CHAPTER 11

Working with Spatial Data

In Chapter 10, I discussed the concepts associated with spatial data, which SQL Server implements using the GEOMETRY and GEOGRAPHY data types. In this chapter, I will examine how to work with these data types. First, I will discuss the methods that can be used to construct surfaces, before reviewing how to query spatial data. Finally, you will see how to design and create spatial indexes.

Caution A geospatial object can be referred to as a geometry, with simple objects referred to as primitive geometries and collections of objects referred to as geometry collections. Because geometry is also the name of the data type that implements spatial data as a flat-earth model, this can cause confusion. Therefore, please note that when this chapter refers to a geospatial object, the word *geometry* will be used in lowercase. When referring to the data type, the uppercase word GEOMETRY will be used.

P. A. Carter, *SQL Server Advanced Data Types*,
https://doi.org/10.1007/978-1-4842-3901-8_11

Constructing Spatial Data

The GEOMETRY and GEOGRAPHY data types expose a number of methods that can be used to interact with spatial data. Many of these methods form part of Open Geospatial Consortium (OGC) standards, while others are an extension of that standard. Table 11-1 details the methods that can be used for constructing geometries, all of which are from OGC specifications and are exposed through both GEOMETRY and GEOGRAPHY.

Table 11-1. *Methods for Constructing Geometries*

Spatial Type	From WKT	From WKB
Point	STPointFromText	STPointFromWKB
LineString	STLineFromText	STLineFromWKB
Polygon	STPolyFromText	STPolyFromWKB
Any Primitive Spatial Instance	STGeomFromText	STGeomFromWKB
MultiPoint	STMPointFromText	STMPointFromWKB
MultiLineString	STMLineFromText	STMLineFromWKB
MultiPolygon	STMPolyFromText	STMPolyFromWKB
Any Multi Spatial Instance	STMGeomCollFromText	STMGeomCollFromWKB

Each of these methods accepts two parameters. The first parameter is the well-known text (WKT) or well-known binary (WKB) of the spatial instance. The second parameter is the SRID (please refer to Chapter 10, for further details on SRIDs) that the spatial instance should use. When called against a column or variable of type GEOMETRY, 0 can be passed as the SRID, as map projections are not required in a Euclidean model (see Chapter 10). When called against a column or variable of type GEOGRAPHY, however, a valid SRID must be used, and passing 0 will result in an error being thrown by the .NET framework.

The script in Listing 11-1 will create a LineString in a variable of type GEOMETRY, by using well-known-text.

Listing 11-1. Creating a LineString with Well-Known Text

```
DECLARE @LineString GEOMETRY ;

SET @LineString = GEOMETRY::STLineFromText('LINESTRING(0 0, 4.5
5, 4.5 0, 0 0)', 0)

SELECT @LineString ;
```

The graphical results of this script can be found in Figure 11-1.

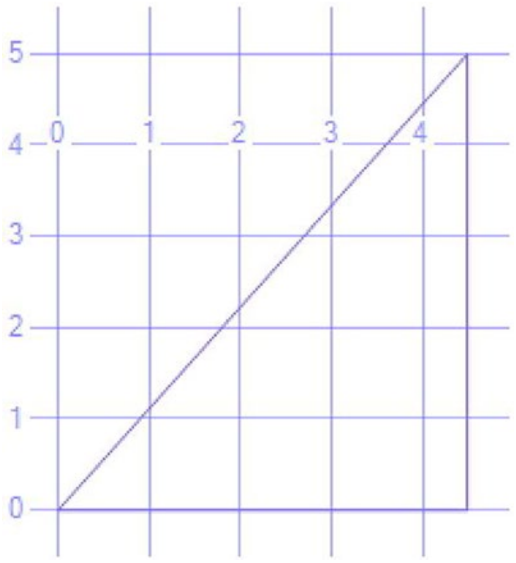


Figure 11-1. *Results of creating a LineString from WKT*

Alternatively, the same spatial instance could be created with WKB, by using the script in Listing 11-2.

Listing 11-2. Creating a LineString from a Well-Known Binary

```
DECLARE @LineString GEOMETRY ;

SET @LineString = GEOMETRY::STLineFromWKB(0x01020000000400000
0000000000000000000000000000000000000000000000124000000000000
0014400000000000000012400000000000000000000000000000000000000000
0000000, 0);

SELECT @LineString ;
```

Tip An additional extended method exists for constructing geometries. `GeomFromGML` allows you to instantiate an object, based on the Geography Markup Language (GML). For example, the script in Listing 11-3 will create a LineString from GML. A full discussion of GML is beyond the scope of this book, but the OGC specification can be found at `www.opengeospatial.org/standards/gml`.

Listing 11-3. Creating a LineString from GML

```
DECLARE @LineString GEOMETRY ;

SET @LineString =
GEOMETRY::GeomFromGml('<LineString xmlns="http://www.opengis.
net/gml">
<posList>0 0 4.5 5 4.5 0 0 0</posList> </LineString>', 0) ;

SELECT @LineString ;
```

SQL Server also provides an extended method for creating a spatial instance, by defining the x and y coordinates of a point. Listing 11-4 demonstrates the use of the `Point` method against a `GEOMETRY` variable. The method accepts three parameters. The first is the x axis coordinate, the second is the y axis coordinate, and the third is the SRID.

Listing 11-4. Using the `Point` Method

```
DECLARE @Point GEOMETRY ;

SET @Point = GEOMETRY::Point(10, 10, 0) ;

SELECT @Point ;
```

Spatial instances can also be instantiated by simply passing the well-known text or well-known binary as a value. For example, consider the script in Listing 11-5.

Listing 11-5. Passing Well-Known Text to a Variable

```
DECLARE @Polygon GEOMETRY ;

SET @Polygon = 'POLYGON((0 0, 10 0, 10 10, 0 10, 0 0))' ;

SELECT @Polygon ;
```

The graphical results of this script are displayed in Figure 11-2.

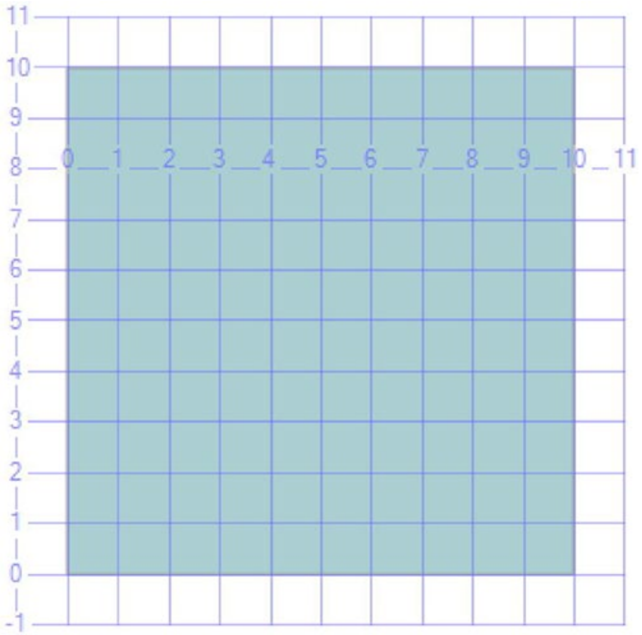


Figure 11-2. *Results of passing well-known text to a variable*

A spatial instance can be set to NULL, either by passing a NULL value directly or by using the read-only Null property of the instance. This is demonstrated in Listing 11-6.

