



# SQL Server Spatial的基本几何模型及空间索引 与PostGIS的差异

地信1601

王泺棋

浙江大学



## Part 0: SQL Server Spatial 技术简介

- SQL SERVER 2008的spatial技术可以实现空间数据与属性数据的一体化存储与管理并实现对空间数据和属性数据的查询。对象-关系模型是spatial支持的表现空间数据的模型。

空间数据表示有关几何对象的物理位置和形状的信息。这些对象可以是点位置或更复杂的对象，例如国家/地区、公路或湖泊。

SQL Server 支持两种空间数据类型：**geometry** 数据类型和 **geography** 数据类型。

- **Geometry** 类型表示欧几里得（平面）坐标系中的数据。
- **Geography** 类型表示圆形地球坐标系中的数据。

这两种数据类型在 SQL Server中都是作为 .NET 公共语言运行时 (CLR) 数据类型实现的。

浙江大学



## Part 1: 空间数据类型概述

### Spatial Data Types Overview

- Point
- LineString
- CircularString
- CompoundCurve
- Polygon
- CurvePolygon
- MultiPoint
- MultiLineString
- MultiPolygon
- GeometryCollection



# Point

- 在 SQL Server 空间数据中， **Point** 是表示单个位置的零维对象，可能包含 Z（elevation）和 M（measure）值。

- Geography 数据类型

Geography 数据类型的 Point 类型表示单个位置，其中 *Lat* 表示纬度， *Long* 表示经度。维度和经度值以度数进行衡量。纬度值始终处于间隔  $[-90, 90]$  内，如果输入的值超出此范围，将引发异常。经度值始终处于间隔  $(-180, 180]$  内，如果输入的值超出此范围，将对值进行回绕以便适合此范围。例如，如果为经度值输入 190，则该值将被回绕到值 -170。 *SRID* 表示您希望返回的 **geography** 实例的空间引用 ID。

- Geometry 数据类型

Geometry 数据类型的 Point 类型表示单个位置，其中 *X* 表示要生成的点的 X 坐标， *Y* 表示要生成的点的 Y 坐标。 *SRID* 表示您希望返回的 **geometry** 实例的空间引用 ID。



# Point —— Example

下面的示例创建一个表示点 (3, 4) 的 `geometry Point` 实例，该实例的 SRID 为 0。

```
DECLARE @g geometry;  
SET @g = geometry::STGeomFromText('POINT (3 4)', 0);
```

[复制](#)

下一个示例创建一个表示点 (3, 4) 的 `geometry``Point` 实例，该实例的 Z（仰角）值为 7，M（度量）值为 2.5，默认 SRID 为 0。

```
DECLARE @g geometry;  
SET @g = geometry::Parse('POINT(3 4 7 2.5)');
```

[复制](#)

最后一个示例返回 `geometry``Point` 实例的 X、Y、Z 和 M 值。

```
SELECT @g.STX;  
SELECT @g.STY;  
SELECT @g.Z;  
SELECT @g.M;
```

[复制](#)

Z 和 M 值可以显式指定为 NULL，如下例所示。

```
DECLARE @g geometry;  
SET @g = geometry::Parse('POINT(3 4 NULL NULL)');
```

[复制](#)

浙江大学



# MultiPoint && Example

- **MultiPoint** 是零个点或更多个点的集合。 **MultiPoint** 实例的边界为空。

The following example creates a `geometry MultiPoint` instance with SRID 23 and two points: one point with the coordinates (2, 3), one point with the coordinates (7, 8), and a Z value of 9.5.

```
DECLARE @g geometry;  
SET @g = geometry::STGeomFromText('MULTIPOINT((2 3), (7 8 9.5))', 23);
```

[Copy](#)

This `MultiPoint` instance can also be expressed using `STMPPointFromText()` as shown below.

```
DECLARE @g geometry;  
SET @g = geometry::STMPPointFromText('MULTIPOINT((2 3), (7 8 9.5))', 23);
```

[Copy](#)

浙江大学



# LineString

- **LineString** 是一个一维对象，表示一系列点和连接这些点的线段。

- **Accepted Instances**

Accepted LineString instances can be input into a geometry variable, but they may not be valid LineString instances.

The following criteria must be met for a LineString instance to be accepted. The instance must be formed of at least two points or it must be empty.

- **Valid Instances**

For a **LineString** instance to be valid it must meet the following criteria.

1. The **LineString** instance must be accepted.
2. If a **LineString** instance is not empty then it must contain at least two distinct points.
3. The **LineString** instance cannot overlap itself over an interval of two or more consecutive points.

浙江大学



# LineString — Examples

The following example shows how to create a `geometry::LineString` instance with three points and an SRID of 0:

```
DECLARE @g geometry;  
SET @g = geometry::STGeomFromText('LINESTRING(1 1, 2 4, 3 9)', 0);
```

Copy

下面的示例说明如何创建一个包含两个相同点的 `geometry LineString` 实例。对 `IsValid` 的调用指示 **LineString** 实例无效，并且对 `MakeValid` 的调用会将 **LineString** 实例转换为 **Point**。

SQL

复制

```
DECLARE @g geometry  
SET @g = geometry::STGeomFromText('LINESTRING(1 3, 1 3)',0);  
IF @g.STIsValid() = 1  
BEGIN  
    SELECT @g.ToString() + ' is a valid LineString.';  
END  
ELSE  
BEGIN  
    SELECT @g.ToString() + ' is not a valid LineString.';  
    SET @g = @g.MakeValid();  
    SELECT @g.ToString() + ' is a valid Point.';  
END
```

上面的代码段将返回以下结果：

```
LINESTRING(1 3, 1 3) is not a valid LineString  
POINT(1 3) is a valid Point.
```

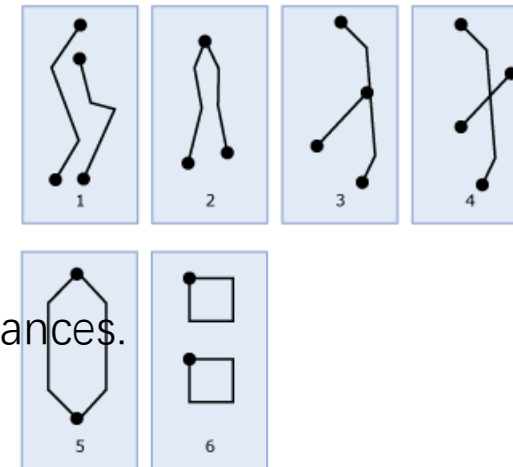
复制

浙江大学





# MultiLineString && Example



- A **MultiLineString** is a collection of zero or more **geometry** or **geographyLineString** instances.

