

**Experiment No.5**

**Title: Execution of Spatial database queries**

Page No:

**Batch: B2 Roll No.:16010421119 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 or distance) 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**](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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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** |

# Range query in Postgis SELECT ST\_Reclass(rast, 1,

SELECT ST\_Distance(geom, 'SRID=3005;POINT(1011102 450541)'::geometry) as d,edabbr, vaabbr

FROM va2005

ORDER BY d limit 10;

d

| edabbr | vaabbr

+ +

**(10 rows)**

'[0-90]:0,(90-100):1,[100-1000):2',

'4BUI', 0) AS rast FROM sometable WHERE filename = '123.tif';

create extension postgis;

select \* from ind\_rails;

select \* from ind\_adm1;

select \* from spatial\_ref\_sys;

SELECT fid\_rail\_d

FROM ind\_rails

WHERE ST\_Equals(

geom,

'010500000001000000010200000004000000A873B97F14E852400892E9BF4322404032FEADFF7BE75240862C4F9F312240401FA7B29F1AE7524067FF5ABFE8214040E7E762E094E65240F61247A085214040');

SELECT ind\_rails.fid\_rail\_d , ind\_rails.gid

FROM ind\_rails , ind\_adm1

WHERE ST\_Within(ind\_rails.geom, ind\_adm1.geom);

SELECT ind\_rails.exs\_descri , ind\_rails.fco\_descri

FROM ind\_rails , ind\_adm1

WHERE ST\_Touches(ind\_rails.geom, ind\_adm1.geom);

SELECT \*

FROM ind\_rails

WHERE ST\_DWithin(geom, ST\_MakePoint(72.9097089,19.0767072), 5);

# Results: (Program printout with output)

# Adding Spatial Database through link

# 

# Connection Status

# 

# Database which is imported is seen in PostGres.

# 

# Importing 2nd File.

# 

# Using ST\_Within

# SELECT ind\_rails.fid\_rail\_d , ind\_rails.gid

# FROM ind\_rails , ind\_adm1

# WHERE ST\_Within(ind\_rails.geom, ind\_adm1.geom);

# 

# Using ST\_Crosses

SELECT ind\_rails.fid\_rail\_d , ind\_rails.gid

FROM ind\_rails , ind\_adm1

WHERE ST\_Crosses(ind\_rails.geom, ind\_adm1.geom);

# 

# 

# Using Range Query

SELECT \*

FROM ind\_rails

WHERE ST\_DWithin(geom, ST\_MakePoint(72.9097089,19.0767072), 5);

# 

**Questions:**

# Explain the spatial functions used for these queries in detail.

# Spatial queries are queries in a spatial database that can be answered on the basis of geometric information only, i.e., the spatial position and extent of the objects involved. A spatial query is defined by a query space S, i.e., either the whole spatial database, or a portion of it obtained through suitable filters; by a query object q that can either belong or not belong to the database; and by a spatial relation ℜ. A generic query is thus defined as follows:

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

**Ans: -**

**1) 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 analysing 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.**

**2) Disaster and Emergency**

**All phases of managing diaster and emergencies**

**depends on data from various sources. The relevant data has**

**to be collected, organized and displayed logically to determine**

**the area (size) and scope of emergency management programs.**

**By the use of GIS, all departments could share information**

**through a database on computer-generated map in one**

**location. Without this capability, emergency workers must**

**must need to have access to a number of department managers,**

**its unique maps and its unique data. Most emergencies do not**

**provide time to gather these resources. This results in**

**emergency responders having to guess, estimate, or make**

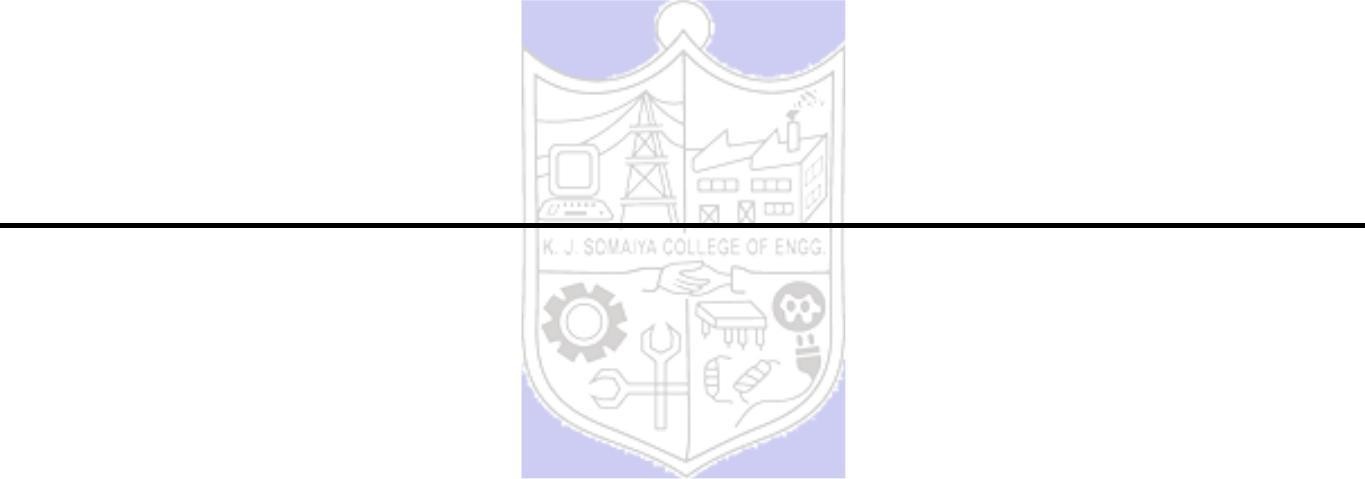
**decisions without adequate information.**

# 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)**

**We can conclude that we have learnt about execution of spatial database queries.**



**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>