

White Paper



Spatial Data in SQL Server

Manoj Chandran Nair

Abstract

Even though there are information systems in the market that make use of spatial data, the adoption of its technical capabilities is fairly limited. Developers need to have specialized knowledge on handling spatial information to utilize the features provided by vendors. More recently database system providers like Oracle, IBM have widened the use of spatial techniques by providing the features as optional add-in. With the advent of spatial support in SQL Server 2008, Microsoft has reduced the obstacles to create spatially enabled applications.

Introduction

This paper will cover the spatial data features introduced as part of SQL Server 2008 suite starting with detailed description of spatial data and its representation, and then explaining the support provided for spatial data in the latest version of SQL Server. This paper assumes, the reader is familiar with the relational data concepts and SQL Server 2005 and 2008 databases. This paper will be useful for database architects, designers, and developers who want to implement spatial features into their applications or database systems.

What is Spatial Data?

Any data that describes the position, shape, and orientation of objects in space is known as spatial data. Mostly such data is used to describe places or objects on the earth and hence, sometimes it is also called geospatial data. Various objects can be represented using spatial data ranges like crop fields, boundary between countries, buildings and mountains.

In today's world, spatial information can be put to a variety of uses, such as:

- Analyzing sales trends in areas on a national or international level
- In GPS systems for deciding a path to a particular destination
- Locating fast food corners near to a given address
- Creating map reports to represent geographic information instead of simple tabular or chart reports

Before we delve into SQL related details, let's try to understand some basic concepts of spatial data and how to convert real-world objects into spatial information.

Mapping Features of Earth into Data

Although it is very difficult to exactly map objects on the earth (since most of them are very complex and have irregular shapes), spatial data uses shapes that will be close approximations of these objects. Such shapes are known as geometries.

SQL Server supports three basic types of geometries: Point, LineString, and Polygon. Any object can be represented using any one or a collection of these geometries.

Point

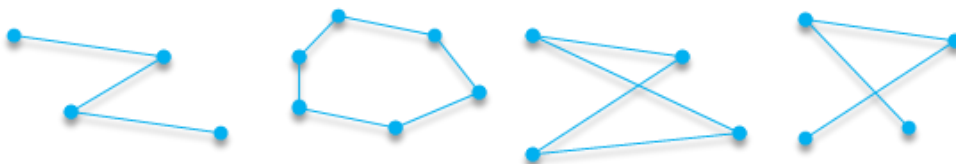
This is the most fundamental type of geometry without any length or area (0-dimension). It is used to define a position on the surface of the earth. For example, the location of office / building, street address, city etc.

LineString

A LineString can be defined as a collection of two or more points and the line segments that connect these points. It has a specified length but no area (1-dimension). LineString can be categorized as:

- Simple: No line segments connecting the points cross each other
- Closed: the start and the end points for the geometry are the same
- Ring: Both simple and closed

Note that even though it is closed, the LineString represents only the points and line segments and does not define the enclosed area. Below are some examples, which show simple, closed, non-simple and closed, non-simple LineString.



Polygon

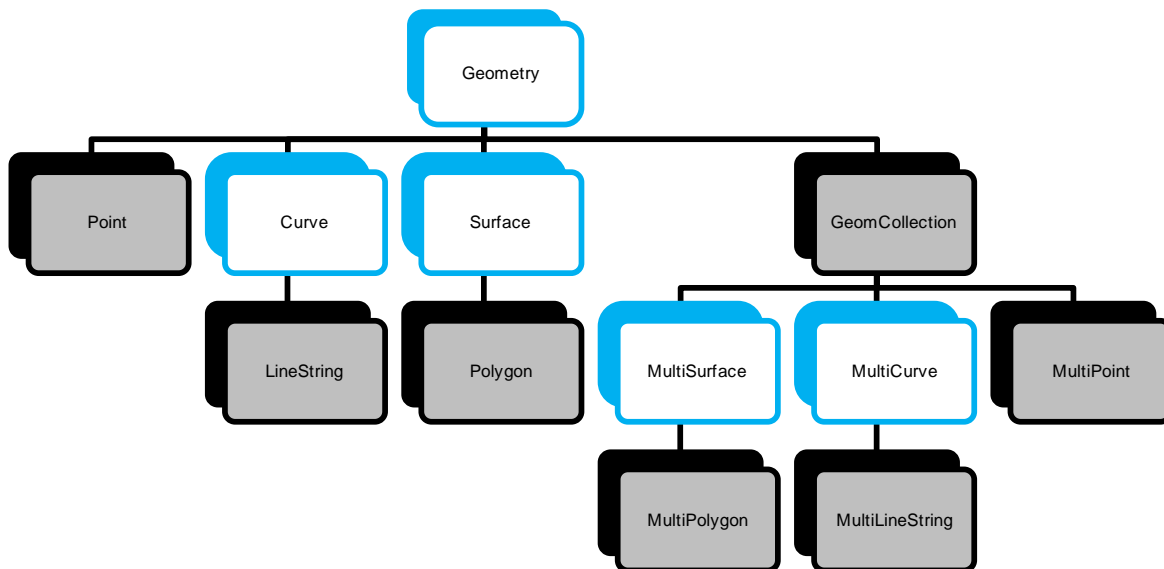
A Polygon is represented by the boundary of points (known as the exterior ring) that form the LineString object. In addition to the closed LineString, this geometry also contains the points that lie in the enclosed area of the LineString object. In other words, they have an associated length and area (2-dimension). A polygon object can contain interior rings referred as holes. When calculating the area of a polygon object, the area enclosed inside the holes will not be considered. Shown below are the examples of simple polygon objects and polygon with interior rings:



Polygons are used to represent geographical areas like stadiums, lakes, islands, states, etc.

The objects on the surface of the earth can be represented using the Point, LineString, and Polygon types. In some cases, it might be required to group such types together, say office buildings in the North campus (collection of points), or a society with a swimming pool (group of points and polygon objects), etc. Such collection is referred to as a *GeometryCollection*. Based on single type of geometry contained in the collection, it can be further categorized as *MultiPoint*, *MultiLineString*, and *MultiPolygon* objects.

The diagram below shows the hierarchy of the geometry objects. The highlighted ones (in grey) are the spatial data types that can be defined in SQL Server:



The *Curve* and the *Surface* types are part of the hierarchy as defined by OGC. Yet these cannot be instantiated in SQL Server as no support is provided in this version. But *LineString* and the *Polygon* types, which inherit properties of *Curve* and *Surface*, respectively can be instantiated with an additional condition that these types should contain straight line segments to join the points.

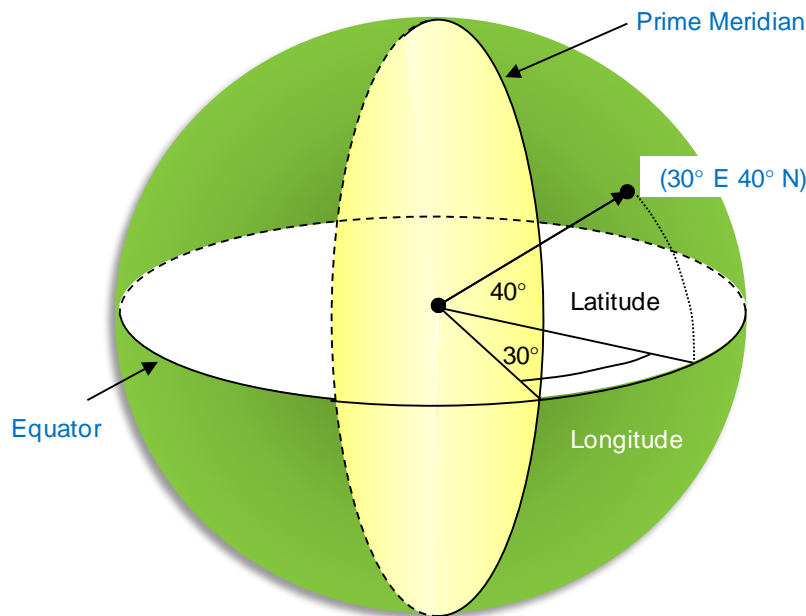
Positioning objects using coordinate systems

Once the objects are created, they need to be associated with a particular position on the surface of the earth. A system that is used to uniquely identify each point on a surface is known as spatial reference system. Usually a spatial reference system is based on a coordinate system. For example, x and y coordinates are used to define points while drawing graphs and each point is represented as (x, y). Similarly, a set of n-coordinates can be used to define a point in an n-dimensional system.

In SQL Server 2008, there are mainly two types of coordinate systems:

Geographic Coordinate System (GCS)

In this coordinate system, a point on the surface of the earth is defined using latitudes and longitudes. The diagram below represents a point using geographic coordinate system.



