

## SDE 研究系列（1）：BLOB 数据在 oracle 地理数据库中的存储方式

大讲堂首发，转载请注明出处

BLOB 数据在 oracle 地理数据库中的存储方式

BLOB (binary large object) 是数据库中用来存储二进制文件的字段类型。它是几年前 Oracle 公司实现的替代 LONG RAW 方式存储二进制数据的技术。

BLOB 的结构分为 3 部分：BLOB 列 (BLOB columns)，LOB 段 (LOB segment)，LOB 索引 (LOB index)。BLOB 列存储 LOB 定位器 (LOB locator) (36 字节) 和二进制数据（数据要小于 3,965 字节，并且 in-row storage 选项没有被禁用）。

注意：ESRI 的测试表明使用 in-row storage 方式能够达到最高的效率，因此不要禁用 in-row storage 选项。

如果二进制数据大于 3,964 字节，BLOB 列中的 in-row storage 空间不会被分配，LOB 定位器指向存储在 LOB 段中的二进制数据。因此，启用 in-row storage 选项的 BLOB 列最少空间是 36 字节（只分配空间给 LOB 定位器），最大 4000 字节（存储 LOB 定位器和 3,964 字节的二进制数据）。

LOB 段被分成很多 chunk，chunks 必须是 oracle 数据块大小的整数倍，例如如果 oracle 数据块的大小是 8k，LOB 段最小 chunk 大小是 8k。如果存储在 LOB 段中的数据是 5,000 字节，并且假设 oracle 数据块的大小是 8k，那么在这个 chunk 中有 3,192 字节的空间空闲。从 LONG RAW 到 BLOB 的方式转换数据的时候需要更多的存储空间（大概会增长 30%）。

经验表明，chunk 大小设置为 8k 能够在浪费最少空间的基础上达到最佳 I/O 的效率。

chunk 大小设置为 16k 比 chunk 大小设置为 8k 浪费更多的空间。因此为了减少空间的浪费，有二种选择：1) 重新创建数据库，把块大小设置为 8k。2) 在块大小为 8k 的 tablespace 上面创建 LOB 段，并且需要在 SGA 中分配一个 8k 的缓冲缓存。

Chunk 的大小为 4k 或者 2k 的时候会更节省空间，但会降低 I/O 的效率。

LOB 索引只有在 LOB 段的 Chunk 个数大于 12 的时候使用，当 Chunk 个数小于等于 12 的时候，通过 LOB 定位器直接指向二进制数据，当 Chunk 个数大于 12 的时候 LOB 定位器指向 LOB 索引，通过 LOB 索引访问二进制数据。

下面三个图分别是上面提到的三种情况。

图 1，二进制数据为 3000 字节 < 3,964 字节

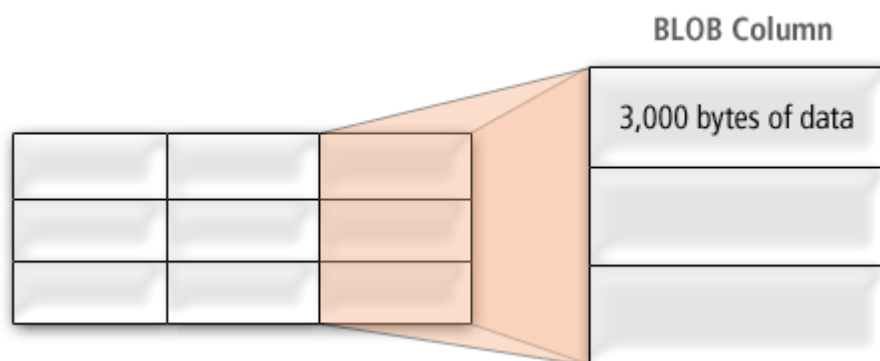


图 2 ， 二进制数据为 81,920 字节 >3,964 字节 但小于 12\*8k 字节

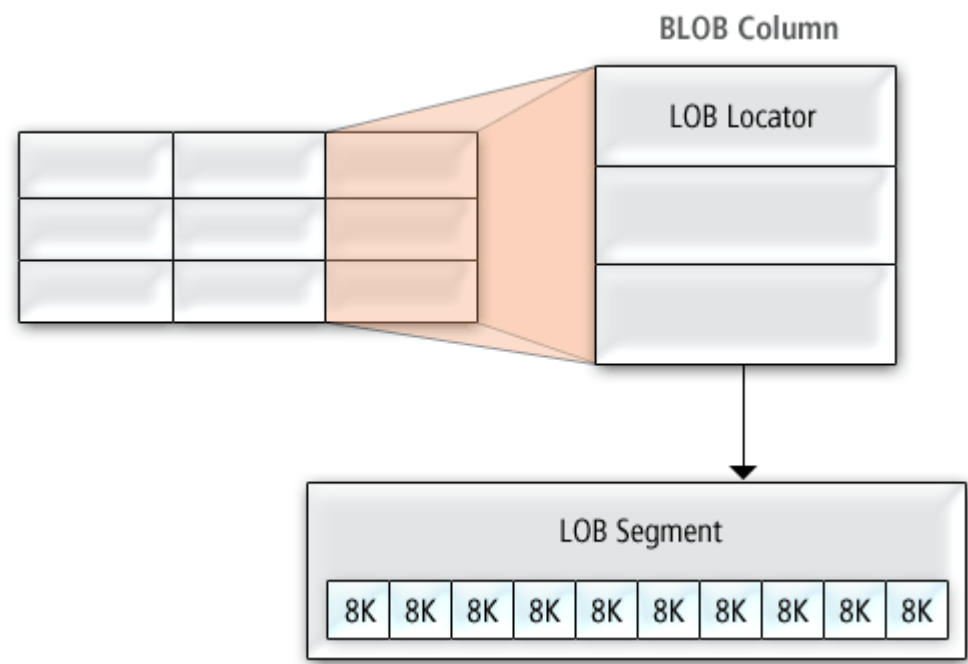
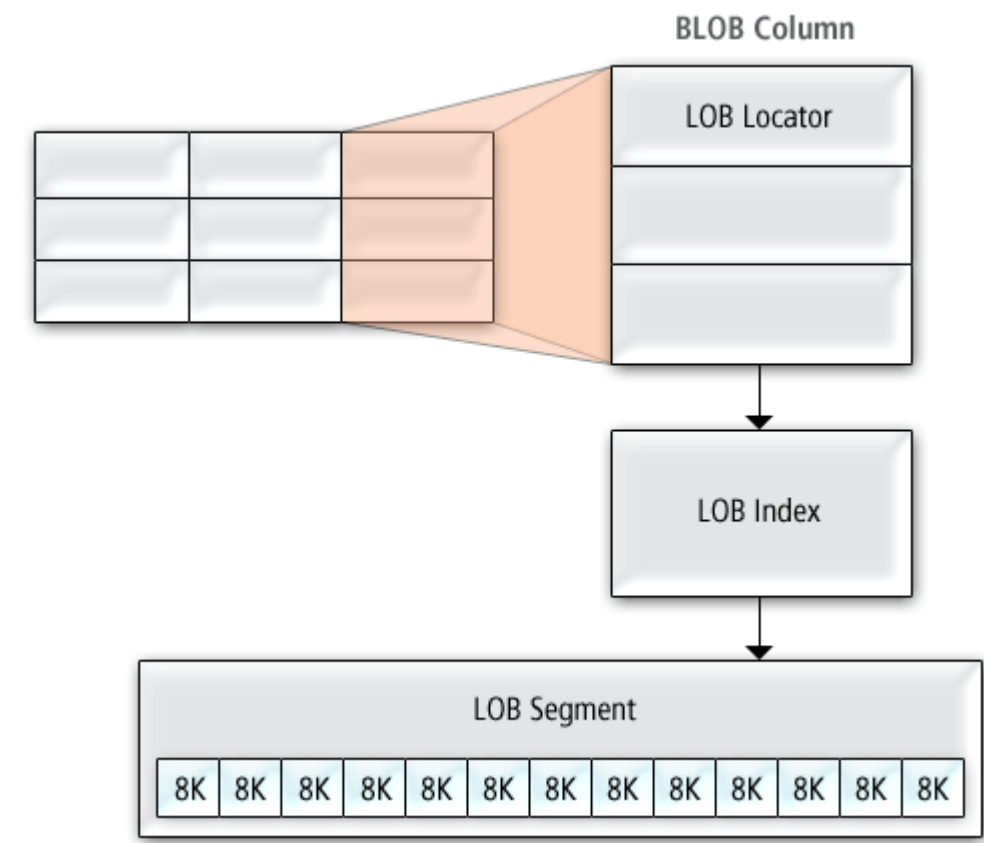


图 2  
图三， 二进制数据为 106,496 字节 > 12\*8k 字节



## SDE 研究系列 (2): 设置 DBTUNE 参数存储 BLOB 列

DBTUNE 表中的存储参数控制着在 oracle 中如何创建表和索引。其中一些存储参数定义在创建表的时候使用的数据类型。

ArcSDE 的 DBTUNE 存储参数, 例如 GEOMETRY\_STORAGE, RASTER\_STORAGE, and ATTRIBUTE\_BINARY 定义了存储 ArcSDE 数据的时候使用的 oracle 数据类型。

注意从 ArcSDE 9.2 开始 RASTER\_BINARY\_TYPE 参数被 RASTER\_STORAGE 替代。

GEOMETRY\_STORAGE 控制矢量数据 (featureClass) 的存储。RASTER\_STORAGE 控制栅格数据 (raster dataset, raster catalog, or raster attribute) 的存储。ATTRIBUTE\_BINARY 控制其他二进制数据的存储 (非矢量和栅格数据)。

在 ArcSDE 中使用 BLOB 列存储数据, DBTUNE 关键字设置如下:

GEOMETRY\_STORAGE SDELOB

RASTER\_STORAGE BLOB

ATTRIBUTE\_BINARY BLOB

ESRI 建议使用如下设置来存储矢量和栅格数据:

- 1) 总是使用 in-row storage 选项。因为在 GIS 系统中大多数的要素数据都小于 3,964 字节。使用 in-row storage 选项能达到较好的性能。
  - 2) 为需要经常读的数据使用缓冲 (Cache);
  - 3) 如果 ArcSDE 不经常更新 BLOB 数据而是经常插入或删除 BLOB 数据, 那么设置 PCT\_VERSION 参数为 0, 告诉 oracle 在 LOB 段上不需要维护老版本的数据。
  - 4) chunk 的大小不应该小于 8k。测试表明, 存储 GIS 数据, 数据块设置为 8k 是最合适的。
- 下面是 RASTER\_STORAGE 参数的设置例子

RASTER\_STORAGE "BLOB"

BLK\_STORAGE "PCTFREE 0 INITRANS 4 TABLESPACE RASTER  
LOB (BLOCK\_DATA) STORE AS  
(TABLESPACE RASTER\_LOB\_SEGMENT  
CACHE PCTVERSION 0)"

