esri

ST_Geometry SQL 函数引用

THE SCIENCE OF WHERE

内容列表

与 ST_Geometry 配合使用的 SQL 函数

本参考文档提供了一个可与 Esri ST_Geometry 空间数据类型在 Oracle、PostgreSQL 和 SQLite 中配合使用的函数列表及其描述。

进行以下任意操作时，会创建 Esri ST_Geometry SQL 函数和类型：

- 在 Oracle 数据库中创建地理数据库。
- 在 PostgreSQL 数据库中创建地理数据库时使用 ST_Geometry。
- 在 Oracle 或 PostgreSQL 数据库中安装 ST_Geometry 空间数据类型。
- 使用创建 SQLite 数据库地理处理工具或 ArcPy 函数创建包含 ST_Geometry 空间数据类型的 SQLite 数据库，并加载 ST_Geometry 函数以与数据库配合使用。
- 加载 ST_Geometry 函数以用于移动地理数据库。

在 Oracle 和 PostgreSQL 数据库中，ST_Geometry 类型及其函数创建于名为 sde 的方案中。在 SQLite 中，类型和函数存储在对 SQLite 数据库或移动地理数据库运行 SQL 之前必须加载的库中。

提示：

有关 Esri ST_Geometry 类型的信息，请参阅以下 ArcGIS Pro 帮助页面：

- PostgreSQL 中的 ST_Geometry
- Oracle 中的 ST_Geometry
- 数据库和 ST_Geometry
- 加载 SQLite ST_Geometry 库
- 将 ST_Geometry 加载到移动地理数据库以进行 SQL 访问

SQL 函数页面的格式

此文档中的函数页面结构如下：

- 定义 - 函数功能的简要说明
- 语法 - 使用函数的 SQL 语法

注：

对于关系运算符来说，参数的指定顺序十分重要：第一个参数必须表示从中进行选择的表，而第二个参数必须表示正用作过滤器的表。

- 返回类型 - 执行函数时所返回的数据类型
- 示例 - 使用特定函数的示例

SQL 函数的列表

单击下面的链接将会转到可与 Oracle、PostgreSQL 和 SQLite 中的 ST_Geometry 类型配合使用的函数。

在 Oracle 中使用 ST_Geometry 函数时，必须使用 sde 对函数和运算符进行限定。例如，ST_Buffer 将为 sde.ST_Buffer。添加 sde. 将向软件说明该函数存储在 sde 用户的方案中。对于 PostgreSQL，可选择是否进行限

定，但包括限定符是一个很好的做法。 与 SQLite 配合使用函数时，不包括进行限定，因为在 SQLite 数据库中没有 sde 方案。

当使用 ST_Geometry SQL 函数提供熟知文本字符串作为输入时，可以使用科学记数法来指定非常大或非常小的值。例如，如果在构建要素时使用熟知的文本指定坐标，并且其中一个坐标为 0.000023500001816501026，则可以键入 `2.3500001816501026e-005` 代替。

提示：

对于其他空间类型（例如 PostGIS 类型、Oracle SDO_Geometry、Microsoft SQL Server 空间类型、IBM Db2 ST_Geometry 或 SAP HANA ST_Geometry），请参阅数据库管理系统供应商提供的文档，以获取有关这些空间类型所使用的函数的信息。

以下 Esri ST_Geometry SQL 函数根据用途进行分组。

构造函数

构造函数会获取一种几何类型或几何的文本说明，然后创建一个几何。 下表列出了构造函数，并指出哪些 ST_Geometry 实现支持所有的构造函数。

构造函数

功能	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
ST_MPolyFromText	X		X
ST_MPolyFromWKB	X	X	X
ST_MultiCurve	X		
ST_MultiLineString	X	X	X
ST_MultiPoint	X	X	X

功能	Oracle	PostgreSQL	SQLite
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

存取器函数

有很多函数都采用一个或多个几何作为输入，并返回关于几何的特定信息。

其中一些存取器函数会检查要素是否符合特定条件。 如果几何满足条件，则函数会返回 1（Oracle 和 SQLite）或 t (PostgreSQL)（表示 true）。 如果几何不满足条件，则函数会返回 0（Oracle 和 SQLite）或 f (PostgreSQL)（表示 false）。

如无特殊说明，这些函数适用于所有实现。

存取器函数

ST_Area
ST_AsBinary
ST_AsText
ST_CoordDim
ST_Dimension
ST_EndPoint
ST_Entity
ST_Equalsrs（仅 PostgreSQL）
ST_ExteriorRing
ST_GeomFromCollection（仅 PostgreSQL）
ST_GeometryType
ST_GeoSize（仅 PostgreSQL）
ST_Is3d
ST_IsClosed
ST_IsEmpty
ST_IsMeasured
ST_IsRing

ST_IsSimple
ST_Length
ST_M
ST_MaxM
ST_MaxX
ST_MaxY
ST_MaxZ
ST_MinM
ST_MinX
ST_MinY
ST_MinZ
ST_NumGeometries
ST_NumInteriorRing
ST_NumPoints
ST_Perimeter
ST_SRID
ST_StartPoint
ST_X
ST_Y
ST_Z

关系函数

关系函数将几何作为输入并确定各几何之间是否存在空间关系。 如果满足空间关系的条件，则这些函数会返回 1（Oracle 和 SQLite）或 t (PostgreSQL)（表示 true）。 如果未满足条件（不存在关系），则这些函数会返回 0（Oracle 和 SQLite）或 f (PostgreSQL)（表示 false）。

如无特殊说明，这些函数适用于所有实现。

关系函数

ST_Contains
ST_Crosses
ST_Disjoint
ST_Distance
ST_DWithin
ST_EnvIntersects（仅 Oracle 和 SQLite）
ST_Equals
ST_Intersects

ST_OrderingEquals（仅 Oracle 和 PostgreSQL）
ST_Overlaps
ST_Relate
ST_Touches
ST_Within

几何运算函数

这些函数利用空间数据、执行空间运算并返回几何。

如无特殊说明，这些函数适用于所有实现。

几何运算函数

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

SQL 和 Esri ST_Geometry

可以使用数据库管理系统的结构化查询语言 (SQL)、数据类型和表格式来处理存储在已安装 ST_Geometry 类型的地理数据库或数据库中的信息。 SQL 数据库语言支持数据定义和数据操作命令。

通过 SQL 访问数据允许外部应用程序使用地理数据库或数据库管理的表格数据。 这些外部应用程序可以是非空间数据库应用程序或自定义空间应用程序。

使用 SQL 将数据插入地理数据库或数据库中或在其中编辑数据时，请在运行 SQL 语句后发出 `COMMIT` 或 `ROLLBACK` 语句，以确保更改已提交到数据库或已撤消更改。 这有助于防止在您正在编辑的行、页面或表上保持锁定。

使用 SQL 插入 ST_Geometry 数据

可以使用 SQL 将空间数据插入到具有 ST_Geometry 列的数据库或地理数据库表中。 可以使用 ST_Geometry 构造函数插入特定的几何类型。 还可以指定将某些空间运算函数的输出输出到现有表中。

使用 SQL 将几何图形插入表时，请注意以下事项：

- 必须指定有效的空间参考 ID (SRID)。
- 同一列中的所有几何必须使用相同的 SRID。
- 要继续在 ArcGIS 中使用该表，用作对象 ID 的字段不能为空或包含非唯一值。

空间参考 ID

将几何插入到 Oracle 中使用 ST_Geometry 空间类型的表时指定的 SRID 必须位于 ST_SPATIAL_REFERENCES 表中，并且在 SDE.SPATIAL_REFERENCES 表中具有匹配记录。 在将几何插入到 PostgreSQL 中使用 ST_Geometry 空间类型的表时指定的 SRID 必须位于 public.sde_spatial_references 表中。 这些表预先填充了空间参考和 SRID。

在将几何插入到 SQLite 中使用 ST_Geometry 空间类型的表时指定的 SRID 必须位于 st_spatial_reference_systems 表中。

如果您需要使用表中不存在的自定义空间参考，最简单的方法是使用加载或创建具有所需空间参考值的要素类。 确保您创建的要素类使用 ST_Geometry 存储。 这将在 Oracle 的 SDE.SPATIAL_REFERENCES 和 ST_SPATIAL_REFERENCES 表中创建一条记录，在 PostgreSQL 的 public.sde_spatial_references 表中创建一条记录，或者在 SQLite 的 st_aux_spatial_reference_systems_table 中创建一条记录。

在地理数据库中，可以查询 LAYERS (Oracle) 或 sde_layers (PostgreSQL) 表以查找分配给空间表的 SRID。 然后，您可以在创建空间表并使用 SQL 插入数据时使用该 SRID。

注：

为了使用本文档中的示例，已将一条记录添加到 ST_SPATIAL_REFERENCES 和 sde_spatial_references 表以表示未知空间参考。 此记录的 SRID 为 0 您可以将此 SRID 用于本文档中的示例。 但是，这不是官方的 SRID - 提供它是为了执行示例 SQL 代码。 建议不要将此 SRID 用于生产数据。

对象 ID

为了 ArcGIS 能够查询数据，表需要包含唯一对象标识符字段。

使用 ArcGIS 创建的要素类始终具有用作标识符字段的对象 ID 字段。 使用 ArcGIS 将记录插入要素类时，始终会向对象 ID 字段插入唯一值。 地理数据库表中的对象 ID 字段由 ArcGIS 维护。 从 ArcGIS 创建的数据库表中的对象 ID 字段由数据库管理系统维护。

使用 SQL 将记录插入地理数据库表时，必须插入有效的唯一对象 ID 值。

您在 ArcGIS 外部创建的数据库表必须具有 ArcGIS 可用作对象 ID 的一个字段（或一组字段）。 如果您对表中的 ID 字段使用数据库的本地自动递增数据类型，则当使用 SQL 插入记录时，该字段将由数据库填充。 如果您手动维护唯一标识符字段中的值，请确保在从 SQL 编辑表时为 ID 提供唯一值。

注：

不能从具有唯一标识符字段的表中发布数据，该字段不是由 ArcGIS 或数据库管理系统维护的。

使用 SQL 编辑 ST_Geometry 数据

对现有记录进行的 SQL 编辑通常会影响存储在表中的非空间属性；但是，您可以使用 SQL UPDATE 语句中的构造函数编辑 ST_Geometry 列中的数据。

如果数据存储在地理数据库中，则在使用 SQL 进行编辑时必须遵循其他准则：

- 如果数据已注册为版本化或已启用地理数据库存档，则不要使用 SQL 更新记录。
- 切勿修改影响数据库中参与地理数据库行为的其他对象的任何属性，例如关系类、关联要素的注记、拓扑、属性规则或网络。
- 切勿使用 SQL 来更改表模式。

警告:

使用 SQL 访问地理数据库会忽略地理数据库功能，如版本化、拓扑、网络、地形、关联要素的注记或其他类或工作空间扩展模块。 可以使用数据库管理系统功能（例如触发器和存储过程）来维护某些地理数据库功能所需的表之间的关系。 但是，如果对数据库执行 SQL 命令而不考虑此附加功能（例如，执行 INSERT 语句将记录添加到已启用地理数据库存档的表或向现有要素类添加列）将忽略地理数据库功能，并且可能破坏地理数据库中数据之间的关系。

加载 SQLite ST_Geometry 库

在针对 SQLite 数据库运行包含 ST_Geometry 函数的 SQL 命令之前，请执行以下操作：

1. 从 My Esri 中下载 ArcGIS Pro ST_Geometry 库 (SQLite) zip 文件并对其进行解压。
2. 在与数据库相同的计算机上安装 SQL 编辑器。
3. 将 ST_Geometry 文件放置在 SQLite 数据库和 SQL 编辑器可访问的位置，将从该位置中加载 ST_Geometry。
   如果 SQLite 数据库位于 Microsoft Windows 计算机上，请使用 `stgeometry_sqlite.dll` 文件。 如果 SQLite 数据库位于 Linux 计算机上，请使用 `libstgeometry_sqlite.so` 文件。
4. 打开 SQL 编辑器并连接到 SQLite 数据库。
5. 加载 ST_Geometry 库。
   在下面的第一个示例中，为 Windows 计算机上的 SQLite 数据库加载了库。 在第二个示例中，为 Linux 计算机上的 SQLite 数据库加载了库。

```
--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'
);
```

您现在可以针对 SQLite 数据库运行包含 ST_Geometry 函数的 SQL 命令。

ST_Geometry 的构造函数

构造函数可根据熟知文本描述或熟知二进制创建几何。

在提供熟知文本描述以构造几何时，必须在最后指定测量坐标。 例如，如果文本中含有 x、y、z 和 m 坐标，则必须按此顺序提供。

几何可以有零个或更多个点。 如果几何有零个点，则其被视为空几何。 点子类型是唯一的一种限制为零个点或一个点的几何；其他所有子类型都可有零个或更多个点。

以下各部分将介绍几何超类和子类几何，还列出了可以构造每种几何的函数。

您还可以通过对现有几何执行空间运算来构建几何。

几何超类

ST_Geometry 超类不能实例化；尽管您可以定义一个类型为 ST_Geometry 的列，但插入的实际数据将被定义为点、线串、面、多点、多线串或多面实体。

您可以使用以下函数来创建超类，用于保存前面提到的任何实体类型。

- ST_Geometry
- ST_GeomFromText（仅限 Oracle 和 SQLite）
- ST_GeomFromWKB

子类

您可以将一个要素定义为具体子类，在这种情况下，只能插入该子类允许的实体类型。 例如，ST_PointFromWKB 只能构造点实体。

ST_Point

ST_Point 是零维度几何，它在坐标空间中占据单个位置。 ST_Point 具有单个 x,y 坐标值（始终是简单的），并且边界为空。 ST_Point 可以用来定义诸如油井、地标和水样采集点等要素。

创建点的函数如下所示：

- ST_Point
- ST_PointFromText（仅限 Oracle 和 SQLite）
- ST_PointFromWKB

ST_MultiPoint

ST_MultiPoint 是 ST_Point 的集合，并且其维度为 0，与其元素相同。 如果 ST_MultiPoint 中的元素占据的坐标空间互不相同，则它是简单的。 ST_MultiPoint 的边界为空。 ST_MultiPoint 可以定义诸如天线广播模式和疾病爆发事件等。

创建多点几何的函数如下所示：

- ST_MultiPoint
- ST_MPointFromText（仅限 Oracle）
- ST_MPointFromWKB

ST_LineString

ST_LineString 是一维对象，它以点序列的形式进行存储，用于定义线性插值路径。 如果 ST_LineString 不与其内部相交，则 ST_LineString 为简单几何。 闭合的 ST_LineString 的端点（边界）占据空间中的相同点。 如果 ST_LineString 是闭合的并且是简单的，那么它是一个环。 与从超类 ST_Geometry 继承的其他属性一样，ST_LineString 具有长度属性。 ST_LineString 通常用于定义线状要素，如道路、河流和电力线。

端点通常形成 ST_LineString 的边界，除非 ST_LineString 是闭合的（在这种情况下边界为空）。 ST_LineString 的内部是端点之间的连通路径，除非它是闭合的（在这种情况下内部是连续的）。

创建线串的函数包括：

- ST_LineString
- ST_LineFromText（仅限 Oracle 和 SQLite）
- ST_LineFromWKB
- ST_Curve（仅限 Oracle 和 SQLite）

ST_MultiLineString

ST_MultiLineString 是若干 ST_LineString 的集合。

ST_MultiLineString 的边界是 ST_LineString 元素的非相交端点。 如果 ST_MultiLineString 的所有元素的所有端点都相交，则它的边界为空。 ST_MultiLineString 不仅具有从超类 ST_Geometry 继承的其他属性，还具有长度属性。 ST_MultiLineString 用于定义不毗连的线状要素，如河流或道路网络。

构造多线串的函数如下所示：

- ST_MultiLineString
- ST_MLineFromText（仅限 Oracle 和 SQLite）
- ST_MLineFromWKB
- ST_MultiCurve（仅限 Oracle）

ST_Polygon

ST_Polygon 是二维表面，它以点序列的形式存储，用于定义其外接环以及 0 个或更多内部环。 ST_Polygon 始终是简单的。 ST_Polygon 用于定义具有空间范围的要素，例如地块、水体以及管辖区。

下图显示了 ST_Polygon 对象的示例：1 是边界由外部环定义的 ST_Polygon。 2 是边界由外部环和两个内部环定义的 ST_Polygon。 内部环内的区域是 ST_Polygon 外部环的一部分。 3 是有效 ST_Polygon，因为环之间相切。

外部环和任意内部环确定了 ST_Polygon 的边界，环之间的封闭空间确定了 ST_Polygon 的内部。 ST_Polygon 的环可以相切，但绝不可以相交。 ST_Polygon 不仅具有从超类 ST_Geometry 继承的其他属性，还具有面积属性。

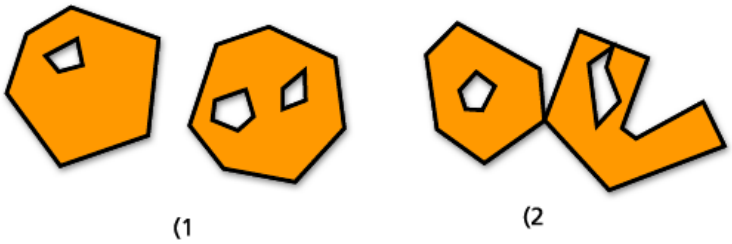
创建面的函数包括：

- ST_Polygon
- ST_PolyFromText（仅限 Oracle 和 SQLite）
- ST_PolyFromWKB
- ST_Surface（仅限 Oracle 和 SQLite）

ST_MultiPolygon

ST_MultiPolygon 的边界是其元素外部环和内部环的累积长度。 ST_MultiPolygon 的内部被定义为其 ST_Polygon 元素的累积内部。 ST_MultiPolygon 的元素的边界只能相切。 ST_MultiPolygon 不仅具有从超类 ST_Geometry 继承的其他属性，还具有面积属性。 ST_MultiPolygon 用于定义诸如森林地层或不毗连的地块（如太平洋岛链）等要素。

下图提供了 ST_MultiPolygon 的示例：1 是具有两个 ST_Polygon 元素的 ST_MultiPolygon。 边界由两个外部环和三个内部环定义。 2 也是具有两个 ST_Polygon 元素的 ST_MultiPolygon，但边界由两个外部环和两个内部环定义，两个 ST_Polygon 元素相切。



创建多面的函数如下所示：

- ST_MultiPolygon
- ST_MPolyFromText（仅限 Oracle 和 SQLite）
- ST_MPolyFromWKB
- ST_MultiSurface（仅限 Oracle）

从现有几何构造几何

虽然不是严格的构造函数，但以下函数通过将现有几何作为输入并对其执行分析来返回新几何：

- ST_Aggr_ConvexHull（仅 Oracle 和 SQLite）
- ST_Aggr_Intersection（仅 Oracle 和 SQLite）
- ST_Aggr_Union
- ST_Boundary
- ST_Buffer
- ST_Centroid
- ST_ConvexHull
- ST_Difference
- ST_Envelope
- ST_ExteriorRing

