

# ArcSDE vs. Oracle Spatial

---

吴泳锋

2010/5/26

# 目录

目录.....	1
I. 几何对象 .....	5
• Oracle Spatial .....	5
1. SDO_GEOMETRY .....	5
2. 常见几何对象的构造 .....	8
3. 导入数据到 Oracle Spatial .....	10
4. 基于 SDO_GEOMETRY 的 ST_GEOMETRY .....	12
• ArcSDE .....	13
1. ST_GEOMETRY .....	13
2. 常见几何对象的构造 .....	16
• 几何对象的性能.....	17
1. 创建几何对象的性能.....	17
2. 几何对象的存储空间.....	18
II. 空间索引 .....	21
• Oracle Spatial .....	21
1. 索引类型 SPATIAL_INDEX .....	21
2. R 树空间索引.....	27
3. 四叉树空间索引 .....	29
• ArcSDE .....	31

1. 索引类型 ST_SPATIAL_INDEX .....	31
2. 格网空间索引 .....	32
3. 格网索引的创建和调整.....	35
• 空间索引的性能.....	40
1. 空间索引的用处 .....	40
2. 主过滤/ST_ENVINTERSECTS 的比较 .....	41
III. 空间关系运算.....	44
• Oracle Spatial .....	44
• ArcSDE .....	47
• 空间关系运算的性能 .....	50
1. 相交 .....	50
IV. 几何处理 .....	52
• 缓冲分析 .....	52
• 距离量测 .....	55
• 面积、长度量测.....	56
• 凸包运算 .....	57
• 几何对象组合 .....	58
• 几何对象聚合 .....	60
V. 线性参考 .....	63
• Oracle Spatial .....	63
1. 创建线性参考的空间对象.....	63

2. 根据线性参考定位点.....	64
• ArcSDE .....	65
1. 创建线性参考的空间对象.....	65
2. 根据线性参考定位点.....	66
VI. 其它 .....	68
• 还没有涉及的话题 .....	68
• 采用 Oracle Spatial 而完全摒弃 ArcSDE 的解决方案？ .....	69

Oracle Spatial 的出现让很多人产生这样一个想法：“Oracle 对自己的数据库产品肯定是最熟悉的，Oracle Spatial 的性能也肯定是最好的，相比之下 ArcSDE for Oracle 这个后娘养的哪天还是踹掉算了”。这话听起来有一点糙，不过好像也挺在理。就在此摇摆不定之际，一旁的微软泪流满面：“作为对 Windows 最熟悉的厂商我要说一句，为什么还有人用 Windows 竟然都不选择 SQL Server 而非要用天杀的 Oracle !”

——打住，以上情节纯属虚构，如有雷同，纯属巧合。

书归正传，自打 Oracle Spatial 的出现，从开始的点线面的简单支持到栅格（Raster）、拓扑（Topology）、网络（Network）的支持；从纯粹数据库的功能到 OGC Web 服务的支持都能看出 Oracle 对 Spatial 模块的进取心。然而，SDO\_GEOMETRY 这种有点晦涩的存储方式及其空间操作方法的风格也可以看出 Oracle——或许也就是它的 CEO 埃里森的一贯作风——强势、特立独行。

当然，不管是 Oracle 亲生的还是 ESRI 后娘养的，名分都是浮云，能更好地支持我们的 GIS 应用才是王道。所以，这里主要对 ArcSDE for Oracle 和 Oracle Spatial 的功能、使用、性能等方面进行一个相对比较全面的比较。这里并不想泛泛而论到底是 A 好还是 O 好的话题，术业有专攻，对于软件来说也是“仅此而已”。

# I. 几何对象

- Oracle Spatial

## 1. SDO\_GEOMETRY

Oracle Spatial 在 MDSYS 模式下定义了一系列几何类型、函数来支持空间数据的存储和使用，最为人耳熟能详的就是 SDO\_GEOMETRY 这种类型——当然，ArcSDE 也可以使用这种类型进行存储。让我们首先来看一下

SDO\_GEOMETRY 的定义：

```
CREATE OR REPLACE
TYPE SDO_GEOMETRY AS OBJECT (
    SDO_GTYPE      NUMBER,
    SDO_SRID       NUMBER,
    SDO_POINT      SDO_POINT_TYPE,
    SDO_ELEM_INFO  SDO_ELEM_INFO_ARRAY,
    SDO_ORDINATES  SDO_ORDINATE_ARRAY,
    MEMBER FUNCTION GET_GTYPE
    RETURN NUMBER DETERMINISTIC,
    MEMBER FUNCTION GET_DIMS
    RETURN NUMBER DETERMINISTIC,
    MEMBER FUNCTION GET_LRS_DIM
    RETURN NUMBER DETERMINISTIC)
...
```

其中的 SDO\_GTYPE 属性是一个 4 位的 NUMBER 类型对象，这个 4 位的数字表示了这个几何对象的类型，让我们来看看这 4 个数字如何来约定几何类型。

这 4 个数字的形式为 “DLTT”，其中 D 代表维数，可以取值为 2、3、4；L 代表线性参考，如果没有取值为 0；TT 代表几何类型，取值为 00 到 09，这 00 到 09 的取值也是有讲究的，下表就是不同值对应的几何类型：

DL00	UNKNOWN_GEOMETRY	未知类型
DL01	POINT	点

DL02	LINE or CURVE	线/曲线
DL03	POLYGON or SURFACE	面/曲面
DL04	COLLECTION	其它任何几何类型的集合
DL05	MULTIPOINT	多点，POINT 的集合
DL06	MULTILINE or MULTICURVE	多线，LINE or CURVE 的集合
DL07	MULTIPOLYGON or MULTISURFACE	多面，POLYGON or SURFACE 的集合
DL08	SOLID	实体，由若干个三维闭合的表面组成
DL09	MULTISOLID	SOLID 的集合

表 1 SDO\_GTYPE 的约定

SDO\_GEOMETRY 中的 SDO\_SRID 属性代表了几何对象的坐标系统，它要么是空，要么是一个在 SDO\_COORD\_REF\_SYS 中定义的 SRID 值，比如我们最熟悉的 4326 等等。

SDO\_GEOMETRY 中的 SDO\_POINT 属性是一个 SDO\_POINT\_TYPE 类型的对象，这个类型很简单：

```
CREATE OR REPLACE
TYPE SDO_POINT_TYPE AS OBJECT (
    X      NUMBER,
    Y      NUMBER,
    Z      NUMBER)
```

其实，SDO\_POINT\_TYPE 类型对象就代表了一个点，而在 SDO\_GEOMETRY 中，只有当 SDO\_ELEM\_INFO 和 SDO\_ORDINATES 属性都为空的时候，这个 SDO\_POINT 属性才会被解析。事实上，从下面我们可以看到，SDO\_ELEM\_INFO 和 SDO\_ORDINATES 属性才是一般存放几何对象的地方。但是，Oracle 建议如果是一个空间表中只有点数据，那么使用 SDO\_POINT 进行存储会提高性能。不得不说，Oracle 在这个地方的定义实在是有点不太优雅。

SDO\_GEOMETRY 的 SDO\_ELEM\_INFO 和 SDO\_ORDINATES 属性分别是 SDO\_ELEM\_INFO\_ARRAY 和 SDO\_ORDINATE\_ARRAY 类型，这两个类型都是存放 NUMBER 的数组。SDO\_ELEM\_INFO 和 SDO\_ORDINATES 属性是要配合

起来解析的：

SDO\_ELEM\_INFO 指定了在 SDO\_ORDINATES 属性中存储的坐标形式应当如何解释，依次每 3 个值一组，每个组代表一个 “Element”，每组中的 3 个值分别是：SDO\_STARTING\_OFFSET、SDO\_ETYPE、SDO\_INTERPRETATION，SDO\_STARTING\_OFFSET 代表了从 SDO\_ORDINATES 中开始解析的坐标序号；SDO\_ETYPE 和 SDO\_INTERPRETATION 则配合起来解析（又一对需要配合起来解析属性！）：

SDO_ETYPE	SDO_INTERPRETATION	含义
0	任意值	不支持的类型，比如曲线和样条曲线
1	1	点
1	0	有向点，除了实际点的坐标，还包含虚拟的指向方向
1	N>1	有 n 个点的点集
2	1	直线
2	2	圆弧构成的多段线，1、2、3 点构成第一个圆弧，3、4、5 点构成第二个圆弧...
1003 或 2003	1	多边形，最后一个点必须和第一个点相同以闭合
1003 或 2003	2	圆弧构成的多边形，1、2、3 点构成第一个圆弧，3、4、5 点构成第二个圆弧...
1003 或 2003	3	矩形，包含两个点：左下角和右上角
1003 或 2003	4	圆形，包含 3 个在圆上的点
4	N>1	复合线，有些线段是直线，有些线段是圆弧
1005 或 2005	N>1	复合多边形，有些线段是直线，有些线段是圆弧

表 2 SDO\_ELEM\_INFO 的含义

SDO\_ORDINATES 属性相对就简单一点了，它就是记录所有点坐标的数组，具体如何解析和 SDO\_ETYPE 有关，注意，对于多边形来说，外环坐标点应按逆时针排列，内环坐标点应按顺时针排列，且都需要闭合。Btw：SDO\_ORDINATE 的最大大小为 1,048,576，因此，一个 SDO\_GEOMETRY 对象最大的顶点数：二维点 524,288 个，三维点 349,525 个，四维点 262,144 个。



OK , 到此为止 , Oracle Spatial 对于 SDO\_GEOMETRY 类型的描述算是基本结束了 , 是不是有点让你感到混沌、还带上一点战战兢兢的感觉 ? 希望下面的这个图可以让你稍微减轻一点这种感觉 :

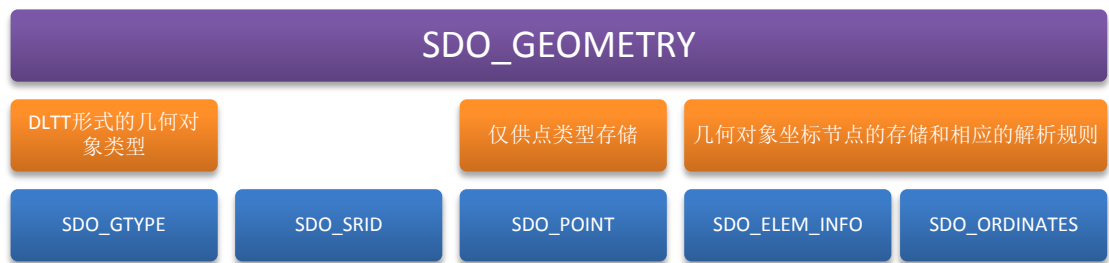
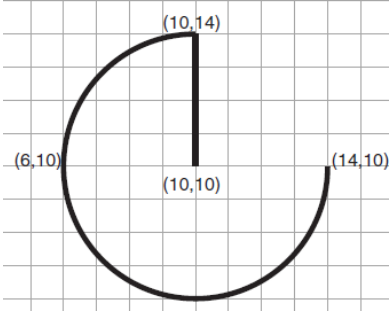
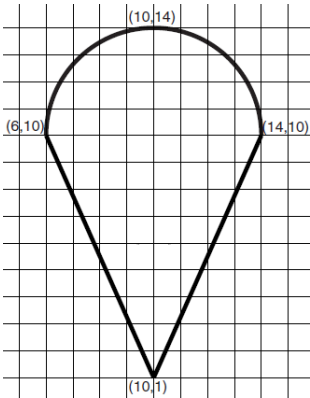
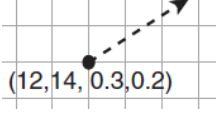


图 1 SDO\_GEOMETRY 的描述

2. 常见几何对象的构造

对于 SDO\_GEOMETRY 这种定义晦涩的类型 , 最好是搞几个例子来对照一下 , 好在《Oracle Spatial Developer' s Guide》里提供了几个简单的 Example , 还配上了很好看的插图 , 在这里一并引用一下 :

<p>矩形</p>	<pre>SDO_GEOMETRY(   2003,-- 面, 表 1   4326,   NULL,   SDO_ELEM_INFO_ARRAY(1,1003,3),-- 矩形, 表 2   SDO_ORDINATE_ARRAY(1,1, 5,7) -- 左下角和右上角 )</pre>
<p>有洞的多边形</p>	<pre>SDO_GEOMETRY(   2003,-- 面, 表 1   4326,   NULL,   SDO_ELEM_INFO_ARRAY(1,1003,1, 19,2003,1), --   从 1 开始多边形, 从 19 开始多边形, 表 2   SDO_ORDINATE_ARRAY(2,4, 4,3, 10,3, 13,5, 13,9,   11,13, 5,13, 2,11, 2,4,--外多边形, 逆时针   7,5, 7,10, 10,10, 10,5, 7,5)-- 内多边形, 顺时针 )</pre>

 <p style="text-align: center;">复合线</p>	<pre>SDO_GEOMETRY(     2002,-- 线,表 1     4326,     NULL,     SDO_ELEM_INFO_ARRAY(1,4,2, 1,2,1, 3,2,2), -- 从     1 开始复合线,从 1 开始直线,从 3 开始圆弧,表 2     SDO_ORDINATE_ARRAY(10,10, 10,14, 6,10, 14,10) )</pre>
 <p style="text-align: center;">复合多边形</p>	<pre>SDO_GEOMETRY(     2003,-- 面,表 1     4326,     NULL,     SDO_ELEM_INFO_ARRAY(1,1005,2, 1,2,1, 5,2,2), --     从 1 开始复合多边形,从 1 开始直线,从 5 开始圆弧     构成的多段线,表 2     SDO_ORDINATE_ARRAY(6,10, 10,1, 14,10, 10,14,     6,10) )</pre>
<p style="text-align: center;">点</p>	<pre>SDO_GEOMETRY(     2001,-- 点,表 1     4326,     SDO_POINT_TYPE(116.39, 39.9, NULL),--使用     SDO_POINT 属性     NULL,     NULL )</pre>
 <p style="text-align: center;">有向点</p>	<pre>SDO_GEOMETRY(     2001,-- 点,表 1     4326,     NULL,     SDO_ELEM_INFO_ARRAY(1,1,1, 3,1,0), --从 1 开始     点,从 3 开始方向,表 2     SDO_ORDINATE_ARRAY(12,14, 0.3,0.2) )</pre>

### 3. 导入数据到 Oracle Spatial

Oracle Spatial 并没有像 ArcGIS 那样有一套从桌面到数据库到服务器到开发包的全方位 GIS 产品体系，因此向 Oracle 中加载数据相比用 ArcGIS 要麻烦得多，同时也没有那么多的数据源格式的支持。Oracle 本身则提供了一个工具 shp2sdo<sup>1</sup>，可以帮助你导入 Shapefile 到 Oracle Spatial 中，这个工具的使用分三个步骤：

第一步，使用 shp2sde 工具生成脚本和数据：

```
[oracle@test2 shp2sdo_linux]$ ./shp2sdo.exe

shp2sdo - Shapefile(r) To Oracle Spatial Converter
Version 2.15 21-May-2004
Copyright 1997,2004 Oracle Corporation
For use with Oracle Spatial.

Input shapefile (no extension): /home/wuyf/world/cities
  Shape file /home/wuyf/world/cities.shp contains 2539 points
Output table [/home/wuyf/world/cities]: cities
Output data model [O]:
Geometry column [GEOM]:
ID column []:
Points stored in SDO_POINT_TYPE ? [Y]:
Use a spatial reference system ID (SRID) ? [N]:
Change tolerance value from the default (0.00000005) ? [N]:
Generate data inside control files ? [N]:
Target database Oracle8i? [N]:
Spatial Data requires more than 6 digits precision? [N]:
Bounds: X=[-176.151564,179.221888] Y=[-54.792000,78.200001]
Override ? [N]:
Processing shapefile /home/wuyf/world/cities into spatial table CITIES
Data model is object-relational
  Geometry column is GEOM
  Points stored in SDO_POINT attributes
  Data is in a separate file(s)
  Control file generation for Oracle9i or higher
  Spatial data loaded with 6 digits of precision
Conversion complete : 2539 points processed
The following files have been created:
  cities.sql : SQL script to create the table
```

---

<sup>1</sup> <http://www.oracle.com/technology/software/products/spatial/index.html>

```
cities.ctl : Control file for loading the table
cities.dat : Data file
```

第二步，在数据库中执行 shp2sdo 生成的 <name>.sql 脚本：

```
SQL> @/home/oracle/shp2sdo_linux/cities.sql
```

第三步，使用 SQL\*Loader 加载 shp2sdo 生成的 <name>.ctl 数据：

```
[oracle@test2 shp2sdo_linux]$ sqlldr spatial/esrichina cities.ctl

SQL*Loader: Release 11.2.0.1.0 - Production on 星期一 4月 19 14:02:26 2010

Copyright (c) 1982, 2009, Oracle and/or its affiliates. All rights reserved.

达到提交点 - 逻辑记录计数 64
达到提交点 - 逻辑记录计数 128
达到提交点 - 逻辑记录计数 192
达到提交点 - 逻辑记录计数 256
达到提交点 - 逻辑记录计数 320
达到提交点 - 逻辑记录计数 384
达到提交点 - 逻辑记录计数 448
...
```

导入成功后我们可以在 Oracle Spatial 中看到这个空间表，其中 GEOM 是导入时设置的几何字段，我们可以打印些数据看一下：

```
SQL> select geom from cities where rownum<10;

GEOM(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
-----
SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(-66.348, -33.316002, NULL), NULL, NULL)
SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(-57.140001, -25.387002, NULL), NULL, NULL)
SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(-56.428002, -25.452999, NULL), NULL, NULL)
SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(-57.150999, -25.623999, NULL), NULL, NULL)
SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(-56.450997, -25.784001, NULL), NULL, NULL)
SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(-58.176999, -26.182998, NULL), NULL, NULL)
SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(-58.295997, -26.867997, NULL), NULL, NULL)
SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(-58.986999, -27.457002, NULL), NULL, NULL)
SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(-58.817997, -27.486999, NULL), NULL, NULL)

已选择 9 行。
```

至于其它一些第三方的工具，能作数据导入的估计也有不少，不过同时也免费估计就够呛了。

## 4. 基于 SDO\_GEOMETRY 的 ST\_GEOMETRY

Oracle Spatial 虽然使用了 SDO\_GEOMETRY 类型作为几何对象的存储，不过它也提供了另外的一种类型 ST\_GEOMETRY，同时在此基础上还提供了一些符合 OGC [Simple Features Access](#) 规范的操作。