Listing 11-6. Setting an Instance As NULL

```
--SET Variable As NULL by passing A NULL Directly

DECLARE @Polygon GEOMETRY ;

SET @Polygon = NULL ;

SELECT @Polygon ;
```

```
--Set A Variable As NULL by using the read-only Null Property

SET @Polygon = GEOMETRY::[Null] ;

SELECT @Polygon ;
```

It is possible to check the validity of your well-known text or well-known binary by using the `STIsValid` method. If the instance is not valid, it can be made valid, by using the `MakeValid` method. For example, consider the script in Listing 11-7, which instantiates a Polygon that overlaps itself.

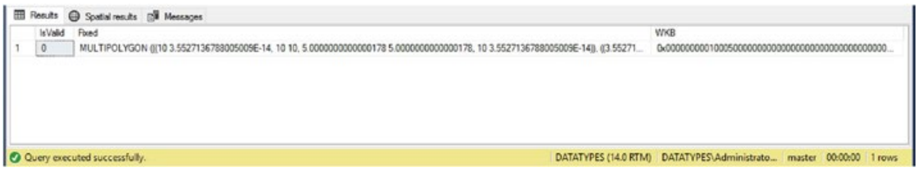
Listing 11-7. Validating and Correcting an Instance

```
DECLARE @Polygon GEOMETRY ;

SET @Polygon = GEOMETRY::STGeomFromText('POLYGON((0 0, 10 10,
10 0, 0 10, 0 0))', 0) ;

SELECT
          @Polygon.STIsValid() AS IsValid
        , @Polygon.MakeValid().ToString() AS Fixed
        , @Polygon AS WKB ;
```

The results, shown in Figure 11-3, show that the original value has returned 0, because it is not valid, and the fixed version has converted it to a MultiPolygon.



```
   IsValid  Fixed                                                                                                              WKB
1  0        MULTIPOLYGON (((10 3.5527136788005009E-14, 10 10, 5.0000000000000178 5.0000000000000178, 10 3.5527136788005009E-14)), ((3.55271...   0x00000000010005000000000000000000000000000000...

Query executed successfully.          DATATYPES (14.0 RTM)   DATATYPES\Administrato...   master   00:00:00   1 rows
```

Figure 11-3. *Results of validating and correcting an instance*

The graphical results of the query are shown in Figure 11-4.

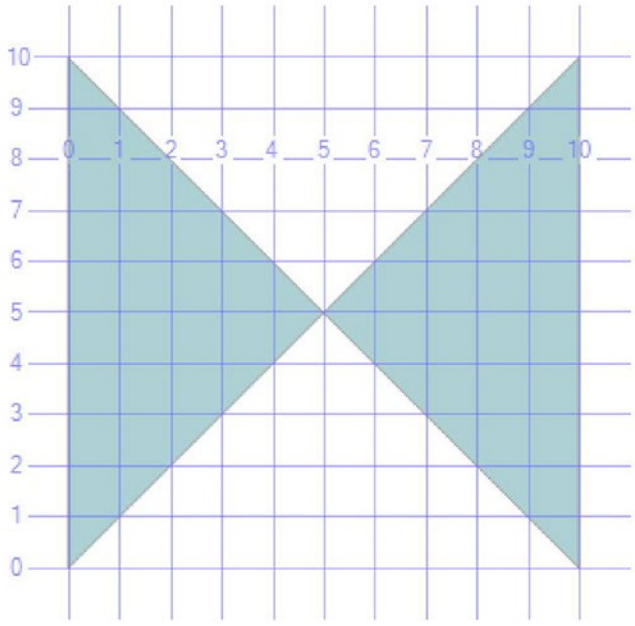


Figure 11-4. *Graphical results of validating and correcting an instance*

It is worth noting that because of the way in which spatial data is parsed, the use of `STIsValid` and `MakeValid` are somewhat limited. For example, consider the script in Listing 11-8, which attempts to instantiate a Polygon that is not enclosed.

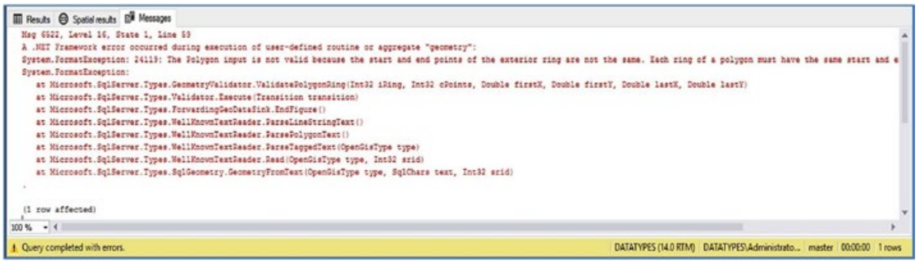
Listing 11-8. Attempt to Instantiate a Non-enclosed Polygon

```
DECLARE @LineString GEOMETRY ;

SET @LineString = GEOMETRY::STGeomFromText('POLYGON((0 0, 10
10, 10 0, 0 10))', 0) ;
```

```
SELECT
          @LineString.STIsValid() AS IsValid
        , @LineString.MakeValid().ToString() AS Fixed
        , @LineString AS WKB ;
```

Because the Polygon is not enclosed, it fails to parse, and the error, illustrated in Figure 11-5, is thrown.



```
Msg 6522, Level 16, State 1, Line 59
A .NET Framework error occurred during execution of user-defined routine or aggregate "geometry":
System.FormatException: 24119: The Polygon input is not valid because the start and end points of the exterior ring are not the same. Each ring of a polygon must have the same start and e
System.FormatException:
   at Microsoft.SqlServer.Types.GeometryValidator.ValidatePolygonRing(Int32 iRing, Int32 cPoints, Double firstX, Double firstY, Double lastX, Double lastY)
   at Microsoft.SqlServer.Types.Validator.Execute(Transition transition)
   at Microsoft.SqlServer.Types.ForwardingGeoDataSink.EndFigure()
   at Microsoft.SqlServer.Types.WellKnownTextReader.ParseLineStringText()
   at Microsoft.SqlServer.Types.WellKnownTextReader.ParsePolygonText()
   at Microsoft.SqlServer.Types.WellKnownTextReader.ParseTaggedText(OpenGisType type)
   at Microsoft.SqlServer.Types.WellKnownTextReader.Read(OpenGisType type, Int32 srid)
   at Microsoft.SqlServer.Types.SqlGeometry.GeometryFromText(OpenGisType type, SqlChars text, Int32 srid)
.

(1 row affected)
```

Query completed with errors. | DATATYPES (14.0 RTM) | DATATYPES\Administrato... | master | 00:00:00 | 1 rows

Figure 11-5. *Error thrown from instantiating a non-enclosed Polygon*

Querying Spatial Data

SQL Server provides numerous methods for interacting with spatial data. An overview of these methods can be found in Table 11-2. The table details the name of each method, whether it is an OGC standard or a Microsoft extension, and the data types the method works with. The final column provides a brief description of the method's ability.