- ST_Intersection
- ST_SymmetricDiff
- ST_Transform
- ST_Union

ST_Geometry 的空间存取器函数

空间存取器函数返回几何的属性。 有一些存取器函数可用于确定 ST_Geometry 要素的下列属性：

维数

几何的维度是定义几何的空间范围所需的最小坐标（无, x, y）。

几何可具有 0、1 或 2 个维度。

维度所代表的意义如下：

- 0 - 既不具有长度也不具有面积
- 1 - 具有长度（x 或 y）
- 2 - 包含面积（x 和 y）

点和多点子类型的维度为 0。 点表示零维要素，可使用单个坐标为其建模，而多点表示必须使用离散坐标集进行建模的数据。

线串和多线串子类型的维度为 1。 它们存储诸如路段、分支河流系统等要素以及其他任何具有线性本质的要素。

面和多面子类型的维度为 2。 林分、宗地、水体以及其他周长包围某一可定义区域的要素都可以按面或多面数据类型渲染。

维度不仅是重要的子类型属性，它在确定两个要素的空间关系时也很重要。 生成要素的维度可以确定运算是否成功。 空间存取器函数检查要素的维度以确定应如何对其进行比较。

要评估几何的维度，请使用 ST_Dimension 函数，该函数采用 ST_Geometry 要素并将其维度以整数形式返回。

几何的坐标也具有维度。如果几何仅具有 x 和 y 坐标，则坐标维度为 2。如果几何具有 x、y 和 z 坐标，则坐标维度为 3。如果几何具有 x、y、z 和 m 坐标，则坐标维度为 4。

可以使用 ST_CoordDim 函数来确定几何中存在的坐标维度。

Z 坐标

一些几何具有相关的高度或深度（第三维度）。形成要素几何的每个点都可以包含表示相对地球表面的高度或深度的可选 z 坐标。

ST_Is3d 谓词函数将 ST_Geometry 作为输入，如果函数具有 z 坐标，则返回 true；如果没有，则返回 false。

可以使用 ST_Z 函数来确定点的 Z 坐标。

ST_MaxZ 函数返回几何的最大 z 坐标，ST_MinZ 函数返回几何的最小 z 坐标。

测量值

测量值是分配给每个坐标的值，其用于线性参考和动态分段应用。例如，沿公路的里程标志位置可包含指示其位置的测量值。值将表示可以双精度数形式存储的任何项目。

ST_IsMeasured 谓词函数处理几何，如果几何包含测量值，则返回 true；如果不包含，则返回 false。 该函数仅与 ST_Geometry 的 Oracle 和 SQLite 实施配合使用。

可以使用 ST_M 函数来获得某点的测量值。

ST_MaxM 函数返回几何的 m 坐标最大值，而 ST_MinM 函数返回几何的 m 坐标最小值。

几何类型

几何类型指的是几何实体的类型。其中包括以下内容：

- 点和多点
- 线和多线
- 面和多面

ST_Geometry 是可以存储各种子类型的超类。 要确定某个几何属于哪种子类型，请使用 ST_GeometryType 或 ST_Entity（仅限 Oracle 和 SQLite）函数。

点（折点）集合与点数

几何可以有零个或更多个点。 如果几何有零个点，则其被视为空几何。 点子类型是唯一的一种限制为零个点或一个点的几何；其他所有子类型都可有零个或更多个点。

ST_Point

ST_Point 是零维度几何，它在坐标空间中占据单个位置。 ST_Point 具有单个 x,y 坐标值（始终是简单的），并且边界为空。 ST_Point 可以用来定义诸如油井、地标和水样采集点等要素。

以下是仅适用于 ST_Point 数据类型的函数：

- ST_X - 以双精度数字形式返回点数据类型的 x 坐标值
- ST_Y - 以双精度数字形式返回点数据类型的 y 坐标值
- ST_Z - 以双精度数字形式返回点数据类型的 z 坐标值
- ST_M - 以双精度数字形式返回点数据类型的 m 坐标值

ST_MultiPoint

ST_MultiPoint 是 ST_Point 的集合，并且其维度为 0，与其元素相同。 如果 ST_MultiPoint 中的元素占据的坐标空间互不相同，则它是简单的。 ST_MultiPoint 的边界为空。 ST_MultiPoint 可以定义诸如天线广播模式和疾病爆发事件等。

可以使用 ST_NumGeometries 函数来确定多点几何中的点数量。

长度、面积和周长

长度、面积和周长是几何的可测量特征。 线串和多线串的元素都是一维的，具有长度特征。 面和多面的元素是二维表面，因此面积和周长可以测量。 可以使用函数 ST_Length、ST_Area 和 ST_Perimeter 来确定这些属性。 测量单位根据数据的存储方式的不同而有所变化。

ST_LineString

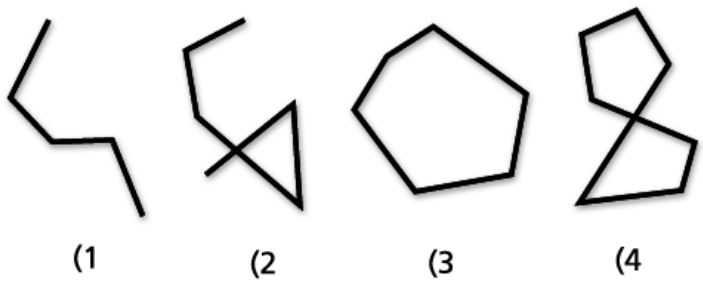
ST_LineString 是一维对象，它以点序列的形式进行存储，用于定义线性插值路径。 如果 ST_LineString 不与其内部相交，则 ST_LineString 为简单几何。 闭合的 ST_LineString 的端点（边界）占据空间中的相同点。 如果 ST_LineString 是闭合的并且是简单的，那么它是一个环。 与从超类 ST_Geometry 继承的其他属性一样，ST_LineString 具有长度属性。 ST_LineString 通常用于定义线状要素，如道路、河流和电力线。

端点通常形成 ST_LineString 的边界，除非 ST_LineString 是闭合的（在这种情况下边界为空）。 ST_LineString 的内部是端点之间的连通路径，除非它是闭合的（在这种情况下内部是连续的）。

以下是适用于 ST_LineString 的函数：

- ST_StartPoint - 返回指定 ST_LineString 的第一个点
- ST_EndPoint - 返回 ST_LineString 的最后一个点
- ST_IsClosed - 如果指定的 ST_LineString 是闭合的（线串起点和端点相交），则谓词函数将返回 true；如果不是，则返回 false
- ST_IsRing - 如果指定的 ST_LineString 为环，则谓词函数将返回 true；如果不是环，则返回 false
- ST_Length - 以双精度数字形式返回 ST_LineString 的长度
- ST_NumPoints - 评估 ST_LineString，并以整数形式返回其序列中的点数
- ST_PointN - 处理 ST_LineString 和第 n 个点的索引，并返回该点

下图显示 ST_LineString 对象的示例：(1 是简单非闭合 ST_LineString；(2 是非简单非闭合 ST_LineString；(3 是闭合简单 ST_LineString，因此是环；(4 是闭合非简单 ST_LineString，但不是环。

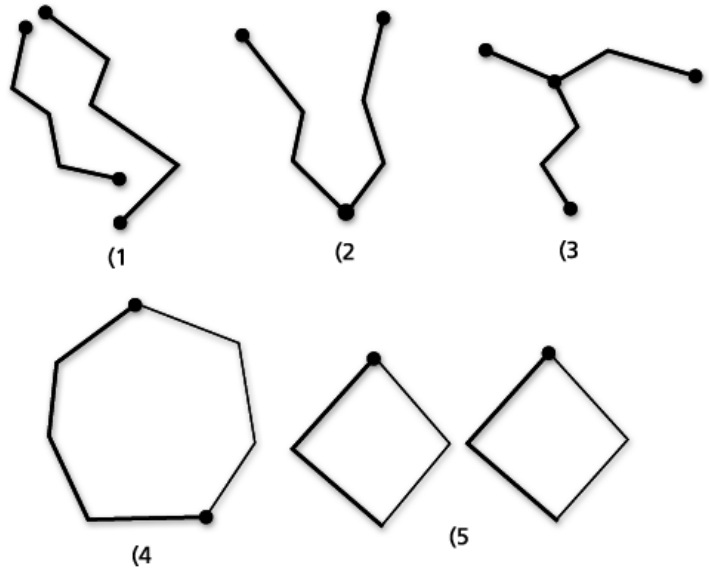


ST_MultiLineString

ST_MultiLineString 是若干 ST_LineString 的集合。 如果 ST_MultiLineString 只在 ST_LineString 元素的端点相交，则它是简单的。 如果 ST_LineString 元素的内部相交，则 ST_MultiLineString 是非简单的。

ST_MultiLineString 的边界是 ST_LineString 元素的非相交端点。 如果 ST_MultiLineString 的所有元素的所有端点都相交，则它的边界为空。 ST_MultiLineString 不仅具有从超类 ST_Geometry 继承的其他属性，还具有长度属性。 ST_MultiLineString 用于定义不毗连的线状要素，如河流或道路网络。

下图提供 ST_MultiLineString 的示例：(1 是简单 ST_MultiLineString，其边界为其中包含的两个 ST_LineString 元素的四个端点。 (2 是简单 ST_MultiLineString，因为只有 ST_LineString 元素的端点是相交的。 边界是两个不相交端点。 (3 是非简单 ST_MultiLineString，因为它的一个 ST_LineString 元素的内部是相交的。 此 ST_MultiLineString 的边界是三个不相交的端点。 (4 是简单非闭合 ST_MultiLineString。 它不是闭合的，因为它的元素 ST_LineString 不闭合。 它是简单的，因为所有 ST_LineString 元素的内部都没有相交。 (5 是单个简单闭合 ST_MultiLineString。 它是闭合的，因为它的所有元素都是闭合的。 它是简单的，因为其所有元素均没有在内部相交。



以下是适用于 ST_MultiLineString 的函数：

- ST_IsClosed - 如果指定的 ST_MultiLineString 是闭合的，则该谓词函数将返回 true 值；如果不是闭合的，则返回 false 值。
- ST_Length - 此函数将评估 ST_MultiLineString 并以双精度数形式返回其所有 ST_LineString 元素的累积长度。
- ST_NumGeometries - 此函数将返回多线串中的线数量。

ST_Polygon

ST_Polygon 是二维表面，它以点序列的形式存储，用于定义其外接环以及 0 个或更多内部环。 ST_Polygon 始终是简单的。 ST_Polygon 用于定义具有空间范围的要素，例如地块、水体以及管辖区。

下图显示了 ST_Polygon 对象的示例：1 是边界由外部环定义的 ST_Polygon。 2 是边界由外部环和两个内部环定义的 ST_Polygon。 内部环内的区域是 ST_Polygon 外部环的一部分。 3 是有效 ST_Polygon，因为环之间相切。

外部环和任意内部环确定了 ST_Polygon 的边界，环之间的封闭空间确定了 ST_Polygon 的内部。 ST_Polygon 的环可以相切，但绝不可以相交。 ST_Polygon 不仅具有从超类 ST_Geometry 继承的其他属性，还具有面积属性。

以下是适用于 ST_Polygon 的函数：

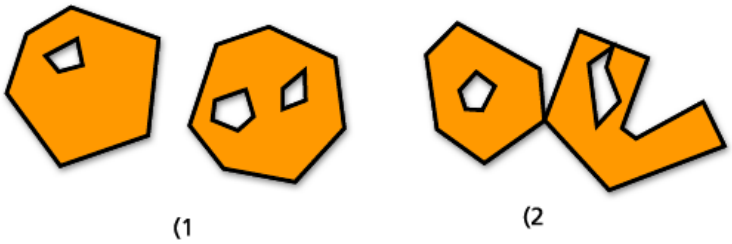
- ST_Area - 以双精度数字形式返回 ST_Polygon 的面积
- ST_Centroid - 返回表示 ST_Polygon 包络中心的 ST_Point
- ST_ExteriorRing - 以 ST_LineString 形式返回 ST_Polygon 的外部环
- ST_InteriorRingN - 评估 ST_Polygon 和索引，并以 ST_LineString 形式返回第 n 个内部环

- ST_NumInteriorRing - 返回 ST_Polygon 包含的内部环计数
- ST_PointOnSurface - 返回保证位于指定的 ST_Polygon 表面的 ST_LineString

ST_MultiPolygon

ST_MultiPolygon 的边界是其元素外部环和内部环的累积长度。 ST_MultiPolygon 的内部被定义为其 ST_Polygon 元素的累积内部。 ST_MultiPolygon 的元素的边界只能相切。 ST_MultiPolygon 不仅具有从超类 ST_Geometry 继承的其他属性，还具有面积属性。 ST_MultiPolygon 用于定义诸如森林地层或不毗连的地块（如太平洋岛链）等要素。

下图提供了 ST_MultiPolygon 的示例：1 是具有两个 ST_Polygon 元素的 ST_MultiPolygon。 边界由两个外部环和三个内部环定义。 2 也是具有两个 ST_Polygon 元素的 ST_MultiPolygon，但边界由两个外部环和两个内部环定义，两个 ST_Polygon 元素相切。



以下是适用于 ST_MultiPolygon 的函数：

- ST_Area - 以双精度数形式返回 ST_MultiPolygon 的 ST_Polygon 元素的累积 ST_Area。
- ST_Centroid - 以 ST_Point 形式返回 ST_MultiPolygon 包络的中心。
- ST_NumGeometries - 返回多面中的面数量计数。
- ST_PointOnSurface - 评估 ST_MultiPolygon，并返回保证是其中一个 ST_Polygon 元素表面的法线的 ST_Point。

多部分几何中的简单几何

多部分几何由各种简单几何组成。

您可能希望确定多部分几何中有多少单个几何，例如 ST_MultiPoint、ST_MultiLineString 和 ST_MultiPolygon。 为此，请使用 ST_NumGeometries 谓词函数。 此函数返回几何集合中各种元素的计数。

可以使用 ST_GeometryN 函数确定多部分几何中位置 N 处存在哪种几何；N 是您为此函数提供的数值。 例如，如果您想要返回多点几何的第三个点，则可以在执行此函数时输入“3”。

要从 PostgreSQL 的多部分几何中返回各个几何及其位置，请使用 ST_GeomFromCollection 函数。

内部、边界、外部

所有几何均在空间中占据由其内部、边界和外部所限定的位置。几何的外部是未被几何占据的所有空间。几何的内部是被几何占据的空间。几何的边界是介于几何的内部和外部之间的位置。子类型可直接继承内部和外部属性；而边界属性则因内部或外部而各不相同。

使用 ST_Boundary 函数以确定源 ST_Geometry 的边界。

简单与非简单

ST_Geometry 的某些子类型始终是简单的，例如 ST_Point 或 ST_Polygon。 但是，ST_LineString、ST_MultiPoint 和 ST_MultiLineString 的子类型可能是简单的，也可能是非简单的。 如果它们遵守所有对其施加的拓扑规则，则它们是

简单的；如果不遵守，则是非简单的。

以下是拓扑规则：

- 如果 ST_LineString 内部没有相交，则其为简单的；如果相交，则为非简单的。
- 如果 ST_MultiPoint 中的元素均没有占据相同的坐标空间（即 x,y 坐标相同），则它是简单的；如果有，则是非简单的。
- 如果 ST_MultiLineString 中的任何元素内部均没有与自身内部相交，则它是简单的；如果元素内部确实相交，则它是非简单的。

ST_IsSimple 谓词函数用于确定 ST_LineString、ST_MultiPoint 或 ST_MultiLineString 是简单的还是非简单的。ST_IsSimple 使用 ST_Geometry，在几何为简单时，将返回 true；否则返回 false。

空与非空

如果几何没有任何点，则该几何为空。空几何具有空的包络矩形、边界、内部和外部。空几何始终是简单的。空线串和空多线串的长度为 0。空面和空多面的面积为 0。

ST_IsEmpty 谓词函数可用于确定几何是否为空。它分析 ST_Geometry 并在几何为空时返回 true，否则返回 false。

IsClosed 和 IsRing

线串几何可以是闭合的，也可以是环。线串可以闭合，但不是环。您可以使用 ST_IsClosed 谓词函数确定线串是否闭合；如果线串的起点和终点相交，则返回 true。环是闭合且简单的线串。ST_IsRing 谓词函数可用于测试线串是否确实为环；如果线串是闭合且简单的，则该函数将返回 true。

包络矩形

每个几何都具有包络矩形。几何的包络矩形是由最小和最大 x,y 坐标所形成的边界几何。对于点几何，因为最小和最大 x,y 坐标相同，所以围绕这些坐标可创建矩形或包络矩形。对于线几何，线的端点表示包络矩形的两侧，另外两侧正好在线的上方和下方创建。

ST_Envelope 函数处理 ST_Geometry 并返回表示源 ST_Geometry 包络矩形的 ST_Geometry。

要查找几何的 x,y 坐标的最小值和最大值，请使用函数 ST_MinX、ST_MinY、ST_MaxX 和 ST_MaxY。

空间参考系统

空间参考系统可识别每个几何的坐标变换矩阵，其由坐标系、分辨率和容差组成。

地理数据库已知的所有空间参考系统都存储在地理数据库系统表中。

以下函数获取有关几何空间参考系统的信息：

- ST_SRID - 处理 ST_Geometry 并以整数形式返回其空间参考标识符。
- ST_Equalsrs - 确定两个不同的要素类的空间参考系统是否相同 (true) 或不同 (false)。

要素大小（仅限 PostgreSQL）

要素（表中的空间记录）占用一定数量的存储空间（以字节为单位）。可以使用 ST_GeoSize 函数来确定表中每个要素的大小。

几何的文本和二进制定义

要获取空间表的特定行中几何的熟知文本定义或熟知二进制定义，请分别使用 ST_AsText 和 ST_AsBinary 函数。

空间关系

GIS 的主要功能是确定要素之间的空间关系：它们是否重叠？ 一个是否被另一个包含？ 一个是否与另一个相交？

几何可以以不同方式在空间上相关联。 一个几何与另一个几何在空间上的关联方式的示例如下：

- 几何 A 穿过几何 B。
- 几何 A 完全包含于几何 B。
- 几何 A 完全包含几何 B。
- 几何不彼此相交或接触。
- 几何完全重合。
- 几何彼此重叠。
- 几何接触于一点。

要确定这些关系是否存在，请使用空间关系函数。 这些函数会比较您在查询中指定的以下几何属性：

- 几何的外部 (E) - 外部是未被几何占据的所有空间。
- 几何的内部 (I) - 内部是被几何占据的空间。
- 几何的边界 (B) - 边界是几何内部和外部的接口部分。

构建空间关系查询时，请指定要查找的空间关系类型和要比较的几何。 查询返回 true 或 false。 换言之，几何将在指定的空间关系中彼此参与或彼此不参与。 在多数情况下，您通过将空间关系查询放入 WHERE 子句中，来将其用于过滤结果。

