CSE 594: Spatial Data Science & Engineering

Lecture 3

Spatial SQL Part 1

Database

- A large organized collection of data

DBMS

- A software system to store, retrieve, and manipulate data
- Example – PostgreSQL, MySQL

Relational Database

- A collection of structured data organized as a set of tables with rows and columns

Why Storing Data in a Database?

Why Relational Database?

- Easy to use
- Flexibility of making changes
- Concurrent collaboration among multiple users
- Compliance with ACID properties
- Reduces redundancy through normalization

SQL – Structured Query Language

- A declarative programming language
  - ➢ Define only what to do
  - ➢ How to do is a black box

- Data Definition Language
  - ➢ Create, alter, delete tables and their attributes

- Data Manipulation Language
  - ➢ Retrieve, insert, delete, modify rows in the tables

Schemas/Tables in SQL

Table Name

Column Name

Employees

id	name	designation	salary
1	Thomas	Manager	60000
2	George	Sr. Manager	85000
3	Harry	Junior Manager	45000
4	Jack	Manager	65000

Tuple/Row

Data Types in SQL

- String data types
  - CHAR, VARCHAR, TEXT

- Numeric data types
  - INT, FLOAT, DOUBLE, BIGINT

- Other data types
  - DATE, DATETIME, TIMESTAMP, YEAR

SQL Query Form

1. **SELECT** [**DISTINCT**] Attribute_List **FROM** R1,R2....RM
2. [**WHERE** condition]
3. [**GROUP BY** (Attributes)[**HAVING** condition]]
4. [**ORDER BY**(Attributes)[**DESC**]];

Sample SQL Queries

Employees

id	name	designation	salary
1	Thomas	Manager	60000
2	George	Sr. Manager	85000
3	Harry	Junior Manager	45000
4	Jack	Manager	65000

SELECT name, salary

FROM employees

name	salary
Thomas	60000
George	85000
Harry	45000
Jack	65000

Sample SQL Queries

Employees

id	name	designation	salary
1	Thomas	Manager	60000
2	George	Sr. Manager	85000
3	Harry	Junior Manager	45000
4	Jack	Manager	65000

SELECT name, salary

FROM employees

WHERE salary > 50000

name	salary
Thomas	60000
George	85000
Jack	65000

Sample SQL Queries

Employees

id	name	designation	salary
1	Thomas	Manager	60000
2	George	Sr. Manager	85000
3	Harry	Junior Manager	45000
4	Jack	Manager	65000

SELECT designation, **AVG**(salary)
FROM employees
WHERE salary > 50000
GROUP BY designation
ORDER BY designation DESC

designation	AVG(salary)
Sr. Manager	85000
Manager	62500

Spatial SQL

- Uses the same elements and structure of normal SQL
- Allows to work with geospatial types such as geometries and geographies

Why Spatial SQL?

- Accessible to wider community
- Versatility with many supporting databases and data warehouses
- Multi-dimensional spatial indexing and built-in functions for managing geometry operations
- Efficiency in everyday workflows and task management
- Cross functionality in the organization
- Work with large scale data in SQL enabled data warehouses

Why Spatial SQL?

Efficiency in everyday workflows and task management

- No need to load the same data every time you start a project
- Create new features, join and aggregate data on the fly
- Create indices on geometries to run geometry operations faster
- Update the same table as new data is available
- Create your own user defined functions

Why Spatial SQL?

Work with large scale data in SQL enabled data warehouses

- Can speed up processing significantly
- An experiment of creating points in PostGIS take less than 10% of the time taken by QGIS