Table 11-2. Spatial Methods

Method	OGC/Extended	Works With	Description
AsBinaryZM	Extended	GEOGRAPHY & GEOMETRY	Returns the well-known binary of a spatial instance, augmented with Z and M values, if present
AsGml	Extended	GEOGRAPHY & GEOMETRY	Returns the GML of a spatial instance
AsTextZM	Extended	GEOGRAPHY & GEOMETRY	Returns the well-known text of a spatial instance, augmented with Z and M values, if present
BufferWithCurves	Extended	GEOGRAPHY & GEOMETRY	Accepts a single parameter, which designates a distance. Returns an object representing all points, within the distance supplied, of the spatial instance against which the method was called
BufferWithTolerance	Extended	GEOGRAPHY & GEOMETRY	Accepts three parameters, denoting a distance from a spatial object, the tolerance, and a bit flag called relative that denotes if the tolerance is relative. Returns a spatial object representing the union of all points, whose distance from a geography instance is less than or equal to the distance specified, allowing for the specified tolerance

CurveToLineWithTolerance	Extended	GEOGRAPHY & GEOMETRY	Accepts two parameters, designating a tolerance, and a bit flag, indicating if the tolerance is relative. Returns a Polygon that represents an approximation of the instance against which the method was called, which contains no arc segments
EnvelopeAngle	Extended	GEOGRAPHY Only	Returns the maximum angle (in degrees) between the center point of an instance's bounding circle and a point in the geography instance
EnvelopeCenter	Extended	GEOGRAPHY Only	Returns the center of a spatial instance's bounding circle
Filter	Extended	GEOGRAPHY & GEOMETRY	Attempts to determine if a spatial object, passed as a parameter, intersects with the spatial object the method is called against. Returns 1 if the objects potentially intersect and 0 if they don't. Results may be unreliable and return false positives.
HasM	Extended	GEOGRAPHY & GEOMETRY	Validates if an M value has been specified for a spatial instance. Returns 1 if an M value exists and 0 if it does not

(continued)

Table 11-2. (*continued*)

Method	OGC/Extended	Works With	Description
`HasZ`	Extended	GEOGRAPHY & GEOMETRY	Validates if an Z value has been specified for a spatial instance. Returns 1 if a Z value exists and 0 if it does not
`InstanceOf`	Extended	GEOGRAPHY & GEOMETRY	Accepts a single parameter, which denotes a spatial type. Returns 1 if the spatial instance the method is called against is of the specified type or 0 if the instance is a different type
`IsNull`	Extended	GEOGRAPHY & GEOMETRY	Validates if a spatial instance is `Null`. Returns 1 if it is and 0 if it contains an object
`IsValidDetailed`	Extended	GEOGRAPHY & GEOMETRY	If an object is valid, the method returns 24400. If the object is not valid, it returns an error code describing why it is not valid. These error codes and their meanings can be found in Table 11-3.
`Lat`	Extended	GEOGRAPHY Only	Returns the latitude of a Point

`Long`	Extended	GEOGRAPHY Only	Returns the longitude of a Point
`M`	Extended	GEOGRAPHY & GEOMETRY	Returns the M value of the spatial instance
`MakeValid`	Extended	GEOGRAPHY & GEOMETRY	Converts a spatial instance that is not well-formed to a spatial instance that is well-formed
`MinDbCompatibilityLevel`	Extended	GEOGRAPHY & GEOMETRY	Returns the minimum database compatibility level, which provides support for the type of spatial object that the method is called against
`NumRings`	Extended	GEOGRAPHY Only	Called against a Polygon. Returns the total number of rings within the Polygon
`Reduce`	Extended	GEOGRAPHY & GEOMETRY	Accepts a single parameter that details a tolerance. Runs a Douglas-Peucker algorithm against a spatial instance and returns the approximate result*
`ReorientObject`	Extended	GEOGRAPHY Only	Changes the ring orientation of a Polygon

(*continued*)

Table 11-2. (*continued*)

Method	OGC/Extended	Works With	Description
`RingN`	Extended	GEOGRAPHY Only	Accepts a single parameter that defines a 1-based index. Returns a ring from the spatial instance the method is called against, which resides at the given index number
`ShortestLineTo`	Extended	GEOGRAPHY & GEOMETRY	Accepts a spatial object as a parameter. Returns a LineString that represents the shortest distance between the spatial object that the method is called against and the spatial object that is passed as a parameter
`STArea`	OGC	GEOGRAPHY & GEOMETRY	Returns the surface area of an instance
`STAsBinary`	OGC	GEOGRAPHY & GEOMETRY	Returns the WKB of an instance
`STAsText`	OGC	GEOGRAPHY & GEOMETRY	Returns the WKT of an instance

`STBoundary`	`OGC`	`GEOMETRY` Only	Returns the boundary of an instance. For example, calling the method against a Polygon will return a LineString marking the Polygon's border. Calling the method against a LineString will return a MultiPoint, with the start and end point of the LineString.
`STBuffer`	`OGC`	GEOGRAPHY & `GEOMETRY`	Accepts a single parameter, defining distance, and returns all points within that distance of instance. Distance can be positive or negative. When negative, the buffer will be interior to the boundary.
`STCentroid`	`OGC`	`GEOMETRY` Only	Returns the geometric center of an object. The instance that the method is invoked against must contain one of more Polygons; otherwise, the method returns `NULL`.
`STContains`	`OGC`	GEOGRAPHY & `GEOMETRY`	Evaluates if a spatial instance is contained by another. Should be called against the containing instance and accepts a single parameter for the contained instance. Returns 1 if the value is contained and 0 if it is not

(*continued*)

Table 11-2. *(continued)*

Method	OGC/Extended	Works With	Description
`STConvexHull`	`OGC`	GEOGRAPHY & GEOMETRY	Returns the smallest possible convex Polygon that contains the geometry instances that the method is called against
`STCrosses`	`OGC`	GEOMETRY Only	Validates if a given spatial instance crosses the spatial instance that the method is called against. Accepts a single parameter that describes the second spatial instance. Returns 1 if the second spatial instance crosses the first and 0 if it does not
`STCurveN`	`OGC`	GEOGRAPHY & GEOMETRY	Returns the curve of a LineString, CircularString, CompoundCurve, or MultiLineString spatial instance. Accepts a single parameter, which defines the index of the curve that should be returned, where a spatial instance has multiple curves
`STCurveToLine`	`OGC`	GEOGRAPHY & GEOMETRY	Creates an approximate polygonal spatial instance, from a curved spatial instance, such as a CircularString or CurvePolygon. Returns an instance of a LineString or Polygon, respectively.

`STDifference`	OGC	GEOGRAPHY & GEOMETRY	Returns the portion of a geography, which the method is called against, that does not reside within the boundaries of a second spatial instance, which is passed as a parameter to the method
`STDimension`	OGC	GEOGRAPHY & GEOMETRY	Returns that maximum number of dimensions of a spatial instance
`STDisjoint`	OGC	GEOGRAPHY & GEOMETRY	Validates if the intersection set between two geometries is empty. Returns 1 if it is empty or 0 if it is not empty. Accepts a single parameter, which is the spatial instance, to intersect with the spatial instance that the method is called against
`STDistance`	OGC	GEOGRAPHY & GEOMETRY	Calculates the approximate shortest distance between the spatial instance that the method is called against and a spatial instance that is passed to the method as a parameter
`STEndpoint`	OGC	GEOGRAPHY & GEOMETRY	Returns the final Point used to define a spatial instance

(*continued*)

Table 11-2. *(continued)*

Method	OGC/Extended	Works With	Description
`STEnvelope`	OGC	GEOMETRY Only	Returns a rectangular Polygon that encloses the spatial instance that the method has been called against
`STEquals`	OGC	GEOGRAPHY & GEOMETRY	Validates if the spatial instance that the method is called against has the same points set as a spatial instance that is passed to the method as a parameter. Returns 1 if the points sets are the same and 0 if they are not
`STExteriorRing`	OGC	GEOMETRY Only	When called against a Polygon, the method will return the Polygon's exterior ring.
`STGeometryN`	OGC	GEOGRAPHY & GEOMETRY	Returns a specific object from a geometry collection. Accepts a single parameter, which is the 1-based index of the object to return
`STGeometryType`	OGC	GEOGRAPHY & GEOMETRY	Returns the type of a spatial instance, as defined by the OGC

