Migration from SpatialWare To SQL Server 2008

This document is intended for users familiar with the capabilities of Pitney Bowes Software Inc.'s SpatialWare software who are interested to know about the spatial capabilities in SQL Server. In particular, this document serves to highlight the issues involved in the migration from SpatialWare to SQL Server, and the effort involved in such an action.

United States:
Phone: 518.285.6000
Fax: 518.285.6070
Sales: 800.327.8627
Government Sales: 800.619.2333
Technical Support: 518.285.7283
Technical Support Fax: 518.285.6080
pbinsight.com

Canada:
Phone: 416.594.5200
Fax: 416.594.5201
Sales: 800.268.3282
Technical Support:.518.285.7283
Technical Support Fax: 518.285.6080
pbinsight.ca

Europe/United Kingdom:
Phone: +44.1753.848.200
Fax: +44.1753.621.140
Technical Support: +44.1753.848.229
pbinsight.co.uk

Asia Pacific/Australia:
Phone: +61.2.9437.6255
Fax: +61.2.9439.1773
Technical Support: 1.800.648.899
pbinsight.com.au

September 7, 2011

Table of Contents

Overview

SpatialWare for SQL Server was developed to extend the capabilities of SQL Server through the addition of spatial functionality. Subsequently, Microsoft has released SQL Server 2008, which, among other things, incorporates native spatial functionality within SQL Server. From SQL Server 2008 to SQL Server 2008 R2, there were additional minor changes in spatial capabilities within the database engine. The next release of SQL Server (code-named Denali) is expected to deliver more improvements to these capabilities.

This gives the SpatialWare user the opportunity to migrate to SQL Server 2008, in order to benefit from the tight integration of spatial within SQL Server 2008. There are a few caveats, when considering such a migration:

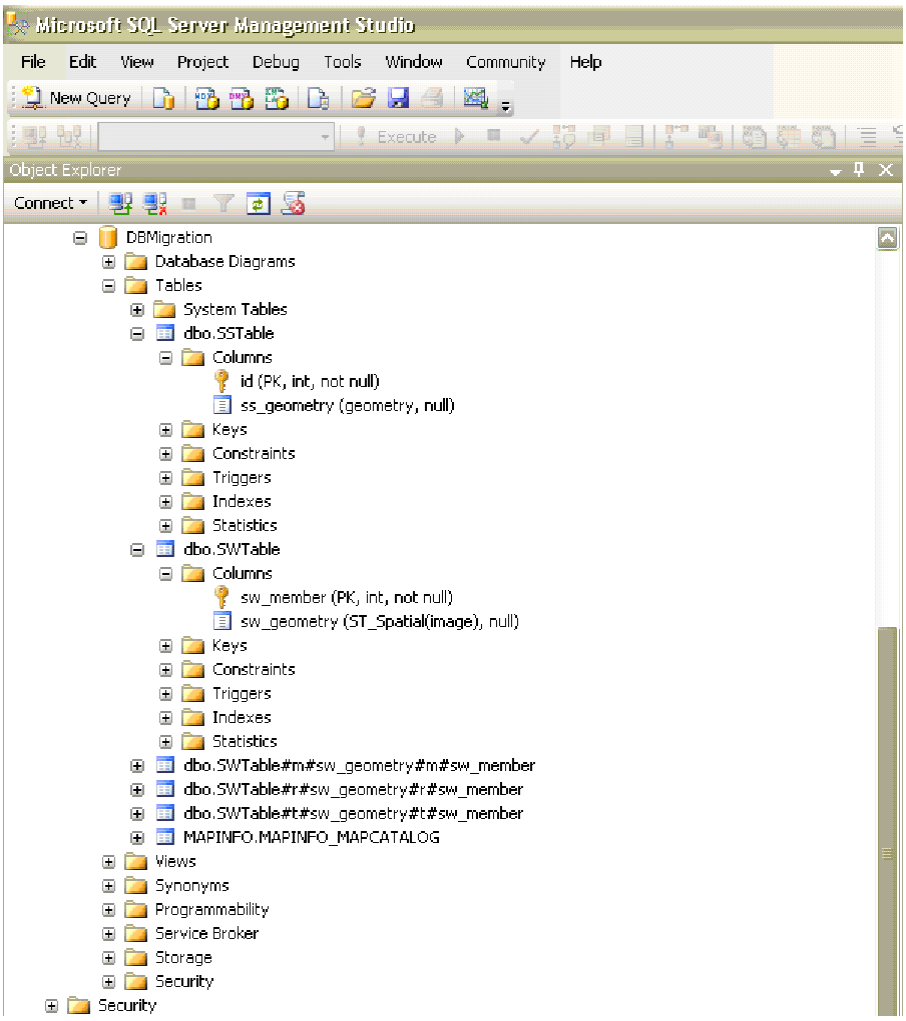
- If you are accessing SpatialWare through a client application from PBBI, only MapInfo Professional and MapXtreme 2008 support SQL Server 2008 spatial data.
- There are not direct equivalents in SQL Server 2008 for all the spatial functionality in SpatialWare. In most cases, it is not very difficult to discover the required functionality in SQL Server 2008. Some suggestions are provided in the document appendix. Users may want to consider use of capabilities in SQL Server that are not available in SpatialWare.
- An SRID being used in SpatialWare may be unavailable in SQL Server 2008. This case will be very rare.