RASTER\_STORAGE "BLOB" 表示使用 BLOB 方式存储栅格数据。如果栅格的 block pixel 数据小于 3,965 字节, 则存储在 RASTER 表空间的 BLOCK\_DATA 列中。如果大于 3,964 字节则存储在 RASTER\_LOB\_SEGMENT 表空间的 LOB 段中。LOB index 只有在 chunks 个数超过 12 的时候才会使用。

下面是 GEOMETRY\_STORAGE 参数的设置例子

GEOMETRY\_STORAGE "SDELOB"

F\_STORAGE "PCTFREE 0 INITRANS 4 TABLESPACE VECTOR  
LOB (POINTS) STORE AS  
(TABLESPACE VECTOR\_LOB\_SEGMENT  
CACHE PCTVERSION 0)"

GEOMETRY\_STORAGE "SDELOB" 表示使用 BLOB 方式存储矢量数据。如果要素的二进制数据小于 3,965 字节, 则存储在 VECTOR 表空间的 POINTS 列中。如果超过这个大小则存储在 VECTOR\_LOB\_SEGMENT 表空间的 LOB 段中。

下面是 ATTRIBUTE\_BINARY "BLOB" 参数的设置例子

ATTRIBUTE\_BINARY "BLOB"

B\_STORAGE "PCTFREE 0 INITRANS 4 TABLESPACE BIZZTABS  
LOB (DOCUMENT) STORE AS  
(TABLESPACE BIZZ\_LOB\_SEGMENT

CACHE PCTVERSION 0)"

ATTRIBUTE\_BINARY "BLOB"表示使用 BLOB 方式存储属性数据（非矢量和栅格数据）。如果 b 表的二进制数据小于 3,965 字节，则存储在 BIZZTABS 表空间的 DOCUMENT 列中。如果超过这个大小则存储在 BIZZ\_LOB\_SEGMENT 表空间的 LOB 段中。如果在创建 b 表的时候没有 DOCUMENT 列，则 oracle 会报下面的错误：

ORA-00904: "DOCUMENT": invalid identifier

因此，不建议在 DBTUNE 的默认关键字的存储参数中指定特定的列。可以创建单独的 DBTUNE 关键字来设置这些参数。关于 DBTUNE 关键字的创建和参数设置请参考 DBTUNE 的相关主题。

### SDE 研究系列（3）：使用 ST\_Geometry 存储空间数据（oracle）

#### 一、简介

ArcSDE for Oracle 提供了 ST\_Geometry 类型来存储几何数据。ST\_Geometry 是一种遵循 ISO 和 OGC 规范的，可以通过 SQL 直接读取的空间信息存储类型。采用这种存储方式能够更好的利用 oracle 的资源，更好的兼容 oracle 的特征，比如复制和分区，并且能够更快的读取空间数据。使用 ST\_Geometry 存储空间数据，可以把业务数据和空间数据存储到一张表中（使用 SDENBLOB 方式业务数据和空间数据是分开存储在 B 表和 F 表中的），因此可以很方便的在业务数据中增加空间数据（只需要在业务表中增加 ST\_Geometry 列）。使用这种存储方式还能够简化多用户的读取，管理（只需要管理一张表）。

从 ArcGIS 9.3 开始，新的 ArcSDE geodatabases for Oracle 会默认使用 ST\_Geometry 方式来存储空间数据。它实现了 SQL3 规范中的用户自定义类型（user-defined data types），允许用户使用 ST\_Geometry 类型创建列来存储诸如界址点，街道，地块等空间数据。

使用 ST\_Geometry 类型存储空间数据，具有以下优势：1)通过 SQL 函数（ISO SQL/MM 标准）直接访问空间数据；2)使用 SQL 语句存储、检索操纵空间数据，就像其他类型数据一样。3)通过存储过程来进行复杂的空间数据检索和分析。4)其他应用程序可以通过 SQL 语句来访问存储在 geodatabase 中的数据。从 ArcGIS 9.3 开始，新的 ArcSDE geodatabases for Oracle 要求所以 ST 函数调用的时候前面都要加上 SDE schema 名称。例如：要对查询出来的空间数据进行 union 操作，则 SQL 函数需要这样写："sde.ST\_Union",在 9, 2 版本之前，可以不 加 SDE schema 名称。

#### 二 存储结构

ST\_Geometry 存储空间数据的结构如下表：

Name	Type
ENTITY	NUMBER(38)
NUMPTS	NUMBER(38)
MINX	FLOAT(64)
MINY	FLOAT(64)
MAXX	FLOAT(64)
MAXY	FLOAT(64)
MINZ	FLOAT(64)
MAXZ	FLOAT(64)
MINM	FLOAT(64)
MAXM	FLOAT(64)

AREA FLOAT(64)  
LEN FLOAT(64)  
SRID NUMBER(38)  
POINTS BLOB

Entity 为要素类型, 包括(linestring, multilinestring, multipoint, multipolygon, point, or polygon)。  
具体的值对应的类型可以通过 st\_geom\_util 存储过程获得。 NUMPTS 为坐标点的个数  
Minx, miny, maxx, maxy 几何的外包络矩形  
Area 几何的面积  
Len 几何的周长  
SRID 空间参考系 ID, 对应 ST\_Spatial\_References 表中的空间参考信息  
POINTS 坐标序列

### 三、操作函数

下面是一些针对 ST\_Geometry 进行操作的函数, 输入为 ST\_Geometry 类型数据, 输出为 Number 型数据、  
ST\_Area 返回几何的面积。  
ST\_Len 返回几何的周长。  
ST\_Entity 返回几何类型。  
ST\_NumPoints 返回几何坐标点的个数。  
ST\_MinM, ST\_MinX, ST\_MinY, ST\_MinZ 返回几何不同维度的最小坐标. ST\_MaxM, ST\_MaxX, ST\_MaxY, ST\_MaxZ 返回几何不同维度的最大坐标. ST\_SRID 返回空间参考系 ID.

Get\_release 返回版本信息.

如下面例子, 在 us\_states 表中查找所有 state 的名字并计算 state 的面积。

```
SELECT name, st_area(geometry)
FROM us_states
ORDER BY name;
```

### 四、构造 ST\_Geometry 对象

ST\_LineString, ST\_MultiLineString, ST\_MultiPoint, ST\_MultiPolygon, ST\_Point, and ST\_Polygon 全部是 ST\_Geometry 的子类. ST\_Geometry 和他的子类共享属性和方法. ST\_LineString, ST\_MultiLineString, ST\_MultiPoint, ST\_MultiPolygon, ST\_Point and ST\_Polygon 的构造函数的定义是相同的, 构造函数的名字就是类型名。

ST\_Point 是个有限对象 (只有一个点), 因此可以使用下面的方法来构造。

- 1, 使用 xy 坐标和 SRID 来构造 ST\_Point
- 1, 使用 xy 坐标和 SRID 来构造 ST\_Point

#### METHOD

FINAL CONSTRUCTOR FUNCTION ST\_POINT RETURNS SELF AS RESULT

Argument Name	Type	In/Out Default?
PT_X	NUMBER	IN
PT_Y	NUMBER	IN

SRID                                      NUMBER                                      IN

SQL> insert into sample\_pt values (ST\_Point (10, 20, 1) );

2, 使用 xy 坐标、高程值 (z) 和 SRID 来构造 ST\_Point

METHOD

FINAL CONSTRUCTOR FUNCTION ST\_POINT RETURNS SELF AS RESULT

Argument Name	Type	In/Out Default?
PT_X	NUMBER	IN
PT_Y	NUMBER	IN
PT_Z	NUMBER	IN
SRID	NUMBER	IN

SQL> insert into sample\_pt values (ST\_Point (10, 20, 5, 1) );

3, 使用 xy 坐标、高程值 (z), 量测值 (m) 和 SRID 来构造 ST\_Point

METHOD

FINAL CONSTRUCTOR FUNCTION ST\_POINT RETURNS SELF AS RESULT

Argument Name	Type	In/Out Default?
PT_X	NUMBER	IN
PT_Y	NUMBER	IN
PT_Z	NUMBER	IN
MEASURE	NUMBER	IN
SRID	NUMBER	IN

SQL> insert into sample\_pt values (ST\_Point (10, 20, 5, 401, 1) );

## 五 用户权限限制

在 oracle 使用 ST\_Geometry , 用户必须有以下权限:

CREATE TYPE

UNLIMITED TABLESPACE

CREATE LIBRARY

CREATE OPERATOR

CREATE INDEXTYPE

CREATE PUBLIC SYNONYM

DROP PUBLIC SYNONYM

The CONNECT and RESOURCE roles include these privileges.

## SDE 研究系列 (4): 为使用 ST\_Geometry SQL 函数配置 oracle 的网络服务

访问存储在 oracle 中的 ST\_Geometry 类型数据的 SQL 函数通过扩展 oracle 的 external procedure agent 或者 extproc 来实现, 因此, 直接使用这些 SQL 函数需要配置 oracle 的 listener, 让 oracle 能够找到这些扩展库。如果使用 SDE 读取这些数据, 则不需要配置。这些对 ST\_Geometry 类型数据进行操作的函数是用 PL/SQL 实现的, 在 PL/SQL 中其实是转调的使用 c 语言编写的外部扩展库 (ST\_SHAPELIB)。

关于 oracle 的 listener 的详细配置方法请参考 oracle 的相关文档, 下面主要介绍一下默认情况下如何配置 (windows 下面)。

1) 找到 oracle 数据库的安装目录 (服务器端), 然后定位到 oraclehome\NETWORK\ADMIN

## 目录

- 2) 备份 listener.ora 文件，这点很重要，在对 oracle 的配置做任何更改的时候都要进行备份
- 3) 打开 listener.ora 文件，找到 (PROGRAM = extproc) 这一行，在这行下面添加对 ST\_SHAPELIB 的引用，即指定 ST\_SHAPELIB 的地址，如下：  
(ENVS="EXTPROC\_DLLS=C:\Program Files\ArcGIS\ArcSDE\ora10gexe\bin\st\_shapelib.dll")  
其中“C:\Program Files\ArcGIS\ArcSDE\ora10gexe\bin\st\_shapelib.dll”为 ST\_SHAPELIB 的物理路径，可以根据安装情况自己修改。
- 4) 保存 listener.ora 文件，重新启动监听程序。

## 附录：

未修改前的 listener.ora

```
# listener.ora Network Configuration File: D:\oracle\product\10.2.0\db_1
\network\admin\listener.ora
# Generated by Oracle configuration tools.

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = D:\oracle\product\10.2.0\db_1)
      (PROGRAM = extproc)
    )
  )
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1))
      (ADDRESS = (PROTOCOL = TCP)(HOST = zbc)(PORT = 1521))
    )
  )
```

修改后的 listener.ora

```
# listener.ora Network Configuration File: D:\oracle\product\10.2.0\db_1
\network\admin\listener.ora
# Generated by Oracle configuration tools.

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = D:\oracle\product\10.2.0\db_1)
      (PROGRAM = extproc)
      (ENVS="EXTPROC_DLLS=C:\Program Files\ArcGIS\ArcSDE\ora10gexe\bin\st_shapelib.dll")
    )
  )
LISTENER =
  (DESCRIPTION_LIST =
```

```

(DESCRIPTION =
  (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1))
  (ADDRESS = (PROTOCOL = TCP)(HOST = zbc)(PORT = 1521))
)
)

```

## **SDE 研究系列（5）：创建空间数据存储类型为 ST\_Geometry 的要素类**

创建空间数据存储类型为 ST\_Geometry 的要素类有 2 种方法：

### 1) 使用 SDE 创建要素类

从 9.3 开始，默认创建的要素类都使用 ST\_Geometry 存储空间数据，9.3 版本之前，可以通过配置 dbtune 参数来完成。

### 2) 直接使用 SQL 语句创建要素类。

1. 首先使用 sqlplus 连接到 oracle 服务器，确保登陆用户有如下权限：

```

CREATE TYPE
UNLIMITED TABLESPACE
CREATE LIBRARY
CREATE OPERATOR
CREATE INDEXTYPE
CREATE PUBLIC SYNONYM
DROP PUBLIC SYNONYM

```

The CONNECT and RESOURCE roles include these privileges.

