Spatial Databases

Database II

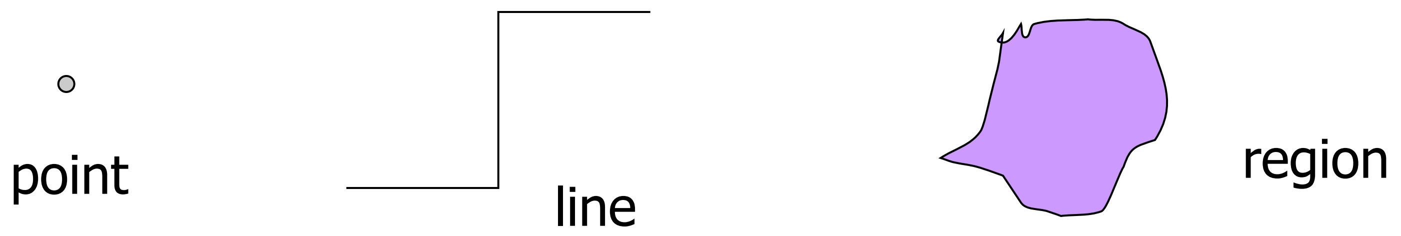
Chea Daly

Spatial Databases

- A spatial database is a database that is optimized for storing and querying data that represents objects defined in a geometric space.
- Most spatial databases allow the representation of simple geometric objects such as points, lines and polygons. Some spatial databases handle more complex structures such as 3D and 4D objects.
- Spatial databases management system (SDBMS)

Spatial Data

- Spatial data represents information about the physical location and shape of geometries such as points, lines, and polygons / regions.

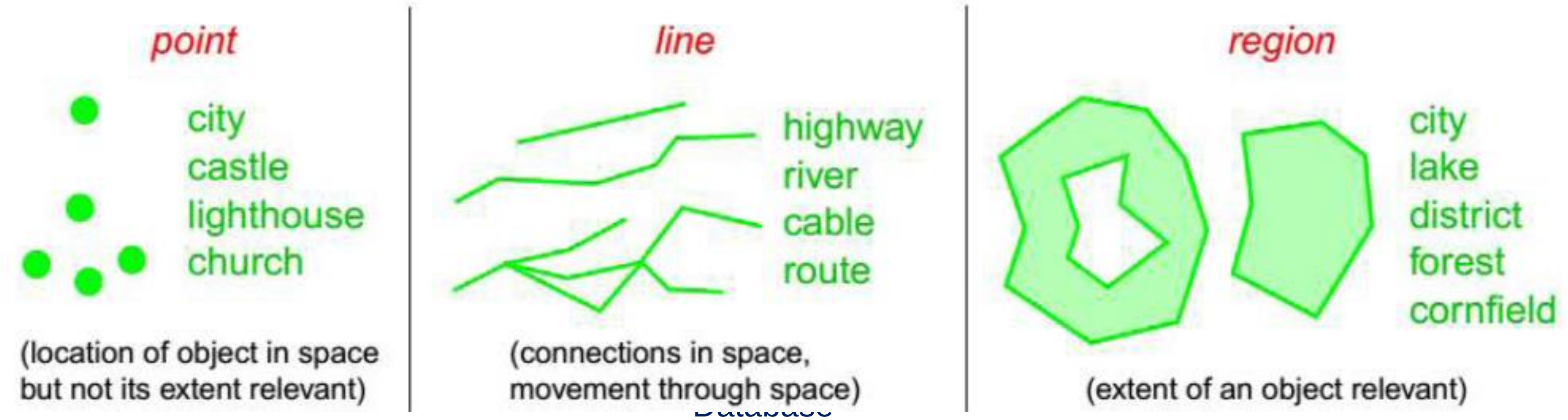


Geometry Types

- The basic geometries are Points, Lines and Polygons. Other geometries are an extended version of these fundamental geometry types.

