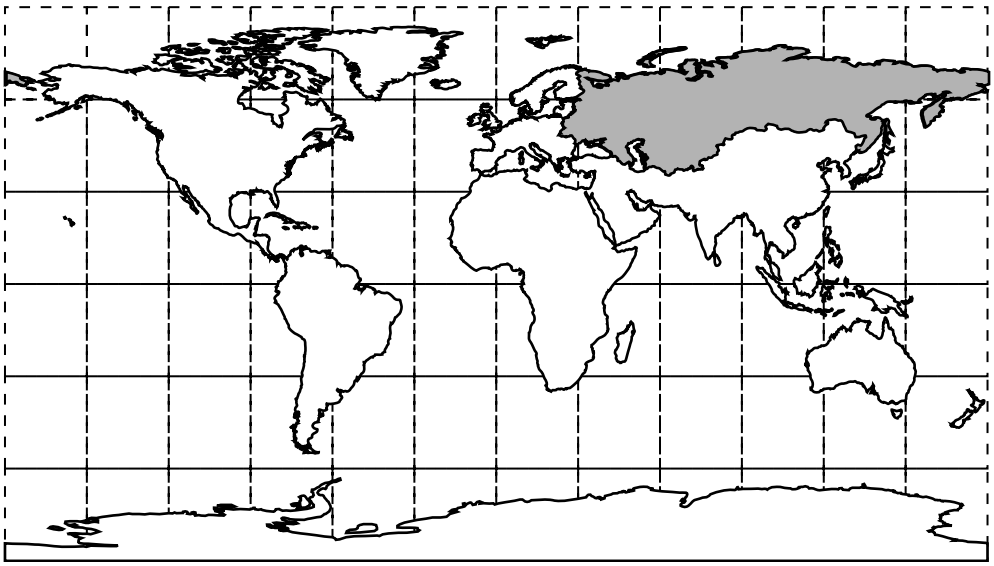

Figure 2-8. *Polygon geometry representing Russia crossing edges of a projection in the geometry datatype*

If you expect to deal with situations where geometries will have to cross the edges of a projected map, then the `geography` datatype would be a better choice in which to store your data.

Presentation

Since the `geography` datatype operates on a three-dimensional model of the earth, if you want to present the results of any `geography` data on a map, you first need to project them. As previously discussed, this introduces distortion. In the example given in Figure 2-7, although the `geography` datatype accurately works out the shortest straight-line route connecting two points, if we were to display this result on a projected map, this "straight" line may appear distorted and curved. The exact effect of this distortion will differ depending on the particular properties of the map projection used.

Figure 2-9 depicts the same route between Vancouver and Tokyo as shown in Figure 2-7, but projected onto a map using the Mercator method, centered on the meridian at 150° longitude. Notice how the route calculated using the `geography` datatype, which appeared to be a direct path when plotted on a globe, appears to curve upward on a flat map due to the effects of distortion caused by projection.

■ **Note** You will often see great elliptic arcs—the shortest path between two points in the `geography` datatype—depicted as curved lines (such as shown in Figure 2-9) on maps used by airlines to illustrate the approximate route taken by airplanes between destinations.

Conversely, since the geometry datatype is based on data that has already been projected onto a plane, no further distortion needs to be introduced to express the results on a map—"straight" lines in the geometry datatype remain straight when drawn on a map (provided that the map is projected using the same projection as the spatial reference system from which the points were obtained).

If you will be using data to represent straight lines between points, and you want those lines to remain straight when displayed on a map, then you should choose the geometry datatype, and select an appropriate spatial reference corresponding to the map on which those results will be displayed.

