Référence d'une fonction SQL ST_Geometry

Table des matières

Fonctions SQL utilisées avec ST_Geometry

Ce document de référence présente une liste et une description des fonctions pouvant être utilisées avec le type de données spatiales Esri ST_Geometry dans Oracle, PostgreSQL et SQLite.

Des fonctions et des types SQL Esri ST_Geometry sont créés lorsque vous effectuez l'une des opérations suivantes :

- Créer une géodatabase dans une base de données Oracle.
- Utiliser ST_Geometry lors de la création d'une géodatabase dans une base de données PostgreSQL.
- Installer le type de données spatiales ST_Geometry dans une base de données Oracle ou PostgreSQL.
- Créer une base de données SQLite qui inclut le type de données spatiales ST_Geometry à l'aide de l'outil de géotraitement Créer une base de données SQLite ou de la fonction ArcPy et charger les fonctions ST_Geometry à utiliser avec la base de données.
- Charger les fonctions ST_Geometry à utiliser avec une géodatabase mobile.

Dans les bases de données Oracle et PostgreSQL, le type ST_Geometry et ses fonctions sont créés dans une structure nommée sde. Dans SQLite, le type et les fonctions sont stockés dans une bibliothèque que vous devez charger avant d'exécuter SQL dans la base de données SQLite ou la géodatabase mobile.

Conseil :

Pour obtenir plus d'informations sur le type Esri ST_Geometry, consultez les pages d'aide ArcGIS Pro suivantes :

- ST_Geometry dans PostgreSQL
- ST_Geometry dans Oracle
- Bases de données et ST_Geometry
- Charger la bibliothèque ST_Geometry SQLite
- Charger ST_Geometry dans une géodatabase mobile pour l'accès SQL

Format des pages de fonction SQL

Les pages de fonction dans ce document sont structurées comme suit :

- Définition - Brève explication de ce que fait la fonction.
- Syntaxe - Syntaxe SQL pour utiliser la fonction.

Remarque :

Avec les opérateurs relationnels, l'ordre dans lequel les paramètres sont spécifiés est important : le premier paramètre correspond à la table à partir de laquelle la sélection est effectuée et le deuxième correspond à la table utilisée comme filtre.

- Type de retour - Type de données qui est renvoyé lorsque la fonction est exécutée.
- Exemple - Exemples utilisant la fonction spécifique.

Liste des fonctions SQL

Cliquez sur les liens ci-dessous pour accéder aux fonctions que vous pouvez utiliser avec le type ST_Geometry dans

Oracle, PostgreSQL et SQLite.

L'utilisation de fonctions ST_Geometry dans Oracle nécessite de qualifier les fonctions et les opérateurs avec `sde`. Par exemple, ST_Buffer devient sde.ST_Buffer. L'ajout de `sde.` indique au logiciel que la fonction est stockée dans la structure de l'utilisateur sde. Pour PostgreSQL, la qualification est facultative, mais il est recommandé d'inclure le qualificateur. N'incluez pas la qualification lorsque vous utilisez les fonctions avec SQLite, car les bases de données SQLite ne comportent pas de structure sde.

Lorsque vous indiquez des chaînes de texte connu en entrée pour une fonction SQL ST_Geometry, vous pouvez utiliser une notation scientifique pour spécifier des valeurs très grandes ou très petites. Par exemple, si vous spécifiez des coordonnées sous forme de texte connu lors de la création d'une entité, et que l'une des coordonnées est 0.000023500001816501026, vous pouvez saisir `2.3500001816501026e-005` à la place.

Conseil :

Pour d'autres types spatiaux, tels que les types PostGIS, Oracle SDO_Geometry, les types spatiaux Microsoft SQL Server, IBM Db2 ST_Geometry ou SAP HANA ST_Geometry, consultez la documentation mise à disposition par le fournisseur du système de gestion de bases de données pour plus d'informations sur les fonctions utilisées.

Les fonctions SQL Esri ST_Geometry ci-dessous peuvent être groupées selon leur utilisation.

Fonctions constructeur

Les fonctions constructeur partent d'un type de géométrie ou d'un texte de description de géométrie et créent une géométrie. Le tableau suivant répertorie les fonctions constructeur et indique les implémentations ST_Geometry qui les prennent en charge.

Fonctions constructeur

Fonction	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

Fonction	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

Fonctions accesseur

Il existe plusieurs fonctions qui prennent une ou plusieurs géométries en entrée et renvoient des informations spécifiques à leur sujet.

Certaines fonctions accesseur s'attachent à déterminer si une ou plusieurs fonctions répondent à certains critères. Si la géométrie répond aux critères, la fonction renvoie 1 (Oracle et SQLite) ou t (PostgreSQL) pour true. Dans le cas contraire, elle renvoie 0 (Oracle et SQLite) ou f (PostgreSQL) pour false.

Ces fonctions s'appliquent à toutes les implémentations, sauf mention contraire.

Fonctions accesseur

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

ST_GeoSize (PostgreSQL uniquement)
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

Fonctions relationnelles

Les fonctions relationnelles acceptent des géométries en entrée et déterminent s'il existe une relation spatiale entre elles. Si les conditions de la relation spatiale sont remplies, ces fonctions renvoient 1 (Oracle et SQLite) ou t (PostgreSQL) pour true. Dans le cas contraire (il n'existe aucune relation), ces fonctions renvoient 0 (Oracle et SQLite) ou f (PostgreSQL) pour false.

Ces fonctions s'appliquent à toutes les implémentations, sauf mention contraire.

Fonctions relationnelles

ST_Contains

ST_Crosses
ST_Disjoint
ST_Distance
ST_DWithin
ST_EnvIntersects (Oracle et SQLite uniquement)
ST_Equals
ST_Intersects
ST_OrderingEquals (Oracle et PostgreSQL uniquement)
ST_Overlaps
ST_Relate
ST_Touches
ST_Within

Fonctions d'opération de géométrie

Ces fonctions partent de données spatiales, effectuent des opérations spatiales dessus et renvoient une géométrie.

Ces fonctions s'appliquent à toutes les implémentations, sauf mention contraire.

Fonctions d'opération de géométrie

ST_Aggr_ConvexHull (Oracle et SQLite uniquement)
ST_Aggr_Intersection (Oracle et SQLite uniquement)
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 et Esri ST_Geometry

Vous pouvez utiliser le langage SQL (Structured Query Language), les types de données et les formats de table du système de gestion de base de données pour travailler avec les informations stockées dans une géodatabase ou une base de données dans laquelle le type ST_Geometry est installé. SQL est un langage de base de données prenant en charge les commandes de définition et de manipulation de données.

L'accès aux données via SQL permet aux applications externes d'utiliser les données tabulaires gérées par la géodatabase ou la base de données. Ces applications externes peuvent être des applications de base de données non spatiales ou des applications spatiales personnalisées.

Lorsque vous insérez ou modifiez des données dans une géodatabase ou une base de données à l'aide de SQL, émettez une instruction `COMMIT` ou `ROLLBACK` après l'exécution de l'instruction SQL pour vous assurer que les modifications sont validées dans la base de données ou annulées. Cela permet de prévenir les verrous sur les lignes, les pages ou les tables que vous modifiez.

Insérer des données ST_Geometry avec SQL

Vous pouvez utiliser SQL pour insérer des données spatiales dans une table de base de données ou de géodatabase comportant une colonne ST-Geometry. Utilisez les fonctions constructeur ST_Geometry pour insérer des types de géométrie spécifiques. Vous pouvez également spécifier que la sortie de certaines fonctions d'opération spatiale soit générée dans une table existante.

Lorsque vous insérez une géométrie dans une table avec SQL, tenez compte de ce qui suit :

- Vous devez spécifier un identifiant de référence spatiale (SRID) valide.
- Toutes les géométries de la même colonne doivent utiliser le même SRID.
- Pour que vous puissiez continuer à utiliser la table avec ArcGIS, le champ utilisé en tant qu'identifiant d'objet ne peut pas être nul ou contenir des valeurs non uniques.

Identifiants de référence spatiale

Le SRID que vous spécifiez lorsque vous insérez dans une table dans Oracle une géométrie qui utilise le type spatial ST_Geometry doit se trouver dans la table ST_SPATIAL_REFERENCES et avoir un enregistrement correspondant dans la table SDE.SPATIAL_REFERENCES. Le SRID que vous spécifiez lorsque vous insérez dans une table dans PostgreSQL une géométrie qui utilise le type spatial ST_Geometry doit se trouver dans la table public.sde_spatial_references. Ces tables sont prérenseignées avec des références spatiales et des SRID.

Le SRID que vous spécifiez lorsque vous insérez dans une table dans SQLite une géométrie qui utilise le type spatial ST_Geometry doit se trouver dans la table st_spatial_reference_systems.

Si vous devez utiliser une référence spatiale personnalisée ne figurant pas dans la table, le plus simple pour cela est d'utiliser pour charger ou créer une classe d'entités possédant les valeurs de référence spatiale que vous souhaitez. Assurez-vous que la classe d'entités que vous créez utilise le stockage ST_Geometry. Cela crée un enregistrement dans les tables SDE.SPATIAL_REFERENCES et ST_SPATIAL_REFERENCES dans Oracle, un enregistrement dans la table public.sde_spatial_references dans PostgreSQL ou un enregistrement dans la table st_aux_spatial_reference_systems_table dans SQLite.

Dans les géodatabases, vous pouvez interroger la table LAYERS (Oracle) ou sde_layers (PostgreSQL) pour découvrir le SRID affecté à la table spatiale. Vous pouvez alors utiliser ce SRID lorsque vous créez des tables spatiales et insérez des données avec SQL.

Remarque :
Dans le but d'utiliser les échantillons de ce document, un enregistrement a été ajouté aux tables ST_SPATIAL_REFERENCES et sde_spatial_references pour indiquer une référence spatiale inconnue. Cet enregistrement possède pour SRID 0. Vous pouvez utiliser ce SRID pour les exemples de ce document. Il ne s'agit toutefois pas d'un SRID officiel ; il est fourni en vue d'exécuter l'exemple de code SQL. Il est recommandé de ne pas utiliser ce SRID pour vos données de production.

Identifiants d'objets

Pour que ArcGIS interroge des données, la table doit contenir un champ d'identifiant d'objet unique.

Les classes d'entités créées avec ArcGIS possèdent toujours un champ d'identifiant d'objet utilisé comme champ d'identifiant. Lors de l'insertion d'enregistrements dans la classe d'entités avec ArcGIS, une valeur unique est toujours insérée dans le champ d'identifiant d'objet. Le champ d'identifiant d'objet dans une table de géodatabase est géré par ArcGIS. Le champ d'identifiant d'objet dans une table de base de données créée dans ArcGIS est géré par le système de gestion de base de données.

Lorsque vous insérez des enregistrements dans une table de géodatabase avec SQL, vous devez insérer une valeur d'identifiant d'objet unique valide.

Les tables de base de données que vous créez en dehors de ArcGIS doivent posséder un champ (ou un ensemble de champs) que ArcGIS peut utiliser en tant qu'identifiant d'objet. Si vous utilisez le type de données d'auto-incrémentation natif de votre base de données pour le champ d'identifiant dans votre table, ce champ est renseigné par la base de données lorsque vous insérez un enregistrement avec SQL. Si vous gérez manuellement les valeurs dans votre champ d'identifiant unique, assurez-vous de fournir une valeur unique pour l'identifiant lorsque vous modifiez la table dans SQL.

Remarque :
Vous ne pouvez pas publier de données à partir de tables ayant un champ d'identifiant unique non géré par ArcGIS ou le système de gestion de base de données.

Modifier des données ST_Geometry avec SQL

Les modifications SQL apportées à des enregistrements existants affectent souvent les attributs non spatiaux stockés dans la table. Toutefois, vous pouvez modifier les données dans la colonne ST_Geometry avec des fonctions constructeur dans des instructions `UPDATE` SQL.

Si les données sont stockées dans une géodatabase, vous devez suivre d'autres directives pour effectuer des modifications avec SQL :

- Ne mettez pas à jour des enregistrements avec SQL si les données ont été inscrites comme versionnées ou activées pour l'archivage de géodatabase.
- Ne modifiez aucun attribut affectant d'autres objets dans la base de données qui participent au comportement de géodatabase (par exemple, des classes de relations, des annotations liées aux entités, une topologie, des règles attributaires ou des réseaux).
- N'utilisez pas SQL pour modifier des structures de table.

⚠ Attention :

Utiliser SQL pour accéder à la géodatabase annule des fonctionnalités de géodatabase, telles que le versionnement, la topologie, les réseaux, les MNT, les annotations liées aux entités ou d'autres extensions d'espace de travail ou de classe. Il peut s'avérer possible d'utiliser des fonctions de système de gestion de base de données, telles que les déclencheurs et les procédures stockées, afin de conserver les relations entre les tables nécessaires à certaines fonctionnalités de géodatabase. Cependant, le fait d'exécuter des commandes SQL sur la géodatabase sans tenir compte de cette fonctionnalité supplémentaire (il peut s'agir, par exemple, de l'exécution d'instructions `INSERT` pour ajouter des enregistrements à une table où l'archivage de géodatabase est activé ou ajouter une colonne à une classe d'entités) contourne les fonctionnalités de géodatabase et peut éventuellement altérer les relations entre les données de votre géodatabase.

Charger la bibliothèque ST_Geometry SQLite

Avant d'exécuter des commandes SQL contenant des fonctions ST_Geometry dans une base de données SQLite, procédez comme suit :

1. Téléchargez le fichier zip ArcGIS Pro ST_Geometry Libraries (SQLite) depuis My Esri et décompressez-le.
2. Installez un éditeur SQL sur la même machine que la base de données.
3. Placez le fichier ST_Geometry dans un emplacement accessible par la base de données SQLite et l'éditeur SQL à partir duquel vous allez charger ST_Geometry.
   Si la base de données SQLite se trouve sur une machine Microsoft Windows, utilisez le fichier `stgeometry_sqlite.dll`. Si la base de données SQLite se trouve sur une machine Linux, utilisez le fichier `libstgeometry_sqlite.so`.
4. Ouvrez l'éditeur SQL et connectez-vous à la base de données SQLite.
5. Chargez la bibliothèque ST_Geometry.
   Dans le premier exemple ci-dessous, la bibliothèque est chargée pour une base de données SQLite sur une machine Windows. Le deuxième exemple charge la bibliothèque pour une base de données SQLite sur une machine Linux.

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

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

Vous pouvez désormais exécuter des commandes SQL qui contiennent des fonctions ST_Geometry dans la base de données SQLite.

Fonctions constructeur pour ST_Geometry

Les fonctions constructeur créent une géométrie à partir d'un texte de description ou d'un fichier binaire connu.

Lorsque vous proposez une description textuelle connue pour créer une géométrie, la coordonnée de mesure doit être indiquée en dernier. Par exemple, si votre texte contient les coordonnées pour x, y, z et m, celles-ci doivent être fournies dans cet ordre.

Une géométrie peut avoir zéro points ou plus. Une géométrie est considérée vide si elle a zéro points. Le sous-type point est la seule géométrie limitée à zéro ou un point ; tous les autres sous-types peuvent en comprendre zéro ou plus.

Les sections suivantes décrivent la superclasse et les sous-classes de géométrie, et fournissent une liste des fonctions permettant de créer l'une ou l'autre.

Vous pouvez également construire des géométries en tant que résultat d'une opération spatiale effectuée sur des géométries existantes.

Superclasse de géométrie

La superclasse ST_Geometry ne peut pas être instanciée ; même si vous pouvez définir une colonne en tant que type ST_Geometry, les données réelles insérées peuvent être définies en tant qu'entités point, chaîne de lignes, polygon, multi-points, multilinestring ou multipolygon.

Vous pouvez utiliser les fonctions suivantes pour créer une superclasse pouvant contenir tous les types d'entités susmentionnés.

- ST_Geometry
- ST_GeomFromText (Oracle et SQLite uniquement)
- ST_GeomFromWKB

Sous-classes

Vous pouvez définir une entité en tant que sous-classe spécifique, auquel cas seul le type d'entité autorisé pour cette sous-classe pourra être inséré. Par exemple, ST_PointFromWKB peut uniquement construire des entités point.

ST_Point

Un objet ST_Point est une géométrie à zéro dimension, occupant une seule localisation dans l'espace de coordonnées. Un objet ST_Point a une seule valeur de coordonnée x,y, est toujours simple et a une limite NULL. Les objets ST_Point permettent de définir des entités telles que les puits de pétrole, les points de repère ou les sites de prélèvement d'échantillons d'eau.

Les fonctions suivantes créent un point :

- ST_Point
- ST_PointFromText (Oracle et SQLite uniquement)
- ST_PointFromWKB

ST_MultiPoint

Un objet ST_MultiPoint est un ensemble d'objets ST_Point et, comme ses éléments, sa dimension est de 0. Un objet ST_MultiPoint est simple si ses éléments occupent tous des espaces de coordonnées différents. La limite d'un objet

ST_MultiPoint est NULL. Les objets ST_MultiPoint permettent de définir par exemple des réseaux de diffusion aérienne ou des foyers de déclaration d'une maladie.

Les fonctions suivantes créent une géométrie multi-points :

- ST_MultiPoint
- ST_MPointFromText (Oracle uniquement)
- ST_MPointFromWKB

ST_LineString

Un objet ST_LineString est un objet linéaire stocké comme une séquence de points définissant un chemin interpolé linéaire. L'objet ST_LineString est simple s'il n'intersecte pas son intérieur. Les extrémités (la limite) d'un objet ST_LineString fermé occupent le même point dans l'espace. Un objet ST_LineString est une boucle s'il est à la fois fermé et simple. Parmi les propriétés héritées de la superclasse ST_Geometry, les objets ST_LineString présentent une longueur. Les objets ST_LineString permettent généralement de définir des entités linéaires telles que les routes, les rivières et les lignes à haute tension.

Les extrémités forment normalement la limite d'un objet ST_LineString, sauf s'il est fermé auquel cas la limite est NULL. L'intérieur d'un objet ST_LineString est le chemin continu entre les extrémités, sauf s'il est fermé auquel cas l'intérieur est continu.

Les fonctions qui créent des objets linestring sont notamment les suivantes :

- ST_LineString
- ST_LineFromText (Oracle et SQLite uniquement)
- ST_LineFromWKB
- ST_Curve (Oracle et SQLite uniquement)

ST_MultiLineString

Un objet ST_MultiLineString est un ensemble d'éléments ST_LineString.

La limite d'un objet ST_MultiLineString est constituée par les extrémités non intersectées des éléments ST_LineString. La limite d'un objet ST_MultiLineString est NULL si toutes les extrémités de tous les éléments sont intersectées. En plus des propriétés héritées de la superclasse ST_Geometry, les objets ST_MultiLineString présentent une longueur. Les objets ST_MultiLineString permettent de définir des entités linéaires discontinues, telles que des cours d'eau ou des réseaux routiers.

Les fonctions qui construisent des objets multilinestring sont les suivantes :

- ST_MultiLineString
- ST_MLineFromText (Oracle et SQLite uniquement)
- ST_MLineFromWKB
- ST_MultiCurve (Oracle uniquement)

ST_Polygon

Un objet ST_Polygon est une surface bidimensionnelle stockée comme une séquence de points définissant sa boucle de contour extérieur et 0 ou plus boucles intérieures. Les objets ST_Polygon sont toujours simples. Les objets

ST_Polygon définissent des entités ayant une étendue spatiale, telle que les parcelles, les plans d'eau et les zones de compétence juridique.

Ce graphique présente des exemples d'objets ST_Polygon : 1 est un objet ST_Polygon dont la limite est définie par une boucle extérieure. 2 est un objet ST_Polygon dont la limite est définie par une boucle extérieure et deux boucles intérieures. La surface à l'intérieur des boucles intérieures fait partie de l'extérieur de l'objet ST_Polygon. 3 est un objet ST_Polygon autorisé, car les boucles s'intersectent en un seul point tangent.

La boucle extérieure et toutes les boucles intérieures définissent la limite d'un objet ST_Polygon, et l'espace situé entre les boucles définit l'intérieur de l'objet ST_Polygon. Les boucles d'un objet ST_Polygon peuvent s'intersecter en un point tangent mais ne peuvent se croiser. En plus des autres propriétés héritées de la superclasse ST_Geometry, les objets ST_Polygon présentent une surface.

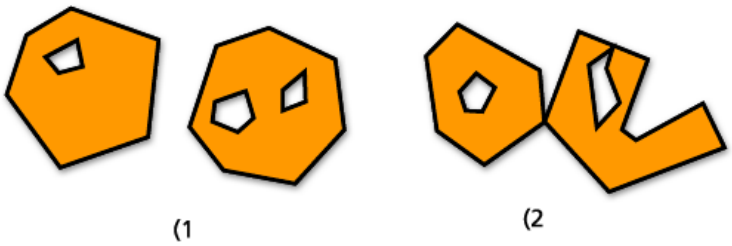
Les fonctions qui créent des objets polygon sont notamment les suivantes :

- ST_Polygon
- ST_PolyFromText (Oracle et SQLite uniquement)
- ST_PolyFromWKB
- ST_Surface (Oracle et SQLite uniquement)

ST_MultiPolygon

La limite d'un objet ST_MultiPolygon est la longueur cumulée des boucles extérieures et intérieures de ses éléments. L'intérieur d'un objet ST_MultiPolygon est défini par les intérieurs cumulés de ses éléments ST_Polygon. La limite d'un objet ST_MultiPolygon ne peut s'intersecter qu'en un point tangent. En plus des propriétés héritées de la superclasse ST_Geometry, les objets ST_MultiPolygon présentent une longueur. Les objets ST_MultiPolygon définissent des entités telles qu'une strate forestière ou une parcelle discontinue, par exemple une chaîne d'îlots du Pacifique.

L'illustration suivante montre des exemples d'objets ST_MultiPolygon : 1 est un objet ST_MultiPolygon avec deux éléments ST_Polygon. Sa limite est définie par les deux boucles extérieures et les trois boucles intérieures. 2 est également un objet ST_MultiPolygon avec deux éléments ST_Polygon, mais sa limite est définie par les deux boucles extérieures et les deux boucles intérieures, et les deux éléments ST_Polygon s'intersectent en un point tangent.



Les fonctions suivantes créent des objets multipolygon :

- ST_MultiPolygon
- ST_MPolyFromText (Oracle et SQLite uniquement)
- ST_MPolyFromWKB
- ST_MultiSurface (Oracle uniquement)

Création de géométries à partir de géométries existantes

Bien qu'il ne s'agisse pas strictement de fonctions constructeur, les fonctions suivantes renvoient une nouvelle géométrie en prenant les géométries existantes comme entrée et en effectuant des analyses dessus :

- ST_Aggr_ConvexHull (Oracle et SQLite uniquement)
- ST_Aggr_Intersection (Oracle et SQLite uniquement)
- 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

Accesseurs spatiaux pour ST_Geometry

Les accesseurs spatiaux renvoient la propriété d'une géométrie. Certains accesseurs permettent de déterminer les propriétés suivantes d'une entité ST_Geometry :

Dimensionnalité

Les dimensions d'une géométrie sont les coordonnées minimales (aucune, x, y) requises pour définir l'étendue spatiale de la géométrie.

Une géométrie peut avoir une dimension de 0, 1 ou 2.

Ces dimensions sont les suivantes :

- 0 — N'a ni longueur ni surface
- 1 - A une longueur (x ou y)
- 2 - Contient une surface (x et y)

Les sous-types point et multipoint ont une dimension de 0. Les points représentent des entités à zéro dimension pouvant être modélisées avec une seule coordonnée, alors que les objets multi-points représentent des données devant être modélisées avec une grappe de coordonnées non connectées.

Les sous-types linestring et multilinestring ont une dimension de 1. Ils stockent des entités telles que des segments de route, des cours d'eau ramifiés et toute autre entité de nature linéaire.

Les sous-types polygon et multipolygon ont une dimension de 2. Il s'agit de peuplements forestiers, de parcelles, de plans d'eau et d'autres entités ayant un périmètre qui délimite une surface définissable pouvant être restituée par le type de données polygon ou multipolygon.

La dimension est importante non seulement comme propriété du sous-type mais également pour déterminer la relation spatiale entre deux entités. La dimension de l'entité ou des entités résultantes indique si l'opération a réussi ou non. Les accesseurs spatiaux examinent les dimensions des entités pour déterminer leur mode de comparaison.

La fonction ST_Dimension permet d'évaluer la dimension d'une géométrie : elle part d'une entité ST_Geometry et renvoie sa dimension sous forme de nombre entier.

Les coordonnées d'une géométrie ont également des dimensions. Si une géométrie a uniquement des coordonnées x et y, la dimension des coordonnées est 2. Si une géométrie a des coordonnées x, y et z, la dimension des coordonnées est 3. Si une géométrie a des coordonnées x, y, z et m, la dimension des coordonnées est 4.

Vous pouvez utiliser la fonction ST_CoordDim pour déterminer les dimensions de coordonnées présentes dans une géométrie.

Coordonnées z

Certaines géométries ont une altitude ou une profondeur associée, une troisième dimension. Chacun des points qui forment la géométrie d'une entité peut comprendre une coordonnée z facultative qui représente une altitude ou une profondeur par rapport à la surface de la terre.

Le prédicat ST_Is3d prend un élément ST_Geometry comme entrée et renvoie true si la fonction a des coordonnées z ou false dans le cas contraire.

Vous pouvez déterminer la coordonnée z d'un point à l'aide de la fonction ST_Z.

La fonction ST_MaxZ renvoie la coordonnée z maximale et la fonction ST_MinZ renvoie la coordonnée z minimale

d’une géométrie.

Mesures

Les mesures sont des valeurs attribuées à chaque coordonnée. Elles servent pour les applications de référencement linéaire et de segmentation dynamique. Par exemple, les localisations des bornes kilométriques situées le long d'une autoroute peuvent contenir des mesures indiquant leur position. La valeur peut représenter toute donnée stockable comme nombre à double précision.

Le prédicat ST_IsMeasured part d’une géométrie et renvoie true si elle contient des mesures et false dans le cas contraire. Cette fonction est utilisée avec les implémentations Oracle et SQLite de ST_Geometry uniquement.

Vous pouvez découvrir la valeur de mesure d’un point à l’aide de la fonction ST_M.

La fonction ST_MaxM renvoie la coordonnée m maximale et la fonction ST_MinM renvoie la coordonnée m minimale d’une géométrie.

Type de géométrie

Le type de géométrie fait référence au type d'entité géométrique. Il s'agit notamment des entités suivantes :

- Points et multi-points
- Lignes et multilignes
- Polygones et multipolygones

ST_Geometry est une superclasse qui peut stocker différents sous-types. Pour déterminer le sous-type d’une géométrie, utilisez la fonction ST_GeometryType ou ST_Entity (Oracle et SQLite uniquement).

Ensemble de points (sommets) et nombre de points

Une géométrie peut avoir zéro points ou plus. Une géométrie est considérée vide si elle a zéro points. Le sous-type point est la seule géométrie limitée à zéro ou un point ; tous les autres sous-types peuvent en comprendre zéro ou plus.

ST_Point

Un objet ST_Point est une géométrie à zéro dimension, occupant une seule localisation dans l'espace de coordonnées. Un objet ST_Point a une seule valeur de coordonnée x,y, est toujours simple et a une limite NULL. Les objets ST_Point permettent de définir des entités telles que les puits de pétrole, les points de repère ou les sites de prélèvement d'échantillons d'eau.

Parmi les fonctions applicables uniquement au type de données ST_Point, citons :

- ST_X - Renvoie la valeur de coordonnée x d'un type de données ponctuel comme nombre à double précision
- ST_Y - Renvoie la valeur de coordonnée y d'un type de données ponctuel comme nombre à double précision
- ST_Z- Renvoie la valeur de coordonnée z d'un type de données ponctuel comme nombre à double précision
- ST_M - Renvoie la valeur de coordonnée m d'un type de données ponctuel comme nombre à double précision

ST_MultiPoint

Un objet ST_MultiPoint est un ensemble d'objets ST_Point et, comme ses éléments, sa dimension est de 0. Un objet ST_MultiPoint est simple si ses éléments occupent tous des espaces de coordonnées différents. La limite d'un objet ST_MultiPoint est NULL. Les objets ST_MultiPoint permettent de définir par exemple des réseaux de diffusion

aérienne ou des foyers de déclaration d'une maladie.

Vous pouvez utiliser la fonction ST_NumGeometries pour déterminer le nombre de points dans une géométrie multi-points.

Longueur, aire et périmètre

La longueur, l'aire et le périmètre sont des mesures propres à la géométrie. Les objets de chaîne de lignes et les éléments des objets multilinestring sont unidimensionnels et ont comme caractéristique la longueur. Les polygones et autres éléments multi-surfaciques désignent des surfaces bidimensionnelles. Par conséquent, vous pouvez mesurer leur aire et leur périmètre. Utilisez les fonctions ST_Length, ST_Area et ST_Perimeter pour déterminer ces propriétés. Les unités de mesure varient selon le stockage des données.

