# Experiment No.5

## Title: Execution of Spatial database queries

**Name: Sahil Biswas  Batch: A1  Roll No.: 16010422013  Experiment No.:5**

**Aim: To execute spatial queries using PostGIS.**

---

**Resources needed:** PostgreSQL 9.6, PostGIS 2.0

---

**Theory**

**PostGIS** is an open source software program that adds support for geographic objects to the PostgreSQL object-relational database. PostGIS follows the Simple Features for SQL specification from the Open Geospatial Consortium (OGC). PostGIS turns the PostgreSQL Database Management System into a spatial database by adding support for the three features: spatial types, indexes, and functions. Because it is built on PostgreSQL, PostGIS automatically inherits important “enterprise” features as well as open standards for implementation. PostgreSQL is a powerful, object-relational database management system (ORDBMS). It is also open source software.

**Features of** PostGIS

- Geometry types for points, line strings, polygons, multi-points, multi-line-strings, multi-polygons and geometry collections.
- Spatial predicates for determining the interactions of geometries using the 3x3 Egenhofer matrix (provided by the GEOS software library).
- Spatial operators for determining geospatial measurements like area, distance, length and perimeter.
- Spatial operators for determining geospatial set operations, like union, difference, symmetric difference and buffers (provided by GEOS).
- R-tree-over-GiST (Generalised Search Tree) spatial indexes for high speed spatial querying.
- Index selectivity support, to provide high performance query plans for mixed spatial/non-spatial queries.
- For raster data

Geometry is and abstract type and concrete subtypes can be **atomic** or **collection** types

- Atomic
    - Point : It represents a single location in coordinate space
      e.g. POINT(3, 4), POINT (3,5,4,8)

    - LineString : It is a 1-dimensional line formed by a contiguous sequence of line segments. Each line segment is defined by two points, with the end point of one segment forming the start point of the next segment
      e.g. LINESTRING (1 2, 3 4, 5 6)

    - LineRing : It is a LineString which is both closed and simple. The first and last points must be equal, and the line must not self-intersect
      e.g. LINEARRING (0 0 0, 4 0 0, 4 4 0, 0 4 0, 0 0 0)

    - Polygon : It is a 2-dimensional planar region, delimited by an exterior boundary (the shell) and zero or more interior boundaries (holes). Each boundary is a LinearRing.
      e.g. 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))

- **Collection**

- MultiPoint : It is a collection of points
  e.g. MULTIPOINT ( (0 0), (1 2) )

- MultiLineString : It is a collection of LineStrings. A MultiLineString is closed if each of its elements is closed
  e.g. MULTILINESTRING ( (0 0,1 1,1 2), (2 3,3 2,5 4) )

- MultiPolygon : It is a collection of non-overlapping, non-adjacent polygons. Polygons in the collection may touch only at a finite number of points.
  e.g. MULTIPOLYGON (((1 5, 5 5, 5 1, 1 1, 1 5)), ((6 5, 9 1, 6 1, 6 5)))

- GeometryCollection : It is a is a heterogeneous (mixed) collection of geometries
  e.g. GEOMETRYCOLLECTION ( POINT(2 3), LINESTRING(2 3, 3 4))

- Also there are PolyHedralSurface, Triangle and TIN

PostGIS provides different functions for determining relationships(topological ordistance) between geometries, compute measurements, overlays and geometry construction also besides other provisions.
Few of the functions are

**Measurement functions**
ST_Area : **`float ST_Area(geometry *g1*);`**
Returns the area of a polygonal geometry

ST_Length : **`float ST_Length(geometry *a_2dlinestring*);`** R
Returns the 2D Cartesian length of the geometry if it is a LineString, MultiLineString, ST_Curve, ST_MultiCurve

ST_Perimeter : **`float ST_Perimeter(geometry *g1*);`**
Returns the 2D perimeter of the geometry/geography if it is a ST_Surface, ST_MultiSurface (Polygon, MultiPolygon)