首先让我们来看一下这个 ST\_GEOMETRY 的定义：

```
CREATE OR REPLACE
TYPE ST_GEOMETRY AS OBJECT (
  GEOM SDO_GEOMETRY,
  MEMBER FUNCTION GET_SDO_GEOM RETURN SDO_GEOMETRY DETERMINISTIC,
  ...
)
```

可见，Oracle Spatial 的 ST\_GEOMETRY 事实上还是通过对 SDO\_GEOMETRY 的包装实现的。因此，我们可以通过最简单的构造函数来构造一个 ST\_GEOMETRY 对象：

```
ST_GEOMETRY(geom SDO_GEOMETRY);
```

或者，也可以使用具体的 ST\_GEOMETRY 类型来构造，比如用 ST\_POINT 来构造一个点，你会发现内部 Oracle Spatial 还是根据参数生成了一个 SDO\_GEOMETRY 对象存放了起来：

```
SQL> select st_point(1,1) from dual;

ST_POINT(1,1)(GEOM(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES))
-----
ST_POINT(SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(1, 1, NULL), NULL, NULL))
```

因此，表面上看 Oracle Spatial 中与 OGC 规范相关的主要就包括三部分（见图 2）：两张记录几何字段和空间参考的系统表、一些 ST 打头的对象类型和一些 OGC 打头的空间操作函数。但是，由于 ST\_GEOMETRY 使用的是 SDO\_GEOMETRY 进行存储，因此这些 SDO 相关的内容也会被使用到。

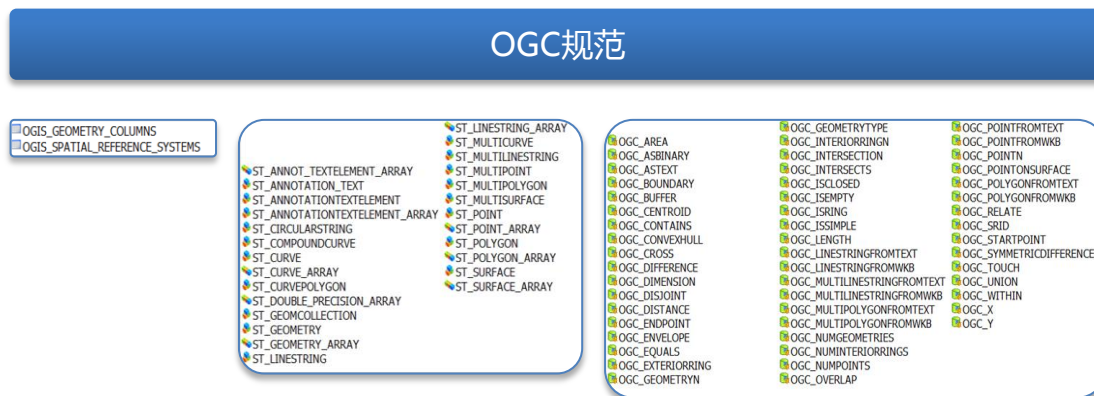


图 2 Oracle Spatial 中与 OGC 规范相关的内容

## • ArcSDE

从 ArcSDE 的 Post Installation 就可以看到 , ArcSDE 支持多种数据存储方式 ,不但支持 ESRI 自身的 ST\_GEOMETRY ,也支持 BLOB 和 SDO\_GEOMETRY 类型。

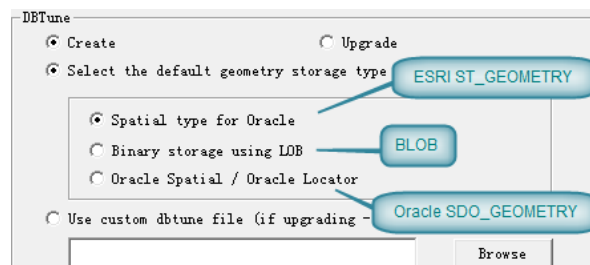


图 3 ArcSDE 支持的存储格式

## 1. ST\_GEOMETRY

ArcSDE 中的 ST\_GEOMETRY 和 Oracle Spatial 中基于 SDO\_GEOMETRY 的 ST\_GEOMETRY 完全是不相干的两个东西。首先让我们来看一下 ArcSDE 中 ST\_GEOMETRY 类型的定义 :

```

CREATE OR REPLACE
Type st_geometry Authid current_user AS object
(
    entity integer,numpts integer,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 integer,points blob,
    constructor Function st_geometry(geom_str clob,srid number) Return self AS result deterministic,
    member Function st_area Return number,
    member Function st_len Return number,
    member Function st_length Return number,
    member Function st_entity Return number,
    member Function st_numpts Return number,
    member Function st_minx Return number,
    member Function st_maxx Return number,
    member Function st_miny Return number,
    member Function st_maxy Return number,
    member Function st_minm Return number,
    member Function st_maxm Return number,
    member Function st_minz Return number,
    member Function st_maxz Return number,
    member Function st_srid Return number,
    static Function get_release Return number) NOT final;

```

可见，一个 ST\_GEOMETRY 中包含了一个几何对象的 x、y、z、m 坐标的范围、空间参考 id、实体个数和点个数、几何对象的面积和长度、具体的节点坐标等信息。其中，节点坐标属性 points 比较特殊，这是一个 blob 类型的值。

让我们打开 ST\_GEOMETRY 类型的构造函数就可以发现，事实上对几何对象的构造是通过 ST\_GEOMETRY\_SHAPELIB\_PKG 这个包进行处理的，而这个包事实上会调用 ESRI 的 ST\_SHAPELIB 这个外部动态链接库，我们可以看一下 ST\_GEOMETRY\_SHAPELIB\_PKG 包里的 geometryfromtext 这个存储过程，这个过程在 ST\_GEOMETRY 的构造函数中被调用了：

```

Procedure geometryfromtext (shptxt      IN      clob,
                           ...,
                           points      IN Out blob )

AS
    language c
    name      "GeometryFromText"
    library  st_shapelib
    WITH CONTEXT
    parameters (

```

CONTEXT, shptxt	ociloblocator, shptxt Indicator short,
..., points );	ociloblocator, points Indicator short

再具体一点，比如我们创建了一个 ST\_POINT 对象，我们可能比较关心在 Oracle 的层面到底上发生了些什么事情？比如我们通过 SQL 执行了这样一条语句：

```
SQL> select sde.st_point(1,1,0) from dual;
```

显然，它会调用 ST\_POINT 的构造函数，这个构造函数大体是这个样子的：

```
constructor Function st_point(pt_x number,pt_y number,srid number)
Return self AS result
IS
...;

Begin
...;

geom_clob := 'POINT('||pt_x||' '||pt_y||')';

SDE.st_geometry_shapelib_pkg.geometryfromtext(geom_clob,spref_r.srid,spref_r.x_offset,spref_r.y_offset,spref_r.xyu
nits, spref_r.z_offset,spref_r.z_scale,spref_r.m_offset,spref_r.m_scale,
spref_r.Definition,geom_type,shape.numpts,shape.entity,shape.minx,shape.miny,
shape.maxx,shape.maxy,shape.minz,shape.maxz,shape.minm,shape.maxm, shape.area,shape.len,shape.points);
...;
End;
```

这个构造函数做了 2 件事情：一是通过参数构造了一个 WKT<sup>1</sup>表述的几何对象字符串；二是通过调用 ST\_SHAPELIB 链接库实现了从字符串到几何对象的转化，返回的一系列值构成了 ST\_POINT 对象。最后存在 Oracle 中的实际上是二进制对象：

```
SQL> select sde.st_point(1,1,0) from dual;

SDE.ST_POINT(1,1,0)(ENTITY, NUMPTS, MINX, MINY, MAXX, MAXY, MINZ, MAXZ, MINM, MAXM, AREA, LEN,
SRID, POINTS)
-----
ST_POINT(1, 1, 1, 1, 1, 1, NULL, NULL, NULL, NULL, 0, 0, 0, '0C0000000100000080A8B3D7AB1780A8B3D7AB17')
```

<sup>1</sup> [http://en.wikipedia.org/wiki/Well-known\\_text](http://en.wikipedia.org/wiki/Well-known_text)



## 2. 常见几何对象的构造

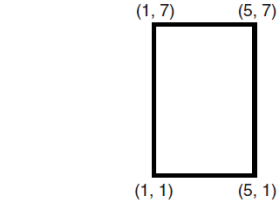
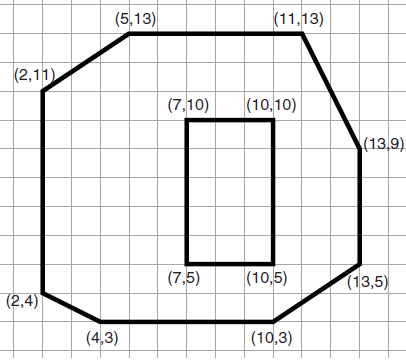
介绍 ESRI 的 ST\_GEOMETRY 的几何对象构造方法比 SDO\_GEOMETRY 就轻松多了，我们已经知道 ST\_GEOMETRY 的各个子类型中的构造函数构造几何对象时都会首根据参数拼装成 WKT 格式的字符串，然后调用 ST\_SHAPELIB 链接库转化到二进制的几何对象返回，那相对更通用的方法就是直接使用 ST\_GEOMETRY 的构造函数，给一个 WKT 格式的字符串作为初始化参数就可以了：

```
SQL> select sde.st_geometry('POINT(1 1)',0) from dual;

SDE.ST_GEOMETRY('POINT(11)',0)(ENTITY, NUMPTS, MINX, MINY, MAXX, MAXY, MINZ, MAXZ, MINM, MAXM,
AREA, LEN, SRID, POINTS)
-----
ST_GEOMETRY(1, 1, 1, 1, 1, 1, NULL, NULL, NULL, NULL, 0, 0, 0,
'0C00000001000000080A8B3D7AB1780A8B3D7AB17')
```

在下面给出一些常见几何对象的示例，其中前两个对象的定义可以和上面

Oracle Spatial 中相同对象的定义作下比较：

 <p>矩形</p>	<p>POLYGON((1 1,5 1,5 7,1 7,1 1))</p>
	<p>POLYGON((2 4,4 3,10 3,13 5,13 9,11 13,5 13,2 11,2 4),(7 5,10 5,10 10,7 10,7 5))</p>

有洞的多边形	
点	POINT(1 1)
线	LINESTRING(0 0,1 1,1 2)
面 ( 多边形 )	POLYGON(((0 0,1 0,1 1, 0 1,0 0),(2 0, 3 0,3 1,2 1,2 0)))
多点	MULTIPOINT(0 0,1 2)
多线	MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4))
多面	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)))
几何对象集合	GEOMETRYCOLLECTION(POINT(2 3),LINESTRING((2 3,3 4)))

## • 几何对象的性能

在几何对象的角度，抛却数据定义的难易程度，使用的关键在于性能，而几何对象的性能在于数据库创建几何对象的速度和几何对象存储占据的数据空间大小，在这里我们分别通过一个存储过程来测试创建常见几何对象的速度，然后通过一个只包含几何字段的表来测试几何对象所占据的存储空间。

### 1.创建几何对象的性能

这里使用这样的程序块进行循环创建几何对象，其中具体的执行块根据不同的存储类型进行调整：

```

declare
    i int;
    obj sde.st_geometry;
begin
    for i in 1..10000 loop
        select sde.st_geometry('POINT(116.39 39.9)',0) into obj from dual;
    end loop;
end;
```

/

对于点、线、面等不同类型对象，下表记录了创建 10000 个对象累计的耗时（秒）。

几何对象 ( WKT 描述 )	Oracle SDO_GEOMETRY		Oracle ST_GEOMETRY		ESRI ST_GEOMETRY	
POINT(116.39 39.9)	SDO_POINT	0.8	FROM_WKT	43	ST_GeomFromText	43
	非 SDO_POINT	1.1	ST_POINT <sup>1</sup>	3.6	ST_POINT	23
LINESTRING(0 0,1 2,2 3,4 3,5 8)	1		FROM_WKT	57	ST_ GeomFromText	46
POLYGON((2 4,4 3,10 3,13 5,13 9,11 13,5 13,2 11,2 4),(7 5,10 5,10 10,7 10,7 5))	1.1		FROM_WKT	57	ST_ GeomFromText	46

从这个表里可以看出，如果使用 SQL 直接创建几何对象，Oracle Spatial 创建 SDO\_GEOMETRY 对象是非常快的，而 ST\_GEOMETRY 因为要解析 WKT，因此创建速度相对就慢很多。总的来说 ESRI ST\_GEOMETRY 比 Oracle ST\_GEOMETRY 要稍微快一点，不过差距不明显。

不过需要说明一下，以上的测试由于都是在 SQL 语句中做，因此 Oracle Spatial 有比较明显的优势，这是由于 SDO\_GEOMETRY 类型会直接读取并存储节点坐标数值，而 ESRI 的 ST\_GEOMETRY 类型不但需要解析文本，而且还要压缩数据。事实上，在实际应用中，ESRI 会通过 ArcGIS 的产品生成二进制的几何对象直接存储到数据库中，这个效率会高很多。

## 2.几何对象的存储空间

为了测试 Oracle Spatial 和 ArcSDE 在真实环境中几何对象存储空间的大小，

---

<sup>1</sup> 使用 ST\_POINT 构造函数创建

测试分别使用了 2 个包含约 1 亿个要素的线数据和面数据进行。其中线数据只包含几何信息，面数据还包含一些属性信息。大部分线数据只包含 2 个节点，而面数据一般都是上百个节点以上。

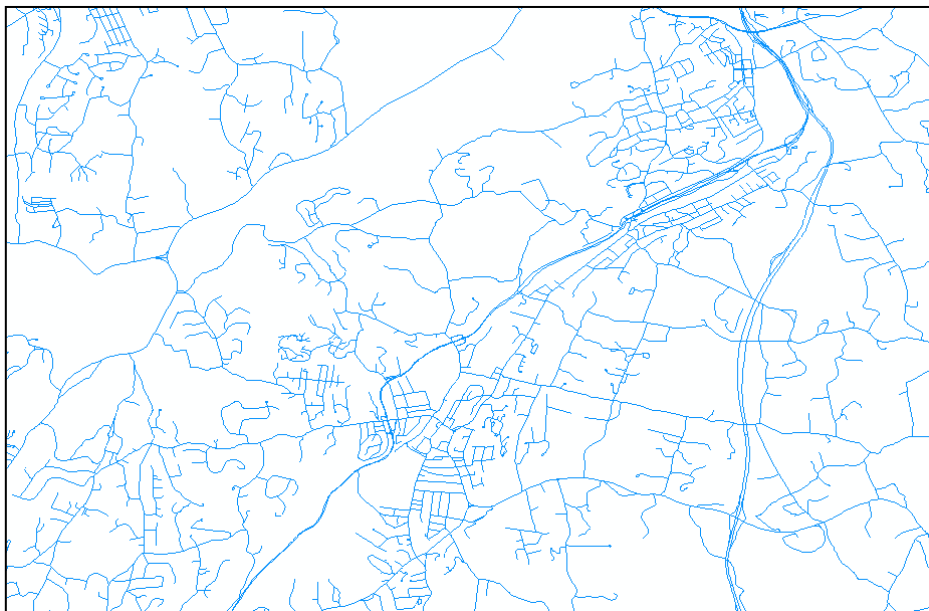


图 4 线数据预览

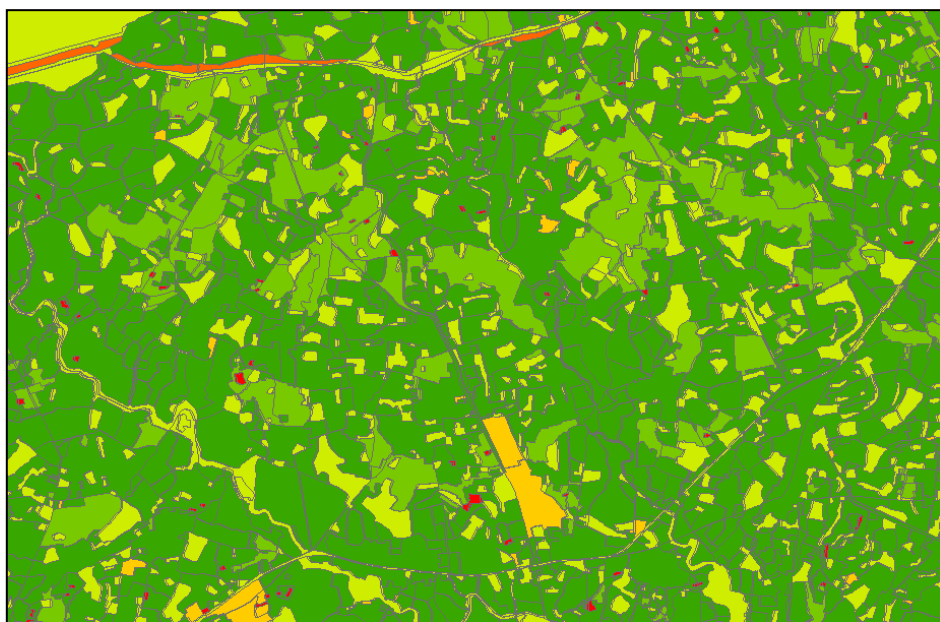


图 5 面数据预览

对这 2 个数据分别进行了存储空间的比较：

数据	存储类型	空间数据表存储大小(G)
线数据	ST_GEOMETRY	14.72
	SDO_GEOMETRY	16.51
面数据	ST_GEOMETRY	49.93
	SDO_GEOMETRY	87.87

可见，随着几何对象节点数的增加，SDO\_GEOMETRY 存储耗费的空间要多不少。

## II. 空间索引

对于空间数据中几何对象的索引显然不能套用数据库现有的索引类型,其不确定的数据结构和数据操作方法使现有的索引并不能适用,因此无论是 Oracle Spatial 还是 ArcSDE 都是采用域索引 ( Domain Index<sup>1</sup> ) 来实现,简单地说就是创建一个数据库中原来没有的新的索引类型。