例如，如果您有一个存储拟议开发地点位置的表和另一个存储重要考古遗址地点位置的表，您可以发出查询以确保没有任何开发地点与考古遗址相交，如果存在相交，将返回这些拟议开发工程 ID。 在此示例中，ST_Disjoint 函数用于 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
```

此查询返回非不相交的开发工程名称 (projname) 和考古地点 ID (siteid)，即相交地点。 它返回一个与考古地点 A1009 相交的开发工程 Bow Wow Chow。

ST_Geometry 的关系函数

关系函数使用谓词测试空间关系的不同类型。 测试通过比较以下之间的关系来实现：

- 几何的外部 (E) - 外部是未被几何占据的所有空间。
- 几何的内部 (I) - 内部是被几何占据的空间。
- 几何的边界 (B) - 边界是几何内部和外部的接口部分。

谓词测试关系。 如果比较满足函数的条件，则将返回 1（Oracle 和 SQLite）或 t (PostgreSQL)；否则，将返回 0（Oracle 和 SQLite）或 f (PostgreSQL)。 用于测试空间关系的谓词将比较可能具有不同类型或维度的几何对。

谓词比较所提交几何的 x 和 y 坐标。 如果 z 坐标和测量值存在，将被忽略。 具有 z 坐标或测量值的几何可与不具 z 坐标或测量值的几何比较。

Clementini, et al. 开发的维度扩展九交模型 (DE-9IM) 在维度上扩展 Egenhofer 和 Herring 的九交模型。 DE-9IM 是一种数学方法，它定义了不同类型和维度的几何之间的对式空间关系。 该模型将所有几何类型之间的空间关系表示为它们的内部、边界和外部的成对交集，并考虑了生成交集的维度。

给定几何 a 和 b，I(a)、B(a) 和 E(a) 表示 a 的内部、边界和外部，I(b)、B(b) 和 E(b) 表示 b 的内部、边界和外部。 I(a)、B(a) 和 E(a) 与 I(b)、B(b) 和 E(b) 的交集生成一个 3x3 矩阵。 每个交集可生成不同维度的几何。 例如，两个面边界的交集可包含点和线串，在这种情况下，dim（维度）函数将返回最大维度 1。

dim 函数返回值 -1、0、1 或 2。 -1 对应于未找到交集时返回的空集合或 dim(Ãƒ˜)。

	内部	边界	外部
内部	dim(I(a) intersects I(b))	dim(I(a) intersects B(b))	dim(I(a) intersects E(b))
边界	dim(B(a) intersects I(b))	dim(B(a) intersects B(b))	dim(B(a) intersects E(b))
外部	dim(E(a) intersects I(b))	dim(E(a) intersects B(b))	dim(E(a) intersects E(b))

谓词交集示例

通过将谓词的结果与表示 DE-9IM 可接受值的模式矩阵进行比较，可以理解或验证空间关系谓词的结果。

模式矩阵包含每个交集矩阵单元格的可接受值。 可能的模式值如下：

T - 交集必须存在；dim = 0、1 或 2

F - 交集不得存在；dim = -1

* - 是否存在交集无关紧要；dim = -1、0、1 或 2

0 - 交集必须存在，其最大维度必须为 0；dim = 0

1 - 交集必须存在，其最大维度必须为 1；dim = 1

2 - 交集必须存在，其最大维度必须为 2；dim = 2

每个谓词至少有一个模式矩阵，但某些谓词需要多个模式矩阵以描述各种几何类型组合的关系。

几何组合的 ST_Within 谓词的模式矩阵具有以下形式：

			几何 b	
		内部	边界	外部
几何 a	内部	T	*	F

示例模式矩阵

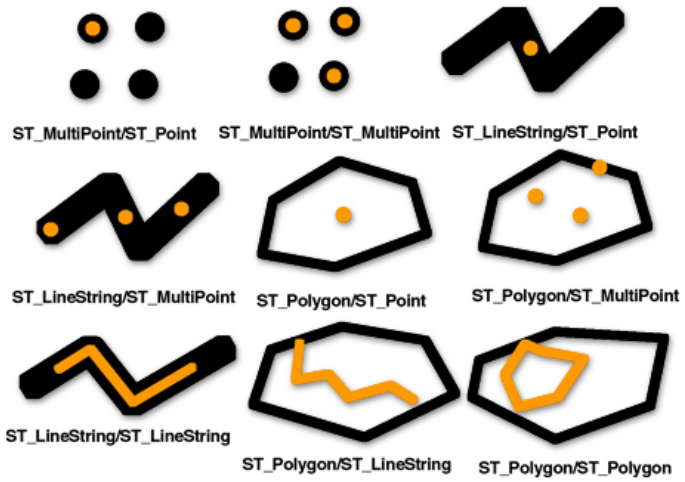
	边界	*	*	F
	外部	*	*	*

两个几何相交且几何 a 的内部和边界不与几何 b 的外部相交时，ST_Within 谓词为 true。 所有其他条件无关紧要。

以下部分描述了空间关系使用的不同谓词。 在这些部分的逻辑示意图中，第一个列出的输入几何显示为黑色，第二个显示为橙色。

ST_Contains

如果第二个几何完全包含于第一个几何中，则 ST_Contains 将返回 1 或 t (true)。 ST_Contains 谓词将返回与 ST_Within 谓词完全相反的结果。

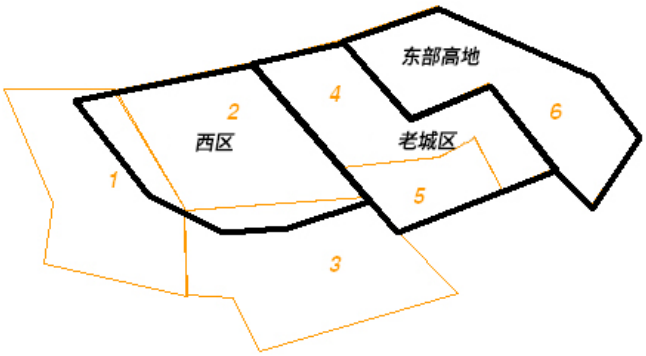


ST_Contains 谓词的模式矩阵表明两个几何的内部必须相交且次要几何（几何 b） 的内部和边界不能与主要几何（几何 a） 的外部相交。

			几何 b	
		内部	**边界**	**外部**
几何 a	**内部**	T	*	*
	边界	*	*	*
	外部	F	F	*

ST_Contains 矩阵

ST_Within 和 ST_Contains 函数仅识别完全落入另一几何内的几何。 这有助于从您的选择中消除可能会扭曲结果的要素。 在以下示例中，一位流动冰淇淋供应商想要确定哪些社区包含的儿童（潜在客户）数量最多并将他的路径限制为这些区域。 供应商将指定社区面与具有 16 岁以下儿童总数属性的人口普查区域进行比较。



除非居住在人口普查区域 1 和人口普查区域 3 的所有儿童都居住在位于 Westside 内的狭长土地上，否则在选择中包括这些区域可能会错误地增加 Westside 社区的儿童人数。 通过指定仅包括完全位于社区内的人口普查区域 (ST_Within = 1)，冰淇淋供应商可以避免进入 Westside 的这些部分，从而可能节省时间和金钱。

ST_Crosses

如果交集生成的几何维度比两个源几何的最大维度小 1 并且交集位于两个源几何的内部，则 ST_Crosses 将返回 1 或 t (true)。 ST_Crosses 仅对 ST_MultiPoint/ST_Polygon、ST_MultiPoint/ST_LineString、ST_Linestring/ST_LineString、ST_LineString/ST_Polygon 和 ST_LineString/ST_MultiPolygon 比较返回 1 或 t (true)。

ST_MultiPoint/ST_LineString

ST_MultiPoint/ST_Polygon

ST_Linestring/ST_Linestring

ST_Linestring/ST_Polygon

以下 ST_Crosses 谓词模式矩阵适用于 ST_MultiPoint/ST_LineString、ST_MultiPoint/ST_MultiLineString、ST_MultiPoint/ST_Polygon、ST_MultiPoint/ST_MultiPolygon、ST_LineString/ST_Polygon 和 ST_LineString/ST_MultiPolygon。 矩阵表明内部必须相交，并且至少主要几何（几何 a）的内部必须与次要几何（几何 b）的外部相交。

			几何 b	
		内部	边界	**外部**
几何 a	**内部**	T	*	T
	边界	*	*	*
	外部	*	*	*

ST_Crosses 矩阵 1

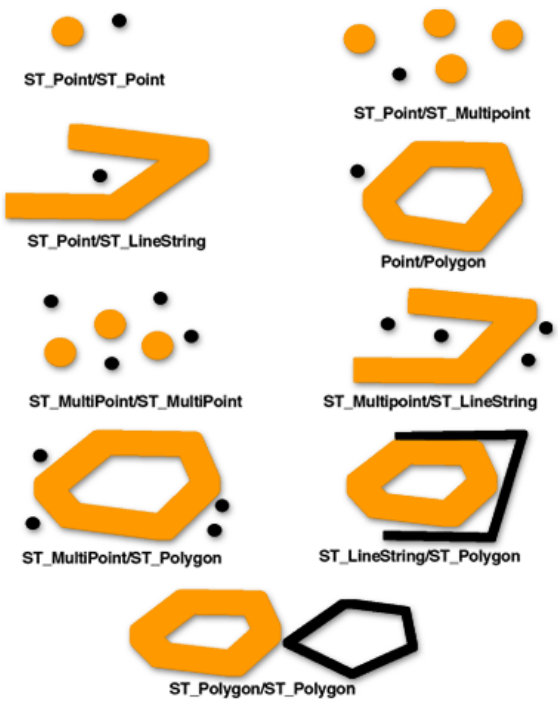
以下 ST_Crosses 谓词矩阵适用于 ST_LineString/ST_LineString、ST_LineString/ST_MultiLineString 和 ST_MultiLineString/ST_MultiLineString。 矩阵表明内部交集的维度必须为 0（相交于点）。 如果此交集的维度为 1（相交于线串），则 ST_Crosses 谓词将返回 false，而 ST_Overlaps 谓词将返回 true。

			几何 b	
		内部	边界	外部
几何 a	内部	0	*	*
	边界	*	*	*
	外部	*	*	*

ST_Crosses 矩阵 2

ST_Disjoint

如果两个几何的交集为空集，则 ST_Disjoint 将返回 1 或 t (true)。 换言之，如果几何彼此不相交，则几何不相交。



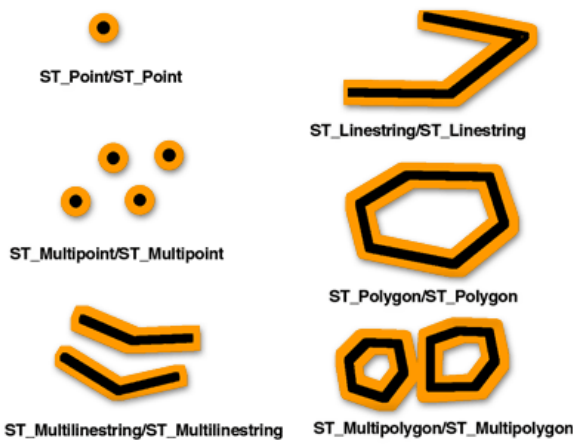
ST_Disjoint 谓词的模式矩阵表明各几何的内部和边界都不相交。

			几何 b	
		内部	边界	外部
几何 a	内部	F	F	*
	边界	F	F	*
	外部	*	*	*

ST_Disjoint 矩阵

ST_Equals

如果相同类型的两个几何具有相同的 x,y 坐标值，则 ST_Equals 将返回 1 或 t (true)。 办公楼的第一和第二层可能具有相同的 x,y 坐标，因此相等。 ST_Equals 也可识别两个要素是否被错误叠加。



相等的 DE-9IM 模式矩阵可以确保内部相交，并且任一几何的内部或边界的任何部分都不会与另一几何的外部相交。

			几何 b	
		内部	边界	**外部**
几何 a	**内部**	T	*	F
	边界	*	*	F
	外部	F	F	*

ST_Equals 矩阵

ST_Intersects

如果交集不是空集，则 ST_Intersects 将返回 1 或 t (true)。 ST_Intersects 返回与 ST_Disjoint 完全相反的结果。

如果以下任何模式矩阵的条件返回 true，ST_Intersects 谓词将返回 true。

如果两个几何的内部相交，则 ST_Intersects 谓词将返回 true。

			几何 b	
		内部	边界	**外部**
几何 a	**内部**	T	*	*
	边界	*	*	*
	外部	*	*	*

ST_Intersects 矩阵 1

如果第一个几何的内部与第二个几何的边界相交，则 ST_Intersects 谓词将返回 true。

			几何 b	
		内部	边界	**外部**
几何 a	**内部**	*	T	*
	边界	*	*	*

ST_Intersects 矩阵 2

	外部	*	*	*

如果第一个几何的边界与第二个几何的内部相交，则 ST_Intersects 谓词将返回 true。

			几何 b	
		内部	边界	**外部**
几何 a	**内部**	*	*	*
	边界	T	*	*
	外部	*	*	*

ST_Intersects 矩阵 3

如果任一几何的边界相交，则 ST_Intersects 谓词将返回 true。

			几何 b	
		内部	边界	**外部**
几何 a	**内部**	*	*	*
	边界	*	T	*
	外部	*	*	*

ST_Intersects 矩阵 4

ST_Overlaps

如果两个几何的交集生成与两个输入几何不同但具有相同维度的几何，则 ST_Overlaps 将比较两个具有相同维度的几何并返回 1 或 t (true)。

仅当两个几何的交集生成具有几何具有相同维度的几何时，ST_Overlaps 才会针对具有相同维度的几何返回 1 或 t (true)。 换言之，如果两个 ST_Polygons 的交集生成一个 ST_Polygon，重叠将返回 1 或 t (true)。

ST_LineString/ST_LineString

ST_Polygon/ST_Polygon

ST_MultiPoint/ST_MultiPoint

该模式矩阵适用于 ST_Polygon/ST_Polygon、ST_MultiPoint/ST_MultiPoint 和 ST_MultiPolygon/ST_MultiPolygon overlays。 对于这些组合，如果两个几何的内部与另一个几何的内部和外部相交，则重叠谓词将返回 true。

			几何 b	

ST_Overlaps 矩阵 1

		内部	边界	外部
几何 a	**内部**	T	*	T
	边界	*	*	*
	外部	T	*	*

以下模式矩阵适用于 ST_LineString/ST_LineString 和 ST_MultiLineString/ST_MultiLineString overlaps。 在这种情况下，几何的交集必须生成维度为 1 的几何（另一个 ST_LineString 或 ST_MultiLineString）。 如果内部交集的维度为 0（一个点），则 ST_Overlaps 谓词将返回 false。 但是，ST_Crosses 谓词会返回 true。

			几何 b	
		内部	边界	外部
几何 a	**内部**	1	*	T
	边界	*	*	*
	外部	T	*	*

ST_Overlaps 矩阵 2

ST_Relate

如果模式矩阵指定的空间关系有效，则 ST_Relate 将返回值 1 或 t (true)。 值 1 或 t (true)表明几何之间存在某种空间关系。

如果几何 a 和 b 的内部和边界以任何方式相关联，则 ST_Relate 为 true。 一个几何的外部是否与另一个几何的内部或边界相交是无关紧要的。

			几何 b	
		内部	边界	外部
几何 a	**内部**	T	T	*
	边界	T	T	*
	外部	*	*	*

ST_Relate 矩阵

ST_Touches

如果两个几何的共有点都没有与两个几何的内部相交，则 ST_Touches 将返回 1 或 t (true)。 至少一个几何必须为 ST_LineString、ST_Polygon、ST_MultiLineString 或 ST_MultiPolygon。

当几何的内部不相交而任一几何的边界与另一个几何内部或边界相交时，模式矩阵将显示 ST_Touches 谓词为 true。

如果几何 b 的边界与几何 a 的内部相交，但内部互相不相交，则 ST_Touches 谓词**将返回** true。

			几何 b	
		内部	**边界**	**外部**
几何 a	**内部**	F	T	*
	边界	*	*	*
	外部	*	*	*

ST_Touches 矩阵 1

如果几何 a 的边界与几何 b 的内部相交，但内部互相不相交，则 ST_Touches 谓词**将返回** true。

			几何 b	
		内部	**边界**	**外部**
几何 a	**内部**	F	*	*
	边界	T	*	*
	外部	*	*	*

ST_Touches 矩阵 2

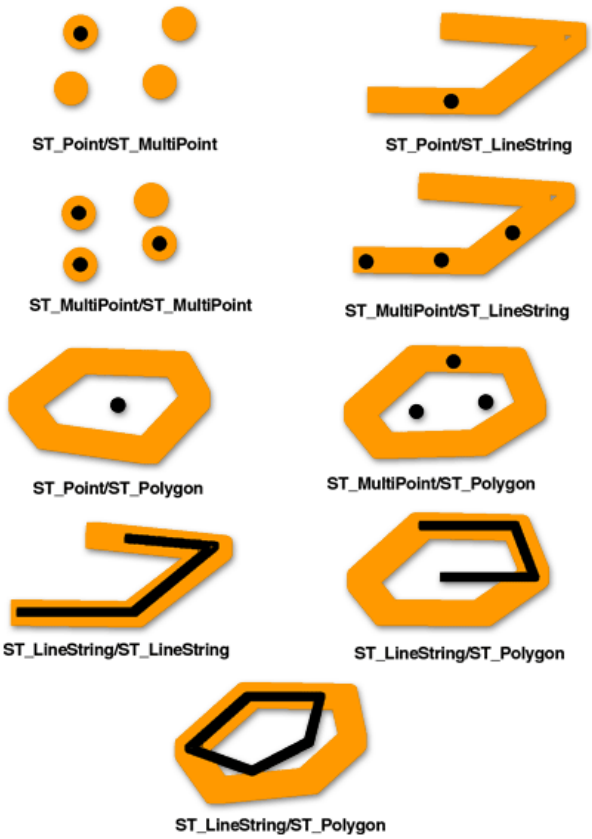
如果两个几何的边界相交，但内部互相不相交，则 ST_Touches 谓词**将将返回** true。

			几何 b	
		内部	**边界**	**外部**
几何 a	**内部**	F	*	*
	边界	*	T	*
	外部	*	*	*

ST_Touches 矩阵 3

ST_Within

如果第一个几何完全位于第二个几何中，则 ST_Within 将返回 1 或 t (true)。 ST_Within 测试与 ST_Contains 完全相反的结果。



ST_Within 谓词模式矩阵表明两个几何的内部必须相交，并且主要几何（几何 a）的内部和边界不能与次要几何（几何 b）的外部相交。

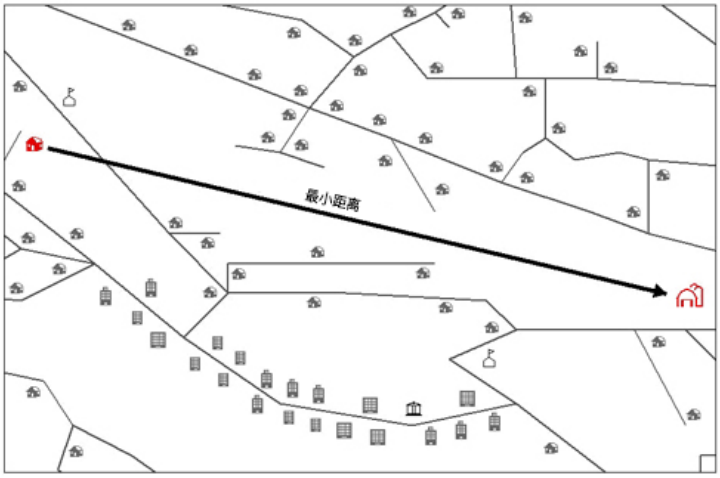
			几何 b	
		内部	边界	外部
几何 a	内部	T	*	F
	边界	*	*	F
	外部	*	*	*

ST_Within 矩阵

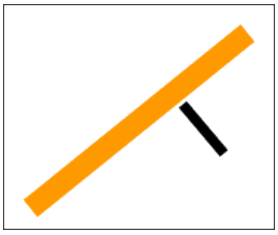
其他空间关系

以下函数将比较几何之间的空间关系，但它们不仅仅比较几何的内部、边界和外部。

- ST_Distance - 此函数将两个不相交的几何作为输入并返回两者之间的最小距离。 如果几何不是不相交的（换句话说，它们是重合的），该函数将报告一个为零的最小距离。
  要素间的最小距离代表两个位置之间的最短距离。 例如，这不是您从一个位置行驶到另一个位置时的行驶距离，而是在地图上的两个位置之间画一条直线时计算的距离。



- ST_DWithin - 您提供距离值以及要比较的几何。 如果几何彼此间距离在指定范围内，则 ST_DWithin 将返回 true。
- ST_EnvIntersects - 此函数评估指定几何的空间包络矩形是否相交；而 ST_Intersects 评估几何本身是否相交。 在以下示例中，两条线的包络矩形相交，但线本身不相交：



- ST_OrderingEquals - 此函数扩展了 ST_Equals 所做的比较，它还可以比较几何坐标是否以相同的顺序定义（x,y 与 y,x）。 即使几何占据相同的空间，如果它们的 x 和 y 坐标未按相同顺序定义，ST_OrderingEquals 也会返回 false。

空间运算

空间运算使用几何函数来获取空间数据并将其作为输入，对其执行分析，然后生成输出数据，输出数据派生自对输入数据执行的分析。

可以从空间运算中获取的派生数据包括：

- 作为输入要素周围缓冲区的面
- 作为对几何集合执行的分析结果的单个要素
- 作为比较结果的单个要素，可确定与另一要素不在同一物理空间的要素部分
- 作为比较结果的单个要素，可查找与另一要素物理空间相交的要素部分
- 由两个输入要素的不同部分组成的多部分要素，这些要素彼此不在同一物理空间中
- 要素是两个几何的并集

对输入数据执行的分析将返回生成几何的坐标或文本表示。 您可以使用此信息作为更大查询的一部分来执行进一步分析，或可以将其结果用作另一表的输入。

例如，您可以在相交查询的 WHERE 子句中包含缓冲区运算，以确定指定的几何是否与另一个几何周围指定大小的区域相交。

注：
以下示例使用了 ST_Geometry 函数。 有关用于其他数据库和空间数据类型的特定几何函数和语法，请阅读特定于该数据库和数据类型的文档。

在此示例中，必须向街道封闭 1,000 英尺范围内的所有业主发送通知。 WHERE 子句在将要封闭的街道周围生成 1,000 英尺的缓冲区。 然后将该缓冲区与该区域中的属性进行比较，以查看哪些属性与缓冲区相交。