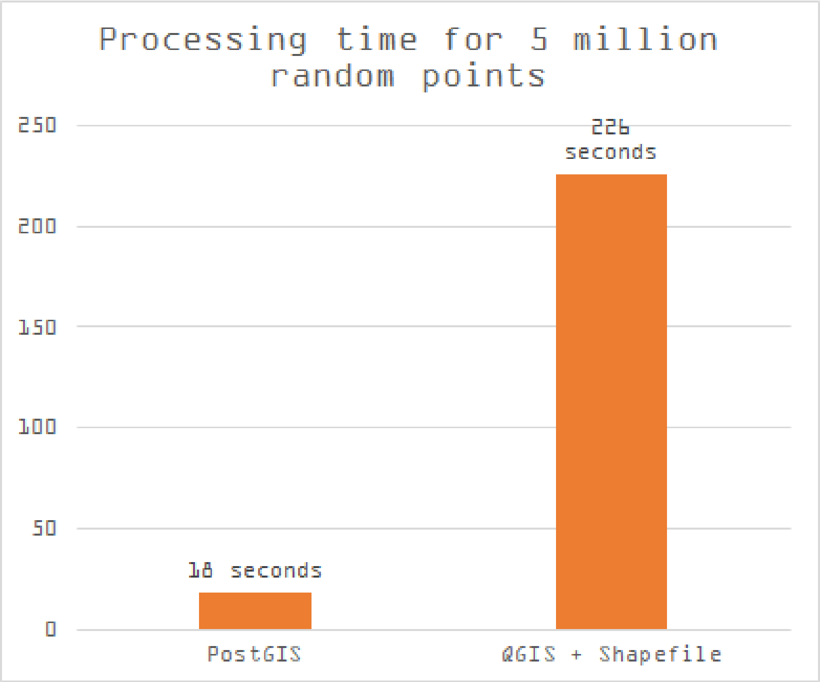


Image Source: https://medium.com/@tjukanov/why-should-you-care-about-postgis-a-gentle-introduction-to-spatial-databases-9eccd26bc42b

Supports for Spatial SQL

Databases

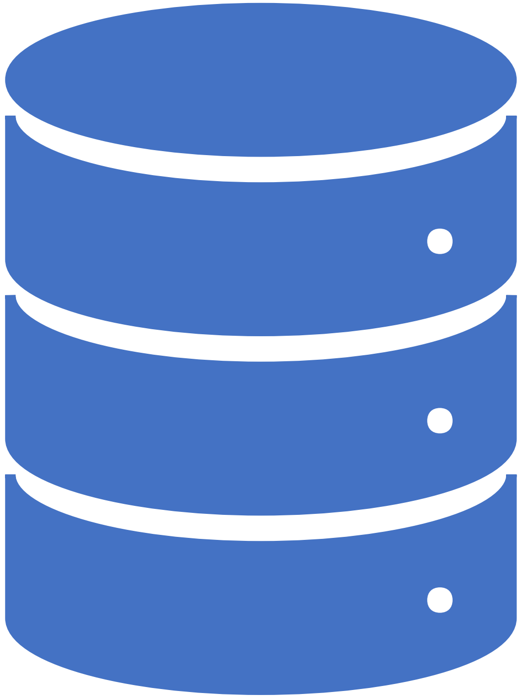
- PostgreSQL with PostGIS
- Microsoft SQL Server
- MySQL
- SQLite with Spatialite
- Oracle Spatial