ST_LineString

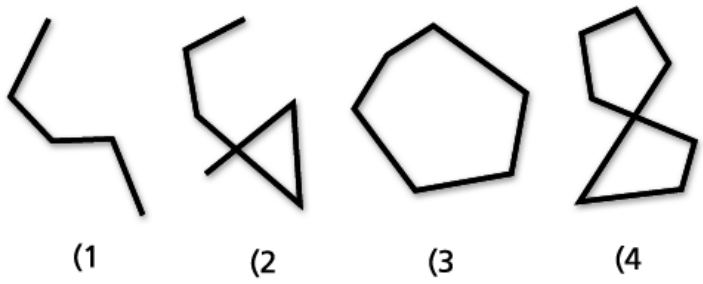
Un objet ST_LineString est un objet linéaire stocké comme une séquence de points définissant un chemin interpolé linéaire. L'objet ST_LineString est simple s'il n'intersecte pas son intérieur. Les extrémités (la limite) d'un objet ST_LineString fermé occupent le même point dans l'espace. Un objet ST_LineString est une boucle s'il est à la fois fermé et simple. Parmi les propriétés héritées de la superclasse ST_Geometry, les objets ST_LineString présentent une longueur. Les objets ST_LineString permettent généralement de définir des entités linéaires telles que les routes, les rivières et les lignes à haute tension.

Les extrémités forment normalement la limite d'un objet ST_LineString, sauf s'il est fermé auquel cas la limite est NULL. L'intérieur d'un objet ST_LineString est le chemin continu entre les extrémités, sauf s'il est fermé auquel cas l'intérieur est continu.

Parmi les fonctions applicables aux objets ST_LineString, citons :

- ST_StartPoint - Renvoie le premier point de l'objet ST_LineString donné
- ST_EndPoint - Renvoie le dernier point d'un objet ST_LineString
- ST_IsClosed - Prédicat qui renvoie true si l'objet ST_LineString spécifié est fermé (les deux extrémités de l'objet linestring s'intersectent) et false dans le cas contraire
- ST_IsRing - Prédicat qui renvoie true si l'objet ST_LineString spécifié est une boucle et false dans le cas contraire
- ST_Length - Renvoie la longueur d'un objet ST_LineString comme nombre à double précision
- ST_NumPoints - Evalue un objet ST_LineString et renvoie le nombre de points de sa séquence comme nombre entier
- ST_PointN - Part d'un objet ST_LineString et de l'index d'un point énième et renvoie ce point

L'illustration ci-dessous montre des exemples d'objets ST_LineString : (1 est un objet ST_LineString simple, non fermé ; (2 est un objet ST_LineString non simple, non fermé ; (3 est un objet ST_LineString fermé, simple et donc une boucle ; enfin, (4 est un objet ST_LineString fermé, non simple, mais ce n'est pas une boucle.

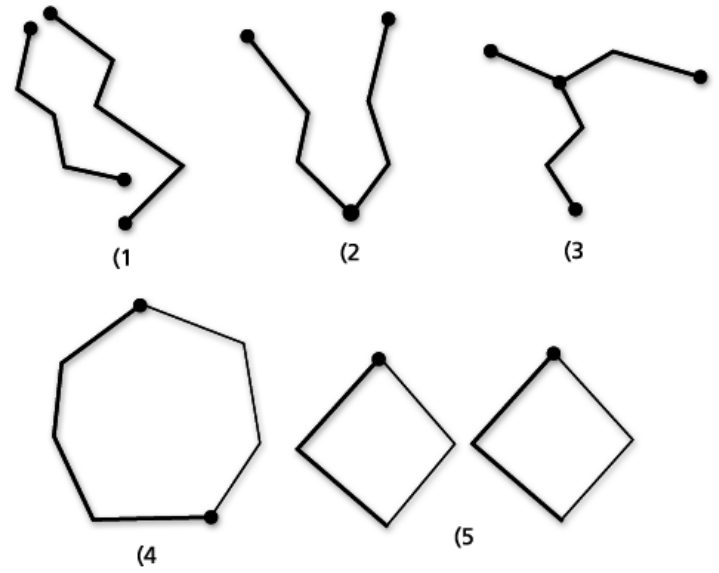


ST_MultiLineString

Un objet ST_MultiLineString est un ensemble d'éléments ST_LineString. Les objets ST_MultiLineString sont simples s'ils ne s'intersectent qu'aux extrémités des éléments ST_LineString. Les objets ST_MultiLineString sont non simples si les intérieurs des éléments ST_LineString s'intersectent.

La limite d'un objet ST_MultiLineString est constituée par les extrémités non intersectées des éléments ST_LineString. La limite d'un objet ST_MultiLineString est NULL si toutes les extrémités de tous les éléments sont intersectées. En plus des propriétés héritées de la superclasse ST_Geometry, les objets ST_MultiLineString présentent une longueur. Les objets ST_MultiLineString permettent de définir des entités linéaires discontinues, telles que des cours d'eau ou des réseaux routiers.

L'illustration suivante montre des exemples d'objets ST_MultiLineString : (1 est un objet ST_MultiLineString simple dont la limite est constituée par les quatre extrémités de ses deux éléments ST_LineString. (2 est un objet ST_MultiLineString simple car seules les extrémités des éléments ST_LineString s'intersectent. La limite est constituée par les deux extrémités non intersectées. (3 est un objet ST_MultiLineString non simple, car l'intérieur de l'un de ses éléments ST_LineString est intersecté. La limite de cet objet ST_MultiLineString est constituée par les trois extrémités non intersectées. (4 est un objet ST_MultiLineString simple non fermé. Il n'est pas fermé car ses éléments ST_LineString ne sont pas fermés. Il est simple car aucun des intérieurs des éléments ST_LineString n'est intersecté. (5 est un objet ST_MultiLineString unique, simple et fermé. Il est fermé car tous ses éléments sont fermés. Il est simple car aucun de ses éléments n'intersecte les intérieurs.



Parmi les fonctions applicables aux objets ST_MultiLineString, citons :

- ST_IsClosed - Ce prédicat renvoie la valeur true si l'objet ST_MultiLineString indiqué est fermé et la valeur false dans le cas contraire.
- ST_Length - Cette fonction évalue un objet ST_MultiLineString et renvoie la longueur cumulée de tous ses éléments ST_LineString sous forme de nombre à double précision.
- ST_NumGeometries - Cette fonction renvoie le nombre de lignes figurant dans un objet multilinestring.

ST_Polygon

Un objet ST_Polygon est une surface bidimensionnelle stockée comme une séquence de points définissant sa boucle de contour extérieur et 0 ou plus boucles intérieures. Les objets ST_Polygon sont toujours simples. Les objets ST_Polygon définissent des entités ayant une étendue spatiale, telle que les parcelles, les plans d'eau et les zones de compétence juridique.

Ce graphique présente des exemples d'objets ST_Polygon : 1 est un objet ST_Polygon dont la limite est définie par une boucle extérieure. 2 est un objet ST_Polygon dont la limite est définie par une boucle extérieure et deux boucles intérieures. La surface à l'intérieur des boucles intérieures fait partie de l'extérieur de l'objet ST_Polygon. 3 est un objet ST_Polygon autorisé, car les boucles s'intersectent en un seul point tangent.

La boucle extérieure et toutes les boucles intérieures définissent la limite d'un objet ST_Polygon, et l'espace situé entre les boucles définit l'intérieur de l'objet ST_Polygon. Les boucles d'un objet ST_Polygon peuvent s'intersecter en un point tangent mais ne peuvent se croiser. En plus des autres propriétés héritées de la superclasse ST_Geometry, les objets ST_Polygon présentent une surface.

Parmi les fonctions applicables à un objet ST_Polygon, citons :

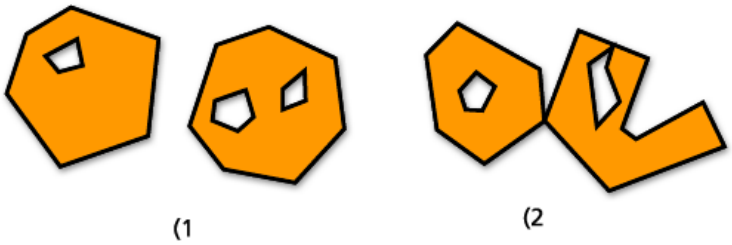
- ST_Area - Renvoie la surface d'un objet ST_Polygon comme nombre à double précision
- ST_Centroid - Renvoie un objet ST_Point représentant le centre de l'enveloppe de l'objet ST_Polygon
- ST_ExteriorRing - Renvoie la boucle extérieure d'un objet ST_Polygon comme objet ST_LineString
- ST_InteriorRingN - Évalue un objet ST_Polygon et un index, et renvoie la énième boucle intérieure comme objet ST_LineString
- ST_NumInteriorRing - Renvoie le nombre de boucles intérieures d'un objet ST_Polygon
- ST_PointOnSurface - Renvoie un objet ST_Point nécessairement situé dans la surface de l'objet ST_Polygon donné

ST_MultiPolygon

La limite d'un objet ST_MultiPolygon est la longueur cumulée des boucles extérieures et intérieures de ses éléments. L'intérieur d'un objet ST_MultiPolygon est défini par les intérieurs cumulés de ses éléments ST_Polygon. La limite d'un objet ST_MultiPolygon ne peut s'intersecter qu'en un point tangent. En plus des propriétés héritées de la superclasse ST_Geometry, les objets ST_MultiPolygon présentent une longueur. Les objets ST_MultiPolygon

définissent des entités telles qu'une strate forestière ou une parcelle discontinue, par exemple une chaîne d'îlots du Pacifique.

L'illustration suivante montre des exemples d'objets ST_MultiPolygon : 1 est un objet ST_MultiPolygon avec deux éléments ST_Polygon. Sa limite est définie par les deux boucles extérieures et les trois boucles intérieures. 2 est également un objet ST_MultiPolygon avec deux éléments ST_Polygon, mais sa limite est définie par les deux boucles extérieures et les deux boucles intérieures, et les deux éléments ST_Polygon s'intersectent en un point tangent.



Parmi les fonctions applicables aux objets ST_MultiPolygon, citons :

- ST_Area - Renvoie un nombre à double précision représentant la valeur ST_Area cumulée des éléments ST_Polygon d'un objet ST_MultiPolygon.
- ST_Centroid - Renvoie un objet ST_Point qui est le centre de l'enveloppe d'un objet ST_MultiPolygon.
- ST_NumGeometries - Renvoie le nombre de polygones dans un objet MultiPolygon.
- ST_PointOnSurface - Évalue un objet ST_MultiPolygon et renvoie un objet ST_Point nécessairement situé dans la surface de l'un de ses éléments ST_Polygon.

Géométries simples dans une géométrie multi-parties

Les géométries multi-parties sont composées de géométries individuelles simples.

Il est possible de déterminer le nombre de géométries individuelles comprises dans une géométrie multi-parties, telles qu'un objet ST_MultiPoint, ST_MultiLineString ou ST_MultiPolygon. Pour cela, utilisez le prédicat ST_NumGeometries. Cette fonction renvoie le nombre d'éléments individuels d'un ensemble de géométries.

Avec la fonction ST_GeometryN, vous pouvez déterminer quelle géométrie dans la géométrie multi-parties existe à la position N, N étant un nombre que vous fournissez avec la fonction. Par exemple, si vous souhaitez retourner le troisième point d'une géométrie multi-points, vous devez inclure 3 quand vous exécutez la fonction.

Pour renvoyer les géométries individuelles et leur position à partir d'une géométrie multi-parties dans PostgreSQL, utilisez la fonction ST_GeomFromCollection.

Intérieur, limite, extérieur

Toutes les géométries occupent une position dans l'espace, définie par leurs intérieurs, leurs limites et leurs extérieurs. L'extérieur d'une géométrie est l'ensemble de l'espace non occupé par la géométrie. L'intérieur est l'espace occupé par la géométrie. La limite d'une géométrie est la localisation entre son intérieur et son extérieur. Le code hérite directement des propriétés intérieures et extérieures ; toutefois, la propriété de limite diffère pour chaque code.

Utilisez la fonction ST_Boundary pour déterminer la limite du ST_Geometry source.

Simple ou non simple

Certains sous-types ST_Geometry sont toujours simples, tels que ST_Point ou ST_Polygon. Toutefois, les sous-types

ST_LineString, ST_MultiPoint et ST_MultiLineString peuvent être simples ou non simples. Ils sont simples s'ils obéissent à l'ensemble des règles topologiques qui leur sont imposées et non simple sinon.

Les règles topologiques incluent les éléments suivants :

- Un objet ST_LineString est simple s'il n'intersecte pas son intérieur et non simple sinon.
- Un objet ST_MultiPoint est simple si ses deux éléments occupent des espaces de coordonnées différents (ont des coordonnées x,y différents) et non simple sinon.
- Un objet ST_MultiLineString est simple si aucun des intérieurs de ses éléments n'est intersecté par son propre intérieur et non simple si les intérieurs de ses éléments s'intersectent.

Le prédicat ST_IsSimple permet de déterminer si un ST_LineString, ST_MultiPoint, ou ST_MultiLineString est simple ou non simple. ST_IsSimple part d'un objet ST_Geometry et renvoie true si la géométrie est simple, et false dans le cas contraire.

Vide ou non vide

Une géométrie est vide si elle n'a pas de points. Une géométrie vide a une enveloppe, une limite, un intérieur et un extérieur de valeur nulle. Une géométrie vide est toujours simple. Les objets linestring et multilinestring vides ont une longueur de 0. Les objets polygon et multipolygon vides ont une surface de 0.

Le prédicat ST_IsEmpty permet de déterminer si une géométrie est vide. Il analyse un objet ST_Geometry et renvoie true si la géométrie est vide, et false dans le cas contraire.

Fermé/Boucle

Les géométries de type linestring peuvent être fermées ou être des boucles. Les objets linestring peuvent être fermés sans être des boucles. Vous pouvez déterminer si un objet linestring est fermé en utilisant le prédicat ST_IsClosed ; il renvoie true si les deux extrémités de l'objet linestring s'intersectent. Les boucles sont des objets linestring fermés et simples. Le prédicat ST_IsRing permet de tester si un objet linestring est vraiment une boucle ; il renvoie true si l'objet linestring est fermé et simple.

Enveloppe

Chaque géométrie a une enveloppe. L'enveloppe d'une géométrie est la géométrie de son emprise, formée par ses coordonnées x,y minimales et maximales. S'agissant des géométries de point, les coordonnées x,y minimales et maximales étant identiques, un rectangle (ou enveloppe) est créé autour de ces coordonnées. Dans le cas des géométries de ligne, les extrémités de la ligne représentent deux côtés de l'enveloppe. Les deux autres côtés sont créés juste au-dessus et au-dessous de la ligne.

La fonction ST_Envelope part d'un objet ST_Geometry et renvoie un objet ST_Geometry représentant l'enveloppe de l'objet source.

Pour trouver les coordonnées x,y individuelles minimales et maximales d'une géométrie, utilisez les fonctions ST_MinX, ST_MinY, ST_MaxX et ST_MaxY.

Système de référence spatiale

Le système de référence spatiale identifie la matrice de transformation de coordonnées de chaque géométrie. Il est composé d'un système de coordonnées, d'une résolution et d'une tolérance.

Tous les systèmes de référence spatiale connus de la géodatabase sont stockés dans une table système de géodatabase.

Les fonctions suivantes permettent d'obtenir des informations sur les systèmes de référence spatiale des géométries :

- ST_SRID - Part d'un objet ST_Geometry et renvoie son identifiant de référence spatiale (SRID) sous forme de nombre entier.
- ST_Equalsrs - Détermine si les systèmes de référence spatiale de deux classes d'entités différentes sont identiques (true) ou non (false).

Taille des entités (PostgreSQL uniquement)

Les entités (enregistrements spatiaux dans une table) acceptent un certain espace de stockage en octets. Vous pouvez utiliser la fonction ST_GeoSize pour déterminer la taille de chaque entité dans une table.

Définitions textuelles et binaires d'une géométrie

Pour obtenir la définition textuelle ou binaire de la géométrie sur une ligne spécifique de la table spatiale, utilisez les fonctions ST_AsText et ST_AsBinary, respectivement.

Relations spatiales

Une fonction principale d'un SIG est de déterminer les relations spatiales existant entre les entités : se chevauchent-elles ? L'une est-elle contenue dans l'autre ? L'une croise-t-elle l'autre ?

Les géométries peuvent être spatialement liées de différentes manières. Les exemples suivants montrent comment une géométrie peut être spatialement liée à une autre.

- La géométrie A traverse la géométrie B.
- La géométrie A est entièrement contenue dans la géométrie B.
- La géométrie B est entièrement contenue dans la géométrie A.
- Les géométries ne s'intersectent ou ne se touchent pas les unes les autres.
- Les géométries coïncident entièrement.
- Les géométries se superposent les unes les autres.
- Les géométries se touchent à un point.

Pour déterminer si ces relations existent, utilisez les fonctions de relation spatiale. Ces fonctions comparent les propriétés suivantes des géométries que vous spécifiez dans une requête :

- L'extérieur (E) des géométries - L'extérieur représente la totalité de l'espace non occupé par une géométrie.
- L'intérieur (I) des géométries - L'intérieur représente l'espace occupé par une géométrie.
- La limite (L) des géométries - La limite est l'interface entre l'intérieur d'une géométrie et son extérieur.

Lorsque vous construisez une requête de relation spatiale, spécifiez le type de relation spatiale que vous recherchez et les géométries que vous souhaitez comparer. Les requêtes renvoient soit "true" soit "false". En d'autres termes, les géométries participent ou non les unes aux autres dans la relation spatiale spécifiée. Dans la plupart des cas, vous utilisez une requête de relation spatiale pour filtrer un résultat défini en le plaçant dans la clause `WHERE`.

Par exemple, si vous disposez d'une table qui stocke les emplacements de sites de développement proposés et d'une autre qui stocke l'emplacement de sites archéologiquement significatifs, vous pouvez exécuter une requête pour vous assurer qu'aucun des sites de développement n'intersecte des sites archéologiques et, si certains le font, renvoyez l'identifiant de ces développement proposés. Dans cet exemple, la fonction ST_Disjoint est utilisée dans PostgreSQL.

