom Typ ST_Point.

Mit den INSERT-Anweisungen werden zwei Zeilen eingefügt. Eine Zeile enthält einen Punkt ohne Z-Koordinate oder Messwert. Die andere Spalte enthält sowohl eine Z-Koordinate als auch einen Messwert.

Die SELECT-Abfrage verwendet die Funktion "ST_X", um die X-Koordinate jedes Punkt-Features abzurufen.

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
```

Die Funktion "ST_X" kann auch verwendet werden, um den Koordinatenwert eines vorhandenen Punktes zu aktualisieren. In diesem Beispiel wird der X-Koordinatenwert des ersten Punktes in "x_Test" mit "ST_X" aktualisiert.

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

ST_Y

Definition

ST_Y wählt einen Eingabeparameter vom Typ "ST_Point" aus und gibt dessen Y-Koordinate zurück. In SQLite kann "ST_Y" auch die Y-Koordinate eines ST_Point aktualisieren.

Syntax

Oracle und PostgreSQL

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

SQLite

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

Rückgabetyp

Doppelte Genauigkeit

Die Funktion "ST_Y" kann mit SQLite verwendet werden, um die Y-Koordinate eines Punktes zu aktualisieren. In diesem Fall wird ein geometryblob zurückgegeben.

Beispiel

Die Tabelle "y_test" wird mit zwei Spalten erstellt: der Spalte "gid", durch die die einzelnen Zeilen eindeutig gekennzeichnet werden, und der Spalte "pt1" vom Typ ST_Point.

Mit den INSERT-Anweisungen werden zwei Zeilen eingefügt. Eine Zeile enthält einen Punkt ohne Z-Koordinate oder Messwert. Die andere Zeile enthält sowohl eine Z-Koordinate als auch einen Messwert.

Die SELECT-Abfrage verwendet die Funktion "ST_Y", um die Y-Koordinate jedes Punkt-Features abzurufen.

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
```

Die Funktion "ST_Y" kann auch verwendet werden, um den Koordinatenwert eines vorhandenen Punktes zu aktualisieren. In diesem Beispiel wird der Y-Koordinatenwert des zweiten Punktes in "y_Test" mit "ST_Y" aktualisiert.

```
UPDATE y_test
 SET pt1=st_y(
  (SELECT pt1 FROM y_test WHERE gid=2),
  20.1
  )
 WHERE gid=2;
```

ST_Z

Definition

ST_Z wählt einen Eingabeparameter vom Typ "ST_Point" aus und gibt dessen Z-Koordinate (Höhe) zurück. In SQLite kann "ST_Z" auch die Z-Koordinate eines ST_Point aktualisieren.

Syntax

Oracle und PostgreSQL

```
sde.st_z (geometry1 sde.st_point)
```

SQLite

```
st_z (geometry  geometryblob)
st_z (input_shape geometryblob, new_Zvalue double)
```

Rückgabetyp

Oracle

Zahl

PostgreSQL

Ganzzahl

SQLite

Doppelte Genauigkeit wird zurückgegeben, wenn "ST_Z" verwendet wird, um die Z-Koordinate eines Punktes zurückzugeben. Ein geometryblob wird zurückgegeben, wenn "ST_Z" verwendet wird, um die Z-Koordinate eines Punktes zu aktualisieren.

Beispiel

Die Tabelle "z_test" wird mit zwei Spalten erstellt: die Spalte "id", durch die die einzelnen Zeilen eindeutig gekennzeichnet werden, und die Spalte "geometry" vom Typ ST_Point. Mit der INSERT-Anweisung wird eine Zeile in die Tabelle "z_test" eingefügt.

Mit der SELECT-Anweisung werden die Spalte "id" und die Z-Koordinate des Punktes, der mit der vorigen Anweisung eingefügt wurde, aufgelistet.

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
```

Die Funktion "ST_Z" kann auch verwendet werden, um den Koordinatenwert eines vorhandenen Punktes zu aktualisieren. In diesem Beispiel wird der Z-Koordinatenwert des ersten Punktes in "z_Test" mit "ST_Z" aktualisiert.

```
UPDATE z_test
 SET pt1=st_z(
  (SELECT pt1 FROM z_test where id=1), 32.04)
 WHERE id=1;
```