**Named Spatial Relationships**
For determining common spatial relationships, OGC SFS defines a set of named spatial relationship predicates. PostGIS provides these as the functions
ST_Contains : **`boolean ST_Contains(geometry geomA, geometry geomB);`**
ST_Crosses : **`boolean ST_Crosses(geometry g1, geometry g2);`**
ST_Disjoint : **`boolean ST_Disjoint( geometry A , geometry B );`**
ST_Equals : **`boolean ST_Equals(geometry A, geometry B);`**
ST_Intersects : **`boolean ST_Intersects( geometry geomA , geometry geomB );`**
ST_Overlaps : **`boolean ST_Overlaps(geometry A, geometry B);`**
ST_Touches : **`boolean ST_Touches(geometry A, geometry B);`**
ST_Within. : **`boolean ST_Within(geometry A, geometry B);`**

It also defines the non-standard relationship predicates
ST_Covers : **`boolean ST_Covers(geometry geomA, geometry geomB);`**
ST_CoveredBy : **`boolean ST_CoveredBy(geometry geomA, geometry geomB);`**
ST_ContainsProperly : **`boolean ST_ContainsProperly(geometry geomA, geometry geomB);`**

Spatial predicates are usually used as conditions in SQL WHERE or JOIN clauses.

```
SELECT city.name, state.name, city.geom
```

```
FROM city JOIN state ON ST_Intersects(city.geom, state.geom);
```

---

**Procedure:**

1. Installation of relational database PostgreSQL 9.6 (download from http://www.enterprisedb.com/products-services-training/pgdownload )
2. Installation of PostGIS using Application stack builder.
3. Download spatial data from **`https://www.diva-gis.org/gdata`** (OR similar website with FREE usable data) Get it for any country with minimum 3 subjects.
4. Import the data in your PostgreSQL
5. Identify spatial relationship between any two geometric entities (any 3 named relationships)
6. Perform any two measurement functions for geometric data.
7. Execute any one range query

```
SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)'::geometry) as d,edabbr,
vaabbr

FROM va2005

ORDER BY d limit 10;


        d         | edabbr | vaabbr
------------------+--------+--------

                0 | ALQ    | 128

 5541.57712511724 | ALQ    | 129A

 5579.67450712005 | ALQ    | 001

  6083.4207708641 | ALQ    | 131

  7691.2205404848 | ALQ    | 003

 7900.75451037313 | ALQ    | 122

 8694.20710669982 | ALQ    | 129B

 9564.24289057111 | ALQ    | 130

  12089.665931705 | ALQ    | 127

 18472.5531479404 | ALQ    | 002

(10 rows)
```

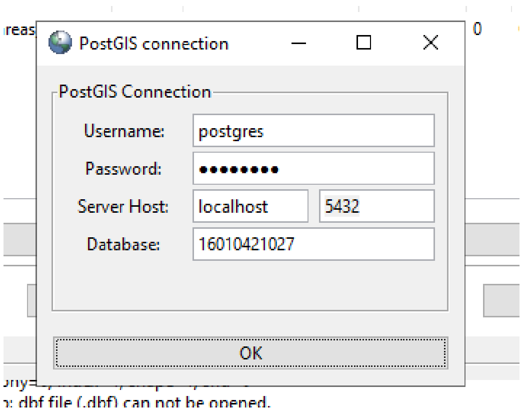
**Range query in Postgis**
**SELECT ST_Reclass(rast, 1,**
  '[0-90]:0,(90-100):1,[100-1000):2',
    '4BUI', 0) AS rast FROM sometable
  WHERE filename = '123.tif';