Data Warehouses

- Google BigQuery
- Snowflake
- AWS Redshift

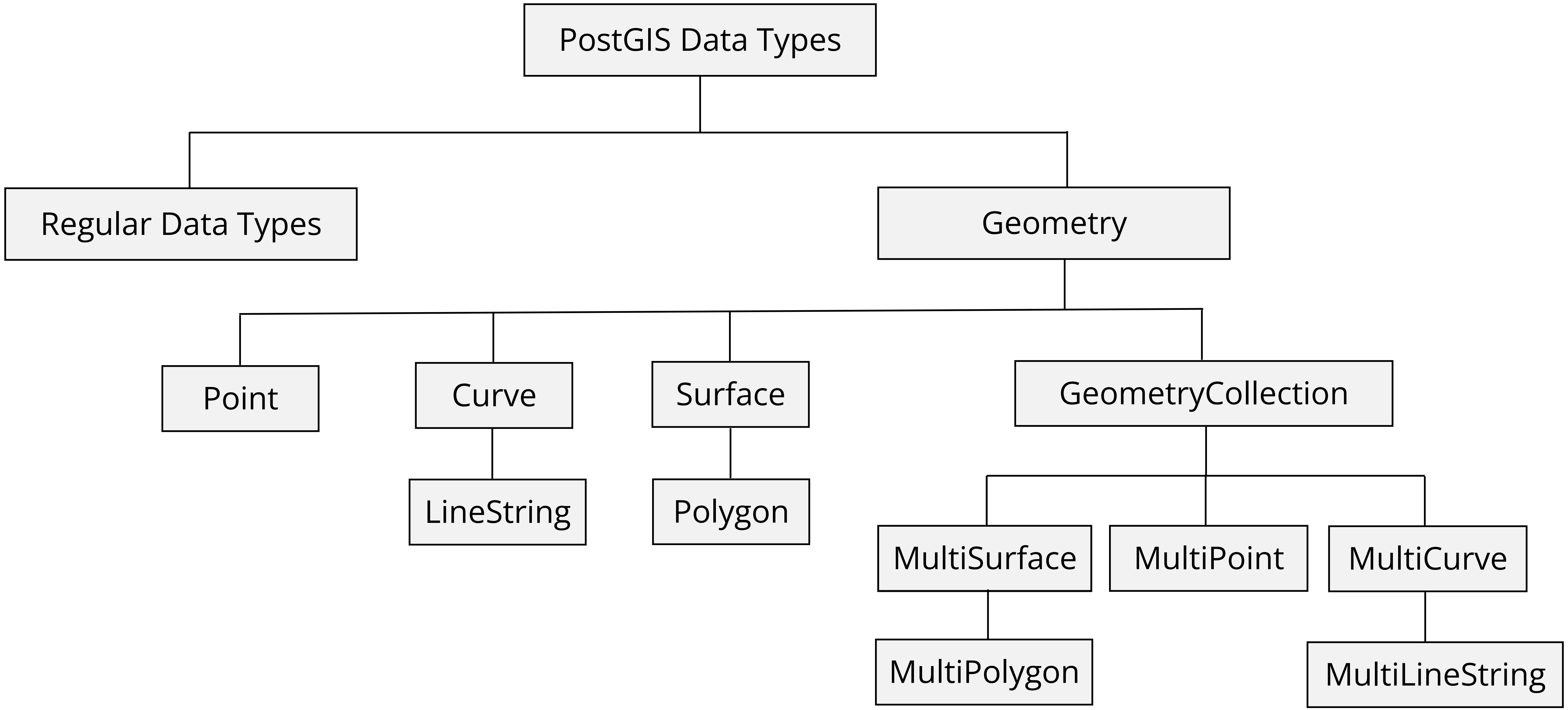
Spatial Extensions of SparkSQL

- Apache Sedona, formerly known as GeoSpark
- GeoMesa



PostGIS Data Model

PostGIS Data Types



PostGIS Representation of Geometry Data Type

Point

- A 0-dimensional geometry that represents a single location in coordinate space

```
POINT (1 2)
```

LineString

- A 1-dimensional line formed by contiguous sequences of line segments
- Each line segment is defined by two points, with the end point of one segment forming the starting point of second segment

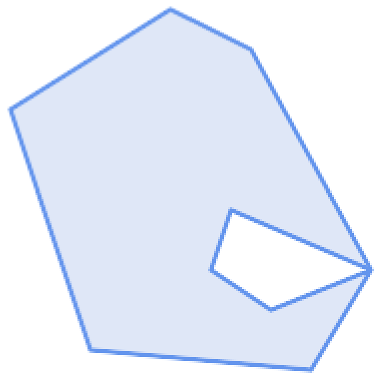
```
LINESTRING (1 2, 3 4, 5 6)
```

PostGIS Representation of Geometry Data Type

Polygon

- A 2-dimensional planar region, delimited by an exterior boundary and zero or more interior boundaries (holes)

```
POLYGON ((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0))
```