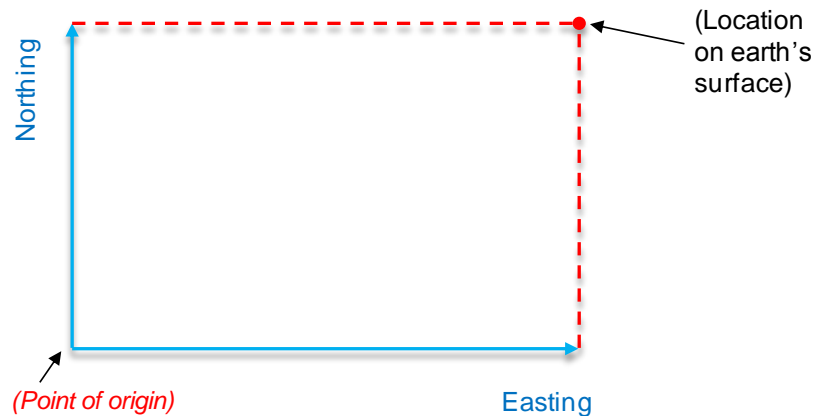
Latitude: This is measured as the angle from the equatorial plane to the line drawn from the center of the earth to the point on the surface. In the above diagram, it measures 40°.

Longitude: This is measured on the equatorial plane and is defined as the angle between the line drawn from the center of the earth to the point and the line drawn from center of the earth to the prime meridian. In the above diagram, it measures 30°.

Usually the longitude values range from 180° (to the east of prime meridian and suffixed with 'E') to -180° (to the west of prime meridian and suffixed with 'W') and the latitude values range from 90° (toward the North Pole and suffixed with 'N') to -90° (towards the South Pole and suffixed with 'S'). The point mentioned in the diagram above will be represented as 30°E 40°N.

Projected Coordinate System (PCS)

In contrast to geographical coordinate system, the projected coordinate system describes the position of points on earth's surface as measured on a two-dimensional plane. It uses Cartesian coordinates of x-axis and y-axis to define locations on the earth's surface. In PCS, the coordinate values are referred to as easting (x-coordinate) and northing (y-coordinate) measured from the point of origin.



Spatial Reference System and SRID

In previous section, there was a reference to the term “spatial reference system”, which uniquely identifies a particular point on the surface of the earth. Also, we identified two coordinate systems that will be used to measure the location of any point. But the coordinate systems alone are not enough for this purpose. For example, while measuring the location, we made use of prime meridian in GCS and point of origin, but these reference points or line segment could be located anywhere on the surface of the earth. Hence, we need to provide some additional information along with the coordinate system to uniquely identify any position on the earth, namely the *datum*, *prime meridian* and *unit of measurement*.

Datum

The actual shape of the earth is very complex since there are lot of mountains and valleys on the surface and hence it is very difficult to model this shape (known as geoid) accurately for conducting spatial analysis. For easy interpretation, the spatial system is usually based on an approximation of the geoid. This approximation is called the *reference ellipsoid*.

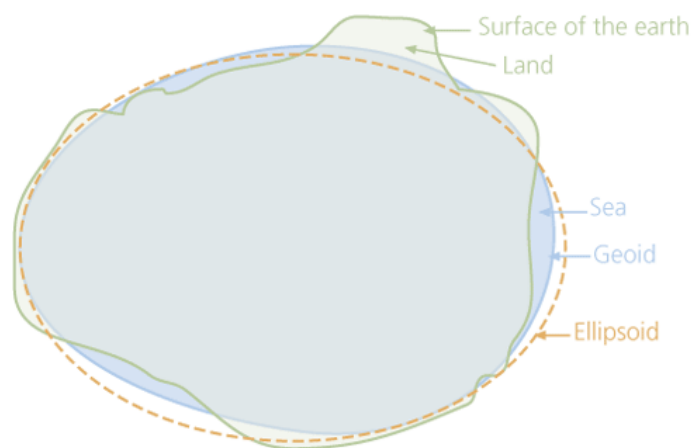


Figure: Geoid and Ellipsoid representations (Source: [ESRI](#))

There is various reference ellipsoids commonly used to approximate the shape of the geoid. Depending on the type of analysis, a reference ellipsoid should be selected. Selecting a particular reference ellipsoid will affect how well the coordinate system defines a particular position on the earth. Some of the commonly used reference ellipsoids are NAD 27 and NAD 83 (used in North America), Airy 1830 (used in Great Britain) and WGS 84 (Global).

Prime Meridian

As discussed in the earlier section, prime meridian is required to measure the longitude value for a particular position on the earth. Hence while defining a position, we need to mention what line segment will be used as the prime meridian based on which the coordinate values will be evaluated and this will be considered as the zero longitude. One of the commonly used prime meridians is the one that passes through Greenwich, London.

Unit of Measurement

Every spatial reference system should be associated with specific unit of measurement to measure the coordinate values. For example, to measure longitude and latitude values, degree or radian can be used. Also for projected coordinate system, a linear unit of measure like meter or foot can be used.

Projection

The Projection property specifies the type of projection to be used to define a position on the earth using projected coordinate system. Projection is a process of creating a two-dimensional representation of the earth. However, note that it is not possible to model the earth accurately as this process might affect shape, distance, or area of the earth, modeled as it is on a two-dimensional plane. No map projection exists that represent the earth correctly, though there are many trying to minimize the effect of the distortions created during projection. For example, while exploring the regions in Asia, we can use a projection that maximizes the accuracy near countries like India, China, etc. but reduces the accuracy near the countries located in North America or Africa. Some of the commonly used map projections are:

- Sinusoidal Projection
- Mercator Projection
- Equirectangular Projection
- Hammer Projection

For a complete list of map projections, refer the link available in the **References** section.

In SQL Server 2008, the details of all the available spatial reference systems are stored in a system table called *sys.spatial_reference_systems*, where each row uniquely identifies a single spatial reference system. The screenshot below shows the sample data present in the table:

spatial_reference_id	authority_name	authorized_spatial_reference_id	well_known_text	unit_of_measure	unit_conversion_factor
1	EPSG	4120	GEOGCS["Greek", DATUM["Greek", ELLIPSOID["Bessel 1841", 6377397.155, 299.1528128]], PRI...	metre	1
2	EPSG	4121	GEOGCS["GGRS87", DATUM["Greek Geodetic Reference System 1987", ELLIPSOID["GRS 1980"...	metre	1
3	EPSG	4122	GEOGCS["ATS77", DATUM["Average Terrestrial System 1977", ELLIPSOID["Average Terrestrial Sy...	metre	1
4	EPSG	4123	GEOGCS["KKJ", DATUM["Kartastokoordinaattijärjestelmä (1966)", ELLIPSOID["International 1924", ...	metre	1
5	EPSG	4124	GEOGCS["RT90", DATUM["Rikets koordinatsystem 1990", ELLIPSOID["Bessel 1841", 6377397.15...	metre	1
6	EPSG	4127	GEOGCS["Tete", DATUM["Tete", ELLIPSOID["Clarke 1866", 6378206.4, 294.978698213898]], PR...	metre	1
7	EPSG	4128	GEOGCS["Madzansua", DATUM["Madzansua", ELLIPSOID["Clarke 1866", 6378206.4, 294.97869...	metre	1

Let's try to understand how the coordinate system, prime meridian, and unit of measurement is stored for each spatial reference system in the above table by selecting one row as shown in the query below:

```
SELECT well_known_text
FROM sys.spatial_reference_systems
WHERE authority_name = 'EPSG'
AND authorized_spatial_reference_id = '4326'
```