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

projname        siteid

bow wow chow    A1009
```

Cette requête renvoie le nom du développement (projname) et l'identifiant du site archéologique (siteid) qui ne sont pas disjoint (en d'autres termes, les sites qui s'intersectent). Elle renvoie un projet de développement, Bow Wow Chow, qui intersecte le site archéologique A1009.

Fonctions relationnelles pour ST_Geometry

Les fonctions relationnelles utilisent des prédicats pour tester différents types de relations spatiales. Pour ce faire, les tests comparent les relations entre les éléments suivants :

- L'extérieur (E) des géométries - L'extérieur représente la totalité de l'espace non occupé par une géométrie.
- L'intérieur (I) des géométries - L'intérieur représente l'espace occupé par une géométrie.
- La limite (L) des géométries - La limite est l'interface entre l'intérieur d'une géométrie et son extérieur.

Les prédicats testent les relations. Ils renvoient 1 (Oracle et SQLite) ou t (PostgreSQL) si une comparaison correspond aux critères de la fonction ; sinon, ils renvoient 0 (Oracle et SQLite) ou f (PostgreSQL). Les prédicats qui testent une relation spatiale comparent des paires de géométries qui peuvent être de dimension ou type différent.

Les prédicats comparent les coordonnées x et y des géométries envoyées. Les coordonnées z et les valeurs de mesure sont ignorées, le cas échéant. Les géométries qui comportent des coordonnées z ou des mesures peuvent être comparées avec celles qui n'en contiennent pas.

La matrice DE-9IM (Dimensionally Extended 9 Intersection Model (DE-9IM) développée par Clementini, et al. étend dimensionnellement la matrice 9 Intersection Model d'Egenhofer et Herring. DE-9IM est une approche mathématique qui définit la relation spatiale par paire entre des géométries de dimensions et types différents. Ce modèle exprime les relations spatiales entre tous les types de géométrie sous forme d'intersections par paire de leur intérieur, limite et extérieur, en prenant en compte la dimension des intersections générées.

Les géométries indiquées a et b, I(a), B(a) et E(a) représentent l'intérieur, la limite et l'extérieur de a, tandis que I(b), B(b) et E(b) représentent l'intérieur, la limite et l'extérieur de b. Les intersections de I(a), B(a) et E(a) avec I(b), B(b) et E(b) génèrent une matrice trois par trois. Chaque intersection peut générer des géométries de dimensions différentes. Par exemple, l'intersection des limites de deux polygones pourrait être composée d'un point et d'une chaîne de lignes. Dans ce cas, la fonction dim (dimension) renverrait la dimension maximale 1.

La fonction dim renvoie la valeur -1, 0, 1 ou 2. La valeur -1 correspond à l'ensemble nul qui est renvoyé lorsqu'aucune intersection n'est trouvée ou dim(Ãƒ˜).

	Ad-Dākhilīyah	**Limite**	**Extérieur**
Ad-Dākhilīyah	dim(I(a) s'intersecte avec I(b))	dim(I(a) s'intersecte avec B(b))	dim(I(a) s'intersecte avec E(b))
Limite	dim(B(a) s'intersecte avec I(b))	dim(B(a) s'intersecte avec B(b))	dim(B(a) s'intersecte avec E(b))
Extérieur	dim(E(a) s'intersecte avec I(b))	dim(E(a) s'intersecte avec B(b))	dim(E(a) s'intersecte avec E(b))

Exemple d'intersection d'un prédicat

Les résultats des prédicats de relation spatiale peuvent être interprétés ou vérifiés en comparant les résultats du prédicat avec une matrice modèle qui représente les valeurs acceptables pour la matrice DE-9IM.

La matrice modèle contient les valeurs acceptables pour chaque cellule de la matrice d'intersection. Les valeurs de modèle possibles sont les suivantes :

T - Il doit exister une intersection ; dim = 0, 1 ou 2

F - Il ne doit pas exister d'intersection ; dim = -1

* - Le fait qu'il existe ou non une intersection n'a pas d'importance ; dim = -1, 0, 1 ou 2

0 - Il doit exister une intersection et sa dimension maximale doit être 0 ; dim = 0

1 - Il doit exister une intersection et sa dimension maximale doit être 1 ; dim = 1

2 - Il doit exister une intersection et sa dimension maximale doit être 2 ; dim = 2

Chaque prédicat comporte au moins une matrice modèle, mais certains en nécessitent plusieurs pour décrire les relations de diverses associations de types de géométries.

La matrice modèle du prédicat ST_Within pour les associations de géométries prend la forme suivante :

			Géométrie b	
		Ad-Dākhilīyah	**Limite**	**Extérieur**
Géométrie a	**Ad-Dākhilīyah**	J	*	F
	Limite	*	*	F
	Extérieur	*	*	*

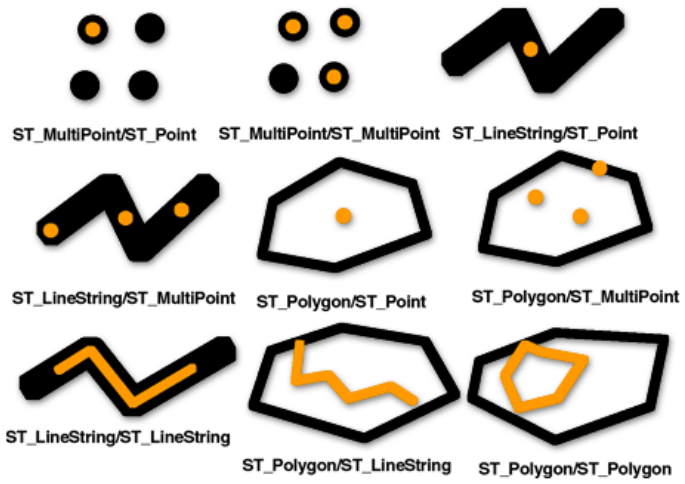
Exemple de matrice modèle

Le prédicat ST_Within renvoie la valeur true lorsque les intérieurs des deux géométries s'intersectent, et que l'intérieur et la limite de la géométrie a ne s'intersectent pas avec l'extérieur de la géométrie b. Toutes les autres conditions ne sont pas prises en compte.

Les sections ci-dessous décrivent les différents prédicats utilisés pour les relations spatiales. Dans les diagrammes de ces sections, la première géométrie en entrée répertoriée apparaît en noir, tandis que la deuxième s'affiche en orange.

ST_Contains

ST_Contains renvoie 1 ou t (true) si la deuxième géométrie est entièrement contenue dans la première géométrie. Le prédicat ST_Contains renvoie le résultat exactement contraire du prédicat ST_Within.

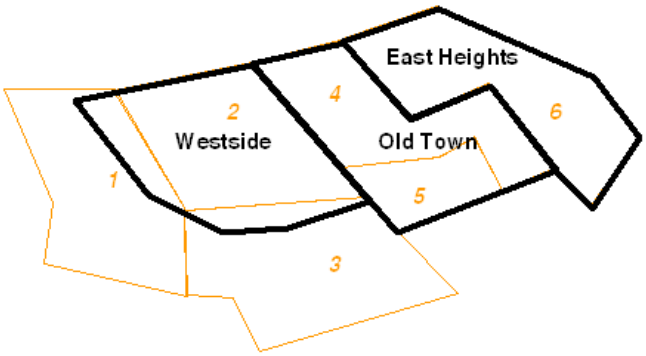


La matrice modèle du prédicat ST_Contains définit que les intérieurs des deux géométries doivent s'intersecter, et que l'intérieur et la limite de la géométrie secondaire (géométrie b) ne doivent pas s'intersecter avec l'extérieur de la géométrie principale (géométrie a).

			Géométrie b	
		Ad-Dākhilīyah	Limite	Extérieur
Géométrie a	Ad-Dākhilīyah	J	*	*
	Limite	*	*	*
	Extérieur	F	F	*

Matrice ST_Contains

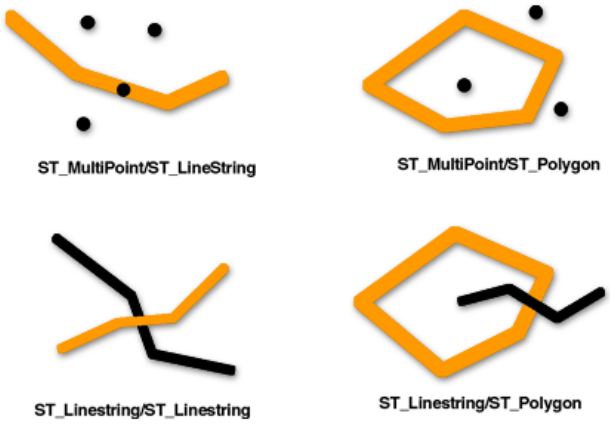
Les fonctions ST_Within et ST_Contains identifient uniquement les géométries qui se trouvent entièrement à l'intérieur d'une autre géométrie. Cela permet d'éliminer de votre sélection les entités qui pourraient fausser vos résultats. Dans l'exemple ci-dessous, un marchand de glaces itinérant veut déterminer quels quartiers comportent le plus d'enfants (clients potentiels) afin de limiter son itinéraire à ces zones. Le marchand compare les polygones des quartiers désignés aux secteurs de recensement, qui possèdent un attribut indiquant le nombre total d'enfants de moins de 16 ans.



Sauf si tous les enfants qui résident dans le secteur de recensement 1 et le secteur de recensement 3 habitent dans les micropolygones de terrain qui se trouvent à l'intérieur de Westside, l'inclusion de ces secteurs dans la sélection pourrait faire augmenter de façon erronée le nombre d'enfants dans le quartier de Westside. En spécifiant de n'inclure que les secteurs de recensement entièrement situés à l'intérieur de certains quartiers (ST_Within = 1), le marchand de glaces peut économiser du temps et de l'argent en ne se rendant pas dans ces parties de Westside.

ST_Crosses

ST_Crosses renvoie 1 ou t (true) si l'intersection génère une géométrie comportant une dimension inférieure de un à la dimension maximale des deux géométries source et que l'ensemble d'intersections est intérieur aux deux géométries source. ST_Crosses renvoie 1 ou t (true) uniquement pour les associations ST_MultiPoint/ST_Polygon, ST_MultiPoint/ST_LineString, ST_Linestring/ST_LineString, ST_LineString/ST_Polygon et ST_LineString/ ST_MultiPolygon.



La matrice modèle suivante du prédicat ST_Crosses s'applique à ST_MultiPoint/ST_LineString, ST_MultiPoint/ST_MultiLineString, ST_MultiPoint/ST_Polygon, ST_MultiPoint/ST_MultiPolygon, ST_LineString/ST_Polygon et ST_LineString/ST_MultiPolygon. La matrice définit que les intérieurs doivent s'intersecter et qu'au moins l'intérieur de la géométrie principale (géométrie a) doit s'intersecter avec l'extérieur de la géométrie secondaire (géométrie b).

			Géométrie b	
		Ad-Dākhilīyah	**Limite**	**Extérieur**
Géométrie a	**Ad-Dākhilīyah**	J	*	J
	Limite	*	*	*
	Extérieur	*	*	*

Matrice ST_Crosses 1

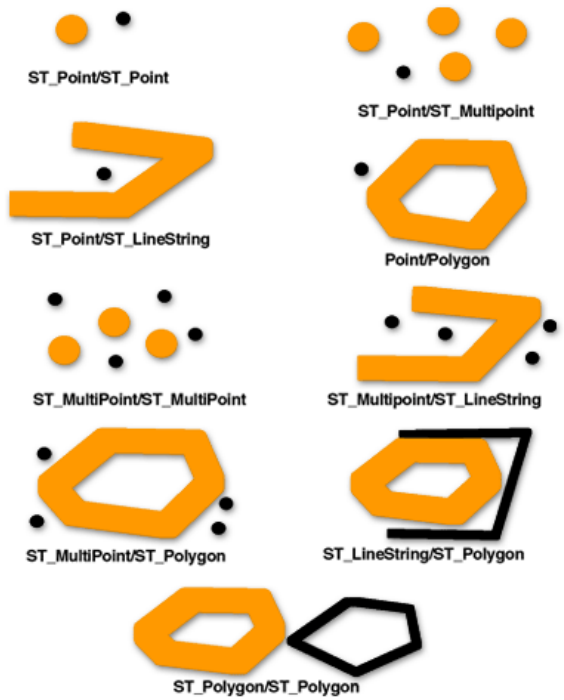
La matrice suivante du prédicat ST_Crosses s'applique à ST_LineString/ST_LineString, ST_LineString/ST_MultiLineString et ST_MultiLineString/ST_MultiLineString. La matrice définit que la dimension de l'intersection des intérieurs doit être 0 (intersection sur un point). Si la dimension de cette intersection est 1 (intersection sur une chaîne de lignes), le prédicat ST_Crosses renvoie false, tandis que le prédicat ST_Overlaps renvoie true.

			Géométrie b	
		Ad-Dākhilīyah	**Limite**	**Extérieur**
Géométrie a	**Ad-Dākhilīyah**	0	*	*
	Limite	*	*	*
	Extérieur	*	*	*

Matrice ST_Crosses 2

ST_Disjoint

ST_Disjoint renvoie 1 ou t (true) si l'intersection des deux géométries est un ensemble vide. En d'autres termes, les géométries sont disjointes si elles ne s'intersectent pas.



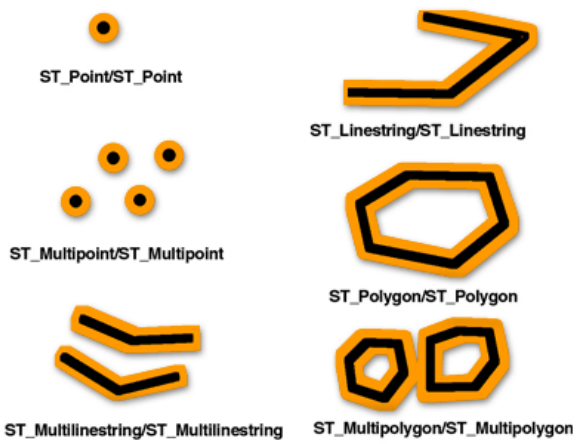
La matrice modèle du prédicat ST_Disjoint définit que ni les intérieurs ni les limites de l'une des géométries ne s'intersectent.

			Géométrie b	
		Ad-Dākhilīyah	**Limite**	**Extérieur**
Géométrie a	**Ad-Dākhilīyah**	**F**	**F**	*****
	Limite	**F**	**F**	*****
	Extérieur	*	*	*

Matrice ST_Disjoint

ST_Equals

ST_Equals renvoie 1 ou t (true) si deux géométries du même type possèdent des valeurs de coordonnées x,y identiques. Le premier et le deuxième étages d'un immeuble de bureaux peuvent avoir des coordonnées x,y identiques et par conséquent les géométries peuvent être égales. ST_Equals peut également identifier si deux entités ont été placées par erreur l'une au-dessus de l'autre.



La matrice modèle DE-9IM pour l'égalité s'assure que les intérieurs s'intersectent et qu'aucune partie de l'intérieur ou de la limite de l'une des géométries ne s'intersecte avec l'extérieur de l'autre.

			Géométrie b	
		Ad-Dākhilīyah	**Limite**	**Extérieur**
Géométrie a	**Ad-Dākhilīyah**	J	*	F
	Limite	*	*	F
	Extérieur	F	F	*

Matrice ST_Equals

ST_Intersects

ST_Intersects renvoie 1 ou t (true) si l'intersection ne génère pas d'ensemble vide. ST_Intersects renvoie le résultat exactement contraire de ST_Disjoint.

Le prédicat ST_Intersects renvoie true si les conditions de l'une des matrices modèles suivantes renvoient true.

Le prédicat ST_Intersects renvoie true si les intérieurs des deux géométries s'intersectent.

			Géométrie b	
		Ad-Dākhilīyah	**Limite**	**Extérieur**
Géométrie a	**Ad-Dākhilīyah**	J	*	*
	Limite	*	*	*
	Extérieur	*	*	*

Matrice ST_Intersects 1

Le prédicat ST_Intersects renvoie true si l'intérieur de la première géométrie s'intersecte avec la limite de la deuxième géométrie.

			Géométrie b	
		Ad-Dākhilīyah	**Limite**	**Extérieur**
Géométrie a	**Ad-Dākhilīyah**	*	J	*

Matrice ST_Intersects 2

	Limite	*	*	*
	Extérieur	*	*	*

Le prédicat ST_Intersects renvoie true si la limite de la première géométrie s'intersecte avec l'intérieur de la deuxième.

			Géométrie b	
		Ad-Dākhilīyah	**Limite**	**Extérieur**
Géométrie a	**Ad-Dākhilīyah**	*	*	*
	Limite	J	*	*
	Extérieur	*	*	*

Matrice ST_Intersects 3

Le prédicat ST_Intersects renvoie true si les limites des deux géométries s'intersectent.

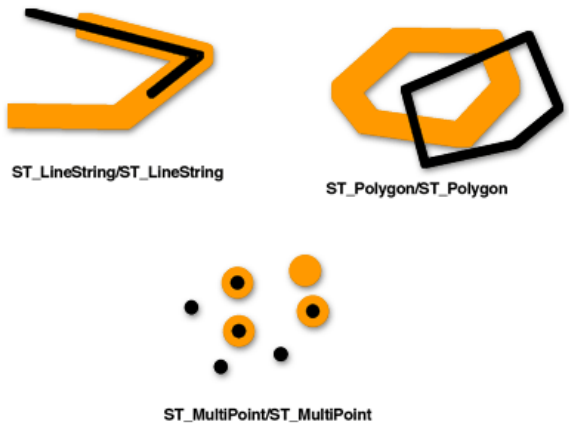
			Géométrie b	
		Ad-Dākhilīyah	**Limite**	**Extérieur**
Géométrie a	**Ad-Dākhilīyah**	*	*	*
	Limite	*	J	*
	Extérieur	*	*	*

Matrice ST_Intersects 4

ST_Overlaps

ST_Overlaps compare deux géométries de même dimension et renvoie 1 ou t (true) si leur ensemble d'intersections génère une géométrie différente des deux géométries en entrée mais est de même dimension.

ST_Overlaps renvoie 1 ou t (true) uniquement pour les géométries de même dimension et seulement si leur ensemble d'intersections génère une géométrie de même dimension. En d'autres termes, si l'intersection de deux objets ST_Polygon génère un objet ST_Polygon, la supersposition renvoie 1 ou t (true).



Cette matrice modèle s'applique aux superpositions ST_Polygon/ST_Polygon, ST_MultiPoint/ST_MultiPoint et ST_MultiPolygon/ST_MultiPolygon. Pour ces associations, le prédicat de superposition renvoie true si l'intérieur des

deux géométries s'intersecte avec l'intérieur et l'extérieur de l'autre.

			Géométrie b	
		Ad-Dākhilīyah	**Limite**	**Extérieur**
Géométrie a	**Ad-Dākhilīyah**	J	*	J
	Limite	*	*	*
	Extérieur	J	*	*

Matrice ST_Overlaps 1

La matrice modèle suivante s'applique aux superpositions ST_LineString/ST_LineString et ST_MultiLineString/ ST_MultiLineString. Dans ce cas, l'intersection des géométries doit générer une géométrie avec une dimension de 1 (un autre objet ST_LineString ou ST_MultiLineString). Si la dimension de l'intersection des intérieurs a généré la valeur 0 (un point), le prédicat ST_Overlaps renvoie false. Toutefois, le prédicat ST_Crosses aurait renvoyé true.

			Géométrie b	
		Ad-Dākhilīyah	**Limite**	**Extérieur**
Géométrie a	**Ad-Dākhilīyah**	1	*	J
	Limite	*	*	*
	Extérieur	J	*	*

Matrice ST_Overlaps 2

ST_Relate

ST_Relate renvoie la valeur 1 ou t (true) si la relation spatiale spécifiée par la matrice modèle est valide. La valeur 1 ou t (true) indique qu'il existe une sorte de relation spatiale entre les géométries.

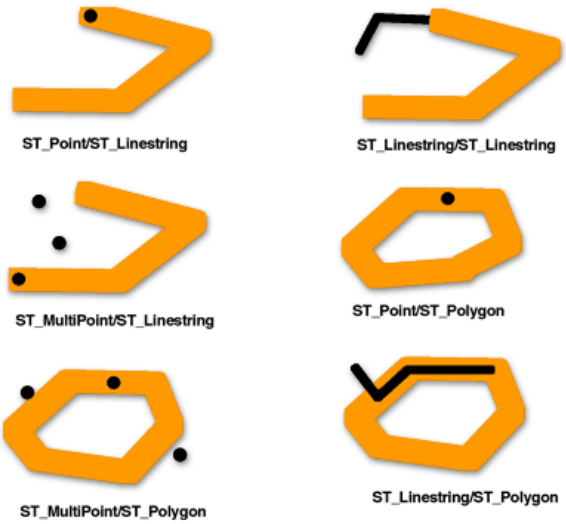
Si les intérieurs ou limites des géométries a et b sont reliées d'une quelconque façon, ST_Relate renvoie true. Il importe peu que les extérieurs d'une géométrie s'intersectent ou non avec l'intérieur ou la limite de l'autre.

			Géométrie b	
		Ad-Dākhilīyah	**Limite**	**Extérieur**
Géométrie a	**Ad-Dākhilīyah**	J	J	*
	Limite	J	J	*
	Extérieur	*	*	*

Matrice ST_Relate

ST_Touches

ST_Touches renvoie 1 ou t (true) si aucun des points communs aux deux géométries ne s'intersectent avec les intérieurs des deux géométries. Au moins une géométrie doit être un objet ST_LineString, ST_Polygon, ST_MultiLineString ou ST_MultiPolygon.



Les matrices modèles montrent que le prédicat ST_Touches renvoie true lorsque les intérieurs de la géométrie ne s'intersectent pas et que la limite de l'une de géométries s'intersecte avec l'intérieur ou la limite de l'autre.

Le prédicat ST_Touches renvoie true si la limite de la géométrie b s'intersecte avec l'intérieur de la géométrie a, mais que les intérieurs ne s'intersectent pas.

			Géométrie b	
		Ad-Dākhilīyah	**Limite**	**Extérieur**
Géométrie a	**Ad-Dākhilīyah**	F	J	*
	Limite	*	*	*
	Extérieur	*	*	*

Matrice ST_Touches 1

Le prédicat ST_Touches renvoie true si la limite de la géométrie a s'intersecte avec l'intérieur de la géométrie b, mais que les intérieurs ne s'intersectent pas.

			Géométrie b	
		Ad-Dākhilīyah	**Limite**	**Extérieur**
Géométrie a	**Ad-Dākhilīyah**	F	*	*
	Limite	J	*	*
	Extérieur	*	*	*

Matrice ST_Touches 2

Le prédicat ST_Touches renvoie true si les limites des deux géométries s'intersectent, mais pas les intérieurs.

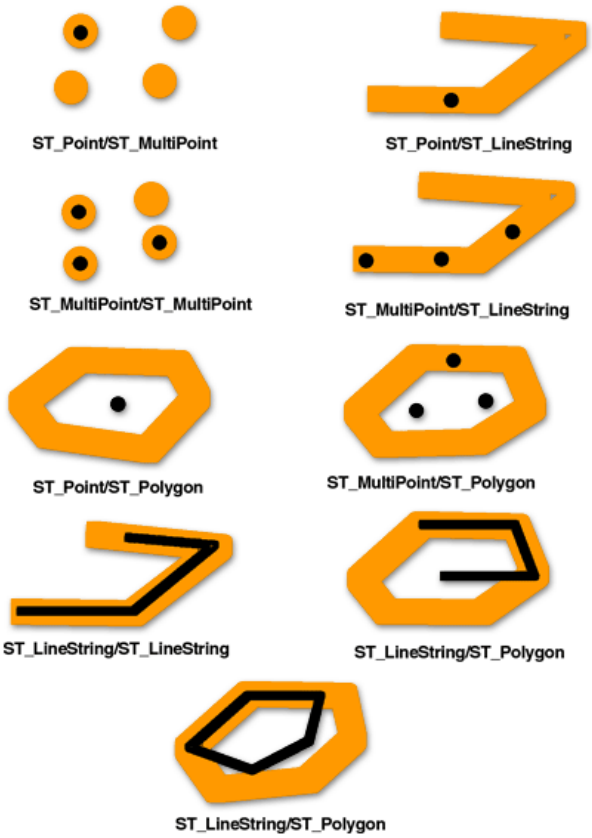
			Géométrie b	
		Ad-Dākhilīyah	**Limite**	**Extérieur**
Géométrie a	**Ad-Dākhilīyah**	F	*	*
	Limite	*	J	*

Matrice ST_Touches 3

	Extérieur	*	*	*

ST_Within

ST_Within renvoie 1 ou t (true) si la première géométrie se trouve entièrement à l'intérieur de la deuxième géométrie. ST_Within teste le résultat exactement contraire de ST_Contains.



La matrice modèle du prédicat ST_Within définit que les intérieurs des deux géométries doivent s'intersecter, et que l'intérieur et la limite de la géométrie principale (géométrie a) ne doivent pas s'intersecter avec l'extérieur de la géométrie secondaire (géométrie b).

			Géométrie b	
		Ad-Dākhilīyah	**Limite**	**Extérieur**
Géométrie a	**Ad-Dākhilīyah**	J	*	F
	Limite	*	*	F
	Extérieur	*	*	*

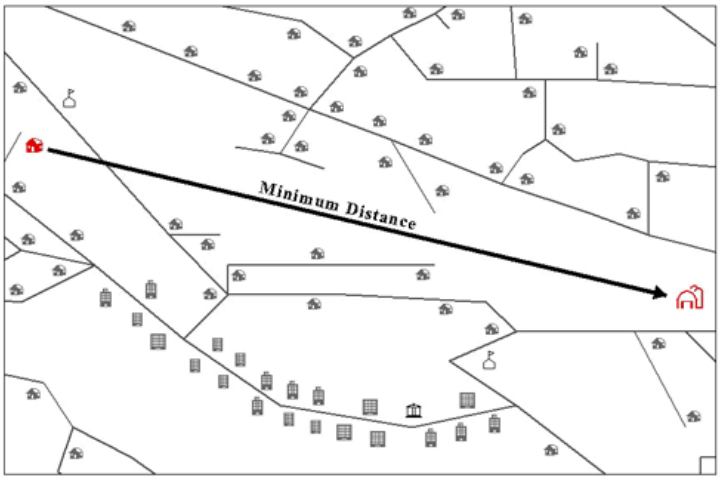
Matrice ST_Within

Autres relations spatiales

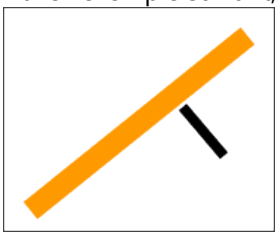
Les fonctions suivantes comparent la relation spatiale entre les géométries, mais comparent davantage que simplement l'intérieur, la limite et les extérieurs des géométries.

- ST_Distance - Cette fonction accepte deux géométries disjointes en entrée et renvoie la distance minimale entre

elles. Si les géométries ne sont pas disjointes (en d'autres termes, si elles sont coïncidentes), la fonction signale une distance minimale égale à zéro.
La distance minimale qui sépare des entités représente la distance la plus courte entre deux emplacements. Par exemple, il ne s'agit pas de la distance que vous devriez parcourir si vous alliez en voiture d'un emplacement à l'autre, mais de celle que vous calculeriez si vous traciez une ligne droite entre deux emplacements sur une carte.



- ST_DWithin - Vous indiquez une valeur de distance avec les géométries à comparer. ST_DWithin renvoie la valeur true si les géométries se trouvent à l'intérieur de la distance spécifiée par rapport à l'autre.
- ST_EnvIntersects - Cette fonction évalue si les enveloppes spatiales des géométries spécifiées s'intersectent, tandis que ST_Intersects évalue si les géométries elles-mêmes s'intersectent.
  Dans l'exemple suivant, les enveloppes des deux lignes s'intersectent mais pas les lignes elles-mêmes :



- ST_OrderingEquals - Cette fonction étend la comparaison effectuée par ST_Equals pour comparer également que les coordonnées des géométries sont définies dans le même ordre (x,y contre y,x). Même si les géométries occupent le même espace, si leurs coordonnées x et y ne sont pas définies dans le même ordre, ST_OrderingEquals renvoie la valeur false.

Opérations spatiales

Les opérations spatiales utilisent des fonctions géométriques pour extraire les données spatiales en tant qu'entrée, analysent les données, puis produisent les données en sortie qui dérivent de l'analyse effectuée sur les données en entrée.

Les données dérivées que vous pouvez obtenir d'une opération spatiale incluent les suivantes :

- Un polygone qui est une zone tampon autour d'une entité en entrée
- Une entité unique qui résulte de l'analyse effectuée sur un ensemble de géométries
- Une entité unique qui résulte d'une comparaison pour déterminer la partie d'une entité qui n'occupe pas le même espace physique qu'une autre entité
- Une entité unique qui résulte d'une comparaison pour rechercher les parties d'une entité qui intersectent l'espace physique d'une autre entité
- Une entité multi-parties composée des parties des deux entités en entrée qui n'occupent pas chacune le même espace physique
- Une entité qui constitue l'union de deux géométries

L'analyse effectuée sur les données en entrée renvoie les coordonnées ou la représentation textuelle des géométries résultantes. Vous pouvez utiliser ces informations dans le cadre d'une requête plus vaste pour effectuer une analyse approfondie ou utiliser les résultats en tant qu'entrée d'une autre table.

Par exemple, vous pourriez inclure une opération de zone tampon dans la clause `WHERE` d'une requête d'intersection pour déterminer si la géométrie spécifiée intersecte une zone d'une taille spécifiée autour d'une autre géométrie.

Remarque :
Les exemples suivants utilisent des fonctions ST_Geometry. Pour les fonctions géométriques spécifiques et la syntaxe utilisées pour un autre type de base de données et de données spatiales, consultez la documentation spécifique à ce type.

Dans cet exemple, des notifications doivent être envoyées à tous les propriétaires situés à moins de 1 000 pieds d'une rue fermée. La clause `WHERE` génère une zone tampon de 1 000 pieds autour de la rue allant être fermée. La zone tampon est alors comparée aux propriétés situées dans la zone pour voir lesquelles elle intersecte.