```
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';
```

在此示例中，在 WHERE 子句中选择了一条特定街道（主街道），然后在街道周围创建一个缓冲区，并与宗地表中的要素进行比较以确定它们是否相交。* 对于与主街道上的缓冲区相交的所有宗地，将返回宗地所有者名称和地址。

注：
* WHERE 子句各部分的运行顺序取决于数据库优化器。

以下示例中演示了如何通过对包含社区和学区的表执行空间运算（并集）获取结果并将该结果要素插入另一个表的示例：

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

有关将空间运算符与 ST_Geometry 结合使用的详细信息，请参阅 ST_Geometry 的空间运算函数。

ST_Geometry 的空间运算函数

空间运算使用几何函数来获取空间数据并将其作为输入，对其执行分析，然后生成输出数据，输出数据派生自对输入数据执行的分析。

您可以执行以下部分中描述的运算以从输入要素创建要素。

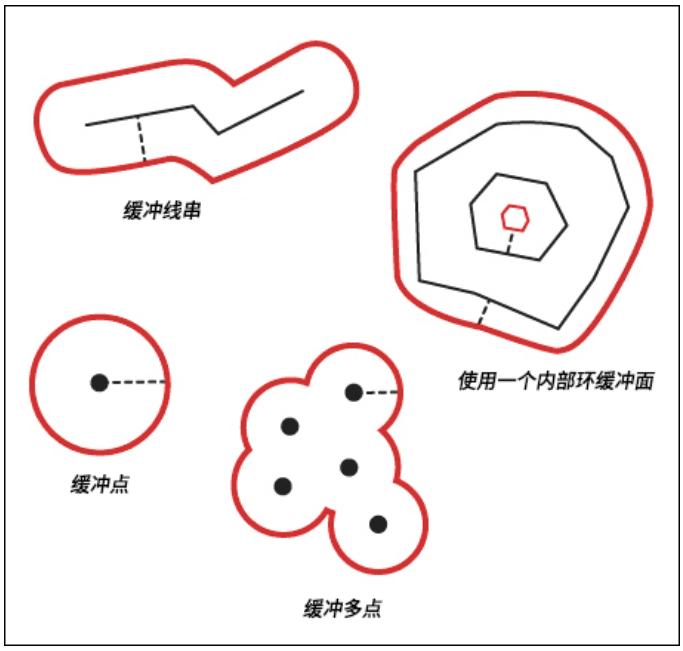
缓冲几何

ST_Buffer 函数通过以指定距离环绕指定几何来生成几何。 当缓冲主几何或集合的缓冲面足够接近重叠时，将生成单个面。 当缓冲集合的元素之间存在足够大的间隔时，单个缓冲 ST_Polygons 会生成 ST_MultiPolygon。

ST_Buffer 函数接受正距离和负距离，但您只能将负距离应用于二维几何（ST_Polygon 和 ST_MultiPolygon）。 当源几何的维度少于二维时，ST_Buffer 将使用缓冲距离的绝对值，换句话说，所有几何都不是 ST_Polygon 或 ST_MultiPolygon。 正缓冲距离会生成远离源几何中心的面环，而对于 ST_Polygon 或 ST_MultiPolygon 的外部环，当距离为负时，会生成朝向中心的面环。 对于 ST_Polygon 或 ST_MultiPolygon 的内部环，当缓冲距离为正时，缓冲环朝向中心，当缓冲距离为负时，缓冲环远离中心。

缓冲过程合并重叠的缓冲面。 大于面最大内部宽度二分之一的负距离会生成空几何。

在以下逻辑示意图中，缓冲区将以红色绘制。



ConvexHull

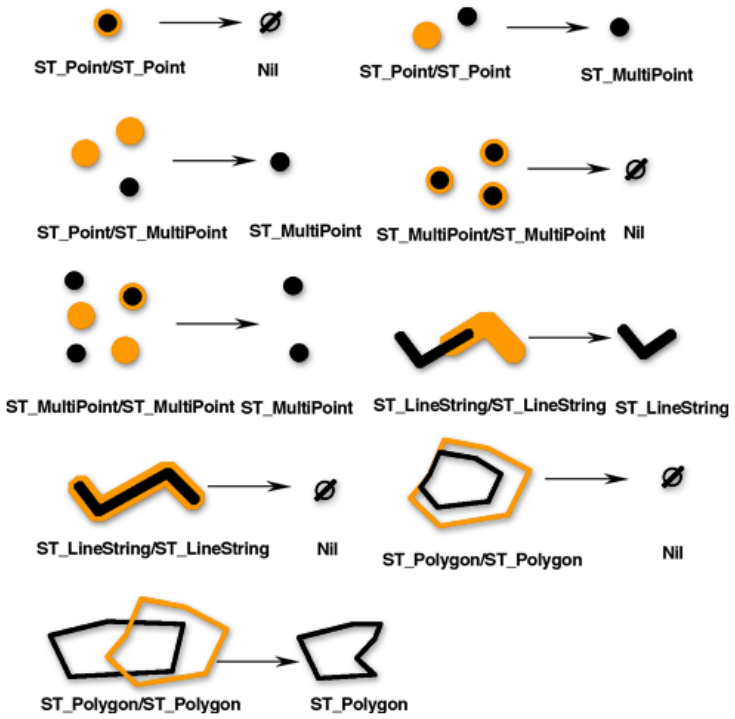
ST_ConvexHull 函数返回任何几何的凸包面，该几何至少有三个折点形成一个凸面。 如果几何的折点不形成凸面，则 ST_ConvexHull 返回 null。 例如，在由两个折点组成的线上使用 ST_ConvexHull 将返回 null。 同样，对点要素使用 ST_ConvexHull 运算将返回 null。 要通过细化一组点来创建不规则三角网 (TIN)，通常需要线创建凸包。

几何的差集

ST_Difference 函数返回未与次要几何相交的主要几何部分。 这是空间的逻辑 AND NOT。

ST_Difference 函数仅对相似维度的几何进行运算，并返回与源几何具有相同维度的集合。 如果源几何相等，则将返回空几何。

在下面的逻辑示意图中，第一个输入几何为黑色，第二个输入几何为橙色。

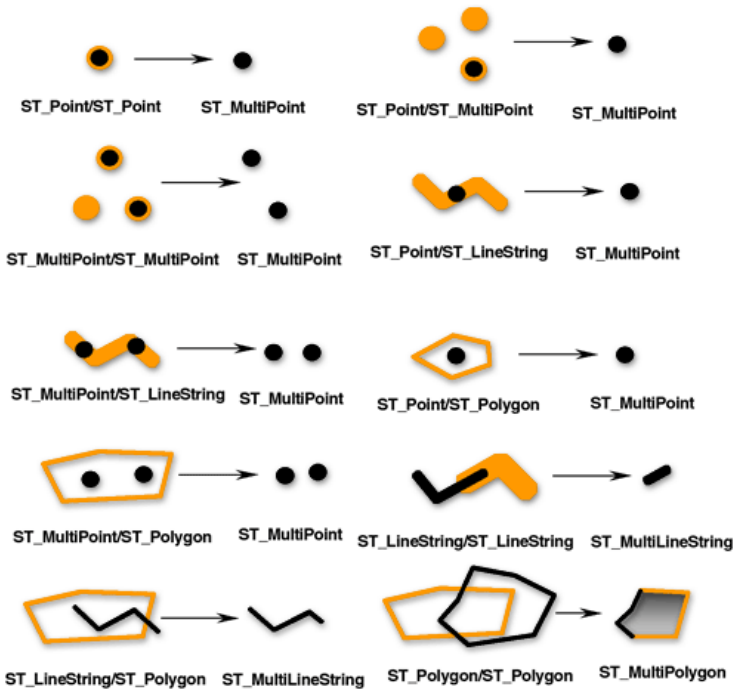


几何的交集

ST_Intersection 函数返回两个几何的交集。 交集始终作为源几何的最小维度的集合返回。

例如，对于与 ST_Polygon 相交的 ST_LineString，ST_Intersection 函数会将 ST_Polygon 内部和边界共有的 ST_LineString 部分作为 ST_MultiLineString 返回。 如果源 ST_LineString 与 ST_Polygon 有两个或多个不连续段相交，则 ST_MultiLineString 包含多个 ST_LineString。 如果几何不相交或者如果相交导致维度小于两个源几何，则返回将空几何。

下图说明了 ST_Intersection 函数的示例。 第一个输入几何为黑色，第二个输入几何为橙色。

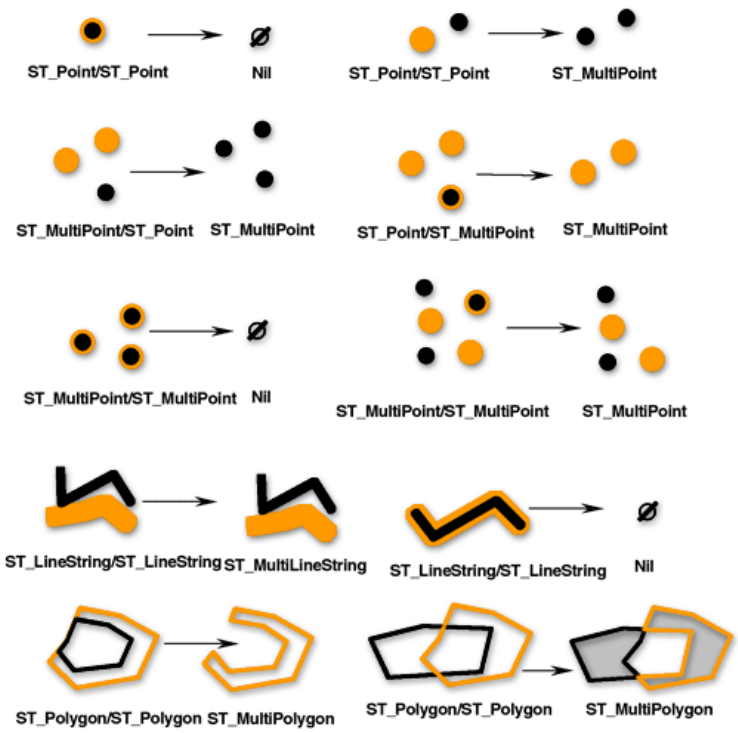


几何的对称差集

ST_SymmetricDiff 函数将返回不属于交集的源几何部分。 这是空间的逻辑 XOR。

源几何必须具有相同的维度。 如果几何相等，则 ST_SymmetricDiff 函数将返回空几何；否则，函数会将结果作为集合返回。

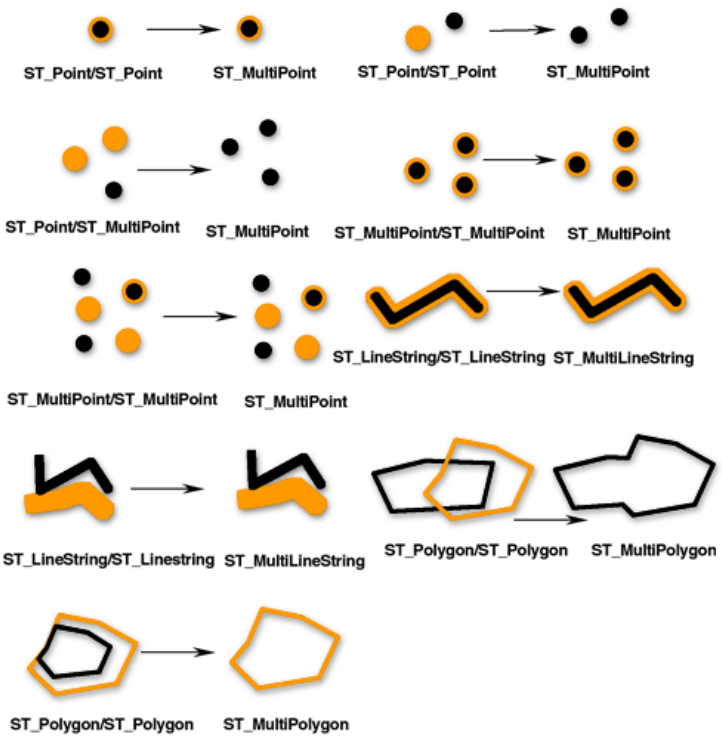
在下面的逻辑示意图中，第一个输入几何为黑色，第二个输入几何为橙色。



几何的并集

ST_Union 函数返回两个几何的并集。 这是空间的布尔逻辑 OR。 源几何必须具有相同的维度。 ST_Union 始终将结果作为集合返回。

在下面的逻辑示意图中，第一个输入几何为黑色，第二个输入几何为橙色。



聚合

作为对几何集合执行的分析结果，聚合运算返回单个几何。 ST_Aggr_ConvexHull 函数返回由每个输入几何的凸包面组成的多面。 任何少于三个折点的输入几何都没有凸包。 如果所有输入几何的折点少于三个，则 ST_Aggr_ConvexHull 将返回 null。

ST_Aggr_Intersection 函数将返回所有输入几何交集聚合的单个几何。

ST_Aggr_Intersection 将查找多个几何的交集，而 ST_Intersection 仅查找两个几何之间的交集。 例如，如需查找由不同特定服务（例如特定学区、电话服务和高速互联网提供商）覆盖并由特定委员会人员代表的财产，您需要找到所有这些区域的交集。 仅查找其中两个区域的交集不会返回您需要的所有信息，因此您将使用 ST_Aggr_Intersection 函数，以便可以在同一查询中评估所有区域。

作为进一步示例，当在两个要素类中查找线和点的交叉点时，每个函数将返回以下值：

- ST_Intersection - 为每个交叉点返回一个 ST_Point 几何。
- ST_Aggr_Intersection - 返回一个由所有交叉点组成的 ST_MultiPoint 几何。 （但是，如果只有一个点要素和一个线要素相交，则您将获得 ST_Point 几何。）

ST_Aggr_Union 函数将返回所有提供几何的并集几何。

输入几何必须具有相同类型；例如，您可以将 ST_LineString 与 ST_LineString 联合，或者您可以将 ST_Polygon 与 ST_Polygon 联合，但您不能将 ST_LineString 要素类与 ST_Polygon 要素类联合。

聚合并集生成的几何通常是一个集合。 例如，如果您想要所有小于半英亩的闲置宗地聚合并集，则将返回多面几何；如果满足条件的所有宗地都是连续的，则将返回一个面。

最小距离

之前的函数返回了新几何。 ST_Distance 函数将执行空间运算（它将评估两个几何之间的最小距离），但不会返回新

几何。

参数圆、参数椭圆和参数楔形

可以使用 ST_Geometry 函数在 ST_Geometry 列中创建并查询参数圆、参数椭圆或参数楔形。

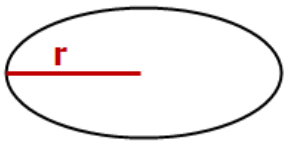
参数圆、参数椭圆和参数楔形是由特定参数（例如坐标值、角度和半径）定义的面。数据库存储的是这些参数，而不是特定的折点和线。比起以多边形表达形式存储的形状，参数形状通过存储用于定义形状的所有参数而使存储更加精确并且所占用的存储空间更少。使用参数形状时还允许将 z 坐标和测量 (m) 值参数包括在内。

在创建圆时需要以下七个参数：

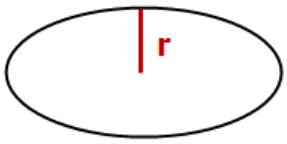
- 圆中心点的 x 坐标值
- 圆中心点的 y 坐标值
- 圆中心点的 z 坐标值
- m 值
- 要创建的圆的半径
- 用于定义圆的点数
  您可指定的最小点数是 9。如果未指定点数，默认情况下使用的数量是 50。这些点不会随形状存储，但会在生成圆时生成，以用于对形状进行验证。
- 在空间中放置圆时使用的空间参考 ID (SRID)

在创建椭圆时需要以下九个参数：

- 椭圆中心点的 x 坐标值
- 椭圆中心点的 y 坐标值
- 椭圆中心点的 z 坐标值
- m 值
- 椭圆的长半轴
  长半轴是椭圆的最长半径。为长半轴指定的值必须大于短半轴的值。



- 椭圆的短半轴
  短半轴是椭圆的最短半径。为短半轴指定的值必须大于 0.0。



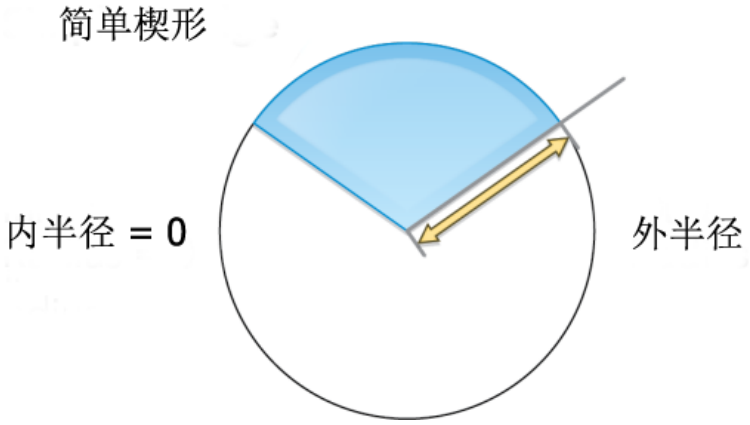
- 椭圆的旋转角度
  为旋转角度指定的值是以度为单位，并且该值必须大于 0.0 且小于 360。按顺时针方向旋转。
- 用于定义椭圆的点数
  您可指定的最小点数是 9。如果未指定点数，默认情况下将使用 50 个点。这些点不会随形状存储，但会在生成椭

圆时生成，以用于对形状进行验证。

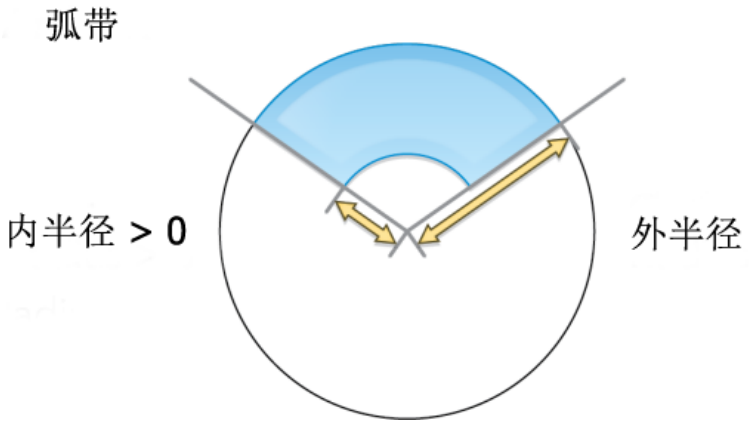
- 在空间中放置椭圆时使用的 SRID

创建楔形时需要以下十个参数：

- 定义楔形的圆的中心点对应的 x 坐标值
- 定义楔形的圆的中心点对应的 y 坐标值
- 定义楔形的圆的中心点对应的 z 坐标值
- m 值
- 楔形的起始角度
  起始角度以从 0 度逆时针测量所得的度数形式定义楔形的起始位置。
- 楔形的终止角度
  终止角度以从 0 度逆时针测量所得的度数形式定义楔形的终止位置。
- 外半径
  外半径定义了从圆心到楔形最外部点的距离。
- 内半径
  内半径定义了从圆心到楔形最内部点的距离，从而定义楔形的起始位置。如果内半径为 0，则该形状为简单楔形。



如果内半径大于 0，则该楔形从技术上讲为弧带。



- 用于定义楔形的点数
  您可指定的最小点数是 9。如果未指定点数，默认情况下将使用 80 个点。这些点不会随形状存储，但会在生成楔

形时生成，以用于对形状进行验证。

- 在空间中放置楔形时使用的 SRID

定义所有半径（包括长半轴和短半轴以及内半径和外半径）时使用的单位取决于 SRID 指定的坐标参考。

有关创建参数圆、参数椭圆或参数楔形的语法和示例，请参阅 ST_Geometry 函数。

ST_Aggr_ConvexHull

注：
仅限 Oracle 和 SQLite

定义

ST_Aggr_ConvexHull 将创建单个几何，该几何是由所有输入几何联合产生的几何凸包。实际上，ST_Aggr_ConvexHull 等同于 ST_ConvexHull(ST_Aggr_Union(geometries))。

语法

Oracle

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

SQLite