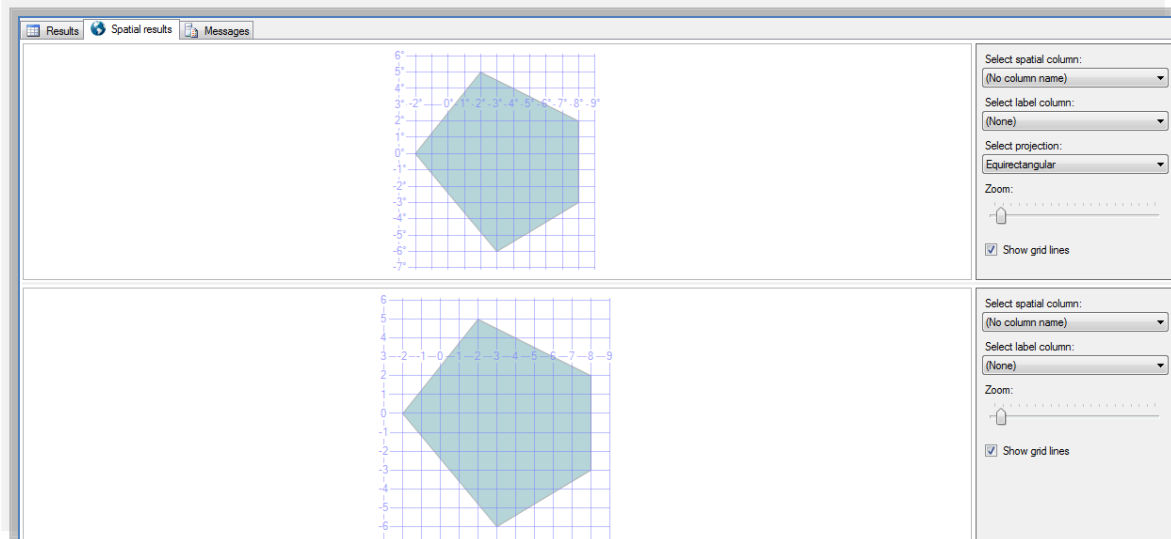
The result of the above query is shown below:

```
GEOGCS
["WGS 84",
  DATUM
  ["World Geodetic System 1984",
    ELLIPSOID
    ["WGS 84", 6378137, 298.257223563]
  ],
  PRIMEM
  ["Greenwich", 0],
  UNIT
  ["Degree", 0.0174532925199433]
]
```

Let's try to identify each component of a spatial reference system from above.

- *Coordinate system:* The first line mentions the coordinate system. In this case, GEOGCS stands for the geographical coordinate system (PROJCS for projected coordinate system). 'WGS 84' is the name of the spatial reference.
- *Datum:* It refers to the World Geodetic System and specifies what ellipsoid has been used to model the earth with semi-major axis of 6,378,137m and an inverse-flattening ratio of 298.257223563. For more details on the reference ellipsoid, please refer to the link in References section.
- *Prime Meridian:* Here it is 'Greenwich'.
- *Unit of measurement:* It is 'Degree' with conversion factor of 0.0174532925199433.

While defining any spatial data object, we need to mention the SRID (from the `spatial_reference_id` column), which will specify the spatial reference system to be used in conjunction with the coordinate values. WGS 84 is the most commonly used geodetic spatial reference system and it is referenced with ID 4326.



Some of the differences that we can note here are:

- Coordinates for the @geog variable are represented in degrees, i.e. latitude and longitude values whereas for @geom it is planar values (x and y coordinates)
- There is an option for selecting the projection to be used to represent the Polygon object defined in @geog variable. Available values are *Equirectangular*, *Mercator*, *Robinson* and *Bonne*. No such option is available for @geom variable.

For more details, check the link on Map Projections in the *References* section.

In both the declarations, we have used the same spatial reference ID (SRID) of 4326. Note that SRID can differ since SQL Server supports lot of spatial reference systems and we need to ensure that while doing spatial analysis, the spatial objects involved belong to the same reference system. While storing spatial objects in SQL Server tables, by default, there is no restriction on the SRID that can be used to represent these objects. So there is a possibility that you might come across an operation analogous to *15 US dollars + 20 GBP = ?* Usually SQL Server will produce NULL as output in such cases.

In order to avoid the above scenario, we can enforce a constraint on the spatial column to consider objects using the same spatial reference system. This can be done, as shown below, where we are set the constraint to use only SRID 4326 for Location column:

```
CREATE TABLE dbo.TempData
(
  ID int NOT NULL,
  Location geography NOT NULL)
ALTER TABLE TempData
ADD CONSTRAINT [enforce_srid_geogcolumn]
CHECK (Location.STSrid = 4326)
```

If we try to insert a spatial object with any other SRID, then it will produce an exception as shown below:

The INSERT statement conflicted with the CHECK constraint "enforce_srid_geogcolumn". The conflict occurred in database "AdventureWorks2008", table "dbo.TempData", column 'Location'

Size Limitation: While working with *geography data* type we need to be aware about the size limitation (There is no size limitation in case of a *geometry data* type). SQL Server has applied a limitation that no spatial object of geography data type

should exceed a single hemisphere. A hemisphere here does not mean the Northern or Southern hemisphere but one half of the earth model centered on any point on the surface of the earth. If at all this happens, SQL Server will produce an error message as shown below:

```
A .NET Framework error occurred during execution of user-defined routine or aggregate "geography":  
  
Microsoft.SqlServer.Types.GLArgumentException: 24205: The specified input does not represent a valid geography instance because it exceeds a single hemisphere. Each geography instance must fit inside a single hemisphere. A common reason for this error is that a polygon has the wrong ring orientation.  
  
Microsoft.SqlServer.Types.GLArgumentException:  
at Microsoft.SqlServer.Types.GLNativeMethods.ThrowExceptionForHr(GL_HResult errorCode)  
at Microsoft.SqlServer.Types.GLNativeMethods.GeodeticIsValid(GeoData g)  
at Microsoft.SqlServer.Types.SqlGeography.IsValidExpensive()  
at Microsoft.SqlServer.Types.SqlGeography.ConstructGeographyFromUserInput(GeoData g, Int32 srid)  
at Microsoft.SqlServer.Types.SqlGeography.GeographyFromText(OpenGisType type, SqlChars taggedText, Int32 srid)
```

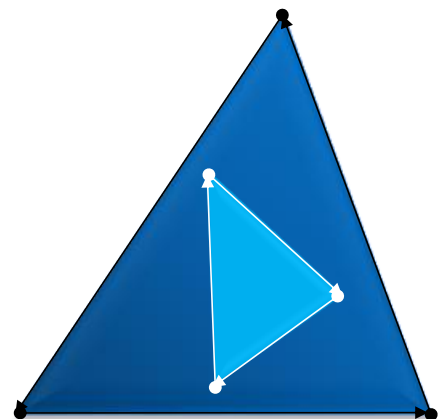
To work around this limitation, try to divide the spatial object into specific number of individual objects. For a country, try to create objects for the state and regions with the country. To represent the country, we can combine these small objects.

Ring Orientation: This is another concept one should be aware of for a geography data type. If you noticed, there is a phrase in the error message, above, which mentions about wrong ring orientation. This means that the order in which the points are considered in the polygon object makes the object exceed a single hemisphere.

Another issue is to identify which region is actually encompassed by these points. Since earth is represented by a geodetic model, we can start at any point on the surface, move forward, and reach back the same point. If the points representing the polygon object lies on the equator, then there will be an ambiguity as to whether the polygon represents the Northern hemisphere or the Southern hemisphere. Hence, we need to identify which side of the line segment joining these points represents the actual object. To resolve this ambiguity, SQL Server follows these rules:

- The line segment joining the points in a polygon should be drawn in an anti-clockwise direction and area to the left of the line segment should be considered.
- The line segment joining the points of an interior ring should be drawn in a clockwise direction and area to the left of the line segment should be considered.

The picture below indicates how the above two rules are applied for any object with interior rings. As per the rules, the points on the equator, discussed in the example above, will represent the Northern hemisphere.



The table below summarizes the properties related to geography and geometry data types:

Property	Geometry data type	Geography data type
Shape of the earth	Flat	Round
Coordinate system	Projected (planar)	Geographic
Coordinate values	Cartesian (x and y)	Latitude and Longitude
Unit of measurement	Same as coordinate values	Defined in sys.spatial_reference_systems
Spatial reference identifier	Not enforced	Enforced
Default SRID	0	4326 (WGS 84)
Size Limitation	None	No object may occupy more than one hemisphere
Ring Orientation	Not significant	Significant

Methods for Creating Spatial Data

For creating any type of spatial objects, we need to make use of static methods provided in SQL Server. One such method is the STGeomFromText, which helps to create a polygon object in the previous section. Similarly, various other static methods are available and categorized based on the type of object that needs to be created. The syntax for creating a spatial object is shown below:

```
datatype::method('objectdefinition',SRID)
```

- Datatype will be geography or geometry.
- Method will be decided based on the type of object to be created and the spatial object definition. There are three formats in which an object can be defined; Well-Known Text (WKT), Well-Known Binary (WKB), and Geography Markup Language (GML).