SpatialWare vs. SQL Server 2008

The following table highlights some of the differences between SpatialWare (which is installed on the SQL Server platform, possibly SQL Server 2008) and SQL Server 2008 (where here we are referring to SQL Server 2008 with its own spatial capabilities):

	SpatialWare	SQL Server 2008
Spatial data types	ST_Spatial	geometry, geography
Spherical support	spherical functions	geography type
Coordinate system	external to ST_Spatial	SRID stored in data
Spatial indexes	rtree	grid
Spatial queries	sp_spatial_query, sp_nearest and SQL statements using UDFs	SQL statements using spatial methods

The following screenshot of Microsoft SQL Server Management Studio illustrates some differences. The SpatialWare table SWTable and its auxiliary tables such as SWTable#r#sw_geometry#r#sw_member stand in contrast to the SQL Server 2008 spatial table SSTable.



Data Types

SpatialWare added a user-defined data type called ST_Spatial to store spatial data. SQL Server now offers 2 system spatial data types: geometry and geography. Geometry is intended to store data on a flat 2-dimensional surface, with optional third dimension z values and measure m values. Geography is intended for data on the surface of a sphere, such as the earth, defined using latitude and longitude, again with optional z and m values.

SQL Server 2008 has stronger support for the geometry type than for the geography type. Also, the geometry type is more forgiving than geography, in terms of what is valid. Therefore, where possible, users should consider migrating SpatialWare data to type geometry. However, if you need spherical functions on the data, such as the SpatialWare HG_SphericalArea, or if the data is inherently spherical and the geography type has enough functionality for your purposes and your data can be stored as valid geographies, then you should choose type geography. With SQL Server 2008, STArea will return the correct spherical area for a geography, but "square degrees" for latitude/longitude data represented as a geometry.

The ST_Spatial data does not carry the coordinate system within it. It is either known independently by the user, or the SRID can sometimes be determined from information stored within the master.dbo.HG_SpatialRef table and MAPINFO.MAPINFO_MAPCATALOG table.

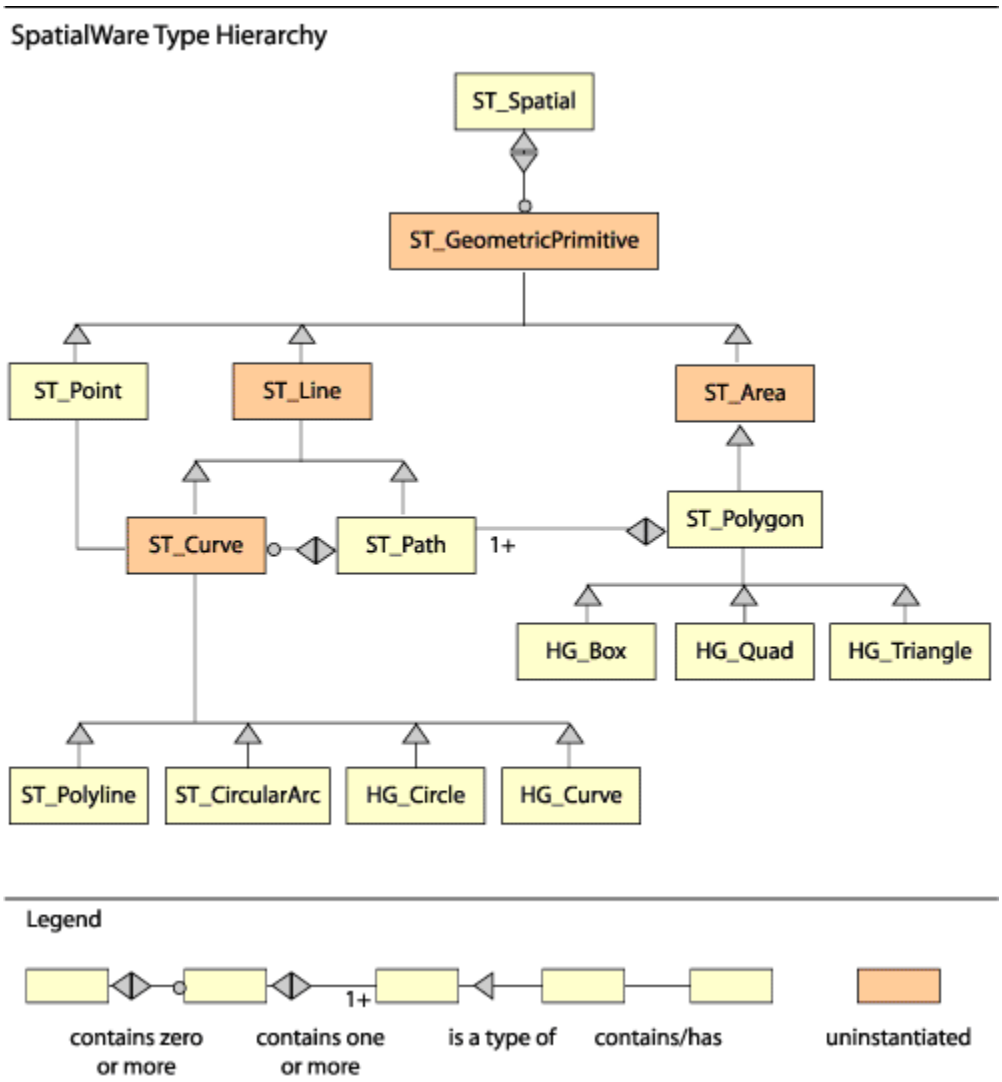
ST_Spatial data can include attributes, such as an assembled flag, line direction, or orientation characteristics, although these are not commonly used. These concepts are not found in SQL Server 2008.