---

**Results: (Program printout with output)**

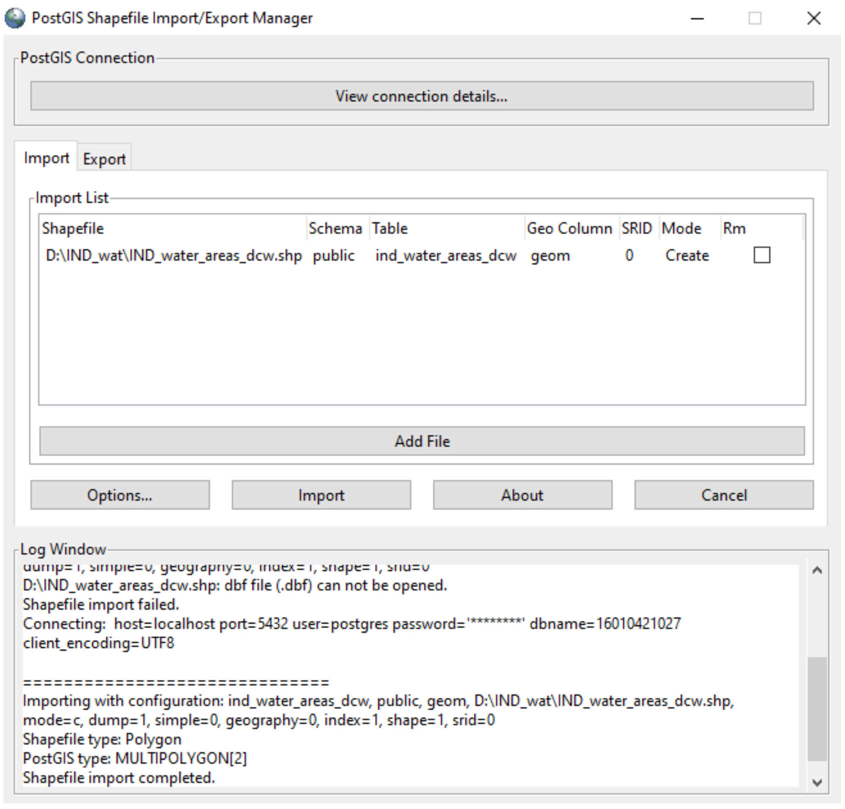
```
create extension postgis;

 CREATE EXTENSION

 Query returned successfully in 3 secs.
```

Establishing the connection



PostGIS Connection
Username: postgres
Password: ●●●●●●●●
Server Host: localhost 5432
Database: 16010421027

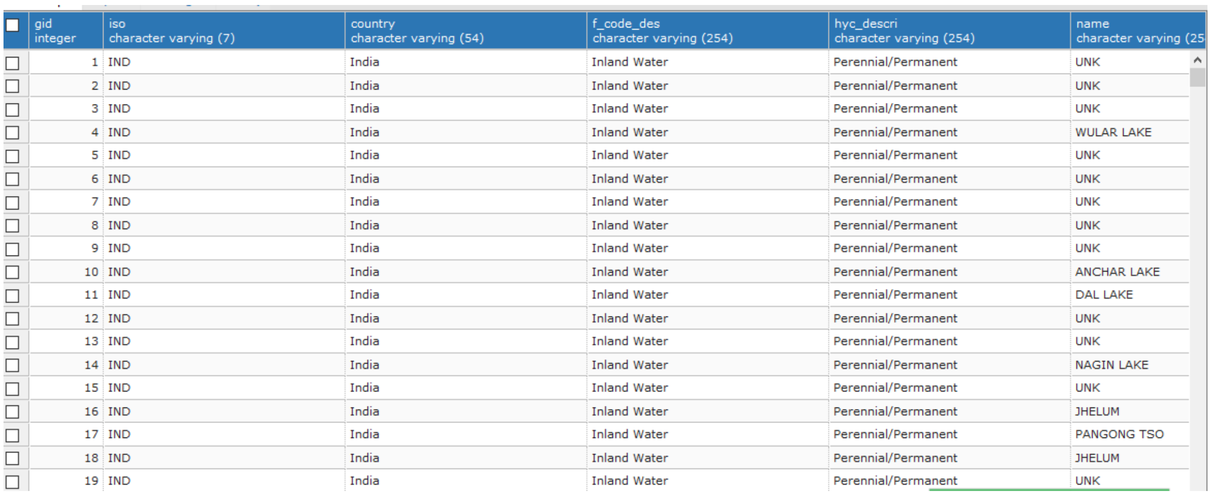
Adding the shp file



| Shapefile | Schema | Table | Geo Column | SRID | Mode | Rm |
|---|---|---|---|---|---|---|
| D:\IND_wat\IND_water_areas_dcw.shp | public | ind_water_areas_dcw | geom | 0 | Create | ☐ |

