International Journal of
Emerging Research in Management &Technology
ISSN: 2278-9359 (Volume-6, Issue-6)

Research Article | **June 2017**

A Comparative Analysis of Spatial Indexing Schemes in PostGreSQL/PostGIS Using Java-API

Monika Yadav
Dept. of Computer Science and Applications, Kurukshetra University,
Kurukshetra, Haryana, India

Abstract-

There are several indexing techniques used in retrieval of spatial data. In this paper a comparative analysis is performed on three spatial indexing techniques- GiST (Generalized Search Tree), SP_GiST (Space partition Generalized Search Tree) and R-tree (Rectangle tree) in spatial database PostgreSQL/Postgre using Java API. The comparison is performed on five categories of spatial and non spatial queries ,namely Simple SQL, Geometry, Spatial Relational ship, Spatial Join and Nearest Neighborhood based on benchmark dataset of New York city. This paper performs experiment in all five categories and compares performance for each category on index structure.

Keywords- GiST, PostgreSQL/PostGIS, R-tree, Spatial index, SP-GiST,

I. INTRODUCTION

Indexing helps in information retrieval in database system [1]. But traditional indexing techniques like B-tree, B+-tree and many other hashing techniques are only suitable for point query or one dimensional data. Some information in databases is represented in form of polygon, lines and points. So there is a need of advanced indexing technique which indexes this type of information. Spatial indexing is used to index spatial database which stores multidimensional data. Spatial database is used to store both fixed objects like roads, buildings, streets and moving objects like vehicles, mobile phones etc [2]. Moving objects database changes over time. Spatial indexing also used in Geographic Information System (GIST) and resource management . All information in spatial indexing is retrieved from SDBMS. Data is retrieved fast with the help of this indexing. However the complexity of spatial indexing is more than traditional indexing techniques because spatial indexing algorithm considers neighbors of objects in order to extract information. There are various spatial indexing techniques are proposed and each of them indexing technique is good for certain purpose. In this paper three spatial indexing techniques are compared on different types of query. This paper compare R-tree, Gist and SP_GiST index on spatial database management system PostgreSQL. Geographic data is present in form of shape files. The shape files are benchmark data of New York city, of 58,505, records that consists of points, lines, polygons and multi-polygons represented by x-y geometry [3].

This paper compares the three indexes GiST, SP_GiST and R-tree on PostgreSQL using JAVA API based on five types of queries like simple SQL, spatial relationship, nearest neighborhoods, spatial join and geometry. The organization of paper is followed. Section 2 gives brief background on indexing structure (GiST, SP_GiST and R-tree),dataset use in experiment and amount of time consumed in the construction of indexes. In section 3 all five different queries are presented. The experimental results are discussed in section 4 followed by conclusion in section 5.

II. BACKGROUND

A. Indexing Structure

a) R-TREE [4]- R-tree is a height balanced tree like B-tree [5]. B-tree is only appropriate for point query. But R-tree is extension of B-tree which used for both point and range query [6]. It helps in find out answer for queries like find all hospitals with in 5 km of my current location; retrieve all road segments within 8 km of my location. The queries in R-tree are executed in fraction of second. Its index is also like B-tree i.e. it contains root node, child nodes. A node in R-tree contains a maximum level and node becomes split when maximum level reached. Every node in R-tree corresponds to a disk page in the disk. R-tree makes a group of near objects called MBR. Let M be the maximum number of entries fit in one node and m<=M/2 be the minimum number of entries in a node then R-tree satisfy following properties [2]-

a) Each leaf node index records are between m and M.
b) In each leaf node index is (I, tuple-identifier) where I is smallest rectangle that spatially includes n-dimensional data object.
c) Each non-leaf node children are between m and M unless it is root.
d) In each non leaf index is (I, child-pointer) where I is smallest rectangle that spatially includes the rectangles in child node.
e) In R-tree root node contains at least two children unless it is leaf.
f) All leaves in R-tree ate at same level.