Note that the SpatialWare ST_Spatial type can be dropped at the end of a migration, if it is no longer wanted, by using the SQL Server sp_droptype stored procedure.

Data

SpatialWare stores spatial values in columns of type ST_Spatial, which can be defined in terms of the SpatialWare type hierarchy below. This is based on the OGC Simple Feature Access Specification Geometry class hierarchy (see **Part 1: Common Architecture** and **Part 2: SQL Option Version 1.1.0**), but allows some other extended data, such as true curves, using ST_CircularArc or HG_Circle for example. Some SpatialWare data of these types that are not supported in SQL Server need to migrate to close approximations. For example, HG_Circle needs to be approximated by a linestring, which has straight edges. However, in practice, SpatialWare data typically does not include extended data.

SQL Server 2008 geometry and geography data follow OGC standards too, so in general there is a direct correspondence between this data and SpatialWare data. SpatialWare and SQL Server geometry are more forgiving than SQL Server geography, so there are some data coming from SpatialWare into SQL Server geography that are invalid as geography type, and so must be temporarily entered as geometry data and made valid before becoming geography data. For example, external rings of polygons must be in counterclockwise order for geography, but can be either direction for SpatialWare and SQL Server geometry. Other cases of invalid geographies are polygons with self-intersecting rings, and geographies larger than a hemisphere. Ed Katibah's blog on "Working with Invalid Data and the SQL Server 2008 Geography Data Type" contains useful information for generating valid instances of geography type from invalid ones (see **Part 1** and **Part 1b**). There is also an illustration of how to accomplish this using **SQL Server Spatial Tools project on CodePlex** in Ed Katibah's blog **Part 2**.



In SpatialWare, the stored procedure sp_sw_spatialize_column can be executed, so that if a text representation of an ST_Spatial object is inserted into a database table, a trigger of the form table#iu#spatialcolumn#iu#keycolumn executes to re-write the ST_Spatial in a binary SpatialWare format. It also uses a table of the form table#t#spatialcolumn#t#keycolumn. Triggers and tables of these forms are not created for SQL Server spatial columns of type geometry or geography.

Spatial Indexes

In SpatialWare, spatial columns can be indexed using rtree indexes. This happens when the stored procedure sp_sw_create_rtree is executed. Tables of the form table#m#spatialcolumn#m#keycolumn and table#r#spatialcolumn#r#keycolumn are created by sp_sw_create_rtree to hold rtree index data. In SQL Server 2008, grid indexes are created using the CREATE SPATIAL INDEX statement. They can be optimized for particular sets of spatial data, by specifying a tessellation scheme for the index (see SQL Server documentation for more information). For these indexes, there are no auxiliary tables created of the special forms above.

Neither rtree indexes nor grid indexes are uniformly better than the other, in terms of performance. Therefore, some queries could run faster and others slower, after migration to SQL Server 2008. Performance should be monitored and grid indexes tuned if necessary.