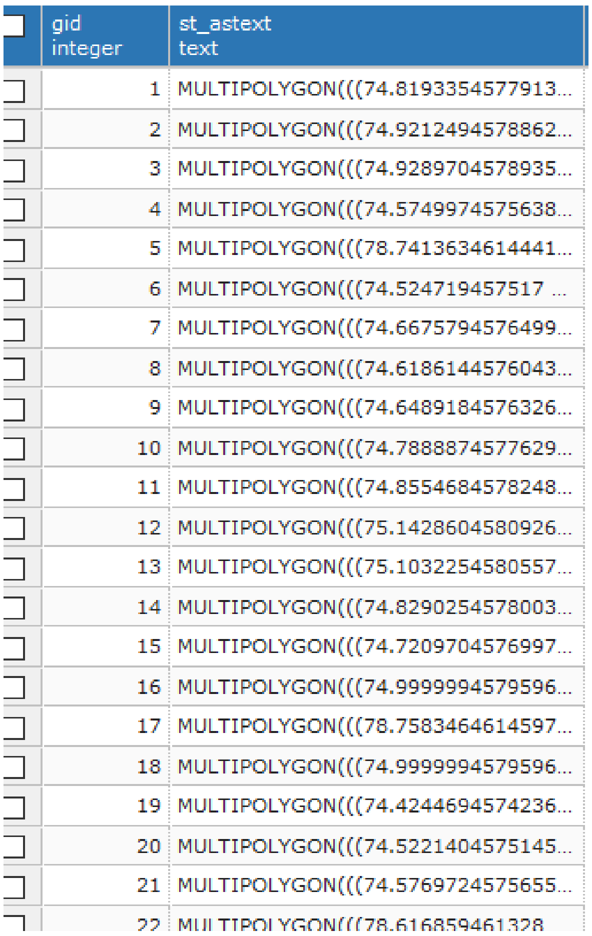
```
dump=1, simple=0, geography=0, index=1, shape=1, srid=0
D:\IND_water_areas_dcw.shp: dbf file (.dbf) can not be opened.
Shapefile import failed.
Connecting:  host=localhost port=5432 user=postgres password='********' dbname=16010421027
client_encoding=UTF8

==============================
Importing with configuration: ind_water_areas_dcw, public, geom, D:\IND_wat\IND_water_areas_dcw.shp,
mode=c, dump=1, simple=0, geography=0, index=1, shape=1, srid=0
Shapefile type: Polygon
PostGIS type: MULTIPOLYGON[2]
Shapefile import completed.
```

```
select * from ind_water_areas_dcw;
```