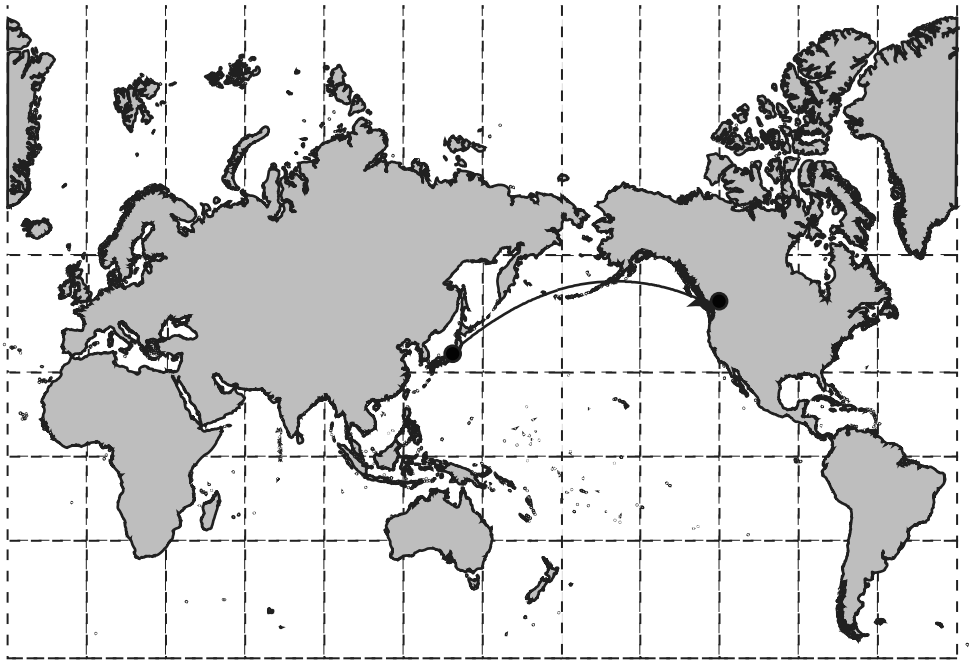

Figure 2-9. *The shortest route between Tokyo and Vancouver using the geography datatype, as projected using the Mercator projection*

Performance

Performing spherical computations uses more computing resources than does performing Cartesian computations. As a result, performing spatial calculations on the ellipsoidal model used by the geography datatype may take longer to compute than the equivalent operations using the geometry datatype. This will only affect methods where the geography datatype has to calculate metrics based on the geodetic model of the earth, such as distances, lengths, or areas of geometrical objects. When you use any methods that return properties of objects that do not need to take account of the model of the earth, such as counting the number of points in an object or describing the type of geometry used to represent a feature, there is no difference in performance between the geography and geometry types.

Standards Compliance

According to its web site, http://www.opengeospatial.org/, the Open Geospatial Consortium (OGC) is "a non-profit, international, voluntary consensus standards organization that is leading the development of standards for geospatial and location based services." The OGC administers a number of industry-wide standards for dealing with spatial data. By conforming to these standards, different systems can ensure core levels of common functionality, which means that spatial information can be more easily shared between different vendors and systems. In October 2007, Microsoft joined the OGC as a principal member, and the spatial datatypes implemented in SQL Server 2008 are largely based on the standards defined by the OGC:

The geometry datatype conforms to the OGC Simple Features for SQL Specification v1.1.0 (http://www.opengeospatial.org/standards/sfs) and implements all the required methods to meet that standard.

The geography datatype implements many of the same methods as the geometry datatype, although it does not completely conform to the required OGC standards.

As such, if it is important to you to use spatial methods in SQL Server 2008 that adhere to accepted OGC standards, or if you want to ensure compatibility with another system based on those standards, you should use the geometry datatype.

How Spatial Data Is Stored

geometry and geography are both variable-length datatypes. This means that, in contrast to a fixed-length datatype such as int or datetime, the actual amount of storage required for an item of spatial data depends on the complexity of the object that the data describes.

SQL Server 2008 stores the information contained in the geography and geometry datatypes as a stream of binary data. Each stream begins with a header section that defines basic information such as the type of geometry being described, the spatial reference system used, and the overall number of points in the object. This header is immediately followed by the coordinate values of each x and y (or longitude and latitude) coordinate in the geometry, represented in 8-byte binary format. The more points that an object has in its definition, the longer this binary stream will be, and therefore the more storage space that it will require.

■ **Note** SQL Server 2008 stores each coordinate as a double-precision binary floating-point number, following the specifications of the IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754-2008). This format is able to store floating-point numbers with 15 significant digits of precision—which is generally sufficient to describe any position with submillimeter accuracy.