### • Oracle Spatial

#### 1.索引类型 SPATIAL\_INDEX

Oracle Spatial 中的空间索引极其重要,没建空间索引的空间表就像断了腿的兔子,不但跑不起来,甚至可能比乌龟还慢。

让我们从索引的创建入手来看一下 Oracle Spatial 的空间索引机制:

```
SQL> create index idx_test_index_geom on spatial.test_index(geom) indextype is mdsys.spatial_index;
```

索引已创建。

如果你的数据是用 shp2sdo 等工具导入的,那么直接就可以创建索引;如果是自己创建的空间表,那么必然会遇到 ORA-13203 错误,因为 Oracle Spatial 在创建空间索引时会读取 USER\_SDO\_GEOM\_METADATA 视图中相关的信息,如果没有就会报错,因此,对于自己创建的空间表,又需要创建索引的话(这简直是一句废话,没索引的残废空间表有什么用),就一定要在这个视图插入这个空间表的相关信息:

---

<sup>1</sup> [http://download.oracle.com/docs/cd/E11882\\_01/server.112/e10592/statements\\_5013.htm](http://download.oracle.com/docs/cd/E11882_01/server.112/e10592/statements_5013.htm)

```
SQL> insert into mdsys.user_sdo_geom_metadata(table_name,column_name,diminfo) values
('TEST_INDEX','GEOM',
 SDO_DIM_ARRAY(
 SDO_DIM_ELEMENT('X',1,100000,1),
 SDO_DIM_ELEMENT('Y',1,100000,1)));
```

这里我们告诉 Oracle Spatial，我的这个空间表名字叫“TEST\_INDEX”，几何字段名字叫“GEOM”，X、Y 方向上的坐标范围都是 1 到 100000，容差是 1。顺便看一下 SDO\_DIM\_ELEMENT 的定义，其中几个属性的含义在下面写了点注释：

```
CREATE OR REPLACE
TYPE SDO_DIM_ELEMENT AS OBJECT (
    SDO_DIMNAME    VARCHAR(64), --坐标维的名称
    SDO_LB         NUMBER,      --坐标下界
    SDO_UB         NUMBER,      --坐标上界
    SDO_TOLERANCE  NUMBER )    --坐标容差
```

好了，再把视线转回到空间索引上，上面我们在 TEST\_INDEX 表的 GEOM 字段上创建了基于 R 树<sup>1</sup>空间索引 IDX\_TEST\_INDEX\_GEOM，这个空间索引的索引类型是 MDSYS.SPATIAL\_INDEX，这个肯定比较关键，下面我们看一下这个索引类型的定义：

```
CREATE OR REPLACE INDEXTYPE "MDSYS"."SPATIAL_INDEX" FOR
    "MDSYS"."LOCATOR_WITHIN_DISTANCE" ("MDSYS"."SDO_GEOMETRY", "MDSYS"."SDO_GEOMETRY",
    VARCHAR2),
    "MDSYS"."SDO_ANYINTERACT" ("MDSYS"."SDO_GEOMETRY", "MDSYS"."SDO_GEOMETRY") REWRITE JOIN,
    "MDSYS"."SDO_ANYINTERACT" ("MDSYS"."SDO_TOPO_GEOMETRY",
    "MDSYS"."SDO_TOPO_OBJECT_ARRAY"),
    "MDSYS"."SDO_ANYINTERACT" ("MDSYS"."SDO_TOPO_GEOMETRY", "MDSYS"."SDO_GEOMETRY"),
    "MDSYS"."SDO_ANYINTERACT" ("MDSYS"."SDO_TOPO_GEOMETRY", "MDSYS"."SDO_TOPO_GEOMETRY"),
    "MDSYS"."SDO_ANYINTERACT" ("MDSYS"."SDO_GEOMETRY", "MDSYS"."ST_GEOMETRY"),
    "MDSYS"."SDO_ANYINTERACT" ("MDSYS"."ST_GEOMETRY", "MDSYS"."ST_GEOMETRY"),
    "MDSYS"."SDO_ANYINTERACT" ("MDSYS"."ST_GEOMETRY", "MDSYS"."SDO_GEOMETRY"),
    "MDSYS"."SDO_CONTAINS" ("MDSYS"."SDO_GEOMETRY", "MDSYS"."SDO_GEOMETRY"),
    "MDSYS"."SDO_CONTAINS" ("MDSYS"."SDO_TOPO_GEOMETRY", "MDSYS"."SDO_TOPO_OBJECT_ARRAY"),
    "MDSYS"."SDO_CONTAINS" ("MDSYS"."SDO_TOPO_GEOMETRY", "MDSYS"."SDO_GEOMETRY"),
    "MDSYS"."SDO_CONTAINS" ("MDSYS"."SDO_TOPO_GEOMETRY", "MDSYS"."SDO_TOPO_GEOMETRY"),
    "MDSYS"."SDO_CONTAINS" ("MDSYS"."SDO_GEOMETRY", "MDSYS"."ST_GEOMETRY"),
    "MDSYS"."SDO_CONTAINS" ("MDSYS"."ST_GEOMETRY", "MDSYS"."ST_GEOMETRY"),
    "MDSYS"."SDO_CONTAINS" ("MDSYS"."ST_GEOMETRY", "MDSYS"."SDO_GEOMETRY"),
```

<sup>1</sup> 关于不同算法的空间索引见后续章节

[illegible]



```

"MDSYS"."SDO_ON" ("MDSYS"."SDO_TOPO_GEOMETRY", "MDSYS"."SDO_GEOMETRY"),
"MDSYS"."SDO_ON" ("MDSYS"."SDO_TOPO_GEOMETRY", "MDSYS"."SDO_TOPO_GEOMETRY"),
"MDSYS"."SDO_ON" ("MDSYS"."SDO_GEOMETRY", "MDSYS"."ST_GEOMETRY"),
"MDSYS"."SDO_ON" ("MDSYS"."ST_GEOMETRY", "MDSYS"."ST_GEOMETRY"),
"MDSYS"."SDO_ON" ("MDSYS"."ST_GEOMETRY", "MDSYS"."SDO_GEOMETRY"),
"MDSYS"."SDO_OVERLAPBDYDISJOINT" ("MDSYS"."SDO_GEOMETRY", "MDSYS"."SDO_GEOMETRY"),
"MDSYS"."SDO_OVERLAPBDYDISJOINT" ("MDSYS"."SDO_TOPO_GEOMETRY",
"MDSYS"."SDO_TOPO_OBJECT_ARRAY"),
"MDSYS"."SDO_OVERLAPBDYDISJOINT" ("MDSYS"."SDO_TOPO_GEOMETRY", "MDSYS"."SDO_GEOMETRY"),
"MDSYS"."SDO_OVERLAPBDYDISJOINT" ("MDSYS"."SDO_TOPO_GEOMETRY",
"MDSYS"."SDO_TOPO_GEOMETRY"),
"MDSYS"."SDO_OVERLAPBDYDISJOINT" ("MDSYS"."SDO_GEOMETRY", "MDSYS"."ST_GEOMETRY"),
"MDSYS"."SDO_OVERLAPBDYDISJOINT" ("MDSYS"."ST_GEOMETRY", "MDSYS"."ST_GEOMETRY"),
"MDSYS"."SDO_OVERLAPBDYDISJOINT" ("MDSYS"."ST_GEOMETRY", "MDSYS"."SDO_GEOMETRY"),
"MDSYS"."SDO_OVERLAPBDYINTERSECT" ("MDSYS"."SDO_GEOMETRY", "MDSYS"."SDO_GEOMETRY"),
"MDSYS"."SDO_OVERLAPBDYINTERSECT" ("MDSYS"."SDO_TOPO_GEOMETRY",
"MDSYS"."SDO_TOPO_OBJECT_ARRAY"),
"MDSYS"."SDO_OVERLAPBDYINTERSECT" ("MDSYS"."SDO_TOPO_GEOMETRY", "MDSYS"."SDO_GEOMETRY"),
"MDSYS"."SDO_OVERLAPBDYINTERSECT" ("MDSYS"."SDO_TOPO_GEOMETRY",
"MDSYS"."SDO_TOPO_GEOMETRY"),
"MDSYS"."SDO_OVERLAPBDYINTERSECT" ("MDSYS"."SDO_GEOMETRY", "MDSYS"."ST_GEOMETRY"),
"MDSYS"."SDO_OVERLAPBDYINTERSECT" ("MDSYS"."ST_GEOMETRY", "MDSYS"."ST_GEOMETRY"),
"MDSYS"."SDO_OVERLAPBDYINTERSECT" ("MDSYS"."ST_GEOMETRY", "MDSYS"."SDO_GEOMETRY"),
"MDSYS"."SDO_OVERLAPS" ("MDSYS"."SDO_GEOMETRY", "MDSYS"."SDO_GEOMETRY"),
"MDSYS"."SDO_OVERLAPS" ("MDSYS"."SDO_TOPO_GEOMETRY", "MDSYS"."SDO_TOPO_OBJECT_ARRAY"),
"MDSYS"."SDO_OVERLAPS" ("MDSYS"."SDO_TOPO_GEOMETRY", "MDSYS"."SDO_GEOMETRY"),
"MDSYS"."SDO_OVERLAPS" ("MDSYS"."SDO_TOPO_GEOMETRY", "MDSYS"."SDO_TOPO_GEOMETRY"),
"MDSYS"."SDO_OVERLAPS" ("MDSYS"."SDO_GEOMETRY", "MDSYS"."ST_GEOMETRY"),
"MDSYS"."SDO_OVERLAPS" ("MDSYS"."ST_GEOMETRY", "MDSYS"."ST_GEOMETRY"),
"MDSYS"."SDO_OVERLAPS" ("MDSYS"."ST_GEOMETRY", "MDSYS"."SDO_GEOMETRY"),
"MDSYS"."SDO_RELATE" ("MDSYS"."SDO_GEOMETRY", "MDSYS"."SDO_GEOMETRY", VARCHAR2) REWRITE
JOIN,
"MDSYS"."SDO_RELATE" ("MDSYS"."ST_GEOMETRY", "MDSYS"."SDO_GEOMETRY", VARCHAR2),
"MDSYS"."SDO_RELATE" ("MDSYS"."ST_GEOMETRY", "MDSYS"."ST_GEOMETRY", VARCHAR2),
"MDSYS"."SDO_RELATE" ("MDSYS"."SDO_GEOMETRY", "MDSYS"."ST_GEOMETRY", VARCHAR2),
"MDSYS"."SDO_RTREE_FILTER" ("MDSYS"."SDO_GEOMETRY", "MDSYS"."SDO_GEOMETRY", VARCHAR2),
"MDSYS"."SDO_RTREE_RELATE" ("MDSYS"."SDO_GEOMETRY", "MDSYS"."SDO_GEOMETRY", VARCHAR2),
"MDSYS"."SDO_TOUCH" ("MDSYS"."SDO_GEOMETRY", "MDSYS"."SDO_GEOMETRY"),
"MDSYS"."SDO_TOUCH" ("MDSYS"."SDO_TOPO_GEOMETRY", "MDSYS"."SDO_TOPO_OBJECT_ARRAY"),
"MDSYS"."SDO_TOUCH" ("MDSYS"."SDO_TOPO_GEOMETRY", "MDSYS"."SDO_GEOMETRY"),
"MDSYS"."SDO_TOUCH" ("MDSYS"."SDO_TOPO_GEOMETRY", "MDSYS"."SDO_TOPO_GEOMETRY"),
"MDSYS"."SDO_TOUCH" ("MDSYS"."ST_GEOMETRY", "MDSYS"."SDO_GEOMETRY"),
"MDSYS"."SDO_TOUCH" ("MDSYS"."ST_GEOMETRY", "MDSYS"."ST_GEOMETRY"),
"MDSYS"."SDO_TOUCH" ("MDSYS"."SDO_GEOMETRY", "MDSYS"."ST_GEOMETRY"),
"MDSYS"."SDO_WITHIN_DISTANCE" ("MDSYS"."SDO_GEOMETRY", "MDSYS"."SDO_GEOMETRY", VARCHAR2)
REWRITE JOIN,
"MDSYS"."SDO_WITHIN_DISTANCE" ("MDSYS"."ST_GEOMETRY", "MDSYS"."SDO_GEOMETRY", VARCHAR2),
"MDSYS"."SDO_WITHIN_DISTANCE" ("MDSYS"."ST_GEOMETRY", "MDSYS"."ST_GEOMETRY", VARCHAR2),
"MDSYS"."SDO_WITHIN_DISTANCE" ("MDSYS"."SDO_GEOMETRY", "MDSYS"."ST_GEOMETRY", VARCHAR2)
USING "MDSYS"."SDO_INDEX_METHOD_10I"
WITH REBUILD ONLINE
WITH ORDER BY "MDSYS"."SDO_NN_DISTANCE" (NUMBER),
"MDSYS"."SDO_NN_DISTANCE" (NUMBER),

```

```
"MDSYS"."SDO_NN_DISTANCE" (NUMBER)
WITH LOCAL RANGE PARTITION
```

是不是翻页翻得有点不耐烦？之所以把这个空间索引所有的内容贴上来，是想看看在 Oracle Spatial 中到底哪些操作支持空间索引，这里发现一个很关键的问题是：为什么没有 OGC 的操作？如果在 Oracle Spatial 中使用一个 OGC\_INTERSECTS 操作会有什么后果？下面就此进行了一个测试：

首先新建一个几何字段为 MDSYS.ST\_GEOMETRY 类型的空间表名为“TEST\_INDEX\_ST”，插入 10 万条记录，并创建空间索引：

```
SQL> desc test_index_st
 名称                                是否为空? 类型
-----
GEOM                                MDSYS.ST_GEOMETRY

SQL> select INDEX_NAME from user_indexes where table_name='TEST_INDEX_ST';

INDEX_NAME
-----
IDX_TEST_INDEX_ST_GEOM
SYS_IL0000079131C00009$$
SYS_IL0000079131C00008$$
```

如果使用上面 SPATIAL\_INDEX 中包含的操作如 SDO\_ANYINTERACT，设定空间过滤条件取其中 1 条记录，通过执行计划可以发现，查询会走空间索引：

```
SQL> set autot on
SQL> select * from test_index_st where SDO_ANYINTERACT(geom,
ST_GEOMETRY(SDO_GEOMETRY(2001,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1,1),SDO_ORDINATE_ARRAY(10
000,10000))))='TRUE';

GEOM(GEOM(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES))
-----
ST_GEOMETRY(SDO_GEOMETRY(2001, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 1),
SDO_ORDINATE_ARRAY(10000, 10000)))

已用时间: 00: 00: 00.01

执行计划
-----
Plan hash value: 2758081403

-----
| Id | Operation                                | Name                                | Rows | Bytes | Cost (%CPU)| Time |
-----
```

```

-----
| 0 | SELECT STATEMENT          |          | 1104 | 4283K | 184  (0)| 00:00:03 |
| 1 |  TABLE ACCESS BY INDEX ROWID| TEST_INDEX_ST | 1104 | 4283K | 184  (0)| 00:00:03 |
|* 2 |  DOMAIN INDEX                | IDX_TEST_INDEX_ST_GEOM |      |      |      |      |
-----

Predicate Information (identified by operation id):
-----

 2 -
access("MDSYS"."SDO_ANYINTERACT"("GEOM","ST_GEOMETRY"("MDSYS"."SDO_GEOMETRY"(2001,NULL,
NULL,"SDO_ELEM_INFO_ARRAY"(1,1,1),"SDO_ORDINATE_ARRAY"(10000,10000))))='TRUE')

```

但是如果是 OGC\_INTERSECTS 操作 ,同样的过滤条件 ,取其中的一条记录 ,  
你会发现 Oracle 会进行全表扫描 :

```

SQL> set autot on
SQL> select * from test_index_st where MDSYS.OCG_INTERSECTS(geom,
ST_GEOMETRY(SDO_GEOMETRY(2001,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1,1),SDO_ORDINATE_ARRAY(10
000,10000))))=1;

GEOM(GEOM(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES))
-----
ST_GEOMETRY(SDO_GEOMETRY(2001, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 1),
SDO_ORDINATE_ARRAY(10000, 10000)))

已用时间: 00: 04: 01.04

执行计划
-----
Plan hash value: 1104487544

-----
| Id | Operation          | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----
| 0  | SELECT STATEMENT   |               | 1104  | 4270K | 439  (3)| 00:00:06 |
|* 1  |  TABLE ACCESS FULL| TEST_INDEX_ST | 1104  | 4270K | 439  (3)| 00:00:06 |
-----

Predicate Information (identified by operation id):
-----

 1 - filter("MDSYS"."OCG_INTERSECTS"("GEOM","ST_GEOMETRY"("MDSYS"."SDO_GE
OMETRY"(2001,NULL,NULL,"SDO_ELEM_INFO_ARRAY"(1,1,1),"SDO_ORDINATE_ARRAY"(10
000,10000))))=1)

```

对于 OGC 来说 ,这简直就是一个杯具。可见 ,有了空间索引的 Oracle  
Spatial 虽然不是断腿的兔子 ,但是在 OGC 操作的场景下 就像把 Oracle Spatial

这只兔子扔到水里和乌龟比游泳——而且，或许别人还不是乌龟，有可能是忍者神龟呢？

## 2.R 树空间索引

自然界中的几何对象都是奇形怪状的，对这样的数据进行管理是一件很头疼的事情。因此，GIS 的工程师们化繁为简，一种方法就是把几何对象的边界范围拿出来作为一个检索的依据（几何对象肯定在它内部），这是一个矩形的范围，再对它进行管理就方便多了。在 Oracle Spatial 中，基于 R 树的空间索引采用的就是这种方法（这也是推荐的索引算法），这个矩形叫 MBR：Minimum Bounding Rectangle。

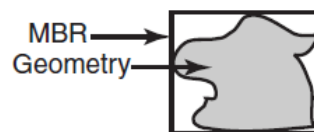


图 6 Oracle Spatial 中几何对象的 MBR

在 Oracle Spatial 中，一个空间表里所有的几何对象就是通过它们的 MBR 的 R 树索引进行管理的，从下面的图中很容易可以理解，MBR 索引的根节点包括 A、B 两个较大的 MBR，当一个查询可以很快速地找到 A 后，再查找其中的 a 或 b 这两个比较小的 MBR，如果找到了 a，那么再最终找到 1 或者 2 这两个最小的 MBR。



### 3. 四叉树空间索引

Oracle Spatial 并不鼓励使用四叉树空间索引，除非某些特殊场景并且你确认采用四叉树索引会带来好处<sup>1</sup>。Oracle 提供了一个表格对比了 R 树空间索引和四叉树空间索引的一些特点。

R 树	四叉树
不能很好地近似几何对象（MBR 对奇形怪状的几何对象的近似比较粗略）	可以较好地近似几何对象
索引创建和调整比较简单	索引调整较复杂，而且索引的参数对性能影响较大
存储空间较小	存储空间较大
SDO_NN 操作较快，同时还可以指定操作的 SDO_BATCH_SIZE 参数	相反
大批量更新数据时可能会降低空间表的访问效率	数据更新于访问效率无关
可以对 4 个维度进行索引	只能索引 2 个维度
对于地理坐标系的数据，并会使用到 SDO_WITHIN_DISTANCE 操作时，建议使用 R 树索引	相反
对于 whole-Earth 数据 <sup>2</sup> 只能使用 R 树索引	相反

四叉树空间索引的原理像地图切片的原理，如图 8，可以想象，分级的级别越多，对几何对象的描述会更准确。

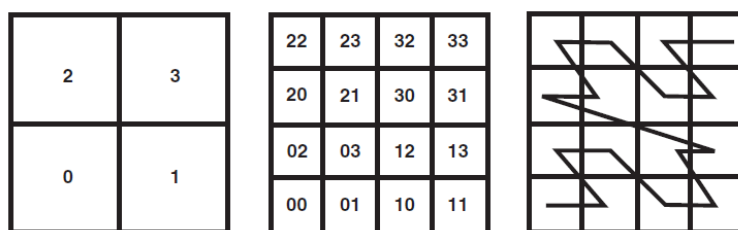


图 8 Oracle Spatial 四叉树空间索引示意图

<sup>1</sup> 《Oracle Spatial Quadtree Indexing》:The use of spatial quadtree indexes is discouraged. You are strongly encouraged to use R-tree indexing for spatial indexes, unless you need to continue using quadtree indexes for special situations.