```
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';
```

Dans cet exemple, une rue spécifique (Main) est choisie dans la clause `WHERE`, puis une zone tampon est créée autour de la rue et comparée aux entités situées dans la table de parcelles pour déterminer si elles s'intersectent.* Pour toutes les parcelles que la zone tampon intersecte sur Main Street, le nom et l'adresse du propriétaire de la parcelle sont renvoyés.

Remarque :
*L'ordre dans lequel les parties de la clause `WHERE` sont exécutées dépend de l'optimiseur de la base de données.

L'exemple suivant illustre l'extraction des résultats d'une opération spatiale (union) effectuée sur des tables qui contiennent des zones de voisinage et de district scolaire, ainsi que l'insertion des entités résultantes dans une autre table :

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

Pour plus d'informations sur l'utilisation d'opérateurs spatiaux avec ST_Geometry, reportez-vous à la rubrique Fonctions d'opération spatiale pour ST_Geometry.

Fonctions d'opérations spatiales pour ST_Geometry

Les opérations spatiales utilisent des fonctions géométriques pour extraire les données spatiales en tant qu'entrée, analysent les données, puis produisent les données en sortie qui dérivent de l'analyse effectuée sur les données en entrée.

Vous pouvez effectuer les opérations décrites dans les sections suivantes pour créer des entités à partir d'entités en entrée.

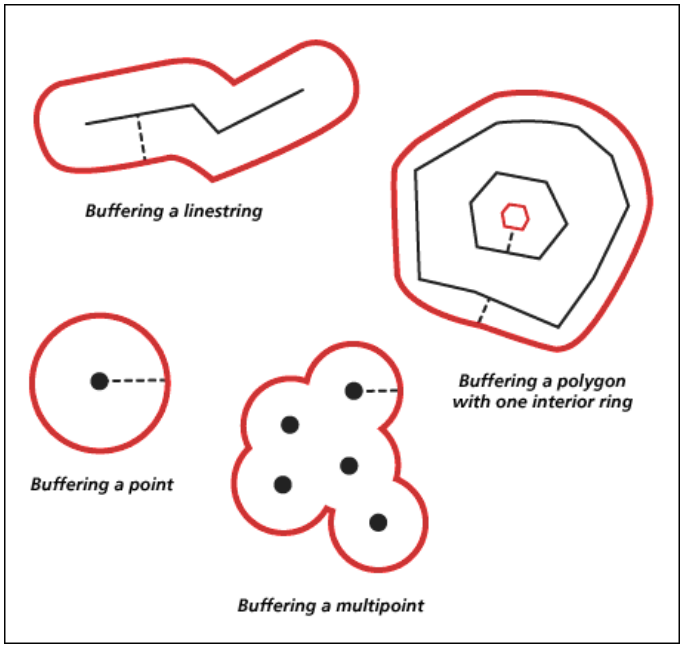
Création d'une zone tampon autour de la géométrie

La fonction ST_Buffer génère une géométrie en l'entourant de la distance que vous spécifiez. Un polygone unique est généré lorsqu'une géométrie principale est bufférisée ou lorsque les polygones de zone tampon d'une collection sont suffisamment proches pour se chevaucher. Lorsque les éléments d'une collection bufférisée sont suffisamment éloignés les uns des autres, les objets ST_Polygon de la zone tampon génèrent un objet ST_MultiPolygon.

La fonction ST_Buffer accepte des distances positives et négatives, mais vous pouvez appliquer des distances négatives uniquement à des géométries à deux dimensions (ST_Polygon et ST_MultiPolygon). ST_Buffer utilise la valeur absolue de la distance de la zone tampon lorsque la géométrie source comporte moins de deux dimensions. En d'autres termes, cela concerne toutes les géométries qui ne sont ni ST_Polygon, ni ST_MultiPolygon. Les distances positives de la zone tampon génèrent des anneaux surfaciques qui sont éloignés du centre de la géométrie source et, pour la boucle extérieure d'un objet ST_Polygon ou ST_MultiPolygon, proche du centre lorsque la distance est négative. Pour les boucles intérieures d'un objet ST_Polygon ou ST_MultiPolygon, l'anneau de zone tampon est proche du centre lorsque la distance de la zone tampon est positive et éloigné du centre lorsqu'elle est négative.

Le processus de création de zone tampon fusionne les polygones de zone tampon qui se chevauchent. Les distances négatives supérieures à la moitié de la largeur intérieure maximale d'un polygone génèrent une géométrie vide.

Dans le diagramme suivant, les zones tampon apparaissent en rouge.



ConvexHull

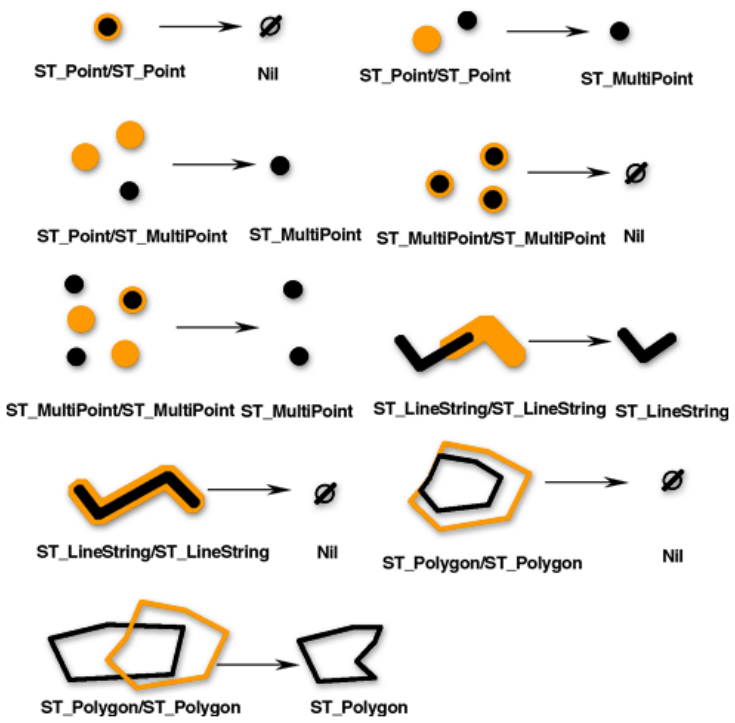
La fonction ST_ConvexHull renvoie le polygone d'enveloppe convexe d'une géométrie qui comporte au moins trois sommets formant un objet convexe. Si les sommets de la géométrie ne forment pas d'objet convexe, ST_ConvexHull renvoie une valeur nulle. Par exemple, l'utilisation de ST_ConvexHull sur une ligne composée de deux sommets renverra une valeur nulle. De même, l'utilisation de l'opération ST_ConvexHull sur une entité ponctuelle renverra une valeur nulle. La création d'une enveloppe convexe constitue souvent la première étape de l'assemblage d'un ensemble de points pour créer un réseau triangulé irrégulier (TIN).

Différence de géométries

La fonction ST_Difference renvoie la portion de la géométrie principale qui n'est pas intersectée par la géométrie secondaire. Il s'agit de l'opérateur logique `AND NOT` de l'espace.

La fonction ST_Difference fonctionne uniquement sur des géométries de dimension similaire et renvoie une collection qui a la même dimension que les géométries source. Si les géométries source sont égales, une géométrie vide est renvoyée.

Dans le diagramme ci-dessous, les premières géométries en entrée apparaissent en noir, tandis que les deuxièmes géométries en entrée s'affichent en orange.



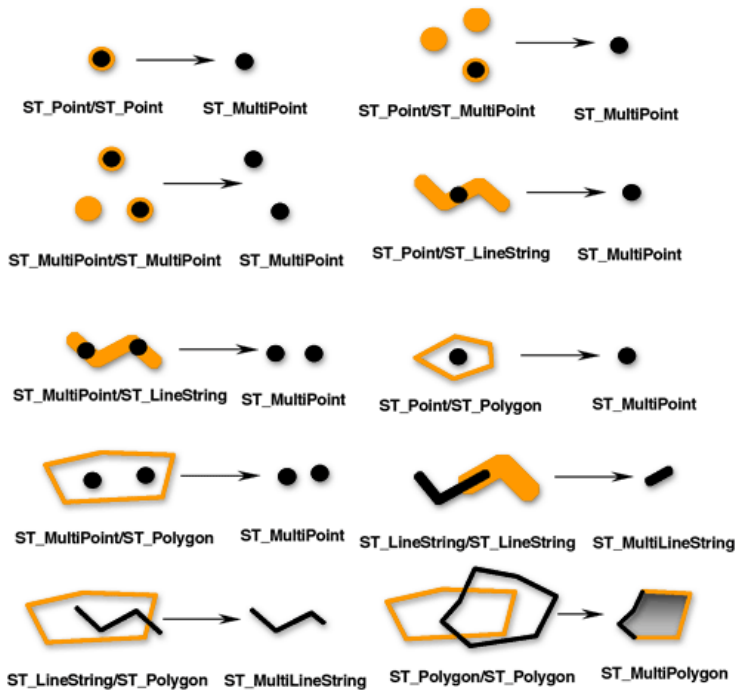
Intersection de géométries

La fonction ST_Intersection renvoie l'ensemble d'intersections de deux géométries. L'ensemble d'intersections est toujours renvoyé sous la forme d'une collection qui correspond à la dimension minimale des géométries source.

Par exemple, pour un objet ST_LineString qui s'intersecte avec un objet ST_Polygon, la fonction ST_Intersection renvoie cette portion de l'objet ST_LineString commune à l'intérieur et à la limite de l'objet ST_Polygon sous forme d'objet ST_MultiLineString. L'objet ST_MultiLineString contient plusieurs objets ST_LineString si l'objet ST_LineString

source a intersecté l'objet ST_Polygon avec au moins deux segments discontinus. Si les géométries ne s'intersectent pas ou si l'intersection génère une dimension inférieure aux deux géométries source, une géométrie vide est renvoyée.

La figure suivante présente des exemples de la fonction ST_Intersection. Les premières géométries en entrée apparaissent en noir, tandis que les deuxièmes géométries en entrée s'affichent en orange.

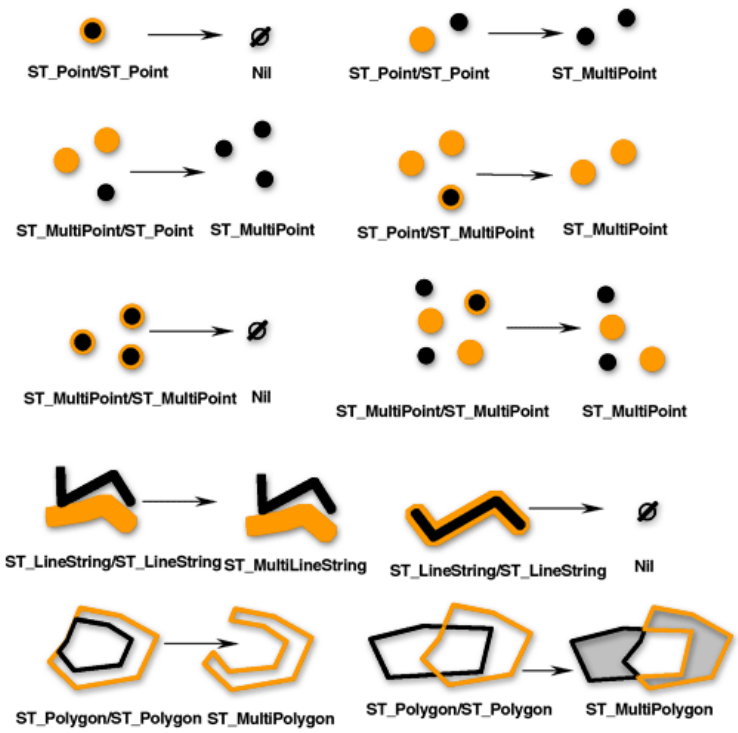


Différence symétrique de géométries

La fonction ST_SymmetricDiff renvoie les portions des géométries source qui ne font pas partie de l'ensemble d'intersections. Il s'agit de l'opérateur logique `XOR` de l'espace.

Les géométries source doivent avoir la même dimension. Si les géométries sont égales, la fonction ST_SymmetricDiff renvoie une géométrie vide ; sinon, la fonction renvoie le résultat sous forme de collection.

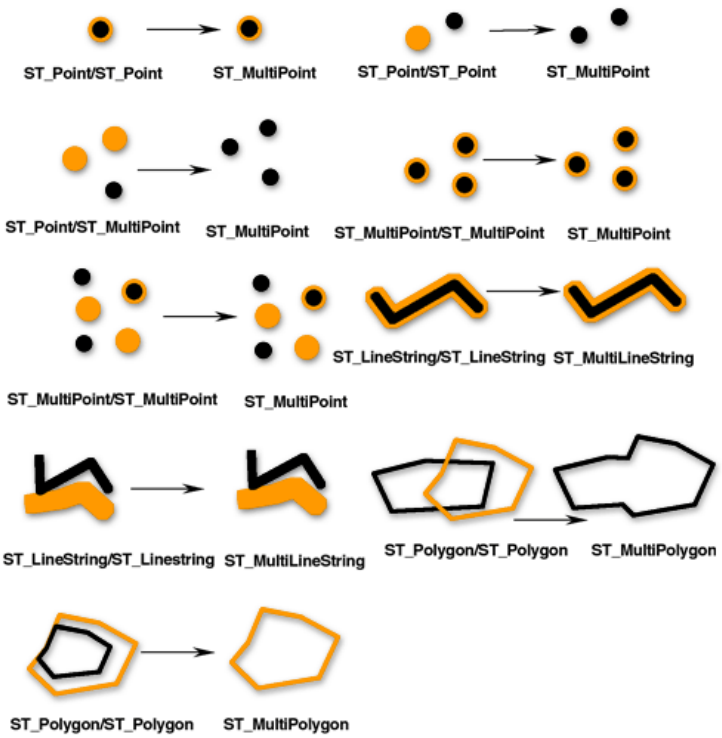
Dans le diagramme ci-dessous, les premières géométries en entrée apparaissent en noir, tandis que les deuxièmes géométries en entrée s'affichent en orange.



Union de géométries

La fonction ST_Union renvoie l'ensemble d'unions de deux géométries. Il s'agit de l'opérateur logique booléen `OR` de l'espace. Les géométries source doivent avoir la même dimension. ST_Union renvoie toujours le résultat sous forme de collection.

Dans le diagramme ci-dessous, les premières géométries en entrée apparaissent en noir, tandis que les deuxièmes géométries en entrée s'affichent en orange.



Agrégats

Les opérations d'agrégation renvoient une géométrie unique qui résulte de l'analyse effectuée sur une collection de géométries. La fonction ST_Aggr_ConvexHull renvoie le multipolygone composé des polygones d'enveloppe convexe de chacune des géométries en entrée. Une géométrie en entrée comportant moins de trois sommets n'aura pas d'enveloppe convexe. Si toutes les géométries en entrée comportent moins de trois sommets, ST_Aggr_ConvexHull renvoie une valeur nulle.

La fonction ST_Aggr_Intersection renvoie une géométrie unique qui correspond à une agrégation des intersections de toutes les géométries en entrée.

ST_Aggr_Intersection trouve l'intersection de plusieurs géométries, tandis que ST_Intersection trouve uniquement l'intersection entre deux géométries. Par exemple, si vous vouliez trouver la propriété couverte par différents services spécifiques, comme une carte scolaire, un service téléphonique et un fournisseur Internet haut débit donnés et représentée par un conseiller spécifique, vous devez trouver l'intersection de toutes ces zones. Étant donné que la recherche de l'intersection de seulement deux de ces zones ne renverrait pas toutes les informations nécessaires, vous devez utilisez la fonction ST_Aggr_Intersection afin que toutes les zones puissent être évaluées dans la même requête.

De même, lors de la recherche des intersections de lignes et de points dans deux classes d'entités, chaque fonction renverra les éléments suivants :

- ST_Intersection - Une géométrie ST_Point est renvoyée pour chaque intersection.
- ST_Aggr_Intersection - Une géométrie ST_MultiPoint composée de tous les points d'intersection est renvoyée. (Toutefois, si une seule entité ponctuelle et une seule entité linéaire s'intersectent, vous obtiendrez une géométrie ST_Point.)

La fonction ST_Aggr_Union renvoie une géométrie correspondant à l'union de toutes les géométries fournies.

Les géométries en entrée doivent être de même type ; par exemple, vous pouvez créer une union entre ST_LineStrings et ST_LineStrings ou entre ST_Polygons et ST_Polygons, mais pas entre une classe d'entités ST_LineString et une classe d'entités ST_Polygon.

La géométrie résultant de l'union agrégée est généralement une collection. Par exemple, si vous voulez créer une union agrégée de toutes les parcelles vacantes inférieures à un demi-acre, la géométrie renvoyée sera un multipolygone, sauf si toutes les parcelles répondant aux critères sont contiguës. Dans ce cas, un polygone sera renvoyé.

Distance minimale

Les fonctions précédentes ont renvoyé de nouvelles géométries. La fonction ST_Distance effectue une opération spatiale (elle évalue la distance minimale entre deux géométries), mais ne renvoie pas une nouvelle géométrie.

Cercles, ellipses et secteurs paramétriques

Vous pouvez créer et interroger des cercles, des ellipses ou des secteurs paramétriques dans les colonnes ST_Geometry à l'aide de la fonction ST_Geometry.

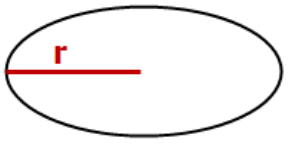
Les cercles, les ellipses et les secteurs paramétriques sont des polygones définis par des paramètres spécifiques, tels que les valeurs de coordonnées, les angles et les rayons. La base de données stocke ces paramètres au lieu de sommets et lignes spécifiques. En stockant les paramètres qui définissent la forme, les formes paramétriques peuvent être plus précises et utiliser moins d'espace que si vous les stockez sous forme de représentations d'entités surfaciques à plusieurs côtés. L'utilisation des formes paramétriques vous permet également d'inclure des paramètres de coordonnée z et de valeur de mesure (m).

Sept paramètres permettent généralement de créer un cercle :

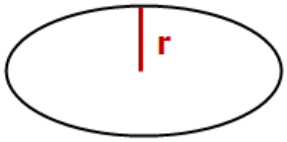
- Valeur de coordonnée x du centre du cercle
- Valeur de coordonnée y du centre du cercle
- Valeur de coordonnée z du centre du cercle
- Valeur m
- Rayon du cercle à créer
- Nombre de points utilisés pour définir le cercle
  Le nombre minimal de points que vous pouvez spécifier est 9. Si vous n'indiquez pas le nombre de points, 50 points sont utilisés par défaut. Ces points ne sont pas stockés avec la forme mais sont générés lors de la génération du cercle pour valider la forme.
- Identifiant de référence spatiale (SRID) utilisé pour insérer le cercle dans l'espace

Neuf paramètres permettent généralement de créer une ellipse :

- Valeur de coordonnée x du point central de l'ellipse
- Valeur de coordonnée y du point central de l'ellipse
- Valeur de coordonnée z du point central de l'ellipse
- Valeur m
- Demi-grand axe de l'ellipse
  Le demi-grand axe est le plus long rayon d'une ellipse. La valeur spécifiée pour le demi-grand axe doit être supérieure au demi-petit axe.



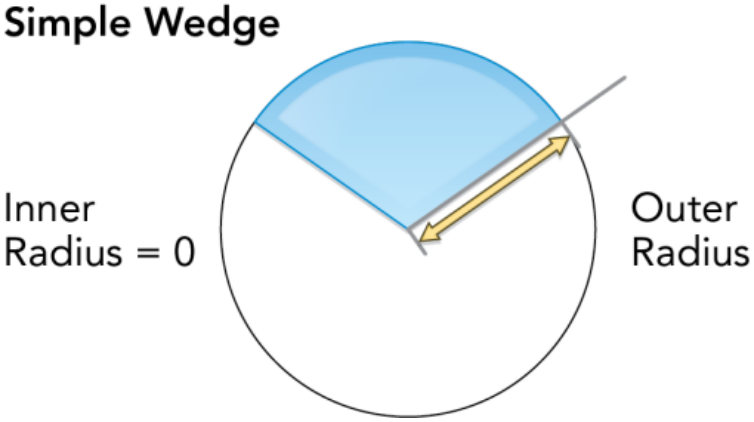
- Demi-petit axe de l'ellipse
  Le demi-petit axe est le plus court rayon d'une ellipse. La valeur spécifiée pour le demi-petit axe doit être supérieure à 0,0.



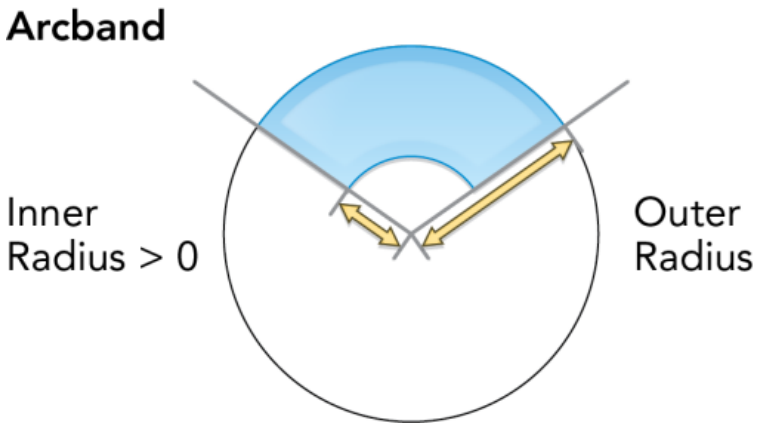
- Angle de rotation de l'ellipse
  La valeur spécifiée pour l'angle de rotation est exprimée en degrés et doit être supérieure à 0,0, mais inférieure à 360. La rotation suit le sens horaire.
- Nombre de points utilisés pour définir l'ellipse
  Le nombre minimal de points que vous pouvez spécifier est 9. Si vous n'indiquez pas le nombre de points, 50 points sont utilisés par défaut. Ces points ne sont pas stockés avec la forme mais sont générés lors de la génération de l'ellipse pour valider la forme.
- SRID utilisé pour insérer l'ellipse dans l'espace

Dix paramètres permettent généralement de créer un secteur :

- Valeur de coordonnée x du centre du cercle qui définit le secteur
- Valeur de coordonnée y du centre du cercle qui définit le secteur
- Valeur de coordonnée z du centre du cercle qui définit le secteur
- Valeur m
- Angle de départ du secteur
  L'angle de départ définit le début du secteur comme suit : degrés mesurés dans le sens anti-horaire à partir de 0°.
- Angle d'arrivée du secteur
  L'angle d'arrivée définit la fin du secteur comme suit : degrés mesurés dans le sens anti-horaire à partir de 0°.
- Rayon extérieur
  Le rayon extérieur définit la distance du centre du cercle au point le plus à l'extérieur du secteur.
- Rayon intérieur
  Le rayon intérieur détermine la distance du centre du cercle au point le plus intérieur du secteur, définissant ainsi le début de ce dernier. Si le rayon intérieur est nul, la forme est un secteur simple.



Si le rayon intérieur est supérieur à 0, le secteur est techniquement une bande d'arc.



- Nombre de points utilisés pour définir le secteur
  Le nombre minimal de points que vous pouvez spécifier est 9. Si vous n'indiquez pas le nombre de points, 80 points sont utilisés par défaut. Ces points ne sont pas stockés avec la forme mais sont générés lors de la génération du secteur pour valider la forme.
- SRID utilisé pour insérer le secteur dans l'espace

Tous les rayons, y compris les demi-grand et demi-petit axes, ainsi que les rayons intérieur et extérieur, sont définis dans les unités déterminées par le référentiel de coordonnées spécifié avec le SRID.

Reportez-vous à la fonction ST_Geometry pour découvrir la syntaxe et des exemples de création de cercles, ellipses ou secteurs paramétriques.

ST_Aggr_ConvexHull

Remarque :
Oracle et SQLite uniquement

Définition

La fonction ST_Aggr_ConvexHull crée une géométrie unique qui constitue une enveloppe convexe d'une géométrie résultant d'une union de toutes les géométries en entrée. En effet, ST_Aggr_ConvexHull équivaut à ST_ConvexHull(ST_Aggr_Union(geometries)).

Syntaxe

Oracle

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

SQLite

```
st_aggr_convexhull (geometry st_geometry)
```

Type de retour

Oracle

ST_Geometry

SQLite

Geometryblob

Exemple

L'exemple crée une table service_territories et exécute une instruction `SELECT` qui ajoute toutes les géométries et ainsi génère une géométrie unique représentant l'enveloppe convexe de l'union de toutes les formes.

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

Remarque :
Oracle et SQLite uniquement

Définition

ST_Aggr_Intersection renvoie une géométrie unique correspondant à l'union de l'intersection de toutes les géométries en entrée.

Syntaxe

Oracle

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

SQLite

```
st_aggr_intersection (geometry1 geometryblob)
```

Type de retour

Oracle

ST_Geometry

SQLite

Geometryblob

Exemple

Dans cet exemple, un biologiste essaie de trouver l'intersection de trois habitats naturels.

Oracle

En premier lieu, créez la table qui stocke les habitats.

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

Ensuite, insérez les trois polygones à la table.

```
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)
);
```

Enfin, sélectionnez l'intersection des habitats.

```
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

En premier lieu, créez la table qui stocke les habitats.

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

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

Ensuite, insérez les trois polygones à la table.

```
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)
);
```

Enfin, sélectionnez l'intersection des habitats.

```
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

Définition

ST_Aggr_Union renvoie une géométrie unique correspondant à l'union de toutes les géométries en entrée.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_aggr_union(geometry geometryblob)
```

Type de retour

Oracle et PostgreSQL

ST_Geometry

SQLite

Geometryblob

Exemple

Un analyste marketing doit créer une géométrie unique regroupant toutes les zones de desserte pour lesquelles les ventes ont dépassées 1 000 unités. Pour cet exemple, vous allez créer une table service_territories1 et la renseigner avec des valeurs de ventes. Vous utiliserez ensuite st_aggr_union dans une instruction SELECT pour renvoyer le multipolygone correspondant à l'union de toutes les géométries pour lesquelles les ventes étaient égales ou supérieures à 1 000 unités.

Oracle et 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

Définition

ST_Area retourne la surface d'un objet polygon ou multipolygon.

Syntaxe

Oracle et 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)
```

Type de retour

Double précision

Exemple

L'ingénieur municipal a besoin d'une liste des zones à bâtir. Pour créer cette liste, un technicien SIG sélectionne l'identifiant de bâtiment et la surface de chaque emprise de bâtiment.

Les emprises de bâtiments sont stockées dans la table bfp.

Pour répondre à la demande de l'ingénieur municipal, le technicien sélectionne la clé unique, l'identifiant building_id et la surface de chaque emprise de bâtiment de la table bfp.

Oracle

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

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

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

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

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

BUILDING_ID       Area

          1        100
          2        200
          3         25
```

PostgreSQL

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

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

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

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

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

building_id       area
```

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

SQLite

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

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

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

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

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

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

building_id       area

          1        100.0
          2        200.0
          3         25.0
```

ST_AsBinary

Définition

ST_AsBinary accepte un objet géométrie et retourne sa représentation binaire connue.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_asbinary (geometry geometryblob)
```

Type de retour

Oracle et PostgreSQL

ST_Geometry

SQLite

Geometryblob

Exemple

Cet exemple renseigne la colonne WKB de l'enregistrement 1111 avec le contenu issu de la colonne GEOMETRY de l'enregistrement 1100.

Oracle

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

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

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

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

ID              Point
```

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

PostgreSQL

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

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

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

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

ID              st_astext
1111     POINT (10 20)
```

SQLite

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

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

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

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

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

ID              st_astext
2        POINT (10.00000000 20.00000000)
```

ST_AsText

Définition

ST_AsText accepte une géométrie et retourne sa représentation textuelle connue.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_astext (geometry geometryblob)
```

Type de retour

Oracle

CLOB

PostgreSQL et SQLite

Texte

Exemple

La fonction ST_AsText convertit le point de localisation hazardous_sites en description textuelle.

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

Définition

ST_Boundary prend une géométrie et renvoie sa limite combinée sous forme d'objet géométrie.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_boundary (geometry geometryblob)
```

Type de retour

Oracle et PostgreSQL

ST_Geometry

SQLite

Geometryblob

Exemple

Dans cet exemple, la table des limites est créée avec deux colonnes : type et geometry. Les instructions INSERT suivantes ajoutent un enregistrement pour chacune des géométries de sous-classe. La fonction ST_Boundary extrait la limite de chaque sous-classe stockée dans la colonne geometry. Notez que la dimension de la géométrie générée est toujours inférieure de un à la géométrie d'entrée. Les points et les multi-points génèrent toujours une limite qui est une géométrie vide, de dimension –1. Les objets de chaîne de lignes et de chaîne multi-ligne renvoient une limite multi-point, de dimension 0. Un polygone ou un multipolygone renvoie toujours une limite de chaîne multi-ligne, de dimension 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

Définition

ST_Buffer accepte un objet géométrie et une distance et retourne un objet géométrie qui est la zone tampon entourant l'objet source.

Syntaxe

`Unit_name` est l'unité de mesure de la distance de la zone tampon (par exemple, `meters (mètres)`, `kilometers (kilomètres)`, `feet (pieds)` ou `mile`). Reportez-vous à la première table dans le `Projected coordinate system tables.pdf`, accessible via Systèmes de coordonnées, projections et transformations.

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)
```

Type de retour

Oracle et PostgreSQL

ST_Geometry

SQLite

Geometryblob

Exemple

Cet exemple crée deux tables, sensitive_areas et hazardous_sites, remplit ces tables, utilise ST_Buffer pour générer une zone tampon autour des polygones de la table hazardous_sites et trouve les zones de chevauchement entre ces zones tampons et les polygones sensitive_areas.

Oracle

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

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

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

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

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

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

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

PostgreSQL

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

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

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

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

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

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

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

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

SQLite

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

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

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

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

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

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

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

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

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

Sensitive Areas               Hazardous Sites
```

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

ST_Centroid

Définition

La fonction ST_Centroid prend une entité surfacique, multisurfacique ou multichaîne de lignes et renvoie le point se trouvant au centre de l'enveloppe de la géométrie. Cela signifie que le centroïde se trouve à mi-chemin entre les étendues x et y minimales et maximales de la géométrie.

Syntaxe

Oracle et 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)
```

Type de retour

ST_Point

Exemple

Le technicien SIG municipal souhaite afficher les entités multisurfaciques des emprises de bâtiments en tant que points isolés dans une illustration de la densité de construction. Les emprises de bâtiments sont stockées dans la table bfp qui a été créée et renseignée à l'aide des instructions affichées pour chaque base de données :

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

Définition

ST_Contains accepte deux objets géométrie et renvoie 1 (Oracle et SQLite) ou t (PostgreSQL) si le premier objet contient complètement le second ; dans le cas contraire, la fonction renvoie 0 (Oracle et SQLite) ou f (PostgreSQL).

Syntaxe

Oracle et PostgreSQL

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

SQLite

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

Type de retour

Booléen

Exemple

Dans les exemples ci-dessous, deux tables sont créées : la première, buildingfootprints, contient les emprises des bâtiments d'une municipalité, tandis que l'autre, lots, contient les parcelles. L'ingénieur municipal souhaite s'assurer que toutes les emprises des bâtiments se trouvent entièrement à l'intérieur de leurs parcelles.

L'ingénieur municipal utilise ST_Intersects et ST_Contains pour sélectionner les bâtiments qui ne se trouvent pas entièrement dans un terrain.

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

Définition

La fonction ST_ConvexHull renvoie l'enveloppe convexe d'un objet ST_Geometry.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_convexhull (geometry1 geometryblob)
```

Type de retour

Oracle et PostgreSQL

ST_Geometry

SQLite

Geometryblob

Exemple

Ces exemples créent la table sample_geometries avec trois colonnes : id, spatial_type et geometry. Le champ spatial_type stocke le type de géométrie créée dans la colonne geometry. Trois entités (une chaîne de ligne, un polygone et un multi-points) sont insérées dans la table.

L'instruction `SELECT` qui inclut la fonction ST_ConvexHull renvoie l'enveloppe convexe de chaque géométrie.

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

Définition

ST_CoordDim renvoie les dimensions des valeurs de coordonnées d'une colonne de géométrie.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_coorddim (geometry1 geometryblob)
```

Type de retour

Entier

2 = coordonnées x,y

3 = coordonnées x,y,z ou x,y,m

4 = coordonnées x,y,z,m

Exemple

Dans ces exemples, la table coorddim_test est créée avec les colonnes geotype et g1. La colonne geotype stocke le nom de la sous-classe de géométrie et de la dimension dans la colonne de géométrie g1.

L'instruction SELECT répertorie le nom de la sous-classe stocké dans la colonne geotype avec la dimension des coordonnées de cette géométrie.

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

Définition

ST_Crosses accepte deux objets ST_Geometry et renvoie 1 (Oracle et SQLite) ou t (PostgreSQL) si leur intersection résulte en un objet geometry dont la dimension est inférieure d'une unité à la dimension maximale des objets source. L'objet d'intersection doit contenir des points intérieurs aux deux géométries source et qui ne sont pas égaux à l'un des objets source. Dans le cas contraire, la fonction retourne 0 (Oracle et SQLite) ou f (PostgreSQL).

Syntaxe

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

Oracle et PostgreSQL

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

SQLite

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

Type de retour

Booléen

Exemple

Le conseil municipal envisage un nouveau règlement énonçant que les dépôts de déchets dangereux dans la municipalité ne doivent pas se trouver à une distance spécifique des cours d'eau. Le responsable SIG municipal dispose d'une représentation exacte des fleuves et des rivières stockée sous forme d'objets linestring dans la table des cours d'eau, mais il ne dispose que d'une localisation ponctuelle pour chacun des dépôts de déchets dangereux.

Pour déterminer s'il doit signaler au responsable de l'aménagement du territoire l'existence de dépôts enfreignant le règlement envisagé, le responsable SIG doit créer des zones tampon autour des localisations hazardous_sites pour déterminer si des rivières ou ruisseaux recoupent ces zones tampon. Le prédicat cross compare les points bufférisés hazardous_sites avec les cours d'eau et ne renvoie que les enregistrements où le cours d'eau recoupe le rayon réglementaire proposé par la municipalité.

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

Remarque :
Oracle et SQLite uniquement

Définition

La fonction ST_Curve crée une entité de courbe à partir d'une représentation textuelle connue.

Syntaxe

Oracle

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

SQLite

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

Type de retour

ST_LineString

Exemple

Cet exemple crée une table avec une géométrie de courbe, y insère des valeurs et sélectionne une entité dans la table

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

Définition

La fonction ST_Difference part de deux objets géométrie et renvoie un objet géométrie qui est la différence des objets source.

Syntaxe

Oracle et PostgreSQL

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

SQLite

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

Type de retour

Oracle et PostgreSQL

ST_Geometry

SQLite

Geometryblob

Exemple

Dans les exemples suivants, l'ingénieur municipal veut connaître la surface totale de terrain à bâtir de la municipalité non couverte par des bâtiments. Elle doit par conséquent calculer la somme de la surface de terrain après soustraction de la surface des bâtiments.

L'ingénieur municipal joint de façon égale la table des emprises et des terrains sur l'identifiant lot_id et calcule la somme des surfaces correspondant aux terrains moins les emprises.

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

Définition

ST_Dimension renvoie la dimension d'un objet géométrie. Dans ce cas, la dimension se réfère à la longueur et la largeur. Par exemple, si un point n'a ni longueur ni largeur, sa dimension est 0 ; en revanche, comme une ligne comporte une longueur mais aucune largeur, sa dimension est 1.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_dimension (geometry1 geometryblob)
```

Type de retour

Entier

Exemple

La table dimension_test est créée avec les colonnes geotype et g1. La colonne geotype stocke le nom de la sous-classe stockée dans la colonne de géométrie g1.

L'instruction SELECT répertorie le nom de sous-classe stocké dans la colonne geotype avec la dimension de ce geotype.

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

Définition

ST_Disjoint accepte deux géométries et renvoie 1 (Oracle et SQLite) ou t (PostgreSQL) si l'intersection de deux géométries est un ensemble vide. Dans le cas contraire, la fonction renvoie 0 (Oracle et SQLite) ou f (PostgreSQL).

Syntaxe

Oracle et PostgreSQL

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

SQLite

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

Type de retour

Booléen

Exemple

Dans cet exemple, deux tables sont créées (distribution_areas et factories) et des valeurs sont insérées dans chacune d'elles. Ensuite, une zone tampon autour des usines et de st_disjoint est utilisée pour identifier les zones tampon des usines qui ne recoupent pas de zones de répartition.

Conseil :

Dans cette requête, vous pouvez également utiliser la fonction ST_Intersects en égalant le résultat de la fonction à 0, puisque ST_Intersects et ST_Disjoint renvoient des résultats opposés. La fonction ST_Intersects utilise l'index spatial lors de l'évaluation de la requête, ce qui n'est pas le cas de la fonction ST_Disjoint.

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

Définition

ST_Distance retourne la plus petite distance séparant deux géométries. La distance est mesurée entre les sommets les plus proches des deux géométries.

Syntaxe

Oracle et 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)
```

Les unités valides sont les suivantes :

Millimètre	Pouce	Yard	Lier
Centimètre	Inch_US	Yard_US	Link_US
Décimètre	Pied	Yard_Clarke	Link_Clarke
Mètres	Foot_US	Yard_Sears	Link_Sears
Meter_German	Foot_Clarke	Yard_Sears_1922_Truncated	Link_Sears_1922_Truncated
Kilomètre	Foot_Sears	Yard_Benoit_1895_A	Link_Benoit_1895_B
50_Kilometers	Foot_Sears_1922_Truncated	Yard_Indian	Chaîne
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	Fathom	Chain_Sears_1922_Truncated
	Foot_Indian_1962	Mile_US	Chain_Benoit_1895_A
	Foot_Indian_1975	Statute_Mile	Perche
	Foot_Gold_Coast	Nautical_Mile	Rod_US
	Foot_British_1936	Nautical_Mile_US	
		Nautical_Mile_UK	

Type de retour

Double précision

Exemple

Deux tables (study1 et zones) sont créées et renseignées. La fonction ST_Distance est ensuite utilisée pour déterminer la distance entre la limite de chaque sous-zone et les polygones dans la table des zones study1 dont le code d'utilisation est 400. Comme trois zones correspondent à cette forme, trois enregistrements doivent être renvoyés.

Si vous ne spécifiez pas d'unités, ST_Distance utilise les unités du système de projection des données. Dans le premier exemple, il s'agit des degrés décimaux. Dans les deux derniers exemples, comme kilomètre est spécifié, la distance est renvoyée en kilomètres.

Oracle et 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

Définition

ST_DWithin accepte deux géométries en entrée et renvoie la valeur true si les géométries sont éloignées de la distance spécifiée. Sinon, la valeur false est renvoyée. Le système de référence spatiale des géométries détermine l'unité de mesure appliquée à la distance spécifiée. Ainsi, les géométries fournies à ST_DWithin doivent toutes deux utiliser les mêmes projection de coordonnées et ID de référence spatiale (SRID).

Syntaxe

Oracle et PostgreSQL

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

SQLite

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

Type de retour

Booléen

Exemples

Dans les exemples suivants, deux tables sont créées et des entités y sont insérées. Ensuite, la fonction ST_DWithin est utilisée dans deux instructions SELECT différentes : l'une pour déterminer si un point de la première table se situe à 100 mètres maximum d'un polygone dans la seconde table, et l'autre pour déterminer quelles entités se trouvent à 300 mètres maximum l'une de l'autre.

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)
 )
;
```

Ensuite, utilisez ST_DWithin pour identifier les entités de chaque table qui sont à 100 mètres maximum les unes des autres et celles qui ne le sont pas. La fonction ST_Distance est incluse dans cette instruction pour indiquer la distance réelle entre les entités.

```
--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;
```

L'instruction renvoie les données suivantes :

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

Dans l'exemple suivant, ST_DWithin est utilisée pour identifier les entités qui se trouvent à 300 mètres maximum les unes des autres :

```
--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;
```

Cette seconde instruction Select renvoie les données suivantes lorsqu'elle est exécutée sur des données dans Oracle :

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

PostgreSQL

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

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

--Insert features into each table.