- Valid Instances**

- All instances comprising the **MultiLineString** instance must be valid **LineString** instances.
- No two **LineString** instances comprising the **MultiLineString** instance may **overlap** over an interval.

The **LineString** instances can only intersect or touch themselves or other **LineString** instances at a finite number of points.

The following example creates a simple `geometry`MultiLineString` instance containing two `LineString` elements with the SRID 0.

```
DECLARE @g geometry;  
SET @g = geometry::Parse('MULTILINESTRING((0 2, 1 1), (1 0, 1 1))');
```

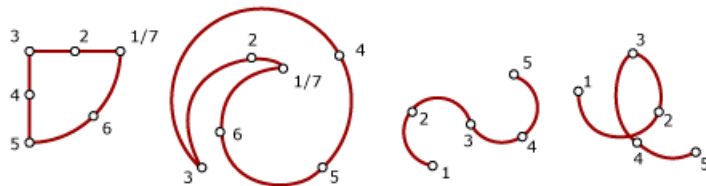
To instantiate this instance with a different SRID, use `STGeomFromText()` or `STMLineStringFromText()`. You can also use `Parse()` and then modify the SRID, as shown in the following example.

```
DECLARE @g geometry;  
SET @g = geometry::Parse('MULTILINESTRING((0 2, 1 1), (1 0, 1 1))');  
SET @g.STSrid = 13;
```

浙江大學



# CircularString



- **CircularString** 是零个或多个连续圆弧线段的集合。圆弧线段是二维平面中由三个点定义的曲线段；第一个点不能与第三个点相同。如果圆弧线段的所有三个点共线，则将该圆弧线段视为一条直线段。

- **Accepted Instances**

A **CircularString** instance is accepted if it is either empty or contains an odd number of points,  $n$ , where  $n > 1$ .

- **Valid Instances**

1. It must contain at least one circular arc segment (that is, have a minimum of three points).
2. The last endpoint for each circular arc segment in the sequence, except for the last segment, must be the first endpoint for the next segment in the sequence.
3. It must have an odd number of points.
4. It cannot overlap itself over an interval.
5. Although **CircularString** instances may contain line segments, these line segments must be defined by three collinear points.



# CircularString — Examples

The following example shows how to create a **CircularString** instance with more than one circular arc segment (full circle):

SQL

Copy

```
DECLARE @g geometry;  
SET @g = geometry::Parse('CIRCULARSTRING(2 1, 1 2, 0 1, 1 0, 2 1)');  
SELECT 'Circumference = ' + CAST(@g.STLength() AS NVARCHAR(10));
```

This produces the following output:

Copy

```
Circumference = 6.28319
```

This snippet shows how to declare and instantiate a **geometry** instance with a **CircularString** in the same statement:

SQL

Copy

```
DECLARE @g geometry = 'CIRCULARSTRING(0 0, 1 2.1082, 3 6.3246, 0 7, -3 6.3246, -1 2.1082, 0 0)';
```

This snippet shows how to declare and instantiate a **geometry** instance with a **CircularString** in the same statement:

SQL

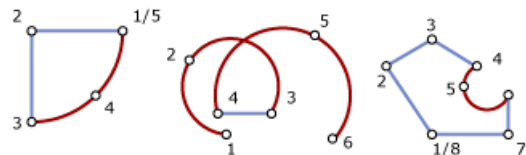
Copy

```
DECLARE @g geometry = 'CIRCULARSTRING(0 0, 1 2.1082, 3 6.3246, 0 7, -3 6.3246, -1 2.1082, 0 0)';
```

浙江大学



# CompoundCurve



- **CompoundCurve** 是几何图形或地域类型的零个或多个连续 **CircularString** 或 **LineString** 实例的集合。
- **Accepted Instances**
  1. All the instances contained by **CompoundCurve** instance are accepted circular arc segment instances.
  2. All of the circular arc segments in the **CompoundCurve** instance are connected. The first point for each succeeding circular arc segment is the same as the last point on the preceeding circular arc segment.
- **Valid Instances**
  1. The **CompoundCurve** instance is accepted.
  2. All circular arc segment instances contained by the **CompoundCurve** instance are valid instances.



# CompoundCurve — Examples

The following example shows how to declare and initialize a `geometry` instance with a `CompoundCurve` in the same statement:

SQL

Copy

```
DECLARE @g geometry = 'COMPOUNDCURVE ((2 2, 0 0),CIRCULARSTRING (0 0, 1 2.1082, 3 6.3246, 0 7, -3 6.3246, -1 2.1082, 0 0))';
```

The following example uses two different ways to use a `CompoundCurve` instance to store a square.

SQL

Copy

```
DECLARE @g1 geometry, @g2 geometry;  
SET @g1 = geometry::Parse('COMPOUNDCURVE((1 1, 1 3), (1 3, 3 3),(3 3, 3 1), (3 1, 1 1))');  
SET @g2 = geometry::Parse('COMPOUNDCURVE((1 1, 1 3, 3 3, 3 1, 1 1))');  
SELECT @g1.STLength(), @g2.STLength();
```

The following example shows how multiple `CircularString` and `LineString` instances can be stored by using a `CompoundCurve`.

SQL

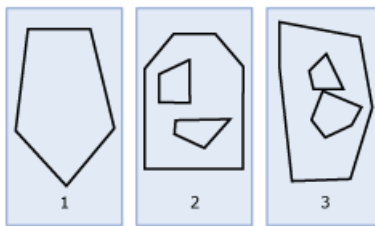
Copy

```
DECLARE @g geometry  
SET @g = geometry::Parse('COMPOUNDCURVE((3 5, 3 3), CIRCULARSTRING(3 3, 5 1, 7 3), (7 3, 7 5), CIRCULARSTRING(7 5, 5 7, 3 5))');  
SELECT @g.STLength();
```

浙江大学



# Polygon



- **Polygon** 是存储为一系列点的二维表面，这些点定义一个外部边界环和零个或多个内部环。

- **Accepted Instances**

1. An Empty **Polygon** instance
2. A **Polygon** instance that has **an acceptable exterior ring** and **zero or more acceptable interior rings**

The following criteria are needed for a ring to be acceptable.