Migrating Data

There are 3 basic techniques for migrating SpatialWare data:

- EasyLoader. A user's spatial data may already be in .TAB file format, or it can be exported into that format using MapInfo Professional. From this format, EasyLoader can be used to import it directly into SQL Server 2008. EasyLoader is available for download.
- MapInfo Professional. As of version 10, this application from PBBI can be used to read data from a SpatialWare database and write it into SQL Server 2008 spatial tables, using "Save As…" functionality.
- OGC well-known binary (WKB) or well-known text (WKT). If you do not have MapInfo Professional, or if you want more control over validation of spatial data which is accepted in SpatialWare but may be considered invalid geographic data in SQL Server, then the OGC standard formats can be used for intermediate stages in migration. SpatialWare data can be saved in WKB format and moved to the SQL Server geometry type. Then it can be made valid and moved to the geography type if necessary. However, it is important to note that the SpatialWare function HG_AsBinary does not retain any Z values in the data.

In the unlikely case that you have added your own triggers, constraints, stored procedures or functions which involve SpatialWare spatial data, don't forget that additional actions will be required to migrate these to SQL Server along with the spatial data itself. This is similar to the effort in migrating applications.

Migrating Applications

Many SpatialWare users will be using SpatialWare only as a repository for spatial data, for use with PBBI/MapInfo applications. As long as these applications support SQL Server 2008 spatial data (MapInfo Professional, MapXtreme 2008), then the migration is straightforward. Make sure that the table MAPINFO.MAPINFO_MAPCATALOG has been updated to reflect the changes in the database, for example the SPATIALTYPE column.

If you have written your own applications for SpatialWare, then the parts of these applications that access SpatialWare must be modified to access SQL Server 2008 data.

Spatial Queries and Functions

In SpatialWare, the fullest set of spatial queries is accessed through the stored procedures sp_spatial_query and sp_nearest. The query parameter to sp_spatial_query supports a SQL syntax that includes a large number of spatial functions, basically an extension of the OGC standard functions. A subset of these functions are also available as user-defined functions (UDFs), and can be used outside of sp_spatial_query.

SQL Server 2008 supports a set of OGC standard functions, which are generally equivalent to corresponding SpatialWare functions. More functions are supported for geometry type than for geography type. There is no directly built-in nearest neighbor functionality in SQL Server 2008, but see the appendix for suggestions about this topic.

Example: migration via OGC WKB to geometry type

Suppose that you have a SpatialWare table T, with spatial column SW_GEOMETRY of type ST_Spatial and key column SW_MEMBER, and with a spatial rtree index on SW_GEOMETRY. Suppose that the SW_GEOMETRY column contains data in a projected coordinate system, say SRID 2029 (UTM17), that you want to migrate to a column SS_GEOMETRY of type geometry, containing 1 valid and 1 invalid geometry, subsequently making the invalid geometry into a valid one, using the SQL Server geometry method MakeValid().

The following example shows:

- equivalent valid points in SpatialWare and SQL Server 2008, where the spatial object in SQL Server is actually a GEOMETRYCOLLECTION.
- equivalent lines with spikes in SpatialWare and SQL Server, which are both stored as lines with spikes, but which is valid with a length calculation allowed in SpatialWare, but which is invalid with no length calculation allowed.

- the effect of MakeValid() in SQL Server to eliminate the spike and produce a MULTILINESTRING for which a length calculation is allowed.
- the slight shifting of points by the MakeValid() method.
- the length of a point in SpatialWare being NULL, while in SQL Server being 0.
- the length of the line with the spike in SpatialWare being different from the length of the multiline with the spike eliminated in SQL Server.

When you execute the statements:

```
set nocount on
create table T(SW_MEMBER int primary key, SW_GEOMETRY
 ST_Spatial)
exec sp_sw_create_rtree 'dbo','T','SW_GEOMETRY','SW_MEMBER'
insert into T values(1, 'st_point(630084,4833438)')
insert into T values(2, 'st_polyline(list{st_point(630084,4833438),
 st_point(630084,4833439),st_point(630085,4833440),
 st_point(630084,4833439),st_point(630085,4833437)})')
 -- has spike
exec sp_spatial_query 'select HG_GetString(SW_GEOMETRY)
 as swgeo from T'
exec sp_spatial_query 'select ST_Length(SW_GEOMETRY)
 as swgeolen from T'
-- start migration
alter table T add SS_GEOMETRY geometry
exec sp_spatial_query 'select
 SW_MEMBER,HG_AsBinary(SW_GEOMETRY)
 as geowkb into T_wkb from T'
update T set SS_GEOMETRY=(select
 Geometry::STGeomFromWKB(geowkb,2029) from T_wkb
 where T_wkb.SW_MEMBER=T.SW_MEMBER)
select SS_GEOMETRY.STAsText() as ssgeo from T
select SS_GEOMETRY.STLength() as ssgeolen from T
update T set SS_GEOMETRY=SS_GEOMETRY.MakeValid()
select SS_GEOMETRY.STAsText() as ssgeo from T
select SS_GEOMETRY.STLength() as ssgeolen from T
drop table T_wkb
exec sp_sw_drop_rtree 'dbo','T','SW_GEOMETRY','SW_MEMBER'
exec sp_sw_despatialize_column
'dbo','T','SW_GEOMETR Y','SW_MEMBER'
drop table T
```

