Spatial Database – An SQL 2008 View

Vaitheeshwar Ramachandran
B.E Computer Science and Engineering
Sri Sai Ram Engineering College
Chennai, India
rvaitheeshwar@gmail.com

Abstract- A spatial database is a database that is optimized to store and query data related to objects in space, including points, lines and polygons. While typical databases can understand various numeric and character types of data, additional functionality needs to be added for databases to process spatial data types. These are typically called geometry or feature.

Keywords- Spatial Database, SQL 2008

I. INTRODUCTION

In various fields there is a need to manage geometric, geographic, or spatial data, which means data related to space. The space of interest can be, for example, the twodimensional abstraction of (parts of) the surface of the earth – that is, geographic space, the most prominent example –, a man-made space like the layout of a VLSI design, a volume containing a model of the human brain, or another 3d-space representing the arrangement of chains of protein molecules. At least since the advent of relational database systems there have been attempts to manage such data in database systems.

Characteristic for the technology emerging to address these needs is the capability to deal with large collections of relatively simple geometric objects, for example, a set of 100 000 polygons. This is somewhat different from areas like CAD databases (solid modeling etc.) where geometric entities are composed hierarchically into complex structures, although the issues are certainly related.

Several terms have been used for database systems offering such support like pictorial, image, geometric, geographic, or spatial database system. The terms “pictorial” and “image” database system arise from the fact that the data to be managed are often initially captured in the form of digital raster images (e.g. remote sensing by satellites, or computer tomography in medical applications). The term “spatial database system” started becoming popular to some extent through the series of conferences “Symposium on Large Spatial Databases (SSD)” held bi-annually since 1989, and is associated with a view of a database as containing sets of objects in space rather than images or pictures of a space. Indeed, the requirements and techniques for dealing with objects in space that have identity and well-defined extents, locations, and relationships are rather different from those for dealing with raster images.

A spatial database is a database that is optimized to store and query data related to objects in space, including points, lines and polygons. While typical databases can understand various numeric and character types of data, additional functionality needs to be added for databases to process spatial data types. These are typically called geometry or feature.

II. SPATIAL QUERY

A spatial query is a special type of database query supported by geo-databases. The queries differ from SQL queries in several important ways. Two of the most important are that they allow for the use of geometry data types such as points, lines and polygons and that these queries consider the
spatial relationship between these geometries.

Some popular spatial databases
- Oracle
- The Boeing Company's Spatial Query Server
- Microsoft SQL server
- Postgre SQL
- MYSQL
- NEO4j – Graph database

A. Microsoft SQL server

SQL Server 2008 and later versions support spatial data.

There are two types of spatial data.

III. GEOMETRY DATA TYPE

The geometry data type supports planar, or Euclidean (flat-earth), data. The geometry data type conforms to the Open Geospatial Consortium (OGC) Simple Features for SQL Specification version 1.1.0.

The Open Geospatial Consortium (OGC) is an international voluntary consensus standards organization, originated in 1994. In the OGC, more than 370+ commercial, governmental, nonprofit and research organizations worldwide collaborate in an open consensus process encouraging development and implementation of standards for geospatial content and services, GIS data processing and data sharing. The OGC standards baseline comprises more than 30 standards. Out of this SFS - Simple Features is one of it. Simple Features are an OpenGIS standard which specifies digital storage of geographical data (point, line, polygon, multi-point, multi-line, etc) with both spatial and non-spatial attributes. Simple Features are based on 2D geometry with linear interpolation between vertices. In general, a 2D geometry is simple if it contains no self-intersection. The OpenGIS Simple Features specifications define various spatial operators, which can be used to generate new geometries from existing geometries.

The planar spatial data type, geometry, is implemented as a .NET Common Language Runtime (CLR) data type in SQL Server. This type represents data in a Euclidean (flat) coordinate system.

A. Registering the geometry Type

The geometry type is predefined and available in each database. You can create table columns of type geometry and operate on geometry data in the same manner as you would use other CLR types.

B. Examples

The following two examples show how to add and query geometry data. The first example creates a table with an identity column and a geometry column GeomCol1. A third column renders the geometry column into its Open Geospatial Consortium (OGC) Well-Known Text (WKT) representation, and uses the STAsText() method. Two rows are then inserted: one row contains a LineString instance of geometry, and one row contains a Polygon instance.