- The **LineString** instance must be accepted.
- The **LineString** instance must have at least four points.
- The starting and ending points of the **LineString** instance must be the same.

- **Valid Instances**

The interior rings of a **Polygon** can touch both themselves and each other at single tangent points, but if the interior rings of a **Polygon** cross, the instance is not valid..



# Polygon — Examples

The following example creates a simple `geometry`Polygon` instance with a hole and SRID 10.

```
DECLARE @g geometry;  
SET @g = geometry::STPolyFromText('POLYGON((0 0, 0 3, 3 3, 3 0, 0 0), (1 1, 1 2, 2 1, 1 1))', 10);
```

[Copy](#)

An instance that is not valid may be entered and converted to a valid `geometry` instance. In the following example of a `Polygon`, the interior and exterior rings overlap and the instance is not valid.

```
DECLARE @g geometry;  
SET @g = geometry::Parse('POLYGON((1 0, 0 1, 1 2, 2 1, 1 0), (2 0, 1 1, 2 2, 3 1, 2 0))');
```

[Copy](#)

In the following example, the invalid instance is made valid with `MakeValid()`.

```
SET @g = @g.MakeValid();  
SELECT @g.ToString();
```

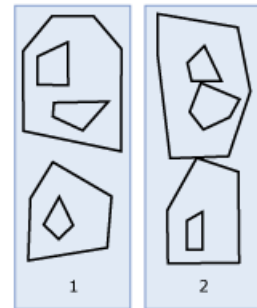
[Copy](#)

浙江大学



# MultiPolygon

- **MultiPolygon** 实例是零个或更多个 **Polygon** 实例的集合。



- **Accepted Instances**

1. It is an empty **MultiPolygon** instance.
2. All instances comprising the **MultiPolygon** instance are accepted **Polygon** instances.

- **Valid Instances**

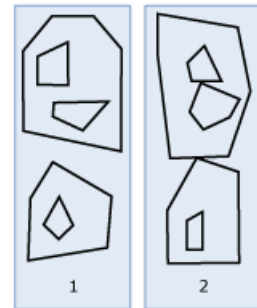
A **MultiPolygon** instance is valid if it is an empty **MultiPolygon** instance or if it meets the following criteria.

1. All of the instances comprising the **MultiPolygon** instance are valid **Polygon** instances.
2. None of the **Polygon** instances comprising the **MultiPolygon** instance overlap.





# MultiPolygon — Example



The following example shows the creation of a `geometry`MultiPolygon` instance and returns the Well-Known Text (WKT) of the second component.

```
DECLARE @g geometry;  
SET @g = geometry::Parse('MULTIPOLYGON(((0 0, 0 3, 3 3, 3 0, 0 0), (1 1, 1 2, 2 1, 1 1)), ((9 9, 9 10, 10 9, 9 9)))');  
SELECT @g.STGeometryN(2).STAsText();
```

This example instantiates an empty `MultiPolygon` instance.

```
DECLARE @g geometry;  
SET @g = geometry::Parse('MULTIPOLYGON EMPTY');
```



# CurvePolygon

- **CurvePolygon** 是由一个外部边界环以及零个或多个内环界定的在拓扑结构上闭合的图面。

**CurvePolygon** 实例不同于 **Polygon** 实例，因为 **CurvePolygon** 实例可以包含以下圆弧线段：

**CircularString** 和 **CompoundCurve**。

- **Accepted Instances**

1. Is an accepted **LineString**, **CircularString**, or **CompoundCurve** instance.
2. Has at least four points.
3. The start and end point have the same X and Y coordinates.

- **Valid Instances**

1. They may only touch at single tangent points.
2. They cannot cross each other.
3. Each ring must contain at least four points.
4. Each ring must be an acceptable curve type.

浙江大学



# CurvePolygon — Examples

This example shows how to store a simple circle in a **CurvePolygon** instance (only an exterior bounding ring is used to define the circle):

SQL

Copy

```
DECLARE @g geometry;  
SET @g = geometry::Parse('CURVEPOLYGON(CIRCULARSTRING(2 4, 4 2, 6 4, 4 6, 2 4))');  
SELECT @g.STArea() AS Area;
```

This example creates a donut in a **CurvePolygon** instance (both an exterior bounding ring and an interior ring is used to define the donut):

SQL

Copy

```
DECLARE @g geometry;  
SET @g = geometry::Parse('CURVEPOLYGON(CIRCULARSTRING(0 4, 4 0, 8 4, 4 8, 0 4), CIRCULARSTRING(2 4, 4 2, 6 4, 4 6, 2 4))');  
SELECT @g.STArea() AS Area;
```

浙江大学



# GeometryCollection

- **GeometryCollection** 是零个或多个 **geometry** 或 **geography** 实例的集合。 **GeometryCollection** 可以为空。

- **Accepted Instances**

For a **GeometryCollection** instance to be accepted, it must either be an empty **GeometryCollection** instance or all the instances comprising the **GeometryCollection** instance must be accepted instances

- **Valid Instances**

A **GeometryCollection** instance is valid when all instances that comprise the **GeometryCollection** instance are valid.

## Examples [🔗](#)

The following example instantiates a `geometry`GeometryCollection` with Z values in SRID 1 containing a `Point` instance and a `Polygon` instance.

```
DECLARE @g geometry;  
SET @g = geometry::STGeomCollFromText('GEOMETRYCOLLECTION(POINT(3 3 1), POLYGON((0 0 2, 1 1 3, 1 0 4, 0 0 2)))', 1);
```

[📄 Copy](#)

浙江大学



## Part 2: 空间索引概述

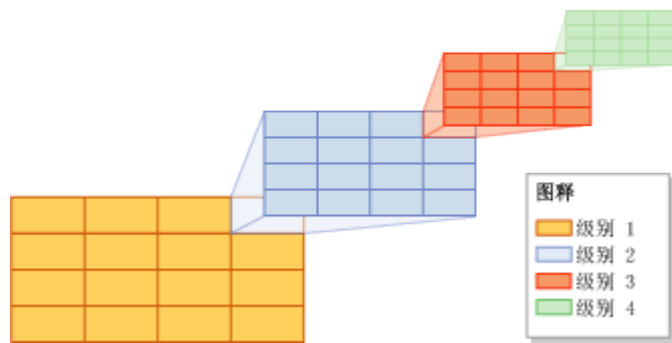
### Spatial Indexs Overview

SQL Server 支持空间数据和空间索引。“空间索引”是一种扩展索引，允许您对空间列编制索引。空间列是包含空间数据类型（如 geometry 或 geography）的数据的表列