STInteriorRingN	OGC	GEOMETRY Only	When called against a Polygon, the method will return the Polygon's interior ring.
STIntersection	OGC	GEOGRAPHY & GEOMETRY	Returns a spatial instance that covers the area that is overlapped by a spatial instance that the method is called against and a spatial instance that is passed to the method as a parameter
STIntersects	OGC	GEOGRAPHY & GEOMETRY	Validates if the spatial instance, against which the method is called, overlaps with a spatial instance that is passed to the method as a parameter. Returns 1 if the instances intersect and 0 if they do not
STIsClosed	OGC	GEOGRAPHY & GEOMETRY	Checks if the start point and end point of a spatial object are the same. Returns 1 if they are the same and 0 if they are not the same
STIsEmpty	OGC	GEOGRAPHY & GEOMETRY	Validates if a spatial instance is empty or contains an object. Returns 1 if the instance is empty and 0 if it contains an object

(continued)

Table 11-2. *(continued)*

Method	OGC/Extended	Works With	Description
`STIsRing`	OGC	`GEOMETRY` Only	Validates if a spatial instance meets the following criteria: • Is a LineString • Does not intersect itself (except at end point) • Multiple objects do not intersect each other (except where the point of intersection is on both object's boundaries) • Is closed Returns 1 if the conditions are all true, or 0 if 1 or more of the conditions are false
`STIsSimple`	OGC	`GEOMETRY` Only	Validates if a spatial instance meets the following conditions: • Does not intersect itself (except at end point) • Multiple objects do not intersect each other (except where the point of intersection is on both objects' boundaries) Returns 1 if both conditions are true. Returns 0 if either condition is false

`STIsValid`	OGC	GEOGRAPHY & GEOMETRY	Validates that a spatial instance is well-formed. Returns 1 if the instance is well-formed and 0 if it is not well-formed
`STLength`	OGC	GEOGRAPHY & GEOMETRY	Returns the total length of a spatial object. For a Polygon, this is the length of the perimeter. For a Point, the length is always 0.
`STNumCurves`	OGC	GEOGRAPHY & GEOMETRY	Must be called against simple, primitive, one-dimensional spatial instances. Returns the number of curves in the instance
`STNumGeometries`	OGC	GEOGRAPHY & GEOMETRY	Returns the number of spatial objects in a multi-geometry. Returns 1 if the instance is a primitive geometry
`STNumInteriorRing`	OGC	GEOMETRY Only	Called against a Polygon and returns the number of interior rings that the Polygon contains
`STNumPoints`	OGC	GEOGRAPHY & GEOMETRY	Returns the number of Points that were used to describe a spatial instance

(continued)

Table 11-2. (*continued*)

Method	OGC/Extended	Works With	Description
STOverlaps	OGC	GEOGRAPHY & GEOMETRY	Validates if the spatial instance that the method was called against overlaps a spatial instance that is passed in as a parameter. Returns 1 if the objects overlap and 0 if they do not
STPointN	OGC	GEOGRAPHY & GEOMETRY	Accepts a single parameter, which describes a 1-based index location of an object within a multipart geometry. Returns the spatial object located at that index
STPointOnSurface	OGC	GEOMETRY Only	Returns an arbitrary point, within the interior of a spatial instance
STRelate	OGC	GEOMETRY Only	Validates if the spatial instance against which the method is called is related to a spatial instance that is passed as the first parameter to the method. Relationships are defined by the OGC Dimensionally Extended 9 Intersection Model (DE-9IM) pattern matrix. The pattern matrix used is the second parameter passed to the method.**

`STSrid`	OGC	GEOGRAPHY & GEOMETRY	Returns the SRID of the spatial instance, against which the method was called
`STStartPoint`	OGC	GEOGRAPHY & GEOMETRY	Returns the first Point used to define a spatial instance
`STSymDifference`	OGC	GEOGRAPHY & GEOMETRY	Returns a spatial instance, which includes all Point that are: • Within the spatial instance the method was called against • Within a spatial instance passed as a parameter • Not within both spatial instances
`STTouches`	OGC	GEOMETRY Only	Validates if the spatial instance against which the method was called spatially touches a spatial instance passed to the method as a parameter. Returns 1 if the instances touch and 0 if they do not

(*continued*)

Table 11-2. *(continued)*

Method	OGC/Extended	Works With	Description
`STUnion`	OGC	GEOGRAPHY & GEOMETRY	Returns a spatial instance that provides the union between the spatial instance against which the method was called and a spatial instance passed to the method as a parameter. The resultant union could be either a primitive or multipart geometry, depending on the inputs.
`STWithin`	OGC	GEOGRAPHY & GEOMETRY	Accepts a single parameter, which defines a spatial object. Validates if the spatial instance passed as a parameter is completely inside the spatial instance against which the method was called. Returns 1 if it is and 0 if it is not
`STX`	OGC	GEOMETRY Only	Called against a Point. Returns the Point's x coordinate

STY	OGC	GEOMETRY Only	Called against a Point. Returns the Point's y coordinate
ToString	Extended	GEOGRAPHY & GEOMETRY	Returns the well-known text of the spatial instance against which the method is called
Z	Extended	GEOGRAPHY & GEOMETRY	Returns the Z value of the spatial instance

**A Douglas-Peucker algorithm takes a curve composed of line segments and attempts to find a similar curve with fewer points.*

***Details of the DE-9IM pattern matrices can be found at* `https://link.springer.com/referenceworkentry/10.1007%2F978-3-319-17885-1_298`.

Table 11-2 called out the IsValidDetailed, which has been implemented by Microsoft as an extended method. Table 11-3 details the error codes that can be thrown by the IsValidDetailed method, when a spatial instance is not valid.

Table 11-3. *`IsValidDetailed` Error Codes*

Error Code	Description
24400	The instance is valid.
24401	The instance is not valid, but the reason is unknown.
24402	The instance is not valid, because a point is isolated, which is not valid for the object's type.
24403	The instance is not valid, because some pair of polygon edges overlap.
24404	The instance is not valid because a polygon ring intersects itself or another ring.
24405	The instance is not valid, because a polygon ring intersects itself or another ring, and the ring number cannot be returned.
24406	The instance is not valid, because a curve degenerates to a point.
24407	The instance is not valid, because a polygon ring collapses to a line.
24408	The instance is not valid, because a polygon ring is not closed.
24409	The instance is not valid, because some portion of a polygon ring lies in the interior of a polygon.
24410	The instance is not valid, because a ring is the first ring in a polygon but is not the exterior ring.
24411	The instance is not valid, because a ring lies outside the exterior ring of its polygon.

(*continued*)

Table 11-3. *(continued)*

Error Code	Description
24412	The instance is not valid, because the interior of a polygon is not connected.
24413	The instance is not valid, because of two overlapping edges in a curve.
24414	The instance is not valid, because an edge of a curve overlaps an edge of another curve.
24415	The instance is not valid, because a polygon has an invalid ring structure.
24416	The instance is not valid, because in a curve, the edge is either a line or a degenerate arc with antipodal end points.

In addition to the methods detailed in Tables 11-1 and 11-2, there are also a number of aggregation methods available when using the `GEOMETRY` and `GEOGRAPHY` data types. These are detailed in Table 11-4.

Table 11-4. *Aggregation Methods*

Method	Description
`CollectionAggregate`	Creates a GeometryCollection from a set of spatial objects
`ConvexHullAggregate`	Returns the convex hull of a set of spatial objects
`EnvelopeAggregate`	Returns a bounding spatial object for a set of spatial objects
`UnionAggregate`	Returns a union of a set of spatial objects

Now that you are aware of the spatial methods that SQL Server exposes, you can look at how they can be employed in practice, by using the WideWorldImporters database. Imagine that we are responsible for delivery routes, and we must plan a route in Alabama.