2. 使用 SQL 语句创建包含 ST\_Geometry 列的表，注意：要使 SDE 能够认识你创建的要素类，新建的表中有且只能有一个 ST\_Geometry 列，并且最好包含唯一标识的列。

执行下面的 sql 语句

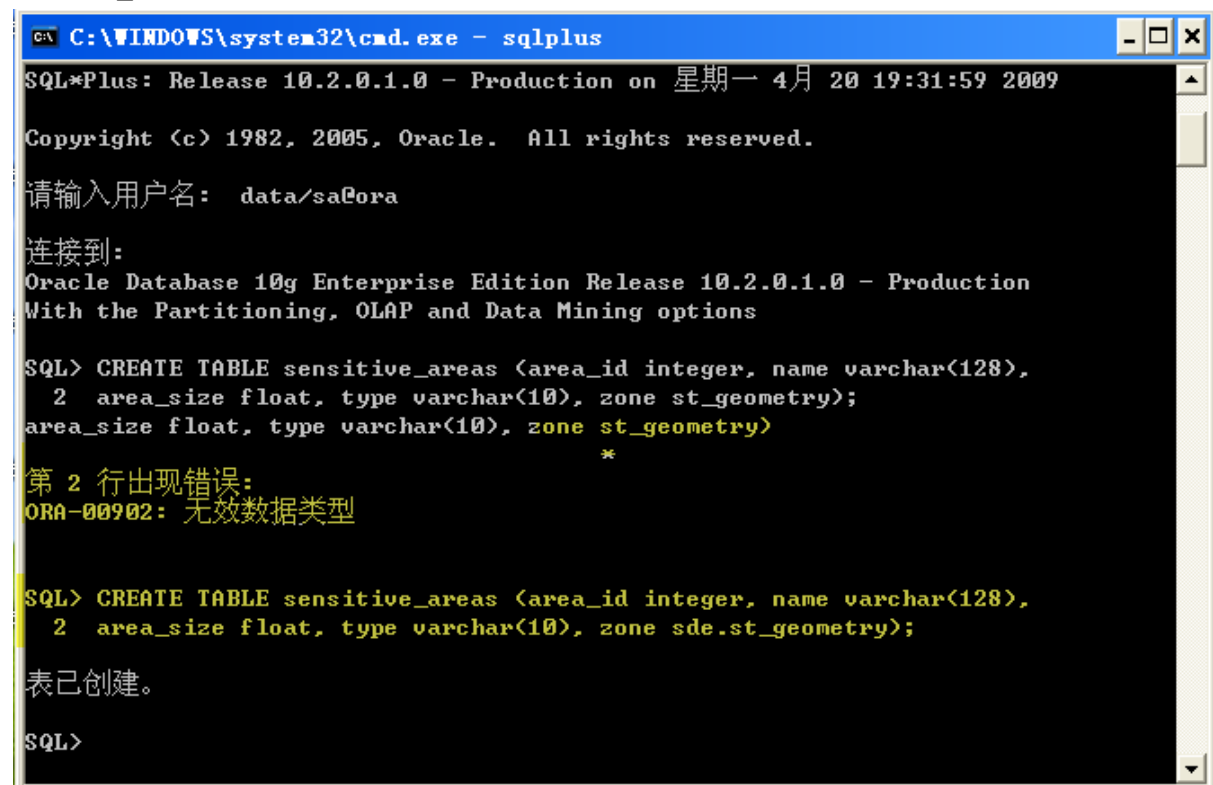
```

CREATE TABLE sensitive_areas (area_id integer, name varchar(128),
area_size float, type varchar(10), zone sde.st_geometry);

```



注意 st\_geometry 前面应添加 sde schema 名, 否则会报错。



```
C:\WINDOWS\system32\cmd.exe - sqlplus
SQL*Plus: Release 10.2.0.1.0 - Production on 星期一 4月 20 19:31:59 2009
Copyright (c) 1982, 2005, Oracle. All rights reserved.
请输入用户名: data/saPora
连接到:
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
With the Partitioning, OLAP and Data Mining options
SQL> CREATE TABLE sensitive_areas (area_id integer, name varchar(128),
  2 area_size float, type varchar(10), zone st_geometry);
area_size float, type varchar(10), zone st_geometry)
*
第 2 行出现错误:
ORA-00902: 无效数据类型
SQL> CREATE TABLE sensitive_areas (area_id integer, name varchar(128),
  2 area_size float, type varchar(10), zone sde.st_geometry);
表已创建。
SQL>
```

3. 使用 sdelayer 命令, 将创建好的表注册到 SDE 中

注册的时候一定要保证下面几点才能成功:

- 1 必须是表的所有者才能注册。
- 2 表中只能有一个 ST\_Geometry 列。
- 3 没有其他用户自定义类型的列。
- 4 必须是简单的集合类型(points, lines, or polygons)。
- 5 Geometry 必须是有效的, 否则读取的时候会产生不可预期的错误。

sdelayer 命令参数如下:

```
sdelayer -o register -l <table,column> -e <entity_mask> [Spatial_Index]
        [{-R <SRID> | [Spatial_Ref_Opts]]} [-P {BASIC | HIGH}]
        [{-C NONE} | [-C
<row_id_column>[, {SDE|USER}[, <min_ID>]]]}
        [-E {empty | xmin, ymin, xmax, ymax}] [-t <storage_type>]
        [-S <layer_description_str>] [-q]
        [-k <config_keyword>] [-i <service>] [-s <server_name>]
        [-u <DB_User_name>] [-p <DB_User_password>] [-D
<database>]
```

在命令行中执行下列命令

```
sdelayer -o register -l sensitive_areas,zone -e a -C area_id,SDE -u data
-p sa -t ST_GEOMETRY
-o 参数为 register -l 参数为 表名/ST_Geometry 列 -e 几何类型 -C 用户唯一
id/SDE 其中 SDE 表示由 sde 维护唯一 id, 使用 USER 选项, 则有用户维护唯一
```

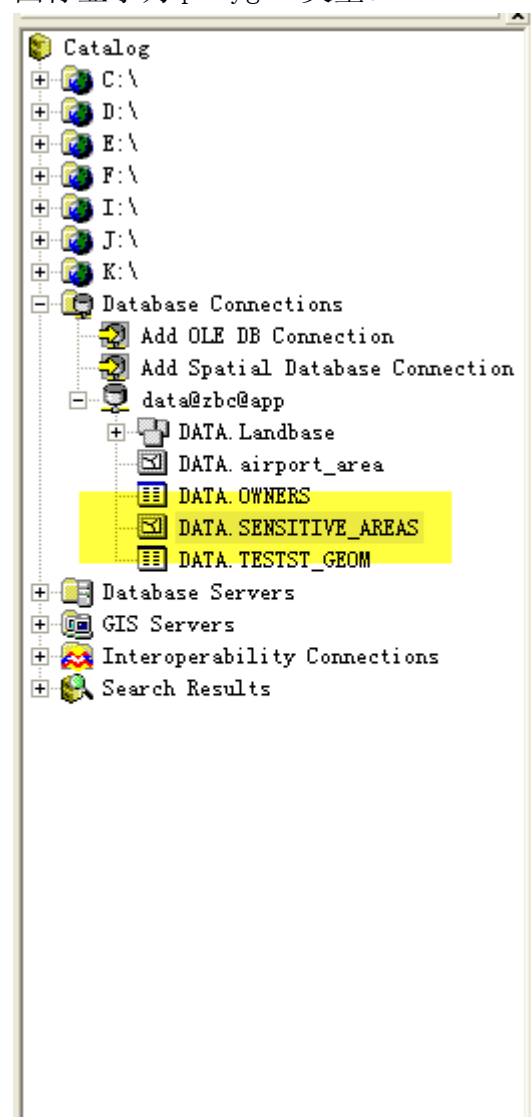
ID -u 注册地用户名 -p 用户密码 -t 数据存储类型

```
C:\>sdelayer -o register -l sensitive_areas.zone -e a -C fid,SDE -u data -p sa -t ST_GEOMETRY

ArcSDE 9.3 for Oracle10g Build 508 Thu Apr 17 12:23:18 2008
Layer Administration Utility
-----
Successfully Created Layer.
C:\>^E
```

barry ~~~~~

4. 在 catalog 中查看，可以看到 sensitive\_areas 已经被注册到 sde 中，并且图标显示为 polygon 类型。



SDE 研究系列（6）：使用 SQL 直接操纵 FeatureClass(oracle)

使用 SQL 直接操纵 FeatureClass(oracle), 这里主要讲存储类型为 ST\_Geometry 的要素类。

对 FeatureClass 的操作主要包括下面几点:

1 数据的插入, 删除, 更新

数据的插入直接使用 insert 语句来进行, 构造 ST\_Geometry 的时候可以通过两种方法来完成:

1) 使用 WKT 编码

2) 使用 WKB 编码

上面两种编码都是 OGC 规范的编码方式, 分别通过 ST\_PolyFromText() 和 ST\_PointFromWKB() 以及一系列类似函数还完成从 WKT 或 WKB 到 ST\_Geometry 转换。

下面为所有相关 ST 函数:

ST\_GeomFromText—Creates an ST\_Geometry from a text representation of any geometry type

ST\_PointFromText—Creates an ST\_Point from a point text representation

ST\_LineFromText—Creates an ST\_LineString from a linestring text representation

ST\_PolyFromText—Creates an ST\_Polygon from a polygon text representation

ST\_MPointFromText—Creates an ST\_MultiPoint from a multipoint representation

ST\_MLineFromText—Creates an ST\_MultiLineString from a multilinestring representation

ST\_MPolyFromText—Creates an ST\_MultiPolygon from a multipolygon representation

ST\_AsText—converts an existing geometry into a text representation

ST\_GeomFromWKB—Creates an ST\_Geometry from a WKB representation of any geometry type

ST\_PointFromWKB—Creates an ST\_Point from a point WKB representation

ST\_LineFromWKB—Creates an ST\_LineString from a linestring WKB representation

ST\_PolyFromWKB—Creates an ST\_Polygon from a polygon WKB representation

ST\_MPointFromWKB—Creates an ST\_MultiPoint from a multipoint WKB representation

ST\_MLineFromWKB—Creates an ST\_MultiLineString from a multilinestring WKB representation

ST\_MPolyFromWKB—Creates an ST\_MultiPolygon from a multipolygon WKB representation

ST\_AsBinary—converts an existing geometry value into well-known binary representation.