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

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

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

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

Ensuite, utilisez ST_DWithin pour identifier les entités de chaque table qui sont à 100 mètres maximum les unes des autres et celles qui ne le sont pas. La fonction ST_Distance est incluse dans cette instruction pour indiquer la distance réelle entre les entités.

```
--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;
```

L'instruction renvoie les données suivantes :

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

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

Dans l'exemple suivant, ST_DWithin est utilisée pour identifier les entités qui se trouvent à 300 mètres maximum les unes des autres :

```
--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;
```

Cette seconde instruction select renvoie les données suivantes :

```
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)
 )
;
```

Ensuite, utilisez ST_DWithin pour identifier les entités de chaque table qui sont à 100 mètres maximum les unes des autres et celles qui ne le sont pas. La fonction ST_Distance est incluse dans cette instruction pour indiquer la distance réelle entre les entités.

```
--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;
```

L'instruction renvoie les données suivantes :

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

Dans l'exemple suivant, ST_DWithin est utilisée pour identifier les entités qui se trouvent à 300 mètres maximum les unes des autres :

```
--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;
```

Cette seconde instruction select renvoie les données suivantes :

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

ST_EndPoint

Définition

ST_EndPoint retourne le dernier point d'un objet linestring.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_endpoint (line1 geometryblob)
```

Type de retour

ST_Point

Exemple

La table endpoint_test stocke la colonne gid integer, qui identifie chaque enregistrement de façon unique, et la colonne ln1 ST_LineString, qui stocke les objets linestring.

Les instructions INSERT suivantes insèrent des chaînes de lignes dans la table endpoint_test. Contrairement à la deuxième, la première chaîne de lignes n'a ni coordonnées z ni mesures.

Cette requête liste la colonne gid et la géométrie ST_Point générées par la fonction ST_EndPoint.

Oracle

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

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

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

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

```
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

Définition

La fonction ST_Entity renvoie le type d'entité spatiale d'un objet géométrie. Le type d'entité spatiale est la valeur stockée dans le champ du membre de l'entité de l'objet géométrie.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_entity (geometry1 geometryblob)
```

Type de retour

Renvoie un nombre (Oracle) ou un entier (SQLite et PostgreSQL) représentant les types d'entités suivants :

0	Forme nil
1	point
2	Ligne (y compris les lignes spaghetti)
4	linestring
8	aire
257	multi-points
258	Multiligne (y compris les lignes spaghetti)
260	multilinestring
264	Multisurface

Exemple

Les exemples suivants permettent de créer une table et d'y insérer différentes géométries. ST_Entity est exécuté sur la table pour renvoyer le sous-type de géométrie de chaque enregistrement dans la table.

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;
```

L'instruction SELECT renvoie les valeurs suivantes :

```
 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;
```

L'instruction SELECT renvoie les valeurs suivantes :

```
 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;
```

L'instruction SELECT renvoie les valeurs suivantes :

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

ST_Envelope

Définition

ST_Envelope renvoie l'emprise minimale d'un objet géométrie sous forme de polygone.

Approfondissement :
Cette fonction est conforme à la spécification Simple Features (Entités simples) Open Geospatial Consortium (OGC), qui établit que ST_Envelope renvoie un polygone. Pour utiliser des cas spéciaux de géométries ponctuelles ou de lignes horizontales ou verticales, la fonction ST_Envelope renvoie un polygone situé autour de ces formes, qui est une petite tolérance d'enveloppe calculée en fonction du facteur d'échelle XY pour le système de référence spatiale de la géométrie. Cette tolérance est soustraite des valeurs x et y minimales, puis est ajoutée aux coordonnées x et y maximales pour renvoyer le polygone situé autour de ces formes.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_envelope (geometry1 geometryblob)
```

Type de retour

Oracle et PostgreSQL

ST_Geometry

SQLite

Geometryblob

Exemple

La colonne geotype de la table envelope_test stocke le nom de la sous-classe de géométrie stockée dans la colonne g1. Les instructions INSERT insèrent chaque sous-classe de géométrie dans la table envelope_test.

Ensuite, la fonction ST_Envelope est exécutée pour renvoyer l'enveloppe du polygone situé autour de chaque géométrie.

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

Remarque :
Oracle et SQLite uniquement

Définition

ST_EnvIntersects renvoie 1 (true) si les enveloppes de deux géométries s'intersectent. Sinon, la fonction renvoie 0 (false).

Syntaxe

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)
```

Type de retour

Booléen

Exemple

Cet exemple recherche une géométrie dont l'enveloppe est intersectée par le polygone défini.

La première instruction SELECT compare les enveloppes de deux géométries et les géométries proprement dites afin de déterminer si les entités ou les enveloppes s'intersectent.

La deuxième instruction SELECT utilise une enveloppe afin de détecter les entités, le cas échéant, qui font partie de l'enveloppe transmise avec la clause WHERE de l'instruction SELECT.

Oracle

```
--Define and populate the table.
CREATE TABLE sample_geoms (
 id integer,
 geometry sde.st_geometry);

INSERT INTO SAMPLE_GEOMS (id, geometry) VALUES (
 1,
 sde.st_geometry ('linestring (10 10, 50 50)', 4326)
);

INSERT INTO SAMPLE_GEOMS (id, geometry) VALUES (
 2,
 sde.st_geometry ('linestring (10 20, 50 60)', 4326)
```

```
);
```

```
--Find the intersection of the geometries and the geometries' envelopes.
SELECT a.id, b.id, sde.st_intersects (a.geometry, b.geometry) Intersects,
sde.st_envintersects (a.geometry, b.geometry) Envelope_Intersects
 FROM SAMPLE_GEOMS a, SAMPLE_GEOMS b
 WHERE a.id = 1 AND b.id=2;

ID      ID      INTERSECTS   ENVELOPE_INTERSECTS

1       2       0            1
```

```
--Find the geometries whose envelopes intersect the specified envelope.
SELECT id
 FROM SAMPLE_GEOMS
 WHERE sde.st_envintersects(geometry, 5, 5, 60, 65) = 1;

ID

1
2
```

SQLite

```
--Define and populate the table.
CREATE TABLE sample_geoms (
 id integer primary key autoincrement not null
);

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

INSERT INTO SAMPLE_GEOMS (geometry) VALUES (
 st_geometry ('linestring (10 10, 50 50)', 4326)
);

INSERT INTO SAMPLE_GEOMS (geometry) VALUES (
 st_geometry ('linestring (10 20, 50 60)', 4326)
);
```

```
--Find the intersection of the geometries and the geometries' envelopes.
SELECT a.id AS aid, b.id AS bid, st_intersects (a.geometry, b.geometry) AS "Intersects",
 st_envintersects (a.geometry, b.geometry) AS "Envelope_Intersects"
 FROM SAMPLE_GEOMS a, SAMPLE_GEOMS b
 WHERE a.id = 1 AND b.id = 2;

aid     bid     Intersects   Envelope_Intersects
```

```
1       2       0            1
```

```
--Find the geometries whose envelopes intersect the specified envelope.
SELECT id
 FROM SAMPLE_GEOMS
 WHERE st_envintersects(geometry, 5, 5, 60, 65) = 1;

ID

1
2
```

ST_Equals

Définition

ST_Equals compare deux géométries et retourne 1 (Oracle et SQLite) ou t (PostgreSQL) si les géométries sont identiques ; dans le cas contraire, la fonction retourne 0 (Oracle et SQLite) ou f (PostgreSQL).

Syntaxe

Oracle et PostgreSQL

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

SQLite

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

Type de retour

booléen

Exemple

Le technicien SIG pense que certaines données de la table studies ont été dupliquées. Afin de lever cette incertitude, il interroge la table pour déterminer si certains multipolygones de forme sont égaux.

La table studies a été créée et remplie à l'aide des instructions ci-dessous. La colonne id identifie les zones d'étude de façon unique et le champ shape stocke la géométrie de la zone.

Ensuite, la table studies est spatialement jointe à elle-même par le prédicat equal, qui renvoie 1 (Oracle et SQLite) ou t (PostgreSQL) dès que deux multipolygones sont égaux. La condition s1.id<>s2.id écarte la comparaison d'une géométrie à elle-même.

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

Remarque :
PostgreSQL uniquement

Définition

La fonction ST_Equalsrs détermine si deux systèmes de référence spatiale de deux classes d'entités différentes sont identiques. Si les systèmes de référence spatiale sont identiques, t (true) est renvoyé. Si les systèmes de référence spatiale ne sont pas identiques, ST_Equalsrs renvoie f (false).

Syntaxe

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

Type de retour

Booléen

Exemple

Dans cet exemple, les identifiants de référence spatiale (SRID) de classes d'entités différentes sont découverts, puis la fonction ST_Equalsrs est utilisée pour savoir si les SRID représentent le même système de référence spatiale.

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

srid	table_name
2	rues
6	transmains

Résultats de la requête sde_layers

Utilisez maintenant la fonction ST_Equalsrs pour déterminer si les systèmes de référence spatiale identifiés par ces deux SRID sont les mêmes.

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

 st_equalsrs

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

f

(1 row)
```

ST_ExteriorRing

Définition

ST_ExteriorRing retourne la boucle extérieure d'un polygone sous la forme d'un objet linestring.

Syntaxe

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

Oracle et PostgreSQL

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

SQLite

```
st_exteriorring (polygon1 geometryblob)
```

Type de retour

ST_LineString

Exemple

Une ornithologue, souhaitant étudier la population d'oiseaux de plusieurs îlots, sait que la zone d'alimentation des espèces d'oiseaux qui l'intéressent est limitée au rivage. Dans le cadre du calcul de la capacité d'accueil de l'îlot, l'ornithologue doit connaître les périmètres des îlots. Certains îlots sont si grands qu'ils comportent plusieurs lacs. Toutefois, le rivage des lacs est habité exclusivement par une autre espèce d'oiseaux plus agressifs. Par conséquent, l'ornithologue n'a besoin que du périmètre de la boucle extérieure des îlots.

Les colonnes ID et name de la table islands identifient chaque île, tandis que la colonne de polygones land stocke la géométrie des îles.

La fonction ST_ExteriorRing extrait la boucle extérieure de chaque polygone d'îlot sous forme d'objet linestring. La longueur de l'objet linestring est calculée par la fonction ST_Length. Les longueurs des objets linestring sont totalisées par la fonction SUM.

Les boucles extérieures des îlots représentent l'interface écologique maritime de chaque îlot.

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

Remarque :
Oracle et PostgreSQL uniquement

Définition

La fonction ST_GeomCollection crée un ensemble de géométries à partir d'une représentation textuelle connue.

Syntaxe

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)
```

Type de retour

ST_GeomCollection

Exemple

Oracle

Créez une table, geomcoll_test, et insérez-y des géométries.

```
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)
);
```

Sélectionnez l'ensemble de géométries dans la table geomcoll_test.

```
SELECT id, sde.st_astext (geometry) Geomcollection
FROM GEOMCOLL_TEST;

        ID     GEOMCOLLECTION

      1901     MULTIPOINT ((1.00000000 2.00000000), (4.00000000 3.00000000),
                      (5.00000000 6.00000000))

      1902     MULTILINESTRING ((33.00000000 2.00000000, 34.00000000
                  3.00000000, 35.00000000 6.00000000),(28.00000000 4.00000000,
                  29.00000000 5.00000000, 31.00000000 8.00000000, 43.00000000
                   12.00000000),(39.00000000 3.00000000, 37.00000000
                   4.00000000, 36.00000000 7.00000000))

      1903     MULTIPOLYGON (((13.00000000 33.00000000, 10.00000000
                  43.00000000, 1.00000000 40.00000000, 7.00000000 36.00000000,
                   13.00000000 33.00000000)),((8.00000000 24.00000000,
                   9.00000000 25.00000000, 1.00000000 28.00000000, 8.00000000
                   24.00000000)), ((3.00000000 3.00000000,5.00000000
                   3.00000000, 4.00000000 6.00000000,3.00000000 3.00000000)))
```

PostgreSQL

Créez une table, geomcoll_test, et insérez-y des géométries.

```
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)
);
```

Sélectionnez l'ensemble de géométries dans la table geomcoll_test.

```
SELECT id, sde.st_astext (geometry)
AS geomcollection
FROM geomcoll_test;

        id     geomcollection

1901     MULTIPOINT (1 2, 4 3, 5 6)

1902     MULTILINESTRING ((33 2, 34        3, 35 6),(28 4,
29 5, 31 8, 43 12),(39 3, 37 4, 36 7))
```

```
1903     MULTIPOLYGON (((13 33, 10 43, 1 40, 7 36,
13 33)),((8 24, 9 25, 1 28, 8 24)), 3 3, 5 3, 4 6, 3 3)))
```

ST_GeomCollFromWKB

Remarque :
PostgreSQL uniquement

Définition

ST_GeomCollFromWKB construit un ensemble de géométries à partir d'une représentation binaire connue.

Syntaxe

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

Type de retour

ST_GeomCollection

Exemple

Remarque :
Les retours à la ligne obligatoires sont insérés par souci de lisibilité. Supprimez-les si vous copiez les instructions.

Créez une table gcoll_test.

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

Insérez des valeurs dans la table.

```
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));
```

Sélectionnez la géométrie à partir de la table 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

Définition

La fonction ST_Geometry crée une géométrie à partir d'une représentation textuelle connue.

Remarque :
Lors de la création de tables spatiales qui seront utilisées avec ArcGIS, il est recommandé de créer la colonne en tant que supertype de la géométrie (par exemple, ST_Geometry) plutôt que de spécifier un sous-type ST_Geometry.

Syntaxe

Oracle

- Pour les chaînes de ligne, les polygones et les points

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

- Pour les points optimisés (qui ne lancent pas d'agent extproc et, par conséquent, traitent la requête plus rapidement)

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

  Utilisez une construction de points optimisée pour effectuer des insertions par lots de quantités élevées de données ponctuelles.

- Pour les cercles paramétriques

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

- Pour les ellipses paramétriques

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

- Pour les secteurs paramétriques

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

PostgreSQL

- Pour les chaînes de ligne, les polygones et les points

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

- Pour les cercles paramétriques

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

- Pour les ellipses paramétriques

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

- Pour les secteurs paramétriques

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

SQLite

- Pour les chaînes de ligne, les polygones et les points

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

- Pour les cercles paramétriques

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

- Pour les ellipses paramétriques

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

- Pour les secteurs paramétriques

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

Type de retour

Oracle et PostgreSQL

ST_Geometry

SQLite

Geometryblob

Exemples

Créer et interroger des entités ponctuelles, de chaîne de ligne et surfaciques

Ces exemples créent une table (geoms) et y insèrent des valeurs ponctuelles, de chaîne de ligne et surfaciques.

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)
);
```

Créer et interroger des cercles paramétriques

Créez une table, radii, et insérez-y des cercles.

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)
);
```

Créer et interroger des ellipses paramétriques

Créez une table, track, et insérez-y des ellipses.

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)
);
```

Créer et interroger des segments paramétriques

Créez une table, pwedge, et insérez-y un secteur.

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

Définition

ST_GeometryN accepte un ensemble et un index entier et retourne l'énième objet ST_Geometry de l'ensemble.

Syntaxe

Oracle et 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)
```

Type de retour

Oracle et PostgreSQL

ST_Geometry

SQLite

Geometryblob

Exemple

Dans cet exemple, un objet multipolygon est créé. Ensuite, la fonction ST_GeometryN est utilisée pour répertorier le deuxième élément de l'objet multipolygon.

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

Définition

ST_GeometryType prend un objet géométrie et renvoie son type de géométrie (par exemple, point, ligne, polygone, multi-point) sous forme de chaîne.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_geometrytype (g1 geometryblob)
```

Type de retour

Varchar(32) (Oracle et PostgreSQL) ou texte (SQLite) contenant l'un des éléments suivants :

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

Exemple

La table geometrytype_test contient la colonne de géométrie g1.

Les instructions INSERT insèrent chaque sous-classe de géométrie dans la colonne g1.

La requête SELECT répertorie le type de géométrie de chaque sous-classe stockée dans la colonne de géométrie g1.

Oracle

```
CREATE TABLE geometrytype_test (g1 sde.st_geometry);

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

INSERT INTO geometrytype_test VALUES (
 sde.st_geometry ('linestring (10.01 20.01, 10.01 30.01, 10.01 40.01)', 4326)
);

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

```
);

INSERT INTO geometrytype_test VALUES (
 sde.st_geometry ('multipoint ((10.02 20.01), (10.32 23.98), (11.92 25.64))', 4326)
);

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

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

```
SELECT UPPER (sde.st_geometrytype (g1)) Geometry_type
FROM GEOMETRYTYPE_TEST;

Geometry_type

ST_POINT
ST_LINESTRING
ST_POLYGON
ST_MULTIPOINT
ST_MULTILINESTRING
ST_MULTIPOLYGON
```

PostgreSQL

```
CREATE TABLE geometrytype_test (g1 sde.st_geometry);

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

INSERT INTO geometrytype_test VALUES (
 sde.st_geometry ('linestring (10.01 20.01, 10.01 30.01, 10.01 40.01)', 4326)
);

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

INSERT INTO geometrytype_test VALUES (
 sde.st_geometry ('multipoint (10.02 20.01, 10.32 23.98, 11.92 25.64)', 4326)
);

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

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

```
);
```

```
SELECT (sde.st_geometrytype (g1))
 AS Geometry_type
 FROM geometrytype_test;

Geometry_type

ST_POINT
ST_LINESTRING
ST_POLYGON
ST_MULTIPOINT
ST_MULTILINESTRING
ST_MULTIPOLYGON
```

SQLite

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

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

INSERT INTO geometrytype_test (g1) VALUES (
 st_geometry ('point (10.02 20.01)', 4326)
);

INSERT INTO geometrytype_test (g1) VALUES (
 st_geometry ('linestring (10.01 20.01, 10.01 30.01, 10.01 40.01)', 4326)
);

INSERT INTO geometrytype_test (g1) VALUES (
 st_geometry ('polygon ((10.02 20.01, 11.92 35.64, 25.02 34.15,
19.15 33.94, 10.02 20.01))', 4326)
);

INSERT INTO geometrytype_test (g1) VALUES (
 st_geometry ('multipoint ((10.02 20.01), (10.32 23.98), (11.92 25.64))', 4326)
);

INSERT INTO geometrytype_test (g1) VALUES (
 st_geometry ('multilinestring ((10.02 20.01, 10.32 23.98,
11.92 25.64), (9.55 23.75, 15.36 30.11))', 4326)
);

INSERT INTO geometrytype_test (g1) VALUES (
 st_geometry ('multipolygon (((10.02 20.01, 11.92 35.64, 25.02 34.15,
19.15 33.94, 10.02 20.01)), ((51.71 21.73, 73.36 27.04, 71.52 32.87,
52.43 31.90,51.71 21.73)))', 4326)
);
```

```
SELECT (st_geometrytype (g1))
 AS "Geometry_type"
 FROM geometrytype_test;

Geometry_type

ST_POINT
ST_LINESTRING
ST_POLYGON
ST_MULTIPOINT
ST_MULTILINESTRING
ST_MULTIPOLYGON
```

ST_GeomFromCollection

Remarque :
PostgreSQL uniquement

Définition

ST_GeomFromCollection renvoie un jeu de lignes st_geometry. Chaque ligne comprend une géométrie et un entier. L'entier représente la position de la géométrie dans le jeu.

Utilisez la fonction ST_GeomFromCollection pour accéder à chaque géométrie individuelle d'une géométrie multi-parties. Lorsque la géométrie en entrée est une collection ou une géométrie multi-parties (par exemple, ST_MultiLineString, ST_MultiPoint, ST_MultiPolygon), ST_GeomFromCollection renvoie un enregistrement pour chaque composant de la collection, et le chemin exprime la position du composant dans la collection.

Si vous utilisez ST_GeomFromCollection sur une géométrie simple (par exemple, ST_Point, ST_LineString, ST_Polygon), un seul enregistrement est renvoyé avec un chemin vide puisqu'il n'existe qu'une seule géométrie.

Syntaxe

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

Pour renvoyer uniquement la géométrie, utilisez `(sde.st_geomfromcollection (shape)).st_geo`.

Pour renvoyer uniquement la position de la la géométrie, utilisez `(sde.st_geomfromcollection (shape)).path[1]`.

Type de retour

Jeu ST_Geometry

Exemple

Dans cet exemple, créez une classe d'entités multiligne (ghanasharktracks) contenant une seule entité avec une forme à quatre parties.

```
--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
  )
 );
```

Pour avoir la confirmation que le champ contient des données, interrogez la table. Utilisez ST_AsText directement sur le champ de forme pour afficher les coordonnées de la forme en tant que texte. Notez que le texte descriptif de l'objet multilinestring est renvoyé.

```
--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))"
```

Pour renvoyer chaque géométrie linestring individuellement, utilisez la fonction ST_GeomFromCollection. Pour afficher la géométrie sous forme de texte, cet exemple utilise la fonction ST_AsText avec la fonction ST_GeomFromCollection.

```
--Return each linestring in the multilinestring
SELECT sde.st_astext((sde.st_geomfromcollection(gst.shape)).st_geo) shapetext,
((sde.st_geomfromcollection(gst.shape)).path[1]) path FROM ghanasharktracks gst;
shapetext
      path
------------------------------------------------------------------------------------------
------------------
"LINESTRING Z ( 1.00000000 1.00000000 0.00000000, 1.00000000 6.00000000
0.00000000)"          1
"LINESTRING Z ( 1.00000000 3.00000000 0.00000000, 3.00000000 3.00000000
0.00000000)"          2
"LINESTRING Z ( 3.00000000 1.00000000 0.00000000, 3.00000000 3.00000000
0.00000000)"          3
"LINESTRING Z ( 4.00000000 1.00000000 0.00000000, 4.00000000 6.00000000
0.00000000)"          4
```

ST_GeomFromText

Remarque :
Utilisée dans Oracle et SQLite uniquement ; pour PostgreSQL, utilisez ST_Geometry.

Définition

ST_GeomFromText accepte une représentation textuelle connue et un ID de référence spatiale et retourne un objet géométrie.

Syntaxe

Oracle

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

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

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

SQLite

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

```
st_geomfromtext (wkt text)
```

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

Type de retour

Oracle

ST_Geometry

SQLite

Geometryblob

Exemple

La table geometry_test contient la colonne d'identifiants entiers gid qui identifie chaque ligne de façon unique et la colonne g1 qui stocke la géométrie.

Les instructions INSERT suivantes insèrent les données dans les colonnes gid et g1 de la table geometry_test. La fonction ST_GeomFromText effectue la conversion des représentations textuelles des géométries en leurs sous-classes instantiables respectives. L'instruction SELECT à la fin permet de s'assurer que les données ont été insérées dans la colonne g1.

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

Définition

ST_GeomFromWKB accepte une représentation binaire connue (WKB) et un ID de référence spatiale pour retourner un objet géométrie.

Syntaxe

Oracle

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

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

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

PostgreSQL

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

```
sde.st_geomfromwkb (esri_shape bytea, srid integer)
```

SQLite

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

```
st_geomfromwkb (wkb blob)
```

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

Type de retour

Oracle et PostgreSQL

ST_Geometry

SQLite

Geometryblob

Exemple

Dans l'exemple suivant, les lignes de résultats ont été remises en forme pour améliorer leur lisibilité. L'espacement de vos résultats variera selon votre affichage en ligne. Le code suivant montre comment utiliser la fonction

ST_GeomFromWKB pour créer et insérer une ligne à partir d'une représentation WKB de ligne. L'exemple suivant insère un enregistrement dans la table sample_gs avec un ID et une géométrie dans le système de référence spatiale 4326 dans une représentation WKB.

Oracle

```
CREATE TABLE sample_gs (
 id integer,
 geometry sde.st_geometry,
 wkb blob
);
INSERT INTO sample_gs (id, geometry) VALUES (
 1901,
 sde.st_geomfromtext ('point (1 2)', 4326)
);
INSERT INTO sample_gs (id, geometry) VALUES (
 1902,
 sde.st_geomfromtext ('linestring (33 2, 34 3, 35 6)', 4326)
);
INSERT INTO sample_gs (id, geometry) VALUES (
 1903,
 sde.st_geomfromtext ('polygon ((3 3, 4 6, 5 3, 3 3))', 4326)
);

UPDATE sample_gs
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 1901;
UPDATE sample_gs
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 1902;
UPDATE sample_gs
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 1903;
SELECT id, sde.st_astext (sde.st_geomfromwkb (wkb, 4326))
 FROM sample_gs;
ID   GEOMETRY
1901 POINT (1.00000000 2.00000000)
1902 LINESTRING (33.00000000 2.00000000, 34.00000000 3.00000000, 35.00000000
6.00000000)
1903 POLYGON ((3.00000000 3.00000000, 5.00000000 3.00000000, 4.00000000 6.00000000,
3.00000000 3.00000000))
```

PostgreSQL

```
CREATE TABLE sample_gs (
 id integer,
 geometry sde.st_geometry,
 wkb bytea);
INSERT INTO sample_gs (id, geometry) VALUES (
 1901,
 sde.st_geometry ('point (1 2)', 4326)
);
INSERT INTO sample_gs (id, geometry) VALUES (
 1902,
 sde.st_geometry ('linestring (33 2, 34 3, 35 6)', 4326)
);
INSERT INTO sample_gs (id, geometry) VALUES (
 1903,
 sde.st_geometry ('polygon ((3 3, 4 6, 5 3, 3 3))', 4326)
```

```
);

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

Remarque :
PostgreSQL uniquement

Définition

ST_GeoSize accepte un objet ST_Geometry et renvoie sa taille en octets.

Syntaxe

```
st_geosize (st_geometry)
```

Type de retour

Entier

Exemple

Vous pouvez découvrir la taille des entités créées dans l'exemple ST_GeomFromWKB en interrogeant la colonne geometry de la table sample_geometries.

```
SELECT st_geosize (geometry)
FROM sample_geometries;

st_geosize
        512
        592
        616
```

ST_InteriorRingN

Définition

ST_InteriorRingN renvoie la énième boucle intérieure d'un polygone sous la forme d'un objet ST_LineString.

L'ordre des boucles ne peut être prédéfini puisque les boucles sont agencées d'après les règles définies par les routines internes de vérification de la géométrie et non d'après leur orientation géométrique. Si la valeur d'index dépasse le nombre de boucles intérieures d'un polygone, la fonction retourne la valeur NULL.

Syntaxe

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)
```

Type de retour

ST_LineString

Exemple

Créez une table, sample_polys, et ajoutez un enregistrement, puis sélectionnez l'ID et la géométrie de la boucle intérieure.

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

Définition

ST_Intersection accepte deux objets géométrie et retourne l'ensemble d'intersection sous la forme d'un objet géométrie bidimensionnel.

Syntaxe

Oracle et PostgreSQL

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

SQLite

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

Type de retour

Oracle et PostgreSQL

ST_Geometry

SQLite

Geometryblob

Exemple

Le capitaine des pompiers doit connaître les surfaces des hôpitaux, écoles et maisons de retraite situées dans un rayon de risque de contamination par des déchets dangereux.

Les hôpitaux, écoles et maisons de retraite sont stockés dans la table population créée à l'aide de l'instruction CREATE TABLE ci-dessous. La colonne shape, définie avec le type polygone, stocke les contours respectifs des zones sensibles.

Les sites à risque sont stockés dans la table waste_sites créée à l'aide de l'instruction CREATE TABLE ci-dessous. La colonne site, définie avec le type point, stocke un emplacement qui est le centre géographique de chaque site à risque.

La fonction ST_Buffer génère une zone tampon qui entoure les sites de dépôt de déchets dangereux. La fonction ST_Intersection génère des polygones à partir de l'intersection entre les zones tampons des sites de dépôt de déchets dangereux et les zones sensibles.

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

Définition

ST_Intersects retourne 1 (Oracle et SQLite) ou t (PostgreSQL) si l'intersection de deux géométries n'est pas un ensemble vide ; dans le cas contraire, la fonction retourne 0 (Oracle et SQLite) ou f (PostgreSQL).

Syntaxe

Oracle et PostgreSQL

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

SQLite

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

Type de retour

Booléen

Exemple

Le capitaine des pompiers demande une liste des zones sensibles situées dans un rayon donné d'un site de dépôt de déchets dangereux.

Les zones sensibles sont stockées dans la table sensitive_areas. La colonne shape, définie avec le type polygone, stocke les contours respectifs des zones sensibles.

Les sites dangereux sont stockés dans la table hazardous_sites. La colonne site, définie avec le type point, stocke un emplacement qui est le centre géographique de chaque site à risque.

La requête SELECT crée un rayon de zone tampon autour de chaque site dangereux et renvoie une liste de zones sensibles qui intersectent les zones tampon des sites dangereux.

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

Définition

ST_Is3d prend un objet ST_Geometry comme paramètre d'entrée et renvoie 1 (Oracle et SQLite) ou t (PostgreSQL) si la géométrie donnée contient des coordonnées z ; sinon, il renvoie 0 (Oracle et SQLite) ou f (PostgreSQL).

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_is3d (geometry1 geometryblob)
```

Type de retour

Booléen

Exemple

Cet exemple crée une table, is3d_test, et l'alimente avec des enregistrements.

Ensuite, à l'aide de ST_Is3d, vérifiez si l'un des enregistrements contient une coordonnée z.

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

Définition

ST_IsClosed accepte un objet ST_LineString ou ST_MultiLineString et renvoie 1 (Oracle et SQLite) ou t (PostgreSQL) s'il est fermé. Dans le cas contraire, la fonction renvoie 0 (Oracle et SQLite) ou f (PostgreSQL).