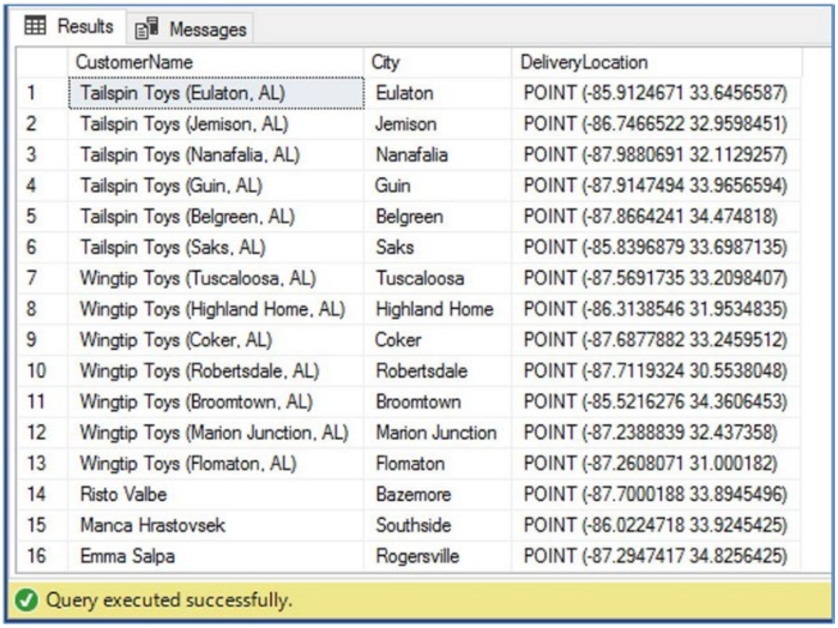
The first thing that we want to do is validate our data. Are all our customers, with delivery locations marked as being in Alabama, actually within the Alabama state border? The Sales.Customers table contains a `GEOGRAPHY` column called DeliveryLocation, with each row containing a Point object that maps their delivery address. We can use the `STWithin` method against this column, passing in the Border column from the Application.StateProvinces table, which maps the state border. This is demonstrated in Listing 11-9.

Listing 11-9. Validating That All Addresses Are Within the Alabama State Border

```
DECLARE @StateBorder GEOGRAPHY = (
                SELECT Border
                FROM Application.StateProvinces
                WHERE StateProvinceName = 'Alabama') ;

SELECT
           Customer.CustomerName AS CustomerName
         , City.CityName AS City
         , Customer.DeliveryLocation.ToString() AS
         DeliveryLocation
FROM SALES.Customers Customer
INNER JOIN Application.Cities City
        ON City.CityID = Customer.DeliveryCityID
WHERE Customer.DeliveryLocation.STWithin(@StateBorder) = 1 ;
```

The results of this query (which are displayed in Figure 11-6) validate that all 16 customers with Alabama delivery addresses actually reside in Alabama.



Results | Messages

	CustomerName	City	DeliveryLocation
1	Tailspin Toys (Eulaton, AL)	Eulaton	POINT (-85.9124671 33.6456587)
2	Tailspin Toys (Jemison, AL)	Jemison	POINT (-86.7466522 32.9598451)
3	Tailspin Toys (Nanafalia, AL)	Nanafalia	POINT (-87.9880691 32.1129257)
4	Tailspin Toys (Guin, AL)	Guin	POINT (-87.9147494 33.9656594)
5	Tailspin Toys (Belgreen, AL)	Belgreen	POINT (-87.8664241 34.474818)
6	Tailspin Toys (Saks, AL)	Saks	POINT (-85.8396879 33.6987135)
7	Wingtip Toys (Tuscaloosa, AL)	Tuscaloosa	POINT (-87.5691735 33.2098407)
8	Wingtip Toys (Highland Home, AL)	Highland Home	POINT (-86.3138546 31.9534835)
9	Wingtip Toys (Coker, AL)	Coker	POINT (-87.6877882 33.2459512)
10	Wingtip Toys (Robertsdale, AL)	Robertsdale	POINT (-87.7119324 30.5538048)
11	Wingtip Toys (Broomtown, AL)	Broomtown	POINT (-85.5216276 34.3606453)
12	Wingtip Toys (Marion Junction, AL)	Marion Junction	POINT (-87.2388839 32.437358)
13	Wingtip Toys (Flomaton, AL)	Flomaton	POINT (-87.2608071 31.000182)
14	Risto Valbe	Bazemore	POINT (-87.7000188 33.8945496)
15	Manca Hrastovsek	Southside	POINT (-86.0224718 33.9245425)
16	Emma Salpa	Rogersville	POINT (-87.2947417 34.8256425)

Query executed successfully.

Figure 11-6. *Results of validating that all addresses are within the Alabama state border*

Next, we want to check the distance of each delivery location from our depot and order the results by that distance. This will help us plan the route. This technique is known as finding nearest neighbors and is demonstrated in Listing 11-10, which calculates the distance and orders the results, by using the DeliveryLocation from the Application.SystemParameters table, which marks the location of WideWorldImporters.

Note The WideWorldImporters depot is in California, but while the distances to the Alabama locations are vast, they still allow us to plan the driver's most sensible route.

Listing 11-10. Finding Nearest Neighbors

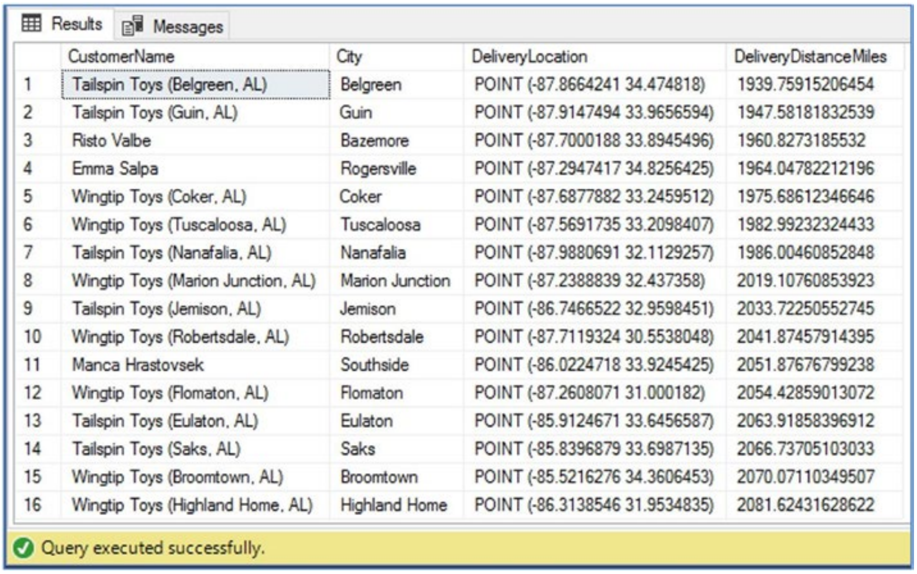
```
DECLARE @StateBorder GEOGRAPHY = (
                SELECT Border
                FROM Application.StateProvinces
                WHERE StateProvinceName = 'Alabama') ;

DECLARE @Office GEOGRAPHY = (
                SELECT DeliveryLocation
                FROM Application.SystemParameters) ;

DECLARE @MilesRatio INT = 0.000621371 ;

SELECT
           Customer.CustomerName AS CustomerName
         , City.CityName AS City
         , Customer.DeliveryLocation.ToString() AS
         DeliveryLocation
         , Customer.DeliveryLocation.STDistance(@Office) * @
         MilesRatio AS DeliveryDistanceMiles
FROM SALES.Customers Customer
INNER JOIN Application.Cities City
        ON City.CityID = Customer.DeliveryCityID
WHERE Customer.DeliveryLocation.STWithin(@StateBorder) = 1
ORDER BY DeliveryDistanceMiles ;
```

The results of this query can be seen in Figure 11-7.