MultiPoint

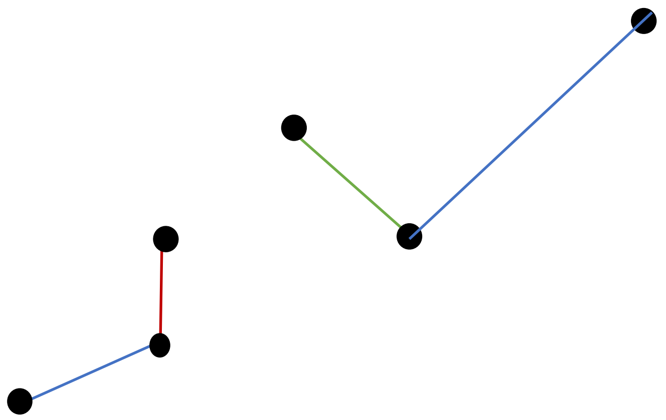
- A collection of Points

```
MULTIPOINT ( (0 0), (1 2) )
```

PostGIS Representation of Geometry Data Type

MultiLineString

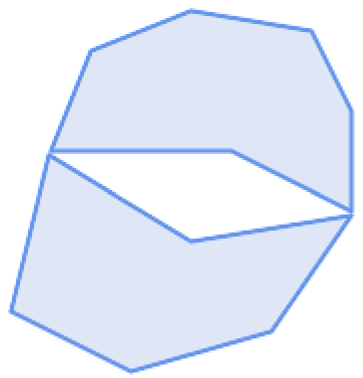
- A collection of LineStrings



```
MULTILINESTRING ( (0 0,1 1,1 2), (2 3,3 2,5 4) )
```

MultiPolygon

- A collection of non-overlapping, non-adjacent polygons



```
MULTIPOLYGON (((1 5, 5 5, 5 1, 1 1, 1 5)), ((6 5, 9 1, 6 1, 6 5)))
```

PostGIS Representation of Geometry Data Type

GeometryCollection

- A heterogeneous or mixed collection of geometries

```
GEOMETRYCOLLECTION ( POINT(2 3), LINESTRING(2 3, 3 4))
```

Triangle

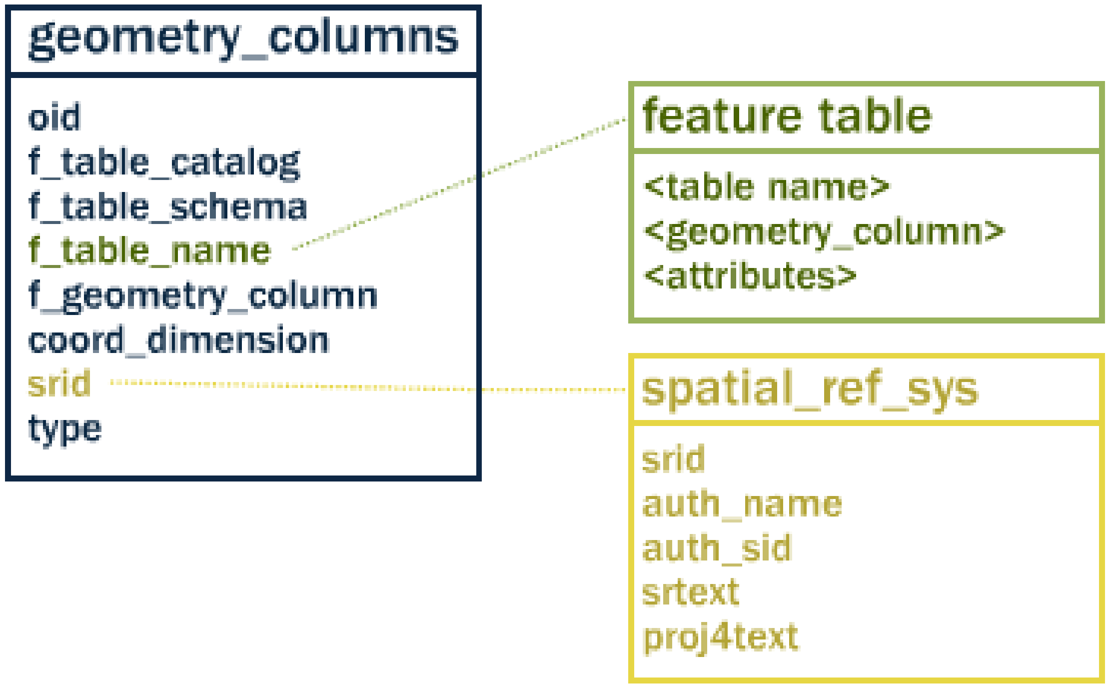
- A Polygon defined by three distinct, non-collinear vertices
- Specified by four coordinates with first and fourth being equal

```
TRIANGLE ((0 0, 0 9, 9 0, 0 0))
```