For example, to create a Point object with definition in text format, we have to use the STPointFromText method and it will be created as follows:

```
DECLARE @geog geography  
  
SET @geog =geography::STPointFromText('POINT(2 3)', 4326)  
  
DECLARE @geom geometry  
  
SET @geom =geometry::STPointFromText('POINT(2 3)', 4326)
```

Let's check out on how to create a LineString object in all the three formats and store it in a geography variable:

Well-Known Text

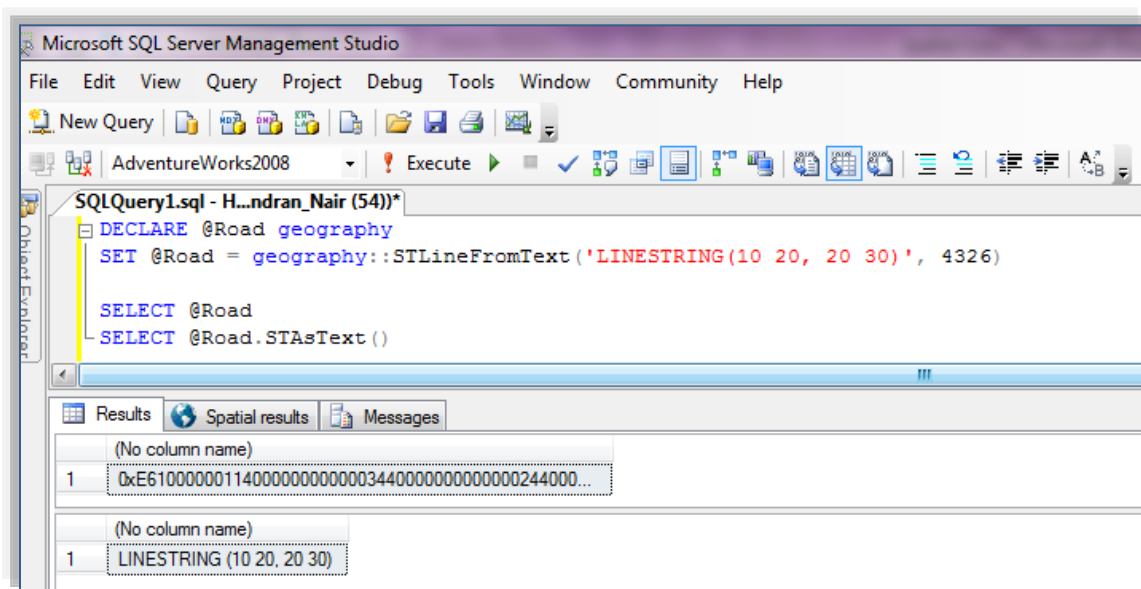
WKT is a commonly used format to demonstrate and share spatial data among users due to its readability and conciseness. However, there are some drawbacks with WKT, like:

- Representing floating point coordinates with absolute precision not possible due to rounding issues.
- Method execution is slower than the methods for other formats.

The example below shows how to create LineString object using WKT:

```
DECLARE @Road geography
SET @Road = geography::STLineFromText ('LINESTRING(10 20, 20 30)', 4326)
```

Below is the output:



Based on the type of object and the object definition, the method used here is `STLineFromText`. The values are always stored in binary format inside the `geography` variable. SQL Server provides a method `STAsText` using which the value can be displayed in text format.

Well-Known Binary

WKB is another standardized way of representing spatial data where the object is represented as contiguous stream of bytes in binary format. It is better than WKT and GML formats when it comes to performance and representing floating point coordinates with accurate precision. The only drawback is the format is difficult to understand for a human reader and would be hard to detect errors.

Listed below are some of the terms that we need to be aware of when defining a spatial object in binary format.

- *ByteOrder*: This indicates whether the rest of the bytes are arranged using big-endian (0x00) or little endian (0x01) byte order. For details on *Endianness*, refer the link in the References section. But to tell you the difference in short, a little-endian value 0x1234 is same as the big-endian value 0x3412 (not 0x4321) where each byte is represented by two hexadecimal characters

- **Type:** This indicates the type of geometry to be defined. Values are 1-Point, 2-LineString, 3-Polygon, 4-Multipoint, 5-MultiLineString, 6-MultiPolygon and 7- *GeometryCollection*.
- **NumberOfGeometries:** This indicates the number of geometries involved in the definition. For Point object, this number is not applicable. For LineString object, it will denote the number of points, for Polygon object, it denotes the number of rings (exterior or interior) and for Multi-element geometry it denotes the number of different elements in the collection.

Shown below are the syntaxes for each of the objects:

Point

```
[ByteOrder] [Type] [X] [Y]
```

LineString

```
[ByteOrder] [Type] [NumberOfGeometries] [X1] [Y1] [X2] [Y2]..... [Xn] [Yn]
```

Polygon

```
[ByteOrder] [Type] [NumberOfGeometries]
```

```
[NumPoints] [X1] [Y1]-[Xn] [Yn] [NumPoints] [X1] [Y1]-[Xn] [Yn]
```



Ring1

Ring2

Multielement Geometry

```
[ByteOrder][Type][NumGeometries]<Geometry1><Geometry2>.....<GeometryN>
```

Coming back to our example, we can define a LineString object as follows:

```
DECLARE @ByteOrder bit
DECLARE @Type int
DECLARE @NumofGeometries int
DECLARE @X1 float
DECLARE @Y1 float
DECLARE @X2 float
DECLARE @Y2 float
SET @ByteOrder = 0
SET @Type = 2
SET @NumofGeometries = 2
SET @X1 = 10
SET @Y1 = 20
SET @X2 = 20
SET @Y2 = 30
DECLARE @WKB varbinary(max)
```

```

DECLARE @Road geography

SET @WKB =

    CAST(@ByteOrder AS binary(1))

    +CAST(@Type AS binary(4))

    +CAST(@NumofGeometries AS binary(4))

    +CAST(@X1 AS binary(8))

    +CAST(@Y1 AS binary(8))

    +CAST(@X2 AS binary(8))

    +CAST(@Y2 AS binary(8))

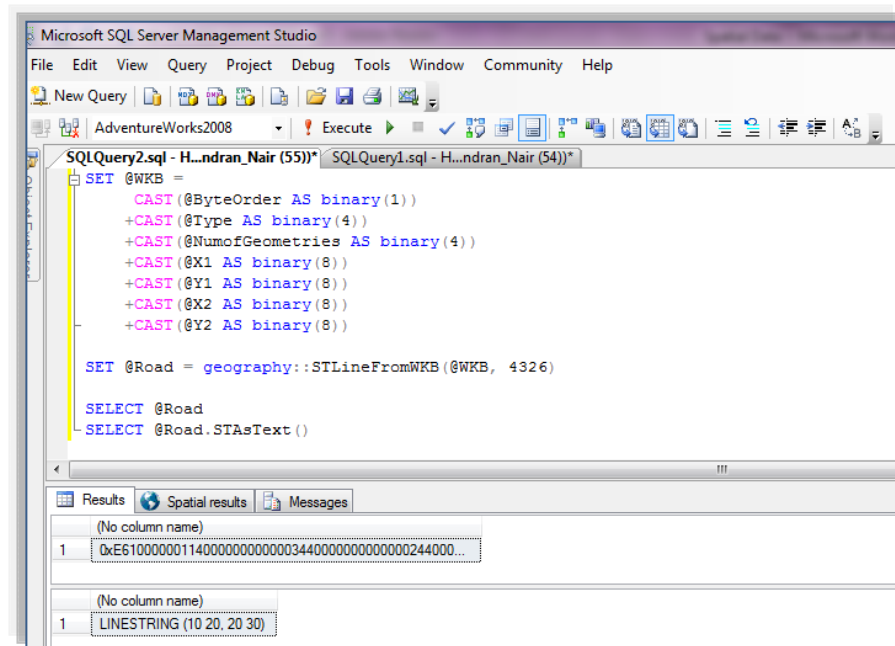
SET @Road =geography::STLineFromWKB(@WKB, 4326)

SELECT @Road

SELECT @Road.STAsText()

```

The output of the above code will be similar to the WKT sample code.