## 将索引空间分解成网格层次结构

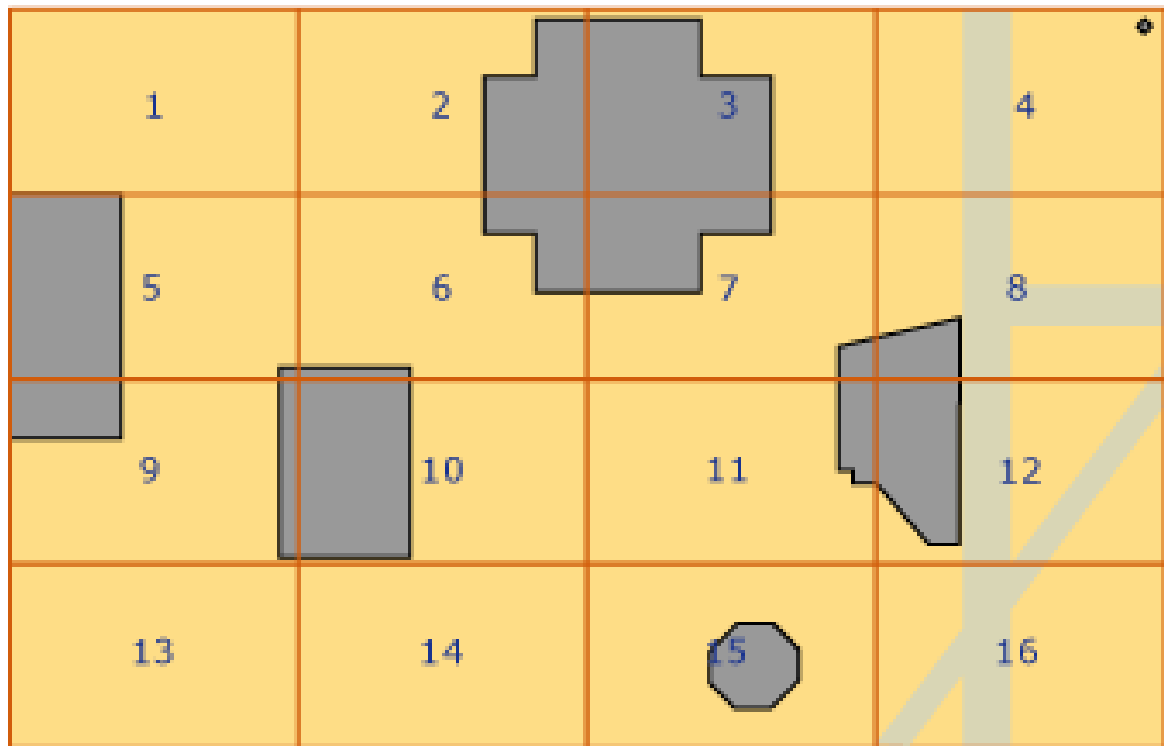
- 在 SQL Server 中，空间索引使用 B 树构建而成，也就是说，这些索引必须按 B 树的线性顺序表示二维空间数据。因此，将数据读入空间索引之前，SQL Server 先实现对空间的分层均匀分解。索引创建过程会将空间分解成一个四级网格层次结构。这些级别指的是第 1 级（顶级）、第 2 级、第 3 级和第 4 级。
- 每个后续级别都会进一步分解其上一级，因此上一级别的每个单元都包含下一级别的整个网格。在给定级别上，所有网格沿两个轴都有相同数目的单元（例如 4x4 或 8x8），并且单元的大小都相同。
- 下图显示了网格层次结构每个级别的右上角单元被分解成 4x4 网格的情况。事实上，所有单元都是以这种方式分解的。因此，以此为例，将一个空间分解成四个级别的 4x4 网格实际上会总共产生 65,536 个第四级单元。





## 将索引空间分解成网格层次结构

- 网格层次结构的单元是利用多种 Hilbert 空间填充曲线以线性方式编号的。然而，出于演示目的，这里使用的是简单的按行编号，而不是由 Hilbert 曲线实际产生的编号。在下图中，几个表示建筑物的多边形和表示街道的线已经放进了一个 4x4 的 1 级网格中。





## 网格密度

- 沿网格轴的单元数目确定了网格的“密度”：单元数目越大，网格的密度越大。例如，8x8 网格（产生 64 个单元）的密度就大于 4x4 网格（产生 16 个单元）的密度。网格密度是以每个级别为基础定义的。
- 空间索引的 CREATE SPATIAL INDEX Transact-SQL 语句支持 GRIDS 子句，使用该子句可以在不同级别指定不同的网格密度。可以使用下列关键字之一指定给定级别的网格密度。

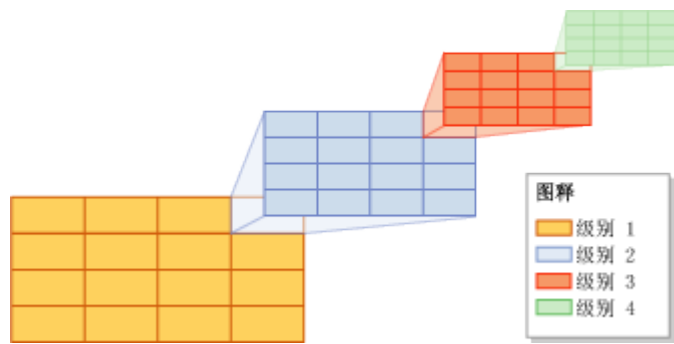
关键字	网格配置	单元数目
LOW	4X4	16
MEDIUM	8X8	64
HIGH	16X16	256