PostGIS Metadata Tables

- Table spatial_ref_sys defines all spatial reference systems known to the database
- Table geometry_columns provides a listing of all features and details of those features
- f_table_catelog, f_table_schema, f_table_name provide the full name of a feature table containing a geometry
- F_geometry_column is the name of the geometry column
- coord_dimension and srid define the dimension of the geometry and spatial reference system identifier
- The type column defines the type of geometry

Table Relationships



PostGIS SQL

Creating Table with Geometry Type Column

```
CREATE TABLE TABLE_NAME (name varchar, GEOMETRY_COLUMN_NAME geometry);
```

Default srid is 0

```
CREATE TABLE TABLE_NAME (name varchar, GEOMETRY_COLUMN_NAME geometry(POINT));
```

```
CREATE TABLE TABLE_NAME (name varchar, GEOMETRY_COLUMN_NAME geometry(POINT, 4267));
```

```
CREATE TABLE TABLE_NAME (name varchar, GEOMETRY_COLUMN_NAME geometry(LINESTRING, 4267));
```

```
CREATE TABLE TABLE_NAME (name varchar, GEOMETRY_COLUMN_NAME geometry(POLYGON, 4267));
```

```
CREATE TABLE TABLE_NAME (name varchar, GEOMETRY_COLUMN_NAME geometry(POINTZ, 3005));
```

PostGIS SQL

Input Output Conversions for WKB and WKT Spatial Objects

```
byte WKB = ST_AsBinary(geometry);

text WKT = ST_AsText(geometry);

geometry = ST_GeomFromWKB(byte WKB, SRID);

geometry = ST_GeometryFromText(text WKT, SRID);
```

Insert Geometry Data into Table

```
INSERT INTO TABLE_NAME (name, GEOMETRY_COLUMN_NAME )
VALUES ('Location-1’, ST_GeomFromText('POINT(-126.4 45.32)’, 4267));
```

PostGIS SQL

Creating Table with Geography Type Column

Default srid is 4326

```
CREATE TABLE TABLE_NAME (name varchar, GEOGRAPHY_COLUMN_NAME geography(POINT));
```

```
CREATE TABLE TABLE_NAME (name varchar, GEOGRAPHY_COLUMN_NAME geography(POINT, 4267));
```

```
CREATE TABLE TABLE_NAME (name varchar, GEOGRAPHY_COLUMN_NAME geography(LINESTRING, 4267));
```

```
CREATE TABLE TABLE_NAME (name varchar, GEOGRAPHY_COLUMN_NAME geography(POLYGON, 4267));
```

```
CREATE TABLE TABLE_NAME (name varchar, GEOGRAPHY_COLUMN_NAME geography(POINTZ, 3005));
```

PostGIS SQL

Insert Geography Data into Table

```
INSERT INTO TABLE_NAME (name, GEOGRAPHY_COLUMN_NAME )
VALUES ('Location-1’, ‘SRID=4326;POINT(-126.4 45.32)’);
```

PostGIS SQL Functions

Related to Metadata

- ST_GeometryType(geometry) :- Returns the type of a geometry
- ST_NDims(geometry) :- Returns the number of dimensions in a geometry
- ST_SRID(geometry) :- Returns the spatial reference identifier number of a geometry

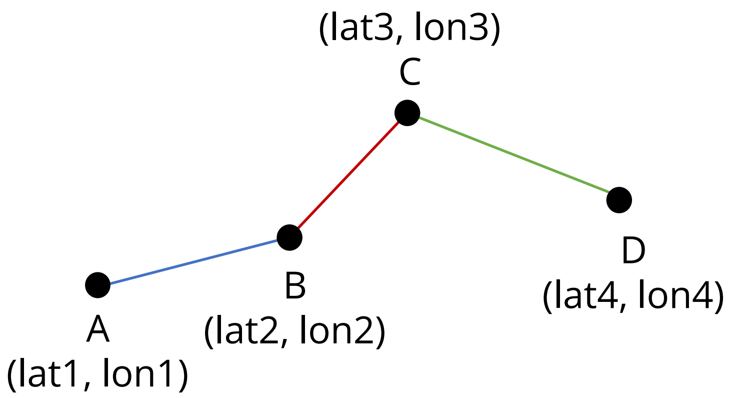
Related to Point Coordinates

- ST_X(geometry) :- Returns the X-coordinate of a point geometry
- ST_Y(geometry) :- Returns the Y-coordinate of a point geometry