<sup>2</sup> Oracle Spatial 中包含一个 whole-Earth 几何模型，可以直接在地理坐标系的数据上根据地球表面曲率等参数进行真实长度、面积的计算，类似内置的投影算法

一般在 Oracle Spatial 创建四叉树空间索引采用固定切片范围的方式,这种方式根据当前的地图范围和 SDO\_LEVEL 参数的设置,如果 SDO\_LEVEL 为 1,那就相当于整个图层一个切片 如果 SDO\_LEVEL 为 2 则整个图层等分为 4 块;依次类推。另外,在四叉树索引中,几何对象的每个 Element 都会被索引,如图 9。

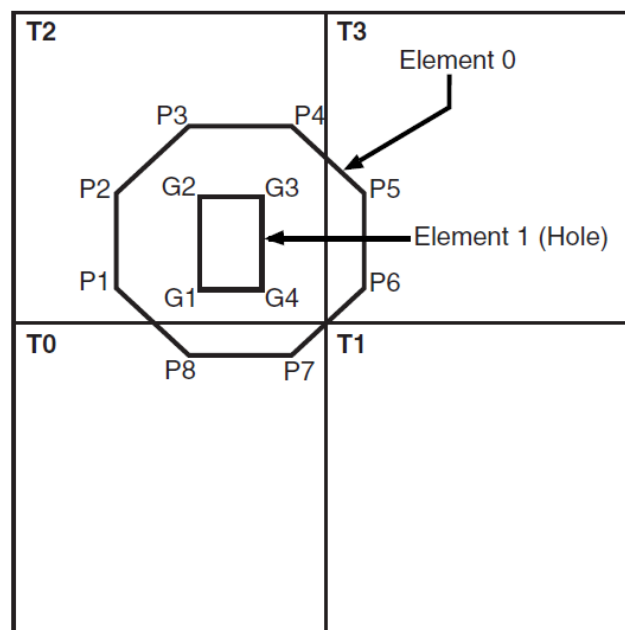


图 9 四叉树空间索引中几何对象的 Element

通过如下的方式可以创建一个固定切片大小、8 级深度的四叉树空间索引：

```
SQL> create index idx_continent_geom on spatial.continent(geom) indextype is mdsys.spatial_index
parameters('SDO_LEVEL=8');
```

索引已创建。

已用时间: 00:05:11.48

```
SQL> select sdo_index_type,sdo_index_table from user_sdo_index_metadata where
sdo_index_name='IDX_CONTINENT_GEOM';
```

SDO\_INDEX\_TYPE

SDO\_INDEX\_TABLE

QTREE

MDQT\_13537\$

这里存放索引数据的表名字叫 MDQT\_13537\$ ,可以查看这个表的结构和 R 树索引的表已经完全不同了：

```
SQL> desc MDQT_13537$
```

名称	是否为空?	类型
SDO_CODE		RAW(16)
SDO_ROWID		ROWID
SDO_STATUS		VARCHAR2(1)

```
SQL> select * from MDQT_13537$ where rownum<10;
```

SDO_CODE	SDO_ROWID	SDO_STATUS
CBD9	AAATUWAAGAAABNGAAA	B
CBDC	AAATUWAAGAAABNGAAA	B
CBDD	AAATUWAAGAAABNGAAA	I
CBDE	AAATUWAAGAAABNGAAA	B
CBDF	AAATUWAAGAAABNGAAA	B
CC00	AAATUWAAGAAABNGAAA	I
CC01	AAATUWAAGAAABNGAAA	I
CC02	AAATUWAAGAAABNGAAA	I
CC03	AAATUWAAGAAABNGAAA	I

其中 SDO\_STATUS 的值如果是 B 表示几何对象和当前的切片边界相交；如果是 I 表示几何对象在当前切片之内。

## • ArcSDE

### 1.索引类型 ST\_SPATIAL\_INDEX

这里我们首先看一下 ArcSDE 的索引类型 ST\_SPATIAL\_INDEX 的定义：

```
CREATE OR REPLACE INDEXTYPE "SDE"."ST_SPATIAL_INDEX" FOR
  "SDE"."ST_CONTAINS" ("SDE"."ST_GEOMETRY", "SDE"."ST_GEOMETRY"),
  "SDE"."ST_CROSSES" ("SDE"."ST_GEOMETRY", "SDE"."ST_GEOMETRY"),
  "SDE"."ST_ENVINTERSECTS" ("SDE"."ST_GEOMETRY", NUMBER, NUMBER, NUMBER, NUMBER),
  "SDE"."ST_ENVINTERSECTS" ("SDE"."ST_GEOMETRY", "SDE"."ST_GEOMETRY"),
  "SDE"."ST_EQUALS" ("SDE"."ST_GEOMETRY", "SDE"."ST_GEOMETRY"),
  "SDE"."ST_INTERSECTS" ("SDE"."ST_GEOMETRY", "SDE"."ST_GEOMETRY"),
  "SDE"."ST_ORDERINGEQUALS" ("SDE"."ST_GEOMETRY", "SDE"."ST_GEOMETRY"),
  "SDE"."ST_OVERLAPS" ("SDE"."ST_GEOMETRY", "SDE"."ST_GEOMETRY"),
  "SDE"."ST_RELATE" ("SDE"."ST_GEOMETRY", "SDE"."ST_GEOMETRY", VARCHAR2),
  "SDE"."ST_TOUCHES" ("SDE"."ST_GEOMETRY", "SDE"."ST_GEOMETRY"),
  "SDE"."ST_WITHIN" ("SDE"."ST_GEOMETRY", "SDE"."ST_GEOMETRY")
  USING "SDE"."ST_DOMAIN_METHODS"
```



我们知道 ESRI 的 ST\_GEOMETRY 类型是符合 OGC 规范的几何类型，上述定义中所有基于 ST\_GEOMETRY 的空间关系操作也都符合 OGC 规范；除了 OGC 规范，ArcSDE 中还包括诸如 ST\_ENVINTERSECTS 这样非常有用的操作。数据库在执行所有这些操作的时候，都会判断是否应该使用索引来加快查询。

## 2. 格网空间索引

ArcSDE 在不同的数据库中使用不同的索引算法，比如在 PostgreSQL 和 Informix 中使用的是 R 树索引；而 Oracle 和 DB2 中使用的是格网索引。我们现在的的环境是 Oracle，因此这里的空间索引就是格网索引。

如果打开 ArcCatalog，打开一个 Feature Class 的属性对话框可以看到这样的内容：

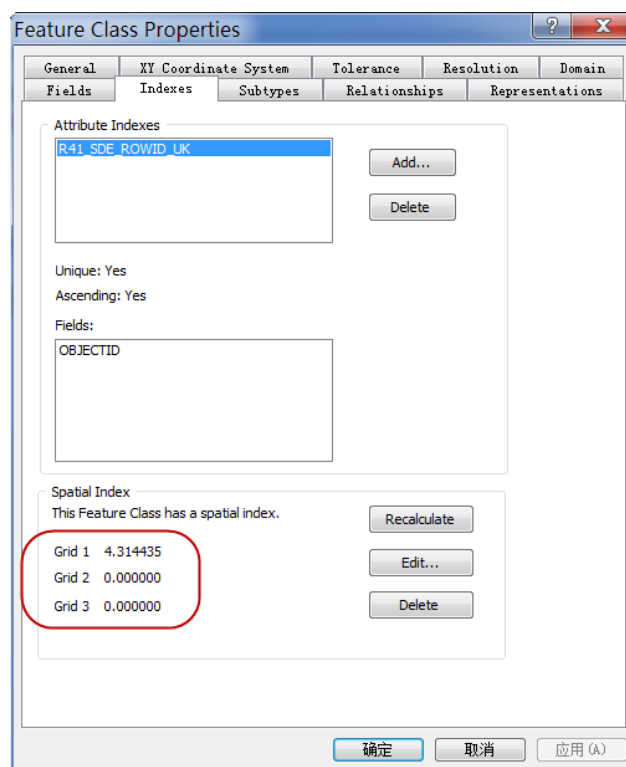


图 10 ArcCatalog 中 Feature Class 的空间索引属性

这里的 3 个“Grid <n>”属性对应的就是不同级别的网格的尺寸。在 ArcSDE 中最多可以对空间数据创建 3 级格网索引，每一级索引都有相应的网格尺寸：第 1 级索引的网格最小，第 2 级的网格必须是第 1 级大小的 3 倍以上，第 3 级网格也必须是第 2 级大小的三倍以上。当然，不少情况下使用 1 级索引就够了，那这时 2 级索引和 3 级索引的网格大小就是 0。

比如上面在 ArcCatalog 中查看过的名为“CITIES”的 Feature Class，如果查看一下它的空间索引定义是这样的：

```
CREATE INDEX "SDE"."A3_IX1" ON "SDE"."CITIES" ("SHAPE")
  INDEXTYPE IS "SDE"."ST_SPATIAL_INDEX" PARAMETERS ('ST_GRIDS = 4.314434698311 ST_SRID = 2
  ST_COMMIT_ROWS = 10000 PCTFREE 0 INITRANS 4')
```

从这个定义可以知道，我们在这个空间表上创建了 1 级空间索引 A3\_IX1，空间索引的网格大小约为 4.3，因为这是一个世界城市的分布，其数据分布在 X 方向约-180 到 180、Y 方向约-60 到 80 的范围内，因此这个数据集大概会被分为 80×30 的格网。

这个索引对应的索引组织表名叫 S3\_IDX\$，在这个表里存放了所有几何对象的索引数据，查看一下相关的信息：

```
SQL> desc S3_IDX$
 名称                                是否为空? 类型
-----
GX                                NOT NULL NUMBER(38)
GY                                NOT NULL NUMBER(38)
MINX                              NOT NULL NUMBER(38)
MINY                              NOT NULL NUMBER(38)
MAXX                              NOT NULL NUMBER(38)
MAXY                              NOT NULL NUMBER(38)
SP_ID                             NOT NULL ROWID

SQL> select count(*) from S3_IDX$;

COUNT(*)
-----
      2539
```

```

已用时间: 00: 00: 00.01
SQL> select * from S3_IDX$ where rownum<10;

      GX      GY      MINX      MINY      MAXX      MAXY SP_ID
-----
      75      103 3.2435E+11 4.4537E+11 3.2435E+11 4.4537E+11 AAASkyAAFAAAB7SAAM
      75      103 3.2635E+11 4.4554E+11 3.2635E+11 4.4554E+11 AAASkyAAFAAAB7SAAL
      76       80 3.2907E+11 3.4683E+11 3.2907E+11 3.4683E+11 AAASkyAAFAAAB7KAAb
      76       80 3.3077E+11 3.4838E+11 3.3077E+11 3.4838E+11 AAASkyAAFAAAB7KAAZ
      76       80 3.3170E+11 3.4521E+11 3.3170E+11 3.4521E+11 AAASkyAAFAAAB7KAAC
      76       83 3.3172E+11 3.6106E+11 3.3172E+11 3.6106E+11 AAASkyAAFAAAB7LAAE
      76       84 3.2834E+11 3.6458E+11 3.2834E+11 3.6458E+11 AAASkyAAFAAAB7LAAB
      76       84 3.2902E+11 3.6541E+11 3.2902E+11 3.6541E+11 AAASkyAAFAAAB7LAAA
      76       84 3.2935E+11 3.6652E+11 3.2935E+11 3.6652E+11 AAASkyAAFAAAB7LAAM

SQL> select min(GX),max(GX),min(GY),max(GY) from S3_IDX$;

  MIN(GX)  MAX(GX)  MIN(GY)  MAX(GY)
-----
       51      134       80      110

```

从上面可以看到，在 S3\_IDX\$表中存放了与 CITIES 表相同数量的记录，因为 CITIES 表是一个点层，因此每个几何对象（点）仅出现在一个空间索引中，如果是线层或者面层会出现比几何对象多的索引记录。S3\_IDX\$表的 SP\_ID 字段存放了该索引对应的空间数据记录的 ROWID；GX、GY 字段分别对应了 X 和 Y 方向的网格序号，在这里我也把 GX 和 GY 的最大最小值打印了出来，你可以发现 Max(GX)-Min(GX)约为 80、Max(GY)-Min(GY)约为 30，和我们上面估算的网格数量相符。同时，你可能也注意到 Min(GX)和 Min(GY)并不是 0，因为这个 Feature Class 的 Domain 最小为 X、Y 均为-400，网格的序号是以此为原点进行计算的。

另外，ArcSDE 对这个索引组织表 S3\_IDX\$还创建有索引，这涉及到另外两个索引 S3\$\_IX1、S3\$\_IX2：

```

CREATE UNIQUE INDEX "SDE"."S3$_IX1" ON "SDE"."S3_IDX$" ("GX", "GY", "MAXX", "MAXY", "MINX", "MINY",
"S3_ID")
PCTFREE 0 INITRANS 4 MAXTRANS 255 COMPUTE STATISTICS
STORAGE(INITIAL 524288 NEXT 524288 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT

```

```

CELL_FLASH_CACHE DEFAULT)
TABLESPACE "SDE"

CREATE INDEX "SDE"."S3$_IX2" ON "SDE"."S3_IDX$" ("SP_ID")
PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
STORAGE(INITIAL 524288 NEXT 524288 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT
CELL_FLASH_CACHE DEFAULT)
TABLESPACE "SDE"

```

总的来说，ArcSDE 中格网索引的结构如图 11 所示，btw，图中的名称实际有可能有变化，比如 A<n>\_IX1 就可以是自己取的任何名字，具体以 ArcSDE 中的 ST\_GEOMETRY\_INDEX 表中的记录为准。

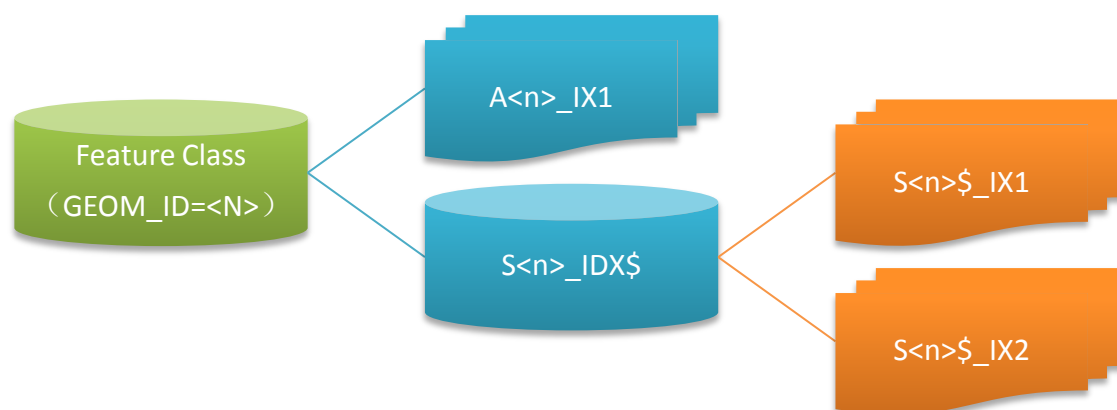


图 11 ArcSDE 格网索引在数据库中的组织结构

### 3. 格网索引的创建和调整

ArcSDE 提供了最多 3 级的空间索引，如何选择空间索引的层级、格网的网格大小设置多少合适等都关系到空间数据的性能。这里我们通过一个简单的 Polygon 图层入手，探讨一些格网空间索引的性能影响因素。

首先我们在 ArcSDE 中准备一个 WGS84 的面层，名为 “TESTGRID”。在这个面层上我们添加 2 个要素，一个较小，一个较大，如图 12 所示。为了提供一些参考信息，图中还将 WGS84 坐标范围按照 30 度的尺寸进行了分割，显示在这 2 个要素之下。