b) GiST [7]**:** GiST or Generalized Search Tree, is a data structure that can be used to build a variety of disk-based search trees. Generalized Search Tree is a generalization of the B+ tree, providing a concurrent and recoverable height-balanced search tree infrastructure without making any assumptions about the type of data being stored. It supports wide range of data types. GiST can also support nearest-neighbor search, and various forms of statistical approximation over large data sets. The PostgreSQL GiST implementation includes support for variable length keys, composite keys, concurrency control and recovery; these features are inherited by all GiST extensions.

c) SP_GiST [8]**:** SP-GiST is an abbreviation for space-partitioned GiST. SP-GiST supports partitioned search trees, which facilitate development of a wide range of different non-balanced data structures, such as quad-trees [9], k-d trees [10], and suffix trees (tries). The common feature of these structures is that they repeatedly divide the search space into partitions that need not be of equal size. Searches that are well matched to the partitioning rule can be very fast. The challenge addressed by SP-GiST is to map search tree nodes to disk pages in such a way that a search need access only a few disk pages, even if it traverses many nodes.Like GiST, SP-GiST is meant to allow the development of custom data types with the appropriate access methods, by an expert in the domain of the data type, rather than a database expert.

B. Spatial Database

a) PostgreSQL- It is a powerful and open source object relational database system. It runs on operating system like Linux, Unix and windows. PostgreSQL is fully ACID compliant and supports SOL standards. PostgreSQL is interface for C/C++, java, .Net, Python, Ruby etc. It supports indexes like B-tree, R-tree, hash or gist storage methods.

b)PostGIS- PostGIS is an extension of PostgreSQL. It supports spatial database and add spatial functions like distance, area, union, intersection and geometry in databases to improve the performance of the databases. In this paper indexes R-tree and GIST are performed on postGIS/PostgreSQL.

C. Dataset

The spatial data use in this paper is set of benchmark data of New York City. In this set subway stations are represented as point data, streets and subway lines are represented as line data, neighborhoods are represented as polygon data and population data represented as non spatial data. There are four tables in this dataset [3]-

1) nyc_census_blocks- This table contains information about population (denoted by popn) for each block (blk).
2) nyc_neighborhoods- This table contain information about neighborhoods.
3) nyc_streets- This table contains streets name, types and other properties like one-way.
4) nyc_subway_stations- This table contains station information as to which subway line run through (routers) and can transfer to via this station. It also contains whether station is express train stop.

D. Index Construction Time

Three types of index i.e. Gist, SP_Gist and R-tree are created on benchmark data in this paper for comparison. Some index can take less time to create and some other take more time. Here Fig1 shows the average execution time for creating each index on different tables. Time should be noted from vertical axis in milliseconds for each index and tables are presented on horizontal axes. All the four tables are indexed on the geometry column.