IF OBJECT_ID ( 'dbo.SpatialTable', 'U' ) IS NOT NULL
    DROP TABLE dbo.SpatialTable;
GO
CREATE TABLE SpatialTable
    ( id int IDENTITY (1,1),
    GeomCol1 geometry,
    GeomCol2 AS GeomCol1.STAsText() );
GO
INSERT INTO SpatialTable (GeomCol1)
VALUES (geometry::STGeomFromText('LINESTRING (100 100, 20 180, 180 180)', 0));
INSERT INTO SpatialTable (GeomCol1)
VALUES (geometry::STGeomFromText('POLYGON ((0 0, 150 0, 150 150, 0 150, 0 0))', 0));
GO

The second example uses the STIntersection() method to return the points where the two previously inserted geometry instances intersect.

DECLARE @geom1 geometry;
DECLARE @geom2 geometry;
DECLARE @result geometry;

The second example uses the STIntersection() method to return the points where the two previously inserted geometry instances intersect.

DECLARE @geom1 geometry;
DECLARE @geom2 geometry;
DECLARE @result geometry;

SELECT @geom1 = GeomCol1 FROM SpatialTable
WHERE id = 1;
SELECT @geom2 = GeomCol1 FROM SpatialTable
WHERE id = 2;
SELECT @result = @geom1.STIntersection(@geom2);
SELECT @result.STAsText();

IV. GEOGRAPHY DATA TYPE

In addition, SQL Server supports the geography data type, which stores ellipsoidal (round-earth) data, such as GPS latitude and longitude coordinates.

The geography spatial data type, geography, is implemented as a .NET common language runtime (CLR) data type in SQL Server. This type represents data in a roundearth coordinate system.

A. Registering the geography Type

The geography type is predefined and available in each database. You can create table columns of type geography and operate on geography data in the same manner as you would use other system-supplied types.

B. Examples

The following examples show how to add and query geography data. The first example creates a table with an identity column and a geography column GeogCol1. A third column renders the geography column into its Open Geospatial Consortium (OGC) Well-Known Text (WKT) representation, and uses the STAsText() method. Two rows are then inserted: one row contains a LineString instance of geography, and one row contains a Polygon instance.

IF OBJECT_ID ( 'dbo.SpatialTable', 'U' ) IS NOT NULL
    DROP TABLE dbo.SpatialTable;
GO
CREATE TABLE SpatialTable
    ( id int IDENTITY (1,1),
    GeogCol1 geography,
    GeogCol2 AS GeogCol1.STAsText() );
GO
INSERT INTO SpatialTable (GeogCol1)
VALUES (geography::STGeomFromText('LINESTRING(-122.360 47.656, -122.343 47.656)', 4326));
INSERT INTO SpatialTable (GeogCol1)
VALUES (geography::STGeomFromText('POLYGON((-122.358 47.653, -122.348 47.649, -122.348 47.658, -122.358 47.658, -122.358 47.653))', 4326));
GO

The second example uses the STIntersection() method to return the points where the two previously inserted geography instances intersect.

DECLARE @geog1 geography;
DECLARE @geog2 geography;
DECLARE @result geography;
SELECT @geog1 = GeogCol1 FROM SpatialTable
WHERE id = 1;
SELECT @geog2 = GeogCol1 FROM SpatialTable
WHERE id = 2;
SELECT @result = @geog1.STIntersection(@geog2);
SELECT @result.STAsText();

V. TYPES OF SPATIAL DATA

The geometry and geography Data Types support eleven spatial data objects, or instance types. However, only seven of these instance types are instantiable; you can create and work with these instances (or instantiate them) in a database. These instances derive certain properties from their parent data types that distinguish them as Points, LineStrings, Polygons, or as multiple geometry or geography instances in a GeometryCollection.

The Fig. 1 depicts the geometry hierarchy upon which the geometry and geography data types are based. The instantiable types of geometry and geography are indicated in blue.

As the Fig. 1 indicates, the seven instantaneous types of the geometry and geography data types are Point, MultiPoint, LineString, MultiLineString, Polygon, MultiPolygon, and GeometryCollection.

The geometry and geography types can recognize a specific instance as long as it is a well-formed instance, even if the instance is not defined explicitly. For example, if you define a Point instance explicitly using the STPointFromText() method, geometry and geography recognize
the instance as a Point, as long as the method input is well-formed. If you define the same instance using the STGeomFromText() method, both the geometry and geography data types recognize the instance as a Point.