Geography Markup Language

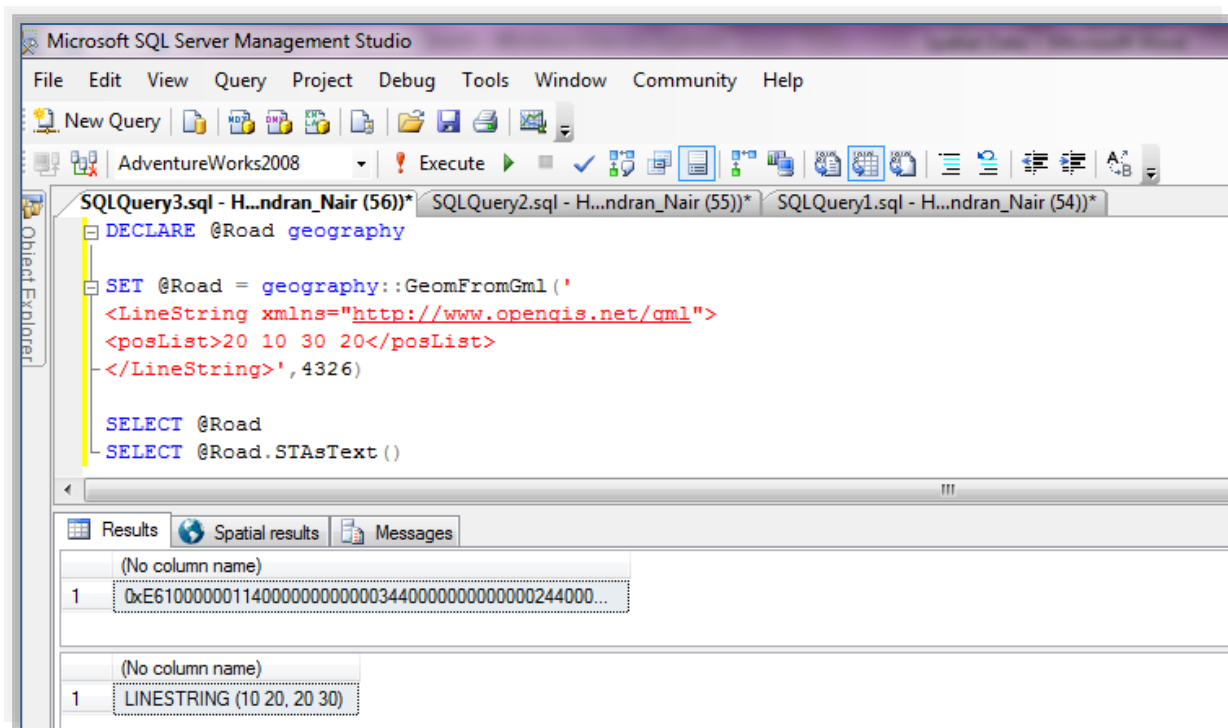
GML is an XML-based markup language for representing spatial information. Each property of the element is contained within specific element tags within the document structure. For example, a point is represented as follows:

```
<Point xmlns="http://www.opengis.net/gml">  
  
<pos>10 30</pos>  
  
</Point>
```

Similar WKT, GML is relatively easy to interpret and understand complex objects. However, like WKT, we have to deal with the precision issues. It requires more space than an equivalent WKT representation. Also unlike WKT and WKB, there is only one method *GeomFromGML*, which can be used to create any type of object.

For creating a LineString object, you need to define the object as follows:

```
DECLARE @Road geography  
  
SET @Road = geography::GeomFromGml ('  
  
<LineString xmlns="http://www.opengis.net/gml">  
  
<posList>20 10 30 20</posList>  
  
</LineString>', 4326)  
  
SELECT @Road  
  
SELECT @Road.STAsText ()
```



The output is illustrated below and same as the previous two examples:

One difference to note here is the order in which the coordinates have been defined. The first point (10, 20) has been defined as (20 10) in *GeomFromGML* method. This is because in case of geography variables, the method accepts the coordinates in latitude-longitude order, whereas the *STAsText* method interprets in longitude-latitude order. So in order create an equivalent LineString object as in WKT and WKB examples, the coordinates should be specified as (20 10 30 20) instead of (10 20 20 30).

Based on one's preference, one can go for any of the above three formats to represent the spatial objects. For better analysis, one can convert the object definition into text format using the *STAsText* method. Additionally, there is *STAsBinary* and *AsGml* methods to view the definition in WKB and GML formats.

For information on how to create instances of other types refer the links in the **References** section.

Working with Spatial Data

Once the spatial objects have been created, next step is to analyze and query these data. Though the methods currently being used for other data types cannot be applied on spatial instances, SQL Server provides new methods to work with geography and geometry data types. Not all methods work for both data types:

For easy understanding, let's categorize into three groups:

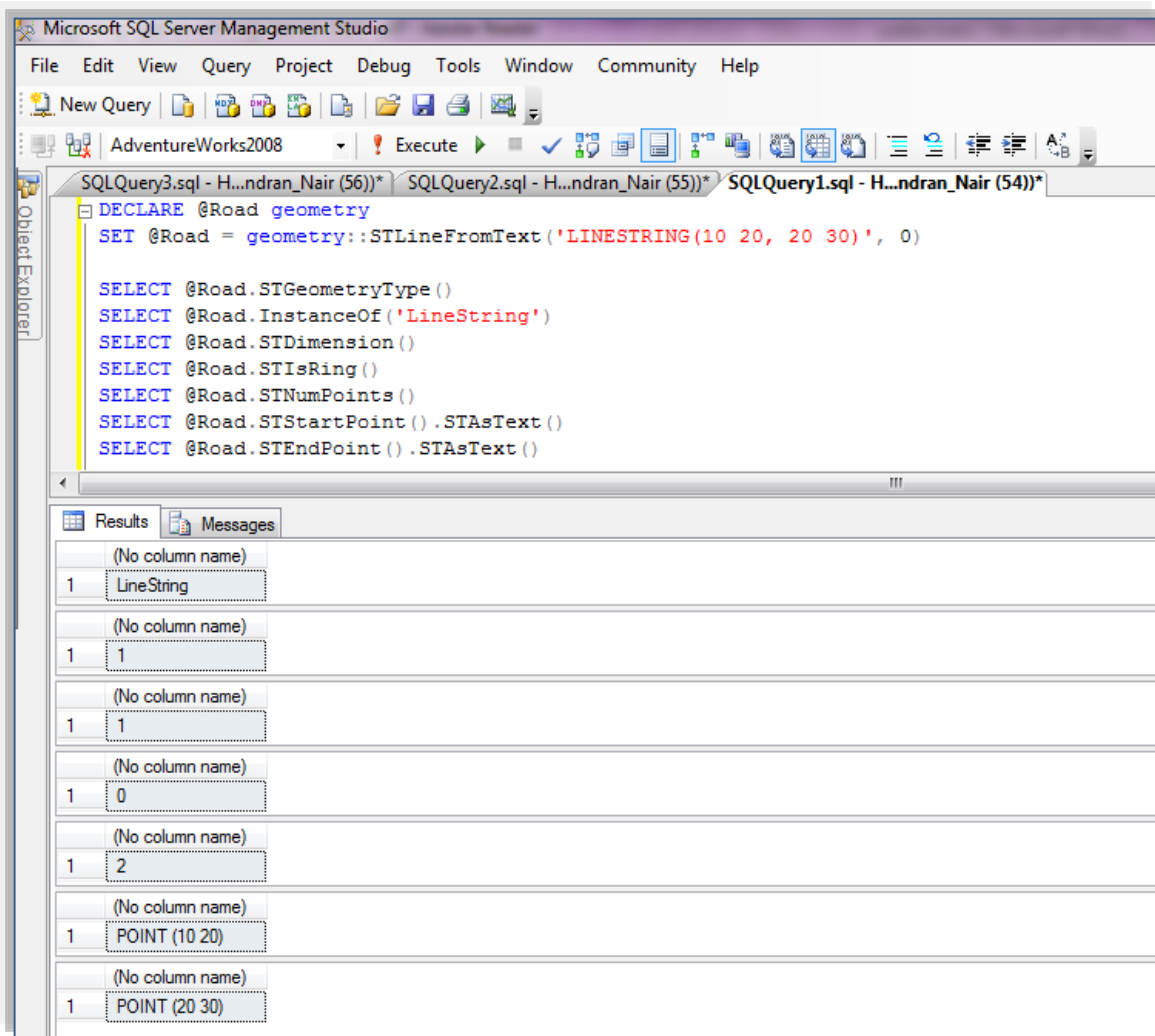
- Description of spatial objects
- Creating new objects for existing objects
- Finding relationships between spatial objects

Description of spatial objects

Static methods are available in SQL Server 2008 for finding out the properties about a particular spatial object. Some of them are listed in the table below:

Method	Description
STGeometryType	Returns the name of the type of the geometry
InstanceOf	Checks whether the object is of a particular instance
STDimension	Returns the number of dimensions on the object
STIsRing	Checks whether the object is a ring (only for geometry)
STNumPoints	Returns the number of points in the object
STStartPoint	Returns first point in the object definition
STEndPoint	Returns last point in the object definition

The output of these methods when executed for LineString object defined below:



Creating new objects for existing objects

These methods can be used to create new spatial objects by combining and modifying existing spatial objects. Let's check out some of the methods.