The following list gives examples of the storage space required to store some common items of spatial data:

- A Point geometry defined with only two coordinates always occupies 22 bytes of storage space.
- A LineString between two points, containing the minimum of four coordinates (latitude and longitude, or x and y values of the start point and the end point), occupies 38 bytes.
- A Polygon geometry occupies a variable amount of space depending on the number of points that make up that Polygon. Any interior rings defined by a Polygon also increase the space required to store that Polygon.

There is no specific maximum size for the data storage space used by a `geometry` or `geography` object. However, SQL Server 2008 has an overall restriction on any kind of large object, which is limited to a size of $2^{31} - 1$ bytes. This is the same limit as is applied to datatypes such as `varbinary(max)` and `varchar(max)`, and equates to approximately 2GB for each individual item of data. You would need to store a very complex geometry object to exceed this limit! If necessary, remember that complex geometries can be broken down into a number of individual objects that each fit within the allowed size.

■ **Tip** You can use the T-SQL `DATALENGTH` function to find out the number of bytes used to store the value of any item of `geometry` or `geography` data.

Converting Between Datatypes

Remember that the `geometry` datatype can be used to store geometries defined using a projected coordinate system, a geographic coordinate system, or a naturally planar coordinate system. The `geography` datatype, in contrast, can only be used to store geometries defined using a geographic coordinate system. Since both spatial datatypes can be used to store geographic coordinate data, any item of data stored using the `geography` datatype can also be expressed using the `geometry` datatype instead. However, you cannot simply use the `CAST` or `CONVERT` function to convert data between the two types. If you try to do so, by running the query

```
DECLARE @geog geography
SET @geog = geography::STGeomFromText('POINT(23 32)',4326)
SELECT CAST(@geog AS geometry)
```

you will receive the following error:

```
Msg 529, Level 16, State 2, Line 5
Explicit conversion from data type sys.geography to sys.geometry is not allowed.
```

Notice that the error states that explicit conversion is not *allowed*—this is a deliberate restriction imposed by SQL Server to ensure that you understand the implications of working with each datatype, and that you do not casually swap data between them. There are occasions, however, when it is helpful to be able to convert data between datatypes, since there are functions available for the `geometry` datatype that are not available when working with data stored using

the geography datatype. In order to convert from geography to geometry, we can instead take advantage of the fact that the value of an item of data in either datatype can be represented as a binary stream. In the following example, the value of the geometry variable, @geom, is set based on the binary stream representation of the geography variable @geog:

```
DECLARE @geog geography
SET @geog = geography::STGeomFromText('POINT(23 32)',4326)
DECLARE @geom geometry
SET @geom = geometry::STGeomFromWKB(@geog.STAsBinary(), @geog.STSrid)
```

While every item of geography data can be expressed using the geometry datatype, not every item of geometry data can be expressed as an item of geography data. In order to convert a value from the geometry datatype to the geography datatype, the x and y coordinate values of the existing geometry instance *must* represent longitude and latitude coordinates taken from a supported geodetic spatial reference system. The data must also conform to all the other requirements of the geography datatype, such as ring orientation and size limitations. If these conditions are all met, you can create a geography instance, @geog, from the binary representation of a geometry instance, @geom, as follows:

```
DECLARE @geom geometry
SET @geom = geometry::STGeomFromText('POINT(23 32)',4326)
DECLARE @geog geography
SET @geog = geography::STGeomFromWKB(@geom.STAsBinary(), @geom.STSrid)
```

If, however, you are storing northing and easting coordinates from a projected system, or other nongeodetic data, that data can only be stored using the geometry datatype, and cannot be converted to the geography datatype.

■ **Caution** There are relatively few cases in which an item of spatial data can be converted between the geometry and the geography datatypes, and even when it is technically possible, it does not always make logical sense to do so. If you find the need to convert between datatypes, it might be an indication that the design of your spatial data is incorrect.