PostGIS SQL Functions

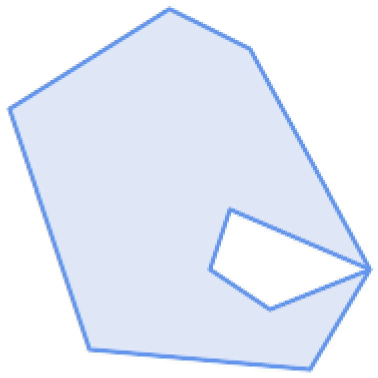
Related to LineString

- ST_Length(geometry) :- Returns the length of the LineString
- ST_StartPoint(geometry) :- Returns the first coordinate as a Point
- ST_EndPoint(geometry) :- Returns the last coordinate as a Point
- ST_NPoints(geometry) :- Returns the number of coordinates in the LineString



Related to Polygon

- ST_Area(geometry) :- Returns the area of the polygon
- ST_NRings(geometry) :- Returns the number of rings (1 if there are no holes)
- ST_ExteriorRing(geometry) :- Returns the outer ring as a linestring



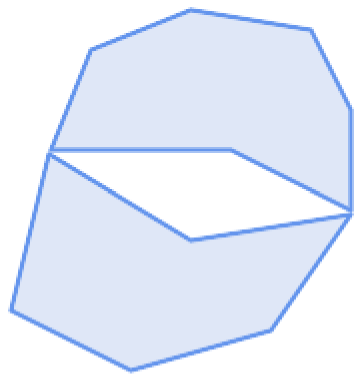
PostGIS SQL Functions

Related to Polygon (Continued…)

- ST_InteriorRingN(geometry, n) :- Returns a specified interior ring as a linestring
- ST_Perimeter(geometry) :- Returns the length of all rings

Related to Collections

- ST_NumGeometries(geometry) :- Returns the number of parts in the collection
- ST_GeometryN(geometry, n) :- Returns the specified part
- ST_Area(geometry) :- Returns the total area of all Polygonal parts
- ST_Length(geometry) :- Returns the total length of all linear parts



Creating PostGIS Geometry Objects

Creating Points

geometry **ST_MakePoint**(float *x*, float *y*);

geometry **ST_MakePoint**(float *x*, float *y*, float *z*);

Creating Lines

geometry **ST_MakeLine**(geometry *geom1*, geometry *geom2*);
geometry **ST_MakeLine**(geometry[] *geoms_array*);
geometry **ST_MakeLine**(geometry set *geoms*);