1) **Point**

A Point is a 0-dimensional object representing a single location and may contain Z (elevation) and M (measure) values.

To create geometry Point instance representing the point (3, 4) with an SRID of 0

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

![Fig. 1](image)

2) **Multi Point**

A MultiPoint is a collection of zero or more points. The boundary of a MultiPoint instance is empty.

To create 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);

3) **LineString**

A LineString is a one-dimensional object representing a sequence of points and the line segments connecting them. A LineString instance must be formed of at least two distinct points, and can also be empty.

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);

4) **MultiLineString**

A MultiLineString is a collection of zero or more geometry or geography LineString instances.

To create 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))');

5) **Polygon**

A Polygon is a two-dimensional surface stored as a sequence of points defining an exterior bounding ring and zero or more interior rings. A Polygon instance can be formed from a ring that has at least three distinct points. A Polygon instance can also be empty.

The exterior and any interior rings of a Polygon define its boundary. The space within the rings defines the interior of the Polygon. 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.

To create 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);
6) MultiPolygon

A MultiPolygon instance is a collection of zero or more Polygon instances.

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

```sql
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();
```

A. GeometryCollection

A GeometryCollection is a collection of zero or more geometry or geography instances. A GeometryCollection can be empty.

Following creates a geometry GeometryCollection with Z values in SRID 1 containing a Point instance and a Polygon instance.

```sql
DECLARE @g geometry;
SET @g = geometry::STGeomCollFromText('GEOMETRYCOLLECTION(POINT(3 3 1), POLYGON((0 0 2, 1 10 3, 1 0 4, 0 0 2)), 1);
```

Differences Between the Two Data Types

The two types of spatial data often behave quite similarly, but there are some key differences in how the data is stored and manipulated.

VI. MEASUREMENTS IN SPATIAL DATA TYPES

In the planar, or flat-earth, system, measurements of distances and areas are given in the same unit of measurement as coordinates. Using the geometry data type, the distance between (2, 2) and (5, 6) is 5 units, regardless of the units used.

In the ellipsoidal, or round-earth system, coordinates are given in degrees of latitude and longitude. However, lengths and areas are usually measured in meters and square meters, though the measurement may depend on the spatial reference identifier (SRID) of the geography instance. The most common unit of measurement for the geography data type is meters.

VII. ORIENTATION OF SPATIAL DATA

In the planar system, the ring orientation of a polygon is not an important factor. For example, a polygon described by 

```sql
((0, 0), (10, 0), (0, 20), (0, 0))
```

is the same as a polygon described by 

```sql
((0, 0), (0, 20), (10, 0), (0, 0))
```

The OGC Simple Features for SQL Specification does not dictate a ring ordering, and SQL Server does not enforce ring ordering.

In an ellipsoidal system, a polygon has no meaning, or is ambiguous, without an orientation. For example, does a ring around the equator describe the northern or southern hemisphere? If we use the geography data type to store the spatial instance, we must specify the orientation of the ring and accurately describe the location of the instance.

SQL Server 2008 places the following restrictions on using the geography data type:

- Each geography instance must fit inside a single hemisphere. No spatial objects larger than a hemisphere can be stored.
- Any geography instance from an Open Geospatial Consortium (OGC) Well-Known Text (WKT) or Well-Known Binary (WKB) representation that produces an object larger than a hemisphere throws an ArgumentException.
- The geography data type methods that require the input of two geography instances, such as STIntersection(), STUnion(), STDifference(), and STSymDifference(), will return null if the results from the methods do not fit inside a single hemisphere. STBuffer() will also return null if the output exceeds a single hemisphere.
A. Outer and Inner Rings Not Important in geography Data Type

The OGC Simple Features for SQL Specification discusses outer rings and inner rings, but this distinction makes little sense for the SQL Server geography data type: any ring of a polygon can be taken to be the outer ring.

Spatial Reference Identifiers (SRIDs)
Each spatial instance has a spatial reference identifier (SRID). The SRID corresponds to a spatial reference system based on the specific ellipsoid used for either flat-earth mapping or round-earth mapping. A spatial column can contain objects with different SRIDs. However, only spatial instances with the same SRID can be used when performing operations with SQL Server spatial data methods on your data. The result of any spatial method derived from two spatial data instances is valid only if those instances have the same SRID that is based on the same unit of measurement, datum, and projection used to determine the coordinates of the instances. The most common units of measurement of a SRID are meters or square meters.