```
st_aggr_convexhull (geometry st_geometry)
```

返回类型

Oracle

ST_Geometry

SQLite

Geometryblob

示例

在本例中将创建一个 service_territories 表并运行一个聚合所有几何的 SELECT 语句，从而生成一个表示所有形状联合凸包的几何。

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

注：
仅限 Oracle 和 SQLite

定义

ST_Aggr_Intersection 用于返回一个单一几何，以表示所有输入几何的相交区域的联合。

语法

Oracle

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

SQLite

```
st_aggr_intersection (geometry1 geometryblob)
```

返回类型

Oracle

ST_Geometry

SQLite

Geometryblob

示例

在本示例中，生物学家尝试找到三个野生动物栖息地的相交区域。

Oracle

首先，创建用于存储栖息地的表文件。

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

接着，将三个面插入表中。

```
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)
);
```

最后，选择栖息地的交集。

```
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

首先，创建用于存储栖息地的表文件。

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

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

接着，将三个面插入表中。

```
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)
);
```

最后，选择栖息地的交集。

```
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

定义

ST_Aggr_Union 返回表示所有输入几何的并集的单个几何。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_aggr_union(geometry geometryblob)
```

返回类型

Oracle 和 PostgreSQL

ST_Geometry

SQLite

Geometryblob

示例

市场分析师需要创建表示销售量超过 1,000 的所有服务区的单个几何。在本示例中，您将创建 service_territories1 表并使用销售值数量填充表。然后使用 SELECT 语句中的 st_aggr_union 返回表示销售量等于或大于 1,000 的所有几何的并集的多面。

Oracle 和 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

定义

ST_Area 返回面或多面的面积。

语法

Oracle 和 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)
```

返回类型

双精度型

示例

城市工程师需要一个建筑物面积的列表。为创建该列表，GIS 技术人员选择建筑物 ID 以及每个建筑物的覆盖区的面积。

建筑物覆盖区存储在 bfp 表中。

为满足城市工程师的请求，技术人员从 bfp 表中选择唯一键 building_id 和每个建筑物覆盖区的面积。

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

定义

ST_AsBinary 获取一个几何对象，然后返回其可识别的二进制表示。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_asbinary (geometry geometryblob)
```

返回类型

Oracle 和 PostgreSQL

ST_Geometry

SQLite

Geometryblob

示例

本示例使用记录 1100 中 GEOMETRY 列的内容填充记录 1111 中的 WKB 列。

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

定义

ST_AsText 获取一个几何，然后返回其可识别的文本表示。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_astext (geometry geometryblob)
```

返回类型

Oracle

CLOB

PostgreSQL 和 SQLite

文本

示例

ST_AsText 函数将 hazardous_sites 位置点转换为文本描述。

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

定义

ST_Boundary 采用几何并将其组合边界作为几何对象返回。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_boundary (geometry geometryblob)
```

返回类型

Oracle 和 PostgreSQL

ST_Geometry

SQLite

Geometryblob

示例

在本例中，创建的边界表包含两列：类型和几何。 随后的 INSERT 语句将为每个子类几何添加一条记录。 ST_Boundary 函数用于检索存储在几何列中的每个子类的边界。 请注意，生成的几何尺寸始终比输入几何小 1。 点和多点始终生成空几何边界，维度为 –1。 线串和多线串返回多点边界，维度为 0。 面或多面始终返回多线串边界，维度为 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

定义

ST_Buffer 获取几何对象和距离，然后返回表示围绕源对象的缓冲区的几何对象。

语法

Unit_name 是缓冲距离的测量单位（例如：米、千米、英尺或英里）。 请参阅 Projected coordinate system tables.pdf 中的第一个表，您可以通过坐标系、投影和变换访问该表。

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)
```

返回类型

Oracle 和 PostgreSQL

ST_Geometry

SQLite

Geometryblob

示例

本示例创建两个表 sensitive_areas 和 hazardous_sites；填充这两个表；使用 ST_Buffer 生成围绕 hazardous_sites 表中的面的缓冲区；然后找出这些缓冲区与 sensitive_areas 面重叠的区域。

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

定义

ST_Centroid 以面、多面或多线串要素为输入，然后返回位于几何的包络矩形中心的点。这意味着，质心点在几何的最小和最大 x 和 y 范围的中间位置。

语法

Oracle 和 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)
```

返回类型

ST_Point

示例

城市 GIS 技术人员想要在建筑物密度图中将建筑物覆盖区多面显示为单个点。建筑物覆盖区存储在 bfp 表中，该表由各数据库显示使用的语句所创建并填充。

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

定义

ST_Contains 获取两个几何对象，如果第一个对象完全包含第二个对象，则返回 1（Oracle 和 SQLite）或 t (PostgreSQL)；否则返回 0（Oracle 和 SQLite）或 f (PostgreSQL)。

语法

Oracle 和 PostgreSQL

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

SQLite

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

返回类型

布尔型

示例

在下面的示例中，创建了两个表。一个是包含城市建筑物覆盖区的 bfp；另一个是包含宗地的 lots。城市工程师想要确保所有建筑物覆盖区都完全位于其宗地内。

城市工程师使用 ST_Intersects 和 ST_Contains 来选择并非完全包含在一个地块中的建筑物。

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

定义

ST_ConvexHull 将返回 ST_Geometry 对象的凸包。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_convexhull (geometry1 geometryblob)
```

返回类型

Oracle 和 PostgreSQL

ST_Geometry

SQLite

Geometryblob

示例

这些示例创建了包含以下三列的 sample_geometries 表：id、spatial_type 和 geometry。 spatial_type 字段用于存储在 geometry 列中创建的几何类型。 将在表中插入三个要素：线串、面和多点。

包含 ST_ConvexHull 函数的 SELECT 语句将返回每个几何的凸包。

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

定义

ST_CoordDim 返回几何列的坐标值维度。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_coorddim (geometry1 geometryblob)
```

返回类型

整型

2 = x,y 坐标

3 = x,y,z 或 x,y,m 坐标

4 = x,y,z,m 坐标

示例

在这些示例中，将创建包含 geotype 和 g1 两列的 coorddim_test 表。geotype 列存储 g1 几何列中存储的几何子类和维度的名称。

SELECT 语句列出 geotype 列中所存储的子类名称以及相应几何的坐标维度。

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

定义

ST_Crosses 以两个 ST_Geometry 对象作为输入，如果这两个对象的交集生成的几何对象的维度小于两个源对象中的最大维度，则返回 1（Oracle 和 SQLite）或 t (PostgreSQL)。交集对象所包含的点必须在两个源几何的内部，并且不等于其中任何一个源对象。否则，返回 0（Oracle 和 SQLite）或 f (PostgreSQL)。

语法

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

Oracle 和 PostgreSQL

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

SQLite

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

返回类型

布尔型

示例

县政府正在考虑制定一项新条例，规定县内所有危险废弃物存储设施不能位于任何水道周围指定半径范围内。该县的 GIS 管理人员拥有河流和溪流的精确数据表示，以线串形式存储在 waterways 表中，但他只拥有每个危险废弃物存储设施的单个点位置。

为确定是否必须警告县监管人员现有设施有可能违反提出的条例，GIS 管理人员需要创建 hazardous_sites 位置的缓冲区，以了解是否有河流或溪流穿过缓冲区面。cross 谓词将已建立缓冲区的 hazardous_sites 点与水道进行比较，仅返回水道穿过县提议条例规定半径的记录。

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

注：
仅限 Oracle 和 SQLite

定义

ST_Curve 通过熟知文本表示构造曲线要素。

语法

Oracle

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

SQLite

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

返回类型

ST_LineString

示例

在本例中，将创建一个具有曲线几何的表，在其中插入值，然后从中选择一个要素。

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

定义

ST_Difference 获取两个几何对象，然后返回表示两个源对象之差的几何对象。

语法

Oracle 和 PostgreSQL

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

SQLite

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

返回类型

Oracle 和 PostgreSQL

ST_Geometry

SQLite

Geometryblob

示例

在下面的示例中，城市工程师需要知道未被建筑物覆盖的城市地块的总面积；因此，她想要知道减去建筑物面积之后的地块面积的总和。

城市工程师通过 lot_id 相等连接 footprints 表和 lots 表，然后获取地块减去覆盖区之后的差值面积之和。

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

定义

ST_Dimension 返回几何对象的维度。 在本例中，维度是指长度和宽度。 例如，一个点既没有长度也没有宽度，所以它的维度是0；而一条线有长度但没有宽度，所以它的维度是1。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_dimension (geometry1 geometryblob)
```

返回类型

整型

示例

使用列 geotype 和 g1 创建 dimension_test 表。 geotype 列用于存储在 g1 几何列中存储的子类名称。

SELECT 语句列出了存储在 geotype 列中的子类名称以及该 geotype 的维度。

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

定义

ST_Disjoint 获取两个几何，如果两个几何的交集生成空集，则返回 1（Oracle 和 SQLite）或 t (PostgreSQL)；否则返回 0（Oracle 和 SQLite）或 f (PostgreSQL)。

语法

Oracle 和 PostgreSQL

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

SQLite

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

返回类型

布尔型

示例

在本例中，将创建两个表（distribution_areas 和 factories），并在每个表中插入值。接下来，将在 factories 和 st_disjoint 周围创建缓冲区，以便查找哪些工厂缓冲区没有跨越分布区域。

提示：

在本查询中，可以使用 ST_Intersects 函数代替，只需将函数的结果等于 0，因为 ST_Intersects 和 ST_Disjoint 返回相反的结果。计算查询时，ST_Intersects 函数使用空间索引，而 ST_Disjoint 函数不使用。

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

定义

ST_Distance 用于返回两个几何之间的距离。这一距离是两个几何的最近折点之间的距离。

语法

Oracle 和 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)
```

有效单位名称如下：

毫米	英寸	码	链接
厘米	Inch_US	Yard_US	Link_US
公寸	英尺	Yard_Clarke	Link_Clarke
米	Foot_US	Yard_Sears	Link_Sears
Meter_German	Foot_Clarke	Yard_Sears_1922_Truncated	Link_Sears_1922_Truncated
米	Foot_Sears	Yard_Benoit_1895_A	Link_Benoit_1895_B
50_Kilometers	Foot_Sears_1922_Truncated	Yard_Indian	测链
150_Kilometers	Foot_Benoit_1895_A	Yard_Indian_1937	Chain_US
瓦拉(美制)	Foot_1865	Yard_Indian_1962	Chain_Clarke
斯穆特	Foot_Indian	Yard_Indian_1975	Chain_Sears
	Foot_Indian_1937	英寻	Chain_Sears_1922_Truncated
	Foot_Indian_1962	Mile_US	Chain_Benoit_1895_A
	Foot_Indian_1975	Statute_Mile	杆
	Foot_Gold_Coast	Nautical_Mile	Rod_US
	Foot_British_1936	Nautical_Mile_US	
		Nautical_Mile_UK	

返回类型

双精度型

示例

会创建并填充两个表 study1 和 zones。然后，ST_Distance 函数会用于在使用代码 400 的 study1 区域表中确定各分区和面的边界之间的距离。由于在此形状上有三个区域，因此应返回三个记录。

如果不指定单位，则 ST_Distance 将使用数据投影系统的单位。在第一个示例中，单位为十进制度。在最后两个示例中，指定的单位为千米；因此返回的距离以千米为单位。

Oracle 和 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

定义

ST_DWithin 将两个几何作为输入，如果几何彼此间距离在指定范围内则返回 true；否则将返回 false。 几何的空间参考系统决定指定距离将应用的测量单位。 因此，提供给 ST_DWithin 的几何必须使用同样的坐标投影和空间参考 ID (SRID)。

语法

Oracle 和 PostgreSQL

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

SQLite

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

返回类型

布尔

示例

在下例中，创建了两张表，并向其中插入了要素。 接下来，在两个不同的 SELECT 语句中使用 ST_DWithin 函数：一个用于确定第一个表中的点是否在第二个表中 100 米的面范围内，另一个用于确定彼此相距 300 米的要素。

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)
 )
;
```

然后，使用 ST_DWithin 确定每个表中有哪些要素彼此相距 100 米以内，而哪些不是。 将 ST_Distance 函数包含在此语句中，以便显示要素间的实际距离。

```
--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;
```

此语句返回如下内容：

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

在下例中，ST_DWithin 用来确定彼此相距 300 米以内的各个要素：

```
--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;
```

运行 Oracle 中的数据时，第二个 SELECT 语句返回以下内容：

```
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)
 )
;
```

然后，使用 ST_DWithin 确定每个表中有哪些要素彼此相距 100 米以内，而哪些不是。 将 ST_Distance 函数包含在此语句中，以便显示要素间的实际距离。

```
--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;
```

此语句返回如下内容：

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

在下例中，ST_DWithin 用来确定彼此相距 300 米以内的各个要素：

```
--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;
```

第二个选择语句返回如下内容：

```
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)
 )
;
```

然后，使用 ST_DWithin 确定每个表中有哪些要素彼此相距 100 米以内，而哪些不是。 将 ST_Distance 函数包含在此语句中，以便显示要素间的实际距离。

```
--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;
```

此语句返回如下内容：

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

在下例中，ST_DWithin 用来确定彼此相距 300 米以内的各个要素：

```
--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;
```

第二个选择语句返回如下内容：

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

```
2|2|201.69531476958|1
```

ST_EndPoint

定义

ST_EndPoint 返回线串的最后一个点。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_endpoint (line1 geometryblob)
```

返回类型

ST_Point

示例

endpoint_test 表存储唯一标识各行的 gid integer 列和存储线串的 ln1 ST_LineString 列。

INSERT 语句将线串插入 endpoint_test 表中。第一个线串没有 z 坐标或测量值，而第二个线串有。

查询将列出 gid 列和 ST_EndPoint 函数所生成的 ST_Point 几何。

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

定义

ST_Entity 返回几何对象的空间实体类型。空间实体类型是存储在几何对象的实体成员字段中的值。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_entity (geometry1 geometryblob)
```

返回类型

将返回数字 (Oracle) 或整数（SQLite 和 PostgreSQL）来表示以下实体类型：

0	nil 形状
1	点
2	线（包括 spaghetti 线）
4	线串
8	面/区域
257	多点
258	多线（包括 spaghetti 线）
260	多线串
264	多区域

示例

以下示例会创建一个表并向表中插入不同的几何。ST_Entity 随即在表上运行并返回表中各记录的几何子类型。

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;
```

SELECT 语句返回以下值：

```
 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;
```

SELECT 语句返回以下值：

```
 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;
```

SELECT 语句返回以下值：

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

ST_Envelope

定义

ST_Envelope 将几何对象的最小边界框作为面返回。

📚 抢先版本：
此函数符合声明 ST_Envelope 返回面的 Open Geospatial Consortium (OGC) 简单要素规范。 要处理点几何或水平或垂直线的特殊情况，ST_Envelope 函数将返回围绕这些形状的面，这是根据几何空间参考系统 XY 比例因子计算的较小包络容差。 从最小 x 和 y 坐标中减去此容差，并在最大 x 和 y 坐标中加上此容差，以返回围绕这些形状的面。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_envelope (geometry1 geometryblob)
```

返回类型

Oracle 和 PostgreSQL

ST_Geometry

SQLite

Geometryblob

示例

envelope_test 表的 geotype 列用于存储在 g1 列中存储的几何子类的名称。 INSERT 语句可将每个几何子类插入到 envelope_test 表中。

接下来，运行 ST_Envelope 函数以返回每个几何周围的面包络。

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

注：
仅限 Oracle 和 SQLite

定义

如果两个几何的包络矩形相交，则 ST_EnvIntersects 返回 1 (true)；否则，返回 0 (false)。

语法

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)
```

返回类型

布尔型

示例

本示例将搜索包络矩形与定义面相交的几何。

第一个 SELECT 语句对两个几何的包络矩形以及几何自身进行比较，以查看要素或包络矩形是否相交。

第二个 SELECT 语句使用一个包络矩形来检测哪些要素（如果存在）落入在通过 SELECT 语句的 WHERE 子句进行传递的包络矩形内部。

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

定义

ST_Equals 比较两个几何，如果这两个几何完全相同，则返回 1（Oracle 和 SQLite）或 t (PostgreSQL)；否则返回 0（Oracle 和 SQLite）或 f (PostgreSQL)。