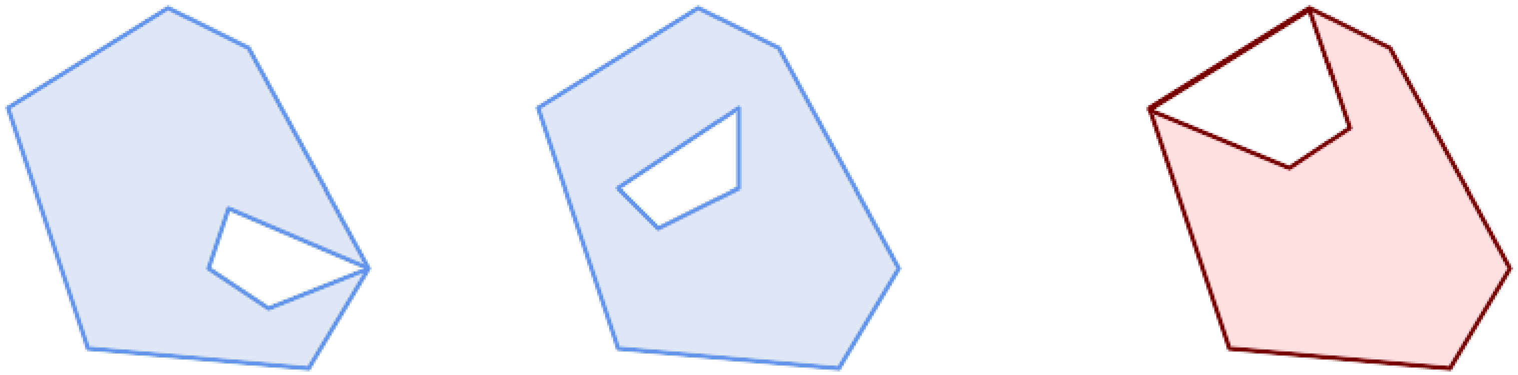
Creating Polygons

geometry **ST_MakePolygon**(geometry *linestring*);
geometry **ST_MakePolygon**(geometry *outerlinestring*, geometry[] *interiorlinestrings*);

Validity of Geometries

Properties of a Valid Polygon

- The boundary rings do not cross and self touch
- The boundary rings may touch at points only as a tangent, not in a line
- Interior rings are within the exterior ring
- The polygon interiors should not touch in a way that splits the polygon into parts

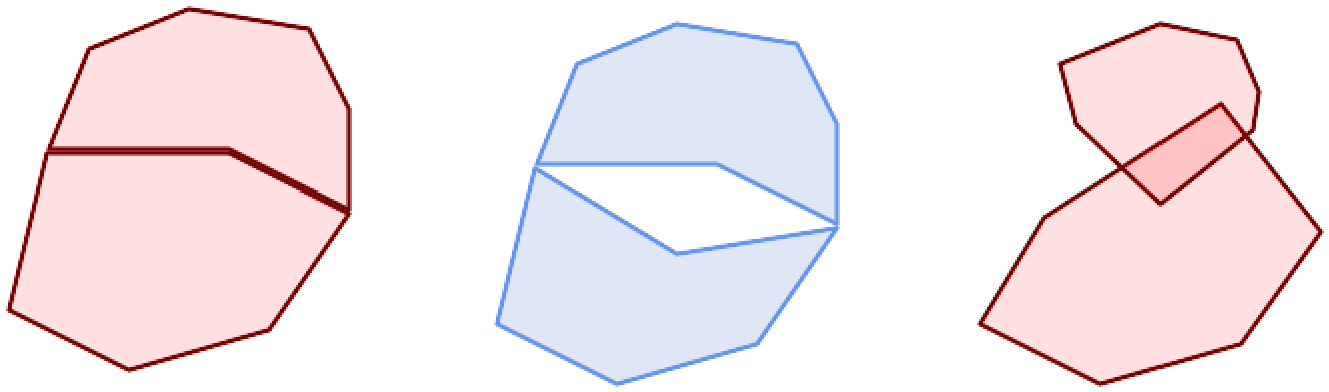


Source: https://postgis.net/docs/using_postgis_dbmanagement.html#LinearRing

Validity of Geometries

Properties of a Valid MultiPolygon

- Element polygons are valid
- Elements do not overlap or their interiors must not intersect
- Elements touch only at points, not along a line



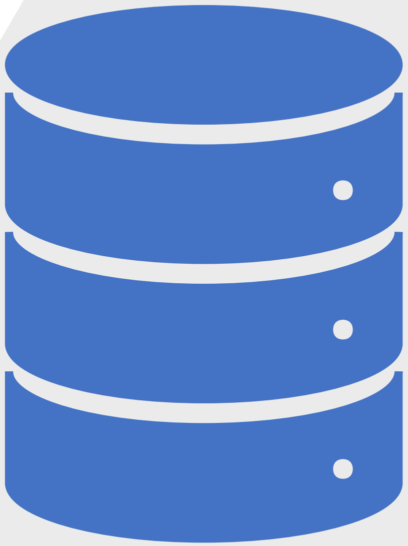
Source: https://postgis.net/docs/using_postgis_dbmanagement.html#LinearRing