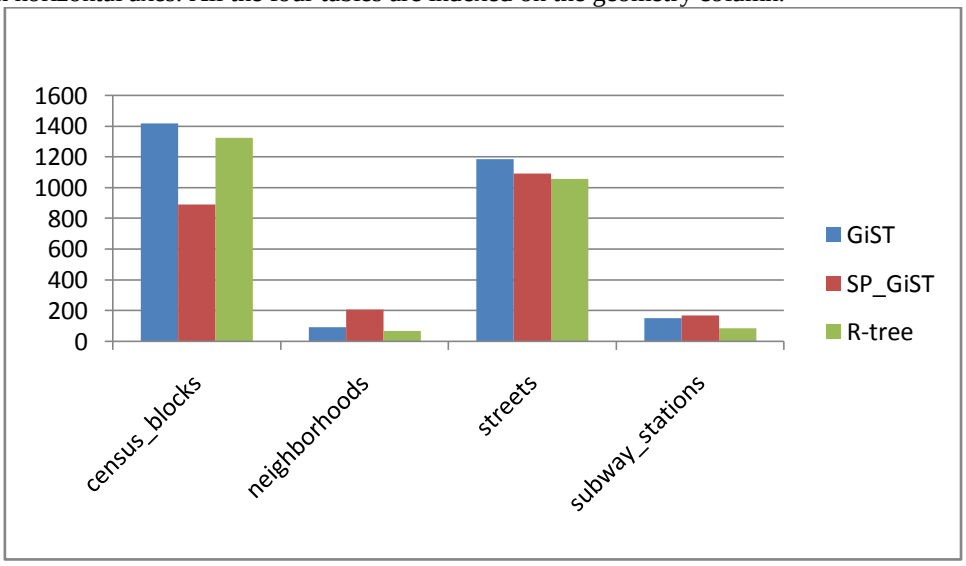

Fig 1 Graph of the time taken by each index

III. SPATIAL QUERIES

The spatial queries on this dataset are divided into five categories and these categories are followings-

1) Simple SQL-These are the queries which do not have spatial data. These are only simple SQL queries and these are given in appendix A (Q1-Q6).
2) Geometry- These are spatial queries which includes measurements like length and area. These are given in appendix B (Q7-Q13).

3) Spatial Relationship Topological- It contains spatial queries which are based on topological spatial relationship and these queries are given in appendix C (Q14-Q17).
4) Spatial join-These are the queries which help in combine information from different tables. These queries are given in appendix D (Q18-Q20).
5) Nearest Neighborhoods-These queries helps in find objects which are close to given object. These are given in appendix E (Q21-Q22).

IV. EXPERIMENTAL RESULTS

In this section, graphs are shown for all five types of queries, with/without an index of the three indexing structures. In this experiment, execution time (in milliseconds) is noted for spatial indexing techniques in JavaAPI connected with PostgreSQL/Postgis.

A. Simple SQL:

The performance results for the simple SQL queries are shown in figure 2 through figure 5. Fig 2 shows graph for without index SQL queries and Fig 3 to Fig 5 shows results for GiST, SP_GiST and R-tree respectively.

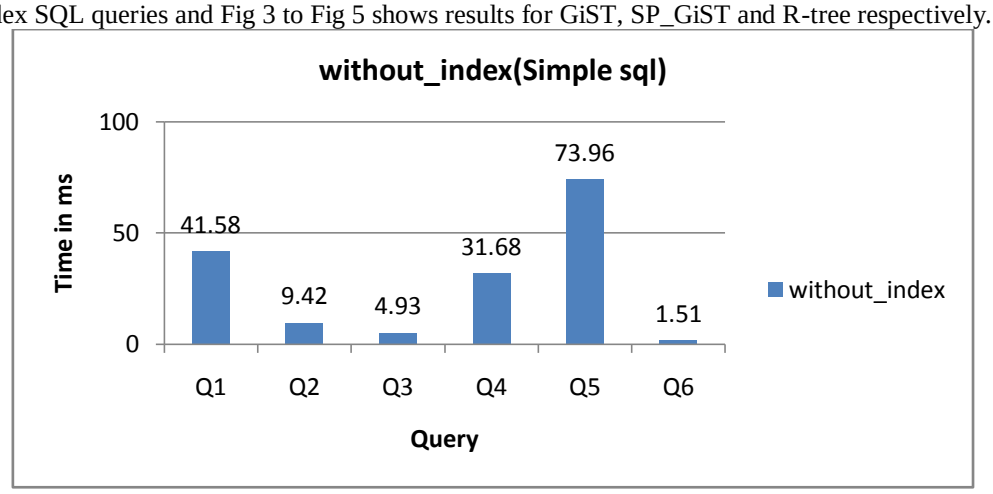

Fig 2. Time Taken without index(Simple SQL)