语法

Oracle 和 PostgreSQL

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

SQLite

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

返回类型

布尔型

示例

城市 GIS 技术人员怀疑 studies 表中的某些数据存在重复。为减轻顾虑，他查询该表，以确定是否存在相等的形状多面。

通过以下语句创建并填充 studies 表。id 列唯一标识研究区域，而形状字段存储区域的几何。

接下来，通过 equal 谓词将 studies 表与其本身进行空间连接，只要发现两个多面相等，就会返回 1（Oracle 和 SQLite）或 t (PostgreSQL)。s1.id<>s2.id 条件不用将几何与其本身进行比较。

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

注：
仅限 PostgreSQL

定义

ST_Equalsrs 用于检查两个不同要素类的两个空间参考系统是否相同。 如果空间参考系统相同，则返回 t (true)。 如果空间参考系统不同，则返回 f (false)。

语法

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

返回类型

布尔

示例

在本例中，发现了不同要素类的空间参考 ID (SRID)，然后使用 ST_Equalsrs 来查看 SRID 是否代表相同的空间参考系统。

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

srid	table_name
2	streets
6	transmains

sde_layers 查询结果

现在，请使用 ST_Equalsrs 来确定这两个 SRID 标识的空间参考系统是否相同。

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

 st_equalsrs

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

f

(1 row)
```

ST_ExteriorRing

定义

ST_ExteriorRing 以线串形式返回面的外部环。

语法

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

Oracle 和 PostgreSQL

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

SQLite

```
st_exteriorring (polygon1 geometryblob)
```

返回类型

ST_LineString

示例

鸟类学家想要研究多个岛屿上的鸟类种群，了解到的情况是她所感兴趣的鸟类物种的摄食区域限制在岸线地带。在计算岛屿的承载能力过程中，鸟类学家需要知道岛屿的周长。一些岛屿很大，上面有几个湖。但是，湖的岸线地带被另一个更具侵略性的鸟类物种所独占栖息。因此，鸟类学家只需要知道岛屿的外部环的周长。

islands 表的 ID 列和名称列标识每个岛屿，而陆地面列存储岛屿的几何。

ST_ExteriorRing 函数从每个岛屿面提取线串形式的外部环。线串的长度使用 ST_Length 函数计算。线串长度使用 SUM 函数汇总。

岛屿的外部环表示每个岛屿与海洋共有的生态分界面。

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

注：
仅限 Oracle 和 PostgreSQL

定义

ST_GeomCollection 通过熟知文本表示构造几何集合。

语法

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)
```

返回类型

ST_GeomCollection

示例

Oracle

创建一个表 geomcoll_test，并在其中插入几何。

```
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)
);
```

从 geomcoll_test 表中选择几何集合。

```
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

创建一个表 geomcoll_test，并在其中插入几何。

```
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)
);
```

从 geomcoll_test 表中选择几何集合。

```
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

注：
仅限 PostgreSQL

定义

ST_GeomCollFromWKB 通过熟知二进制表示来构造一个几何集合。

语法

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

返回类型

ST_GeomCollection

示例

注：
插入硬回车，以增强可读性。如果复制了语句，则会将其移除。

创建表 gcoll_test。

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

将值插入到表中。

```
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));
```

从 gcoll_test 表中选择几何。

```
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

定义

ST_Geometry 通过熟知文本表示构造几何。

注：
创建将与 ArcGIS 配合使用的空间表时，最好将列创建为几何超类型（例如 ST_Geometry），而不是指定 ST_Geometry 子类型。

语法

Oracle

- 用于线串、面和点

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

- 用于优化点（不启动 extproc 代理，因此可以更快地处理查询）

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

在执行大量点数据的批量插入时使用优化点构造。

- 用于参数圆

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

- 用于参数椭圆

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

- 用于参数楔形

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

PostgreSQL

- 用于线串、面和点

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

- 用于参数圆

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

- 用于参数椭圆

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

- 用于参数楔形

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

SQLite

- 用于线串、面和点

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

- 用于参数圆

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

- 用于参数椭圆

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

- 用于参数楔形

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

返回类型

Oracle 和 PostgreSQL

ST_Geometry

SQLite

Geometryblob

示例

创建和查询点、线串或面要素

在这些示例中，将创建一个表 (geoms)，并在其中插入点、线串和面值。

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)
);
```

创建和查询参数圆

创建一个表 radii，并在其中插入圆。

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)
);
```

创建和查询参数椭圆

创建一个表 track，并在其中插入椭圆。

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)
);
```

创建和查询参数楔形

创建一个表 pwedge，并在其中插入楔形。

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

定义

ST_GeometryN 获取一个集合和一个整数索引，然后返回集合中的第 n 个 ST_Geometry 对象。

语法

Oracle 和 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)
```

返回类型

Oracle 和 PostgreSQL

ST_Geometry

SQLite

Geometryblob

示例

在本示例中，将创建一个多面。然后使用 ST_GeometryN 列出多面的第二个元素。

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

定义

ST_GeometryType 采用几何对象并将其几何类型（例如，点、线、面、多点）作为字符串返回。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_geometrytype (g1 geometryblob)
```

返回类型

Varchar(32)（Oracle 和 PostgreSQL）或 text (SQLite) 包含以下内容之一：

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

示例

geometrytype_test 表包含 g1 几何列。

INSERT 语句可将每个几何子类插入到 g1 列中。

SELECT 查询列出了存储在 g1 几何列中的各个子类的几何类型。

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

注：
仅限 PostgreSQL

定义

ST_GeomFromCollection 返回 st_geometry 行集。每行包含几何和整数。整数表示几何在该行集中的位置。

使用 ST_GeomFromCollection 函数访问多部分几何中的各个几何。当输入几何为集合或多部分几何（例如：ST_MultiLineString、ST_MultiPoint、ST_MultiPolygon）时，ST_GeomFromCollection 将返回每个集合组件的记录，同时路径即表示组件在集合中的位置。

如果对简单几何（例如：ST_Point、ST_LineString、ST_Polygon）使用 ST_GeomFromCollection，由于仅有一个几何，因此将返回路径为空的单一记录。

语法

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

要仅返回几何，请使用 `(sde.st_geomfromcollection (shape)).st_geo`。

要仅返回几何位置，请使用 `(sde.st_geomfromcollection (shape)).path[1]`。

返回类型

ST_Geometry 集合

示例

在此示例中，将创建多个要素类 (ghanasharktracks)，其中包含具有四部分形状的单一要素。

```
--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
  )
 );
```

要确认字段包含数据，请查询表。在形状字段中直接输入 ST_AsText，将形状坐标视为文本。请注意，返回多线串的文本描述。

```
--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))"
```

要单独返回每个线串几何，请使用 ST_GeomFromCollection 函数。要将几何视为文本，本示例将 ST_GeomFromCollection 函数与 ST_AsText 函数一起使用。

```
--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

注：
仅用于 Oracle 和 SQLite；对于 PostgreSQL，请使用 ST_Geometry。

定义

ST_GeomFromText 以熟知文本表示和空间参考 ID 作为输入参数，并返回几何对象。

语法

Oracle

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

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

如果您未指定 SRID，则空间参考默认为 4326。

SQLite

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

```
st_geomfromtext (wkt text)
```

如果您未指定 SRID，则空间参考默认为 4326。

返回类型

Oracle

ST_Geometry

SQLite

Geometryblob

示例

geometry_test 表包含整型 gid 列，它唯一标识了每个行和存储几何的 g1 列。

INSERT 语句可将数据插入到 geometry_test 表的 gid 和 g1 列中。 ST_GeomFromText 函数可将每个几何的文本表示转换为其对应的可实例化子类。 执行了结尾处的 SELECT 语句以确保将数据插入到 g1 列中。

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

定义

ST_GeomFromWKB 以熟知二进制 (WKB) 表示和空间参考 ID 作为输入参数，返回几何对象。

语法

Oracle

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

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

如果您未指定 SRID，则空间参考默认为 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)
```

如果您未指定 SRID，则空间参考默认为 4326。

返回类型

Oracle 和 PostgreSQL

ST_Geometry

SQLite

Geometryblob

示例

在以下示例中，对结果行进行了重新格式化，以增强可读性。结果中的间距将根据在线显示而有所不同。以下代码说明了如何使用 ST_GeomFromWKB 函数由 WKB 线表示来创建和插入线。以下示例通过 WKB 表示的空间参考系统 4326 中的 ID 和几何将记录插入到 sample_gs 表中。

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

注：
仅限 PostgreSQL

定义

ST_GeoSize 采用 ST_Geometry 对象作为输入，以字节返回其大小。

语法

```
st_geosize (st_geometry)
```

返回类型

整型

示例

可通过查询 sample_geometries 表的几何列查看在 ST_GeomFromWKB 示例中创建的要素的大小。

```
SELECT st_geosize (geometry)
FROM sample_geometries;

st_geosize
        512
        592
        616
```

ST_InteriorRingN

定义

ST_InteriorRingN 以 ST_LineString 形式返回面的第 n 个内部环。

环的顺序不能预先定义，因为环是按照由内部几何验证例程定义的规则进行组织，而不是按照由几何方向定义的规则进行组织。如果索引超出了面所拥有的内部环数，则返回空值。

语法

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)
```

返回类型

ST_LineString

示例

创建表 sample_polys，并添加一条记录，然后选择内部环的 ID 和几何。

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

定义

ST_Intersection 以两个几何对象作为输入参数，然后以二维几何对象的形式返回交集。

语法

Oracle 和 PostgreSQL

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

SQLite

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

返回类型

Oracle 和 PostgreSQL

ST_Geometry

SQLite

Geometryblob

示例

消防局局长必须获得医院、学校和疗养院与可能的危险废弃物污染范围的交集区域。

医院、学校和疗养院存储在使用后面的 CREATE TABLE 语句创建的 population 表中。定义为 polygon 的 shape 列存储每个敏感区域的轮廓。

危险场地存储在使用后面的 CREATE TABLE 语句创建的 waste_sites 表中。定义为 point 的 site 列存储表示每个危险场地的地理中心的位置。

ST_Buffer 函数生成一个环绕危险废弃物场地的缓冲区域。ST_Intersection 函数将生成表示已创建缓冲区的危险废弃物场地与敏感区域的交集的面。

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

定义

如果两个几何的交集不生成空集，则 ST_Intersects 返回 1（Oracle 和 SQLite）或 t (PostgreSQL)；否则返回 0（Oracle 和 SQLite）或 f (PostgreSQL)。

语法

Oracle 和 PostgreSQL

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

SQLite

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

返回类型

布尔型

示例

消防局局长想要获得在危险废弃物场地的某一半径范围内的敏感区域的列表。

敏感区域存储在 sensitive_areas 表中。定义为 polygon 的 shape 列存储每个敏感区域的轮廓。

危险场地存储在 hazardous_sites 表中。定义为 point 的 site 列存储表示每个危险场地的地理中心的位置。

SELECT 查询在每个危险场地周围创建缓冲半径区，并返回与危险场地缓冲区相交的敏感区域列表。

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

定义

ST_Is3d 将 ST_Geometry 作为输入参数，如果给定几何具有 z 坐标，则返回 1（Oracle 和 SQLite）或 t (PostgreSQL)；否则，将返回 0（Oracle 或 SQLite）或 f (PostgreSQL)。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_is3d (geometry1 geometryblob)
```

返回类型

布尔

示例

在本例中，创建了一个表 is3d_test，并用记录对其进行了填充。

接下来，使用 ST_Is3d 检查是否有任何记录具有 z 坐标。

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

定义

ST_IsClosed 以 ST_LineString 或 ST_MultiLineString 作为输入参数，如果为闭合，则返回 1（Oracle 和 SQLite）或 t (PostgreSQL)；否则返回 0（Oracle 和 SQLite）或 f (PostgreSQL)。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_isclosed (geometry1 geometryblob)
```

返回类型

布尔型

示例

测试线串

创建一个包含单一 linestring 列的 closed_linestring 表。

INSERT 语句将两条记录插入 closed_linestring 表中。第一条记录不是闭合线串，而第二条记录是闭合线串。

查询将返回 ST_IsClosed 函数的结果。因为第一行的线串不闭合，所以返回 0 或 f；而第二行的线串是闭合的，所以返回 1 或 t。

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
```

测试多线串

创建一个包含单一 ST_MultiLineString 列的 closed_mlinestring 表。

INSERT 语句将分别插入一条闭合的和一条不闭合的 ST_MultiLineString 记录。

查询将列出 ST_IsClosed 函数的结果。因为第一行的多线串不闭合，所以返回 0 或 f。而第二行 ln1 列中存储的多线串是闭合的，所以返回 1 或 t。如果多线串的所有线串元素都闭合，则多线串是闭合的。

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

定义

如果 ST_Geometry 对象为空，则 ST_IsEmpty 返回 1（Oracle 和 SQLite）或 t (PostgreSQL)；否则返回 0（Oracle 和 SQLite）或 f (PostgreSQL)。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_isempty (geometry1 geometryblob)
```

返回类型

布尔型

示例

下面的 CREATE TABLE 语句将创建 empty_test 表，其中包含的 geotype 用于存储 g1 列中所存储的子类的数据类型。

此 INSERT 语句将向几何子类（点、线串和面）中插入两条记录：一条记录为空，另一条记录不为空。

此 SELECT 查询将返回 geotype 列中的几何类型和 ST_IsEmpty 函数的结果。

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

定义

ST_IsMeasured 将几何对象作为输入参数，如果给定几何具有测量值，则返回 1（Oracle 和 SQLite）或 t (PostgreSQL)；否则，将返回 0（Oracle 和 SQLite）或 f (PostgreSQL)。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_ismeasured (geometry1 geometryblob)
```

返回类型

布尔

示例

创建表 ism_test，向其中插入值，然后确定 ism_test 表中的哪些行包含测量值。

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

定义

ST_IsRing 以 ST_LineString 作为输入参数，如果是环（如 ST_LineString 是闭合的简单线串），则返回 1（Oracle 和 SQLite）或 t (PostgreSQL)；否则返回 0（Oracle 和 SQLite）或 f (PostgreSQL)。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_isring (line1 geometryblob)
```

返回类型

布尔型

示例

创建一个包含单个 ST_LineString 列 ln1 的 ring_linestring 表。

INSERT 语句将三个线串插入 ln1 列中。第一行包含一个不闭合不是环的线串。第二行包含一个闭合的简单线串（线串是环）。第三行包含一个闭合的非简单线串，因为线串与自己的内部相交。此线串也不是环。

SELECT 查询将返回 ST_IsRing 函数的结果。因为第一行的线串不是环，所以返回 0 或 f；而第二行和第三行的线串是环，所以返回 1 或 t。

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

定义

如果按照开放地理空间联盟 (OGC) 的定义，几何对象是简单对象，则 ST_IsSimple 返回 1（Oracle 和 SQLite）或 t (PostgreSQL)；否则返回 0（Oracle 和 SQLite）或 f (PostgreSQL)。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_issimple (geometry1 geometryblob)
```

返回类型

布尔型

示例

创建一个包含两列的表 issimple_test。pid 列是 smallint 数据类型，包含每行的唯一标识符。g1 列存储的是简单和非简单的几何示例。

此 INSERT 语句用于向 issimple_test 表中插入两条记录。第一条记录是一个简单线串，因为它不与内部相交。按照 OGC 的定义，第二条是非简单线串，因为它与内部相交。

此查询将返回 ST_IsSimple 函数的结果。因为第一条记录的线串是简单线串，所以返回 1 或 t；而第二条记录的线串是非简单线串，所以返回 0 或 f。

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

定义

ST_Length 用于返回线串或多线串的长度。

语法

Oracle 和 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)
```

有关支持的单位名称列表，请参阅 ST_Distance。

返回类型

双精度型

示例

生态学家研究当地水道中鲑鱼种群的迁移模式时，需要获取当地所有溪流与河流系统的长度。

将创建一个包含 ID 和名称列的水道表，这些列标识了表格中所存储的各溪流和河流系统。水体列是多线串类型，因为溪流与河流系统常常是多个线串的聚合体。

SELECT 查询将返回系统名称以及 ST_Length 函数生成的系统长度。在 Oracle 和 PostgreSQL 中，使用坐标系中的单位。在 SQLite 中，指定以千米为单位。

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

注：
仅 Oracle 和 SQLite 支持；对于 PostgreSQL，请使用 ST_LineString。

定义

ST_LineFromText 以 ST_LineString 类型的熟知文本表示和空间参考 ID 作为输入，并返回 ST_LineString 类型的对象。

语法

Oracle

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

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

如果您未指定 SRID，则空间参考默认为 4326。

SQLite

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

```
st_linefromtext (wkt text)
```

如果您未指定 SRID，则空间参考默认为 4326。

返回类型

ST_LineString

示例

创建一个包含单个 ln1 ST_LineString 列的 linestring_test 表。

此 INSERT 语句使用 ST_LineFromText 函数将 ST_LineString 插入到 ln1 列中。

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

定义

ST_LineFromWKB 以 ST_LineString 类型的熟知二进制 (WKB) 表示和空间参考 ID 作为输入，返回 ST_LineString 类型的对象。

语法

Oracle

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

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

如果您未指定 SRID，则空间参考默认为 4326。

PostgreSQL

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

SQLite

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

```
st_linefromwkb (wkb blob)
```

如果您未指定 SRID，则空间参考默认为 4326。

返回类型

ST_LineString

示例

以下命令创建表 (sample_lines)，并使用 ST_LineFromWKB 函数从 WKB 表示插入线。该行将被插入到包含 ID 和使用 WKB 表示的空间参考系统 4326 中的线的 sample_lines 表中。

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

定义

ST_LineString 存取器函数用于从通过熟知文本表示构造线串。

注：
创建将与 ArcGIS 配合使用的空间表时，最好将列创建为几何超类型（例如 ST_Geometry），而不是指定 ST_Geometry 子类型。

语法

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)
```

返回类型

ST_LineString

示例

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

定义

ST_M 以 ST_Point 对象为输入参数，返回其测量 (m) 坐标。

在 SQLite 中，ST_M 也可以用于更新测量值。

语法

Oracle 和 PostgreSQL

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

SQLite

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

返回类型

Oracle 和 PostgreSQL

数值

SQLite

查询测量值时返回双精度型；更新测量值时返回 geometryblob

示例

Oracle

创建表 m_test，并向其中插入三个点。所有三个点均包含测量值。SELECT 语句与 ST_M 函数共同运行，以返回每个点的测量值。

```
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

创建表 m_test，并向其中插入三个点。所有三个点均包含测量值。SELECT 语句与 ST_M 函数共同运行，以返回每个点的测量值。

```
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

在第一个示例中创建表 m_test，并向其中插入三个点。所有三个点均包含测量值。SELECT 语句与 ST_M 函数共同运行，以返回每个点的测量值。

```
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
```

在第二个示例中，更新了 m_test 表中记录 3 的测量值。

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

ST_MaxM

定义

ST_MaxM 以几何对象为输入参数，返回其最大 m 坐标。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_maxm (geometry1 geometryblob)
```

返回类型

Oracle 和 PostgreSQL

数值

如果 m 值不存在，则返回 NULL。

SQLite

双精度型

如果 m 值不存在，则返回 NULL。

示例

创建表 maxm_test，并向其中插入两个面。然后运行 ST_MaxM，以确定每个面的最大 m 值。

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

定义

ST_MaxX 以几何对象为输入参数，返回其最大 x 坐标。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_maxx (geometry1 geometryblob)
```