Geometry Types

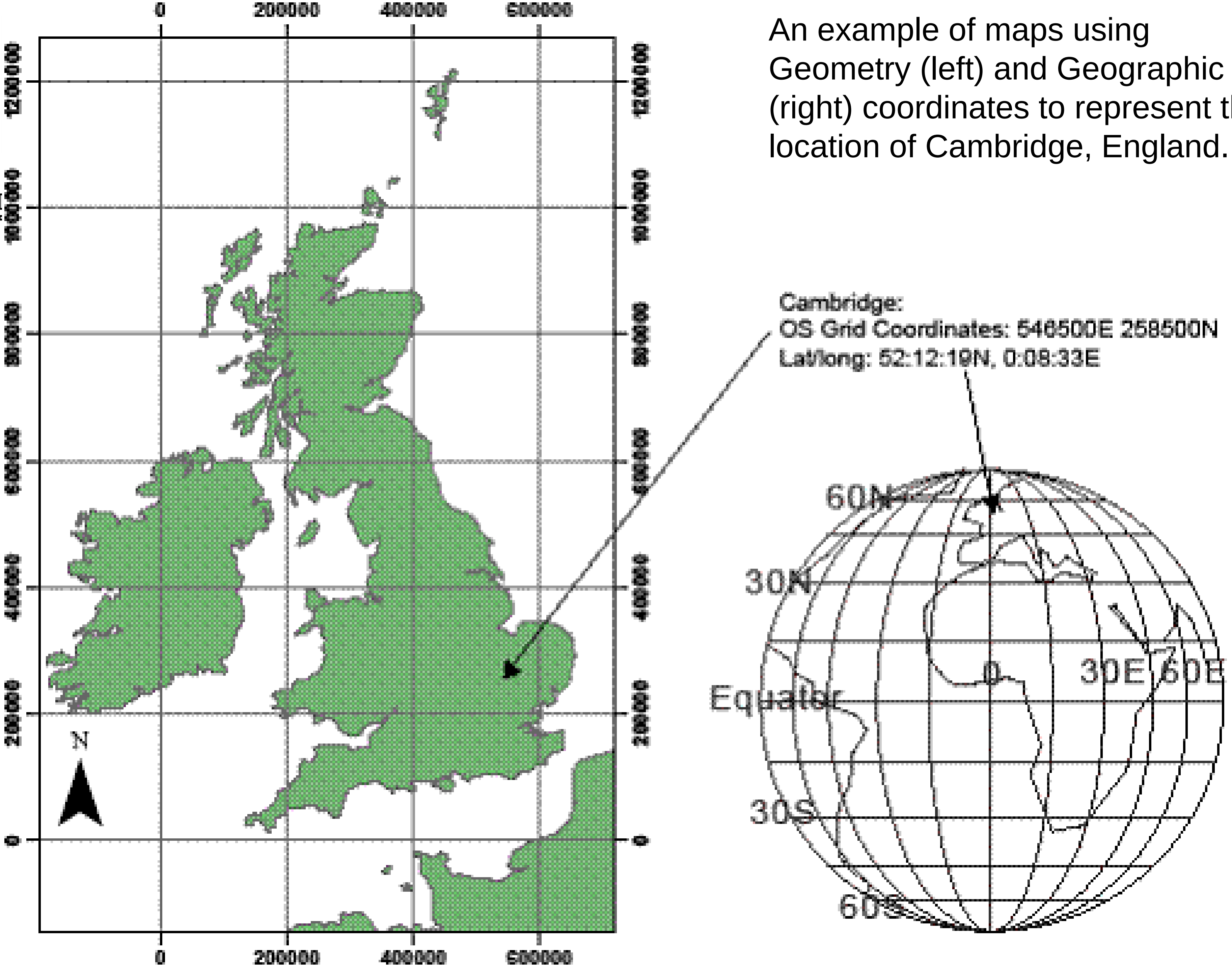
- **Point**: object represented only by its location in space, and not its extent, e.g. a city may be modeled as a point in a model describing a large geographic area (a large scale map).
- **Line**: (actually a curve or polyline): representation of moving through or connections in space, e.g. road, river.
- **Polygon / Region**: representation of an extent in 2D-space, e.g. lake, city. A region may have holes and may also consist of several disjoint pieces.


point
city
castle
lighthouse
church
(location of object in space but not its extent relevant)
line
highway
river
cable
route
(connections in space, movement through space)
region
city
lake
district
forest
cornfield
(extent of an object relevant)

Spatial Data Types

- SQL Server supports two spatial data types:
  - The **geometry** type represents data in a Euclidean (flat) coordinate system. The data type is often used to store the X and Y coordinates that represent geometry objects.
  - The **geography** type represents data in a round-earth coordinate system. The data type is used to store the latitude and longitude coordinates that represent geography objects.