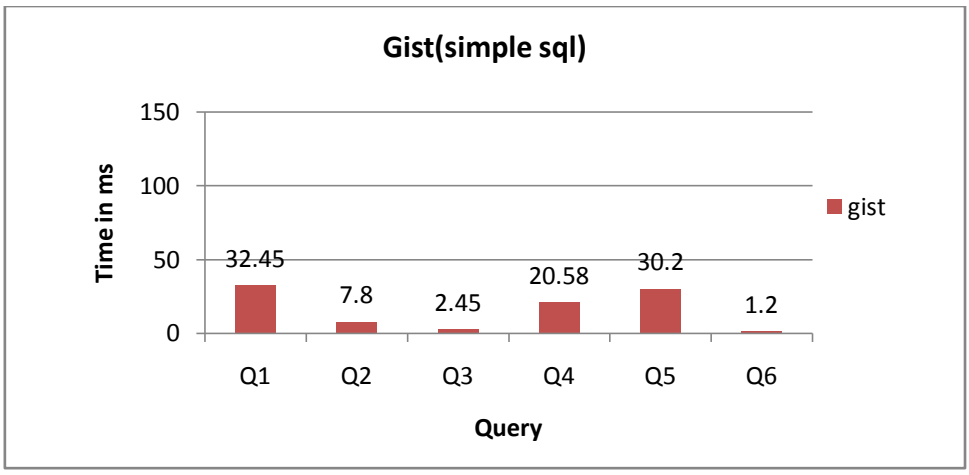

Fig 3. Time Taken GiST index(Simple SQL)

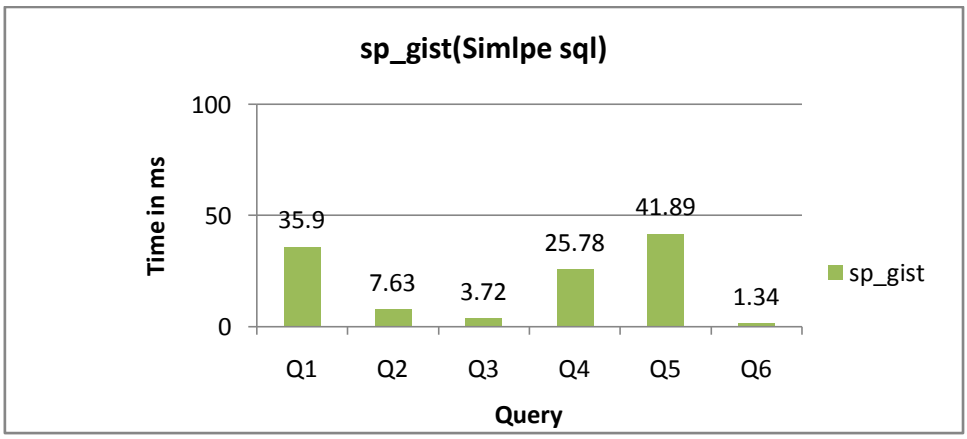

Fig 4. Time Taken SP_GiST index(Simple SQL)

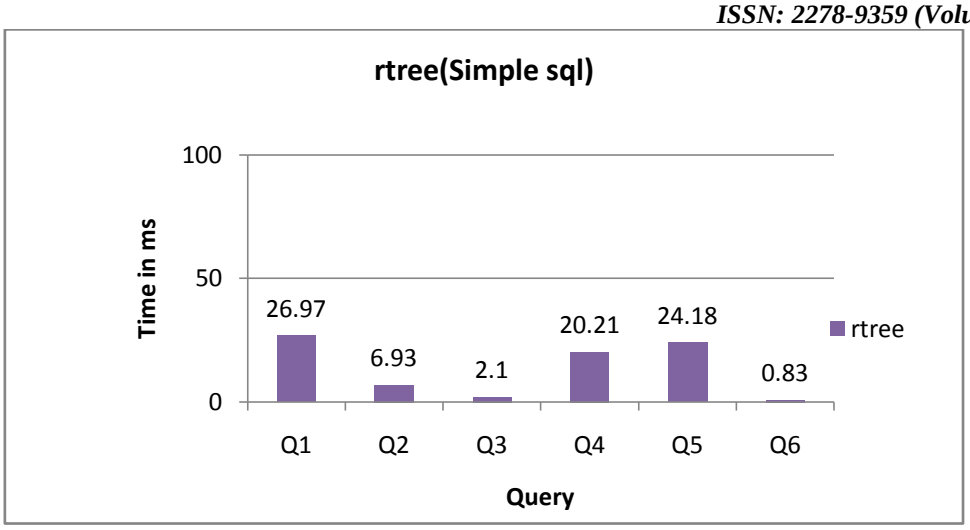

Fig 5. Time Taken R-tree index(Simple SQL)