## 分割

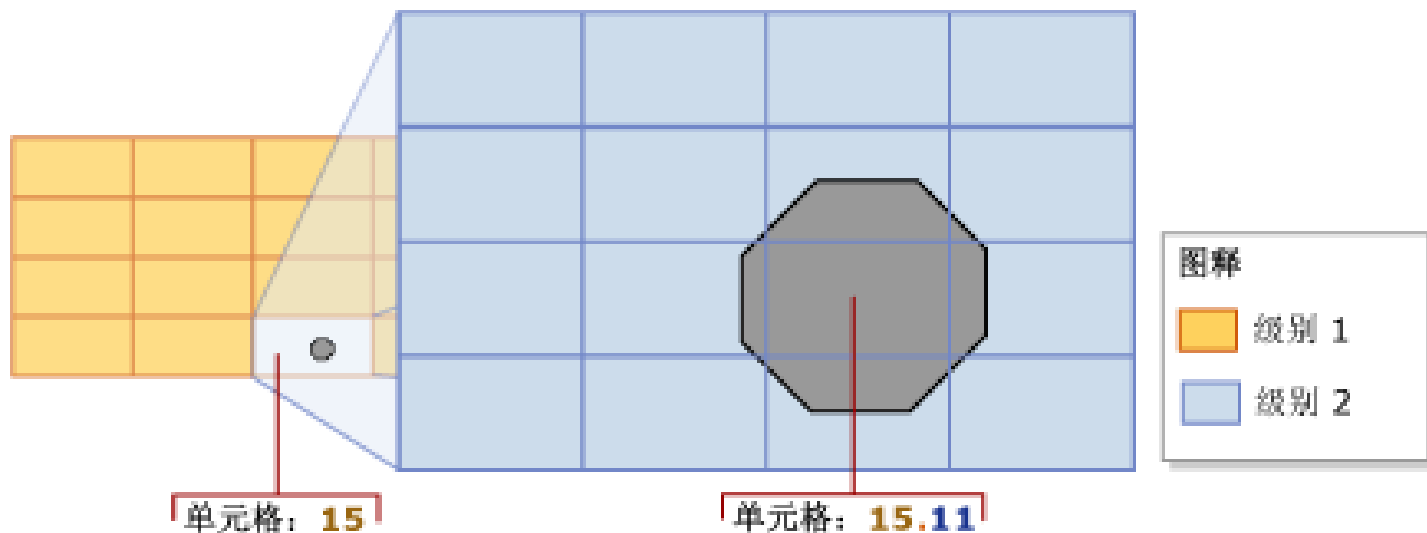
- 将索引空间分解成网格层次结构后，空间索引将逐行读取空间列中的数据。读取空间对象（或实例）的数据后，空间索引将为该对象执行 分割过程。分割过程通过将对象与其接触的网格单元集（“接触单元”）相关联使该对象适合网格层次结构。从网格层次结构的第 1 级开始，分割过程以“广度优先”方式对整个级别进行处理。在可能的情况下，此过程可以连续处理所有四个级别，一次处理一个级别。
- 分割过程的输出为对象的空间索引中所记录的接触单元集。通过引用这些已记录单元，空间索引可以确定该对象在空间中相对于空间列中也存储在索引中的其他对象的位置。





# 分割规则

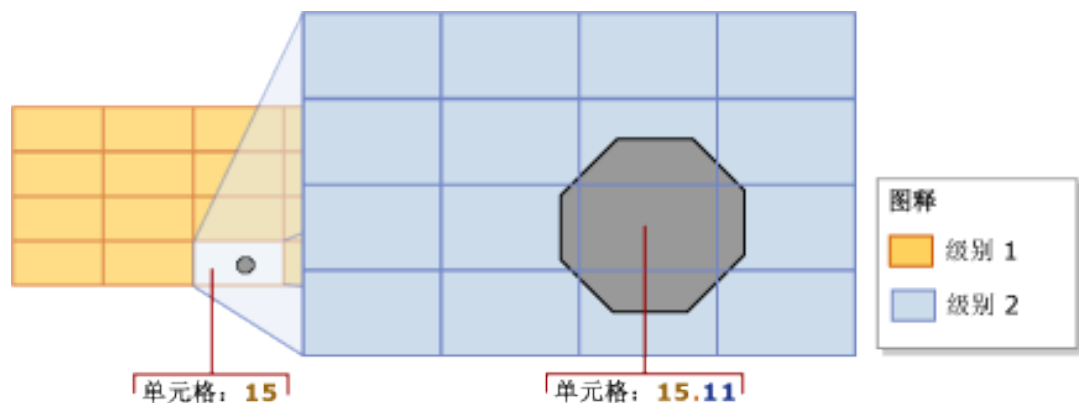
- 覆盖规则
- 如果一个对象完全盖住了某个单元，则称该单元由该对象所“覆盖”。被覆盖的单元会参与计数，但不进行分割。此规则应用于网格层次结构的所有级别。覆盖规则简化了分割过程，并减少了空间索引记录的数据量。





## 分割规则

- 每对象单元数规则
- 此规则强制执行每个对象的单元数限制，该限制确定每个对象可以具有的最大单元数（级别 1 例外）。在较低级别上，每个对象的单元数规则控制可以记录有关该对象的信息量。