Spatial Indexing
A spatial index is defined on a table column that contains spatial data (a spatial column). Each spatial index refers to a finite space. For example, an index for a geometry column refers to a user-specified rectangular area on a plane.

B. Decomposing Indexed Space into a Grid Hierarchy

In SQL Server 2008, spatial indexes are built using Btrees, which means that the indexes must represent the 2-dimensional spatial data in the linear order of B-trees. Therefore, before reading data into a spatial index, SQL Server 2008 implements a hierarchical uniform decomposition of space. The index-creation process decomposes the space into a four-level grid hierarchy. These levels are referred to as level 1 (the top level), level 2, level 3, and level 4.

Fig. 2

Each successive level further decomposes the level above it, so each upper-level cell contains a complete grid at the next level. On a given level, all the grids have the same number of cells along both axes (for example, 4x4 or 8x8), and the cells are all one size. The cells of a grid hierarchy are numbered in a linear fashion by using a variation of the Hilbert space-filling curve.

C. Grid Density

The number of cells along the axes of a grid determines its density: the larger the number, the denser the grid. For example, an 8x8 grid (which produces 64 cells), is denser than a 4x4 grid (which produces 16 cells). Grid density is defined on a per-level basis.

D. Tessellation

After decomposition of an indexed space into a grid hierarchy, the spatial index reads the data from the spatial column, row-by-row. After reading the data for a spatial object (or instance), the spatial index performs a tessellation process for that object. The tessellation process fits the object into the grid hierarchy by associating the object with a set of grid cells that it touches (touched cells). Starting at level 1 of the grid hierarchy, the tessellation process proceeds breadth first across the level. Potentially, the process can continue through all four levels, one level at a time.

The output of the tessellation process is a set of touched cells that are recorded in the spatial index for the object. By referring to these recorded cells, the spatial index can locate the object in space relative to other
objects in the spatial column that are also stored in the index.

**E. Tessellation Schemes**

The behavior of a spatial index depends partly on its tessellation scheme. The tessellation scheme is data-type specific. In SQL Server 2008, spatial indexes support two tessellation schemes:

- Geometry grid tessellation, which is the scheme for the geometry data type.
- Geography grid tessellation, which applies to columns of the geography data type.

1) **Geometry Grid Tessellation Scheme**

Geometry grid tessellation is the default tessellation scheme for the geometry data type, and in SQL Server 2008, it is the only such tessellation scheme. This section discusses aspects of geometry grid tessellation that are relevant to working with spatial indexes: supported methods and bounding boxes.

2) **The Geography Grid Tessellation Scheme**

This tessellation scheme applies only to a geography column. This section summarizes the methods that are supported by geography grid tessellation and discusses how geodetic space is projected onto a plane, which is then decomposed into a grid hierarchy.

**F. Spatial Index Support**

A spatial index can be created only on a spatial column. You can create spatial indexes on any spatial column in a table that supports spatial indexes, and you can create multiple spatial indexes on a given spatial column.

**CONCLUSION**

We usually think of addresses as street, city, state, country, and ZIP code, but an address is a point on the earth and can also be represented as a latitude/longitude pair. Technically, an address can comprise a parcel of land that can be represented as a polygon, but let's stick to a single point in this case, just for simplicity. Knowing where a customer lives as latitude/longitude allows you to answer questions such as:

- What are the three closest bank branches for a specific customer?
- Which salesperson lives closest to the customer?
- How many customers does your company have within a 10 mile radius of, for instance, Seattle, Washington?
- How many customers live more than 2 miles from your nearest branch location?

The process of converting a street address to a latitude/longitude pair is called address geocoding. A number of online services (including MapPoint, Virtual Earth, and Google Earth) provide geocoding as a service. To convert a street address in the U.S. to a point, you can encapsulate a call to the MapPoint geocoding Web service in a SQLCLR function. This is stored in the Database as spatial data.

**REFERENCES**

[3] Professional Microsoft Sql Server 2008 Administration By Brian Knight, Ketan Patel, Ross Loforte, Wayne Snyder, Steven Wort