the output is:

```
swgeo
------------------------------------------------------------
ST_Spatial(ST_Point(630084.000000000000,4833438.000000000000))
ST_Spatial(ST_Polyline(LIST{ST_
Point(630084.000000000000,4833438.000000000000),ST_
Point(630084.000000000000,4833439.000000000000),ST_
Point(630085.000000000000,4833440.000000000000),ST_
Point(630084.000000000000,4833439.000000000000),ST_Point(630085.00000000 . . .

swgeolen
----------------------
NULL
6.06449510224598

ssgeo
------------------------------------------------------------
GEOMETRYCOLLECTION (POINT (630084 4833438))
GEOMETRYCOLLECTION (LINESTRING (630084 4833438, 630084 4833439, 630085 4833440,
630084 4833439, 630085 4833437))
```

```
ssgeolen
----------------------
0
Msg 6522, Level 16, State 1, Line 19
A .NET Framework error occurred during execution of user-defined routine or
aggregate "geometry":
System.ArgumentException: 24144: This operation cannot be completed because the
instance is not valid. Use MakeValid to convert the instance to a valid instance.
Note that MakeValid may cause the points of a geometry instance to shift slightly.
System.ArgumentException:
  at Microsoft.SqlServer.Types.SqlGeometry.ThrowIfInvalid()
  at Microsoft.SqlServer.Types.SqlGeometry.STLength()
.

ssgeo
------------------------------------------------------------GEOMETRYCOLLECTION
(POINT (630084 4833438))
MULTILINESTRING ((630085 4833440, 630084 4833439), (630085 4833437, 630084
4833439, 630084 4833438))

ssgeolen
----------------------
0
4.65028153987288
```

A spatial index can be created on the SS_GEOMETRY column using the CREATE SPATIAL INDEX statement.

If you want to remove SpatialWare, after you are satisfied with the migration, as part of that process you should use sp_sw_drop_rtree and sp_sw_despatialize_column on spatial tables.

Example: migration via OGC WKB to geography type

Suppose that you have a SpatialWare table T, with spatial column SW_GEOMETRY of type ST_Spatial and key column SW_MEMBER, and with a spatial rtree index on SW_GEOMETRY. Suppose that the SW_GEOMETRY column contains data in a spherical coordinate system, say SRID 4326 (WGS84), that you want to migrate to a column SS_GEOGRAPHY of type geography, containing 1 valid and 1 invalid geography, subsequently making the invalid geography into a valid one, using the CodePlex function MakeValidGeographyFromGeometry(). Note that a geometry that has been made valid does not always map directly to a valid geography.

The following example shows:

- the line with spike cannot be represented as a geography type in SQL Server 2008.
- the effect of the CodePlex function MakeValidGeographyFromGeometry() in SQL Server to eliminate the spike and produce a MULTILINESTRING for which a length calculation is allowed.
- the slight shifting of points by the MakeValidGeographyFromGeometry() function.
- the length of a point in SpatialWare being NULL, while in SQL Server being 0.
- the length of the line with the spike in SpatialWare ("in degrees") being different from the correct spherical length of the multiline with the spike eliminated in SQL Server.

When you execute the statements:

```
set nocount on
create table T(SW_MEMBER int primary key, SW_GEOMETRY
 ST_Spatial)
exec sp_sw_create_rtree 'dbo','T','SW_GEOMETRY','SW_MEMBER'
insert into T values(1, 'st_point(-79.38,43.63)')
insert into T values(2, 'st_polyline(list{
 st_point(-79.38,43.63), st_point(-79.38,44),
 st_point(-79,44),st_point(-79.38,44),
 st_point(-79,43)})') -- has spike
exec sp_spatial_query 'select HG_GetString(SW_GEOMETRY)
 as swgeo from T'
```

```
exec sp_spatial_query 'select ST_Length(SW_GEOMETRY)
 as swgeolen from T'
-- start migration
alter table T add SS_GEOGRAPHY geography
exec sp_spatial_query 'select SW_MEMBER,
 HG_AsBinary(SW_GEOMETRY) as geowkb
 into T_wkb from T'
update T set SS_GEOGRAPHY=(select
 Geography::STGeomFromWKB(geowkb,4326) from T_wkb
 where T_wkb.SW_MEMBER=T.SW_MEMBER)
alter table T add SS_GEOMETRY geometry
update T set SS_GEOMETRY=(select
 Geometry::STGeomFromWKB(geowkb,4326) from T_wkb
 where T_wkb.SW_MEMBER=T.SW_MEMBER)
update T set SS_GEOGRAPHY=
 dbo.MakeValidGeographyFromGeometry(SS_GEOMETRY)
select SS_GEOGRAPHY.STAsText() as ssgeo from T
select SS_GEOGRAPHY.STLength() as ssgeolen from T
drop table T_wkb
exec sp_sw_drop_rtree 'dbo','T','SW_GEOMETRY','SW_MEMBER'
exec sp_sw_despatialize_column
 'dbo','T','SW_GEOMETRY','SW_MEMBER'
drop table T
```