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_isclosed (geometry1 geometryblob)
```

Type de retour

Booléen

Exemples

Test d'un objet linestring

La table closed_linestring est créée avec une seule colonne d'objets linestring.

Les instructions INSERT suivantes insèrent deux entrées dans la table closed_linestring. La première entrée n'est pas un objet linestring fermé, la deuxième en est un.

La requête renvoie les résultats de la fonction ST_IsClosed. La première ligne renvoie un 0 ou f puisque l'objet linestring n'est pas fermé et la deuxième ligne renvoie un 1 ou t puisque l'objet linestring est fermé.

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
```

Test d'un objet multilinestring

La table closed_mlinestring est créée avec une seule colonne d'objets ST_MultiLineString.

Les instructions INSERT suivantes insèrent un enregistrement ST_MultiLineString non fermé et un enregistrement fermé.

Cette requête liste les résultats de la fonction ST_IsClosed. La première ligne renvoie la valeur 0 ou f puisque l'objet multilinestring n'est pas fermé. La deuxième ligne renvoie la valeur 1 ou t puisque l'objet multilinestring stocké dans la colonne ln1 est fermé. Un objet multilinestring est fermé si tous ses éléments linestring sont fermés.

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

Définition

La fonction ST_IsEmpty renvoie 1 (Oracle et SQLite) ou t (PostgreSQL) si l'objet ST_Geometry est vide. Dans le cas contraire, elle renvoie 0 (Oracle et SQLite) ou f (PostgreSQL).

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_isempty (geometry1 geometryblob)
```

Type de retour

Booléen

Exemple

L'instruction CREATE TABLE ci-dessous crée la table empty_test avec la valeur geotype, qui stocke le type de données des sous-classes stockées dans la colonne g1.

Les instructions INSERT insèrent deux enregistrements pour chacune des sous-classes de géométrie point, linestring et polygon : une sous-classe est vide et l'autre ne l'est pas.

La requête SELECT retourne le type de géométrie de la colonne geotype et les résultats de la fonction ST_IsEmpty.

Oracle

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

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

INSERT INTO EMPTY_TEST VALUES (
 'Point',
 sde.st_pointfromtext ('point empty', 4326)
);

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

INSERT INTO EMPTY_TEST VALUES (
 'Linestring',
```

```
 sde.st_linefromtext ('linestring empty', 4326)
);

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

INSERT INTO EMPTY_TEST VALUES (
 'Polygon',
 sde.st_polyfromtext('polygon empty', 4326)
);
```

```
SELECT geotype, sde.st_isempty (g1) Is_it_empty
FROM EMPTY_TEST;

GEOTYPE    Is_it_empty

Point         0
Point         1
Linestring    0
Linestring    1
Polygon       0
Polygon       1
```

PostgreSQL

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

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

INSERT INTO empty_test VALUES (
 'Point',
 sde.st_point ('point empty', 4326)
);

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

INSERT INTO empty_test VALUES (
 'Linestring',
 sde.st_linestring ('linestring empty', 4326)
);

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

```
INSERT INTO empty_test VALUES (
 'Polygon',
 sde.st_polygon ('polygon empty', 4326)
);
```

```
SELECT geotype, sde.st_isempty (g1)
 AS Is_it_empty
 FROM empty_test;

geotype   is_it_empty

Point         f
Point         t
Linestring    f
Linestring    t
Polygon       f
Polygon       f
```

SQLite

```
CREATE TABLE empty_test (
 geotype text(20)
);

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

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

INSERT INTO empty_test VALUES (
 'Point',
 st_point ('point empty', 4326)
);

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

INSERT INTO empty_test VALUES (
 'Linestring',
 st_linestring ('linestring empty', 4326)
);

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

```
INSERT INTO empty_test VALUES (
 'Polygon',
 st_polygon ('polygon empty', 4326)
);
```

```
SELECT geotype, st_isempty (g1)
 AS "Is_it_empty"
 FROM empty_test;

GEOTYPE    Is_it_empty

Point         0
Point         1
Linestring    0
Linestring    1
Polygon       0
Polygon       1
```

ST_IsMeasured

Définition

ST_IsMeasured prend un objet géométrie comme paramètre d'entrée et renvoie 1 (Oracle et SQLite) ou t (PostgreSQL) si la géométrie donnée comporte des mesures ; sinon, 0 (Oracle et SQLite) ou f (PostgreSQL) est renvoyé.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_ismeasured (geometry1 geometryblob)
```

Type de retour

Booléen

Exemple

Créez une table, ism_test, insérez-y des valeurs, puis déterminez quelles lignes de la table ism_test contiennent des mesures.

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

Définition

ST_IsRing accepte un objet ST_LineString et retourne 1 (Oracle et SQLite) ou t (PostgreSQL) si l'objet est un anneau (par exemple, l'objet ST_LineString est fermé et simple) ; dans le cas contraire, la fonction retourne 0 (Oracle et SQLite) ou f (PostgreSQL).

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_isring (line1 geometryblob)
```

Type de retour

Booléen

Exemple

La table ring_linestring est créée avec une seule colonne ST_LineString, ln1.

Les instructions INSERT suivantes insèrent trois objets linestring dans la colonne ln1. La première ligne contient une chaîne de lignes non fermée, qui n'est pas une boucle. La deuxième ligne contient une chaîne de lignes fermée et simple qui est une boucle. La troisième ligne contient une chaîne de lignes fermée mais non simple car elle intersecte son propre intérieur. Il ne s'agit pas non plus d'un anneau.

La requête SELECT renvoie les résultats de la fonction ST_IsRing. La première ligne renvoie la valeur 0 ou f puisque les objets linestring ne sont pas des boucles, alors que les deuxième et troisième lignes renvoient la valeur 1 ou t, car ce sont des boucles.

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

Définition

ST_IsSimple renvoie 1 (Oracle et SQLite) ou t (PostgreSQL) si l'objet géométrie est simple conformément à la spécification de l'OGC (Open Geospatial Consortium). Dans le cas contraire, la fonction renvoie 0 (Oracle et SQLite) ou f (PostgreSQL).

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_issimple (geometry1 geometryblob)
```

Type de retour

Booléen

Exemple

La table issimple_test est créée avec deux colonnes. La colonne pid est de type de données smallint et contient l'identifiant unique de chaque enregistrement. La colonne g1 stocke les exemples de géométrie simples et non simples.

Les instructions INSERT suivantes insèrent deux enregistrements dans la table issimple_test. Le premier est un objet linestring simple puisqu'il n'intersecte pas son intérieur. Le second est non simple, conformément à la spécification de l'OGC, puisqu'il intersecte son intérieur.

La requête suivante renvoie les résultats de la fonction ST_IsSimple. Le premier enregistrement renvoie la valeur 1 ou t puisque l'objet linestring est simple, alors que le deuxième enregistrement renvoie la valeur 0 ou f puisque l'objet linestring n'est pas simple.

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

Définition

ST_Length renvoie la longueur d'une chaîne de lignes ou d'une chaîne multiligne.

Syntaxe

Oracle et 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)
```

Pour obtenir la liste des noms d'unités pris en charge, reportez-vous à la rubrique ST_Distance.

Type de retour

Double précision

Exemple

Un écologiste local qui étudie les réseaux migratoires des populations de saumon dans les cours d'eau du pays souhaite connaître la longueur de chaque rivière et système hydrographique du pays.

La table waterways est créée avec les colonnes ID et name, qui identifient chaque réseau ou chevelu hydrographique stocké dans la table. La colonne water est de type multilinestring puisque les rivière et les réseaux hydrographiques sont souvent un agrégat de plusieurs objets linestring.

La requête SELECT renvoie le nom du système, ainsi que la longueur du système générée par la fonction ST_Length. Dans Oracle et PostgreSQL, les unités sont celles utilisées par le système de coordonnées. Dans SQLite, les kilomètres sont spécifiés.

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

Remarque :
Prise en charge dans Oracle et SQLite uniquement ; pour PostgreSQL, utilisez ST_LineString.

Définition

ST_LineFromText accepte une représentation textuelle connue du type ST_LineString et un ID de référence spatiale et retourne un objet ST_LineString.

Syntaxe

Oracle

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

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

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

SQLite

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

```
st_linefromtext (wkt text)
```

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

Type de retour

ST_LineString

Exemple

La table linestring_test est créée avec une seule colonne ln1 de type ST_LineString.

L'instruction INSERT suivante insère un objet ST_LineString dans la colonne ln1 à l'aide de la fonction ST_LineFromText.

Oracle

```
CREATE TABLE linestring_test (ln1 sde.st_geometry);
```

```
INSERT INTO LINESTRING_TEST VALUES (
 sde.st_linefromtext ('linestring (10.01 20.03, 35.93 19.04)', 4326)
);
```

SQLite

```
CREATE TABLE linestring_test (id integer);
SELECT AddGeometryColumn (
 NULL,
 'linestring_test',
 'ln1',
 4326,
 'linestring',
 'xy',
 'null'
);
```

```
INSERT INTO LINESTRING_TEST (id, ln1) VALUES (
 1,
 st_linefromtext ('linestring (10.01 20.03, 35.93 19.04)', 4326)
);
```

ST_LineFromWKB

Définition

ST_LineFromWKB accepte une représentation binaire connue (WKB) de type ST_LineString et un ID de référence spatiale et retourne un objet ST_LineString.

Syntaxe

Oracle

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

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

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

PostgreSQL

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

SQLite

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

```
st_linefromwkb (wkb blob)
```

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

Type de retour

ST_LineString

Exemple

Les commandes suivantes créent une table (sample_lines) et utilisent la fonction ST_LineFromWKB pour insérer des lignes à partir d'une représentation WKB. L'enregistrement est inséré dans la table sample_lines avec un ID et une ligne dans le système de référence spatiale 4326 dans la représentation WKB.

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

Définition

La fonction ST_LineString est une fonction accesseur qui crée une chaîne de lignes à partir d'une représentation textuelle connue.

Remarque :
Lors de la création de tables spatiales qui seront utilisées avec ArcGIS, il est recommandé de créer la colonne en tant que supertype de la géométrie (par exemple, ST_Geometry) plutôt que de spécifier un sous-type ST_Geometry.

Syntaxe

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)
```

Type de retour

ST_LineString

Exemple

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

Définition

ST_M accepte un objet ST_Point comme paramètre en entrée et renvoie sa coordonnée de mesure (m).

Dans SQLite, la fonction ST_M permet également de mettre à jour une valeur de mesure.

Syntaxe

Oracle et PostgreSQL

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

SQLite

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

Type de retour

Oracle et PostgreSQL

Nombre

SQLite

Double précision si la requête porte sur une valeur de mesure ; objet geometryblob pour la mise à jour d'une valeur de mesure

Exemples

Oracle

La table m_test est créée et trois points y sont insérés. Les trois contiennent des valeurs de mesure. Une instruction SELECT est exécutée avec la fonction ST_M pour renvoyer la valeur de mesure de chaque point.

```
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

La table m_test est créée et trois points y sont insérés. Les trois contiennent des valeurs de mesure. Une instruction SELECT est exécutée avec la fonction ST_M pour renvoyer la valeur de mesure de chaque point.

```
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

Dans le premier exemple, la table m_test est créée et trois points y sont insérés. Les trois contiennent des valeurs de mesure. Une instruction SELECT est exécutée avec la fonction ST_M pour renvoyer la valeur de mesure de chaque point.

```
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
```

Dans ce deuxième exemple, la valeur de mesure est mise à jour pour l'enregistrement 3 dans la table m_test.

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

ST_MaxM

Définition

La fonction ST_MaxM part d'une géométrie comme paramètre en entrée et renvoie sa coordonnée m maximale.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_maxm (geometry1 geometryblob)
```

Type de retour

Oracle et PostgreSQL

Nombre

Si aucune valeur m n'est présente, la valeur Null est renvoyée.

SQLite

Double précision

Si aucune valeur m n'est présente, la valeur Null est renvoyée.

Exemple

La table maxm_test est créée et deux polygones y sont insérés. Ensuite, la fonction ST_MaxM est exécutée pour déterminer la valeur m maximale de chaque polygone.

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

Définition

La fonction ST_MaxX part d'une géométrie comme paramètre en entrée et renvoie sa coordonnée x maximale.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_maxx (geometry1 geometryblob)
```

Type de retour

Oracle et PostgreSQL

Nombre

SQLite

Double précision

Exemple

La table maxx_test est créée et deux polygones y sont insérés. Ensuite, la fonction ST_MaxX est exécutée pour déterminer la valeur x maximale de chaque polygone.

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

Définition

La fonction ST_MaxY part d'une géométrie comme paramètre en entrée et renvoie sa coordonnée y maximale.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_maxy (geometry1 geometryblob)
```

Type de retour

Oracle et PostgreSQL

Nombre

SQLite

Double précision

Exemple

La table maxy_test est créée et deux polygones y sont insérés. Ensuite, la fonction ST_MaxY est exécutée pour déterminer la valeur y maximale de chaque polygone.

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

Définition

La fonction ST_MaxZ part d'une géométrie comme paramètre en entrée et renvoie sa coordonnée z maximale.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_maxz (geometry1 geometryblob)
```

Type de retour

Oracle et PostgreSQL

Nombre

Si aucune valeur z n'est présente, la valeur Null est renvoyée.

SQLite

Double précision

Si aucune valeur z n'est présente, la valeur Null est renvoyée.

Exemple

Dans l'exemple suivant, la table maxz_test est créée et deux polygones y sont insérés. Ensuite, la fonction ST_MaxZ est exécutée pour renvoyer la valeur z maximale de chaque polygone.

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

Définition

La fonction ST_MinM part d'une géométrie comme paramètre en entrée et renvoie sa coordonnée m minimale.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_minm (geometry1 geometryblob)
```

Type de retour

Oracle et PostgreSQL

Nombre

Si aucune valeur m n'est présente, la valeur Null est renvoyée.

SQLite

Double précision

Si aucune valeur m n'est présente, la valeur Null est renvoyée.

Exemple

La table minm_test est créée et deux polygones y sont insérés. Ensuite, la fonction ST_MinM est exécutée pour déterminer la valeur de mesure minimale de chaque polygone.

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

Définition

La fonction ST_MinX part d'une géométrie comme paramètre en entrée et renvoie sa coordonnée x minimale.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_minx (geometry1 geometryblob)
```

Type de retour

Oracle et PostgreSQL

Nombre

SQLite

Double précision

Exemple

La table minx_test est créée et deux polygones y sont insérés. Ensuite, la fonction ST_MinX est exécutée pour déterminer la valeur de coordonnée x minimale de chaque polygone.

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

Définition

La fonction ST_MinY part d'une géométrie comme paramètre en entrée et renvoie sa coordonnée y minimale.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_miny (geometry1 geometryblob)
```

Type de retour

Oracle et PostgreSQL

Nombre

SQLite

Double précision

Exemple

La table miny_test est créée et deux polygones y sont insérés. Ensuite, la fonction ST_MinY est exécutée pour déterminer la valeur de coordonnée x minimale de chaque polygone.

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

Définition

La fonction ST_MinZ part d'une géométrie comme paramètre en entrée et renvoie sa coordonnée z minimale.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_minz (geometry1 geometryblob)
```

Type de retour

Oracle et PostgreSQL

Nombre

Si aucune valeur z n'est présente, la valeur Null est renvoyée.

SQLite

Double précision

Si aucune valeur z n'est présente, la valeur Null est renvoyée.

Exemple

La table minz_test est créée et deux polygones y sont insérés. Ensuite, la fonction ST_MinZ est exécutée pour déterminer la valeur de coordonnée z minimale de chaque polygone.

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

Remarque :
Utilisée dans Oracle et SQLite uniquement ; pour PostgreSQL, utilisez ST_MultiLineString.

Définition

ST_MLineFromText accepte une représentation textuelle connue de type ST_MultiLineString et un ID de référence spatiale et retourne un objet ST_MultiLineString.

Syntaxe

Oracle

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

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

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

SQLite

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

```
st_mlinefromtext (wkt text)
```

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

Type de retour

ST_MultiLineString

Exemple

La table mlinestring_test est créée avec la colonne gid de type smallint qui identifie de façon unique la ligne et la colonne ml1 de type ST_MultiLineString.

L'instruction INSERT insère l'objet ST_MultiLineString à l'aide de la fonction ST_MLineFromText.

Oracle

```
CREATE TABLE mlinestring_test (
 gid smallint,
 ml1 sde.st_geometry
);
```

```
INSERT INTO MLINESTRING_TEST VALUES (
 1,
 sde.st_mlinefromtext ('multilinestring ((10.01 20.03, 10.52 40.11, 30.29 41.56,
31.78 10.74), (20.93 20.81, 21.52 40.10))', 4326)
);
```

SQLite

```
CREATE TABLE mlinestring_test (
 gid integer
);
SELECT AddGeometryColumn (
 NULL,
 'mlinestring_test',
 'ml1',
 4326,
 'multilinestring',
 'xy',
 'null'
);
```

```
INSERT INTO MLINESTRING_TEST VALUES (
 1,
 st_mlinefromtext ('multilinestring ((10.01 20.03, 10.52 40.11, 30.29 41.56,
31.78 10.74), (20.93 20.81, 21.52 40.10))', 4326)
);
```

ST_MLineFromWKB

Définition

ST_MLineFromWKB accepte une représentation binaire connue (WKB) de type ST_MultiLineString et un ID de référence spatiale et crée un objet ST_MultiLineString.

Syntaxe

Oracle

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

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

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

PostgreSQL

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

SQLite

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

```
st_mlinefromwkb (wkb blob)
```

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

Type de retour

ST_MultiLineString

Exemple

Cet exemple illustre l'utilisation de la fonction ST_MLineFromWKB pour la création d'un objet multilinestring à partir de sa représentation binaire connue. La géométrie est un objet multilinestring dans le système de référence spatiale 4326. Dans cet exemple, l'objet multilinestring est stocké avec ID = 10 dans la colonne géométrie de la table sample_mlines, et la colonne wkb est mise à jour avec sa représentation binaire connue (avec la fonction ST_AsBinary). Enfin, la fonction ST_MLineFromWKB est utilisée pour renvoyer l'objet multilinestring depuis la colonne wkb. La table sample_mlines comporte une colonne geometry, dans laquelle l'objet multilinestring est stocké, et une colonne wkb dans laquelle la représentation WKB de l'objet multilinestring est stockée.

L'instruction SELECT inclut la fonction ST_MLineFromWKB, qui permet d'extraire l'objet multilinestring de la colonne wkb.

Oracle

```
CREATE TABLE sample_mlines (
 id integer,
 geometry sde.st_geometry,
 wkb blob
);
INSERT INTO SAMPLE_MLINES (id, geometry) VALUES (
 10,
 sde.st_multilinestring ('multilinestring ((61 2, 64 3, 65 6), (58 4, 59 5, 61 8), (69
3, 67 4, 66 7, 68 9))', 4326)
);
UPDATE SAMPLE_MLINES
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 10;
```

```
SELECT id, sde.st_astext (sde.st_mlinefromwkb (wkb,0)) MULTI_LINE_STRING
 FROM SAMPLE_MLINES
 WHERE id = 10;
ID    MULTI_LINE_STRING
10    MULTILINESTRING ((61.00000000 2.00000000, 64.00000000 3.00000000, 65.00000000
6.00000000), (58.00000000 4.00000000, 59.00000000 5.00000000, 61.00000000 8.0000000),
(69.00000000 3.00000000, 67.00000000 4.00000000, 66.00000000 7.00000000, 68.00000000
9.00000000 ))
```

PostgreSQL

```
CREATE TABLE sample_mlines (
 id integer,
 geometry sde.st_geometry,
 wkb bytea);
INSERT INTO sample_mlines (id, geometry) VALUES (
 10,
 sde.st_multilinestring ('multilinestring ((61 2, 64 3, 65 6), (58 4, 59 5, 61 8), (69
3, 67 4, 66 7, 68 9))', 4326)
);
UPDATE sample_mlines
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 10;
```

```
SELECT id, sde.st_astext (sde.st_mlinefromwkb (wkb,4326))
 AS MULTI_LINE_STRING
 FROM sample_mlines
 WHERE id = 10;
id   multi_line_string
10    MULTI_LINE_STRING ((61 2, 64 3, 65 6), (58 4, 59 5,61 8), (69 3, 67 4, 66 7, 68 9
))
```

SQLite

```
CREATE TABLE sample_mlines (
 id integer,
 wkb blob);
SELECT AddGeometryColumn (
 NULL,
```

```
 'sample_mlines',
 'geometry',
 4326,
 'multilinestring',
 'xy',
 'null'
);
INSERT INTO sample_mlines (id, geometry) VALUES (
 10,
 st_multilinestring ('multilinestring ((61 2, 64 3, 65 6), (58 4, 59 5, 61 8), (69 3,
67 4, 66 7, 68 9))', 4326)
);
UPDATE sample_mlines
 SET wkb = st_asbinary (geometry)
 WHERE id = 10;
```

```
SELECT id, st_astext (st_mlinefromwkb (wkb,4326))
 AS MULTI_LINE_STRING
 FROM sample_mlines
 WHERE id = 10;
id   multi_line_string
10    MULTI_LINE_STRING ((61.00000000 2.00000000, 64.00000000 3.00000000, 65.00000000
6.00000000),
 (58.00000000 4.00000000, 59.00000000 5.00000000, 61.00000000 8.00000000),
 (69.00000000 3.00000000, 67.00000000 4.00000000, 66.00000000 7.00000000, 68.00000000
9.00000000 ))
```

ST_MPointFromText

Remarque :
Oracle et SQLite uniquement ; pour PostgreSQL, utilisez ST_MultiPoint.

Définition

ST_MPointFromText accepte une représentation textuelle connue (WKT) de type ST_MultiPoint et un ID de référence spatiale, puis crée un objet ST_Multipoint.

Syntaxe

Oracle

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

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

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

SQLite

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

```
st_mpointfromtext (wkt text)
```

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

Type de retour

ST_MultiPoint

Exemple

La table multipoint_test est créée avec une seule colonne mpt1 de type ST_MultiPoint.

L'instruction INSERT suivante insère un objet multipoint dans la colonne mpt1 à l'aide de la fonction ST_MpointFromText.

Oracle

```
CREATE TABLE multipoint_test (mpt1 sde.st_geometry);
```

```
INSERT INTO MULTIPOINT_TEST VALUES (
 sde.st_mpointfromtext ('multipoint ((10.01 20.03), (10.52 40.11), (30.29 41.56),
(31.78 10.74))', 4326));
```

SQLite

```
CREATE TABLE multipoint_test (id integer);

SELECT AddGeometryColumn (
 NULL,
 'multipoint_test',
 'pt1',
 4326,
 'multipoint',
 'xy',
 'null'
);
```

```
INSERT INTO MULTIPOINT_TEST VALUES (
 1,
 st_mpointfromtext ('multipoint ((10.01 20.03), (10.52 40.11), (30.29 41.56), (31.78
10.74))', 4326));
```

ST_MPointFromWKB

Définition

ST_MPointFromText prend une représentation binaire connue (WKB) de type ST_MultiPoint et un identifiant de référence spatiale et crée un ST_MultiPoint.

Syntaxe

Oracle

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

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

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

PostgreSQL

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

SQLite

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

```
st_mpointfromwkb (wkb blob)
```

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

Type de retour

ST_MultiPoint

Exemple

Cet exemple illustre l'utilisation de la fonction ST_MPointFromWKB pour la création d'un objet multi-points à partir de sa représentation binaire connue. La géométrie est un objet multi-points dans le système de référence spatiale 4326. Dans cet exemple, l'objet multi-points est stocké avec ID = 10 dans la colonne GEOMETRY de la table SAMPLE_MPOINTS, et la colonne WKB est mise à jour avec sa représentation binaire connue (avec la fonction ST_AsBinary). Enfin, la fonction ST_MPointFromWKB est utilisée pour renvoyer l'objet multipoint depuis la colonne WKB. La table SAMPLE_MPOINTS a une colonne GEOMETRY, où est stocké l'objet multipoint et une colonne WKB, où est stockée la représentation binaire connue de l'objet multipoint.

Dans l'instruction SELECT suivante, la fonction ST_MPointFromWKB permet de récupérer l'objet multipoint de la colonne WKB.

Oracle

```
CREATE TABLE sample_mpoints (
 id integer,
 geometry sde.st_geometry,
 wkb blob
);

INSERT INTO SAMPLE_MPOINTS (id, geometry) VALUES (
 10,
 sde.st_multipoint ('multipoint ((4 14), (35 16), (24 13))', 4326)
);

UPDATE SAMPLE_MPOINTS
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 10;
```

```
SELECT id, sde.st_astext (sde.st_mpointfromwkb (wkb,4326)) MULTI_POINT
 FROM SAMPLE_MPOINTS
 WHERE id = 10;


ID           MULTI_POINT

10    MULTIPOINT ((4.00000000 14.00000000), (35.00000000 16.00000000), (24.00000000
13.00000000))
```

PostgreSQL

```
CREATE TABLE sample_mpoints (
 id integer,
 geometry sde.st_geometry,
 wkb bytea
);

INSERT INTO sample_mpoints (id, geometry) VALUES (
 10,
 sde.st_multipoint ('multipoint (4 14, 35 16, 24 13)', 4326)
);

UPDATE sample_mpoints
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 10;
```

```
SELECT id, sde.st_astext (sde.st_mpointfromwkb (wkb,4326))
 AS "MULTI_POINT"
 FROM sample_mpoints
 WHERE id = 10;


id           MULTI_POINT

10     MULTIPOINT (4 14, 35 16, 24 13)
```

SQLite

```
CREATE TABLE sample_mpoints (
 id integer,
 wkb blob
);

SELECT AddGeometryColumn (
 NULL,
 'sample_mpoints',
 'geometry',
 4326,
 'multipointzm',
 'xyzm',
 'null'
);

INSERT INTO SAMPLE_MPOINTS (id, geometry) VALUES (
 10,
 st_multipoint ('multipoint ((4 14), (35 16), (24 13))', 4326)
);

UPDATE sample_mpoints
 SET wkb = st_asbinary (geometry)
 WHERE id = 10;
```

```
SELECT id AS "ID",
 st_astext (st_mpointfromwkb (wkb,4326))
 AS "MULTI_POINT"
 FROM sample_mpoints
 WHERE id = 10;


ID           MULTI_POINT

10    MULTIPOINT ((4.00000000 14.00000000), (35.00000000 16.00000000), (24.00000000
13.00000000))
```

ST_MPolyFromText

Remarque :
Oracle et SQLite uniquement ; pour PostgreSQL, utilisez ST_MultiPolygon.

Définition

ST_MPointFromText accepte une représentation textuelle connue (WKT) de type ST_MultiPolygon et un ID de référence spatiale et retourne un objet ST_MultiPolygon.