An example of maps using Geometry (left) and Geographic (right) coordinates to represent the location of Cambridge, England.



Spatial Data Types

- Each of the spatial data types has its own use.
  - The Geography type is often used to store an application’s GPS (Global Positioning System) data.
  - the Geometry type is often used to map a three-dimensional object, such as a building.
- The two types often behave similarly. There are some key differences in how the data is stored and manipulated.

Spatial Data Objects

Geometry and geography types are divided into:

- Simple types:
  - Point
  - LineString
  - CircularString
  - CompoundCurve
  - Polygon
  - CurvePolygon
- Collection types:
  - MultiPoint
  - MultiLineString
  - MultiPolygon
  - GeometryCollection

Point

Geography Data Type

- The Point (Lat, Long) represents a single location.
- Latitude specifies the north–south position on the surface of Earth. Values always lie in [-90, 90] degrees.
- Longitude specifies the east-west position on the surface of Earth. Values always lie in [-180, 180] degrees.

Geometry Data Type

- The Point (X, Y) represents a single location where X represents the X-coordinate of the Point being generated and Y represents the Y-coordinate of the Point being generated.

Spatial Reference Identifier

- Each spatial instance has a SRID.
- SRID corresponds to a spatial reference system based on the specific ellipsoid used for either flat-earth mapping or round-earth mapping.
- A column may contain objects with different SRID, but we cannot perform operations between objects with different SRID (not based on the same unit of measurement, datum, and projection).
- For geometrical data, the implicit value for SRID is zero
- For the geographical data, it is 4326 (it is also used by Google Maps API).

Point

- Example: creates a geometry point with an SRID of 0.

1).

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

2).

```
DECLARE @g geometry;
SET @g = 'POINT (3 4)';
```

3).

```
DECLARE @g geometry  = 'POINT (3 4)';
```

- Example: returns the X and Y values of a point, `@g`

```
SELECT @g.STX;
SELECT @g.STY;
```

Point

- Example: creates a geography point with an SRID of 4326.

1).

```
DECLARE @g geography;
SET @g = geography::STGeomFromText('POINT(-122.34900 47.65100)', 4326);
```

2).

```
DECLARE @g geography;
SET @g = 'POINT(-122.34900 47.65100)';
```

3).

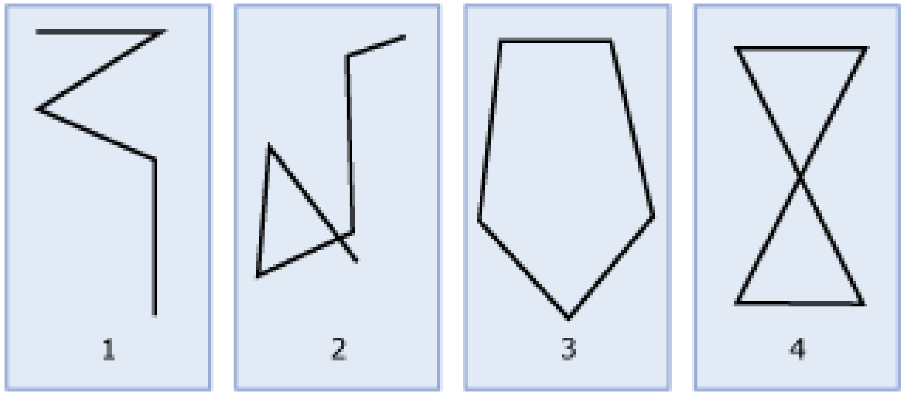
```
DECLARE @g geography = 'POINT(-122.34900 47.65100)';
```

- Example: returns the Lat and Long values of a point, `@g`

```
SELECT @g.Lat;
SELECT @g.Long;
```

LineString

- A LineString is a one-dimensional object representing a sequence of points and the line segments connecting them.
- The drawing below shows valid LineString instances:



#1 is a simple, nonclosed LineString instance.