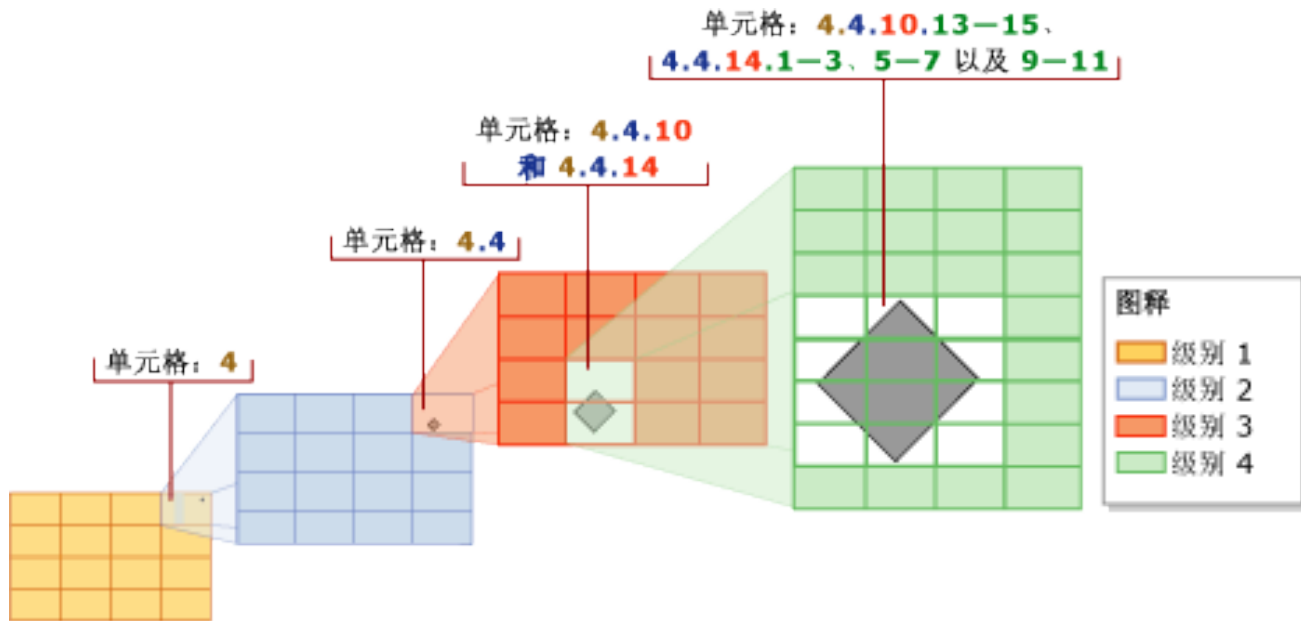


例如，上图显示了一个完全适合第 1 级网格的单元 15 的八边形。在此图中，单元 15 已进行分割，将八边形分成了九个二级单元。此图假定每对象单元数限制为 9 或更大。然而，如果每对象单元数限制为 8 或更小，则单元 15 将不进行分割，而只为该对象对单元 15 进行计数。



# 分割规则

- 最深单元规则
- 最深单元规则通过只记录已为对象分割的最底部单元来生成该对象的最近似对象。父单元不计入每对象单元数，这些单元不记录在索引中。



使用最深单元规则，分割将仅对十二个位于第 4 级的单元进行计数：4.4.10.13-15 以及 4.4.14.1-3、4.4.14.5-7 和 4.4.14.9-11。



## 分割方案

- 空间索引的行为部分取决于“分割方案”。分割方案特定于数据类型。在 SQL Server 中，空间索引支持两种分割方案：
- “几何图形网格分割”，这是适用于 geometry 数据类型的方案。
- “地理网格分割”，该方案适用于数据类型为 geography 的列。



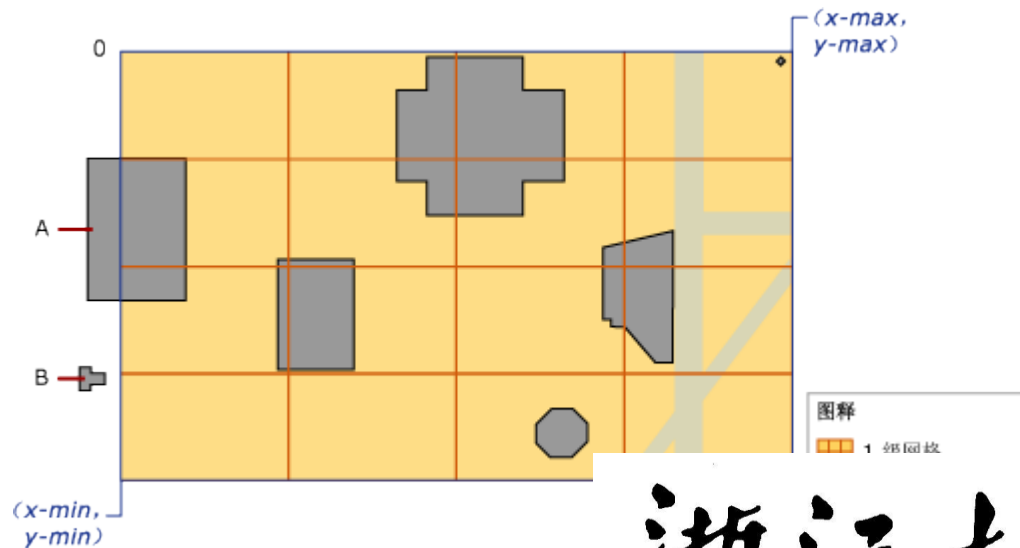
# 几何图形网格分割

- 边界框

- 几何图形数据占有的平面可以是无限的。然而，在 SQL Server 中，空间索引需要有限空间。为了建立有限空间以用于分解，几何图形网格分割方案需要矩形“边界框”。该边界框由四个坐标 ( $x\text{-min}$ 、 $y\text{-min}$ ) 和 ( $x\text{-max}$ 、 $y\text{-max}$ ) 定义，这些坐标存储为空间索引的属性。这些坐标所表示的意义如下：

- $x\text{-min}$  是边界框左下角的  $x$  坐标。
- $y\text{-min}$  是左下角的  $y$  坐标。
- $x\text{-max}$  是右上角的  $x$  坐标。
- $y\text{-max}$  是右上角的  $y$  坐标。