Syntaxe

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 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)
```

Type de retour

ST_MultiPolygon

Exemple

La table multipolygon_test est créée avec une colonne ST_MultiPolygon, mpl1.

L'instruction INSERT insère un objet ST_MultiPolygon dans la colonne mpl1 à l'aide de la fonction ST_MpolyFromText.

Oracle

```
CREATE TABLE mpolygon_test (mpl1 sde.st_geometry);
```

```
INSERT INTO MPOLYGON_TEST VALUES (
 sde.st_mpolyfromtext ('multipolygon (((10.01 20.03, 10.52 40.11, 30.29 41.56,
31.78 10.74, 10.01 20.03), (21.23 15.74, 21.34 35.21, 28.94 35.35,
29.02 16.83, 21.23 15.74)), ((40.91 10.92, 40.56 20.19, 50.01 21.12,
51.34 9.81, 40.91 10.92)))', 4326)
);
```

SQLite

```
CREATE TABLE mpolygon_test (id integer);

SELECT AddGeometryColumn(
 NULL,
 'mpolygon_test',
 'mpl1',
 4326,
 'multipolygon',
 'xy',
 'null'
);
```

```
INSERT INTO MPOLYGON_TEST VALUES (
 1,
 st_mpolyfromtext ('multipolygon (((10.01 20.03, 10.52 40.11, 30.29 41.56,
31.78 10.74, 10.01 20.03), (21.23 15.74, 21.34 35.21, 28.94 35.35,
29.02 16.83, 21.23 15.74)), ((40.91 10.92, 40.56 20.19, 50.01 21.12,
51.34 9.81, 40.91 10.92)))', 4326)
);
```

ST_MPolyFromWKB

Définition

ST_MPointFromWKB accepte une représentation binaire connue (WKB) de type ST_MultiPolygon et un ID de référence spatiale pour retourner un objet ST_MultiPolygon

Syntaxe

Oracle

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

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

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

PostgreSQL

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

SQLite

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

```
st_mpolyfromwkb (wkb blob)
```

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

Type de retour

ST_MultiPolygon

Exemple

Cet exemple illustre comment ST_MPolyFromWKB peut être utilisé pour créer un objet multisurfacique à partir de sa représentation binaire connue. La géométrie est un objet multisurfacique dans le système de référence spatiale 4326. Dans cet exemple, l'objet multisurfacique est stocké avec ID = 10 dans la colonne géométrie de la table sample_mpolys, et la colonne wkb est mise à jour avec sa représentation binaire connue (avec la fonction ST_AsBinary). Enfin, la fonction ST_MPolyFromWKB est utilisée pour renvoyer l'objet multisurfacique depuis la colonne wkb. La table sample_mpolys comporte une colonne geometry dans laquelle l'objet multipolygon est stocké, et une colonne wkb dans laquelle la représentation WKB de l'objet multipolygon est stockée.

L'instruction SELECT inclut la fonction ST_MPolyFromWKB, qui permet de récupérer le multipolygone de la colonne WKB.

Oracle

```
CREATE TABLE sample_mpolys (
 id integer,
 geometry sde.st_geometry,
 wkb blob
);
INSERT INTO SAMPLE_MPOLYS (id, geometry) VALUES (
 10,
 sde.st_multipolygon ('multipolygon (((1 72, 4 79, 5 76, 1 72), (10 20, 10 40, 30 41,
10 20), (9 43, 7 44, 6 47, 9 43)))', 4326)
);
UPDATE SAMPLE_MPOLYS
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 10;
```

```
SELECT id, sde.st_astext (sde.st_mpolyfromwkb (wkb,4326)) MULTIPOLYGON
 FROM SAMPLE_MPOLYS
 WHERE id = 10;
ID   MULTIPOLYGON
10   MULTIPOLYGON (((10.00000000 20.00000000, 30.00000000 41.00000000, 10.00000000
40.00000000, 10.00000000 20.00000000)), (1.00000000 72.00000000, 5.00000000
76.00000000, 4.00000000 79.0000000, 1.00000000 72,00000000)), (9.00000000 43.00000000,
6.00000000 47.00000000, 7.00000000 44.00000000, 9.00000000 43.00000000 )))
```

PostgreSQL

```
CREATE TABLE sample_mpolys (
 id integer,
 geometry sde.st_geometry,
 wkb bytea
);
INSERT INTO sample_mpolys (id, geometry) VALUES (
 10,
 sde.st_multipolygon ('multipolygon (((1 72, 4 79, 5 76, 1 72), (10 20, 10 40, 30 41,
10 20), (9 43, 7 44, 6 47, 9 43)))', 4326)
);
UPDATE sample_mpolys
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 10;
```

```
SELECT id, sde.st_astext (sde.st_mpolyfromwkb (wkb,4326))
 AS MULTIPOLYGON
 FROM sample_mpolys
 WHERE id = 10;
id   multipolygon
10   MULTIPOLYGON (((10 20, 30 41, 10 40, 10 20)),
 ((1 72, 5 76, 4 79, 1 72)), ((9 43, 6 47, 7 44, 9 43)))
```

SQLite

```
CREATE TABLE sample_mpolys (
 id integer,
 wkb blob
);
```

```
SELECT AddGeometryColumn(
 NULL,
 'sample_mpolys',
 'geometry',
 4326,
 'multipolygon',
 'xy',
 'null'
);
INSERT INTO SAMPLE_MPOLYS (id, geometry) VALUES (
 10,
 st_multipolygon ('multipolygon (((1 72, 4 79, 5 76, 1 72), (10 20, 10 40, 30 41, 10
20), (9 43, 7 44, 6 47, 9 43)))', 4326)
);
UPDATE SAMPLE_MPOLYS
 SET wkb = st_asbinary (geometry)
 WHERE id = 10;
```

```
SELECT id, st_astext (st_mpolyfromwkb (wkb,4326))
 AS "Multipolygon"
 FROM sample_mpolys
 WHERE id = 10;
id   Multipolygon
10   MULTIPOLYGON ((( 10.00000000 20.00000000, 30.00000000 41.00000000, 10.00000000
40.00000000, 10.00000000 20.00000000)),
 ((1.00000000 72.00000000, 5.00000000 76.00000000, 4.00000000 79.00000000, 1.00000000
72.00000000)),
 ((9.00000000 43.00000000, 6.00000000 47.00000000, 7.00000000 44.00000000, 9.00000000
43.00000000)))
```

ST_MultiCurve

Remarque :
Oracle uniquement

Définition

La fonction ST_MultiCurve crée une entité multicourbe à partir d'une représentation textuelle connue.

Syntaxe

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

Type de retour

ST_MultiLinestring

Exemple

```
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

Définition

La fonction ST_MultiLineString crée une entité multi-polygones à partir d'une représentation textuelle connue.

Remarque :
Lors de la création de tables spatiales qui seront utilisées avec ArcGIS, il est recommandé de créer la colonne en tant que supertype de la géométrie (par exemple, ST_Geometry) plutôt que de spécifier un sous-type ST_Geometry.

Syntaxe

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)
```

Type de retour

ST_MultiLineString

Exemple

Une table, mlines_test, est créée et un multipoint y est inséré à l'aide de la fonction ST_MultiLineString.

Oracle

```
CREATE TABLE mlines_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO MLINES_TEST VALUES (
 1910,
 sde.st_multilinestring ('multilinestring ((33 2, 34 3, 35 6), (28 4, 29 5, 31 8, 43
12), (39 3, 37 4, 36 7))', 4326)
);
```

PostgreSQL

```
CREATE TABLE mlines_test (
```

```
 id integer,
 geometry sde.st_geometry
);

INSERT INTO mlines_test VALUES (
 1910,
 sde.st_multilinestring ('multilinestring ((33 2, 34 3, 35 6), (28 4, 29 5, 31 8, 43
12), (39 3, 37 4, 36 7))', 4326)
);
```

SQLite

```
CREATE TABLE mlines_test (
 id integer
);

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

INSERT INTO mlines_test VALUES (
 1910,
 st_multilinestring ('multilinestring ((33 2, 34 3, 35 6), (28 4, 29 5, 31 8, 43 12),
(39 3, 37 4, 36 7))', 4326)
);
```

ST_MultiPoint

Définition

La fonction ST_MultiPoint crée une entité multi-polygones à partir d'une représentation textuelle connue.

Remarque :
Lors de la création de tables spatiales qui seront utilisées avec ArcGIS, il est recommandé de créer la colonne en tant que supertype de la géométrie (par exemple, ST_Geometry) plutôt que de spécifier un sous-type ST_Geometry.

Syntaxe

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)
```

Type de retour

ST_MultiPoint

Exemple

Une table, mpoint_test, est créée et un multipoint y est inséré à l'aide de la fonction ST_MultiPoint.

Oracle

```
CREATE TABLE mpoint_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO MPOINT_TEST VALUES (
 1110,
 sde.st_multipoint ('multipoint ((1 2), (3 4), (5 6))', 4326)
);
```

PostgreSQL

```
CREATE TABLE mpoint_test (
 id integer,
```

```
 geometry sde.st_geometry
);

INSERT INTO mpoint_test VALUES (
 1110,
 sde.st_multipoint ('multipoint (1 2, 3 4, 5 6)', 4326)
);
```

SQLite

```
CREATE TABLE mpoint_test (
 id integer
);

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

INSERT INTO mpoint_test VALUES (
 1110,
 st_multipoint ('multipoint ((1 2), (3 4), (5 6))', 4326)
);
```

ST_MultiPolygon

Définition

ST_MultiPolygon crée une entité multi-polygones à partir d'une représentation textuelle connue.

Remarque :
Lors de la création de tables spatiales qui seront utilisées avec ArcGIS, il est recommandé de créer la colonne en tant que supertype de la géométrie (par exemple, ST_Geometry) plutôt que de spécifier un sous-type ST_Geometry.

Syntaxe

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)
```

Type de retour

ST_MultiPolygon

Exemple

Une table, mpoly_test, est créée et un multipolygone y est inséré à l'aide de la fonction ST_MultiPolygon.

Oracle

```
CREATE TABLE mpoly_test (
 id integer,
 geometry sde.st_geometry
);

INSERT INTO MPOLY_TEST VALUES (
 1110,
 sde.st_multipolygon ('multipolygon (((3 3, 4 6, 5 3, 3 3),(8 24, 9 25, 1 28, 8 24),
(13 33, 7 36, 1 40, 10 43, 13 33)))', 4326)
);
```

PostgreSQL

```
CREATE TABLE mpoly_test (
```

```
 id integer,
 geometry sde.st_geometry
);

INSERT INTO mpoly_test VALUES (
 1110,
 sde.st_multipolygon ('multipolygon (((3 3, 4 6, 5 3, 3 3), (8 24, 9 25, 1 28, 8 24),
(13 33, 7 36, 1 40, 10 43, 13 33)))', 4326)
);
```

SQLite

```
CREATE TABLE mpoly_test (
 id integer
);

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

INSERT INTO mpoly_test VALUES (
 1110,
 st_multipolygon ('multipolygon (((3 3, 4 6, 5 3, 3 3), (8 24, 9 25, 1 28, 8 24), (13
33, 7 36, 1 40, 10 43, 13 33)))', 4326)
);
```

ST_MultiSurface

Remarque :
Oracle uniquement

Définition

La fonction ST_MultiSurface crée une entité multisurface à partir d'une représentation textuelle connue.

Syntaxe

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

Type de retour

ST_MultiSurface

Exemple

```
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

Définition

ST_NumGeometries prend un ensemble de géométries et renvoie le nombre de géométries qu'il contient.

Syntaxe

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)
```

Type de retour

Entier

Exemple

Dans l'exemple ci-après, une table nommée sample_numgeom est créée. Un multipolygone et un multi-point sont insérés dedans. Dans l'instruction SELECT, la fonction ST_NumGeometries est utilisée pour déterminer le nombre de géométries (ou entités) dans chaque géométrie.

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

Définition

ST_NumInteriorRing accepte un objet ST_Polygon et retourne le nombre de ses boucles intérieures.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_numinteriorring (polygon1 geometryblob)
```

Type de retour

Entier

Exemple

Une ornithologue souhaite étudier une population d'oiseaux sur plusieurs îles des mers du sud. Elle souhaite identifier les îles possédant un ou plusieurs lacs, car les espèces d'oiseaux auxquelles elle s'intéresse se nourrissent exclusivement dans les lacs d'eau douce.

Les colonnes ID et name de la table islands identifient chaque île, tandis que la colonne ST_Polygon land stocke la géométrie des îles.

Comme les boucles intérieures représentent les lacs, l'instruction SELECT qui comprend la fonction ST_NumInteriorRing répertorie uniquement les îles possédant au moins une boucle intérieure.

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

Définition

ST_NumPoints retourne le nombre de points (sommets) contenus dans une géométrie.

Pour les polygones, les sommets de début et de fin sont comptés, bien qu'ils occupent le même emplacement.

Notez que ce nombre est différent de l'attribut NUMPTS du type ST_Geometry. L'attribut NUMPTS contient le nombre des sommets présents dans toutes les parties de la géométrie, y compris les séparateurs qui se trouvent entre les parties. Un séparateur se trouve entre chaque partie. Par exemple, un objet multi-parties linestring contenant trois parties a deux séparateurs. Dans l'attribut NUMPTS, chaque séparateur est compté comme un sommet. Inversement, la fonction ST_NumPoints n'inclut pas les séparateurs dans le compte des sommets.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_numpoints (geometry1 geometryblob)
```

Type de retour

Entier

Exemple

La table numpoints_test est créée avec la colonne Geotype, qui contient le type de géométrie stocké dans la colonne g1.

Les instructions INSERT insèrent un point, un objet linestring et un polygone.

La requête SELECT utilise la fonction ST_NumPoints pour calculer le nombre de points présents dans chaque entité pour chaque type d'entité.

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

Remarque :
Oracle et PostgreSQL uniquement

Définition

ST_OrderingEquals compare deux objets ST_Geometry et renvoie 1 (Oracle) ou t (PostgreSQL) si les géométries sont égales et si les coordonnées sont dans le même ordre. Sinon, la fonction renvoie 0 (Oracle) ou f (PostgreSQL).

Syntaxe

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

Type de retour

Booléen

Exemple

Oracle

L'instruction `CREATE TABLE` ci-dessous crée la table LINESTRING_TEST, qui comporte deux colonnes d'objets linestring, ln1 et ln2.

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

L'instruction `INSERT` ci-dessous insère deux valeurs ST_LineString dans ln1 et ln2 qui sont égales et dont les coordonnées ont le même ordre.

```
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)
);
```

L'instruction `SELECT` ci-dessous et le jeu de résultats correspondant montrent comment la fonction ST_Equals renvoie 1 (true), quel que soit l'ordre des coordonnées. La fonction ST_OrderingEquals renvoie 0 (false) si les géométries ne sont pas égales et si les coordonnées ont le même ordre.

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

lid Equals     OrderingEquals

1   1          0
```

PostgreSQL

L'instruction `CREATE TABLE` ci-dessous crée la table LINESTRING_TEST, qui comporte deux colonnes d'objets linestring, ln1 et ln2.

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

L'instruction `INSERT` ci-dessous insère deux valeurs ST_LineString dans ln1 et ln2 qui sont égales et dont les coordonnées ont le même ordre.

```
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)
);
```

L'instruction `SELECT` ci-dessous et le jeu de résultats correspondant montrent comment la fonction ST_Equals renvoie 1 (true), quel que soit l'ordre des coordonnées. La fonction ST_OrderingEquals renvoie f (false) si les géométries ne sont pas égales et si les coordonnées ont le même ordre.

```
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

Définition

ST_Overlaps accepte deux objets géométrie et retourne 1 (Oracle et SQLite) ou t (PostgreSQL) si l'intersection des objets résulte en un objet géométrie de la même dimension, mais différent des objets source ; dans le cas contraire, la fonction retourne 0 (Oracle et SQLite) ou t (PostgreSQL).

Syntaxe

Oracle et PostgreSQL

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

SQLite

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

Type de retour

Booléen

Exemple

Le responsable de l'aménagement du territoire a besoin d'une liste des zones sensibles qui sont superposées au rayon de la zone tampon des sites de dépôt de déchets dangereux. La table sensitive_areas contient plusieurs colonnes qui décrivent les institutions menacées en plus de la colonne shape qui stocke les géométries ST_Polygon des institutions.

La table hazardous_sites stocke l'identité des sites dans la colonne id, tandis que l'emplacement géographique réel de chaque site est stocké dans la colonne des points de site.

Les tables sensitive_areas et hazardous_sites sont jointes par la fonction ST_Overlaps qui renvoie l'ID de toutes les lignes de la table sensitive_areas qui contiennent des polygones superposés au rayon bufférisé des points de la table hazardous_sites.

Oracle

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

CREATE TABLE hazardous_sites (
 id integer,
 site sde.st_geometry
);


INSERT INTO sensitive_areas VALUES (
 1,
 sde.st_geometry ('polygon ((.20 .30, .30 .30, .30 .40, .20 .40, .20 .30))', 4326)
);
```

```
INSERT INTO sensitive_areas VALUES (
 2,
 sde.st_geometry ('polygon ((.30 .30, .30 .50, .50 .50, .50 .30, .30 .30))', 4326)
);

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

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

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

```
SELECT UNIQUE (hs.id)
 FROM HAZARDOUS_SITES hs, SENSITIVE_AREAS sa
 WHERE sde.st_overlaps (sde.st_buffer (hs.site, .001), sa.shape) = 1;

ID

4
5
```

PostgreSQL

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

CREATE TABLE hazardous_sites (
 id serial,
 site sde.st_geometry
);


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

INSERT INTO sensitive_areas (shape) VALUES (
 sde.st_geometry ('polygon ((.30 .30, .30 .50, .50 .50, .50 .30, .30 .30))', 4326)
);

INSERT INTO sensitive_areas (shape) VALUES (
 sde.st_geometry ('polygon ((.40 .40, .40 .60, .60 .60, .60 .40, .40 .40))', 4326)
);

INSERT INTO hazardous_sites (site) VALUES (
 sde.st_geometry ('point (.60 .60)', 4326)
);
```

```
INSERT INTO hazardous_sites (site) VALUES (
 sde.st_geometry ('point (.30 .30)', 4326)
);
```

```
SELECT DISTINCT (hs.id) AS "Hazardous Site ID"
 FROM hazardous_sites hs, sensitive_areas sa
 WHERE sde.st_overlaps (sde.st_buffer (hs.site, .001), sa.shape) = 't';

id

1
2
```

SQLite

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

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

CREATE TABLE hazardous_sites (
 id integer primary key autoincrement not null,
 site_name varchar(30)
);

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

INSERT INTO sensitive_areas (shape) VALUES (
 st_geometry ('polygon ((.20 .30, .30 .30, .30 .40, .20 .40, .20 .30))', 4326)
);

INSERT INTO sensitive_areas (shape) VALUES (
 st_geometry ('polygon ((.30 .30, .30 .50, .50 .50, .50 .30, .30 .30))', 4326)
);

INSERT INTO sensitive_areas (shape) VALUES (
 st_geometry ('polygon ((.40 .40, .40 .60, .60 .60, .60 .40, .40 .40))', 4326)
);

INSERT INTO hazardous_sites (site_name, site) VALUES (
 'Kemlabs',
 st_geometry ('point (.60 .60)', 4326)
```

```
);

INSERT INTO hazardous_sites (site_name, site) VALUES (
 'Medi-Waste',
 st_geometry ('point (.30 .30)', 4326)
);
```

```
SELECT DISTINCT (hs.site_name) AS "Hazardous Site"
 FROM hazardous_sites hs, sensitive_areas sa
 WHERE st_overlaps (st_buffer (hs.site, .001), sa.shape) = 1;

Hazardous Site

Kemlabs
Medi-Waste
```

ST_Perimeter

Définition

ST_Perimeter renvoie la longueur de la ligne continue qui forme la frontière d'un polygone fermé ou d'une entité multisurfacique.

Cette fonction est une nouveauté de la version 10.8.1.

Syntaxe

Les deux options principales de chaque section renvoient le périmètre dans les unités du système de coordonnées défini pour l'entité. Les deux options secondaires vous permettent de spécifier l'unité de mesure linéaire. Pour obtenir une liste des valeurs prises en charge pour `linear_unit_name`, reportez-vous à ST_Distance.

Oracle et 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)
```

Type de retour

Double précision

Exemples

Oracle

Dans l'exemple suivant, un écologiste qui étudie les oiseaux du littoral a besoin de déterminer la longueur du littoral pour les lacs d'une zone déterminée. Les lacs sont mémorisés comme des polygones dans la table waterbodies. Une instruction `SELECT` avec la fonction ST_Perimeter est utilisée pour renvoyer le périmètre de chaque lac (entité) dans la table waterbodies.

```
--Create table named waterbodies
CREATE TABLE waterbodies (wbid INTEGER not null, waterbody sde.st_geometry);
--Insert a polygon feature to the waterbodies table
INSERT INTO waterbodies VALUES (
  1,
  sde.ST_Polygon ('polygon ((0 0, 0 4, 5 4, 5 0, 0 0))', 1)
);
--Find the perimeter of the polygon
```

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

L'instruction SELECT renvoie les données suivantes :

```
ID PERIMETER
1 +1.8000000
```

Dans l'exemple suivant, vous allez créer une table intitulée bfp, insérer trois entités, et calculer le périmètre de chaque entité en unités de mesure linéaire :

```
--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;
```

L'instruction SELECT renvoie le périmètre de chaque entité dans trois unités :

```
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

Dans l'exemple suivant, un écologiste qui étudie les oiseaux du littoral a besoin de déterminer la longueur du littoral pour les lacs d'une zone déterminée. Les lacs sont mémorisés comme des polygones dans la table waterbodies. Une instruction SELECT avec la fonction ST_Perimeter est utilisée pour renvoyer le périmètre de chaque lac (entité) dans la table waterbodies.

```
--Create table named waterbodies
CREATE TABLE waterbodies (wbid INTEGER not null, waterbody sde.st_geometry);
--Insert a polygon feature to the waterbodies table
```

```
INSERT INTO waterbodies VALUES (
  1,
  sde.ST_Polygon ('polygon ((0 0, 0 4, 5 4, 5 0, 0 0))', 1)
);
--Find the perimeter of the polygon
SELECT sde.ST_Perimeter (waterbody)
  FROM waterbodies;
```

L'instruction SELECT renvoie les données suivantes :

```
ID PERIMETER
1 +1.8000000
```

Dans l'exemple suivant, vous allez créer une table intitulée bfp, insérer trois entités, et calculer le périmètre de chaque entité en unités de mesure linéaire :

```
--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;
```

L'instruction SELECT renvoie le périmètre de chaque entité dans trois unités :

```
 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

Dans l'exemple suivant, un écologiste qui étudie les oiseaux du littoral a besoin de déterminer la longueur du littoral pour les lacs d'une zone déterminée. Les lacs sont mémorisés comme des polygones dans la table waterbodies. Une instruction SELECT avec la fonction ST_Perimeter est utilisée pour renvoyer le périmètre de chaque lac (entité) dans la table waterbodies.

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

```
CREATE TABLE waterbodies (wbid integer primary key autoincrement not null
);
SELECT AddGeometryColumn(
 NULL,
 'waterbodies',
 'waterbody',
 4326,
 'polygon',
 'xy',
 'null'
);
--Insert a polygon feature to the waterbodies table
INSERT INTO waterbodies VALUES (
  1,
  ST_Polygon ('polygon ((0 0, 0 4, 5 4, 5 0, 0 0))', 1)
);
--Find the perimeter of the polygon
SELECT ST_Perimeter (waterbody)
  FROM waterbodies;
```

L'instruction SELECT renvoie les données suivantes :

```
ID PERIMETER
1 +1.8000000
```

Dans l'exemple suivant, vous allez créer une table intitulée bfp, insérer trois entités, et calculer le périmètre de chaque entité en unités de mesure linéaire :

```
--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;
```

L'instruction SELECT renvoie le périmètre de chaque entité dans trois unités :

```
 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

Définition

La fonction ST_Point accepte un objet texte connu ou des coordonnées et un identifiant de référence spatiale et renvoie un objet ST_Point.

Remarque :
Lors de la création de tables spatiales qui seront utilisées avec ArcGIS, il est recommandé de créer la colonne en tant que supertype de la géométrie (par exemple, ST_Geometry) plutôt que de spécifier un sous-type ST_Geometry.

Syntaxe

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)
```

Type de retour

ST_Point

Exemple

L'instruction `CREATE TABLE` suivante crée la table point_test, qui comporte une colonne de points unique, PT1.

La fonction ST_Point convertit les coordonnées de points en une géométrie ST_Point avant d'être insérée dans la colonne pt1.

Oracle

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

```
INSERT INTO point_test VALUES (
 sde.st_point (10.01, 20.03, 4326)
);
```

PostgreSQL

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

```
INSERT INTO point_test VALUES (
 sde.st_point (10.01, 20.03, 4326)
);
```

SQLite

```
CREATE TABLE point_test (id integer);

SELECT AddGeometryColumn(
 NULL,
 'point_test',
 'pt1',
 4326,
 'point',
 'xy',
 'null'
);
```

```
INSERT INTO point_test VALUES (
 1,
 st_point (10.01, 20.03, 4326)
);
```

ST_PointFromText

Remarque :
Utilisée dans Oracle et SQLite uniquement ; pour PostgreSQL, utilisez ST_Point.

Définition

ST_PointFromText prend une représentation textuelle connue de type point et un ID de référence spatiale, puis renvoie un point.

Syntaxe

Oracle

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

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

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

SQLite

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

```
st_pointfromtext (wkt text)
```

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

Type de retour

ST_Point

Exemple

La table point_test est créée avec la seule colonne pt1 de ST_Point.

La fonction ST_PointFromText convertit les coordonnées textuelles du point en format de point avant que l'instruction INSERT n'insère le point dans la colonne pt1.

Oracle

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

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

SQLite

```
CREATE TABLE pt_test (id integer);
SELECT AddGeometryColumn(
 NULL,
 'pt_test',
 'pt1',
 4326,
 'point',
 'xy',
 'null'
);
```

```
INSERT INTO pt_test VALUES (
 1,
 st_pointfromtext ('point (10.01 20.03)', 4326)
);
```

ST_PointFromWKB

Définition

ST_PointFromWKB accepte une représentation binaire connue (WKB) et un ID de référence spatiale pour renvoyer un objet ST_Point.

Syntaxe

Oracle

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

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

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

PostgreSQL

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

SQLite

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

```
st_pointfromwkb (wkb blob)
```

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

Type de retour

ST_Point

Exemple

Cet exemple illustre comment ST_PointFromWKB peut permettre de créer un point à partir de sa représentation binaire connue. Les géométries sont des points dans le système de référence spatiale 4326. Dans cet exemple, les points sont stockés dans la colonne géométrie de la table sample_points, et la colonne wkb est mise à jour avec leur représentation binaire connue (avec la fonction ST_AsBinary). Enfin, la fonction ST_PointFromWKB permet de renvoyer les points depuis la colonne WKB. La table sample_points comporte une colonne geometry, dans laquelle les points sont stockés, et une colonne wkb dans laquelle les représentations binaires connues des points sont stockées.

Dans l'instruction SELECT, la fonction ST_PointFromWKB permet de récupérer les points de la colonne WKB.

Oracle

```
CREATE TABLE sample_points (
 id integer,
 geometry sde.st_point,
 wkb blob
);
INSERT INTO SAMPLE_POINTS (id, geometry) VALUES (
 10,
 sde.st_point ('point (44 14)', 4326)
);
INSERT INTO SAMPLE_POINTS (id, geometry) VALUES (
 11,
 sde.st_point ('point (24 13)', 4326)
);
UPDATE SAMPLE_POINTS
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 10;
UPDATE SAMPLE_POINTS
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 11;
```

```
SELECT id, sde.st_astext (sde.st_pointfromwkb(wkb, 4326)) POINTS
 FROM SAMPLE_POINTS;
ID POINTS
10 POINT (44.00000000 14.00000000)
11 POINT (24.00000000 13.00000000)
```

PostgreSQL

```
CREATE TABLE sample_points (
 id integer,
 geometry sde.st_point,
 wkb bytea
);
INSERT INTO sample_points (id, geometry) VALUES (
 10,
 sde.st_point ('point (44 14)', 4326)
);
INSERT INTO sample_points (id, geometry) VALUES (
 11,
 sde.st_point ('point (24 13)', 4326)
);
UPDATE sample_points
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 10;
UPDATE sample_points
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 11;
```

```
SELECT id, sde.st_astext (sde.st_pointfromwkb(wkb, 4326))
 AS points
 FROM sample_points;
id points
10 POINT (44 14)
11 POINT (24 13)
```

SQLite

```
CREATE TABLE sample_pts (
 id integer,
 wkb blob
);
SELECT AddGeometryColumn(
 NULL,
 'sample_pts',
 'geometry',
 4326,
 'point',
 'xy',
 'null'
);
INSERT INTO sample_pts (id, geometry) VALUES (
 10,
 st_point ('point (44 14)', 4326)
);
INSERT INTO sample_pts (id, geometry) VALUES (
 11,
 st_point ('point (24 13)', 4326)
);
UPDATE sample_pts
 SET wkb = st_asbinary (geometry)
 WHERE id = 10;
UPDATE sample_pts
 SET wkb = st_asbinary (geometry)
 WHERE id = 11;
```

```
SELECT id, st_astext (st_pointfromwkb(wkb, 4326))
 AS "points"
 FROM sample_pts;
id points
10 POINT (44.00000000 14.00000000)
11 POINT (24.00000000 13.00000000)
```

ST_PointN

Définition

ST_PointN prend un ST_LineString et un index de nombre entier, et renvoie un point qui est le énième sommet dans le chemin du ST_LineString.

Syntaxe

Oracle et PostgreSQL

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

SQLite

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

Type de retour

ST_Point

Exemple

La table pointn_test est créée avec la colonne gid qui identifie chaque ligne de façon unique, et la colonne ln1 ST_LineString. L'instruction INSERT insère deux valeurs linestring. Le premier objet linestring ne possède pas de coordonnée z ni de mesure, alors que le deuxième objet linestring possède les deux.

La requête SELECT utilise les fonctions ST_PointN et ST_AsText pour renvoyer le texte connu du deuxième sommet de chaque objet linestring.