返回类型

Oracle 和 PostgreSQL

数值

SQLite

双精度型

示例

创建表 maxx_test，并向其中插入两个面。然后，使用 ST_MaxM 函数确定每个面的最大 x 值。

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

定义

ST_MaxY 以几何对象为输入参数，返回其最大 y 坐标。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_maxy (geometry1 geometryblob)
```

返回类型

Oracle 和 PostgreSQL

数值

SQLite

双精度型

示例

创建表 maxy_test，并向其中插入两个面。然后，使用 ST_MaxY 函数确定每个面的最大 y 值。

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

定义

ST_MaxZ 以几何对象为输入参数，返回其最大 z 坐标。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_maxz (geometry1 geometryblob)
```

返回类型

Oracle 和 PostgreSQL

数值

如果 z 值不存在，则返回 NULL。

SQLite

双精度型

如果 z 值不存在，则返回 NULL。

示例

在下例中，创建了表 maxz_test，并向其中插入两个面。然后运行 ST_MaxZ 以返回每个面的最大 z 值。

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

定义

ST_MinM 以几何为输入参数，返回其最小 m 坐标。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_minm (geometry1 geometryblob)
```

返回类型

Oracle 和 PostgreSQL

数值

如果 m 值不存在，则返回 NULL。

SQLite

双精度型

如果 m 值不存在，则返回 NULL。

示例

创建表 minm_test，并向其中插入两个面。然后运行 ST_MinM，以确定每个面的最小测量值。

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

定义

ST_MinX 以几何对象为输入参数，返回其最小 x 坐标。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_minx (geometry1 geometryblob)
```

返回类型

Oracle 和 PostgreSQL

数值

SQLite

双精度型

示例

创建表 minx_test，并向其中插入两个面。然后运行 ST_MinX，以确定每个面的最小 x 坐标值。

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

定义

ST_MinY 以几何对象为输入参数，返回其最小 y 坐标。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_miny (geometry1 geometryblob)
```

返回类型

Oracle 和 PostgreSQL

数值

SQLite

双精度型

示例

创建表 miny_test，并向其中插入两个面。然后运行 ST_MinY，以确定每个面的最小 y 坐标值。

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

定义

ST_MinZ 以几何对象为输入参数，返回其最小 z 坐标。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_minz (geometry1 geometryblob)
```

返回类型

Oracle 和 PostgreSQL

数值

如果 z 值不存在，则返回 NULL。

SQLite

双精度型

如果 z 值不存在，则返回 NULL。

示例

创建表 minz_test，并向其中插入两个面。然后运行 ST_MinZ，以确定每个面的最小 z 坐标值。

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

注：
仅用于 Oracle 和 SQLite；对于 PostgreSQL，请使用 ST_MultiLineString。

定义

ST_MLineFromText 以 ST_MultiLineString 类型的熟知文本表示和空间参考 ID 作为输入参数，并返回 ST_MultiLineString 类型的对象。

语法

Oracle

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

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

如果您未指定 SRID，则空间参考默认为 4326。

SQLite

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

```
st_mlinefromtext (wkt text)
```

如果您未指定 SRID，则空间参考默认为 4326。

返回类型

ST_MultiLineString

示例

创建 mlinestring_test 表，其中包含用于唯一标识行的 gid smallint 列以及 ml1 ST_MultiLineString 列。

此 INSERT 语句使用 ST_MLineFromText 函数插入 ST_MultiLineString。

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

定义

ST_MLineFromWKB 以 ST_MultiLineString 类型的熟知二进制 (WKB) 表示和空间参考 ID 为输入参数，以创建 ST_MultiLineString 类型的对象。

语法

Oracle

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

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

如果您未指定 SRID，则空间参考默认为 4326。

PostgreSQL

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

SQLite

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

```
st_mlinefromwkb (wkb blob)
```

如果您未指定 SRID，则空间参考默认为 4326。

返回类型

ST_MultiLineString

示例

本示例说明了如何使用 ST_MLineFromWKB 由熟知二进制表示创建多线串。几何是空间参考系统 4326 中的一个多线串。在本示例中，多线串存储在 sample_mlines 表的几何列中且 ID = 10，然后利用熟知二进制表示对 wkb 列进行更新（使用 ST_AsBinary 函数）。最后，ST_MLineFromWKB 函数用于从 wkb 列中返回多线串。sample_mlines 表具有一个几何列（用于存储多线串）和一个 wkb 列（用于存储多线串的 WKB 表示）。

SELECT 语句包含 ST_MLineFromWKB 函数，该函数用于从 wkb 列检索多线串。

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

注：
仅限 Oracle 和 SQLite；对于 PostgreSQL，请使用 ST_MultiPoint。

定义

ST_MPointFromText 采用 ST_MultiPoint 类型的熟知文本 (WKT) 表示和空间参考 ID，并创建 ST_Multipoint。

语法

Oracle

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

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

如果您未指定 SRID，则空间参考默认为 4326。

SQLite

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

```
st_mpointfromtext (wkt text)
```

如果您未指定 SRID，则空间参考默认为 4326。

返回类型

ST_MultiPoint

示例

使用单个 ST_MultiPoint mpt1 列创建 multipoint_test 表。

INSERT 语句使用 ST_MpointFromText 函数将多点插入到 mpt1 列中。

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

定义

ST_MPointFromText 采用 ST_MultiPoint 类型的熟知二进制 (WKB) 表示和空间参考 ID，并创建 ST_MultiPoint。

语法

Oracle

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

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

如果您未指定 SRID，则空间参考默认为 4326。

PostgreSQL

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

SQLite

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

```
st_mpointfromwkb (wkb blob)
```

如果您未指定 SRID，则空间参考默认为 4326。

返回类型

ST_MultiPoint

示例

此示例说明了如何使用 ST_MPointFromWKB 从熟知的二进制表示创建多点。 几何是空间参考系统 4326 中的多点。 在本示例中，多点存储在 SAMPLE_MPOINTS 表的几何列中且 ID = 10，然后利用熟知二进制表示对 WKB 列进行更新（使用 ST_AsBinary 函数）。 最后，ST_MPointFromWKB 函数用于从 WKB 列返回多点。 SAMPLE_MPOINTS 表具有一个 GEOMETRY 列（用于存储多点）和一个 WKB 列（用于存储多点的熟知二进制表示）。

在以下 SELECT 语句中，ST_MPointFromWKB 函数用于从 WKB 列中检索多点。

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

注：
仅限 Oracle 和 SQLite；对于 PostgreSQL，请使用 ST_MultiPolygon。

定义

ST_MPointFromText 采用 ST_MultiPolygon 类型的熟知文本 (WKT) 表示和空间参考 ID，并返回 ST_MultiPolygon。

语法

如果您未指定 SRID，则空间参考默认为 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)
```

返回类型

ST_MultiPolygon

示例

创建包含 ST_MultiPolygon mpl1 列的 multipolygon_test 表。

此 INSERT 语句使用 ST_MpolyFromText 函数将 ST_MultiPolygon 插入到 mpl1 列中。

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

定义

ST_MPointFromWKB 以 ST_MultiPolygon 类型的熟知二进制 (WKB) 表示和空间参考 ID 为输入参数，返回 ST_MultiPolygon 类型的对象。

语法

Oracle

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

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

如果您未指定 SRID，则空间参考默认为 4326。

PostgreSQL

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

SQLite

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

```
st_mpolyfromwkb (wkb blob)
```

如果您未指定 SRID，则空间参考默认为 4326。

返回类型

ST_MultiPolygon

示例

本示例说明了如何使用 ST_MPolyFromWKB 由熟知二进制表示创建多面对象。几何是空间参考系统 4326 中的多面。在本示例中，多面存储在 sample_mpolys 表的几何列中且 ID = 10，然后利用熟知二进制表示对 wkb 列进行更新（使用 ST_AsBinary 函数）。最后，ST_MPolyFromWKB 函数用于从 wkb 列中返回多面。sample_mpolys 表格具有一个几何列（用于存储多面）和一个 wkb 列（用于存储多面的 WKB 表示）。

SELECT 语句包含 ST_MPolyFromWKB 函数，该函数用于从 WKB 列检索多面。

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

注：
仅限 Oracle

定义

ST_MultiCurve 通过熟知文本表示构造多曲线要素。

语法

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

返回类型

ST_MultiLinestring

示例

```
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

定义

ST_MultiLineString 通过熟知文本表示构造多线串。

注：
创建将与 ArcGIS 配合使用的空间表时，最好将列创建为几何超类型（例如 ST_Geometry），而不是指定 ST_Geometry 子类型。

语法

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)
```

返回类型

ST_MultiLineString

示例

创建表 mlines_test，并使用 ST_MultiLineString 函数向该表中插入一个多线要素。

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

定义

ST_MultiPoint 通过熟知文本表示构造多点要素。

注：
创建将与 ArcGIS 配合使用的空间表时，最好将列创建为几何超类型（例如 ST_Geometry），而不是指定 ST_Geometry 子类型。

语法

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)
```

返回类型

ST_MultiPoint

示例

创建表 mpoint_test，并使用 ST_MultiPoint 函数向该表中插入一个多点要素。

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

定义

ST_MultiPolygon 通过熟知文本表示构造多面要素。

注：
创建将与 ArcGIS 配合使用的空间表时，最好将列创建为几何超类型（例如 ST_Geometry），而不是指定 ST_Geometry 子类型。

语法

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)
```

返回类型

ST_MultiPolygon

示例

创建表 mpoly_test，并使用 ST_MultiPolygon 函数向该表中插入一个多面。

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

注：
仅限 Oracle

定义

ST_MultiSurface 通过熟知文本表示构造多表面要素。

语法

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

返回类型

ST_MultiSurface

示例

```
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

定义

ST_NumGeometries 采用几何集合并返回集合中几何的数量。

语法

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)
```

返回类型

整型

示例

在以下示例中，将创建一个名为 sample_numgeom 的表。 向其中插入一个多面体和一个多点。 在 SELECT 语句中，ST_NumGeometries 函数用于确定各个几何中的几何（或要素）数量。

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

定义

ST_NumInteriorRing 以 ST_Polygon 作为输入参数，并返回其内部环数。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_numinteriorring (polygon1 geometryblob)
```

返回类型

整型

示例

一个鸟类学家想研究几个南部海岛上的鸟群。她想标识哪些岛上有一个或多个湖，因为她感兴趣的鸟类仅在淡水湖中觅食。

岛屿表格的 ID 和名称列标识各岛屿，而陆地 ST_Polygon 列存储岛屿的几何。

因为内部环代表湖泊，所以包含 ST_NumInteriorRing 函数的 SELECT 语句仅列出那些至少有一个内部环的岛屿。

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

定义

ST_NumPoints 用于返回几何中的点（折点）数。

对于面要素，将一并对起始折点和结束折点进行计数，即使它们处于相同的位置。

请注意，该数值与 ST_Geometry 类型的 NUMPTS 属性不同。NUMPTS 属性包含几何的所有部件的折点计数（包括部件间的分隔符）。各部件间有一个分隔符。例如，具有三个部件的多部件线串具有两个分隔符。在 NUMPTS 属性中，每个分隔符被计作一个折点。相反，ST_NumPoints 函数在折点数中不包括分隔符。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_numpoints (geometry1 geometryblob)
```

返回类型

整型

示例

创建包含 geotype 列（包含 g1 列中存储的几何类型）的 numpoints_test 表。

此 INSERT 语句用于插入点、线串和面。

对于每种要素类型的每个要素，SELECT 查询都使用 ST_NumPoints 函数获取其中的点数。

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

注：
仅限 Oracle 和 PostgreSQL

定义

ST_OrderingEquals 用于比较两个 ST_Geometries 对象，如果两个对象的几何相等且坐标顺序相同，则返回 1 (Oracle) 或 t (PostgreSQL)；否则，其将返回 0 (Oracle) 或 f (PostgreSQL)。

语法

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

返回类型

布尔

示例

Oracle

下面的 CREATE TABLE 语句将创建包含两个线串列（ln1 和 ln2）的 LINESTRING_TEST 表。

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

下面的 INSERT 语句将两个 ST_LineString 值插入到相等且具有相同坐标顺序的 ln1 和 ln2 中。

```
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)
);
```

以下 SELECT 语句和相应结果集显示了不考虑坐标顺序时 ST_Equals 函数返回 1 (true) 的条件。 如果两个几何不相等但具有相同的坐标顺序，ST_OrderingEquals 函数将返回 0 (false)。

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

lid Equals     OrderingEquals

1   1          0
```

PostgreSQL

下面的 CREATE TABLE 语句将创建包含两个线串列（ln1 和 ln2）的 LINESTRING_TEST 表。

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

下面的 INSERT 语句将两个 ST_LineString 值插入到相等且具有相同坐标顺序的 ln1 和 ln2 中。

```
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)
);
```

以下 SELECT 语句和相应结果集显示了不考虑坐标顺序时 ST_Equals 函数返回 t (true) 的条件。 如果两个几何不相等但具有相同的坐标顺序，ST_OrderingEquals 函数将返回 f (false)。

```
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

定义

ST_Overlaps 将以两个几何对象作为输入参数，如果两个对象的交集生成的几何对象维度相同但不等于任一源对象，则返回 1（Oracle 和 SQLite）或 t (PostgreSQL)。否则，返回 0（Oracle 和 SQLite）或 f (PostgreSQL)。

语法

Oracle 和 PostgreSQL

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

SQLite

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

返回类型

布尔型

示例

县政委员需要一个包含了与危险废弃物场地缓冲半径区域重叠的敏感区域的列表。除用于存储机构 ST_Polygon 几何的形状列外，sensitive_areas 表还包含几个用于描述受威胁机构的列。

hazardous_sites 表在 ID 列中存储场地标识，而各场地的实际地理位置存储在场地点列中。

ST_Overlaps 函数会将 sensitive_areas 和 hazardous_sites 表连接起来，并返回包含与 hazardous_sites 点缓冲半径区域重叠的面的所有 sensitive_areas 行 ID。

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

定义

ST_Perimeter 返回形成闭合面或多面要素的边界的连续线的长度。

此函数为 10.8.1 版本的新功能。

语法

每个部分中的前两个选项将以为要素所定义的坐标系为单位返回周长。后两个选项可用于指定线性测量单位。有关 linear_unit_name 的受支持的值列表，请参阅 ST_Distance。

Oracle 和 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)
```

返回类型

双精度型

示例

Oracle

在以下示例中，研究海岸线鸟类的生态学家需要确定特定区域内湖泊的海岸线长度。湖泊将在 waterbodies 表中存储为面。一个使用 ST_Perimeter 函数的 SELECT 语句将用于在 waterbodies 表中返回每个湖（要素）的周长。

```
--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;
```

SELECT 语句返回如下内容：

```
ID PERIMETER
1 +1.8000000
```

在下一个示例中，您将创建一个名为 bfp 的表，插入三个要素，并以线性测量单位计算每个要素的周长：

```
--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;
```

SELECT 语句以三个单位返回每个要素的周长：

```
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

在以下示例中，研究海岸线鸟类的生态学家需要确定特定区域内湖泊的海岸线长度。湖泊将在 waterbodies 表中存储为面。一个使用 ST_Perimeter 函数的 SELECT 语句将用于在 waterbodies 表中返回每个湖（要素）的周长。

```
--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;
```

SELECT 语句返回如下内容：

```
ID PERIMETER
1 +1.8000000
```

在下一个示例中，您将创建一个名为 bfp 的表，插入三个要素，并以线性测量单位计算每个要素的周长：

```
--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;
```

SELECT 语句以三个单位返回每个要素的周长：

```
 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

在以下示例中，研究海岸线鸟类的生态学家需要确定特定区域内湖泊的海岸线长度。湖泊将在 waterbodies 表中存储为面。一个使用 ST_Perimeter 函数的 SELECT 语句将用于在 waterbodies 表中返回每个湖（要素）的周长。

```
--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;
```

SELECT 语句返回如下内容：

```
ID PERIMETER
1 +1.8000000
```

在下一个示例中，您将创建一个名为 bfp 的表，插入三个要素，并以线性测量单位计算每个要素的周长：

```
--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;
```

SELECT 语句以三个单位返回每个要素的周长：

```
 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

定义

ST_Point 采用熟知文本对象或坐标及空间参考 ID，并返回 ST_Point 类型的对象。

注：
创建将与 ArcGIS 配合使用的空间表时，最好将列创建为几何超类型（例如 ST_Geometry），而不是指定 ST_Geometry 子类型。

语法

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)
```

返回类型

ST_Point

示例

以下 CREATE TABLE 语句将创建包含单点列 (PT1) 的 point_test 表。

在将点坐标插入 pt1 列之前，ST_Point 函数会将其转换为 ST_Point 几何。

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

注：
仅用于 Oracle 和 SQLite；对于 PostgreSQL，请使用 ST_Point。

定义

在 Oracle 中，ST_PointFromText 以点类型的熟知文本表示和空间参考 ID 作为输入参数，返回点对象。

语法

Oracle

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

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

如果您未指定 SRID，则空间参考默认为 4326。

SQLite

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

```
st_pointfromtext (wkt text)
```

如果您未指定 SRID，则空间参考默认为 4326。

返回类型

ST_Point

示例

创建包含单个 ST_Point 列 pt1 的 point_test 表。

在调用 INSERT 语句将点插入到 pt1 列之前，首先使用 ST_Point 函数将点文本坐标转换为点格式。

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

定义

ST_PointFromWKB 以熟知二进制 (WKB) 表示和空间参考 ID 作为输入参数返回 ST_Point 类型的对象。

语法

Oracle

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

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

如果您未指定 SRID，则空间参考默认为 4326。

PostgreSQL

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

SQLite

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

```
st_pointfromwkb (wkb blob)
```

如果您未指定 SRID，则空间参考默认为 4326。

返回类型

ST_Point

示例

本示例说明了如何使用 ST_PointFromWKB 函数由熟知二进制表示创建点对象。几何是空间参考系统 4326 中的点。在本示例中，点存储在 sample_points 表的几何列中且 ID = 10，然后利用熟知二进制表示对 wkb 列进行更新（使用 ST_AsBinary 函数）。最后，ST_PointFromWKB 函数用于从 WKB 列中返回点。采样点表具有一个几何列（用于存储点）和一个 wkb 列（用于存储点的熟知二进制表示）。

在下面的 SELECT 语句中，ST_PointFromWKB 函数用于从 WKB 列中获取点对象。

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

定义

ST_PointN 采用 ST_LineString 和整型索引，并返回 ST_LineString 的路径中作为第 n 个折点的点。

语法

Oracle 和 PostgreSQL

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

SQLite

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

返回类型

ST_Point

示例

使用唯一识别各行的 gid 列和 ln1 ST_LineString 列创建 pointn_test 表。INSERT 语句插入两个线串值。第一个线串没有 z 坐标和度量值，而第二个线串具有这两者。

SELECT 查询使用 ST_PointN 和 ST_AsText 函数返回每个线串第二个折点的熟知文本。

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

定义

ST_PointOnSurface 以 ST_Polygon 或 ST_MultiPolygon 类型的对象为输入参数，并返回保证位于其表面上的 ST_Point。

语法

Oracle 和 PostgreSQL

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

SQLite

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

返回类型

ST_Point

示例

市政工程师想要为每个历史建筑物的覆盖区创建标注点。历史建筑物覆盖区存储在使用以下 CREATE TABLE 语句创建的 hbuildings 表中：

ST_PointOnSurface 函数生成确保位于建筑物覆盖区表面上的点。ST_PointOnSurface 函数返回点对象以便由 ST_AsText 函数将其转换为文本进一步供应用程序使用。