Results | Messages

	CustomerName	City	DeliveryLocation	DeliveryDistanceMiles
1	Tailspin Toys (Belgreen, AL)	Belgreen	POINT (-87.8664241 34.474818)	1939.75915206454
2	Tailspin Toys (Guin, AL)	Guin	POINT (-87.9147494 33.9656594)	1947.58181832539
3	Risto Valbe	Bazemore	POINT (-87.7000188 33.8945496)	1960.8273185532
4	Emma Salpa	Rogersville	POINT (-87.2947417 34.8256425)	1964.04782212196
5	Wingtip Toys (Coker, AL)	Coker	POINT (-87.6877882 33.2459512)	1975.68612346646
6	Wingtip Toys (Tuscaloosa, AL)	Tuscaloosa	POINT (-87.5691735 33.2098407)	1982.99232324433
7	Tailspin Toys (Nanafalia, AL)	Nanafalia	POINT (-87.9880691 32.1129257)	1986.00460852848
8	Wingtip Toys (Marion Junction, AL)	Marion Junction	POINT (-87.2388839 32.437358)	2019.10760853923
9	Tailspin Toys (Jemison, AL)	Jemison	POINT (-86.7466522 32.9598451)	2033.72250552745
10	Wingtip Toys (Robertsdale, AL)	Robertsdale	POINT (-87.7119324 30.5538048)	2041.87457914395
11	Manca Hrastovsek	Southside	POINT (-86.0224718 33.9245425)	2051.87676799238
12	Wingtip Toys (Flomaton, AL)	Flomaton	POINT (-87.2608071 31.000182)	2054.42859013072
13	Tailspin Toys (Eulaton, AL)	Eulaton	POINT (-85.9124671 33.6456587)	2063.91858396912
14	Tailspin Toys (Saks, AL)	Saks	POINT (-85.8396879 33.6987135)	2066.73705103033
15	Wingtip Toys (Broomtown, AL)	Broomtown	POINT (-85.5216276 34.3606453)	2070.07110349507
16	Wingtip Toys (Highland Home, AL)	Highland Home	POINT (-86.3138546 31.9534835)	2081.62431628622

Query executed successfully.

Figure 11-7. *Results of finding nearest neighbors*

You will notice that in Listing 11-10, the distance is multiplied by 0.000621371. This is because the distance is natively in meters. The unit of measurement of distance is associated with the SRID, so we can double check this, by enhancing our query to expose the SRID, as shown in Listing 11-11. This query exposes the STRid property of each spatial instance and uses this value to join to the sys.spatial_reference_systems system table, to expose the unit of measure column.

Listing 11-11. Exposing the SRID

```
DECLARE @StateBorder GEOGRAPHY = (
                SELECT Border
                FROM Application.StateProvinces
                WHERE StateProvinceName = 'Alabama') ;

DECLARE @Office GEOGRAPHY = (
                SELECT DeliveryLocation
                FROM Application.SystemParameters) ;

DECLARE @MilesRatio INT = 0.000621371 ;

SELECT
           Customer.CustomerName AS CustomerName
         , City.CityName AS City
         , Customer.DeliveryLocation.ToString() AS
         DeliveryLocation
         , Customer.DeliveryLocation.STDistance(@Office) *
         @MilesRatio AS DeliveryDistanceMiles
         , Customer.DeliveryLocation.STSrid AS SRID
         , srid.unit_of_measure
FROM SALES.Customers Customer
INNER JOIN Application.Cities City
        ON City.CityID = Customer.DeliveryCityID
INNER JOIN sys.spatial_reference_systems srid
        ON srid.spatial_reference_id = Customer.
DeliveryLocation.STSrid
WHERE Customer.DeliveryLocation.STWithin(@StateBorder) = 1
ORDER BY DeliveryDistanceMiles ;
```

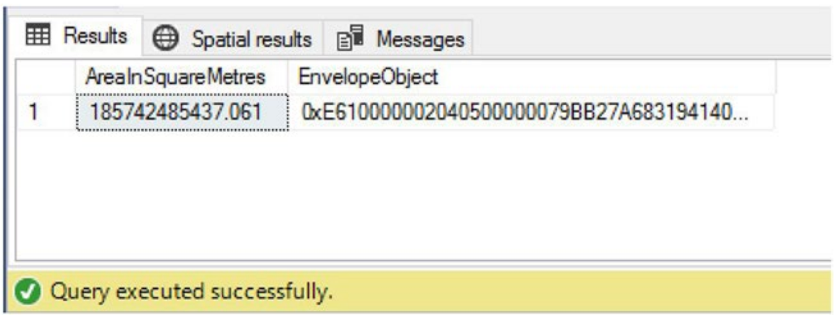
While we know that all deliveries are in the state of Alabama, we may wish to discover how big our actual delivery area is. We can achieve this by creating an envelope and then calculating its area. This approach uses the `EnvelopeAggregate` extended method and the `STArea` method, as demonstrated in Listing 11-12.

Listing 11-12. Calculating the Area of an Aggregate Envelope

```
DECLARE @StateBorder GEOGRAPHY = (
                SELECT Border
                FROM Application.StateProvinces
                WHERE StateProvinceName = 'Alabama') ;

SELECT
          GEOGRAPHY::EnvelopeAggregate(DeliveryLocation).
          STArea() AS AreaInSquareMetres
        , GEOGRAPHY::EnvelopeAggregate(DeliveryLocation)
        AS EnvelopeObject
FROM Sales.Customers
WHERE DeliveryLocation.STWithin(@StateBorder) = 1 ;
```

The results of this query can be seen in Figure 11-8.



	AreaInSquareMetres	EnvelopeObject
1	185742485437.061	0xE6100000020405000000079BB27A683194140...

Query executed successfully.

Figure 11-8. *Results of calculating the area of an aggregate envelope*

The graphical results of the query in Listing 11-12 are illustrated in Figure 11-9.

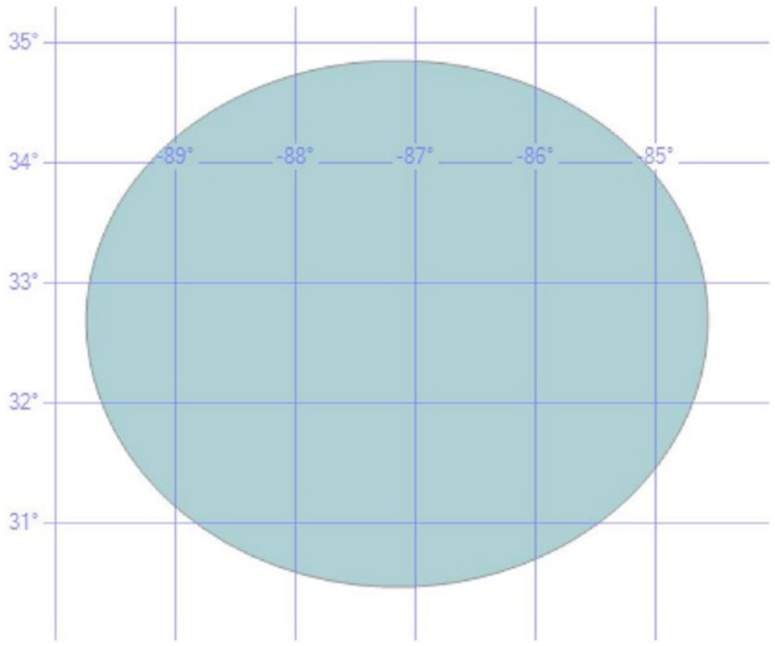


Figure 11-9. *Graphical results of calculating the area of an aggregate envelope*

Indexing Spatial Data

The following sections provide an overview of spatial indexes, before demonstrating how to create them.