the output is:

```
swgeo
------------------------------------------------------------
ST_Spatial(ST_Point(-79.380000000000,43.630000000000))
ST_Spatial(ST_Polyline(LIST{
ST_Point(-79.380000000000,43.630000000000),
ST_Point(-79.380000000000,44.000000000000),
ST_Point(-79.000000000000,44.000000000000),
ST_Point(-79.380000000000,44.000000000000),
ST_Point(-79.000000000000,43.000000000000)}),
ST_Point(-7 ...

swgeolen
----------------------
NULL
2.19976632962529

Msg 6522, Level 16, State 1, Line 16
A .NET Framework error occurred during execution of user-defined routine or
aggregate "geography":
System.ArgumentException: 24200: The specified input does not represent a valid
geography instance.
System.ArgumentException:
   at
Microsoft.SqlServer.Types.SqlGeography.ConstructGeographyFromUserInput(GeoData g,
Int32 srid)
   at Microsoft.SqlServer.Types.SqlGeography.GeographyFromBinary(OpenGisType
type, SqlBytes binary, Int32 srid)
   at Microsoft.SqlServer.Types.SqlGeography.STGeomFromWKB(SqlBytes wkbGeometry,
Int32 srid)
.
The statement has been terminated.
ssgeo
------------------------------------------------------------
POINT (-79.379999999999981 43.63)
MULTILINESTRING ((-78.999999935260576 44.000000148575907,
-79.379999912170916 44.000000181796949,
-79.000000009630512 42.999999892974941),
```

```
(-79.3799999766662 43.630000145834515,
 -79.379999912170916 44.000000181796949))

ssgeolen
----------------------
0
186862.995059838
```

A spatial index can be created on the SS_GEOGRAPHY column using the CREATE SPATIAL INDEX statement.

Example: making an invalid geography polygon valid

The outer ring of a polygon must be counterclockwise in order to be valid (inner rings must be clockwise). The CodePlex function IsValidGeographyFromGeometry() detects and the CodePlex function MakeValidGeographyFromGeometry() fixes various geography validation problems, including this one. This specific problem can also be fixed using one of the ideas in Ed Katibah's blog, sometimes with different shifting of points.

The following example shows:

- the polygon with clockwise outer ring cannot be represented as a geography type in SQL Server 2008.
- 2 methods of making such a polygon valid, each producing different counterclockwise outer rings

When you execute the statements:

```
DECLARE @geog geography;
SET @geog = geography::STGeomCollFromText
 ('GEOMETRYCOLLECTION(POLYGON((-79.38 43.63,-79.38 44,
 -79 44, -78  43,-79 42,-79.38 43.63)))',4326);
DECLARE @geom geometry;
SET @geom = geometry::STGeomCollFromText
 ('GEOMETRYCOLLECTION(POLYGON((-79.38 43.63,-79.38 44,
 -79 44, -78 43,-79 42,-79.38 43.63)))',4326);
SELECT @geom.ToString() as geom;
SELECT dbo.MakeValidGeographyFromGeometry(@geom).ToString()
 as  geomvalid1
SELECT
 @geom.MakeValid().STUnion(@geom.STStartPoint()).ToString()
 as geomvalid2
```

the output is:

```
Msg 6522, Level 16, State 1, Line 2
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)
   at Microsoft.SqlServer.Types.SqlGeography.STGeomCollFromText(SqlChars
geometryCollectionTaggedText, Int32 srid)
.
geom
----------------------------------------------------------
GEOMETRYCOLLECTION (POLYGON ((-79.38 43.63, -79.38 44,
```

```
-79 44, -78 43, -79 42, -79.38 43.63)))

geomvalid1
------------------------------------------------------------
POLYGON ((-79.37999998599949 44.000000022331662,
-79.379999986050734 43.630000023540191,
-78.99999999411132 41.999999984403651,
-77.9999999993165 43.000000000990639,
-79.0000000016003 43.999999988684472,
-79.37999998599949 44.000000022331662))

geomvalid2
------------------------------------------------------------
POLYGON ((-79 42, -78 43, -79 44, -79.379999995231628 44,
-79.379999995231628 43.629999995231628, -79 42))
```