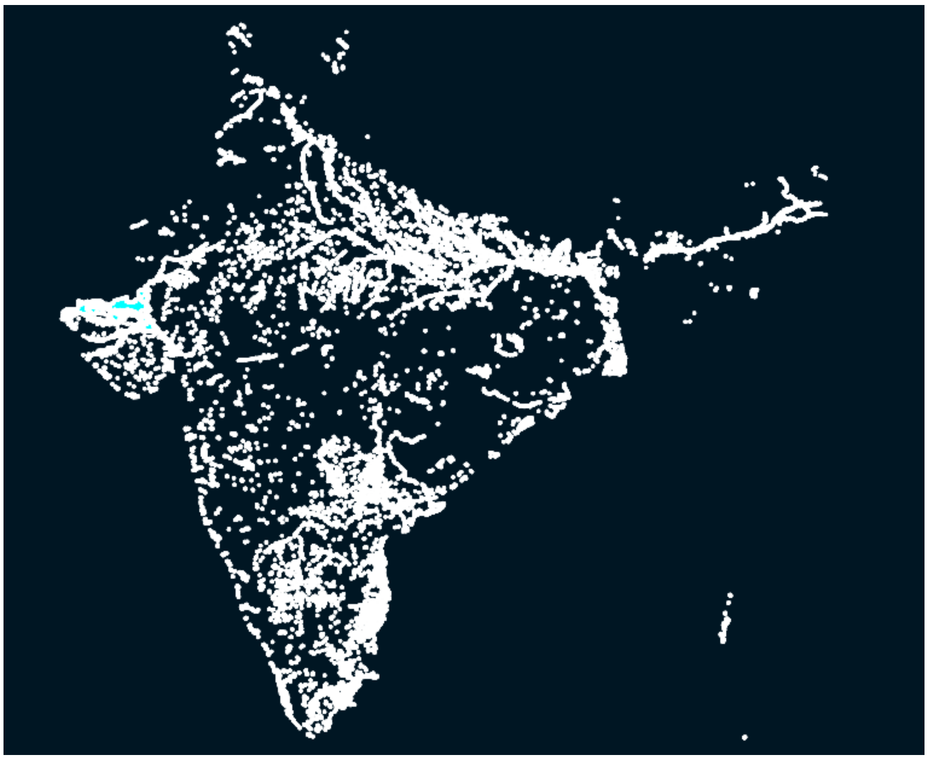
| gid integer | iso character varying (7) | country character varying (54) | f_code_des character varying (254) | hyc_descri character varying (254) | name character varying (25 |
|---|---|---|---|---|---|
| 1 | IND | India | Inland Water | Perennial/Permanent | UNK |
| 2 | IND | India | Inland Water | Perennial/Permanent | UNK |
| 3 | IND | India | Inland Water | Perennial/Permanent | UNK |
| 4 | IND | India | Inland Water | Perennial/Permanent | WULAR LAKE |
| 5 | IND | India | Inland Water | Perennial/Permanent | UNK |
| 6 | IND | India | Inland Water | Perennial/Permanent | UNK |
| 7 | IND | India | Inland Water | Perennial/Permanent | UNK |
| 8 | IND | India | Inland Water | Perennial/Permanent | UNK |
| 9 | IND | India | Inland Water | Perennial/Permanent | UNK |
| 10 | IND | India | Inland Water | Perennial/Permanent | ANCHAR LAKE |
| 11 | IND | India | Inland Water | Perennial/Permanent | DAL LAKE |
| 12 | IND | India | Inland Water | Perennial/Permanent | UNK |
| 13 | IND | India | Inland Water | Perennial/Permanent | UNK |
| 14 | IND | India | Inland Water | Perennial/Permanent | NAGIN LAKE |
| 15 | IND | India | Inland Water | Perennial/Permanent | UNK |
| 16 | IND | India | Inland Water | Perennial/Permanent | JHELUM |
| 17 | IND | India | Inland Water | Perennial/Permanent | PANGONG TSO |
| 18 | IND | India | Inland Water | Perennial/Permanent | JHELUM |
| 19 | IND | India | Inland Water | Perennial/Permanent | UNK |

```
select gid,ST_AsText(geom) from ind_water_areas_dcw;
```



| gid integer | st_astext text |
|---|---|
| 1 | MULTIPOLYGON(((74.8193354577913... |
| 2 | MULTIPOLYGON(((74.9212494578862... |
| 3 | MULTIPOLYGON(((74.9289704578935... |
| 4 | MULTIPOLYGON(((74.5749974575638... |
| 5 | MULTIPOLYGON(((78.7413634614441... |
| 6 | MULTIPOLYGON(((74.524719457517 ... |
| 7 | MULTIPOLYGON(((74.6675794576499... |
| 8 | MULTIPOLYGON(((74.6186144576043... |
| 9 | MULTIPOLYGON(((74.6489184576326... |
| 10 | MULTIPOLYGON(((74.7888874577629... |
| 11 | MULTIPOLYGON(((74.8554684578248... |
| 12 | MULTIPOLYGON(((75.1428604580926... |
| 13 | MULTIPOLYGON(((75.1032254580557... |
| 14 | MULTIPOLYGON(((74.8290254578003... |
| 15 | MULTIPOLYGON(((74.7209704576997... |
| 16 | MULTIPOLYGON(((74.9999994579596... |
| 17 | MULTIPOLYGON(((78.7583464614597... |
| 18 | MULTIPOLYGON(((74.9999994579596... |
| 19 | MULTIPOLYGON(((74.4244694574236... |
| 20 | MULTIPOLYGON(((74.5221404575145... |
| 21 | MULTIPOLYGON(((74.5769724575655... |
| 22 | MULTIPOLYGON(((78.616859461328 |