All ST Functions Supported by PostGIS

Visit the link:

https://postgis.net/docs/manual-1.5/ch08.html

Apache Sedona Spatial SQL Data Model



Supported Geometry Objects

- Point
- MultiPoint
- LineString
- MultiLineString
- Polygon
- MultiPolygon

Sedona SQL Data Structure

- Datasets are represented as Spark DataFrames
- Each DataFrame can be considered as a Table in PostGIS
- Each attribute is a column, while each data instance is a row
- Spatial DataFrame contains a geometry type column

Spatial Operation Through SQL with Spatial DataFrames

A Sample Spatial DataFrame: dfSpatialSample

Name	Area	Geometry
Name-1	470	POLYGON ((933100.92 192536.09, 933091.01 192572.17, 933088.58 192604.97, 933779.28 195908.73, 933841.76 195957.79, 933100.92 192536.09))
Name-2	520	MULTIPOLYGON (((1033269.24 172126.00, 1033439.64 170883.95, 1033473.26 170808.21, 1033269.24 172126.00)), ((1033422.35 157944.65, 1033419.99 157936.99, 1033408.21 157938.17, 1033422.35 157944.65)))
Name-3	300	POLYGON ((933100.92 192536.09, 933091.01 192572.17, 933088.58 192604.97, 933779.28 195908.73, 933841.76 195957.79, 933100.92 192536.09))
Name-4	740	POLYGON ((933100.92 192536.09, 933091.01 192572.17, 933088.58 192604.97, 933779.28 195908.73, 933841.76 195957.79, 933100.92 192536.09))

Spatial Operation Through SQL with Spatial DataFrames

Running SQL Queries on dfSpatialSample

1. Create a temporary view from the DataFrame object

```
dfSpatialSample.createOrReplaceTempView(“sample_spatial_view”)
```

2. Run SQL queries assuming the view name as a table name

```
dfSpatialSample  = sparkSession.sql(“SELECT … FROM sample_spatial_view WHERE …”)
```

3. Display the schema

```
dfSpatialSample.printSchema()
```

Constructors for Creating Geometry Objects

- ST_geomFromText
- ST_GeomFromWkB
- ST_Point
- ST_GeomFromGeoJSON
- ST_PolygonFromEnvelop

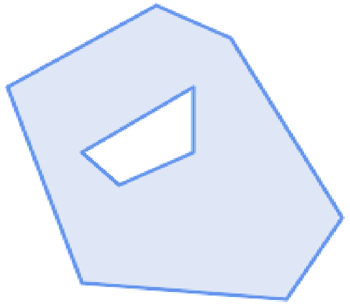
Sample Usage

```
SELECT ST_GeomFromWKT('POINT(40.7128 -74.0060)') AS geometry
```

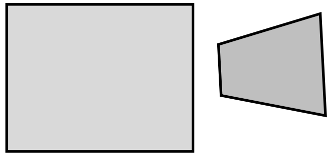
More details: https://sedona.apache.org/api/sql/Constructor/

Predicates Supported by Sedona SQL

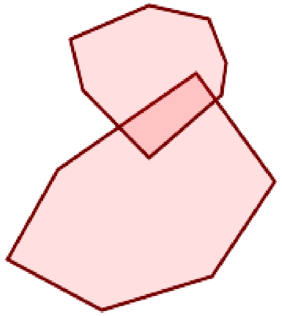
- ST_Within(A, B) – returns True if A is fully contained by B
- ST_Disjoint(A, B) – returns True if A and B are disjoint
- ST_Intersects(A, B) – returns True if A intersects B



Within



Disjoint



Intersects

Sample Usage

```
SELECT * FROM pointdf
WHERE ST_Intersects(ST_PolygonFromEnvelope(1.0,100.0,1000.0,1100.0),
                                        pointdf.arealandmark)
```

More details: https://sedona.apache.org/api/sql/Predicate/

Sedona SQL Functions

- ST_Distance(A, B) – returns the Euclidean distance between A and B
- ST_StartPoint(A) – returns the first point of a given linestring
- ST_GeometryType(A) – returns the type of the geometry as a string

Sample Usage

```
SELECT ST_GeometryType(polygondf.countyshape) FROM polygondf
```

More details: https://sedona.apache.org/api/sql/Function/