APPENDIX: SQL Server 2008 Functionality for SpatialWare Stored Procedures and Functions

Stored Procedures

sp_spatial_query(query,useRtree) will become a SQL Server statement, similar to the query parameter, but changed to provide the correct syntax and spatial methods (see functions below).

sp_nearest(owner,table,spatialcolumn,keycolumn,geo,num,maxdist,query,units). **Isaac Kunen's 2008 blog on nearest neighbor** shows a sample query:

```
DECLARE @start FLOAT = 1000;
WITH NearestPoints AS
 SELECT TOP(1) WITH TIES *, T.g.STDistance(@x) AS dist
 FROM Numbers
 JOIN T WITH(INDEX(spatial_index))
 ON T.g.STDistance(@x) < @start*POWER(2,Numbers.n)
  ORDER BY n)
SELECT TOP(1) * FROM NearestPoints ORDER BY n, dist
```

A **web page about k-nearest neighbors** quotes **Inside Microsoft® SQL Server® 2008: T-SQL Programming Section 14.8.4** to show code to retrieve 10 nearest points:

```
DECLARE @input GEOGRAPHY = 'POINT (-147 61)';
DECLARE @start FLOAT = 1000;
WITH NearestNeighbor AS(
 SELECT TOP 10 WITH TIES *,
  b.GEOG.STDistance(@input) AS dist
 FROM Nums n
 JOIN GeoNames b WITH(INDEX(geog_hhhh_16_sidx))
  -- index hint
 ON b.GEOG.STDistance(@input) < @start*POWER(CAST(2 AS
  FLOAT),n.n)
 AND b.GEOG.STDistance(@input) >=
  CASE WHEN n = 1 THEN 0 ELSE @start*POWER(CAST(2 AS
   FLOAT),n.n-1) END
 WHERE n <= 20
 ORDER BY n
)
SELECT TOP 10 geonameid, name, feature_code, admin1_code,
 dist FROM NearestNeighbor ORDER BY n, dist;
```

Table-valued functions

table_name#p#spatial_column#p#key_column (geometry, predicate, truth) – use the STIntersects method and STEnvelope geometry method.

table_name#s#spatial_column#s#key_column (geometry, predicate, truth) – use the STIntersects method.

table_name#n#spatial_column#n#key_column (geometry, number, ties_tolerance, max_distance, points) is similar to sp_nearest.

fn_sw_udfs(owner, table_name, spatial_column) is not applicable to SQL Server.

Scalar-valued user-defined functions (UDFs)

Some of the following SpatialWare UDFs show suggested SQL Server methods that can be used when migrating. Functions marked as (3D) generally do not have SQL Server 2008 equivalents, but are not commonly used in SpatialWare.

Constructor functions

HG_Box (a polygon) – use STPolyFromText

HG_Circle (a polygon) – not applicable to SQL Server, must be approximated by a polygon with straight edges

ST_Point – use STPointFromText

General functions

HG_Version – not applicable to SQL Server

Measurement functions

HG_Azimuth

HG_Azimuth_2pts

HG_Distance – use STDistance

HG_Height

HG_Separation – use STDistance

HG_Slope – (3D)

HG_Slope_2pts – (3D)

HG_Slope_Avg – (3D)

HG_Slope_Max – (3D)

HG_Slope_Min – (3D)

HG_SphericalArea – geography.STArea

HG_SphericalDist – geography.STDistance

HG_SphericalDistance – geography.STDistance

HG_SphericalLength – geography.STLength

HG_SphericalPerimeter – geography.RingN(1), STLength

HG_Width

ST_Area – STArea

ST_Length – STLength

ST_Length_3D – (3D)

ST_Perimeter – geometry.STBoundary, STLength

ST_Perimeter_3D – (3D)