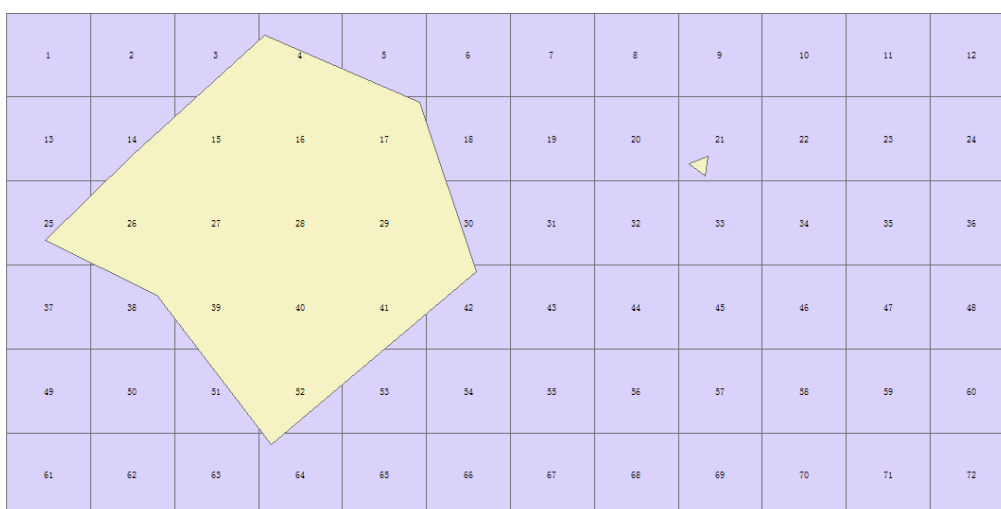


图 12 ArcSDE 中有 2 个要素的面层

上述的数据如果我们创建 3 级索引,为了直观起见,我们可以把最小的网格尺寸定位 30,也就是图 12 中方格的大小。因此网格最大的第 3 级可以选择尺寸为 270×270,那么第 2 级可以设置为 90×90,第 1 级设置为 30×30。我们如下创建空间索引:

```
SQL> CREATE INDEX A141_IX1 on TESTGRID(shape)
      INDEXTYPE is ST_SPATIAL_INDEX
      PARAMETERS ('ST_GRIDS=30,90,270 ST_SRID=2');
odciindexcreate
CREATE TABLE SDE.S141_IDX$ (gx integer, gy integer, minx integer,miny integer, maxx integer, maxy integer,
sp_id
rowid,constraint S141_IX1 primary key(gx,gy,maxx,maxy,minx,miny,sp_id)) organization index pctthreshold 5
pctfree 0
initrans 4

索引已创建。
```

让我们看看这个数据集中的 2 个要素是如何被索引的:

```
SQL> select * from S141_IDX$;

      GX      GY      MINX      MINY      MAXX      MAXY      SP_ID
-----
33554432 33554433 2.3376E+11 3.3580E+11 3.8784E+11 4.8211E+11 AAATb9AAFAAAEJIAAA
33554433 33554433 2.3376E+11 3.3580E+11 3.8784E+11 4.8211E+11 AAATb9AAFAAAEJIAAA
      15      14 4.6386E+11 4.3176E+11 4.7062E+11 4.3886E+11 AAATb9AAFAAAEJIAAB

SQL> select * from testgrid where rowid='AAATb9AAFAAAEJIAAA';
```

```

OBJECTID
-----
SHAPE(ENTITY, NUMPTS, MINX, MINY, MAXX, MAXY, MINZ, MAXZ, MINM, MAXM, AREA, LEN, SRID, POINTS)
-----
1
ST_GEOMETRY(8, 8, -166.23891, -64.197952, -12.163823, 82.105802, NULL, NULL, NULL, NULL, 12078.8885,
439.10602, 2, 'A001
000001000000BEE79CA1B50FB1D3C6A0C819C89FA3CDE701DA85E8C8E2019FEDA487A902F8E9B481920185
D4FDCDB002F9EFC4E08D03A8A697A4A204
99F1A8D8CC03E683F0859701BDBDB2CFC203C4BDF6ED9C0384D1B5DEB201DA88B0B8E002CFA1AFDBBF02')

SQL> select * from testgrid where rowid='AAATb9AAFAAAEJIAAB';

OBJECTID
-----
SHAPE(ENTITY, NUMPTS, MINX, MINY, MAXX, MAXY, MINZ, MAXZ, MINM, MAXM, AREA, LEN, SRID, POINTS)
-----
2
ST_GEOMETRY(8, 4, 63.8600683, 31.7610922, 70.6177474, 38.8566553, NULL, NULL, NULL, NULL, 22.6047828,
21.6768769, 2, '2A
00000001000000B0BF87B0B21BBA86A9DEC519CD81D0AC32F8E6EC9114A89AF7E52ACCE780DD20A5E6D8C
60784CEEDEE34')

```

从节点数目可以判断，'AAATb9AAFAAAEJIAAA'对应的是大的多边形，'AAATb9AAFAAAEJIAAB'对应的是小的多边形。先看小的多边形，GX、GY 分别为 15、14，可见小多边形是在网格尺寸为 30 的第 1 级上被索引的；而大多边形被索引了 2 次，应该是在其它级别上被索引的，但是 GX、GY 的值不能用前面的简单的算法来解释，这是为什么呢？

事实上，这个大的多边形是在第 3 级上进行的索引，也就是说 ArcSDE 自己会判断一个几何对象到底应该在哪几级上进行索引，关于这个算法，可以从 ArcSDE 的 SPX\_UTIL 包里找到：

```

Procedure compute_feat_grid_envp (gsize1  IN integer,
                                gsize2  IN integer,
                                gsize3  IN integer,
                                e_minx  IN integer,
                                e_miny  IN integer,
                                e_maxx  IN integer,
                                e_maxy  IN integer,
                                g_minx  Out integer,
                                g_miny  Out integer,

```

g\_maxx

Out integer,

g\_maxy

Out integer)

IS

Begin

g\_minx := trunc(e\_minx / gsize1);

g\_miny := trunc(e\_miny / gsize1);

g\_maxx := trunc(e\_maxx / gsize1);

g\_maxy := trunc(e\_maxy / gsize1);

If ((g\_maxx - g\_minx + 1) \* (g\_maxy - g\_miny + 1) > max\_grids\_per\_level

AND gsize2 > 0) THEN

g\_minx := trunc(e\_minx / gsize2);

g\_miny := trunc(e\_miny / gsize2);

g\_maxx := trunc(e\_maxx / gsize2);

g\_maxy := trunc(e\_maxy / gsize2);

If ((g\_maxx - g\_minx + 1) \* (g\_maxy - g\_miny + 1) > max\_grids\_per\_level

AND gsize3 > 0) THEN

g\_minx := trunc(e\_minx / gsize3);

g\_miny := trunc(e\_miny / gsize3);

g\_maxx := trunc(e\_maxx / gsize3);

g\_maxy := trunc(e\_maxy / gsize3);

g\_minx := g\_minx + grid\_level\_mask\_2;

g\_miny := g\_miny + grid\_level\_mask\_2;

g\_maxx := g\_maxx + grid\_level\_mask\_2;

g\_maxy := g\_maxy + grid\_level\_mask\_2;

ELSE

g\_minx := g\_minx + grid\_level\_mask\_1;

g\_miny := g\_miny + grid\_level\_mask\_1;

g\_maxx := g\_maxx + grid\_level\_mask\_1;

g\_maxy := g\_maxy + grid\_level\_mask\_1;

End If;

End If;

End compute\_feat\_grid\_envp;

其中有几个常量，在 SPX\_UTIL 的包头里可以找到：

max_grids_per_level	Constant pls_integer := 4;
grid_level_mask_1	Constant pls_integer := 16777216;
grid_level_mask_2	Constant pls_integer := 33554432;

也就是说 ArcSDE 索引一个几何对象的时候是从第 1 级格网开始判断 ,如果某个几何对象跨越的网格数大于 1 个( 或者说 1 个网格不能包含某个几何对象 ) 同时第 2 级的网格尺寸大于 0 的时候， ArcSDE 就会在第 2 级网格上再进行类似的计算；同理，如果还有第 3 级网格可用，ArcSDE 还有可能把该几何对象索

引到第 3 级网格上去。总的来说，ArcSDE 试图用最少的网格来索引几何对象。

另外，为了区分不同级别的网格，这里还有 `grid_level_mask_1=16777216` 和 `grid_level_mask_2=33554432` 两个常量，分别用于第 2 级和第 3 级网格序号的标识，不同级别的网格会根据当前格网的等级分别加上 0（第 1 级）或 `grid_level_mask_1`（第 2 级）或 `grid_level_mask_2`（第 3 级）。由此可见，上面那个大的多边形被索引的两个网格的 GX、GY 应该减去 33554432，也就是分别是 `GX=0, GY=1` 和 `GX=1, GY=1`，这是个第 3 级的索引。

基于上面的原则，我们可以对如何选取空间索引的网格尺寸应该有一些原则性的判断：

- 如果没有特别的需要，可以使用一级索引的话用一级索引会比较好，因为如果有多级索引，查询的时候会扫描所有的索引级别
- 对于几何对象大小变化较大，并且各个量级的对象数量相当的数据才可能需要设置多级索引
- 对于点数据，只需要设置一级索引（几何对象大小没有变化），并且设置网格到较大的尺寸（避免一个网格中只有几个点的情况）
- 如果数据量有了很大的变化以后，可能需要重新计算并调整网格的尺寸

对于网格的尺寸，可以通过数据的范围平均值进行一个估算，一般可以取该值的 3~4 倍，比如对于刚才的那个面层，通过这样的命令可以进行一个大概的估算：

```
SQL> select 4*( avg(t.shape.maxx-t.shape.minx) + avg(t.shape.maxy-t.shape.maxy) )/2  from testgrid t;

4*(AVG(T.SHAPE.MAXX-T.SHAPE.MINX)+AVG(T.SHAPE.MAXY-T.SHAPE.MAXY))/2
-----
160.832765
```



当然，更简单的方法就是通过 ArcCatalog 来计算，如图 13。

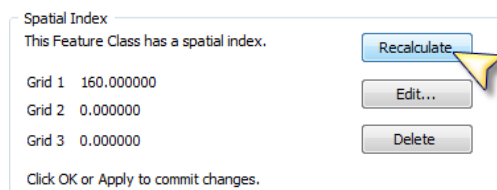


图 13 通过 ArcCatalog 来重新计算适合的网格尺寸

## • 空间索引的性能

### 1. 空间索引的用处

对于空间数据库进行空间查询的操作一般都有两个阶段，对于 Oracle Spatial 而言，一个空间查询分为 2 个步骤，一个称为主过滤 ( Primary Filter )，另外一个称为次过滤 ( Secondary Filter )。主过滤通过矩形的 MBR 相交从海量的数据中首先过滤出可能符合空间查询的一小部分数据，然后再用次过滤中具体的空间关系算法来判断这个小的结果集中到底哪些是满足空间关系的。如图 14，在主过滤的过程中，空间索引会被使用到。

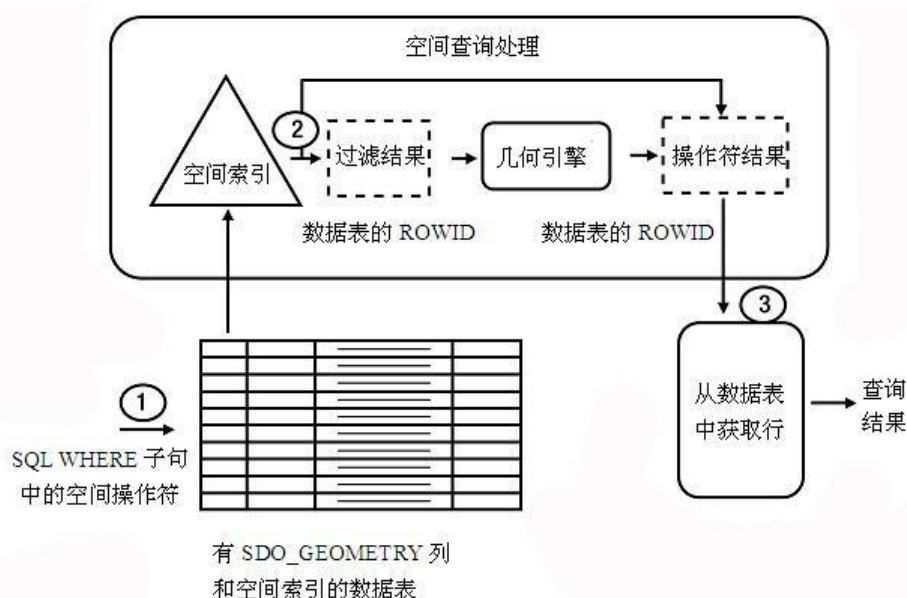


图 14 Oracle Spatial 中空间查询的处理流程

对于 ArcSDE 来说，原理上也是类似，ArcSDE 中有一个函数叫 ST\_ENVINTERSECTS，这就是通过四角坐标来从海量数据中过滤一个矩形范围内的一小部分数据。对于其它一般的空间操作，比如 ST\_WITHIN 等，只要你的数据建立了空间索引，这个操作也会先通过空间索引找到一个小数据集（相当于 ST\_ENVINTERSECTS 操作，或者说 Oracle Spatial 中的主过滤），然后再通过具体的“Within”的空间关系算法再查得准确的结果（相当于 Oracle Spatial 中的次过滤）。当然，如果数据没有空间索引，那么上面的操作就没有空间索引可用，不过 ArcSDE 也能查到结果，这比 Oracle Spatial 不建空间索引就不能作空间查询要好一些。

## 2. 主过滤/ST\_ENVINTERSECTS 的比较

因此，从上一小节可以知道，空间索引的效率可以通过 Oracle Spatial 中的“主过滤”和 ArcSDE 的 ST\_ENVINTERSECTS 的比较得到一个参考的结果。

以下是对上面使用过的约 1 亿条数据的线数据和面数据分别进行 SQL 语句查询的性能。测试代码如下：

```
public class PerformanceTest {

    public static void main(String[] args) {
        PerformanceTest test = new PerformanceTest();
        test.execute();
    }

    private static final String url = "jdbc:oracle:thin:@192.168.200.224:1521:test";
    private static final String user = "sde";
    private static final String password = "sde";

    public void execute() {
        Connection conn = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;
```

```

try {
    Class.forName("oracle.jdbc.driver.OracleDriver");
    conn = DriverManager.getConnection(url, user, password);
    conn.setAutoCommit(false);

    String sql = "select shape from sdo_test.streets where
sdo_filter(shape,SDO_GEOMETRY(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,1),SDO_ORDINATE_ARRAY(-72.
527,41.869,-72.493,41.869,-72.493,41.891,-72.527,41.891,-72.527,41.869)))='TRUE'";

    long start = System.currentTimeMillis();

    stmt = conn.prepareStatement(sql);
    stmt.setFetchSize(1000);

    int count = 0;
    rs = stmt.executeQuery();
    while (rs.next()) {
        Object shape = rs.getObject(1);//读取几何对象值
        count ++;
    }

    long end = System.currentTimeMillis();

    System.out.println("查询结果 : " + count);
    System.out.println("查询时间 : " + (end - start) + "ms");
} catch (Exception ex) {
    ex.printStackTrace();
} finally{
    try {
        rs.close();
    } catch (SQLException e) {
    }
    try {
        stmt.close();
    } catch (SQLException e) {
    }
    try {
        conn.close();
    } catch (SQLException e) {
    }
}
}
}

```

线数据测试结果，耗时单位为秒，每种类型分别对应前后两个结果分别代表  
在数据库中没有数据块缓存和缓存后的结果：

查询范围（真实结果数）	ST_GEOMETRY	SDO_GEOMETRY
-------------	-------------	--------------

范围约 10 个要素(8)	0.357	0.156	0.218	0.203
范围约 100 个要素(114)	0.734	0.203	0.811	0.203
范围约 1000 个要素(1341)	0.920	0.250	0.951	0.234
范围约 10000 个要素(10414)	2.386	0.562	3.104	0.400

面数据测试结果，耗时单位为秒，每种类型分别对应前后两个结果分别代表

在数据库中无数据块缓存和缓存后的结果：

查询范围（真实结果数）	ST_GEOMETRY		SDO_GEOMETRY	
范围约 10 个要素(16)	0.530	0.160	0.780	0.203
范围约 100 个要素(133)	2.636	0.343	1.841	0.220
范围约 1000 个要素(1104)	8.034	0.500	8.517	0.300
范围约 10000 个要素(10494)	29.952	1.310	43.010	0.950

### III. 空间关系运算

以上两个章节是必不可少的基础,涉及到空间数据在数据库中的存储和通过索引加速空间数据的获取。从这句话也可以看出,后续的章节并不是必须的,也就是说某些基于空间数据的应用可能并不需要诸如空间关系判断、几何对象处理等功能。这并不是说这些功能就用不着了,而是这些功能并不一定需要在数据库端执行。比如基于 ArcSDE,这些空间算法和功能在 ArcGIS 的产品线中无处不在,很多时候都不会把这些功能放到数据库上去。

#### • Oracle Spatial

在 Oracle Spatial 中,主要的空间关系操作在下表中列出:

空间操作	描述
SDO_FILTER	主过滤,判断哪些几何对象可能和给定的几何对象相交
SDO_JOIN	基于一定空间关系的表连接
SDO_NN	查询离某个几何对象最近的几何对象
SDO_NN_DISTANCE	查询离某个几何对象最近的几何对象与当前对象的距离
SDO_RELATE	判断两个几何对象是否满足某种空间关系
SDO_WITHIN_DISTANCE	判断两个几何对象间距离是否小于某给定值

其中 SDO\_RELATE 操作比较特殊,它通过 mask 参数来判断几何对象间是否满足某种空间关系,但是事实上如果都用参数来指定空间关系的话使用上就不太方便,因此,Oracle Spatial 还提供了基于 SDO\_RELATE 不同参数代表的不同空间关系的一些操作,如下表:

衍生自 SDO_RELATE 的空间操作	描述
----------------------	----

SDO_ANYINTERACT	任意部分有相交
SDO_CONTAINS	<p>A CONTAINS B B INSIDE A</p> <p>A COVERS B B COVEREDBY A</p> <p>A TOUCH B B TOUCH A</p> <p>A OVERLAPBDYINTERSECT B B OVERLAPBDYINTERSECT A</p> <p>A OVERLAPBDYDISJOINT B B OVERLAPBDYDISJOINT A</p> <p>A EQUAL B B EQUAL A (2 polygons with identical coordinates)</p> <p>A DISJOINT B B DISJOINT A</p> <p>B ON A A COVERS B</p>
SDO_COVEREDBY	
SDO_COVERS	
SDO_EQUAL	
SDO_INSIDE	
SDO_ON	
SDO_OVERLAPBDYDISJOINT	
SDO_OVERLAPBDYINTERSECT	
SDO_OVERLAPS	
SDO_TOUCH	