Spatially Enabling Your Tables

Having chosen the appropriate spatial datatype, you must add a column of that datatype to the SQL Server table in which you plan to store your spatial data. There are two cases to consider: either you are creating a new table, or you are adding a column to an existing table.

Creating a New Table

There are no special attributes or features required to enable spatial data to be stored in a SQL Server database—all that is required is a table that contains at least one geography or geometry column. Since both the geography and geometry column types are already registered, system-defined datatypes, you can use a normal T-SQL CREATE TABLE statement to create a table containing a field of datatype geography or geometry, as follows:

```
CREATE TABLE [dbo].[Cities] (
    [CityName] [varchar](255) NOT NULL,
    [CityLocation] [geometry] NOT NULL
)
GO
```

This example creates a table containing two columns—`CityName`, which can hold a 255-character variable-length string, and `CityLocation`, which can be used to hold the spatial data relating to that city, using the `geometry` datatype.

Adding to an Existing Table

One of the benefits of using the spatial features of SQL Server 2008 is that new `geometry` or `geography` columns can be added to existing tables, enabling spatial information to be seamlessly integrated alongside existing items of data.

Suppose that we have an existing table, `Customer`, that contains the following fields of customer information:

```
CustomerID int,
FirstName varchar(50),
Surname varchar (50),
Address varchar (255),
Postcode varchar (10),
Country varchar(32)
```

Now suppose that we want to add an additional spatial field to this table to record the location of each customer's address. No problem—`geography` and `geometry` fields can be added to existing tables just like any other by using an `ALTER TABLE` statement, as follows:

```
ALTER TABLE [dbo].[Customer]
ADD CustomerLocation geography
GO
```

By extending the table in this way, we have enabled the possibility of using spatial methods in conjunction with our existing customer data, to find answers to questions such as how many customers we have within a certain area, and how far a particular customer lives from their closest store.

Enforcing a Common SRID

The coordinates defining a point only make sense in the context of a given spatial reference system. Although SQL Server 2008 allows you to store data objects from different spatial references in the same column, when performing functions, all of the spatial data items must be defined using the same spatial reference (i.e., have the same SRID). Trying to perform a calculation using coordinates from different systems can be compared to trying to add amounts of money denominated in different currencies:

25 Dollars + 12 Pounds = 37 . . . what?!

To prevent you from accidentally performing a nonsensical calculation such as this, SQL Server will not allow you to perform any operations using data defined in different spatial references. The result of any method that attempts to do so will be NULL.

If you know that you will only be storing data based on one particular spatial reference system in a column, you can enforce the same SRID on all items in that column by adding a *constraint*. You can do this by using the ADD CONSTRAINT statement, as follows:

```
ALTER TABLE [dbo].[Customer]
ADD CONSTRAINT [enforce_srid_geographycolumn]
CHECK (CustomerLocation.STSrid = 4326)
GO
```

This example creates a constraint called enforce_srid_geographycolumn, which ensures that every spatial object inserted into the CustomerLocation field of the Customer table is defined using the SRID 4326. If you attempt to insert a geography object based on a different SRID, you will receive the following error:

```
Msg 547, Level 16, State 0, Line 1
The INSERT statement conflicted with the CHECK constraint
"enforce_srid_geographycolumn".
The conflict occurred in database "Spatial",
table "dbo.Customers", column 'CustomerLocation'.
The statement has been terminated.
```

As a result, you will be able to perform calculations using any two items of data from this column, knowing that they will be defined based on the same spatial reference system.

Summary

In this chapter, you learned how SQL Server 2008 implements spatial data using the geometry and geography datatypes:

- The geography datatype uses geodetic spatial data, accounting for the curvature of the earth.
- The geometry datatype uses planar spatial data, in which all points lie on a flat plane.
- There are a number of factors that you should consider when choosing the appropriate datatype for a given application, including precision, presentation, standards compliance, and consistency with any existing sources of data.
- Internally, SQL Server stores spatial data represented as a stream of binary values.
- You can add spatial columns to an existing table using ALTER TABLE, or create a new spatially enabled table using the CREATE TABLE T-SQL syntax.
- You can add a constraint to a column of spatial data to ensure that only data of a certain SRID can be inserted into that column.