MakeValid

This method can be applied to make a particular geometry conform to OGC standards. This method will convert the existing geometry into one or more valid geometry types. Below is an example where this method is being used:

```
DECLARE @geom geometry

SET @geom = geometry::STLineFromText('LINESTRING(0 0, 2 0, 4 3, 2 0, 6 0)', 0)

SELECT @geom.STAsText()

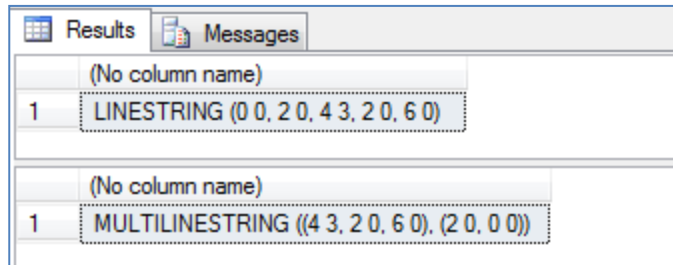
DECLARE @geom1 geometry

SET @geom1 = @geom.MakeValid()

SELECT @geom1.STAsText()
```

The output of the above code will be:

As shown in the above example, the original LineString object had a spike at (2 0, 4 3, 2 0). On applying the MakeValid method, this object gets converted into a MultiLineString object with two LineString objects (4 3, 2 0, 6 0) and (2 0, 0 0).



	(No column name)
1	LINESTRING (0 0, 2 0, 4 3, 2 0, 6 0)
	(No column name)
1	MULTILINESTRING ((4 3, 2 0, 6 0), (2 0, 0 0))

STUnion

This method is used to create a union of two spatial instances of geography or geometry data types.

Example:

```
DECLARE @geom1 geometry, @geom2 geometry, @geomUnion geometry

SET @geom1 =geometry::STPolyFromText('POLYGON((2 0, 0 2, 2 4, 4 4, 6 2, 4 0, 2 0))',
0)

SELECT @geom1

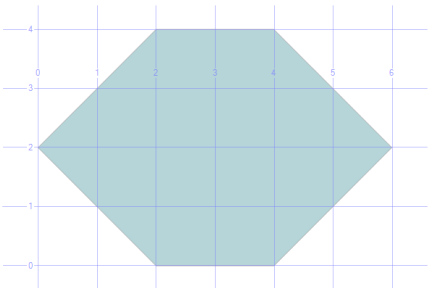
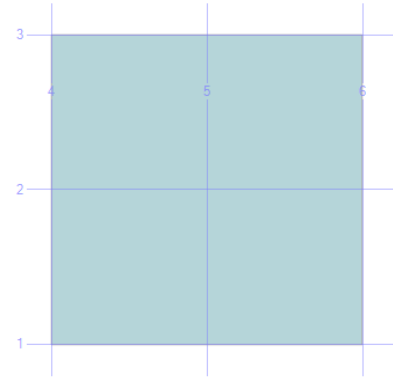
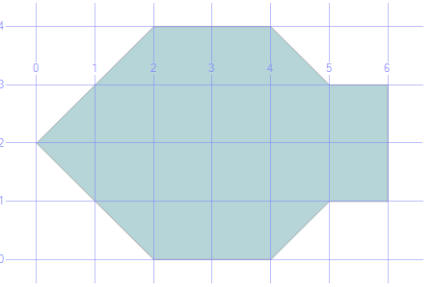
SET @geom2 =geometry::STPolyFromText('POLYGON((4 1, 4 3, 6 3, 6 1, 4 1))', 0)

SELECT @geom2

SET @geomUnion = @geom1.STUnion(@geom2)

SELECT @geomUnion
```

The table below shows the Spatial tab results for the above code:

@geom1	
@geom2	
@geomUnion	

STIntersection

This method creates a new spatial object from the points that are common between the two spatial objects. The output can be a single object like a Point, LineString or Polygon or a *Multielement* geometry.

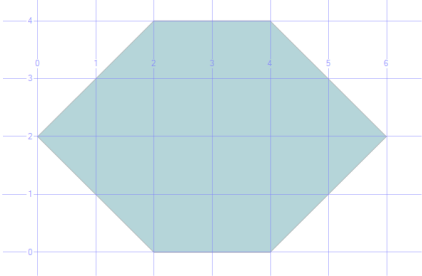
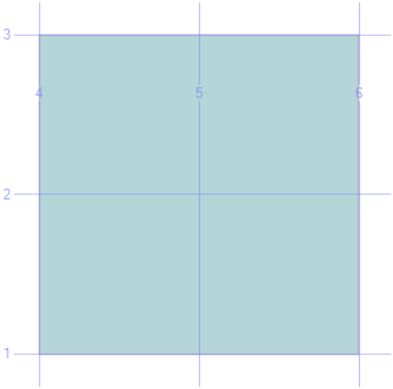
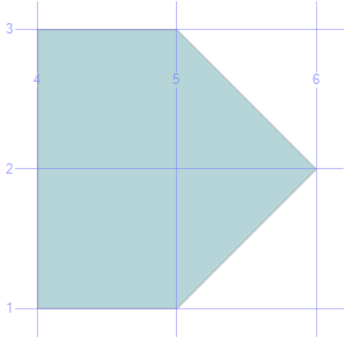
Example:

```
DECLARE @geom1 geometry,@geom2 geometry,@geomInter geometry
SET @geom1 =geometry::STPolyFromText('POLYGON((2 0, 0 2, 2 4, 4 4, 6 2, 4 0, 2 0))', 0)
SELECT @geom1

SET @geom2 =geometry::STPolyFromText('POLYGON((4 1, 4 3, 6 3, 6 1, 4 1))', 0)
SELECT @geom2
```

```
SET @geomInter = @geom1.STIntersection(@geom2)
SELECT @geomInter
```

The table below shows the Spatial tab results for the above code:

<p>@geom1</p>	
<p>@geom2</p>	
<p>@geomInter</p>	

STDifference

This method returns the set of points that are present in one spatial object but not in other. Say in the syntax Object1.STDifference (Object2), it will return the points present in Object1 but not present in Object2.

Example:

```
DECLARE @geom1 geometry,@geom2 geometry,@geomDiff geometry

SET @geom1 =geometry::STPolyFromText('POLYGON((2 0, 0 2, 2 4, 4 4, 6 2, 4 0, 2 0))', 0)

SELECT @geom1

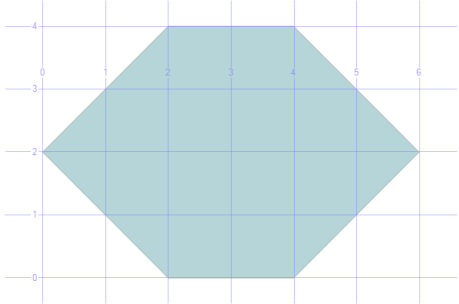
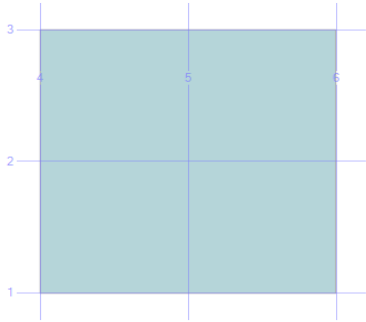
SET @geom2 =geometry::STPolyFromText('POLYGON((4 1, 4 3, 6 3, 6 1, 4 1))', 0)

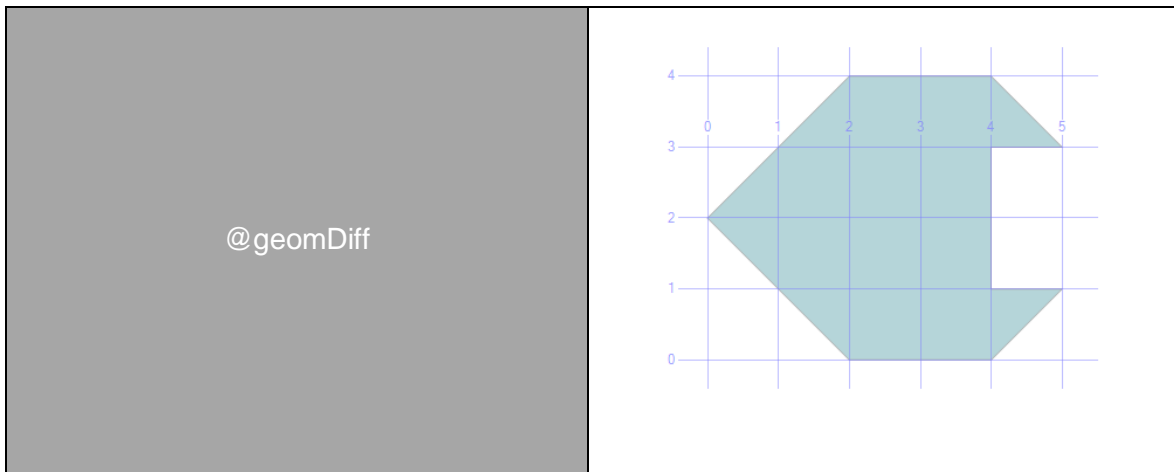
SELECT @geom2

SET @geomDiff = @geom1.STDifference (@geom2)

SELECT @geomDiff
```

The table below shows the Spatial tab results for the above code:

@geom1	
@geom2	



Finding relationships between spatial objects

These methods can be used to find the relation between various spatial objects and mostly used for querying and analysis. Using these methods, we could answer questions such as, 'How far Point A is from Point B?' or 'Does the route from X to Y pass through Z?'