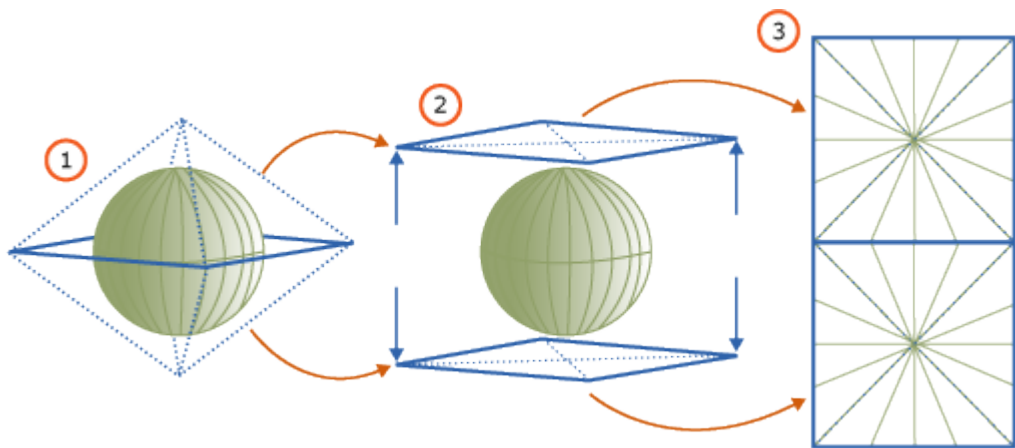
- 边界框的外部空间视作一个编号为 0 的单元。





# 地理网格分割方案

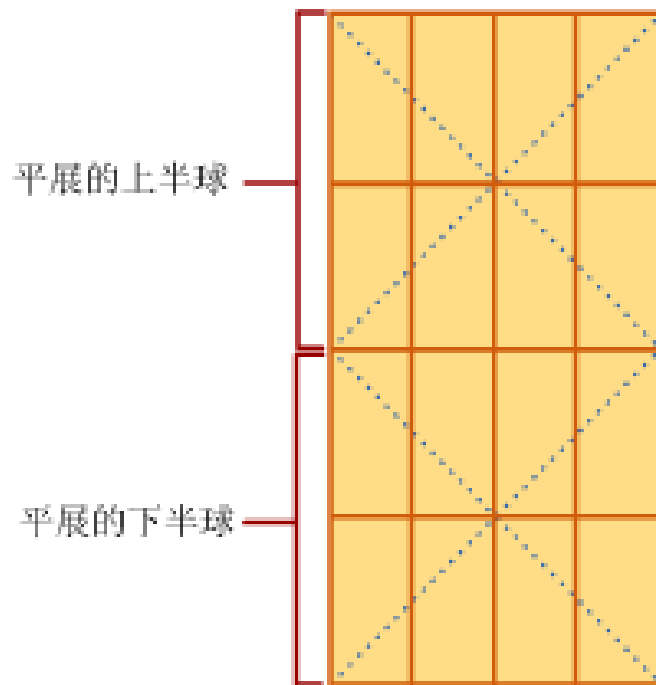
- 将测量空间投影到平面上
- 对 geography 实例（对象）的计算将包含对象的空间视为测量椭圆体。若要分解此空间，地理网格分割方案将椭圆体表面分为上半球和下半球，然后执行下列步骤：
  1. 将每个半球投影在四边形棱锥图面上。
  2. 将两个棱锥图平展开。
  3. 联接平展的棱锥图以形成非欧几里得平面。





## 地理网格分割方案

- 空间投影到平面上之后，此平面将会分解成四级网格层次结构。不同级别可以使用不同的网格密度。下图显示了已分解成一个 4x4 的 1 级网格后的平面。出于演示目的，这里省略了网格层次结构的较低级别。事实上，此平面完全分解成了一个四级网格层次结构。分解过程完成后，将逐行从 geography 列读取地理数据，并为每个对象依次执行分割过程。



浙江大学





# 空间索引支持的方法

## 空间索引支持的几何图形方法

空间索引在某些情况下支持以下面向集合的 geometry 方法：STContains()、STDistance()、STEquals()、STIntersects()、STOverlaps()、STTouches() 和 STWithin()。若要使空间索引支持这些方法，必须在查询的 WHERE 或 JOIN ON 子句中使用这些方法，并且必须在采用如下常规形式的谓词中执行这些方法：

**geometry1.method\_name(geometry2)comparison\_operatorvalid\_number**

若要返回非 NULL 结果， geometry1 和 geometry2 必须具有相同的 空间引用标识符 (SRID)。 否则，该方法将返回 NULL。

空间索引支持以下谓词形式：

**geometry1.STContains(geometry2) = 1**

**geometry1.STDistance(geometry2) < number**

**geometry1.STDistance(geometry2) <= number**

**geometry1.STEquals(geometry2)= 1**

**Geometry1.STIntersects(geometry2)= 1**

**geometry1. STOverlaps (geometry2) = 1**

**Geometry1.STTouches(geometry2) = 1**

**Geometry1.STWithin(geometry2) = 1**

浙江大学



# 空间索引支持的方法

## 空间索引支持的地域方法

在某些条件下，空间索引支持以下面向集合的地理方法：STIntersects()、STEquals() 和 STDistance()。若要使空间索引支持这些方法，必须在查询的 WHERE 子句中使用这些方法，并且必须在采用如下常规形式的谓词中执行这些方法。

**geography1.method\_name(geography2)comparison\_operatorvalid\_number**

若要返回非 NULL 结果，geography1 和 geography2 必须具有相同的空间引用标识符 (SRID)。否则，该方法将返回 NULL。

空间索引支持以下谓词形式：

**geography1.STIntersects(geography2) = 1**

**geography1.STEquals(geography2) = 1**

**geography1.STDistance(geography2) < number**

**geography1.STDistance(geography2) <= number**

浙江大学



# 创建、修改和删改空间索引

## Create, Modify, and Drop Spatial Indexes

空间索引可以更高效地对数据类型为“geometry”或“geography”的列（空间数据列）执行特定操作。 可对空间数据列指定多个空间索引。 这非常有用，例如，对单一系列中的不同分割参数建立索引时，就是如此。



## 创建空间索引

- 使用 Transact-SQL 创建空间索引
- 在 Management Studio 中使用“新建索引”对话框创建空间索引
- 在 Management Studio 中使用表设计器创建空间索引



# 创建空间索引

## • 使用 Transact-SQL 创建空间索引

### A. 对 geometry 列创建空间索引 [🔗](#)

以下示例创建一个名为 `SpatialTable` 的表，表中包含一个 **geometry** 类型的列 `geometry_col`。然后，该示例将对 `SIndx_SpatialTable_geometry_col1` 创建一个空间索引 `geometry_col`。该示例使用默认分割方案并指定边界框。

SQL

🔗 复制

```
CREATE TABLE SpatialTable(id int primary key, geometry_col geometry);
CREATE SPATIAL INDEX SIndx_SpatialTable_geometry_col1
ON SpatialTable(geometry_col)
WITH ( BOUNDING_BOX = ( 0, 0, 500, 200 ) );
```

### F. 对 geography 列创建空间索引

下例对 `SIndx_SpatialTable_geography_col2` 表中的 `geography_col` 列创建第二个空间索引 `SpatialTable2`。该示例指定 `GEOGRAPHY_GRID` 作为分割方案。它还为不同网格级别指定了不同的网格密度，并指定每个对象有 64 个单元格。该示例还将索引填充设为 `ON`。

SQL

🔗 复制

```
CREATE SPATIAL INDEX SIndx_SpatialTable_geography_col2
ON SpatialTable2(object)
USING GEOGRAPHY_GRID
WITH (
GRIDS = (MEDIUM, LOW, MEDIUM, HIGH ),
CELLS_PER_OBJECT = 64,
PAD_INDEX = ON );
```