#2 is a nonsimple, nonclosed LineString instance.

#3 is a closed, simple LineString instance, and therefore is a ring.

#4 is a closed, nonsimple LineString instance, and therefore is not a ring.

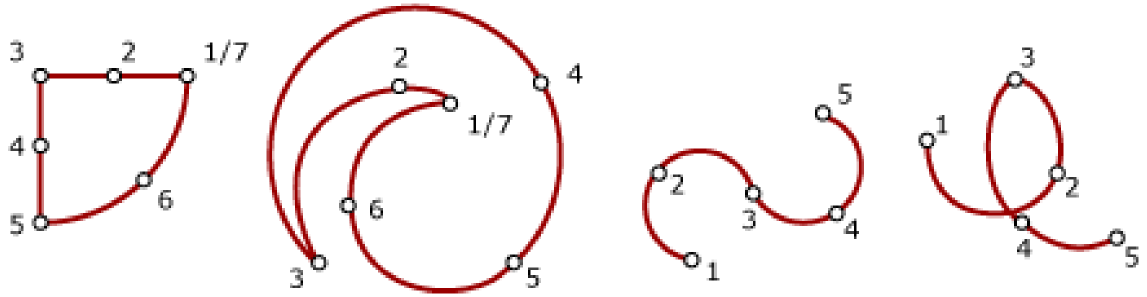
LineString

- Example: create a geometry LineString instance with three points and an SRID of 0, then calculate its length.

```
DECLARE @g geometry = 'LINESTRING(1 1, 2 4, 3 9)';
SELECT @g.STLength() AS LineLength
```

CircularString

- A CircularString is a collection of zero or more continuous circular arc segments which are curved segments defined by three points in a two-dimensional plane.
  - If all three points of a circular arc segment are collinear, the arc segment is treated as a line segment.
- The drawing below shows valid CircularString instances:



CircularString

- Example: create an empty geometry CircularString instance

```
DECLARE @g geometry = 'CIRCULARSTRING EMPTY';
```

- Example: create a geometry CircularString instance with one circular arc segment

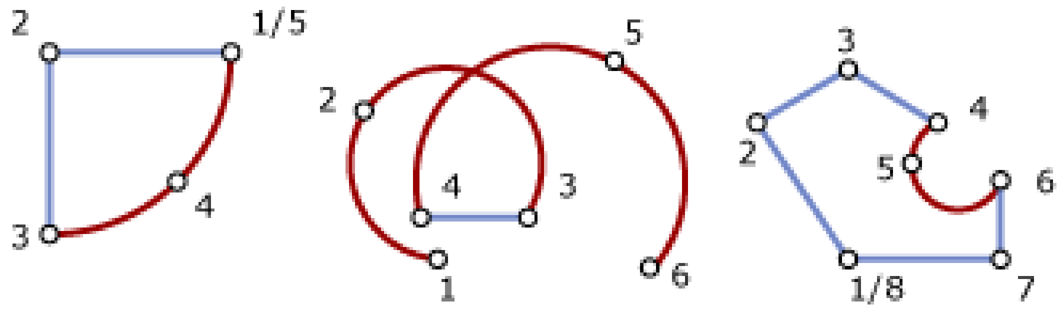
```
DECLARE @g geometry = 'CIRCULARSTRING(2 0, 1 1, 0 0)';
```

- Example: create a geometry CircularString instance with more than one circular arc segment (full circle)

```
DECLARE @g geometry = 'CIRCULARSTRING(2 1, 1 2, 0 1, 1 0, 2 1)';
SELECT @g.STLength() AS Circumference
```

CompoundCurve

- A CompoundCurve is a collection of zero or more continuous CircularString or LineString instances.
- An empty CompoundCurve instance can be instantiated, but for a CompoundCurve to be valid it must meet the following criteria:
  1. It must contain at least one CircularString or LineString instance.
  2. The sequence of CircularString or LineString instances must be continuous.
- The following illustration shows valid CompoundCurve types:



CompoundCurve

- Example: create a geometry CompoundCurve composed of a LineString and a CircularString

```
DECLARE @g geometry;
SET @g = 'COMPOUNDCURVE (
           (2 2, 0 0),
           CIRCULARSTRING (0 0, 1 2.1082, 3 6.3246, 0 7,
                           -3 6.3246, -1 2.1082,0 0)
           )';
```