Oracle

```
CREATE TABLE pointn_test (
 gid integer,
 ln1 sde.st_geometry
);


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

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

```
SELECT gid, sde.st_astext (sde.st_pointn (ln1, 2)) The_2ndvertex
 FROM POINTN_TEST;
```

```
GID The_2ndvertex

1   POINT (23.73 21.92)
2   POINT ZM (23.73 21.92 6.5 7.1)
```

PostgreSQL

```
CREATE TABLE pointn_test (
 gid serial,
 ln1 sde.st_geometry
);


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

INSERT INTO pointn_test (ln1) VALUES (
 sde.st_linestring ('linestring zm(10.02 20.01 5.0 7.0, 23.73 21.92 6.5 7.1, 30.10
40.23 6.9 7.2)', 4326)
);
```

```
SELECT gid, sde.st_astext (sde.st_pointn (ln1, 2))
 AS The_2ndvertex
 FROM pointn_test;

gid the_2ndvertex

1   POINT (23.73 21.92)
2   POINT ZM (23.73 21.92 6.5 7.1)
```

SQLite

```
CREATE TABLE pointn_test (
 gid integer primary key autoincrement not null
);

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

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

INSERT INTO pointn_test (ln1) VALUES (
 st_linestring ('linestring zm(10.02 20.01 5.0 7.0, 23.73 21.92 6.5 7.1, 30.10 40.23
6.9 7.2)', 4326)
);
```

```
SELECT gid, st_astext (st_pointn (ln1, 2))
 AS "Second Vertex"
 FROM pointn_test;

gid  Second Vertex

1   POINT ( 23.73000000 21.92000000)
2   POINT ZM ( 23.73000000 21.92000000 6.50000000 7.10000000)
```

ST_PointOnSurface

Définition

ST_PointOnSurface prend un ST_Polygon ou un ST_MultiPolygon et renvoie un ST_Point qui doit nécessairement résider sur sa surface.

Syntaxe

Oracle et PostgreSQL

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

SQLite

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

Type de retour

ST_Point

Exemple

L'ingénieur municipal souhaite créer un point label pour l'emprise de chaque bâtiment historique. Les emprises des bâtiments historiques sont stockées dans la table hbuildings qui a été créée avec l'instruction CREATE TABLE suivante :

La fonction ST_PointOnSurface génère un point qui doit nécessairement résider sur la surface des emprises de bâtiments. La fonction ST_PointOnSurface renvoie un point que la fonction ST_AsText convertit en texte pour être utilisé par l'application.

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

Remarque :
Oracle et SQLite uniquement

Définition

ST_PolyFromText accepte une représentation textuelle connue et un ID de référence spatiale, puis renvoie un objet ST_Polygon.

Syntaxe

Oracle

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

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

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

SQLite

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

```
st_polyfromtext (wkt text)
```

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

Type de retour

ST_Polygon

Exemple

La table polygon_test est créée avec l'unique colonne de polygone.

L'instruction INSERT insère un polygone dans la colonne de polygone à l'aide de la fonction ST_PolyFromText.

Oracle

```
CREATE TABLE polygon_test (pl1 sde.st_geometry);
```

```
INSERT INTO polygon_test VALUES (
 sde.st_polyfromtext ('polygon ((10.01 20.03, 10.52 40.11, 30.29 41.56, 31.78 10.74,
10.01 20.03))', 4326)
);
```

SQLite

```
CREATE TABLE polygon_test (id integer);
SELECT AddGeometryColumn(
 NULL,
 'polygon_test',
 'pl1',
 4326,
 'polygon',
 'xy',
 'null'
);
```

```
INSERT INTO polygon_test VALUES (
 1,
 st_polyfromtext ('polygon ((10.01 20.03, 10.52 40.11, 30.29 41.56, 31.78 10.74, 10.01
20.03))', 4326)
);
```

ST_PolyFromWKB

Définition

ST_PolyFromWKB accepte une représentation binaire connue (WKB) et un ID de référence spatiale, puis renvoie un objet ST_Polygon.

Syntaxe

Oracle

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

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

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

PostgreSQL

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

SQLite

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

```
st_polyfromwkb (wkb blob)
```

Si aucun identifiant de référence spatiale n'est spécifié, la référence spatiale par défaut est 4326.

Type de retour

ST_Polygon

Exemple

Cet exemple illustre la façon dont ST_PolyFromWKB permet de créer un polygone à partir de sa représentation binaire connue. La géométrie est un polygone dans le système de référence spatiale 4326. Dans cet exemple, le polygone est stocké avec ID = 1115 dans la colonne géométrie de la table sample_polys, et la colonne wkb est mise à jour avec sa représentation WKB (avec la fonction ST_AsBinary). Enfin, la fonction ST_PolyFromWKB permet de renvoyer l'objet multisurfacique depuis la colonne WKB. La table sample_polys comporte une colonne geometry, dans laquelle le polygone est stocké, et une colonne wkb dans laquelle la représentation WKB du polygone est stockée.

Dans l'instruction SELECT, la fonction ST_PointFromWKB permet de récupérer les points de la colonne WKB.

Oracle

```
CREATE TABLE sample_polys (
 id integer,
 geometry sde.st_geometry,
 wkb blob
);
INSERT INTO SAMPLE_POLYS (id, geometry) VALUES (
 1115,
 sde.st_polyfromtext ('polygon ((10.01 20.03, 10.52 40.11, 30.29 41.56, 31.78 10.74,
10.01 20.03))', 4326)
);
UPDATE SAMPLE_POLYS
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 1115;
```

```
SELECT id, sde.st_astext (sde.st_polyfromwkb (wkb, 4326)) POLYS
 FROM SAMPLE_POLYS;
ID            POLYS
1115    POLYGON (10.01000000 20.03000000, 31.78000000 10.74000000, 30.29000000
41.56000000, 10.52000000 40.11000000, 10.01000000 20.03000000)
```

PostgreSQL

```
CREATE TABLE sample_polys (
 id integer,
 geometry sde.st_geometry,
 wkb bytea
);
INSERT INTO sample_polys (id, geometry) VALUES (
 1115,
 sde.st_polygon ('polygon  ((10.01 20.03, 10.52 40.11, 30.29 41.56, 31.78 10.74, 10.01
20.03))', 4326)
);
UPDATE sample_polys
 SET wkb = sde.st_asbinary (geometry)
 WHERE id = 1115;
```

```
SELECT id, sde.st_astext (sde.st_polyfromwkb (wkb, 4326))
 AS POLYS
 FROM sample_polys;
id    polys
1115    POLYGON (10.01000000 20.03000000, 31.78000000 10.74000000, 30.29000000
41.56000000, 10.52000000 40.11000000, 10.01000000 20.03000000)
```

SQLite

```
CREATE TABLE sample_polys(
 id integer,
 wkb blob
);
SELECT AddGeometryColumn(
 NULL,
 'sample_polys',
 'geometry',
```

```
 4326,
 'polygon',
 'xy',
 'null'
);
INSERT INTO sample_polys (id, geometry) VALUES (
 1115,
 st_polyfromtext ('polygon ((10.01 20.03, 10.52 40.11, 30.29 41.56, 31.78 10.74, 10.01
20.03))', 4326)
);
UPDATE sample_polys
 SET wkb = st_asbinary (geometry)
 WHERE id = 1115;
```

```
SELECT id, st_astext (st_polyfromwkb (wkb, 4326))
 AS "polygons"
 FROM sample_polys;
id    polygons
1115    POLYGON (10.01000000 20.03000000, 31.78000000 10.74000000, 30.29000000
41.56000000, 10.52000000 40.11000000, 10.01000000 20.03000000)
```

ST_Polygon

Définition

La fonction accesseur ST_Polygon accepte une représentation textuelle (WKT) connue et un identifiant de référence spatiale (SRID) et génère un objet ST_Polygon.

Remarque :
Lors de la création de tables spatiales qui seront utilisées avec ArcGIS, il est recommandé de créer la colonne en tant que supertype de la géométrie (par exemple, ST_Geometry) plutôt que de spécifier un sous-type ST_Geometry.

Syntaxe

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)
```

Type de retour

ST_Polygon

Exemple

L'instruction `CREATE TABLE` suivante crée les tables polygon_test, qui comportent une colonne unique, p1. L'instruction `INSERT` ultérieure convertit un anneau (un polygone à la fois fermé et simple) en un objet ST_Polygon et l'insère dans la colonne p1 à l'aide de la fonction ST_Polygon.

Oracle

```
CREATE TABLE polygon_test (p1 sde.st_geometry);

INSERT INTO polygon_test VALUES (
 sde.st_polygon ('polygon ((10.01 20.03, 20.94 21.34, 35.93 10.04, 10.01 20.03))', 4326)
);
```

PostgreSQL

```
CREATE TABLE polygon_test (p1 sde.st_geometry);
```

```
INSERT INTO polygon_test VALUES (
 sde.st_polygon ('polygon ((10.01 20.03, 20.94 21.34, 35.93 10.04, 10.01 20.03))', 4326)
);
```

SQLite

```
CREATE TABLE poly_test (id integerp1 geometryblob);

SELECT AddGeometryColumn(
 NULL,
 'poly_test',
 'p1',
 4326,
 'polygon',
 'xy',
 'null'
);

INSERT INTO poly_test VALUES (
 1,
 st_polygon ('polygon ((10.01 20.03, 20.94 21.34, 35.93 10.04, 10.01 20.03))', 4326)
);
```

ST_Relate

Définition

La fonction ST_Relate compare deux géométries et renvoie 1 (Oracle et SQLite) ou t (PostgreSQL) si les géométries remplissent les conditions spécifiées par la chaîne de matrice modèle DE-9IM ; dans le cas contraire, 0 (Oracle et SQLite) ou f (PostgreSQL) est renvoyé.

Une deuxième option est disponible lors de l'utilisation de ST_Relate dans SQLite et Oracle : vous pouvez comparer deux géométries pour renvoyer une chaîne représentant la matrice modèle DE-9IM qui définit la relation des géométries entre elles.

Syntaxe

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)
```

Type de retour

Une valeur booléenne est renvoyée pour PostgreSQL.

L'option 1 pour SQLite et Oracle renvoie un nombre entier.

L'option 2 pour SQLite et Oracle renvoie une chaîne.

Exemples

Une matrice modèle DE-9IM est un périphérique permettant de comparer des géométries. Il en existe plusieurs types. Par exemple, vous pouvez utiliser la fonction ST_Relate et la matrice modèle d'égalité (T*F**FFF*) pour savoir si deux géométries sont égales, mais vous pouvez également fournir le modèle DE-9IM (1*F**FFF*). Avec le dernier modèle, la fonction ST_Relate indique si deux géométries sont égales avec la première position, ce qui indique si les intérieurs de l'intersection des deux géométries sont une ligne (dimension de 1).

Dans les exemples ci-dessous, une table, relate_test, est créée avec trois colonnes spatiales et des entités ponctuelles sont insérées dans chacune d'entre elles. La fonction ST_Relate est utilisée dans l'instruction `SELECT` pour savoir si les points sont égaux.

Pour déterminer si les géométries sont égales et si vous n'avez pas besoin de trouver la dimensionnalité de la relation, utilisez à la place la fonction ST_Equals.

Oracle

Le premier exemple illustre la première option ST_Relate, qui compare les géométries en fonction d'une matrice modèle DE-9IM pour renvoyer 1 si les géométries répondent aux critères définis dans la matrice ou 0 si les géométries n'y répondent pas.

```
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;
```

Il renvoie ce qui suit :

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

1               0               0
```

Cet exemple montre la seconde option. Il compare deux géométries et renvoie la matrice modèle DE-9IM.

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

Il renvoie ce qui suit :

```
g1 rel g2
0FFFFFFF2
```

PostgreSQL

L'exemple compare les géométries en fonction d'une matrice modèle DE-9IM pour renvoyer t si les géométries répondent aux critères définis dans la matrice ou f si les géométries n'y répondent pas.

```
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;
```

Il renvoie ce qui suit :

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

t          f         f
```

SQLite

Ce premier exemple illustre la première option ST_Relate, qui compare les géométries en fonction d'une matrice modèle DE-9IM pour renvoyer 1 si les géométries répondent aux critères définis dans la matrice ou 0 si les

géométries n'y répondent pas.

```
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;
```

Il renvoie ce qui suit :

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

1          0         0
```

Cet exemple montre la seconde option. Il compare deux géométries et renvoie la matrice modèle DE-9IM.

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

Il renvoie ce qui suit :

```
g1 rel g2
0FFFFFFF2
```

ST_SRID

Définition

ST_SRID accepte un objet géométrie et retourne son ID de référence spatiale.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_srid (geometry1 geometryblob)
```

Type de retour

Entier

Exemples

La table suivante est créée :

Dans l'instruction suivante, une géométrie de point située aux coordonnées (10.01, 50.76) est insérée dans la colonne de géométrie g1. Au moment de la création de la géométrie de point, la valeur SRID de 4326 lui est attribuée.

La fonction ST_SRID renvoie l'ID de référence spatiale de la géométrie qui vient d'être saisie.

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

Définition

ST_StartPoint retourne le premier point d'un objet linestring.

Syntaxe

Oracle et PostgreSQL

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

SQLite

```
st_startpoint (ln1 geometryblob)
```

Type de retour

ST_Point

Exemples

La table startpoint_test est créée avec la colonne gid integer, qui identifie de façon unique les enregistrements de la table, et la colonne ST_LineString ln1.

Les instructions INSERT insèrent ST_LineStrings dans la colonne ln1. Le premier ST_LineString ne possède pas de coordonnées z ou de mesures, alors que le deuxième ST_LineString possède les deux.

La fonction ST_StartPoint extrait le premier point de chaque ST_LineString. Le premier point de la liste ne possède pas de coordonnée z ou de mesure, alors que le deuxième point possède les deux, comme l'objet linestring source.

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

Remarque :
Oracle et SQLite uniquement

Définition

La fonction ST_Surface crée une entité de surface à partir d'une représentation textuelle connue. Les surfaces sont similaires aux polygones, mais elles sont dotées de valeurs au niveau de chaque point sur leur étendue.

Syntaxe

Oracle

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

SQLite

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

Type de retour

ST_Polygon

Exemple

Une table, surf_test, est créée et une géométrie de surface y est insérée.

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

Définition

ST_SymmetricDiff accepte deux objets géométrie et retourne un objet géométrie composé des parties des objets source qui ne sont pas communes aux deux.

Syntaxe

Oracle et PostgreSQL

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

SQLite

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

Type de retour

Oracle et PostgreSQL

ST_Geometry

SQLite

Geometryblob

Exemple

Pour un rapport spécial, le responsable de l'aménagement du territoire doit identifier les surfaces des bassins versants et des rayons dangereux qui ne s'intersectent pas.

La table des bassins versants (watershed) contient une colonne id, une colonne permettant de stocker le nom des bassins versants (wname) et une colonne shape qui stocke la géométrie des surfaces des bassins versants.

La table plumes stocke l'identité du site dans la colonne id, tandis que l'emplacement géographique réel de chaque site est stocké dans la colonne des points de site.

La fonction ST_Buffer génère une zone tampon qui entoure les points des sites de dépôt de déchets dangereux. La fonction ST_SymmetricDiff retourne les polygones des sites de dépôt de déchets dangereux bufférisés et des bassins versants qui ne s'intersectent pas.

La différence symétrique entre les sites de dépôt de déchets dangereux et les bassins versants a pour résultat la soustraction des zones qui s'intersectent.

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

Définition

ST_Touches retourne 1 (Oracle et SQLite) ou t (PostgreSQL) si aucun des points communs aux deux géométries n'intersecte les intérieurs des deux géométries ; dans le cas contraire, la fonction retourne 0 (Oracle et SQLite) ou f (PostgreSQL). Au moins une géométrie doit être un ST_LineString, ST_Polygon, ST_MultiLineString ou ST_MultiPolygon.

Syntaxe

Oracle et PostgreSQL

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

SQLite

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

Type de retour

Booléen

Exemple

Le technicien SIG a été chargé par son responsable de fournir une liste de toutes les canalisations d'égout dont les extrémités coupent une autre canalisation d'égout.

La table sewerlines est créée avec trois colonnes. La première colonne, sewer_id, identifie de façon unique chaque canalisation d'égout. La colonne de classe de nombre entier identifie le type de canalisation d'égout qui est en général associé à sa capacité. La colonne des canalisations d'égout stocke la géométrie de la canalisation.

La requête SELECT utilise la fonction ST_Touches pour renvoyer une liste des canalisations d'égout qui se touchent.

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

Définition

ST_Transform prend les données ST_Geometry bidimensionnelles comme entrée et renvoie les valeurs converties en une référence spatiale spécifiée par l'ID de référence spatiale (SRID) que vous fournissez.

⚠ Attention :
Si vous avez inscrit la colonne spatiale auprès de la base de données PostgreSQLà l'aide de la fonction st_register_spatial_column, le SRID au moment de l'inscription est écrit dans la table sde_geometry_columns. Si vous avez créé un index spatial dans la colonne spatiale d'une base de données Oracle, le SRID au moment de la création de l'index spatial est écrit dans la table st_geometry_columns. Si vous utilisez ST_Transform pour modifier le SRID des données ST_Geometry, le SRID n'est pas mis à jour dans la table sde_geometry_columns ou st_geometry_columns.

Lorsque les deux systèmes de coordonnées géographiques sont différents, ST-Transform effectue une transformation géographique. Une transformation géographique consiste à effectuer une conversion entre deux systèmes de coordonnées géographiques. Une transformation géographique est définie dans une direction spécifique, par exemple de NAD 1927 vers NAD 1983, mais la fonction ST_Transform appliquera correctement la transformation quels que soient les systèmes de coordonnées source et de destination.

Les méthodes de transformation géographiques peuvent être classées dans les deux catégories suivantes : basées sur des équations et basées sur des fichiers. Les méthodes basées sur des équations sont autonomes et ne nécessitent aucune information externe. Celles basées sur des fichiers font appel à des fichiers stockés sur le disque pour calculer les valeurs de décalage. Elles sont habituellement plus précises que les méthodes basées sur les équations. Les méthodes basées sur des fichiers sont habituellement utilisées en Australie, au Canada, en Allemagne, en Nouvelle-Zélande, en Espagne et aux Etats-Unis. Vous pouvez obtenir les fichiers (à l'exception de ceux pour le Canada) à partir d'une installation ArcGIS Pro ou directement auprès des diverses agences nationales de cartographie.

Pour prendre en charge les transformations basées sur des fichiers, les fichiers doivent se trouver sur le serveur où est installée la base de données dans la même structure relative de dossiers que dans le dossier pedata du répertoire d'installation d'ArcGIS Pro.

Par exemple, un dossier nommé `pedata` figure dans le dossier `Resources` du répertoire d'installation ArcGIS Pro. Le dossier `pedata` contient plusieurs sous-dossiers, mais les trois dossiers qui contiennent les méthodes basées sur des fichiers prises en charge sont `harn`, `nadcon` et `ntv2`. Copiez le dossier `pedata` et son contenu depuis le répertoire d'installation d'ArcGIS vers le serveur de base de données ou créez un répertoire sur le serveur de base de données qui contient les fichiers et sous-répertoires de méthode de transformation issus du fichier pris en charge. Une fois que les fichiers se trouvent sur le serveur de base de données, définissez une variable d'environnement de système d'exploitation appelée PEDATAHOME sur le même serveur. Définissez la variable PEDATAHOME sur l'emplacement du répertoire contenant les sous-répertoires et les fichiers. Par exemple, si le dossier pedata est copié dans `C:\pedata` sur un serveur Microsoft Windows, définissez la variable d'environnement PEDATAHOME sur `C:\pedata`.

Consultez la documentation qui accompagne votre système d'exploitation pour savoir comment définir une variable d'environnement.

Après avoir défini la variable d'environnement PEDATAHOME, vous devez lancer une nouvelle session SQL pour pouvoir utiliser la fonction ST_Transform. Le serveur n'a toutefois pas besoin d'être redémarré.

Utilisation de ST_Transform avec PostgreSQL

Dans PostgreSQL, vous pouvez procéder à des conversions de références spatiales qui présentent le même système de coordonnées géographiques ou des systèmes de coordonnées géographiques différents.

Si les données sont stockées dans une base de données (au lieu d'une géodatabase), procédez comme suit pour changer la référence spatiale des données ST_Geometry lorsque les systèmes de coordonnées géographiques sont identiques :

1. Créez une copie de sauvegarde de la table.
2. Créez une deuxième colonne ST_Geometry (de destination) dans la table.
3. Inscrivez la colonne ST_Geometry de destination en spécifiant le nouveau SRID.
   Cette opération définit la référence spatiale de la colonne en insérant un enregistrement dans la table système sde_geometry_columns.
4. Exécutez la fonction ST_Transform et faites en sorte que les données transformées soient une sortie de la colonne ST_Geometry de destination.
5. Annulez l'inscription de la première colonne ST_Geometry (source).

Si les données sont stockées dans une géodatabase, vous devez utiliser les outils ArcGIS pour reprojeter les données dans une nouvelle classe d'entités. L'exécution de ST_Transform dans une classe d'entités de géodatabase annule la fonction de mise à jour des tables système de géodatabase avec le nouveau SRID.

Utilisation de ST_Transform avec Oracle

Dans Oracle, vous pouvez procéder à des conversions de références spatiales qui présentent le même système de coordonnées géographiques ou des systèmes de coordonnées géographiques différents.

Si les données sont stockées dans une base de données (au lieu d'une géodatabase) et qu'aucun index spatial n'a été défini dans la colonne spatiale, vous pouvez ajouter une deuxième colonne ST_Geometry et y générer les données transformées. Vous pouvez conserver à la fois la colonne ST_Geometry d'origine (source) et la colonne ST_Geometry de destination dans la table, mais vous ne pouvez afficher qu'une seule colonne en même temps dans ArcGIS à l'aide d'une vue ou en modifiant la définition de couche de requête de la table.

Si les données sont stockées dans une base de données (au lieu d'une géodatabase) et qu'un index spatial a été défini dans la colonne spatiale, vous ne pouvez pas conserver la colonne ST_Geometry d'origine. Lorsqu'un index spatial a été défini dans une colonne ST_Geometry, le SRID est écrit dans la table de métadonnées st_geometry_columns. ST_Transform n'actualise pas cette table.

1. Créez une copie de sauvegarde de la table.
2. Créez une deuxième colonne ST_Geometry (de destination) dans la table.
3. Exécutez la fonction ST_Transform et faites en sorte que les données transformées soient une sortie de la colonne ST_Geometry de destination.
4. Retirez l'index spatial de la colonne ST_Geometry source.
5. Retirez la colonne ST_Geometry source.

6. Créez un index spatial dans la colonne ST_Geometry de destination.

Si les données sont stockées dans une géodatabase, vous devez utiliser les outils ArcGIS pour reprojeter les données dans une nouvelle classe d'entités. L'exécution de ST_Transform dans une classe d'entités de géodatabase annule la fonction de mise à jour des tables système de géodatabase avec le nouveau SRID.

Utilisation de ST_Transform avec SQLite

Dans SQLite, vous pouvez procéder à des conversions de références spatiales qui présentent le même système de coordonnées géographiques ou des systèmes de coordonnées géographiques différents.

Syntaxe

Les références spatiales source et cible ont le même système de coordonnées géographiques.

Oracle et PostgreSQL

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

SQLite

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

Les références spatiales source et cible n'ont pas le même système de coordonnées géographiques.

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])`

L'option 3 vous permet de spécifier l'étendue en tant que liste de coordonnées séparées par une virgule dans l'ordre suivant : coordonnée x en bas à gauche, coordonnée y en bas à gauche, coordonnée x en haut à droite, coordonnée y en haut à droite. Si vous ne précisez pas d'étendue, ST-Transform utilise une étendue plus générale et plus grande.

Lorsque vous spécifiez une étendue, le méridien principal et les paramètres du facteur de conversion unitaire sont en option. Vous ne devez fournir ces informations que si les valeurs d'étendue spécifiées n'utilisent pas le méridien de Greenwich ni les degrés décimaux.

SQLite

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

Type de retour

Oracle et PostgreSQL

ST_Geometry

SQLite

Geometryblob

Exemples

Transformation des données lorsque les références spatiales source et de destination ont le même système de coordonnées géographiques

L'exemple suivant crée une table transform_test qui comporte deux colonnes linestring : ln1 et ln2. Une ligne est insérée dans la colonne ln1 avec un SRID de 4326. La fonction ST_Transform est ensuite utilisée dans une instruction UPDATE pour accepter l'objet linestring de la colonne ln1, le convertir de la référence de coordonnées attribuée à SRID 4326 en référence de coordonnée attribuée à SRID 3857 et le placer dans la colonne ln2.

Remarque :
Les SRID 4326 et 3857 possèdent le même datum géographique.

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);
```

Transformer les données lorsque les références spatiales source et de destination n'ont pas le même système de coordonnées géographiques

Dans l'exemple suivant, la table n27 qui contient une colonne ID et une colonne geometry est créée. Un point est inséré dans la table n27 avec un SRID de 4267. LE SRID 4267 utilise le système de coordonnées géographiques NAD 1927.

Ensuite, la table n83 est créée et la fonction ST_Transform est utilisée pour insérer la géométrie de la table n27 dans la table n83, mais avec un SRID de 4269 et la transformation géographique ID 1241. Le SRID 4269 utilise le système de coordonnées géographiques NAD 1983 et 1241 est l'ID connu de la transformation NAD_1927_To_NAD_1983_NADCON. Cette transformation est basée sur des fichiers et peut être utilisée pour les 48 états des Etats-Unis.

Conseil :

Pour les listes des transformations géographiques prises en charge, reportez-vous à l'article technique d'Esri 000004829 ainsi qu'aux liens fournis dans la section **Informations associées** de l'article.

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
);
```

Si la variable PEDATAHOME est définie correctement, l'exécution d'une instruction SELECT sur la table n83 doit renvoyer le résultat suivant :

```
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

Définition

ST_Union renvoie un objet géométrie qui représente la combinaison de deux objets source.

Syntaxe

Oracle et PostgreSQL

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

SQLite

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

Type de retour

Oracle et PostgreSQL

ST_Geometry

SQLite

Geometryblob

Exemple

La table sensitive_areas stocke les ID des institutions menacées en plus de la colonne shape qui stocke les géométries de type polygone des institutions.

La table hazardous_sites stocke l'identité des sites dans la colonne id, tandis que l'emplacement géographique réel de chaque site est stocké dans la colonne des points de site.

La fonction ST_Buffer génère une zone tampon qui entoure les sites de dépôt de déchets dangereux. La fonction ST_Union génère des polygones à partir de l'union des polygones des sites de dépôt de déchets dangereux placés dans la zone tampon et des zones sensibles. La fonction ST_Area retourne la superficie de ces polygones.

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

Définition

ST_Within renvoie 1 (Oracle et SQLite) ou t (PostgreSQL) si le premier objet ST_Geometry est entièrement inclus dans le second. Dans le cas contraire, la fonction renvoie 0 (Oracle et SQLite) ou f (PostgreSQL).

Syntaxe

Oracle et PostgreSQL

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

SQLite

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

Type de retour

Booléen

Exemple

Dans l'exemple ci-dessous, deux tables sont créées : zones et squares. L'instruction SELECT recherche tous les carrés qui s'intersectent, mais qui ne se trouvent pas entièrement dans un terrain.

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

Définition

ST_X accepte un objet ST_Point comme paramètre en entrée et renvoie sa coordonnée x. Dans SQLite, ST_X peut également mettre à jour la coordonnée x d'une fonction ST_Point.

Syntaxe

Oracle et PostgreSQL

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

SQLite

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

Type de retour

Double précision

La fonction ST_X peut être utilisée avec SQLite pour mettre à jour la coordonnée x d'un point. Dans ce cas, un objet geometryblob est renvoyé.

Exemples

La table x_test est créée avec deux colonnes : la colonne gid qui identifie la ligne de façon unique, et la colonne des points pt1.

Les instructions INSERT insèrent deux lignes. L'une correspond à un point sans coordonnée z ou mesure. L'autre colonne possède une coordonnée z et une mesure.

La requête SELECT utilise la fonction ST_X pour calculer la coordonnée x de chaque entité ponctuelle.

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
```

La fonction ST_X peut également être utilisée pour mettre à jour la valeur de coordonnée d'un point existant. Dans cet exemple, la fonction ST_X est utilisée pour mettre à jour la valeur de coordonnée x du premier point dans x_test.

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

ST_Y

Définition

ST_Y accepte un objet ST_Point comme paramètre en entrée et renvoie sa coordonnée y. Dans SQLite, ST_Y peut également mettre à jour la coordonnée y d'une fonction ST_Point.

Syntaxe

Oracle et PostgreSQL

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

SQLite

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

Type de retour

Double précision

La fonction ST_Y peut être utilisée avec SQLite pour mettre à jour la coordonnée y d'un point. Dans ce cas, un objet geometryblob est renvoyé.

Exemple

La table y_test est créée avec deux colonnes : la colonne gid qui identifie la ligne de façon unique, et la colonne des points pt1.

Les instructions INSERT insèrent deux lignes. L'une correspond à un point sans coordonnée z ou mesure. L'autre possède à la fois une coordonnée z et une mesure.

La requête SELECT utilise la fonction ST_Y pour renvoyer la coordonnée y de chaque entité ponctuelle.

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
```

La fonction ST_Y peut également être utilisée pour mettre à jour la valeur de coordonnée d'un point existant. Dans cet exemple, la fonction ST_Y est utilisée pour mettre à jour la valeur de coordonnée y du deuxième point dans y_test.

```
UPDATE y_test
 SET pt1=st_y(
  (SELECT pt1 FROM y_test WHERE gid=2),
  20.1
  )
 WHERE gid=2;
```

ST_Z

Définition

ST_Z accepte un objet ST_Point comme paramètre en entrée et renvoie sa coordonnée z (altitude). Dans SQLite, ST_Z peut également mettre à jour la coordonnée z d'une fonction ST_Point.

Syntaxe

Oracle et PostgreSQL

```
sde.st_z (geometry1 sde.st_point)
```

SQLite

```
st_z (geometry  geometryblob)
st_z (input_shape geometryblob, new_Zvalue double)
```

Type de retour

Oracle

Nombre

PostgreSQL

Entier

SQLite

Une double précision est renvoyée lorsque la fonction ST_Z est utilisée pour renvoyer la coordonnée z d'un point. Un objet geometryblob est renvoyé lorsque la fonction ST_Z est utilisée pour mettre à jour la coordonnée z d'un point.

Exemple

La table z_test est créée avec deux colonnes : la colonne id qui identifie la ligne de façon unique et la colonne des points geometry. L'instruction INSERT insère un enregistrement dans la table z_test.

L'instruction SELECT répertorie la colonne id et la coordonnée z à double précision du point inséré à l'aide de l'instruction précédente.

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
```

La fonction ST_Z peut également être utilisée pour mettre à jour la valeur de coordonnée d'un point existant. Dans cet exemple, la fonction ST_Z est utilisée pour mettre à jour la valeur de coordonnée z du premier point dans z_test.

```
UPDATE z_test
 SET pt1=st_z(
  (SELECT pt1 FROM z_test where id=1), 32.04)
 WHERE id=1;
```