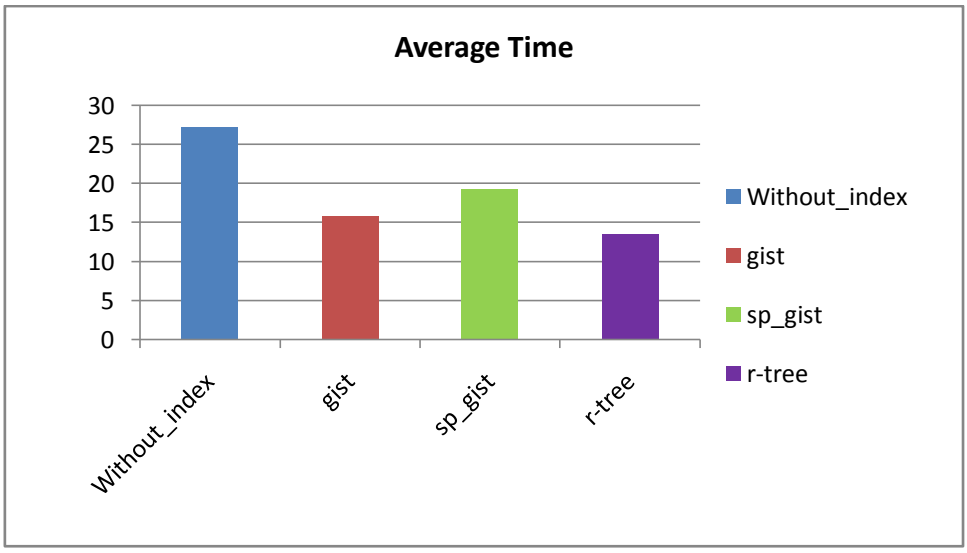

Fig 6 Average of all indexes(Simple SQL)

By all the graphs presented above there is no single best method for executing simple SQL queries. Fig 6 shows an average time graph for all indexes and from this, it is easier to determine the most efficient indexing structure for Simple SQL. Here R-tree takes less time for executing Simple SQL queries.

B. Geometry(Metric Queries):

Fig 7 shows time taken by Geometry queries without index.Fig 8 through fig 10 shows the time taken when indexed using GiST, SP_GiST and R-tree respectively.

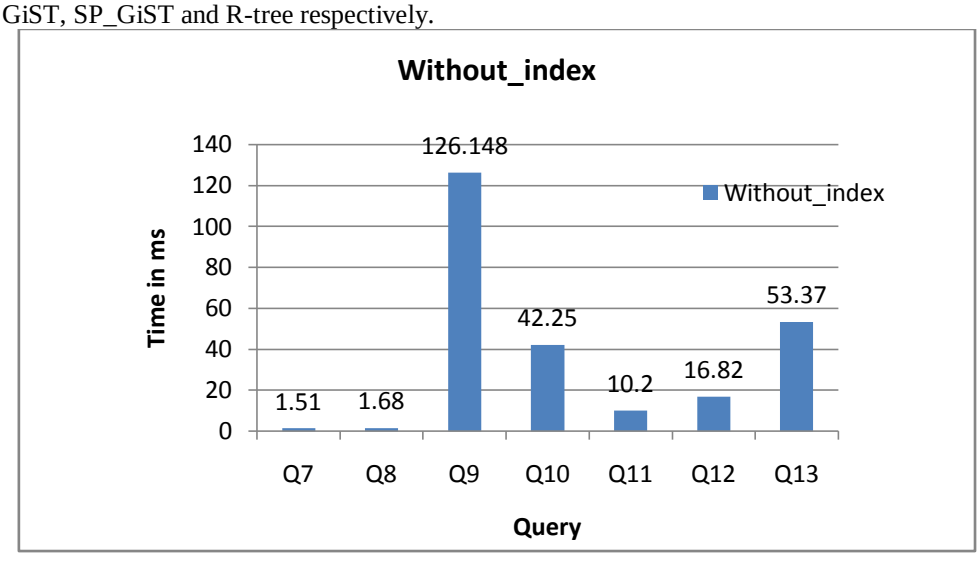

Fig 7. Time taken without index(Geometry)