关于 wkt 和 wkb 的编码方式会在下一节讲。

下面就使用 sql 语句来进行数据的插入, 执行下面的 sql 语句:

```
INSERT INTO sensitive_areas (area_id, name, area_size, type, zone, fid)
VALUES (1, 'Summerhill Elementary School', 67920.64, 'school',
```

```
sde.ST_PolyFromText(' polygon
((52 28,58 28,58 23,52 23,52 28))', 0));
```

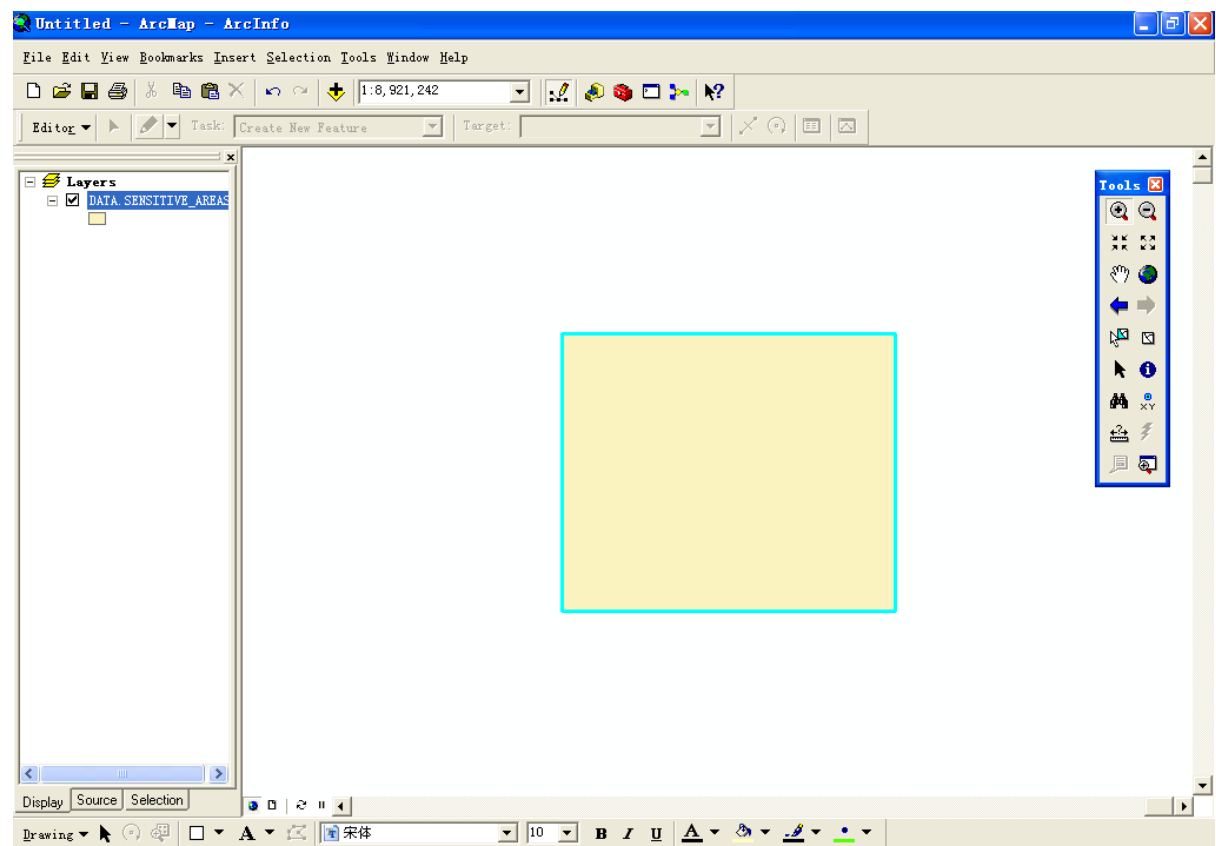
在 sensitive\_areas 表中插入一条记录。

```
SQL> INSERT INTO sensitive_areas (area_id , name, area_size, type , zone,fid) U
VALUES <1, 'Summerhill Elementary School', 67920.64, 'school', sde.ST_PolyFromTex
t('polygon <(52 28,58 28,58 23,52 23,52 28)>','0'),1>;

已创建 1 行。

SQL>
```

打开 ArcMap, 加载这个 featureClass, 可以看到新增加的这条记录。



更新记录

```
UPDATE sensitive_areas
```

```
SET zone= sde.st_pointfromtext(' polygon ((52 30,58 30,58 23,50 23,50
28))', 0))
```

```
WHERE area_id = 1;
```

删除记录

```
DELETE FROM sensitive_areas WHERE names
```

```
(SELECT sa.names
```

```
FROM sensitive_areas sa, hazardous_sites hs
```

```
WHERE sde.st_overlaps (sa.zone, sde.st_buffer (hs.location,.01)) = 1);
```

2 查询

在进行空间查询的时候会用到索引，其中 infomix 使用 R 树索引，PostgreSQL 使用 Generalized Search Tree (GiST) R-tree 索引，oracle 和 DB2 使用网格索引。

在 oracle 中进行查询会执行下面的步骤：

- 1) 首先比较 grid 和查询范围，找出在查询范围内的所有 grid。
- 2) 找出在这些 grid 内的所有要素。
- 3) 将这些要素的外包络矩形和查询范围比较，找出所有在查询范围内以及和查询范围相交的要素。
- 4) 使用 ST 函数进行最终过滤（一般使用 ST\_Intersects 或 ST\_Contains），找到完全符合条件的要素。

下面是一个进行空间查询的例子：

```
SELECT sa.name "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;
```

### 3 创建索引

```
CREATE INDEX sa_idx ON sensitive_areas(zone)
INDEXTYPE IS sde.st_spatial_index
PARAMETERS('st_grids=1,3,0 st_srid=0');
```

```
SQL> CREATE INDEX sa_idx ON sensitive_areas(zone) INDEXTYPE IS sde.st_spatial_in
dex PARAMETERS('st_grids=1,3,0 st_srid=0');
```

索引已创建。

```
SQL>
```

### 4 创建空间参考系

ESRI 建议最好使用 ArcGIS Desktop 来进行空间参考系的创建，但也可以使用 SQL 语句进行空间参考系的创建。执行下面的 sql 语句：

```
INSERT INTO SDE.ST_SPATIAL_REFERENCES VALUES (
    GCS_North_American_1983,
    1,
    -400,
    -400,
    1000000000,
    -100000,
    10000,
    -100000,
    10000,
    9.999E35,
    -9.999E35,
    9.999E35,
    -9.999E35,
    9.999E35,
    -9.999E35,
```

```

9.999E35,
-9.999E35,
4269,
'GCS_North_American_1983',
'PROJECTED',
NULL,
'GEOGCS["GCS_North_American_1983",DATUM["D_North_American_1983",SPHER
OID
["GRS_1980",6378137.0,298.257222101]],PRIMEM["Greenwich",0.0],UNIT
["Degree",0.0174532925199433]]',
'ArcSDE SpRef'
);

```

### SDE 研究系列 (7): wkt 与 wkb

wkt (OGC well-known text) 和 wkb (OGC well-known binary) 是 OGC 制定的空间数据的组织规范, wkt 是以文本形式描述, wkb 是以二进制形式描述。使用 wkt 和 wkb 能够很好到和其他系统进行数据交换, 目前大部分支持空间数据存储的数据库构造空间数据都采用这两种方式。

wkt 的组织结构如下:

Geometry type	Text description	Comment
ST_Point	'point empty'	empty point
ST_Point	'point z empty'	empty point with z-coordinate
ST_Point	'point m empty'	empty point with measure
ST_Point	'point zm empty'	empty point with z-coordinate and measure
ST_Point	'point ( 10.05 10.28 )'	point
ST_Point	'point z( 10.05 10.28 2.51 )'	point with z-coordinate
ST_Point	'point m( 10.05 10.28 4.72 )'	point with measure
ST_Point	'point zm(10.05 10.28 2.51 4.72 )'	point with z-coordinate and measure
ST_LineString	'linestring empty'	empty linestring
ST_LineString	'linestring z empty'	empty linestring with z-coordinates
ST_LineString	'linestring m empty'	empty linestring with measures

ST_LineString	'linestring zm empty'	empty linestring with z-coordinates and measures
ST_LineString	'linestring (10.05 10.28 , 20.95 20.89 )'	linestring
ST_LineString	'linestring z(10.05 10.28 3.09, 20.95 31.98 4.72, 21.98 29.80 3.51 )'	linestring with z-coordinates
ST_LineString	'linestring m(10.05 10.28 5.84, 20.95 31.98 9.01, 21.98 29.80 12.84 )'	linestring with measures
ST_LineString	'linestring zm(10.05 10.28 3.09 5.84, 20.95 31.98 4.72 9.01, 21.98 29.80 3.51 12.84)'	linestring with z-coordinates and measures
ST_Polygon	'polygon empty'	empty polygon
ST_Polygon	'polygon z empty'	empty polygon with z-coordinates
ST_Polygon	'polygon m empty'	empty polygon with measures
ST_Polygon	'polygon zm empty'	empty polygon with z-coordinates and measures
ST_Polygon	'polygon ((10 10, 10 20, 20 20, 20 15, 10 10))'	polygon
ST_Polygon	'polygon z((10 10 3, 10 20 3, 20 20 3, 20 15 4, 10 10 3))'	polygon with z-coordinates
ST_Polygon	'polygon m((10 10 8, 10 20 9, 20 20 9, 20 15 9, 10 10 8 ))'	polygon with measures
ST_Polygon	'polygon zm((10 10 3 8, 10 20 3 9, 20 20 3 9, 20 15 4 9, 10 10 3 8 ))'	polygon with z-coordinates and measures
ST_MultiPoint	'multipoint empty'	empty multipoint
ST_MultiPoint	'multipoint z empty'	empty multipoint with z-coordinates
ST_MultiPoint	'multipoint m empty'	empty multipoint with measures
ST_MultiPoint	'multipoint zm empty'	empty multipoint with z-coordinates and measures
ST_MultiPoint	'multipoint (10 10, 20 20)'	multipoint with two points
ST_MultiPoint	'multipoint z(10 10 2, 20 20	multipoint with