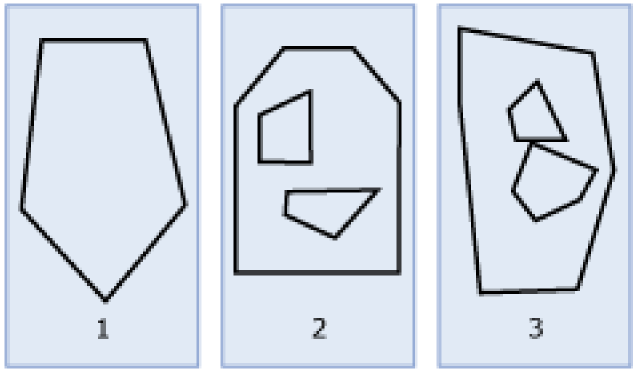
CompoundCurve

- Example: The following example uses two different ways to use a CompoundCurve instance to store a square.

```
DECLARE @g1 geometry, @g2 geometry;
SET @g1 = 'COMPOUNDCURVE((1 1, 1 3), (1 3, 3 3),(3 3, 3 1),(3 1, 1 1))';
SET @g2 = 'COMPOUNDCURVE((1 1, 1 3, 3 3, 3 1, 1 1))';
SELECT @g1.STLength(), @g2.STLength();
```

Polygon

- A Polygon is a two-dimensional surface stored as a sequence of points defining an exterior bounding ring and zero or more interior rings.
  - The exterior and any interior rings of a Polygon define its boundary.
  - The space within the rings defines the interior of the Polygon.
- The illustration below shows examples of Polygon instances.



Polygon

- Example: creates a simple geometry Polygon instance without a hole

```
DECLARE @g geometry
SET @g = 'POLYGON((0 0, 0 3, 3 3, 3 0, 0 0))’
```

- Example: creates a simple geometry Polygon instance with a hole

```
DECLARE @g geometry
SET @g = 'POLYGON((0 0, 0 3, 3 3, 3 0, 0 0),
                  (1 1, 1 2, 2 1, 1 1))'
SELECT @g.STArea() AS PolygonArea
```

CurvePolygon

- Curve polygons are similar to polygons, having at least one ring and zero or more holes (inner rings). Curve polygons are composed of linear strings, circular strings, and/or compound curves.

- Example: creates a CurvePolygon with only an exterior ring

```
DECLARE @g geometry;
SET @g = 'CURVEPOLYGON(CIRCULARSTRING(2 4, 4 2, 6 4, 4 6, 2 4))'
```

CurvePolygon

- Example: creates a CurvePolygon containing interior rings

```
DECLARE @g geometry;
SET @g = 'CURVEPOLYGON(
            CIRCULARSTRING(0 4, 4 0, 8 4, 4 8, 0 4),
           CIRCULARSTRING(2 4, 4 2, 6 4, 4 6, 2 4)
         )';
SELECT @g.STArea() AS Area;
```

CurvePolygon

- Example: creates a CurvePolygon made up of compound curves, themselves made up of circular strings and linear strings.

```
DECLARE @g GEOGRAPHY;
SET @g = '
   CURVEPOLYGON(
     COMPOUNDCURVE(
       (0 -23.43778, 0 23.43778),
       CIRCULARSTRING(0 23.43778, -45 23.43778, -90 23.43778),
       (-90 23.43778, -90 -23.43778),
       CIRCULARSTRING(-90 -23.43778, -45 -23.43778, 0 -23.43778)
     )
   )';
```

MultiPoint

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

- Example: creates a geometry MultiPoint instance with SRID 23 and two points: (2, 3) and (7, 8).