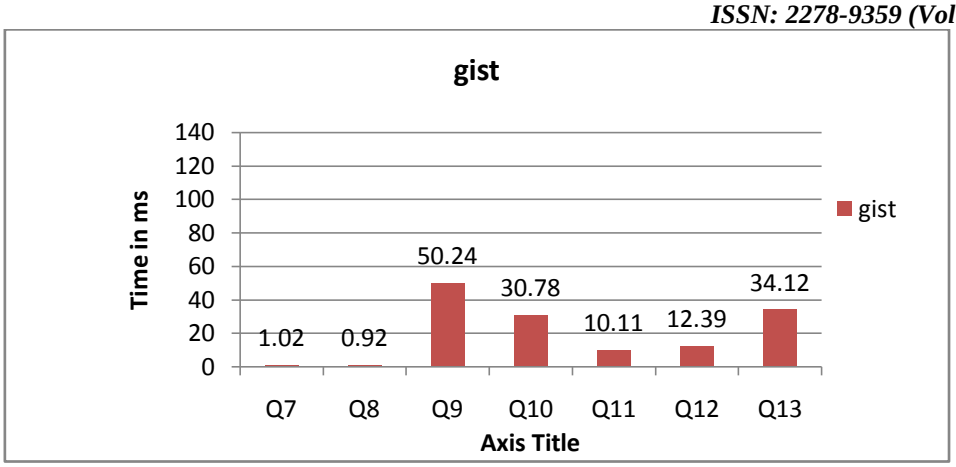

Fig 8. Time taken GiST index(Geometry)

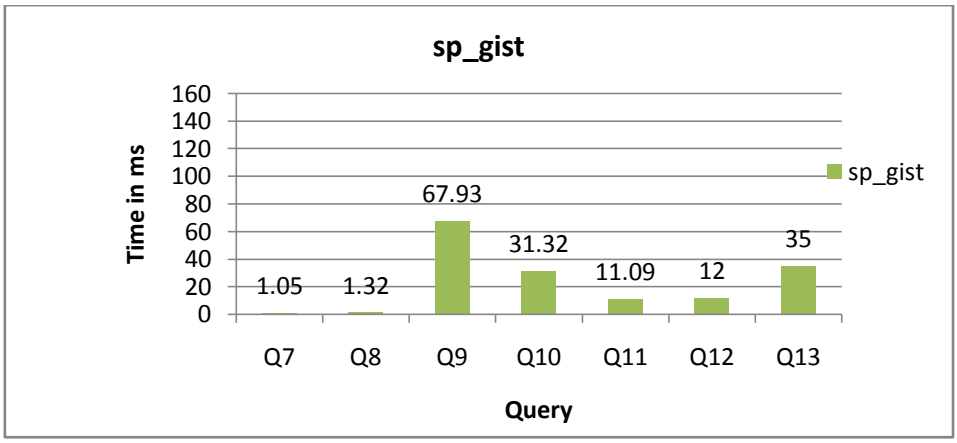

Fig 9. Time taken SP_GiST index(Geometry)