Mentioned below are some of the methods in this group:

STEquals

This method checks whether both the spatial objects involved are equal, i.e. both contain the same set of points. It returns **1** if True and **0** if False.

STDistance

This method will return the shortest distance between the points in the two geometries involved. For a geometry data type, it will return the length of the shortest line that can be drawn between the two objects; and for a geography data type, it will return the length of the shortest arc that can be drawn on the geodetic model between the two objects.

STIntersects

This method checks whether the two geometries intersect each other i.e. if they have at least one point in common whether on the boundary or within the object. Returns **1**, if True and **0** is False. For finding out the area of intersection, we need to use the STIntersection method.

STTouches

This method checks whether the two geometries touch other, i.e. they should have at least one point in common and none of them should be an interior point. Returns **1** if True and **0** is False.

STWithin

This method checks whether on spatial object is contained within another spatial object.

Returns **1** if True and **0** is False.

Example showing how to use the above-mentioned methods:

```
DECLARE @geom1 geometry,@geom2 geometry

SET @geom1 =geometry::STPolyFromText('POLYGON((2 0, 0 2, 2 4, 4 4, 6 2, 4 0, 2 0))', 0)

SET @geom2 =geometry::STLineFromText('LINESTRING(4 1, 2 3)', 0)

SELECT @geom1.STEquals(@geom2)

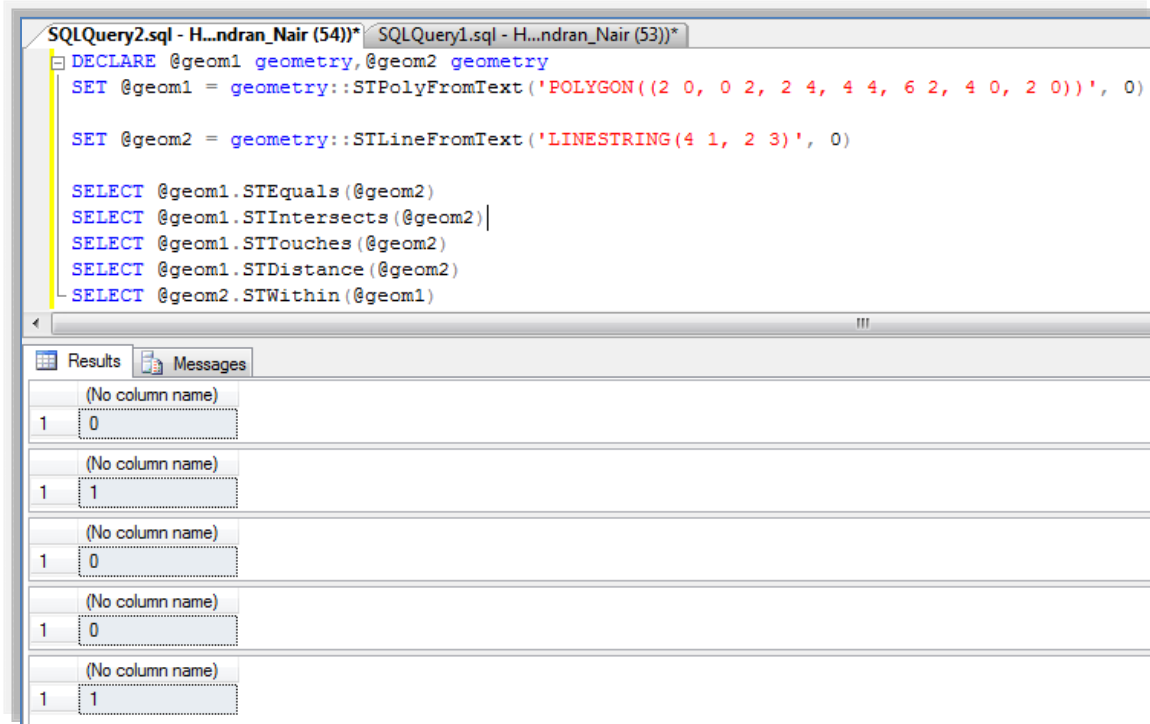
SELECT @geom1.STIntersects(@geom2)

SELECT @geom1.STTouches(@geom2)

SELECT @geom1.STDistance(@geom2)

SELECT @geom2.STWithin(@geom1)
```

The output of the code is shown in the screenshot below:



For exploring other OGC methods, refer the links available in the **References** section.

Spatial Indexing

Indexing is essentially used for speeding up the calculations while querying data. SQL Server provides options to create indexes (clustered or non-clustered) on existing data types like integer and *varchar* etc. In SQL Server 2008, an option to create indexes exists on geography and geometry data types. Such an index is called a spatial index.

The basic of idea on which the spatial index operates is filtering out the data which are not necessary and narrowing down the existing result set. When methods like *STIntersects* or *STDistance* are called, internally SQL Server evaluates each and every point in the geometry to arrive at a solution. This would mean performing thousands of calculations which would require significant amount of processing power. To avoid this, the number of points involved in the calculation should be reduced.

SQL Server follows a two filter process, where the primary filter will select the potential records that are relevant for the calculation. This set will contain the actual result set. This is then passed to the secondary filter which does the expensive operation of finding the exact solution. The spatial index will be used in the primary filter process to narrow down the number of points that will be worked upon by the secondary filter.

For indexing spatial data types, a grid system is defined to which the geometry will be mapped. The grid system can of any type; LOW (a 4X4 grid), MEDIUM (an 8X8 grid) or HIGH (a 16X16 grid). All the points of the geometry object will intersect any one of the cells in the grid system.

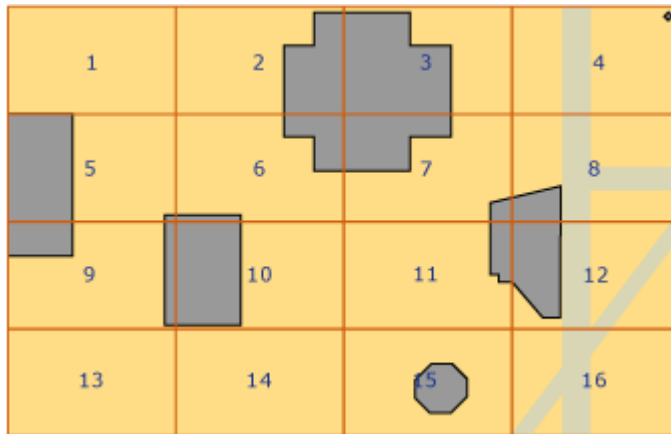


Figure: 4X4 grid representation (source: MSDN)

Once this is done, only the cells that contain the points of the geometry object will be considered for further calculations. These cell addresses will be stored as part of the index data. Also for more accuracy, the cells might need to be further divided as in the case of cell 15 where the geometry is occupying very little space. SQL Server provides the options of defining levels while creating a spatial index. Each level will contain a grid system for a cell in the previous level. Up to four levels can be created and each level can either be LOW, MEDIUM or HIGH.

Though defining levels will help in approximating the geometry object more accurately, this will in turn increase the number of cell addresses to be included in the index definition thus increasing the size of the index. To address this issue, while creating a spatial index, SQL Server applies three rules as mentioned below:

- *Covering rule:* If a cell is completely covered by the geometry object, no need to divide it further.
- *Deepest Cell Rule:* When a partially covered cell is subdivided, only the cells at the deepest level grid should be added to the index.
- *Cells-per-object rule:* This rule states that we can limit the number of cells used to define the object using the `CELLS_PER_OBJECT` parameter while creating the spatial index. If this limit exceeds, then the cell won't be subdivided.

These rules help prevent the need to include each and every cell in the index but only those which can approximate the object precisely.