浙江大学



# 创建空间索引

## • 在 Management Studio 中使用“新建索引”对话框创建空间索引

1. 在对象资源管理器中，连接到 SQL Server 数据库引擎 的实例，然后展开该实例。
  2. 展开“数据库”，展开包含具有指定索引的表的数据库，再展开“表”。
  3. 展开要为其创建索引的表。
  4. 右键单击“索引”，再选择“新建索引”。
  5. 在“索引名称”字段中，输入索引的名称。
  6. 在“索引类型”下拉列表中，选择“空间”。
  7. 若要指定想为其创建索引的空间数据列，请单击“添加”。
  8. 在“从 <table name> 中选择列”对话框中，通过选中相应的复选框来选择类型为“geometry”或“geography”的列。然后，任何其他空间数据列将变为不可编辑状态。  
如果要选择其他空间数据列，必须首先清除当前选定的列。完成后，单击“确定”。
  9. 请在“索引键列”网格中验证您的列选择。
  10. 在“索引属性”对话框的“选择页”窗格中，单击“空间”。
  11. 在“空间”页上，指定要用于索引的空间属性的值。
- 在对类型为“geometry”的列创建索引时，必须指定边界框的（X-min、Y-min）和（X-max、Y-max）坐标。对于“geography”类型列的索引，当你指定“地理网格”分割方案后，边界框字段变为只读状态，因为地理网格分割不使用边界框。
  - 您还可以指定任意级别的分割方案的“每个对象的单元数”字段和网格密度的非默认值。对于 SQL Server 2008，每个对象的默认单元数为 16；对于 SQL Server 2012 (11.x) 或更高版本，则为 8。对于，默认网格密度为“中” SQL Server 2008。
  - 在 SQL Server 中，可以为分割方案选择 GEOMETRY\_AUTO\_GRID 或 GEOGRAPHY\_AUTO\_GRID。选择 GEOMETRY\_AUTO\_GRID 或 GEOGRAPHY\_AUTO\_GRID 时，禁用级别 1、级别 2、级别 3 和级别 4 网格密度选项。



## 创建空间索引

- 在 Management Studio 中使用表设计器创建空间索引

1. 在对象资源管理器中，右键单击要为其创建空间索引的表，然后单击“设计”。
2. 此时，将在表设计器中打开该表。
3. 为索引选择 geometry 列或 geography 列。
4. 在 表设计器 菜单上，单击“空间索引”。
5. 在“空间索引”对话框中，单击“添加”。
6. 在“所选空间索引”列表中选择新的索引，然后在右侧的网格中设置空间索引的属性。



## 更改空间索引

- ALTER INDEX (使用Transact-SQL)
- 使用 SQL Server Management Studio





# 更改空间索引

- ALTER INDEX (使用Transact-SQL)

将以下示例复制并粘贴到查询窗口中，然后单击“执行”。此示例使用 `ProductID` 选项在 `Production.WorkOrder` 表的 `DROP_EXISTING` 列上删除并重新创建现有索引。还设置了 `FILLFACTOR` 和 `PAD_INDEX` 选项。

SQL	复制
<pre>USE AdventureWorks2012; GO CREATE NONCLUSTERED INDEX IX_WorkOrder_ProductID ON Production.WorkOrder(ProductID) WITH (FILLFACTOR = 80,       PAD_INDEX = ON,       DROP_EXISTING = ON); GO</pre>	



## 更改空间索引

- 使用 SQL Server Management Studio

1. 在对象资源管理器中，连接到 SQL Server 数据库引擎 的实例，然后展开该实例。
2. 展开 “**数据库**”，展开该表所属的数据库，再展开 “**表**”。
3. 展开该索引所属的表，再展开 “**索引**”。
4. 右键单击要修改的索引，然后单击“属性”。
5. 在 “**索引属性**” 对话框中进行所需的更改。 例如，您可以从索引键中添加或删除列，或更改索引选项的设置。



## 删除空间索引

- 使用 Transact-SQL 删除空间索引
- 使用 Management Studio 删除索引
  - 通过使用对象资源管理器删除索引
  - 在 Management Studio 中使用表设计器删除空间索引



# 删除空间索引

- 使用 Transact-SQL 删除空间索引

## A. 删除索引

以下示例删除 AdventureWorks2012 数据库中 `ProductVendor` 表上的索引 `IX_ProductVendor_VendorID`。

```
DROP INDEX IX_ProductVendor_BusinessEntityID
ON Purchasing.ProductVendor;
GO
```

[复制](#)

## B. 删除多个索引

以下示例删除 AdventureWorks2012 数据库的单个事务中的两个索引。

```
DROP INDEX
IX_PurchaseOrderHeader_EmployeeID ON Purchasing.PurchaseOrderHeader,
IX_Address_StateProvinceID ON Person.Address;
GO
```

[复制](#)

浙江大学



## 删除空间索引

- 在 Management Studio 通过使用对象资源管理器删除索引

1. 在“对象资源管理器”中，展开包含您要删除索引的表的数据库。
2. 展开“表”文件夹。
3. 展开包含您要删除的索引的表。
4. 展开“索引”文件夹。
5. 右键单击要删除的索引，然后选择“删除”。
6. 在“删除对象”对话框中，确认正确的索引位于“要重删除的对象”网格中，然后单击“确定”。



## 删除空间索引

- 在 Management Studio 中使用表设计器删除空间索引

1. 在对象资源管理器中，右键单击具有要删除的空间索引的表，再单击“设计”。
2. 此时，将在表设计器中打开该表。
3. 在 **表设计器** 菜单上，单击 **“空间索引”**。
4. **“空间索引”** 对话框随即打开。
5. 在 **“所选空间索引”** 列中单击要删除的索引。
6. 单击 **“删除”**。



## Part3: SQL Server Spatial与 PostGIS的差异

- Data Types

a number of invalid polygons that have been accepted by SQL Server, and the lack of a ST\_MakeValid function can make this a bit painful.

- Spatial Functions

Feature	SQL Server 2008 (RC0)	PostgreSQL 8.3/PostGIS 1.3/1.4
Spatial Functions	Both OGC SFSQL MM and Geodetic custom (over 70 functions)	Over 300 functions and operators, no geodetic support except for point-2-point non-indexed distance functions, custom PostGIS for 2D and some 3D, some MM support of circular strings and compound curves



## Part3: SQL Server Spatial与 PostGIS的差异

- OS

SQL Server: Windows only;

PostGIS: Windows 2000+ (including Vista and 2003, haven't tested on 2008), Linux, Unix, Mac

- Spatial indexes

Feature	SQL Server 2008 (RC0)	PostgreSQL 8.3/PostGIS 1.3/1.4
Spatial Indexes	Yes - 4 level Multi-Level grid hierarchy (BOL says its B-Tree based) with tessalation as described Isaac Kunen Multi-Level Grid requires defining an index grid for optimal performance	GIST - a variant of R-Tree

浙江大学





terima Kasih

감사합니다!

ありがとう

Thank you

谢谢

Danke

shokran

Merci

浙江大学