下面我们看一些空间关系操作的大概用法 ,比如我用一个三角形去查询所有满足 “INSIDE” 关系的几何对象 , 我们可以通过以下两种等价的方式操作 :

```
SQL> select continent from continent where sdo_inside(geom,
SDO_GEOMETRY(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,1),SDO_ORDINATE_ARRAY(0,0,180,-90,180
,90,0,0)))='TRUE';

CONTINENT
-----
Australia

SQL> select continent from continent where sdo_relate(geom,
SDO_GEOMETRY(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,1),SDO_ORDINATE_ARRAY(0,0,180,-90,180
,90,0,0)), 'mask=INSIDE')='TRUE';

CONTINENT
-----
Australia
```

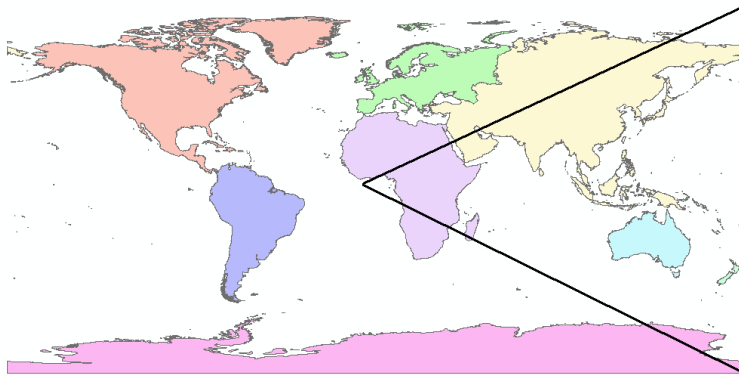


图 15 判断和图示三角形满足一定空间关系的几何对象

虽然有一些比较方便的 SDO\_RELATE 的衍生操作，不过 SDO\_RELATE 支持更灵活的空间关系判断，比如以下的 SQL 语句可以查询和上面的三角形满足“OVERLAPBDYDISJOINT”关系、“INSIDE”关系及同时满足这两种关系的几何对象：

```
SQL> select continent from continent where sdo_relate(geom,
SDO_GEOMETRY(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,1),SDO_ORDINATE_ARRAY(0,0,180,-90,180
,90,0,0)), 'mask=OVERLAPBDYDISJOINT')='TRUE';

CONTINENT
-----
North America

SQL> select continent from continent where sdo_relate(geom,
SDO_GEOMETRY(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,1),SDO_ORDINATE_ARRAY(0,0,180,-90,180
,90,0,0)), 'mask=INSIDE')='TRUE';

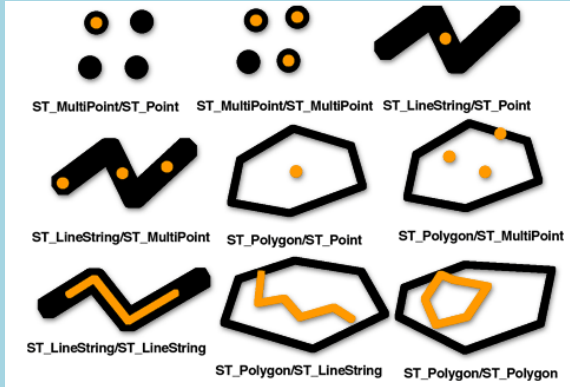
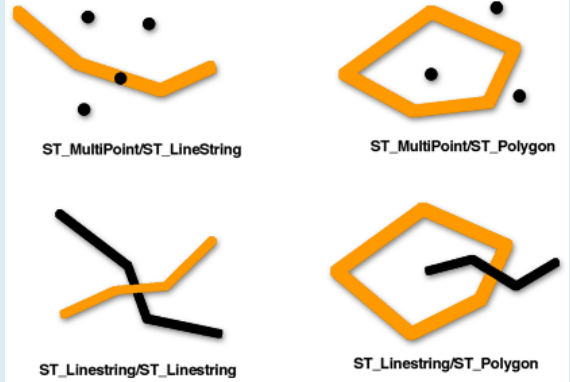
CONTINENT
-----
Australia

SQL> select continent from continent where sdo_relate(geom,
SDO_GEOMETRY(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,1),SDO_ORDINATE_ARRAY(0,0,180,-90,180
,90,0,0)), 'mask=INSIDE+OVERLAPBDYDISJOINT')='TRUE';

CONTINENT
-----
North America
Australia
```

- ArcSDE

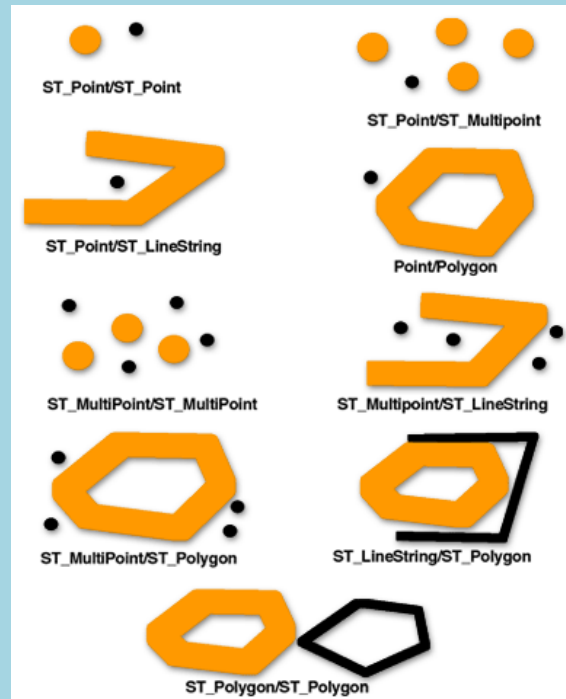
ArcSDE 中有以下的空间关系判断函数：

空间操作	描述
ST_EnvIntersects	矩形的边界满足 ST_Intersects 关系
ST_Intersects	<p>任意部分有相交,等价于判断空间关系的 DE-9IM<sup>1</sup>字符串表达是否是以下之一：</p> <p>T*****</p> <p>*T*****</p> <p>***T*****</p> <p>****T*****</p>
ST_Contains	 <p>T*****FF*</p>
ST_Crosses	 <p>T*T*****</p> <p>0*****</p>

<sup>1</sup> <http://docs.codehaus.org/display/GEOTDOC/Point+Set+Theory+and+the+DE-9IM+Matrix>

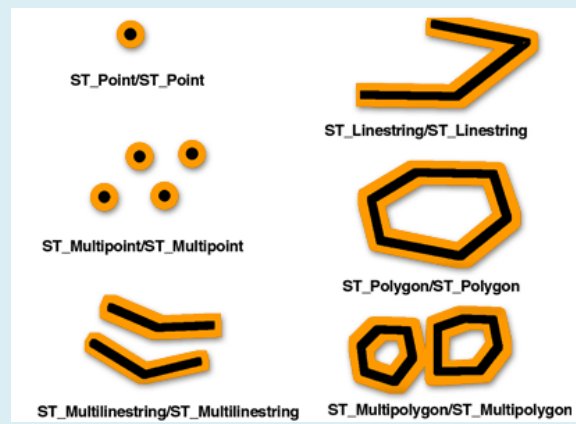


ST\_Disjoint



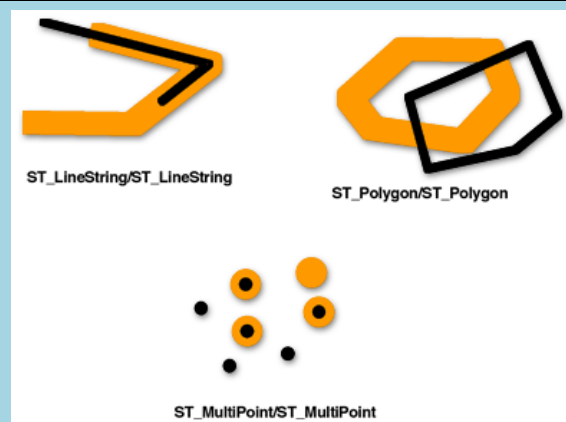
FF\*FF\*\*\*\*

ST\_Equals

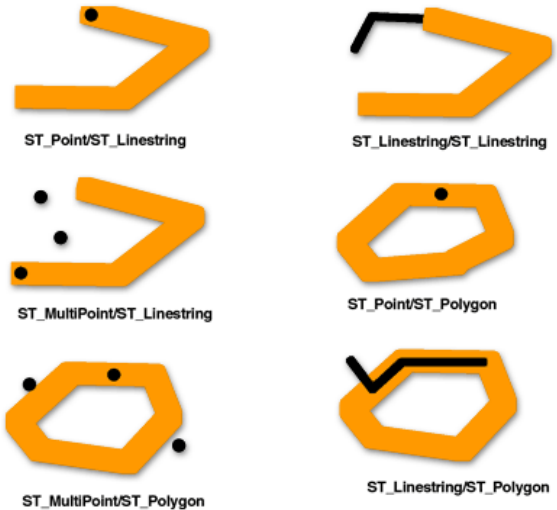
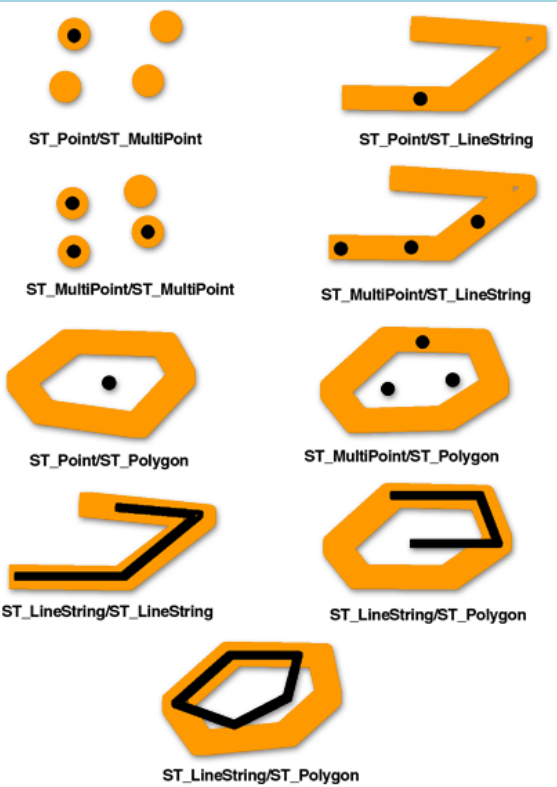


T\*F\*\*FFF\*

ST\_Overlaps



T\*T\*\*\*T\*\*

ST_Touches	 <p>FT*****</p> <p>F**T****</p> <p>F***T****</p>
ST_Within	 <p>T*F**F***</p>
ST_Relate	判断是否满足 DE-9IM 字符串表达关系

类比上一节中的查询条件，以下的 SQL 在 ArcSDE 中实现了相同的操作：

```
SQL> select continent from continent where st_within(shape,st_geometry('POLYGON((0 0,180 -90,180 90,0 0))',2))=1;
```

```
CONTINENT
```

```

-----
Australia

SQL> select continent from continent where st_relate(shape,st_geometry('POLYGON((0 0,180 -90,180 90,0
0))',2), 'T**F**F**')=1;

CONTINENT
-----
Australia

```

## • 空间关系运算的性能

空间关系运算以世界洲界数据进行空间运算测试 ,因为这个数据的边界相对复杂，我们可以大概看一下这个数据的节点个数：

```

SQL> select t.continent,t.shape.numpts from continent t;

CONTINENT          SHAPE.NUMPTS
-----
Asia                42423
North America       65453
Europe              27437
Africa              7178
South America       11564
Oceania             4887
Australia           4018
Antarctica          20205

```

这样数量级（4 千 ~ 6.5 万）节点的数据应该可以更好地反映空间关系运算的性能。

### 1.相交

Oracle Spatial :

```

SQL> select continent from continent where sdo_relate(geom,
SDO_GEOMETRY(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,1),SDO_ORDINATE_ARRAY(0,0,180,-90,180
,90,0,0)), 'mask=ANYINTERACT')='TRUE' ;

CONTINENT
-----
Asia

```

```
Oceania
Antarctica
North America
Africa
Australia
```

已选择 6 行。

已用时间: 00: 00: 04.11

ArcSDE :

```
SQL> select continent from continent where st_intersects(shape,st_geometry('POLYGON((0 0,180 -90,180 90,0
0))',2))=1;
```

```
CONTINENT
```

```
-----
Asia
North America
Africa
Oceania
Australia
Antarctica
```

已选择 6 行。

已用时间: 00: 00: 00.24

## IV. 几何处理

### • 缓冲分析

Oracle Spatial 中缓冲分析使用 SDO\_GEOM.SDO\_BUFFER 函数实现，下面是 SDO\_GEOM 包中 SDO\_BUFFER 函数的原型：

```
function sdo_buffer(geom IN MDSYS.SDO_GEOMETRY,  
                    dist IN NUMBER,  
                    tol IN NUMBER,  
                    params IN VARCHAR2)  
return MDSYS.SDO_GEOMETRY DETERMINISTIC;
```

这里的 dist 为缓冲距离；tol 为容差值；params 代表 2 个可选参数，分别为 unit=<string>和 arc\_tolerance=<number>，这些可选参数可以帮助用户在地理坐标系的数据上直接进行投影距离的缓冲。

下面的操作是对一个经纬度点数据进行一个 1000 千米的缓冲后，再输出经纬度几何对象：

```
SQL> select sdo_geom.sdo_buffer(SDO_GEOMETRY(2001,4326,SDO_POINT_TYPE(0,0,  
NULL),NULL,NULL),1000,1,'arc_tolerance=0.1 unit=km') from dual;  
  
SDO_GEOM.SDO_BUFFER(SDO_GEOMETRY(2001,4326,SDO_POINT_TYPE(0,0,NULL),NULL,NULL),1000,1,'ARC_T  
OLERANCE=0.1UNIT=KM')(SDO_GT  
-----  
SDO_GEOMETRY(2003, 4326, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARRAY(6.40486849,  
6.40776286, 6.22416673, 6.58569883, 6.03847881, 6.758491, 5.84794461, 6.92599964, 5.65270846,  
7.08808893, 5.45291911, 7.24462711, 5.24872964, 7.39548658, 5.04029736, 7.54054407, 4.82778372,  
7.67968076, 4.61135423, 7.81278238, 4.39117828, 7.93973937, 4.16742907, 8.06044696, 3.94028348,  
8.17480534, 3.70992187, 8.28271972, 3.476528, 8.3841005, 3.24028879, 8.47886329, 3.00139424, 8.56692911,  
2.76003717, 8.64822438, 2.5164131, 8.72268111, 2.27072, 8.79023691, 2.02315813, 8.85083509, 1.77392981,  
8.90442474, 1.52323924, 8.95096079, 1.27129222, 8.99040405, 1.01829598, 9.02272129, .764458942,  
9.04788525, .509990466, 9.0658747, .255100632, 9.07667445, 3.7723E-15, 9.08027538, -.25510063,  
9.07667445, -.50999047, 9.0658747, -.76445894, 9.04788525, -1.018296, 9.02272129, -1.2712922, 8.99040405,  
-1.5232392, 8.95096079, -1.7739298, 8.90442474, -2.0231581, 8.85083509, -2.27072, 8.79023691, -2.5164131,  
8.72268111, -2.7600372, 8.64822438, -3.0013942, 8.56692911, -3.2402888, 8.47886329, -3.476528, 8.3841005,  
-3.7099219, 8.28271972, -3.9402835, 8.17480534, -4.1674291, 8.06044696, -4.3911783, 7.93973937,  
-4.6113542, 7.81278238, -4.8277837, 7.67968076, -5.0402974, 7.54054407, -5.2487296, 7.39548658,  
-5.4529191, 7.24462711, -5.6527085, 7.08808893, -5.8479446, 6.92599964, -6.0384788, 6.758491, -6.2241667,
```