For geography data types, we need to apply the same grid system but first have to project the object onto a flat plane using the steps below:

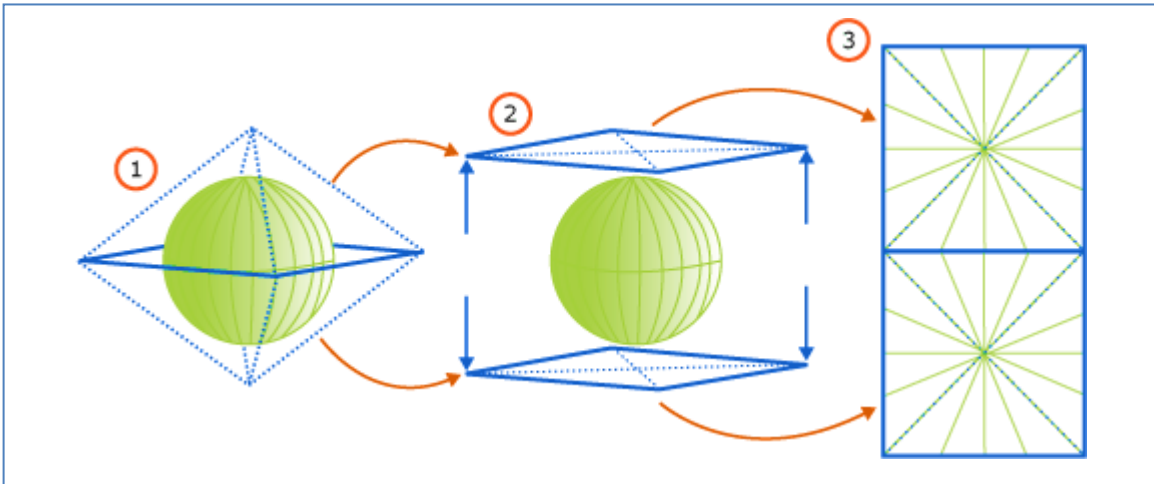


Figure: Steps to project geography object into flat plane (source: MSDN)

- Place two quadrilateral pyramids over the ends of a model of the earth.
- The features of each hemisphere are projected onto the sides of the appropriate pyramid.
- The pyramids are then flattened to form a single projected image.

Creating Indexes

Let's check out how indexes for geometry or a geography data type are created. The concepts above will help to understand it better.

Geometry

```
CREATE SPATIAL INDEX idxGeom on Stadiums(location)
USING GEOMETRY_GRID
WITH (
    BOUNDING_BOX = (-180,-90, 180, 90),
    GRIDS =(
        LEVEL_1 = MEDIUM,
        LEVEL_2 = MEDIUM,
        LEVEL_3 = MEDIUM,
        LEVEL_4 = MEDIUM),
    CELLS_PER_OBJECT = 16
)
```

Geography

```
CREATE SPATIAL INDEX idxGeog on Stadiums(location)
USING GEOGRAPHY_GRID
WITH (
    BOUNDING_BOX = (-180, -90, 180, 90),
    GRIDS = (
        LEVEL_1 = MEDIUM,
        LEVEL_2 = MEDIUM,
        LEVEL_3 = MEDIUM,
        LEVEL_4 = MEDIUM),
    CELLS_PER_OBJECT = 16
)
```

Here, the USING statement specifies which grid system to be used based on the data type. BOUNDING_BOX specifies the extent of data to be indexed; only the data fitting inside this area will be used for indexing. GRIDS property defines the resolution used at each level of the multi-grid, default being MEDIUM. CELLS_PER_OBJECT as discussed above defines the limit for the number of cells used to define the geometry, default value being 16.

For more details, check the link on spatial indexing in the **References** section.

Summary

SQL Server 2008 provides data types that can be used to represent the geographical features on the surface of the earth with conformance to OGC standards. These data types can support both planar data (geometry data type) and ellipsoidal (geography data type) data. It uses different geometric shapes (Point, LineString, and Polygon) to store data about a location, route or a particular area situated anywhere on the earth. Once the data is stored, there are OGC methods available to conduct analysis on the spatial information by modifying the spatial data or using relationships between spatial objects. Also like other SQL data types, indexing can be applied on spatial data types for improving performance while comparing spatial objects using OGC methods.

References

Open Geospatial Consortium

- <http://www.opengeospatial.org>

Map Projections

- http://en.wikipedia.org/wiki/List_of_map_projections

Reference Ellipsoid

- http://en.wikipedia.org/wiki/Reference_ellipsoid

Endianness

- <http://en.wikipedia.org/wiki/Endianness>

Creating geometry and geography instances

- <http://msdn.microsoft.com/en-us/library/bb895235.aspx>
- <http://msdn.microsoft.com/en-us/library/bb895335.aspx>

OGC Methods

- [http://msdn.microsoft.com/en-us/library/bb933960\(v=SQL.105\).aspx](http://msdn.microsoft.com/en-us/library/bb933960(v=SQL.105).aspx)
- [http://msdn.microsoft.com/en-us/library/bb933917\(v=SQL.105\).aspx](http://msdn.microsoft.com/en-us/library/bb933917(v=SQL.105).aspx)

Spatial Indexing

- <http://technet.microsoft.com/en-us/library/bb964712.aspx>

About the author

Manoj Chandran Nair

Technology Architect, Microsoft Technology Center, Infosys Technologies

Manoj has more than seven years of IT experience. He has worked on various Microsoft Technologies like ASP.NET, Windows, SQL Server and its Business Intelligence aspects like Analysis Services, Reporting Services. He was involved in development and support activities for reporting solutions using SSRS and finance management solutions for Microsoft using PerformancePoint Server 2007. Currently he is exploring latest features in SQL Server 2008 related to cloud computing, master data management and geospatial applications. He blogs on the mentioned areas and other database topics related to SQL Server.

Acknowledgement

The author would like to acknowledge the contribution of *Naveen S Kumar* (Principle Architect, MTC), *Virendra Wadekar* (Senior Technology Architect, MTC) and *Phaneendra Subnvis* (Technology Architect, MTC) for their support and guidance to the paper and extending critical inputs to achieve the current structure.



Infosys among the world's top 50 most respected companies

Reputation Institute's Global Reputation Pulse 2009 ranked Infosys among the world's top 50 most respected companies.



About Infosys

Many of the world's most successful organizations rely on Infosys to deliver measurable business value. Infosys provides business consulting, technology, engineering and outsourcing services to help clients in over 30 countries build tomorrow's enterprise.

For more information about Infosys (NASDAQ:INFY), visit www.infosys.com.

Global presence

Americas

[Brazil](#)
[Nova Lima](#)
[Canada](#)
[Calgary](#)
[Montreal](#)
[Toronto](#)
[Mexico](#)
[Monterrey](#)
[United States](#)
[Atlanta](#)
[Bellevue](#)
[Bentonville](#)
[Bridgewater](#)
[Charlotte](#)
[Detroit](#)
[Fremont](#)
[Hartford](#)
[Houston](#)
[Lake Forest](#)
[Lisle](#)
[New York](#)
[Phoenix](#)
[Plano](#)
[Quincy](#)
[Reston](#)

Asia Pacific

[Australia](#)
[Brisbane](#)
[Melbourne](#)
[Perth](#)
[Sydney](#)
[China](#)
[Shanghai](#)
[Hangzhou](#)
[Hong Kong](#)
[Central](#)
[India](#)
[Bangalore](#)
[Bhubaneshwar](#)
[Chandigarh](#)
[Chennai](#)
[Gurgaon](#)
[Hyderabad](#)
[Jaipur](#)
[Mangalore](#)
[Mumbai](#)
[Mysore](#)
[New Delhi](#)
[Pune](#)
[Thiruvananthapuram](#)
[Japan](#)
[Tokyo](#)
[New Zealand](#)
[Wellington](#)
[Philippines](#)
[Metro Manila](#)
[Singapore](#)
[Singapore](#)

Europe

[Belgium](#)
[Brussels](#)
[Czech Republic](#)
[Brno](#)
[Prague](#)
[Denmark](#)
[Copenhagen](#)
[Finland](#)
[Helsinki](#)
[France](#)
[Paris](#)
[Germany](#)
[Frankfurt](#)
[Stuttgart](#)
[Walldorf](#)
[Ireland](#)
[Dublin](#)
[Italy](#)
[Milano](#)
[Norway](#)
[Oslo](#)
[Poland](#)
[Lodz](#)
[Spain](#)
[Madrid](#)
[Sweden](#)
[Stockholm](#)
[Switzerland](#)
[Geneva](#)
[Zurich](#)
[The Netherlands](#)
[Amsterdam](#)
[United Kingdom \(UK\)](#)
[London](#)

Middle East and Africa

[Kingdom of Saudi Arabia](#)
[Riyadh](#)
[Mauritius](#)
[Reduit](#)
[UAE](#)
[Dubai](#)
[Sharjah](#)

For more information, contact askus@infosys.com

www.infosys.com

© 2011 Infosys Technologies Limited, Bangalore, India. Infosys believes the information in this publication is accurate as of its publication date; such information is subject to change without notice. Infosys acknowledges the proprietary rights of the trademarks and product names of other companies mentioned in this document.