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

注：
仅限 Oracle 和 SQLite

定义

ST_PolyFromText 以熟知文本表示和空间参考 ID 作为输入参数，返回 ST_Polygon 类型的对象。

语法

Oracle

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

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

如果您未指定 SRID，则空间参考默认为 4326。

SQLite

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

```
st_polyfromtext (wkt text)
```

如果您未指定 SRID，则空间参考默认为 4326。

返回类型

ST_Polygon

示例

创建包含单个面列的 polygon_test 表。

INSERT 语句使用 ST_PolyFromText 函数将面插入到面列中。

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

定义

ST_PolyFromWKB 以熟知二进制 (WKB) 表示和空间参考 ID 作为输入参数，返回 ST_Polygon 类型的对象。

语法

Oracle

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

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

如果您未指定 SRID，则空间参考默认为 4326。

PostgreSQL

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

SQLite

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

```
st_polyfromwkb (wkb blob)
```

如果您未指定 SRID，则空间参考默认为 4326。

返回类型

ST_Polygon

示例

本示例说明了如何使用 ST_PolyFromWKB 从熟知二进制表示创建面。几何是空间参考系统 4326 中的面。在本示例中，面存储在 sample_polys 表的几何列中且 ID = 1115，然后利用 WKB 表示对 wkb 列进行更新（使用 ST_AsBinary 函数）。最后，ST_PolyFromWKB 函数用于从 WKB 列中返回多面。sample_polys 表具有一个几何列（用于存储面）和一个 wkb 列（用于存储面的 WKB 表示）。

在下面的 SELECT 语句中，ST_PointFromWKB 函数用于从 WKB 列中获取点对象。

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

定义

ST_Polygon 存取器函数使用熟知文本 (WKT) 表示和空间参考 ID (SRID)，并生成 ST_Polygon。

注：
创建将与 ArcGIS 配合使用的空间表时，最好将列创建为几何超类型（例如 ST_Geometry），而不是指定 ST_Geometry 子类型。

语法

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)
```

返回类型

ST_Polygon

示例

以下 CREATE TABLE 语句将创建包含单列 (p1) 的 polygon_test 表。随后的 INSERT 语句会将一个环（封闭的简单面）转换为 ST_Polygon，并使用 ST_Polygon 函数将其插入 p1 列。

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

定义

ST_Relate 比较两个几何，如果几何满足 DE-9IM 模式矩阵字符串指定的条件，则返回 1（Oracle 和 SQLite）或 t (PostgreSQL)；否则，返回 0（Oracle 和 SQLite）或 f (PostgreSQL)。

在 SQLite 和 Oracle 中使用 ST_Relate 时有第二个选项：可以比较两个几何以返回表示定义几何之间关系的 DE-9IM 模式矩阵的字符串。

语法

Oracle

选项 1

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

选项 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

选项 1

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

选项 2

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

返回类型

对于 PostgreSQL，将返回布尔值。

适用于 SQLite 和 Oracle 的选项 1 将返回一个整数。

适用于 SQLite 和 Oracle 的选项 2 将返回一个字符串。

示例

DE-9IM 模式矩阵是一种用于比较几何的设备。这种矩阵包含多种类型。例如，您可以使用 ST_Relate 函数和相等模式矩阵 (T*F**FFF*) 来发现任意两个几何是否相等，但也可以提供 DE-9IM 模式 (1*F**FFF*)。对于后一种模式，

ST_Relate 将告知您两个几何是否与第一个位置相等，从而指示两个几何相交的内部是否是一条线（维度为 1）。

在以下示例中，将创建包含三个空间列的表 relate_test，每个空间列中均插入了点要素。ST_Relate 函数用于在 SELECT 语句中测试各点是否相同。

如果要确定几何是否相等并且不需要查找关系的维度，请改用 ST_Equals 函数。

Oracle

第一个示例显示第一个 ST_Relate 选项，将根据 DE-9IM 模式矩阵比较几何，如果几何满足矩阵中定义的要求，将返回 1，如果不满足，将返回 0。

```
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;
```

它将返回以下内容：

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

1                0                0
```

此示例显示了第二个选项。它将比较两个几何并返回 DE-9IM 模式矩阵。

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

它将返回以下内容：

```
g1 rel g2
```

```
0FFFFFFF2
```

PostgreSQL

第一个示例将根据 DE-9IM 模式矩阵比较几何，如果几何满足矩阵中定义的要求，将返回 t，如果不满足，将返回 f。

```
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;
```

它将返回以下内容：

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

t          f         f
```

SQLite

第一个示例显示第一个 ST_Relate 选项，将根据 DE-9IM 模式矩阵比较几何，如果几何满足矩阵中定义的要求，将返回 1，如果不满足，将返回 0。

```
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;
```

它将返回以下内容：

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

1          0         0
```

此示例显示了第二个选项。 它将比较两个几何并返回 DE-9IM 模式矩阵。

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

它将返回以下内容：

```
g1 rel g2
0FFFFFFF2
```

ST_SRID

定义

ST_SRID 以几何对象作为输入参数，并返回其空间参考 ID。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_srid (geometry1 geometryblob)
```

返回类型

整型

示例

创建下列表格：

在下一语句中，位于坐标 (10.01, 50.76) 处的点几何将插入到几何列 g1 中。创建点几何时，给点几何分配 SRID 值 4326。

ST_SRID 函数返回刚输入的几何的空间参考 ID。

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

定义

ST_StartPoint 用于返回线串的第一个点。

语法

Oracle 和 PostgreSQL

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

SQLite

```
st_startpoint (ln1 geometryblob)
```

返回类型

ST_Point

示例

创建 startpoint_test 表，表中包含用于唯一识别表格行的 gid 整型列和 ln1 ST_LineString 列。

INSERT 语句用于将 ST_LineStrings 插入到 ln1 列中。第一个 ST_LineString 没有 z 坐标或度量值，而第二个 ST_LineString 具有这两者。

ST_StartPoint 函数用于提取各 ST_LineString 的第一个点。列表中的第一个点不包含 z 坐标或度量值，而第二个点则包含这两个值，这是因为此源线串具有这二个属性。

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

注：
仅限 Oracle 和 SQLite

定义

ST_Surface 通过熟知文本表示构造表面要素。 表面类似于面，但它们在其范围内的每个点处均有值。

语法

Oracle

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

SQLite

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

返回类型

ST_Polygon

示例

创建表 surf_test，并在该表中插入一个表面几何。

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

定义

ST_SymmetricDiff 以两个几何对象作为输入参数，并返回由源对象的非公共部分组成的几何对象。

语法

Oracle 和 PostgreSQL

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

SQLite

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

返回类型

Oracle 和 PostgreSQL

ST_Geometry

SQLite

Geometryblob

示例

对于特定报表，县政委员必须确定不相交流域和危险污染源半径区域。

流域表包含 id 列以存储流域名称 (wname)，以及形状列以存储流域区域几何。

污染源表在 id 列中存储场地标识，而各场地的实际地理位置存储在场地点列中。

ST_Buffer 函数生成一个环绕危险废弃物场地点的缓冲区域。ST_SymmetricDiff 函数将返回不相交的缓冲危险废弃物场地和流域的面。

对危险废弃物场地与流域的交集取反可生成相交区域的差集。

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

定义

如果两个几何的公共点都不与两个几何的内部相交，则 ST_Touches 返回 1（Oracle 和 SQLite）或 t (PostgreSQL)；否则，返回 0（Oracle 和 SQLite）或 f (PostgreSQL)。其中至少一个几何必须是 ST_LineString、ST_Polygon、ST_MultiLineString 或 ST_MultiPolygon。

语法

Oracle 和 PostgreSQL

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

SQLite

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

返回类型

布尔型

示例

公司老板要其 GIS 技术人员提供具有与其他下水道管线相交的端点的所有下水道管线的列表。

创建 sewerlines 表，表中包含三个列。第一列 (sewer_id) 用于唯一标识各下水道管线。整型类列用于标识通常与管线容量有关的下水道管线类型。下水道列用于存储下水道管线的几何。

SELECT 查询使用 ST_Touches 函数返回彼此相接的下水道列表。

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

定义

ST_Transform 将二维 ST_Geometry 数据作为输入，并返回已转换为空间参考的值，该空间参考由您所提供的空间参考 ID (SRID) 指定。

⚠ 警告:

如果使用 st_register_spatial_column 函数将空间列注册到 PostgreSQL 数据库，注册时 SRID 将写入 sde_geometry_columns 表。 如果在 Oracle 数据库的空间列中创建了空间索引，创建空间索引时 SRID 将写入 st_geometry_columns 表。 使用 ST_Transform 更改 ST_Geometry 数据的 SRID 不会在 sde_geometry_columns 或 st_geometry_columns 表中更新 SRID。

如果地理坐标系不同，则 ST_Transform 将执行地理变换。 地理变换是指在两个地理坐标系间进行转换。 地理（坐标）变换定义为在特定转换方向上进行，例如，从 NAD 1927 到 NAD 1983，但不管源和目标坐标系是什么，ST_Transform 函数都能正确执行变换。

地理转换方法可以分为两类：基于方程和基于文件。 基于方程的方法是独立的，并且不需要任何外部信息。 基于文件的方法使用磁盘文件来计算偏移值。 与基于方程的方法相比，基于文件的方法通常更加精确。 在澳大利亚、加拿大、德国、新西兰、西班牙和美国，通常使用基于文件的方法。 文件（加拿大除外）可从 ArcGIS Pro 安装获得或直接从各种国家制图机构获得。

要支持基于文件的变换，必须将文件置于安装了数据库的服务器中，其文件夹结构应与 ArcGIS Pro 安装目录中的 pedata 文件夹相同。

例如，名为 `pedata` 的文件夹位于 ArcGIS Pro 安装目录的 `Resources` 文件夹中。 `pedata` 文件夹包含多个子文件夹，但三个包含受支持基于文件的方法的文件夹分别为：`harn`、`nadcon` 和 `ntv2`。 可以将 `pedata` 文件夹及其内容从 ArcGIS 安装目录复制到数据库服务器，或在数据库服务器上创建一个目录，其中包含受支持基于文件的变换方法子目录及文件。 文件复制到数据库服务器之后，可在同一服务器上设置名为 PEDATAHOME 的操作系统环境变量。将 PEDATAHOME 变量设置为包含子目录和文件的目录位置；例如，如果在 Microsoft Windows 服务器中将 pedata 文件夹复制到 `C:\pedata`，则将 PEDATAHOME 环境变量设置为 `C:\pedata`。

有关如何设置环境变量的信息，请参阅操作系统文档。

设置 PEDATAHOME 后，必须先启动新的 SQL 会话，才能使用 ST_Transform 函数；但无需重新启动服务器。

在 PostgreSQL 中使用 ST_Transform

在 PostgreSQL 中，可以在具有相同地理坐标系或不同地理坐标系的空间参考之间进行转换。

如果数据存储在数据库（而非地理数据库）中，请在地理坐标系相同的情况下，执行以下操作更改 ST_Geometry 数据的空间参考：

1. 创建表的备份副本。
2. 在表中创建第二个（目标）ST_Geometry 列。
3. 注册目标 ST_Geometry 列，并指定新的 SRID。
   此操作通过在 sde_geometry_columns 系统表中放置记录来指定列的空间参考。
4. 运行 ST_Transform 函数并指定变换所得数据应输出到目标 ST_Geometry 列。

5. 取消注册第一个（源）ST_Geometry 列。

如果数据存储在地理数据库中，则应使用 ArcGIS 工具将数据重新投影到新要素类。 在地理数据库要素类中运行 ST_Transform 将忽略使用新 SRID 更新地理数据库系统表的功能。

在 Oracle 中使用 ST_Transform

在 Oracle 中，可以在具有相同地理坐标系或不同地理坐标系的空间参考之间进行转换。

如果数据存储在数据库（而不是地理数据库）中，并且没有在空间列上定义空间索引，则可以添加第二个 ST_Geometry 列，并将变换后的数据输出到该列。 尽管使用视图或更改表的查询图层定义每次只能在 ArcGIS 中显示一列，但仍可以在表中同时保留原始（源）ST_Geometry 列和目标 ST_Geometry 列。

如果数据存储在数据库（而不是地理数据库）中，并且已在空间列上定义了空间索引，则无法保留原始 ST_Geometry 列。 在 ST_Geometry 列上定义空间索引后，SRID 将写入 st_geometry_columns 元数据表。 ST_Transform 将不更新该表。

1. 创建表的备份副本。
2. 在表中创建第二个（目标）ST_Geometry 列。
3. 运行 ST_Transform 函数并指定变换所得数据应输出到目标 ST_Geometry 列。
4. 从源 ST_Geometry 列删除空间索引。
5. 删除源 ST_Geometry 列。
6. 在目标 ST_Geometry 列创建空间索引。

如果数据存储在地理数据库中，则应使用 ArcGIS 工具将数据重新投影到新要素类。如果在地理数据库要素类中运行 ST_Transform，则将忽略使用新 SRID 更新地理数据库系统表的功能。

在 SQLite 中使用 ST_Transform

在 SQLite 中，可以在具有相同地理坐标系或不同地理坐标系的空间参考之间进行转换。

语法

源和目标空间参考具有相同的地理坐标系

Oracle 和 PostgreSQL

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

SQLite

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

源和目标空间参考具有不同的地理坐标系

Oracle

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

PostgreSQL

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

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

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

在选项 3 中，可以选择性地按照以下顺序指定范围，即以逗号分隔的坐标列表：左下 x 坐标、左下 y 坐标、右上 x 坐标、右上 y 坐标。 如果未指定范围，则 ST_Transform 将使用更大更常用的范围。

指定范围时，本初子午线和单位转换因子参数为可选项。 仅当您指定的范围值未使用格林尼治本初子午线或十进制度时，才需要提供此信息。

SQLite

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

返回类型

Oracle 和 PostgreSQL

ST_Geometry

SQLite

Geometryblob

示例

源和目标空间参考具有相同的地理坐标系时转换数据

下例中，将创建包含以下两个线串列的表 transform_test：ln1 和 ln2。 向 ln1 插入线，SRID 为 4326。 然后在 UPDATE 语句中使用 ST_Transform 函数获取 ln1 中的线串，将线串由分配给 SRID 4326 的坐标参考转换为分配给 SRID 3857 的坐标参考，再将其置于 ln2 列中。

注：
SRID 4326 和 3857 具有相同的地理基准面。

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);
```

源和目标空间参考不具有相同的地理坐标系时转换数据

在以下示例中，已创建表 n27，其中包含一个 ID 列和一个几何列。 向表 n27 中插入点，SRID 为 4267。 4267 SRID 使用 NAD 1927 地理坐标系。

然后创建表 n83 并使用 ST_Transform 函数将表 n27 中的几何插入到表 n83 中，但 SRID 为 4269，地理变换 ID 为 1241。 SRID 4269 使用 NAD 1983 地理坐标系，1241 为 NAD_1927_To_NAD_1983_NADCON 变换的已知 ID。 此变换是基于文件的变换并且可用于美国本土的 48 个州。

提示：

有关受支持的地理变换列表，请参阅 Esri 技术文章 000004829，以及文章的相关信息部分中提供的链接。

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
);
```

如果 PEDATAHOME 正确定义，则针对 n83 表格的 SELECT 语句运行将返回以下内容：

```
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

定义

ST_Union 返回两个源对象组合而成的几何对象。

语法

Oracle 和 PostgreSQL

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

SQLite

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

返回类型

Oracle 和 PostgreSQL

ST_Geometry

SQLite

Geometryblob

示例

除用于存储机构面几何的形状列外，sensitive_areas 表还存储受威胁机构的 ID。

hazardous_sites 表在 ID 列中存储场地标识，而各场地的实际地理位置存储在场地点列中。

ST_Buffer 函数生成一个环绕危险废弃物场地的缓冲区域。ST_Union 函数从缓冲危险废弃场地和敏感区域面的并集生成面。ST_Area 函数返回这些面的面积。

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

定义

如果第一个 ST_Geometry 对象完全位于第二个 ST_Geometry 对象的范围内，则 ST_Within 返回 1（Oracle 和 SQLite）或 t (PostgreSQL)；否则，返回 0（Oracle 和 SQLite）或 f (PostgreSQL)。

语法

Oracle 和 PostgreSQL

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

SQLite

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

返回类型

布尔型

示例

下例中，创建了两个表文件：zones 和 squares。SELECT 语句将找出相交但不完全处于同一地块内的所有正方形。

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

定义

ST_X 以 ST_Point 作为输入参数，返回其 x 坐标。在 SQLite 中，ST_X 也可以更新 ST_Point 的 x 坐标。

语法

Oracle 和 PostgreSQL

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

SQLite

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

返回类型

双精度型

ST_X 函数可用于 SQLite 更新点的 x 坐标。在这种情况下，将返回 geometryblob。

示例

使用两个列创建 x_test 表：用于唯一标识行的 gid 列，以及 pt1 点列。

INSERT 语句用于插入两行记录。一行是不带 z 坐标或度量值的点。另一列具有 z 坐标和度量值。

SELECT 查询使用 ST_X 函数获取每个点要素的 x 坐标。

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
```

ST_X 函数也可以用于更新现有点的坐标值。在本例中，ST_X 用于更新 x_test 中第一个点的 x 坐标值。

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

ST_Y

定义

ST_Y 以 ST_Point 作为输入参数，返回其 y 坐标。在 SQLite 中，ST_Y 也可以更新 ST_Point 的 y 坐标。

语法

Oracle 和 PostgreSQL

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

SQLite

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

返回类型

双精度型

ST_Y 函数可用于 SQLite 更新点的 y 坐标。在这种情况下，将返回 geometryblob。

示例

创建 y_test 表，表中包含两列数据：用于唯一标识行的 gid 列，以及 pt1 点列。

INSERT 语句用于插入两行记录。一行是不带 z 坐标或度量值的点。另一行是具有 z 坐标和度量值的点。

SELECT 查询使用 ST_Y 函数返回每个点的 y 坐标。

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
```

ST_Y 函数也可以用于更新现有点的坐标值。在本例中，ST_Y 用于更新 y_test 中第二个点的 y 坐标值。

```
UPDATE y_test
 SET pt1=st_y(
  (SELECT pt1 FROM y_test WHERE gid=2),
  20.1
  )
 WHERE gid=2;
```

ST_Z

定义

ST_Z 以 ST_Point 作为输入参数，返回其 z（高程）坐标。在 SQLite 中，ST_Z 也可以更新 ST_Point 的 z 坐标。

语法

Oracle 和 PostgreSQL

```
sde.st_z (geometry1 sde.st_point)
```

SQLite

```
st_z (geometry  geometryblob)
st_z (input_shape geometryblob, new_Zvalue double)
```

返回类型

Oracle

数值

PostgreSQL

整型

SQLite

当 ST_Z 用于返回点的 z 坐标时，返回双精度型。当 ST_Z 用于更新点的 z 坐标时，返回 geometryblob。

示例

创建 z_test 表，表中包含两列数据：用于唯一标识行的 ID 列，以及几何点列。INSERT 语句用于向 z_test 表中插入一行记录。

SELECT 语句用于列出 ID 列以及由上一语句所插入的点的双精度 z 坐标。

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
```

ST_Z 函数也可以用于更新现有点的坐标值。在本例中，ST_Z 用于更新 z_test 中第一个点的 z 坐标值。

```
UPDATE z_test
 SET pt1=st_z(
  (SELECT pt1 FROM z_test where id=1), 32.04)
 WHERE id=1;
```