	3)'	z-coordinates
ST_MultiPoint	'multipoint m(10 10 4, 20 20 5)'	multipoint with measures
ST_MultiPoint	'multipoint zm(10 10 2 4, 20 20 3 5)'	multipoint with z-coordinates and measures
ST_MultiLineString	'multilinestring empty'	empty multilinestring
ST_MultiLineString	'multilinestring z empty'	empty multilinestring with z-coordinates
ST_MultiLineString	'multilinestring m empty'	empty multilinestring with measures
ST_MultiLineString	'multilinestring zm empty'	empty multilinestring with z-coordinates and measures
ST_MultiLineString	'multilinestring ((10.05 10.28 , 20.95 20.89 ), ( 20.95 20.89, 31.92 21.45))'	multilinestring
ST_MultiLineString	'multilinestring z((10.05 10.28 3.4, 20.95 20.89 4.5), ( 20.95 20.89 4.5, 31.92 21.45 3.6))'	multilinestring with z-coordinates
ST_MultiLineString	'multilinestring m((10.05 10.28 8.4, 20.95 20.89 9.5), (20.95 20.89 9.5, 31.92 21.45 8.6))'	multilinestring with measures
ST_MultiLineString	'multilinestring zm((10.05 10.28 3.4 8.4, 20.95 20.89 4.5 9.5), (20.95 20.89 4.5 9.5, 31.92 21.45 3.6 8.6))'	multilinestring with z-coordinates and measures
ST_MultiPolygon	'multipolygon empty'	empty multipolygon
ST_MultiPolygon	'multipolygon z empty'	empty multipolygon with z-coordinates
ST_MultiPolygon	'multipolygon m empty'	empty multipolygon with measures
ST_MultiPolygon	'multipolygon zm empty'	empty
ST_MultiPolygon	'multipolygon (((10 10, 10 20, 20 20, 20 15 , 10 10), (50 40, 50 50, 60 50, 60 40, 50 40)))'	multipolygon
ST_MultiPolygon	'multipolygon z(((10 10 7, 10	multipolygon with



```

20 8, 20 20 7, 20 15 5, 10 10 z-coordinates
7), (50 40 6, 50 50 6, 60 50 5,
60 40 6, 50 40 6)))'
'multipolygon m(((10 10 2, 10
ST_MultiPolygon 20 3, 20 20 4, 20 15 5, 10 10 multipolygon with
2), (50 40 7, 50 50 3, 60 50 4, measures
60 40 5, 50 40 7)))'
'multipolygon zm(((10 10 7 2,
ST_MultiPolygon 10 20 8 3, 20 20 7 4, 20 15 5 multipolygon with
5, 10 10 7 2), (50 40 6 7, 50 z-coordinates and
50 6 3, 60 50 5 4, 60 40 6 5, measures
50 40 6 7)))'

```

wkb 的组织结构如下:

基本类型定义:

byte : 1 byte

uint32 : 32 bit unsigned integer (4 bytes)

double : double precision number (8 bytes)

Building Blocks : Point, LinearRing

Point {

double x;

double y;

};

LinearRing {

uint32 numPoints;

Point points[numPoints];

}

enum wkbGeometryType {

wkbPoint = 1,

wkbLineString = 2,

wkbPolygon = 3,

wkbMultiPoint = 4,

wkbMultiLineString = 5,

wkbMultiPolygon = 6,

wkbGeometryCollection = 7

};

enum wkbByteOrder {

wkbXDR = 0, Big Endian

wkbNDR = 1 Little Endian

};

WKBPoint {

byte byteOrder;

```

uint32    wkbType;          1
Point     point;
}
WKBLineString {
byte      byteOrder;
uint32    wkbType;          2
uint32    numPoints;
Point     points[numPoints];
}
WKBPolygon {
byte      byteOrder;
uint32    wkbType;          3
uint32    numRings;
LinearRing rings[numRings];
}
WKBMultiPoint {
byte      byteOrder;
uint32    wkbType;          4
uint32    num_wkbPoints;
WKBPoint  WKBPoints[num_wkbPoints];
}
WKBMultiLineString {
byte      byteOrder;
uint32    wkbType;          5
uint32    num_wkbLineStrings;
WKBLineString WKBLineStrings[num_wkbLineStrings];
}
wkbMultiPolygon {
byte      byteOrder;

uint32    wkbType;          6
uint32    num_wkbPolygons;
WKBPolygon wkbPolygons[num_wkbPolygons];
}
WKBGeometry {
union {
WKBPoint      point;
WKBLineString linestring;
WKBPolygon     polygon;
WKBGeometryCollection collection;
WKBMultiPoint  mpoint;
WKBMultiLineString mlinestring;
WKBMultiPolygon mpolygon;
}
}

```

```
};
WKBGeometryCollection {
byte    byte_order;
uint32  wkbType;      7
uint32  num_wkbGeometries;
WKBGeometry  wkbGeometries[num_wkbGeometries]
}
```

下面是一个 point(1,1) 使用 WKB 存储的例子:

010100000000000000000000F03F000000000000F03F

这个 2 进制流可以按照 WKBPoint 的结构进行拆分:

Byte order : 01

WKB type : 01000000

X : 000000000000F03F

Y : 000000000000F03F

byte order 要么为 0, 要么为 1, 0 为使用 little-endian 编码(NDR), 1 为使用 big-endian 编码(XDR)。

WKB type 是几何类型, 在 wkbGeometryType 中定义. 值为 1-7, 分别对应 Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, and GeometryCollection.

x, y 为点的坐标值, 为 double 类型。

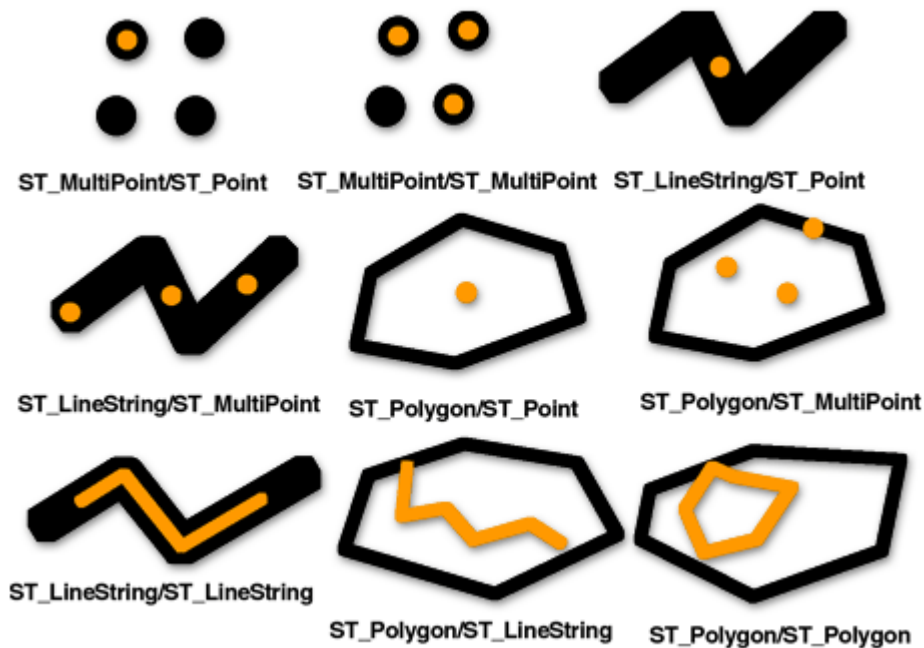
## SDE 研究系列 (8): SQL 函数介绍

在 SDE 中, 可以直接使用 SQL 语句进行空间数据的操作, 包括: 空间关系的判断, 空间分析, 以及空间数据属性的提取。

1 用于空间关系判断的函数主要有:

ST\_Contains

当第二个几何完全被第一个几何包含的时候返回 true



用法:

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

关于 `st_contains` 的详细使用说明请参考 [SQL functions reference](#)

`ST_Crosses`

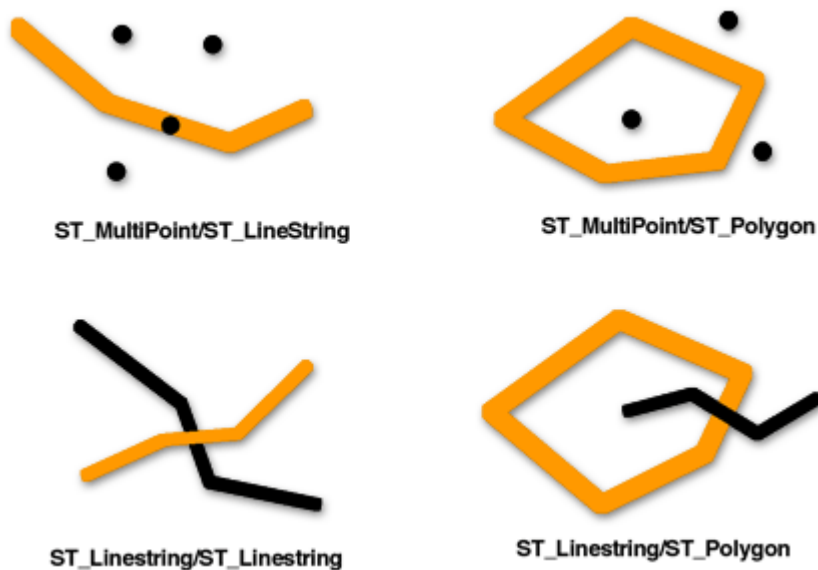
当两个几何相交的时候返回 true ， 这个函数只适用于

`ST_MultiPoint/ST_Polygon`,

`ST_MultiPoint/ST_LineString`,

`ST_LineString/ST_LineString`, `ST_LineString/ST_Polygon`, and

`ST_LineString/ST_MultiPolygon` 之间的比较。



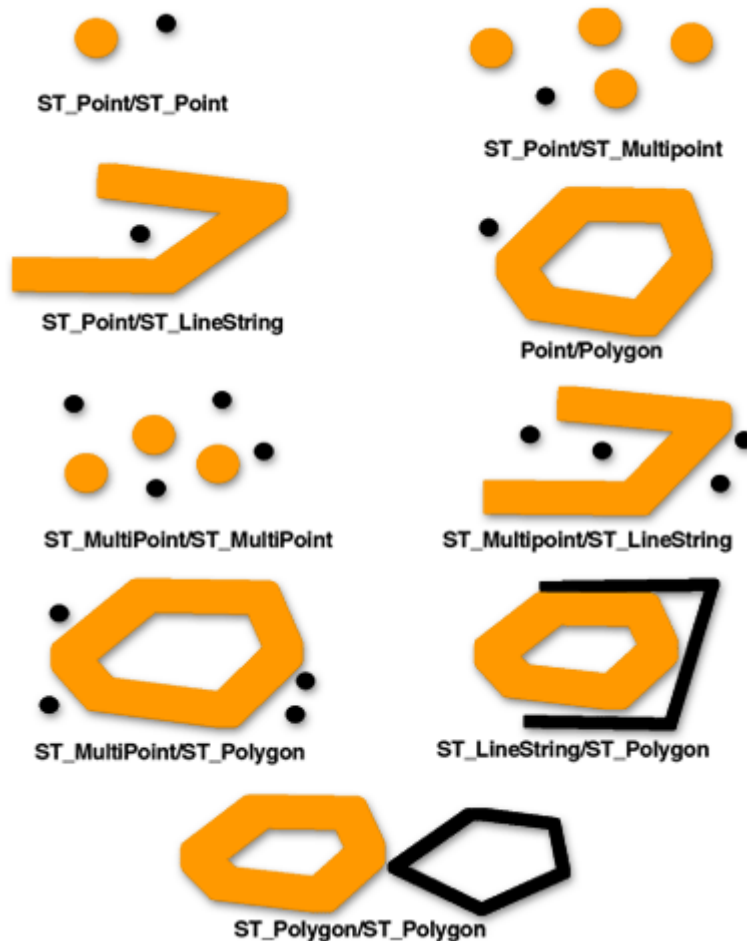
用法:

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

关于 `st_crosses` 的详细使用说明请参考 [SQL functions reference](#)

`ST_Disjoint`

当两个几何不相交的时候返回 `true`



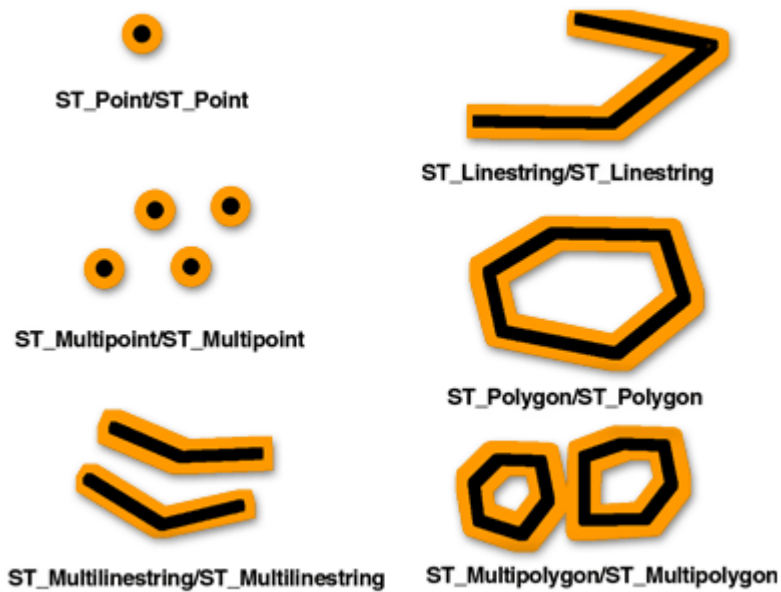
用法:

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

关于 `st_disjoint` 的详细使用说明请参考 [SQL functions reference](#)

`ST_Equals`

当两个几何类型相同，并且坐标序列相同的时候返回 true



用法:

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

关于 `st_equals` 的详细使用说明请参考 [SQL functions reference](#)

**ST\_Intersects**

当两个几何相交的时候返回 true，是 `st_disjoint` 结果的非

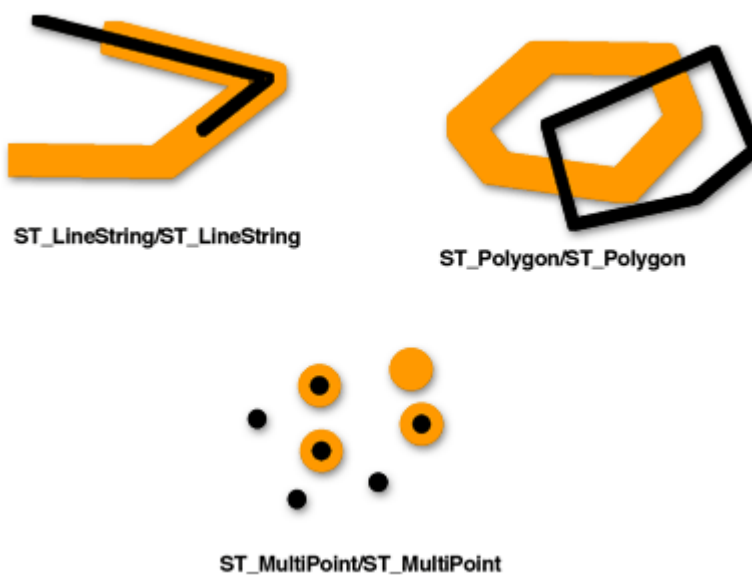
用法:

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

关于 `st_intersects` 的详细使用说明请参考 [SQL functions reference](#)

**ST\_Overlaps**

当比较的 2 个几何维数相同并且相交，返回 true



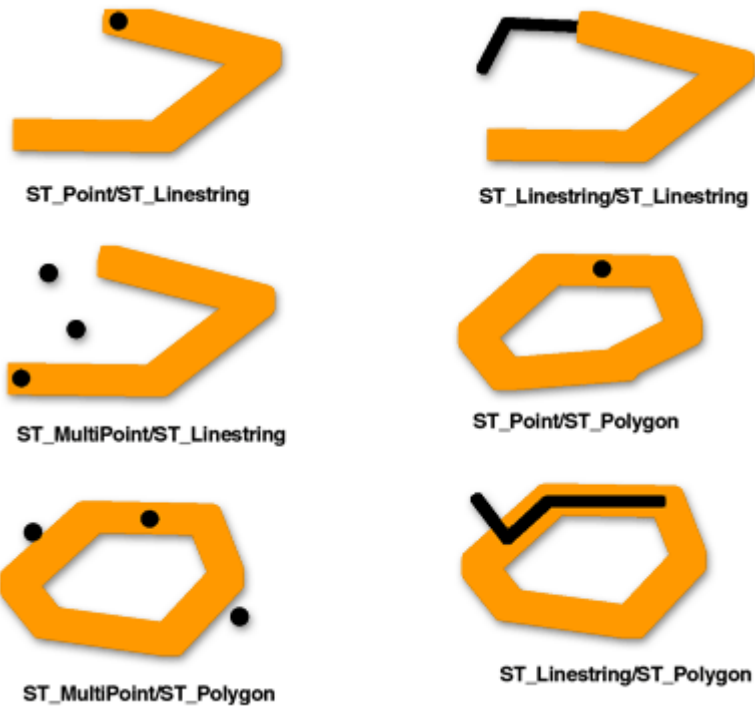
用法:

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

关于 `st_overlaps` 的详细使用说明请参考 [SQL functions reference](#)

`ST_Touches`

如果两个几何相交的部分都不在两个几何的内部，则返回 `true`



用法:

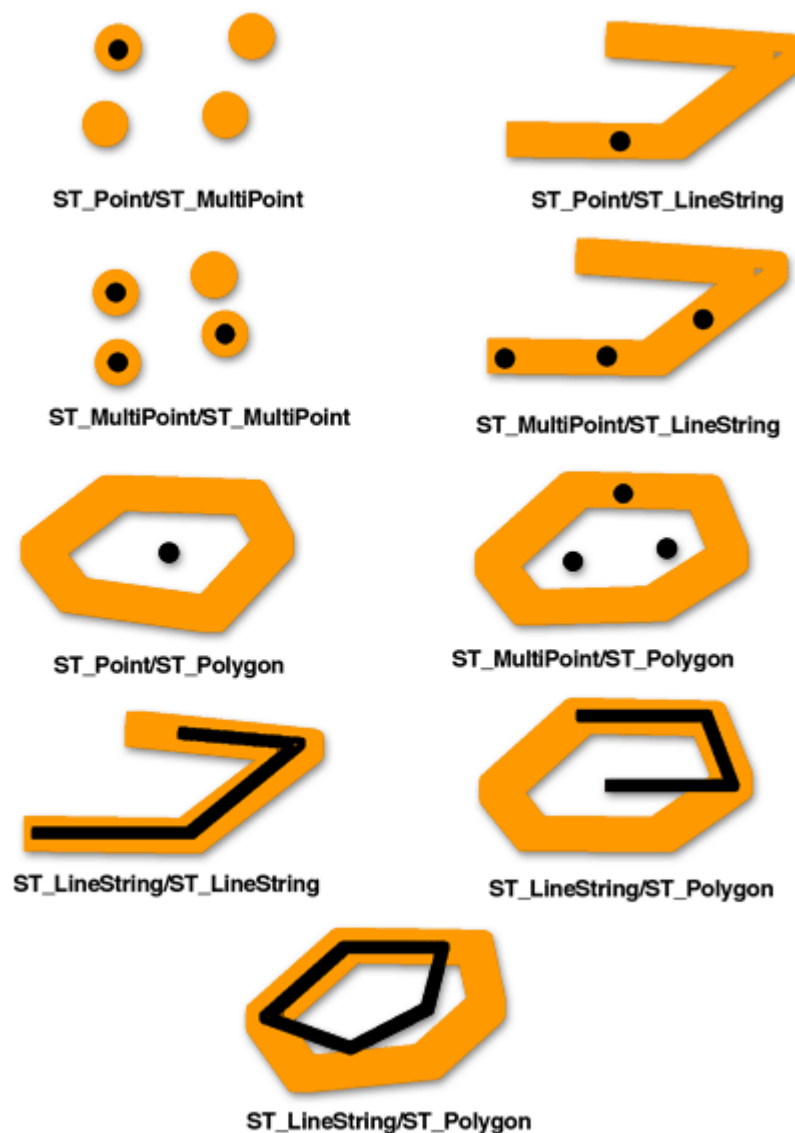
`sde.st_touches (g1 sde.st_geometry, g2 sde.st_geometry)`

关于 `st_touches` 的详细使用说明请参考 [SQL functions reference](#)

`ST_Within`

如果第一个几何完全在第二个几何内部，则返回 `true`，和 `ST_Contains` 的结果

正好相反。



用法:

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

关于 `st_within` 的详细使用说明请参考 [SQL functions reference](#)