Shape file viewer



**Questions:**

**1. Explain the spatial functions used for these queries in detail.**

Spatial databases store and retain any kind of spatial data that is location-related and that represents objects specified in a geometric space. To manage these spatial databases, these are employed. Most simple geometric objects, including 3D objects, topological coverage, linear networks, and TINs, are represented in spatial databases (Triangulated irregular networks).
Three different categories of spatial searches are listed below.

• Closeness questions

It asks for items that are around a given place. A nearness query is one that asks for all hotels that are located within a certain radius of a particular spot. The item that is closest to a given point is what is requested by the nearest-neighbor query. For instance, we might want to locate the closest railroad station.

Noting that there is no requirement that this query mention a distance limit, we can ask it even if we are unsure of the distance to the closest train station.

• Regional searches

It deals with geographical areas. A query can, for instance, ask for things that are entirely or partially present inside a specified region. We may do a search to identify every pharmacy located within the boundaries of a given municipality, or we can find every school open in a specific location.

• Union/Intersection: We may also ask for the intersections and unions of regions in this sort of query. For instance, a query can ask for all regions with a low yearly rainfall and a high population density given region details like annual rainfall and population density. In general, queries on geographic data include both spatial and non-spatial constraints. For instance, we might search for the closest eatery that serves vegetarian options and bills meals at less than $10.
We typically use a graphical query language to query spatial data because they are by their very nature pictorial. Such queries also return results that are presented graphically rather than in tables.
By pointing and clicking on suburbs west of Manhattan, for example, the user can zoom in and out, choose what to display based on selection criteria (such as hotels with more than three stars),

overlay multiple maps (such as hotels with more than three stars overlaid on a map representing areas with low crime rates), and perform other operations on the interface. The front end is made up of the graphical user interface.

SQL extensions have been proposed to enable relational databases to efficiently store and retrieve geographic information as well as to enable the blending of spatial and non-spatial variables in queries. Extensions include allowing geographical conditions and abstract data types like lines, polygons, and bitmaps.

2. **Explain any two applications of spatial database.**

Military:
Operations Spatial data holds crucial importance to the Military Commander in the battle field as it helps in decision-making in the planning and development of a state's growth. Use of GIS in the management of military bases facilitates maintenance of all stores which may be found on the base. "GIS allows military land and facilities managers to reduce base operation and maintenance costs, improve mission effectiveness, provide rapid modelling capabilities for analyzing alternative strategies, and improve communication and to store institutional knowledge."

Farming:
Geographic Information Systems is helpful in being able to map and present current and future changes in rainfall, temperature, crop production and more. By mapping the geographical and geological features of current (and potential) farmlands scientists and farmers could work together in creating more effective and efficient farming techniques; this would increase production of food in different parts of the world that are facing problems in producing enough food for the people around them. GIS helps analyzing soil data which combines with historical farming practices to determine which crops are best to plant, where they should be planted and how to maintain the nutrition level of soil to best benefit the plants.

---

**Outcomes:**

CO 2. Design advanced database systems using Object relational, Spatial and NOSQL databases and its implementation.

---

**Conclusion: (Conclusion to be based on outcomes achieved)**

In this experiment we understood the execution of Spatial Database queries by using PostGIS. We created an extension of postgis and established a connection to the postgres database. The shape file was imported in postgis and the result was displayed by executing the query in postgres. Using a shape file viewer, we determined the area covered by the data.

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of faculty in-charge with date**

**References:**

1. Elmasri and Navathe, "Fundamentals of Database Systems", Pearson Education
2. Raghu Ramakrishnan and Johannes Gehrke, "Database Management Systems" 3rd Edition, McGraw Hill,2002
3. Korth, Silberchatz, Sudarshan, "Database System Concepts" McGraw Hill
4. http://www.bostongis.com/PrinterFriendly.aspx?content_name=postgis_tut01