6.58569883, -6.4048685, 6.40776286, -6.5804488, 6.22482658, -6.7507769, 6.03703716, -6.9157268, 5.84454523, -7.0751771, 5.64750481, -7.2290111, 5.44607313, -7.377117, 5.24041051, -7.5193876, 5.03068022, -7.6557206, 4.81704835, -7.7860186, 4.59968364, -7.9101888, 4.37875739, -8.0281433, 4.15444327, -8.1397992, 3.92691724, -8.2450781, 3.69635739, -8.3439067, 3.4629438, -8.4362164, 3.22685841, -8.5219434, 2.9882849, -8.6010286, 2.74740856, -8.6734179, 2.50441616, -8.7390617, 2.2594958, -8.7979155, 2.01283683, -8.8499391, 1.76462967, -8.8950973, 1.51506573, -8.9333597, 1.26433727, -8.9647005, 1.01263726, -8.9890985, .760159289, -9.0065374, .507097431, -9.0170054, .253646111, -9.0204955, 3.7836E-15, -9.0170054, -.25364611, -9.0065374, -.50709743, -8.9890985, -.76015929, -8.9647005, -1.0126373, -8.9333597, -1.2643373, -8.8950973, -1.5150657, -8.8499391, -1.7646297, -8.7979155, -2.0128368, -8.7390617, -2.2594958, -8.6734179, -2.5044162, -8.6010286, -2.7474086, -8.5219434, -2.9882849, -8.4362164, -3.2268584, -8.3439067, -3.4629438, -8.2450781, -3.6963574, -8.1397992, -3.9269172, -8.0281433, -4.1544433, -7.9101888, -4.3787574, -7.7860186, -4.5996836, -7.6557206, -4.8170484, -7.5193876, -5.0306802, -7.377117, -5.2404105, -7.2290111, -5.4460731, -7.0751771, -5.6475048, -6.9157268, -5.8445452, -6.7507769, -6.0370372, -6.5804488, -6.2248266, -6.4048685, -6.4077629, -6.2241667, -6.5856988, -6.0384788, -6.758491, -5.8479446, -6.9259996, -5.6527085, -7.0880889, -5.4529191, -7.2446271, -5.2487296, -7.3954866, -5.0402974, -7.5405441, -4.8277837, -7.6796808, -4.6113542, -7.8127824, -4.3911783, -7.9397394, -4.1674291, -8.060447, -3.9402835, -8.1748053, -3.7099219, -8.2827197, -3.476528, -8.3841005, -3.2402888, -8.4788633, -3.0013942, -8.5669291, -2.7600372, -8.6482244, -2.5164131, -8.7226811, -2.27072, -8.7902369, -2.0231581, -8.8508351, -1.7739298, -8.9044247, -1.5232392, -8.9509608, -1.2712922, -8.9904041, -1.018296, -9.0227213, -.76445894, -9.0478853, -5.0999047, -9.0658747, -.25510063, -9.0766745, -5.361E-15, -9.0802754, .255100632, -9.0766745, .509990466, -9.0658747, .764458942, -9.0478853, 1.01829598, -9.0227213, 1.27129222, -8.9904041, 1.52323924, -8.9509608, 1.77392981, -8.9044247, 2.02315813, -8.8508351, 2.27072, -8.7902369, 2.5164131, -8.7226811, 2.76003717, -8.6482244, 3.00139424, -8.5669291, 3.24028879, -8.4788633, 3.476528, -8.3841005, 3.70992187, -8.2827197, 3.94028348, -8.1748053, 4.16742907, -8.060447, 4.39117828, -7.9397394, 4.61135423, -7.8127824, 4.82778372, -7.6796808, 5.04029736, -7.5405441, 5.24872964, -7.3954866, 5.45291911, -7.2446271, 5.65270846, -7.0880889, 5.84794461, -6.9259996, 6.03847881, -6.758491, 6.22416673, -6.5856988, 6.40486849, -6.4077629, 6.5804488, -6.2248266, 6.75077693, -6.0370372, 6.91572682, -5.8445452, 7.0751771, -5.6475048, 7.22901112, -5.4460731, 7.377117, -5.2404105, 7.51938761, -5.0306802, 7.65572064, -4.8170484, 7.78601859, -4.5996836, 7.91018877, -4.3787574, 8.02814331, -4.1544433, 8.13979915, -3.9269172, 8.24507809, -3.6963574, 8.34390671, -3.4629438, 8.43621641, -3.2268584, 8.52194338, -2.9882849, 8.60102862, -2.7474086, 8.6734179, -2.5044162, 8.73906175, -2.2594958, 8.79791545, -2.0128368, 8.84993906, -1.7646297, 8.89509732, -1.5150657, 8.93335973, -1.2643373, 8.96470049, -1.0126373, 8.98909849, -.76015929, 9.00653735, -.50709743, 9.01700535, -.25364611, 9.02049547, -3.718E-15, 9.01700535, .253646111, 9.00653735, .507097431, 8.98909849, .760159289, 8.96470049, 1.01263726, 8.93335973, 1.26433727, 8.89509732, 1.51506573, 8.84993906, 1.76462967, 8.79791545, 2.01283683, 8.73906175, 2.2594958, 8.6734179, 2.50441616, 8.60102862, 2.74740856, 8.52194338, 2.9882849, 8.43621641, 3.22685841, 8.34390671, 3.4629438, 8.24507809, 3.69635739, 8.13979915, 3.92691724, 8.02814331, 4.15444327, 7.91018877, 4.37875739, 7.78601859, 4.59968364, 7.65572064, 4.81704835, 7.51938761, 5.03068022, 7.377117, 5.24041051, 7.22901112, 5.44607313, 7.0751771, 5.64750481, 6.91572682, 5.84454523, 6.75077693, 6.03703716, 6.5804488, 6.22482658, 6.40486849, 6.40776286))

这里缓冲的点坐标为(0,0) ,缓冲结果是一个由圆弧段构成的多边形 , X 方向最大值为 9.0802754、Y 方向最大值为 9.0204955——显然 , 这是一个比较扁的椭圆 , 而且在 X 方向最大值 ( 0 纬度上的 9.0802754 度 ) 就是 1000 千米对应的经度值 , 这是符合真实结果的。如果直接作为投影坐标进行缓冲的话 , 结果应该是一个圆形 :

```
SQL> select sdo_geom.sdo_buffer(SDO_GEOMETRY(2001,NULL,SDO_POINT_TYPE(0,0,
NULL),NULL,NULL),10,0.01) from dual;

SDO_GEOM.SDO_BUFFER(SDO_GEOMETRY(2001,NULL,SDO_POINT_TYPE(0,0,NULL),NULL,NULL),10,0.01)(SDO
_GTYPE, SDO_SRID, SDO_POINT(X
-----
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 2), SDO_ORDINATE_ARRAY(-8.882E-16,
-10, 10, -1.776E-15, -8.882E-16, 10, -10, -1.776E-15, -8.882E-16, -10))
```

ArcSDE 中缓冲分析使用 ST\_Buffer 函数实现 ,ST\_Buffer 的函数原型如下 :

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

也就是 ArcSDE 中只需要指定几何对象和缓冲距离即可。为什么 ArcSDE 不支持投影的缓冲呢？我们要时刻想到 ArcSDE 仅仅是 ArcGIS 的空间数据存储方案，复杂的缓冲等几何操作在 ArcGIS 产品线的其它产品中都有实现，而不像 Oracle Spatial 必须在数据库上完成这些功能。因此 ArcSDE 在数据库端实现的空间操作基本只包含 OGC 规范所包含的内容。

下面是通过 ArcSDE 进行缓冲的操作：

```
SQL> select sde.st_astext(sde.st_buffer(sde.st_geomfromtext('POINT(0 0)',0), 10)) from dual;

SDE.ST_ASTEXT(SDE.ST_BUFFER(SDE.ST_GEOMFROMTEXT('POINT(00)',0),10))
-----
POLYGON (( 10.00000000 0.00000000, 9.97858923 0.65403129, 9.91444861 1.30526192, 9.80785280
1.95090322, 9.65925826 2.58819045, 9.46930129 3.21439465, 9.23879533 3.82683432, 8.96872741
4.42288690, 8.66025404 5.00000000, 8.31469612 5.55570233, 7.93353340 6.08761429, 7.51839807
6.59345815, 7.07106781 7.07106781, 6.59345815 7.51839807, 6.08761429 7.93353340, 5.55570233
8.31469612, 5.00000000 8.66025404, 4.42288690 8.96872741, 3.82683432 9.23879533, 3.21439465
9.46930129, 2.58819045 9.65925826, 1.95090322 9.80785280, 1.30526192 9.91444861, 0.65403129
9.97858923, 0.00000000 10.00000000, -0.65403129 9.97858923, -1.30526192 9.91444861, -1.95090322
9.80785280, -2.58819045 9.65925826, -3.21439465 9.46930129, -3.82683432 9.23879533, -4.42288690
8.96872741, -5.00000000 8.66025404, -5.55570233 8.31469612, -6.08761429 7.93353340, -6.59345815
7.51839807, -7.07106781 7.07106781, -7.51839807 6.59345815, -7.93353340 6.08761429, -8.31469612
5.55570233, -8.66025404 5.00000000, -8.96872741 4.42288690, -9.23879533 3.82683432, -9.46930129
3.21439465, -9.65925826 2.58819045, -9.80785280 1.95090322, -9.91444861 1.30526192, -9.97858923
0.65403129, -10.00000000 0.00000000, -9.97858923 -0.65403129, -9.91444861 -1.30526192, -9.80785280
-1.95090322, -9.65925826 -2.58819045, -9.46930129 -3.21439465, -9.23879533 -3.82683432, -8.96872741
-4.42288690, -8.66025404 -5.00000000, -8.31469612 -5.55570233, -7.93353340 -6.08761429, -7.51839807
-6.59345815, -7.07106781 -7.07106781, -6.59345815 -7.51839807, -6.08761429 -7.93353340, -5.55570233
-8.31469612, -5.00000000 -8.66025404, -4.42288690 -8.96872741, -3.82683432 -9.23879533, -3.21439465
-9.46930129, -2.58819045 -9.65925826, -1.95090322 -9.80785280, -1.30526192 -9.91444861, -0.65403129
-9.97858923, 0.00000000 -10.00000000, 0.65403129 -9.97858923, 1.30526192 -9.91444861, 1.95090322
```

```
-9.80785280, 2.58819045 -9.65925826, 3.21439465 -9.46930129, 3.82683432 -9.23879533, 4.42288690
-8.96872741, 5.00000000 -8.66025404, 5.55570233 -8.31469612, 6.08761429 -7.93353340, 6.59345815
-7.51839807, 7.07106781 -7.07106781, 7.51839807 -6.59345815, 7.93353340 -6.08761429, 8.31469612
-5.55570233, 8.66025404 -5.00000000, 8.96872741 -4.42288690, 9.23879533 -3.82683432, 9.46930129
-3.21439465, 9.65925826 -2.58819045, 9.80785280 -1.95090322, 9.91444861 -1.30526192, 9.97858923
-0.65403129, 10.00000000 0.00000000))
```

## ● 距离量测

Oracle Spatial 使用 SDO\_GEOM 包的 SDO\_DISTANCE 函数实现：

```
SQL> select objectid,sdo_geom.sdo_distance(SDO_GEOMETRY(2001,NULL,SDO_POINT_TYPE(0,0,
NULL),NULL,NULL),shape,0.01) from sdo_cities where objectid<10;
```

OBJECTID	SDO_GEOM.SDO_DISTANCE(SDO_GEOMETRY(2001,NULL,SDO_POINT_TYPE(0,0,NULL),NULL,NULL),SHAPE,0.01)
1	177.441709
2	17.0281416
3	78.4360425
4	75.6184995
5	72.5343637
6	73.3851836
7	74.3064287
8	72.7180626
9	70.5420529

ArcSDE 使用 ST\_Distance 函数实现：

```
SQL> select objectid,sde.st_distance(sde.st_geomfromtext('POINT(0 0)',0),shape) from sde_cities where
objectid<10;
```

OBJECTID	SDE.ST_DISTANCE(SDE.ST_GEOMFROMTEXT('POINT(00)',0),SHAPE)
1	177.441709
2	17.0281416
3	78.4360425
4	75.6184995
5	72.5343637
6	73.3851836
7	74.3064287
8	72.7180626
9	70.5420529



- 面积、长度量测

Oracle Spatial 使用 SDO\_GEOM 包的 SDO\_AREA 和 SDO\_LENGTH 函数

实现：

```
SQL> select objectid,sdo_geom.sdo_area(shape,0.01) from sdo_continent;
```

```
OBJECTID SDO_GEOM.SDO_AREA(SHAPE,0.01)
```

```
-----
```

1	5432.08564
2	3707.41866
3	1444.71727
4	2559.07307
5	1539.3129
6	42.5654563
7	695.539935
8	6034.46185

```
SQL> select objectid,sdo_geom.sdo_length(shape,0.01) from sdo_continent;
```

```
OBJECTID SDO_GEOM.SDO_LENGTH(SHAPE,0.01)
```

```
-----
```

1	2331.82054
2	3954.89183
3	1598.02862
4	426.208533
5	622.552604
6	221.581977
7	252.165153
8	1587.22795

ArcSDE 使用 ST\_Area 和 ST\_Length 函数实现：

```
SQL> select objectid,sde.st_area(shape) from sde_continent;
```

```
OBJECTID SDE.ST_AREA(SHAPE)
```

```
-----
```

1	5432.08564
2	3707.41866
3	1444.71727
4	2559.07307
5	1539.3129
6	42.5654563
7	695.539935
8	6034.46185

```
SQL> select objectid,sde.st_length(shape) from sde_continent;
```

OBJECTID	SDE.ST_LENGTH(SHAPE)
1	2331.82054
2	3954.89183
3	1598.02862
4	426.208533
5	622.552604
6	221.581977
7	252.165153
8	1587.22795

## • 凸包运算

Oracle Spatial 使用 SDO\_GEOM 包的 SDO\_CONVEXHULL 函数实现：

```
SQL> select sdo_geom.sdo_convexhull(shape,0.01) from sdo_continent where objectid=1;
```

```
SDO_GEOM.SDO_CONVEXHULL(SHAPE,0.01)(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
-----
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARRAY(122.848862,
-10.929653, 150.209686, -10.700556, 150.426224, -10.689167, 150.880524, -10.652779, 150.8936, -10.648749,
151.229126, -10.201111, 179.604401, 62.7002678, 180, 65.0689087, 180, 65.3986053, 180, 65.9801025, 180,
66.9801025, 180, 67.9801025, 180, 68.9801025, 180, 70.9972076, 180, 71.5358582, 156.722748, 77.1322021,
156.700806, 77.1369324, 156.67746, 77.1405487, 156.653046, 77.1430511, 156.603027, 77.1469421,
95.6994171, 81.2902679, 95.6530457, 81.2905426, 95.5274811, 81.2894287, 90.4994202,
81.2263794, 90.1691437, 81.2188721, 79.6588745, 80.9788666, -179.6286, 71.5771942, -179.90077, 71.54879,
-180, 71.5358429, -180, 70.9972076, -180, 68.9801025, -180, 68.0689087, -180, 67.0689087, -180, 66.0689087,
-180, 65.0689087, 72.4325409, -7.4347382, 72.4381714, -7.4362497, 105.701401, -10.51097, 122.848862,
-10.929653))
```

ArcSDE 使用 ST\_ConvexHull 函数实现：

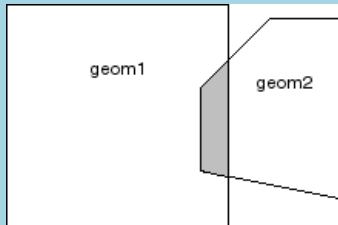
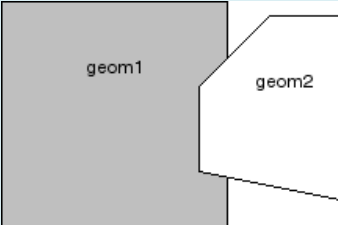
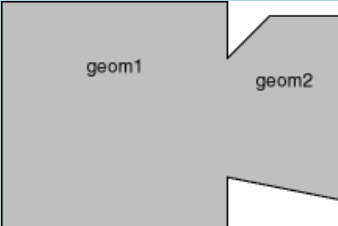
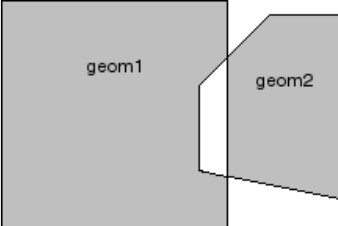
```
SQL> select sde.st_astext(sde.st_convexhull(shape)) from sde_continent where objectid=1;
```

```
SDO.ST_ASTEXT(SDE.ST_CONVEXHULL(SHAPE))
-----
POLYGON ((-180.00000000 71.53584290,-180.00000000 70.99720764,-180.00000000 68.98010254,
-180.00000000 68.06890869,-180.00000000 67.06890869,-180.00000000 66.06890869,-180.00000000
65.06890869,72.43254089 -7.43473816, 72.43817139 -7.43624973, 105.70140076 -10.51097012,
122.84886169 -10.92965317, 150.20968628 -10.70055580, 150.42622376 -10.68916702, 150.88052368
```

```
-10.65277862, 150.89360046 -10.64874935, 151.22912598 -10.20111084, 179.60440064 62.70026779,
180.00000000 65.06890869, 180.00000000 65.39860535, 180.00000000 65.98010254, 180.00000000
66.98010254, 180.00000000 67.98010254, 180.00000000 68.98010254, 180.00000000 70.99720764,
180.00000000 71.53585815, 156.72274780 77.13220215, 156.70080566 77.13693237, 156.67745972
77.14054871, 156.65304565 77.14305115, 156.60302734 77.14694214, 95.69941711 81.29026794,
95.65304565 81.29054260, 95.52748108 81.28942871, 90.49942017 81.22637939, 90.16914368 81.21887207,
79.65887451 80.97886658, -179.62860107 71.57719421, -179.90077210 71.54878998, -180.00000000
71.53584290))
```

## • 几何对象组合

几何对象的组合大体上有以下几种：

几何对象组合	描述（灰色部分为结果）
Intersection	
Difference	
Union	
Xor/SymmetricDiff	

## Oracle Spatial 中的操作：

```
SQL> select
sdo_geom.sdo_intersection(SDO_GEOMETRY(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,1),SDO_ORDINATE_ARRAY(0,0,2,0,2,0,2,0,0)),SDO_GEOMETRY(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,1),SDO_ORDINATE_ARRAY(1,1,3,1,3,3,1,3,1,1)),1) from dual;

SDO_GEOM.SDO_INTERSECTION(SDO_GEOMETRY(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,1),SDO_ORDINATE_ARRAY(0,0,2,0,2,0,2,0,0)
-----
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARRAY(1, 2, 1, 1, 2, 1, 2, 1, 2))

SQL> select
sdo_geom.sdo_difference(SDO_GEOMETRY(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,1),SDO_ORDINATE_ARRAY(0,0,2,0,2,0,2,0,0)),SDO_GEOMETRY(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,1),SDO_ORDINATE_ARRAY(1,1,3,1,3,3,1,3,1,1)),1) from dual;

SDO_GEOM.SDO_DIFFERENCE(SDO_GEOMETRY(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,1),SDO_ORDINATE_ARRAY(0,0,2,0,2,0,2,0,0)
-----
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARRAY(0, 2, 0, 0, 2, 0, 2, 1, 1, 1, 2, 0, 2))

SQL> select
sdo_geom.sdo_union(SDO_GEOMETRY(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,1),SDO_ORDINATE_ARRAY(0,0,2,0,2,0,2,0,0)),SDO_GEOMETRY(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,1),SDO_ORDINATE_ARRAY(1,1,3,1,3,3,1,3,1,1)),1) from dual;

SDO_GEOM.SDO_UNION(SDO_GEOMETRY(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,1),SDO_ORDINATE_ARRAY(0,0,2,0,2,0,2,0,0)),SD
-----
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARRAY(2, 1, 3, 1, 3, 3, 1, 3, 1, 2, 0, 2, 0, 0, 2, 0, 2, 1))

SQL> select
sdo_geom.sdo_xor(SDO_GEOMETRY(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,1),SDO_ORDINATE_ARRAY(0,0,2,0,2,0,2,0,0)),SDO_GEOMETRY(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,1),SDO_ORDINATE_ARRAY(1,1,3,1,3,3,1,3,1,1)),1) from dual;

SDO_GEOM.SDO_XOR(SDO_GEOMETRY(2003,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1003,1),SDO_ORDINATE_ARRAY(0,0,2,0,2,0,2,0,0)),SDO_
-----
SDO_GEOMETRY(2007, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1, 15, 1003, 1),
SDO_ORDINATE_ARRAY(3, 3, 1, 3, 1, 2, 2, 2, 1, 3, 1, 3, 3, 0, 2, 0, 0, 2, 0, 2, 1, 1, 1, 2, 0, 2))
```