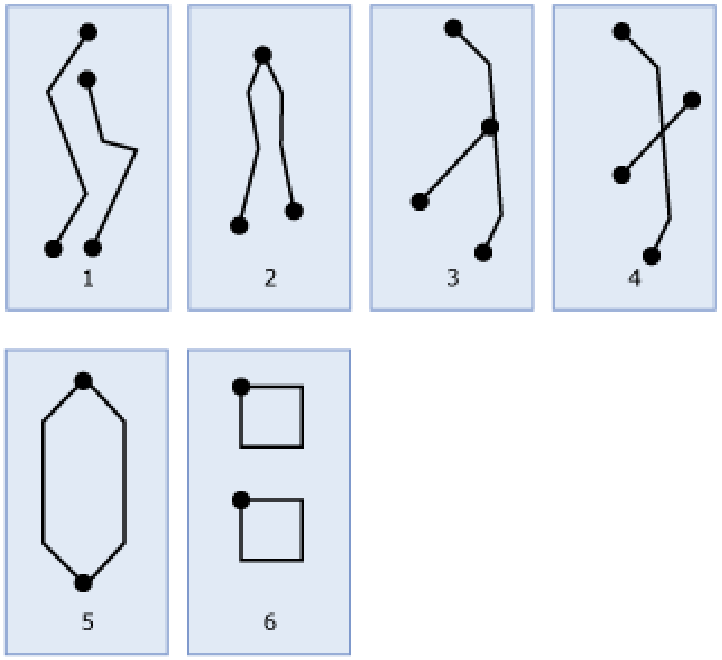
```
DECLARE @g geometry;
SET @g = geometry::STGeomFromText(
            'MULTIPOINT((2 3), (7 8))', 23);
```

- Example: The following example uses the method STGeometryN() to retrieve a Well-Known Text (WKT) of the first point in the collection.

```
SELECT @g.STGeometryN(1).STAsText();
```

MultiLineString

- A MultiLineString is a collection of zero or more geometry or geographyLineString instances.
- The illustration below shows examples of MultiLineString instances.



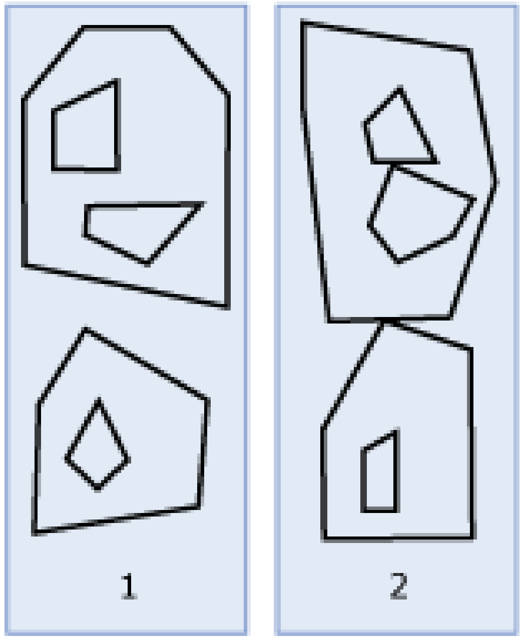
MultiLineString

- Example: creates a simple geometry MultiLineString instance containing two LineString elements with the SRID 12.

```
DECLARE @g geometry;
SET @g = 'MULTILINESTRING((0 2, 1 1), (1 0, 1 1))';
SET @g.STSrid = 12;
```

MultiPolygon

- A MultiPolygon instance is a collection of zero or more Polygon instances.
- The illustration below shows examples of MultiPolygon instances.



MultiPolygon

- Example: creation of a geometry MultiPolygon instance and returns the WKT of the second component.

```
DECLARE @g geometry;
SET @g = 'MULTIPOLYGON(
            ((0 0, 0 3, 3 3, 3 0, 0 0), (1 1, 1 2, 2 1, 1 1)),
            ((9 9, 9 10, 10 9, 9 9))
          )';
SELECT @g.STGeometryN(2).STAsText();
```

GeometryCollection

- A GeometryCollection is a collection of zero or more geometry or geography instances.
- Example: instantiates a geometry GeometryCollection containing a Point instance and a Polygon instance.

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

Methods on Geography Instances

The geography data type provides numerous built-in methods.

- Create, Construct, and Query geography Instances

References

- docs.microsoft.com/en-us/sql/relational-databases/spatial
- docs.microsoft.com/en-us/sql/t-sql/spatial-geography/spatial-types-geography
- docs.microsoft.com/en-us/sql/relational-databases/spatial/create-construct-and-query-geography-instances
- en.wikipedia.org