Understanding Spatial Indexes

Spatial indexes are a special type of index, implemented in SQL Server, that can improve the performance of certain queries against spatial data types. Table 11-5 describes the predicate patterns that can benefit from spatial indexes, when used in a `WHERE` or `JOIN` clause.

***Table 11-5.** Queries That Can Benefit from Spatial Indexes*

Data Type	Method	Operator
GEOMETRY & GEOGRAPHY	STDistance	< <=
GEOMETRY & GEOGRAPHY	`STEquals`	=
GEOMETRY & GEOGRAPHY	`STIntersects`	=
GEOMETRY Only	`STContains`	=
GEOMETRY Only	`STOverlaps`	=
GEOMETRY Only	`STTouches`	=
GEOMETRY Only	`STWithin`	=

Just as with traditional indexes, spatial indexes use a B-Tree structure (see Chapter 5 for further information on B-Tree indexes), meaning that the spatial data must be represented in a linear order. To achieve this, SQL Server decomposes space into a nested grid system, before building an index.

The grid will have four layers. Each cell in the first (Level 1) grid will contain another (Level 2) grid, and so on. Each of the four grid layers can be given a separate density, with low density being defined as 4 × 4 cells, medium density as 8 × 8 cells, and high density as 16 × 16 cells. Each cell within the grid is numbered using a Hilbert space-filling curve algorithm.

Tip A full discussion of the Hilbert space-filling curve is beyond the scope of this book, but further details can be found in many locations online, including `mathworld.wolfram.com/HilbertCurve.html`.

Once the grid system has been created, SQL Server reads the data from the column row by row. For each row, it will associate the spatial object with each Level 1 cell that it touches. For each touched cell, it will then drop down to Level 2 and repeat the process. This then happens again for Level 3 and Level 4, as required. This process is called tessellation. The output of the process is a set of touched cells, which can be stored in the index and subsequently used to calculate their spatial position, relative to other objects.

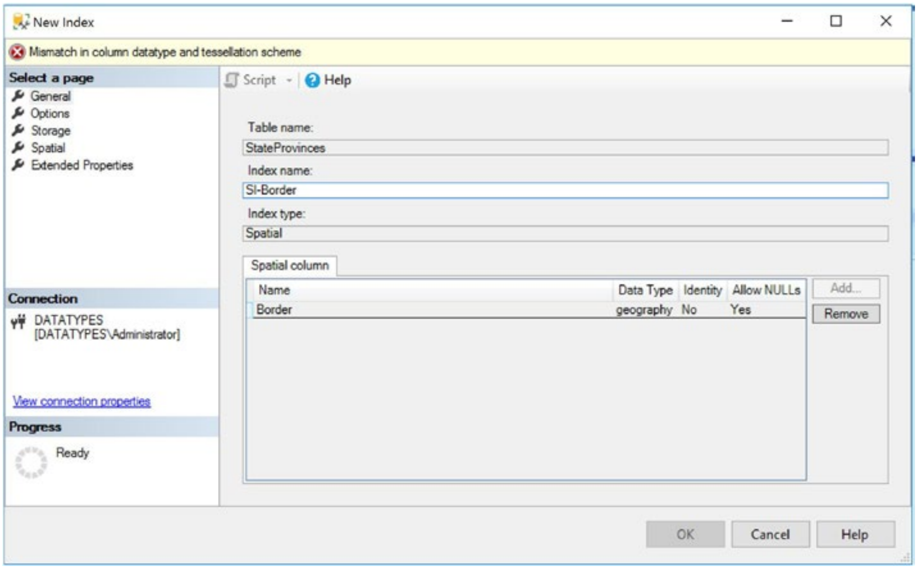
The tessellation process requires a bounding box, and this can behave differently, depending on the tessellation system used. The tessellation systems are data-type dependent and you will have the option of using automatic coordinates for the bounding box or defining your own.

You may further configure the tessellation process, by defining how many tessellation cells should be used. What this means is that you can cap the maximum number of touched cells recorded for a single object. It is worth noting, however, that this only affects Levels 2 through 4. Level 1 will record as many cells as the object touches, regardless of your configuration.

Creating Spatial Indexes

To demonstrate the creation of a spatial index, we will create an index on the Border column of the Application.StateProvinces table in the WideWorldImporters database. We will use an auto grid system, and we will configure medium density for Levels 1 to 3, with high density for Level 4.

To create the index through SQL Server Management Studio, drill through Databases ➤ WideWorldImporters ➤ Tables ➤ Application. StateProvinces in Object Explorer. Then select New Index ➤ Spatial Index from the context menu of the Indexes folder. This will cause the General page of the New Index dialog bog to be invoked, as shown in Figure 11-10.


New Index

Mismatch in column datatype and tesselation scheme

Select a page: General, Options, Storage, Spatial, Extended Properties

Connection: DATATYPES [DATATYPES\Administrator]

View connection properties

Progress: Ready

Table name: StateProvinces

Index name: SI-Border

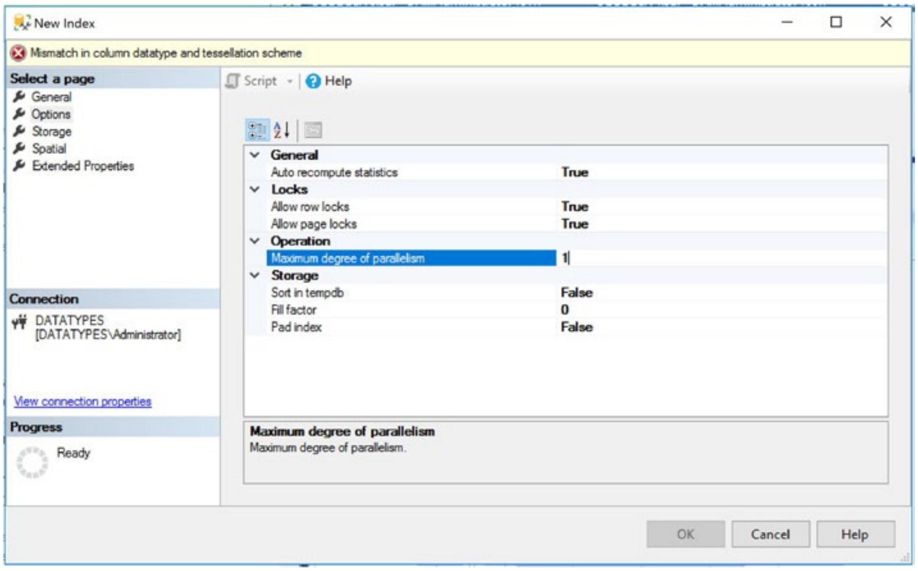
Index type: Spatial

Spatial column

Name	Data Type	Identity	Allow NULLs
Border	geography	No	Yes

Figure 11-10. *New Index dialog box—General page*

On this page of the dialog box, we have given the index a meaningful name and used the Add button, to add the column that we wish to index. We will now progress to the Options page, which is illustrated in Figure 11-11.



Mismatch in column datatype and tessellation scheme

DATATYPES
[DATATYPES\Administrator]

Ready

General
Auto recompute statistics True
Locks
Allow row locks True
Allow page locks True
Operation
Maximum degree of parallelism 1
Storage
Sort in tempdb False
Fill factor 0
Pad index False

Maximum degree of parallelism
Maximum degree of parallelism.

Figure 11-11. *New Index dialog box—Options page*

The Options page will look familiar to anyone who has created a traditional index. Table 11-6 details each of the options available.