Observer functions

HG_BeginPoint – STStartPoint
HG_Cen_X – geometry.STCentroid or STPointOnSurface, STX
HG_Cen_Y – geometry.STCentroid or STPointOnSurface, STY
HG_Cen_Z – geometry.STCentroid or STPointOnSurface, Z
HG_End_Arc_Rot – not applicable to SQL Server
HG_End_Point – ST_EndPoint
HG_End_Tangent – not applicable to SQL Server
HG_Llb_X – geometry.STEnvelope, STX, …
HG_Llb_Y – geometry.STEnvelope, STY, …
HG_Llb_Z
HG_Ncoords – STNumPoints
HG_Ncurves – not applicable to SQL Server
HG_Nitems – STNumGeometries
HG_Npaths
HG_Npoints
HG_Npolygons
HG_Nsubcurves – not applicable to SQL Server
HG_Ori_Rotation – not applicable to SQL Server
HG_Ori_X – not applicable to SQL Server
HG_Ori_Y – not applicable to SQL Server
HG_Ori_Z – not applicable to SQL Server
HG_Pointdyn_Ori_X – not applicable to SQL Server
HG_Pointdyn_Ori_Y – not applicable to SQL Server
HG_Pointdyn_Rot – not applicable to SQL Server
HG_Pointdyn_Xscale – not applicable to SQL Server
HG_Pointdyn_Yscale – not applicable to SQL Server
HG_PointN – STPointN
HG_Radians – not applicable for SQL Server
HG_Start_Arc_Rot – not applicable to SQL Server
HG_Start_Tangent – not applicable to SQL Server
HG_SubCurve – not applicable to SQL server
HG_Urt_X – geometry.STEnvelope, STX, …
HG_Urt_Y – geometry.STEnvelope, STY, …
HG_Urt_Z
ST_X – geometry.STX; geography.Long
ST_Y – geometry.STY; geography.Lat
ST_Z – Z

Spatial functions (Indexable)

HG_Centroid – geometry.STCentroid or STPointOnSurface
HG_Envelope – geometry.STEnvelope
HG_SphericalBuffer – geography.STBuffer

Spatial predicates (Indexable)

HG_At_End_Of
HG_At_Start_Of
HG_Connected_To
HG_Identical
HG_SphericalCircle
ST_Adjacent_To – geometry.STTouches
ST_Contained_By – geometry.STContains (reversed)
ST_Contains – geometry.STContains
ST_Equals – STEquals
ST_Meets – geometry.STTouches
ST_Outside – STDisjoint
ST_Overlaps – STIntersects
ST_Within – geometry.STContains (reversed)

Spatial predicates (Non-indexable)

HG_Above – (3D)
HG_Assembled – not applicable to SQL Server
HG_Below – (3D)
HG_Is_Box
HG_Is_Circle – not applicable to SQL Server
HG_Is_CircularArc – not applicable to SQL Server
HG_Is_Closed – STIsClosed
HG_Is_Contiguous
HG_Is_Forward – not applicable to SQL Server
HG_Is_HG_Curve – not applicable to SQL Server
HG_Is_Invalid – geometry.STIsValid (reversed)
HG_Is_Nulldir– not applicable to SQL Server
HG_Is_Path
HG_Is_Point
HG_Is_Polygon
HG_Is_Polyline
HG_Is_Quad
HG_Is_Reverse – not applicable to SQL Server
HG_Is_Triangle
HG_Is_Valid – geometry.STIsValid
HG_Level – (3D)

Other sp_spatial_query functions

Cast functions

HG_AsBinary – STAsBinary
HG_AsText – STAsText
HG_GeometryFromBinary – STGeomFromWKB
HG_GeometryFromText – STGeomFromText

Aggregate functions

HG_Aggspatial

HG_Aggunion – (CodePlex) GeographyUnionAggregate

HG_Aggintersection

HG_Aggconvex_Hull

HG_Bounding_Box – (CodePlex) GeometryEnvelopeAggregate

Coordinate functions

HG_CSTransform – (CodePlex) SqlProjection type, and look at projection_example.sql there

General functions

HG_GetString – not applicable to SQL Server

Observer functions

HG_Center_Point

HG_Corner

HG_Curve

HG_End_Tangent_P – not applicable to SQL Server

HG_Expanded – not applicable to SQL Server

HG_Exterior_Path – geometry.STBoundary; geography.RingN(1)

HG_Extract_At

HG_GeometryN

HG_Interior_Path

HG_Point

HG_Start_Tangent_P – not applicable to SQL Server

Spatial functions

HG_Affine – (CodePlex) AffineTransform type
HG_Affine3D – (3D)
HG_As_Curves
HG_As_Paths
HG_As_Points
HG_Center_In
HG_Center_In_3D – (3D)
HG_Clean
HG_Clean_I
HG_Clean_S
HG_Combine – STUnion
HG_Connect
HG_Convex_Hull – geometry.STConvexHull
HG_Difference – STDifference
HG_End_Of
HG_Erase – STDifference
HG_Erase_Outside
HG_Filter – Reduce
HG_Filter_Curves
HG_Filter_Paths
HG_Filter_Points
HG_Filter_Polygons
HG_Intersect_In – STIntersection
HG_LL_Circle
HG_Spherical_Circle
HG_Split
HG_Start_Of
HG_Sym_Difference – STSymDifference
HG_Union – STUnion
ST_Adjacent
ST_Buffer – STBuffer
ST_Contain
ST_Overlap – STIntersection
ST_Transform
ST_Transform3D – (3D)

Spatial predicate functions

HG_Is_Curve
HG_Is_Empty
HG_Is_ZNull
ST_Not_Equals – STEquals (reversed)