## ArcSDE 中的操作：

```
SQL> select sde.st_astext(sde.st_intersection(sde.st_geometry('POLYGON((0 0,2 0,2 2,0 2,0 0))',0),sde.st_geometry('POLYGON((1 1,3 1,3 3,1 3,1 1))',0))) from dual;
```

```
SDE.ST_ASTEXT(SDE.ST_INTERSECTION(SDE.ST_GEOMETRY('POLYGON((00,20,22,02,00))',0)
-----
POLYGON (( 1.00000000 1.00000000, 2.00000000 1.00000000, 2.00000000 2.00000000
0, 1.00000000 2.00000000, 1.00000000 1.00000000))
```

```
SQL> select sde.st_astext(sde.st_difference(sde.st_geometry('POLYGON((0 0,2 0,2 2,0 2,0 0))',0),sde.st_geometry('POLYGON((1 1,3 1,3 3,1 3,1 1))',0))) from dual;
```

```
SDE.ST_ASTEXT(SDE.ST_DIFFERENCE(SDE.ST_GEOMETRY('POLYGON((00,20,22,02,00))',0),S
-----
POLYGON (( 0.00000000 0.00000000, 2.00000000 0.00000000, 2.00000000 1.00000000, 1.00000000
1.00000000, 1.00000000 2.00000000, 0.00000000 2.00000000, 0.00000000 0.00000000))
```

```
SQL> select sde.st_astext(sde.st_union(sde.st_geometry('POLYGON((0 0,2 0,2 2,0 2,0 0))',0),sde.st_geometry('POLYGON((1 1,3 1,3 3,1 3,1 1))',0))) from dual;
```

```
SDE.ST_ASTEXT(SDE.ST_UNION(SDE.ST_GEOMETRY('POLYGON((00,20,22,02,00))',0),SDE.ST
-----
POLYGON (( 0.00000000 0.00000000, 2.00000000 0.00000000, 2.00000000 1.00000000, 3.00000000
1.00000000, 3.00000000 3.00000000, 1.00000000 3.00000000, 1.00000000 2.00000000, 0.00000000
2.00000000, 0.00000000 0.00000000))
```

```
SQL> select sde.st_astext(sde.st_symmetricdiff(sde.st_geometry('POLYGON((0 0,2 0,2 2,0 2,0 0))',0),sde.st_geometry('POLYGON((1 1,3 1,3 3,1 3,1 1))',0))) from dual;
```

```
SDE.ST_ASTEXT(SDE.ST_SYMMETRICDIFF(SDE.ST_GEOMETRY('POLYGON((00,20,22,02,00))',0)
-----
MULTIPOLYGON ((( 1.00000000 2.00000000, 2.00000000 2.00000000, 2.00000000 1.00000000, 3.00000000
1.00000000, 3.00000000 3.00000000, 1.00000000 3.00000000, 1.00000000 2.00000000)),(( 0.00000000
0.00000000, 2.00000000 0.00000000, 2.00000000 1.00000000, 1.00000000 1.00000000,
1.00000000 2.00000000, 0.00000000 2.00000000, 0.00000000 0.00000000)))
```

## • 几何对象聚合

Oracle Spatial 有 SDO\_AGGR\_CONVEXHULL 和 SDO\_AGGR\_UNION 函数实现聚合：

```
SQL> select sdo_aggr_convexhull(SDOAGGRTYPE(shape, 0.1)) from sdo_cities where objectid<10;
```

```

SDO_AGGR_CONVEXHULL(SDOAGGRTYPE(SHAPE,0.1))(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z),
SDO_ELEM_INFO, SDO_ORDINATES)
-----
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARRAY(-3.0099955,
16.7599981, 55.1070006, 51.7820041, 56.0960016, 54.8219971, 47.2500006, 56.1500011, -165.27, 64.5862861,
-3.0099955, 16.7599981))

```

```

SQL> select sdo_aggr_union(SDOAGGRTYPE(shape, 0.1)) from sdo_cities where objectid<10;

SDO_AGGR_UNION(SDOAGGRTYPE(SHAPE,0.1))(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z),
SDO_ELEM_INFO, SDO_ORDINATES)
-----
SDO_GEOMETRY(2005, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 9), SDO_ORDINATE_ARRAY(45.1670046,
54.1860031, 48.3620046, 54.3050011, 49.1454636, 55.7330131, 47.2500006, 56.1500011, 51.3559986,
51.2229961, 55.1070006, 51.7820041, 56.0960016, 54.8219971, -3.0099955, 16.7599981, -165.27, 64.5862861))

```

ArcSDE 有 ST\_Aggr\_ConvexHull、ST\_Aggr\_Union、ST\_Aggr\_Intersection 函数实现聚合，不过经过我的测试，ST\_Aggr\_ConvexHull 函数有点问题，对点进行凸包聚合运算结果不正确，这个函数不应被使用，如果有这样的需求可以通过 ST\_ConvexHull(ST\_Aggr\_Union(g st\_geometry))这样的方法实现：

```

SQL> select sde.st_astext(sde.st_convexhull(sde.st_aggr_union(shape))) from sde_cities where objectid<10;

SDE.ST_ASTEXT(SDE.ST_CONVEXHULL(SDE.ST_AGGR_UNION(SHAPE)))
-----
POLYGON (( -165.26999663 64.58628611, -3.00999547 16.75999807, 55.10700058 51.78200410,
56.09600158 54.82199710, 47.25000057 56.15000110, -165.26999663 64.58628611))

```

```

SQL> select sde.st_astext(sde.st_aggr_union(shape)) from sde_cities where objectid<10;

SDE.ST_ASTEXT(SDE.ST_AGGR_UNION(SHAPE))
-----
MULTIPOINT ( -165.26999663 64.58628611, -3.00999547 16.75999807, 45.1670045754.18600310,
47.25000057 56.15000110, 48.36200457 54.30500110, 49.14546357 55.73301310, 51.35599858 51.22299610,
55.10700058 51.78200410, 56.09600158 54.82199710)

```

从结果上可以看到，对一个点数据的部分点进行 Aggr\_Union 运算的结果会是一个多点，而 Aggr\_ConvexHull 运算的结果则是一个多边形。Aggr\_ConvexHull 相当于先执行 Aggr\_Union 运算后再对合并后的几何对象进行 ConvexHull 运算。

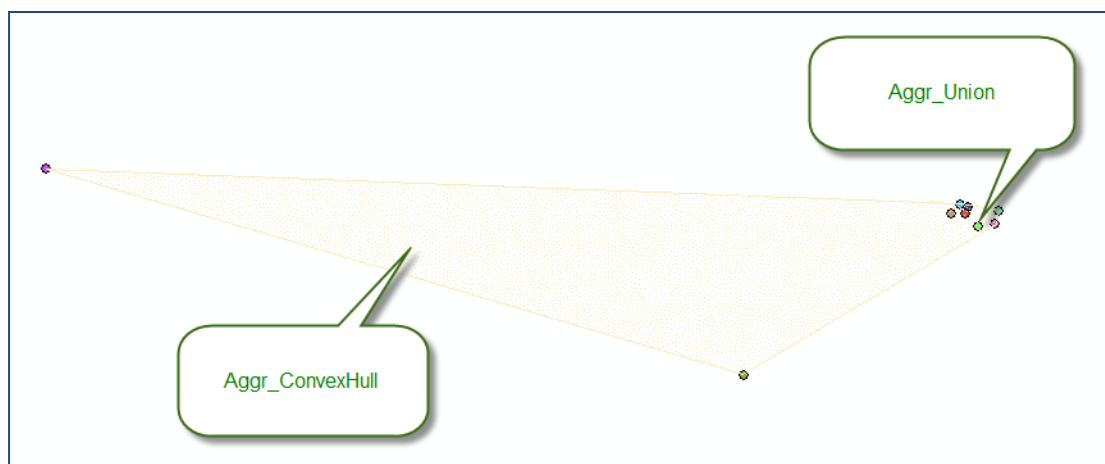


图 16 聚合（Aggr\_ConvexHull 和 Aggr\_Union）运算结果

## V. 线性参考

- Oracle Spatial

### 1. 创建线性参考的空间对象

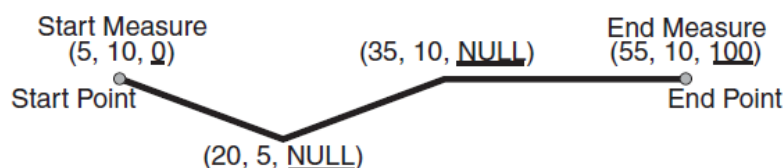


图 17 一个线性参考空间对象的例子

对于如图 17 的例子，Oracle Spatial 中需要通过如下的 SQL 语句进行创建：

```
SQL> select SDO_GEOMETRY(3302, NULL,  
NULL,SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(5,10,0, 20,5,NULL, 35,10,NULL, 55,10,100))  
shape from dual;  
  
SHAPE(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)  
-----  
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(5, 10, 0, 20, 5,  
NULL, 35, 10, NULL, 55, 10, 100))
```

同时，如果要在 Oracle Spatial 中可以使用这个空间数据，除了在相应的空间表中插入这些记录之外，还需要在元数据表中插入相关记录，插入元数据信息示例如下：

```
INSERT INTO user_sdo_geom_metadata(TABLE_NAME,COLUMN_NAME,DIMINFO,SRID)  
VALUES('SDO_TEST','SHAPE',  
SDO_DIM_ARRAY (  
SDO_DIM_ELEMENT('X', 0, 20, 0.005),  
SDO_DIM_ELEMENT('Y', 0, 20, 0.005),  
SDO_DIM_ELEMENT('M', 0, 100, 0.005)),  
NULL);
```

正常情况下，在 USER\_SDO\_GEOM\_METADATA 表中可以查到相关的信息



(关于 USER\_SDO\_GEOM\_METADATA 表的作用及定义可以回顾《II. 1 索引类型 SPATIAL\_INDEX》):

```
SQL> select * from user_sdo_geom_metadata where table_name='SDO_ROUTES';

TABLE_NAME    COLUMN_NAME
-----
DIMINFO(SDO_DIMNAME, SDO_LB, SDO_UB, SDO_TOLERANCE)  SRID
-----

SDO_ROUTES    SHAPE
SDO_DIM_ARRAY(SDO_DIM_ELEMENT(NULL, 2389694, 2569011.02, .0000005), SDO_DIM_ELEMENT(NULL,
581586.917, 760268.123, .0000005), SDO_DIM_ELEMENT('M', -1000, 267435.456, .0000005))
```

## 2. 根据线性参考定位点

对于线性参考的空间对象，就可以根据线性参考来定位对象上的点，比如上述例子中图 17 的对象，如果我们要获得在线方向上位于中点的点坐标，我们可以这样做：

```
SQL> select SDO_LRS.LOCATE_PT(SDO_GEOMETRY(3302, NULL,
NULL,SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(5,10,0, 20,5,NULL, 35,10,NULL, 55,10,100)), 50)
from dual;

SDO_LRS.LOCATE_PT(SDO_GEOMETRY(3302,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(5,10,0,20,5,NULL,35,10,NULL,
-----
SDO_GEOMETRY(3301, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 1), SDO_ORDINATE_ARRAY(29.486833,
8.16227766, 50))
```



图 18 线性参考的线中点

其中 SDO\_LRS 包的 LOCATE\_PT 函数原型如下：

```
SDO_LRS.LOCATE_PT(
geom_segment IN SDO_GEOMETRY,
measure IN NUMBER
```

```
[, offset IN NUMBER
) RETURN SDO_GEOMETRY;
```

注意到这里还可以指定一个 offset 值，这个值代表查找的定位点可以和线性参考的几何对象有一定的偏移。比如，如果上面的例子中这条线代表一条公路，现在我们想找到离公路中点一边（沿线的右手边）距离为 5 的某个点，那就可以使用这个 offset 值，注意，沿线左侧为正、右侧为负：

```
SQL> select SDO_LRS.LOCATE_PT(SDO_GEOMETRY(3302, NULL,
NULL,SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(5,10,0, 20,5,NULL, 35,10,NULL, 55,10,100)), 50,
-5) from dual;

SDO_LRS.LOCATE_PT(SDO_GEOMETRY(3302,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,2,1),SDO_ORDINATE_ARRAY(5,10,0,20,5,NULL,35,10,NULL,
-----
SDO_GEOMETRY(3301, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 1), SDO_ORDINATE_ARRAY(31.0679718,
3.41886117, 50))
```

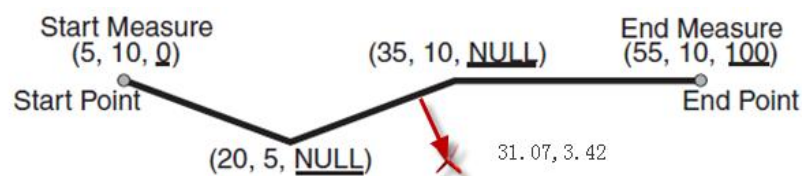


图 19 线性参考的线中点右侧距离 5 的点

## • ArcSDE

### 1. 创建线性参考的空间对象

对于如图 17 的例子，ArcSDE 中如果需要通过 SQL 语句进行创建可以采用以下的方式：

```
SQL> select sde.st_astext(sde.st_geometry ('linestring m(5 10 0, 20 5 16.67, 35 10 33.33, 55 10 53.33)', 0)) from
dual;

SDE.ST_ASTEXT(SDE.ST_GEOMETRY('LINESTRINGM(5100,20516.67,351033.33,551053.33)',0)
-----
LINESTRING M ( 5.00000000 10.00000000 0.00000000, 20.00000000 5.00000000 16.67000000, 35.00000000
10.00000000 33.33000000, 55.00000000 10.00000000 53.33000000)
```

## 2.根据线性参考定位点

ArcSDE 对线性参考几何对象的操作并不在数据库层面实现，而是在各种 ArcGIS 的产品中的 ArcObjects 进行操作，下面演示了采用 ArcGIS Server 的 ArcObjects 进行同上定位的操作：

```
ServerConnection conn = new ServerConnection();
conn.connect("localhost", "*", "arcgismanager", "passwd");
IServerObjectManager som = conn.getServerObjectManager();
IServerContext serverContext = som.createServerContext(null, null);

try {
    Polyline pl = (Polyline)serverContext.createObject(Polyline.getClsid());
    Point pt = null;

    pt = (Point) serverContext.createObject(Point.getClsid());
    pt.setX(5); pt.setY(10); pt.setM(0);
    pl.addPoint(pt, null, null);

    pt = (Point) serverContext.createObject(Point.getClsid());
    pt.setX(20); pt.setY(5); pt.setM(16.67);
    pl.addPoint(pt, null, null);

    pt = (Point) serverContext.createObject(Point.getClsid());
    pt.setX(35); pt.setY(10); pt.setM(33.33);
    pl.addPoint(pt, null, null);

    pt = (Point) serverContext.createObject(Point.getClsid());
    pt.setX(55); pt.setY(10); pt.setM(53.33);
    pl.addPoint(pt, null, null);

    pl.setMAware(true);

    Multipoint mp = (Multipoint)pl.getPointsAtM(26.67, 0);
    IPoint ptM = mp.getPoint(0);
    System.out.println(ptM.getX() + "," + ptM.getY());
} catch (Exception ex) {
    ex.printStackTrace();
} finally {
    serverContext.releaseContext();
}
```

这样的代码可以得到中点的坐标 ( 29.0036,8.0012 )，这个结果和 Oracle

Spatial 中基本相符 ( 这里为了简便采用了 0-16.67-33.33-53.33 这样不同的分段 )。下面通过修改 getPointsAtM()方法的参数查找线中点一边距离为 5 的点 , btw , 这里的 offset 参数的正负规则和 Oracle Spatial 中是恰恰相反的 :

```
Multipoint mp = (Multipoint)pl.getPointsAtM(26.67, 5);
```

这样可以得到结果坐标为 ( 30.5847,3.2578 )。

## VI. 其它

- 还没有涉及的话题

关于 Oracle Spatial 的 GeoRaster 支持本文没做研究。另外 ,Oracle Spatial 中现在还支持一些更 “GIS” 的功能，比如网络、拓扑；还有一些更 “Web” 的功能，比如 Web Service 的支持。对这些功能我的兴趣实在不大，由于精力有限暂时也就不再研究了。事实上从第 III 章开始，后面的比较都有些 “多余”，因为在实际应用中，如果采用 ArcSDE 和 ArcGIS 的产品，很少会直接在数据库服务器上执行这些空间关系运算、几何处理等操作，取而代之的是在客户端的 ArcGIS Engine 或者应用服务器的 ArcGIS Server 等环境中进行的。而与此相对的是，如果仅采用 Oracle Spatial，必然将基本的空间数据存取和其它高级 GIS 业务的压力（通常是高 CPU 资源消耗）都放到数据库服务器，这样的系统设计我觉得也并不太可取。

而且 ,ArcSDE 和 Oracle Spatial 也并不冲突 ,ArcSDE 支持 ST\_Geometry 类型的存储，同样也支持 SDO\_Geometry 类型的存储，使用 ArcSDE 也并不意味着就摒弃了 Oracle Spatial，从这个角度来说采用 ArcSDE 无疑比仅采用 Oracle Spatial 有着更多的技术方案选择。

- **采用 Oracle Spatial 而完全摒弃 ArcSDE 的解决方案？**

有人声称自己的解决方案完全基于 Oracle Spatial 而抛弃了 ArcSDE，前端基于 ArcGIS Engine 开发，我实在无法想象这样的“声称”是如何实现的。ArcGIS 产品中将空间数据类型统一抽象为“Geodatabase”，除非通过 ArcSDE 提供的工具把 Oracle Spatial 的数据注册到 ArcSDE 中成为 Geodatabase 数据，否则 Oracle Spatial 中仅仅包含空间数据，而没有 Geodatabase 模型，那样是不可能直接在 ArcGIS 的产品中被加载使用的。

当然，如果是自己开发客户端或者服务器软件，仅采用 Oracle Spatial 当然是可以的——这就是另外一个话题了。