Table 11-6. *Spatial Index Options*

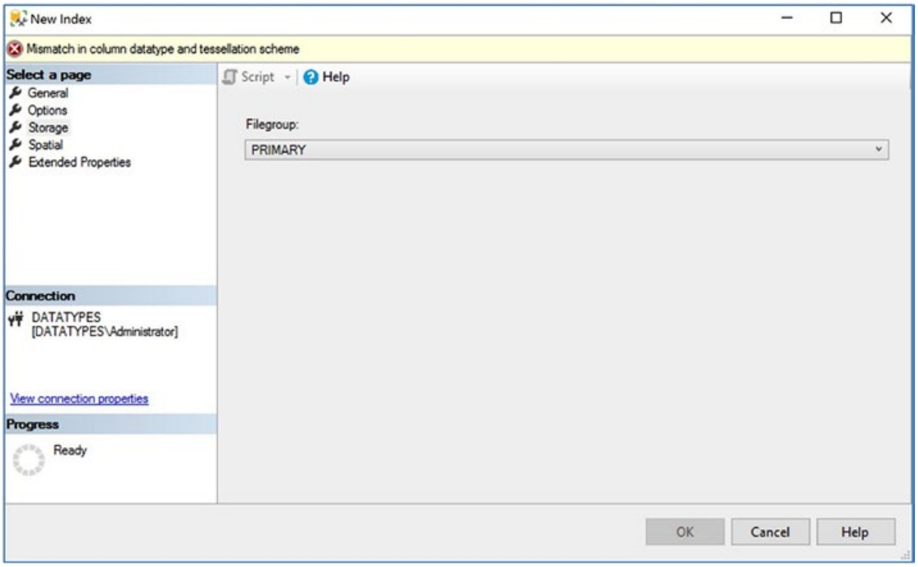
Option	Description
Auto Recompute Statistics	Specifies if statistics should be updated automatically when they are deemed out of date
Allow Row Locks	Specifies if row locks can be acquired when accessing the index
Allow Page Locks	Specifies if page locks can be acquired when accessing the index
Maximum Degree of Parallelism	Has no effect for building primary spatial indexes, as this operation is always single-threaded

(*continued*)

Table 11-6. (*continued*)

Option	Description
Sort in TempDB	If specified, Sort in TempDB will cause the intermediate result set to be stored in TempDB, as opposed to the user database. This could mean that the index is built faster.
Fill Factor	Specifies a percentage of free space that will be left on each index page at the lowest level of the index. The default is 0 (100% full), meaning that only enough space for a single row will be left. Specifying a percentage lower than 100—for example, specifying 70—will leave 30% free space, which can reduce page splits, if there are likely to be frequent row inserts.
Pad Index	Applies a fill factor (see preceding) to the intermediate levels of a B-Tree

On the Storage page, illustrated in Figure 11-12, you can specify the filegroup that the index will be created on. Usually, it is best for indexes to be aligned with the same filegroup (or partition schema) as their table, for performance. From a maintenance perspective, it may be helpful to store the index on a different filegroup when the table is partitioned. If you don't, the index will have to be dropped before the table is repartitioned.

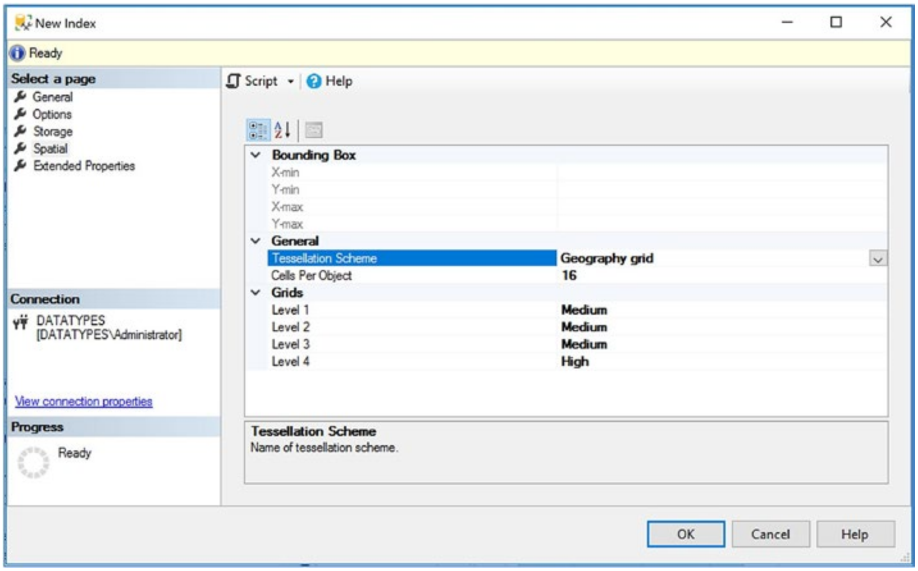

New Index

Mismatch in column datatype and tessellation scheme

Filegroup: PRIMARY

Figure 11-12. *New Index dialog box—Storage page*

The Spatial page is shown in Figure 11-13. This is where we can configure the spatial specific options of our index. In the General section of the page, we can choose our tessellation system—either automatic or manual. If we select manual, the Grids area of the page will become active, and we can select the grid densities that we would like to use. If the manual geometry system is selected, the Bounding Box area of the page will also become active, and we can specify our bounding box coordinates.



Bounding Box
X-min
Y-min
X-max
Y-max
General
Tessellation Scheme: Geography grid
Cells Per Object: 16
Grids
Level 1: Medium
Level 2: Medium
Level 3: Medium
Level 4: High

Tessellation Scheme
Name of tessellation scheme.

Figure 11-13. *New Index dialog box—Spatial page*

Tip A geometry system must be used on a GEOMETRY column and a geography system must be used on a GEOGRAPHY column.

Alternatively, we could use T-SQL to create the index. The script in Listing 11-13 will create the same index as discussed previously.

Listing 11-13. Creating a Spatial Index

```
USE WideWorldImporters
GO

CREATE SPATIAL INDEX [SI-Border]
ON Application.StateProvinces (Border)
USING  GEOGRAPHY_GRID
WITH (GRIDS =(LEVEL_1 = MEDIUM,LEVEL_2 = MEDIUM,LEVEL_3 =
MEDIUM,LEVEL_4 = HIGH),
                CELLS_PER_OBJECT = 16,
                PAD_INDEX = OFF,
                STATISTICS_NORECOMPUTE = OFF,
                SORT_IN_TEMPDB = OFF,
                DROP_EXISTING = OFF,
                ONLINE = OFF,
                ALLOW_ROW_LOCKS = ON,
                ALLOW_PAGE_LOCKS = ON, MAXDOP = 1)
ON [PRIMARY] ;
```

Summary

Spatial objects can be constructed by using various means. They can be created using OGC standard methods, from either well-known text or well-known binary, or they can be constructed by using a Microsoft extended method from GML. SQL Server also supports passing well-known text directly into a variable of a table's cell.

A wealth of methods exist that allow developers to easily interact with spatial data. Commonly used methods include `STDistance`, which will return the distance between two geometries, and `STWithin`, which will check if an instance resides within the same space as another. There are also aggregate methods that allow multiple instances to have union, envelope, and convex hull applied to them as a group, as well as converting multiple instances to a single collection.

SQL Server provides spatial indexes, which can improve the performance of certain types of queries. The index support for queries depends on the data type, the method in the `WHERE` or `JOIN` clause, and the arithmetic operator used.

Spatial indexes use B-Tree structures, but before the B-Tree is created, a grid system of space is created, with four nested levels. The spatial instances from the column are then read one by one into the grid system, and their touched cells are numbered, using a Hilbert curve. This means that spatial instances can be indexes in a linear fashion, but detailing their proximity to each other.