注意: 在使用判断空间关系的 SQL 函数的时候, 应该把 SQL 函数做为在 WHERE 条件语句的第一个语句, 这样

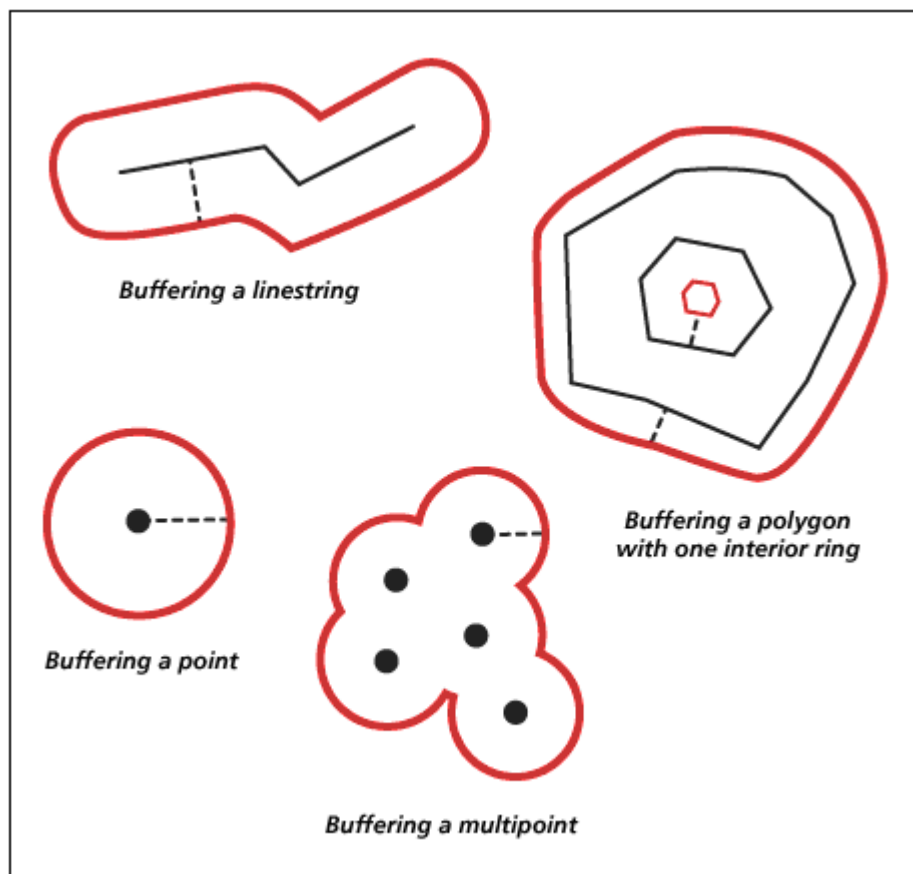
才能使用空间索引, 不然就会使用全表扫描。

2 空间分析函数:

`ST_Buffer`



## 缓冲区分析



用法:

```
sde.st_buffer (g1 sde.st_geometry, distance double_precision)
```

关于 st\_buffer 的详细使用说明请参考 SQL functions reference

ConvexHull

求几何的外包多边形，如果几何的顶点个数小于三个 ConvexHull 函数返回 null

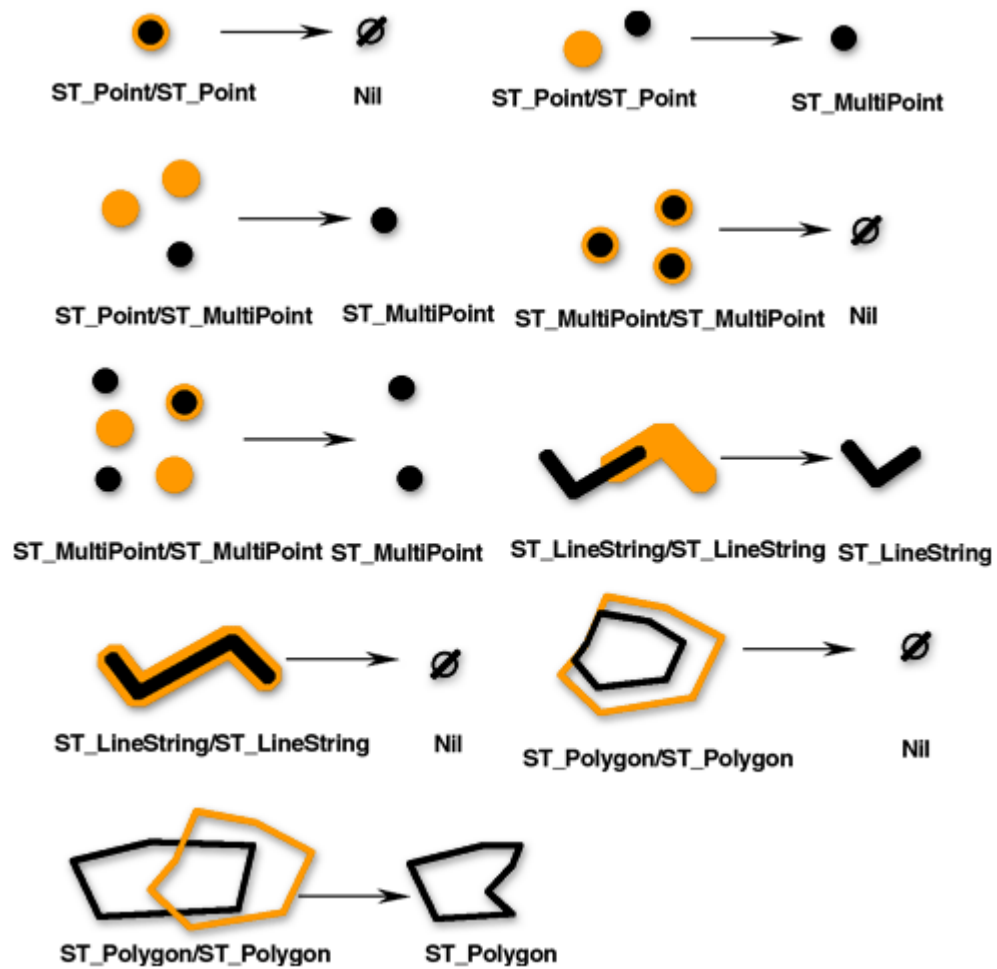
用法:

```
sde.st_convexhull (g1 sde.st_geometry)
```

关于 st\_convexhull 的详细使用说明请参考 SQL functions reference

ST\_Difference

求两个几何的 Difference ，下图中第一个几何为黑色，第二个几何为橘黄色



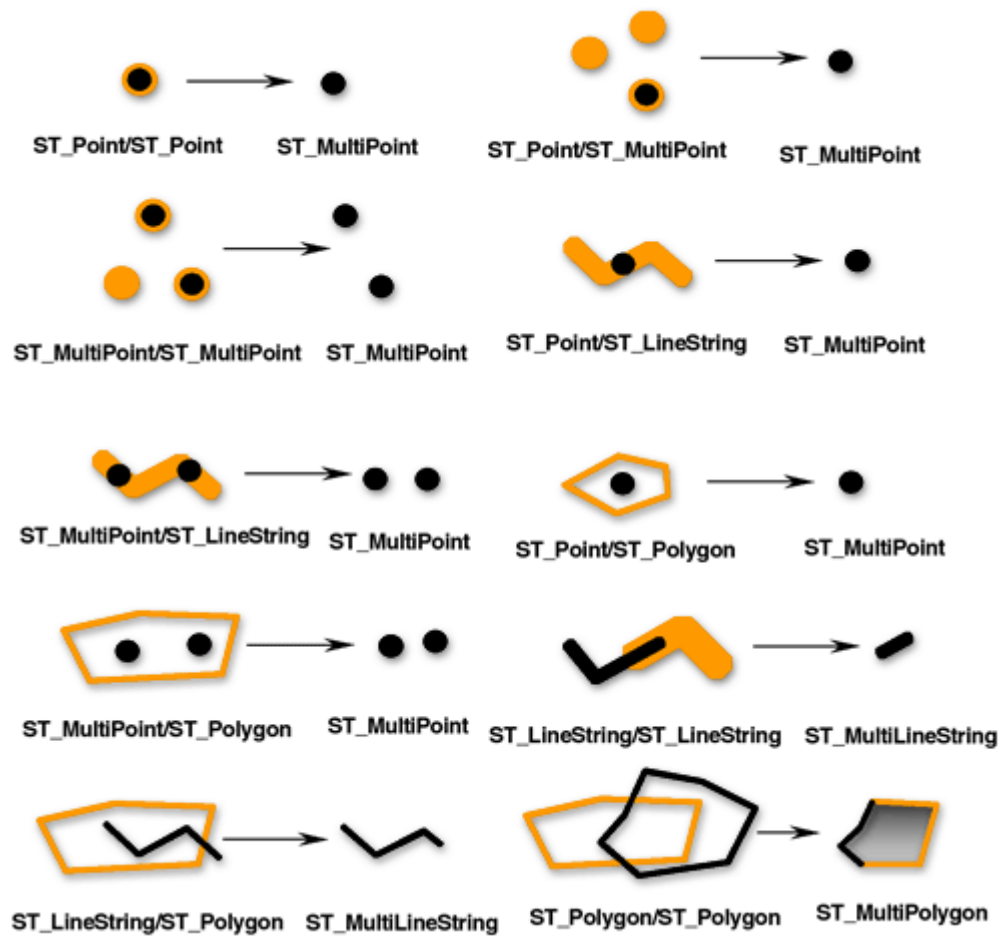
用法:

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

关于 `st_difference` 的详细使用说明请参考 [SQL functions reference](#)

`ST_Intersection`

求两个几何的交



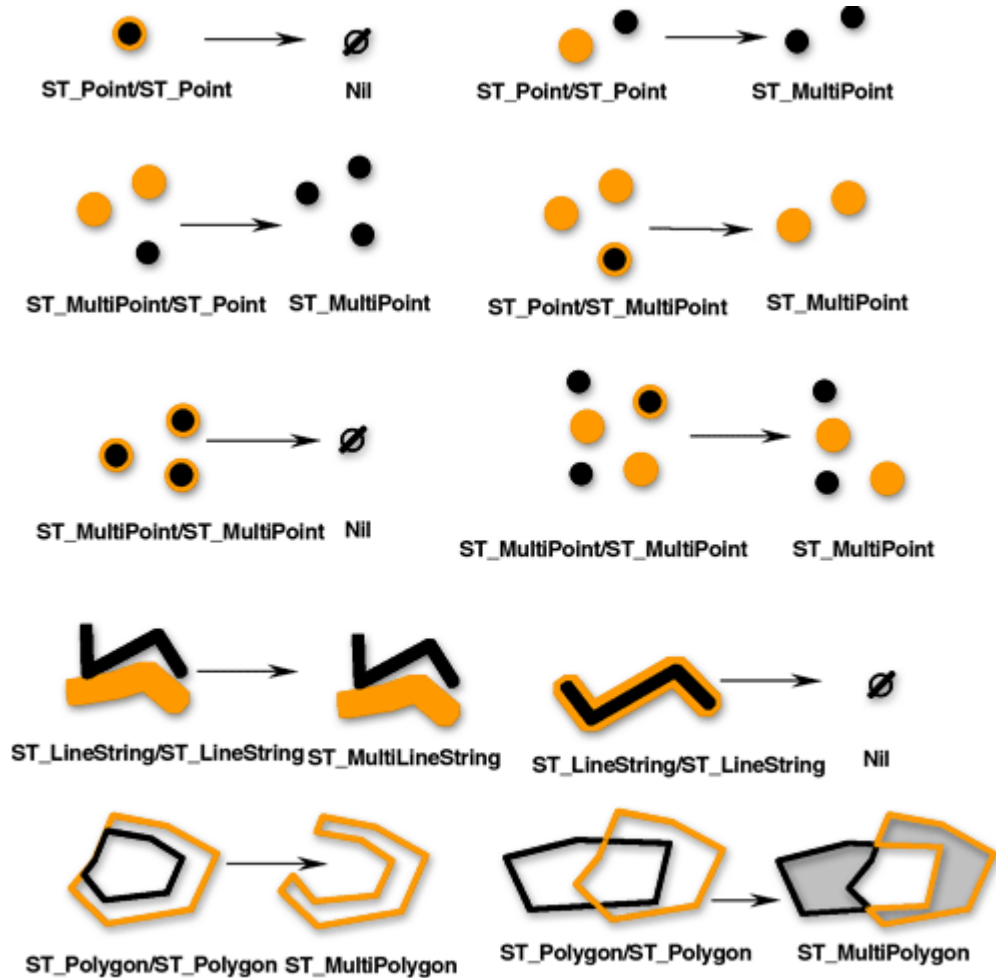
用法:

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

关于 `st_intersection` 的详细使用说明请参考 [SQL functions reference](#)

`ST_SymmetricDiff`

求几何的异或



*ST\_SymmetricDiff returns the portions of the source geometries that are not part of the intersection set. The source geometries must be of the same dimension.*

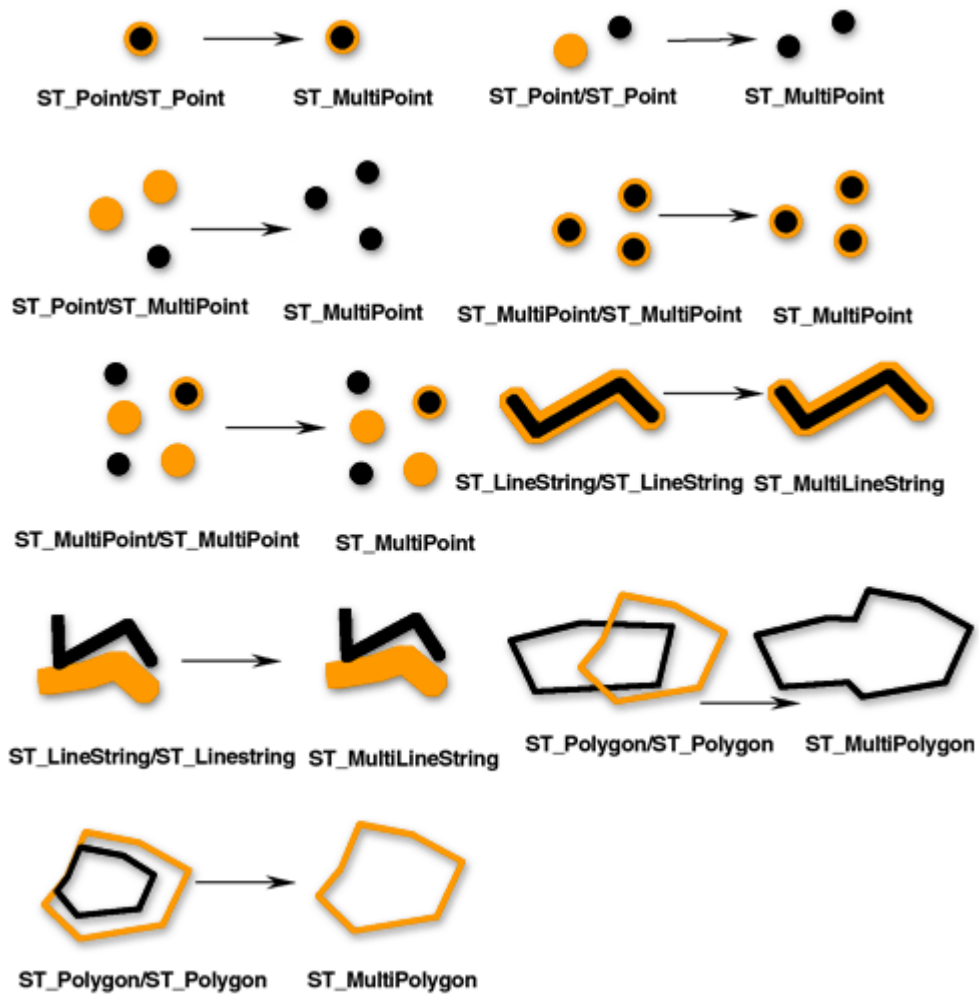
用法:

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

关于 `st_symmetricdiff` 的详细使用说明请参考 [SQL functions reference](#)

`ST_Union`

求几何的并



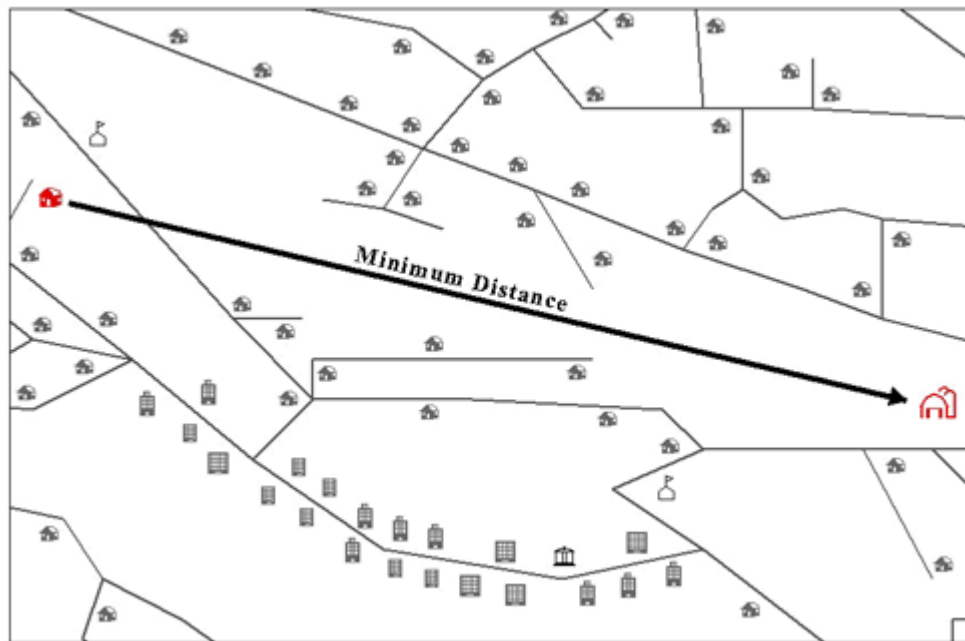
用法:

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

关于 st\_union 的详细使用说明请参考 [SQL functions reference](#)

ST\_Distance

求两个几何间的最短距离



用法:

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

关于 `st_distance` 的详细使用说明请参考 `SQL functions reference`

聚集函数:

`ST_Aggr_ConvexHull`

返回一个 `multipolygon`, 这个 `multipolygon` 是所有输入几何的外包多边形。

`ST_Aggr_Intersection`

求所有输入几何相交的部分, 返回一个几何

`ST_Aggr_Union`

求所有输入几何的并, 返回一个几何, 注意, 这个函数只支持相同类型进行聚合合并

3 空间数据属性提取的函数主要有:

1) 求几何维度的函数:

`ST_Dimension`

`ST_CoordDim`

2) z 值函数

`ST_Is3D`

`ST_Z`

`ST_MaxZ`

`ST_MinZ`

3) 量测值函数

`ST_IsMeasured`

`ST_M`

4) 取几何类型函数

`ST_GeometryType`

ST\_Entity

#### 5) ST\_Point 相关函数

ST\_X

ST\_Y

ST\_Z

ST\_M

#### 6) 面积和长度函数

ST\_Length

ST\_Area

#### 7) ST\_LineString 相关函数

ST\_StartPoint—Returns the first point of the specified ST\_LineString

ST\_EndPoint—Returns the last point of an ST\_LineString

ST\_PointN—Takes an ST\_LineString and an index to nth point and returns that point

ST\_Length—Returns the length of an ST\_LineString as a double-precision number

ST\_NumPoints—Evaluates an ST\_LineString and returns the number of points in its sequence as an integer

ST\_IsRing—A predicate function that returns 1 (TRUE) if the specified ST\_LineString is a ring and 0 (FALSE) otherwise

ST\_IsClosed—A predicate function that returns 1 (TRUE) if the specified ST\_LineString is closed and 0 (FALSE) if it is not

#### 8) ST\_MultiLineString 相关函数

ST\_Length —returns the cumulative length of all of its ST\_LineString elements as a double-precision number.

ST\_IsClosed—returns 1 (TRUE) if the specified ST\_MultiLineString is closed and 0 (FALSE) if it is not closed.

#### 9) ST\_Polygon 相关函数

ST\_Area—Returns the area of an ST\_Polygon as a double-precision number

ST\_Centroid—Returns an ST\_Point that represents the center of the ST\_Polygon's envelope

ST\_ExteriorRing—Returns the exterior ring of an ST\_Polygon as an ST\_LineString

ST\_InteriorRingN—Evaluates an ST\_Polygon and an index and returns the nth interior ring as an ST\_LineString

ST\_NumInteriorRing—Returns the number of interior rings that an ST\_Polygon contains

ST\_PointOnSurface—Returns an ST\_Point that is guaranteed to be on the

surface of the

specified ST\_Polygon

10) ST\_MultiPolygon 相关函数

ST\_Area —returns a double-precision number that represents the cumulative ST\_Area of an

ST\_MultiPolygon's ST\_Polygon elements.

ST\_Centroid — an ST\_Point that is the center of an ST\_MultiPolygon's envelope.

ST\_PointOnSurface —returns an ST\_Point that is guaranteed to be normal to the surface of

one of its ST\_Polygon elements.

11) 求几何坐标点个数函数

ST\_NumGeometries— returns a count of the individual elements in a collection of geometries.

ST\_GeometryN — you can determine which geometry in the multipart geometry exists in

position N; N being a number you provide with the function.

12) 空间参考系相关函数

ST\_SRID

ST\_Equals

13) 其他函数

ST\_Boundary

ST\_IsSimple

ST\_IsClosed

ST\_IsRing

ST\_IsEmpty

ST\_Envelope

## **SDE 研究系列 (9): 后记**

sde for oracle 存储机制研究系列主要关注的是 ST\_Geometry 的存储方式, 以及对

ST\_Geometry 类型数据使用 SQL 直接进行操作等方面。

目前从 SDE v9.3 开始使用 ST\_Geometry 做为默认的存储类型。使用 ST\_Geometry 有以下好处:

- 1 对标准的支持 (OGC 的简单要素规范, 以及 SQL/MM 标准)
- 2 更高的访问速度
- 3 数据更容易共享
- 4 和其他系统更容易集成

使用 ST\_Geometry 存储空间数据, 可以使业务数据和空间数据整合到一张表中, 再加上 ESRI 提供的网格索引, 能够更快的进行访问。

因为 ST\_Geometry 本身就是 OGC 定义的规范, 所以数据可以方便的共享, 其他数据库中的空间数据可以无损的转换到 ArcGIS 的 GeoDatabase 中, 利用 SDE 强大的空间管理优势, 也可以方便的无损的转出到其他系统中。



通过符合 SQL/MM 标准的 SQL 语句就可以访问空间数据，可以方便的和其他支持 ST\_Geometry 存储方式数据库的融合，并且可以充分利用数据库的能力进行空间分析操作。因此，ST\_Geometry 值得我们去关注。