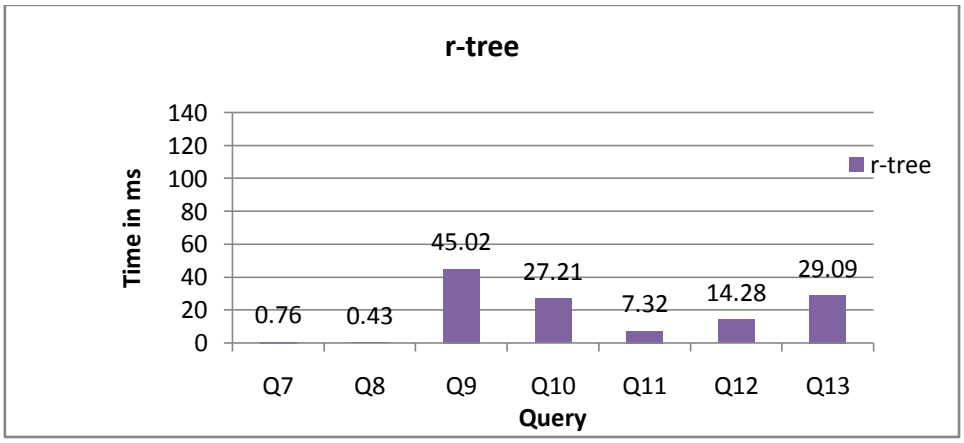

Fig 10. Time taken R-tree index(Geometry)

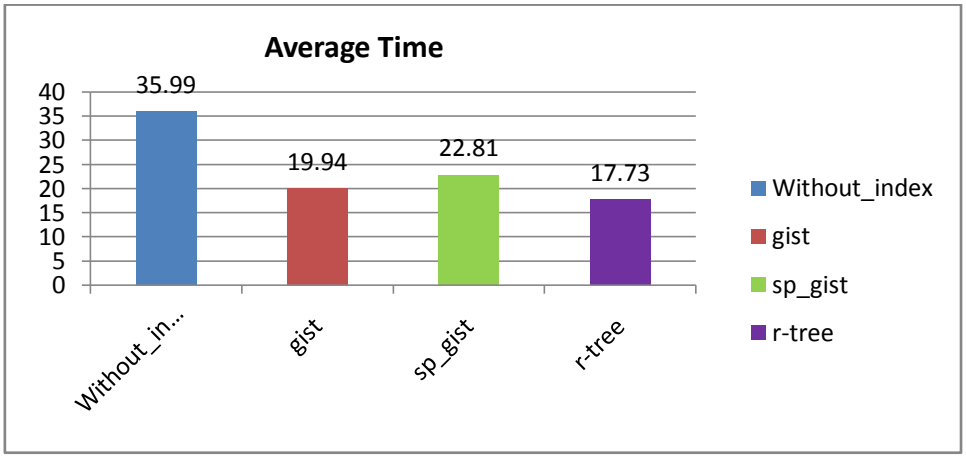

Fig 11. Average time taken by all indexes (Geometry).

According to the average time graph it is shows that R-tree is the best indexing scheme for geometry queries. The average time taken by R-tree is 17.73ms. Gist also gives better performance which takes 19.94ms.

C. Spatial Relationship

The time taken without index is shows in Fig 12 for queries Q14 and Q15.Fig 13 to Fig 15 shows time taken by GiST, SP_GiST and R-tree respectively.

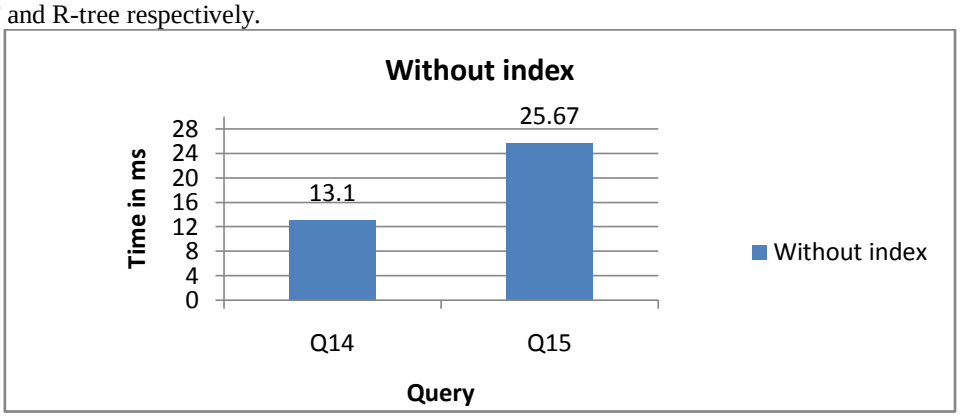

Fig 12 Time taken without index (Spatial Relationship)

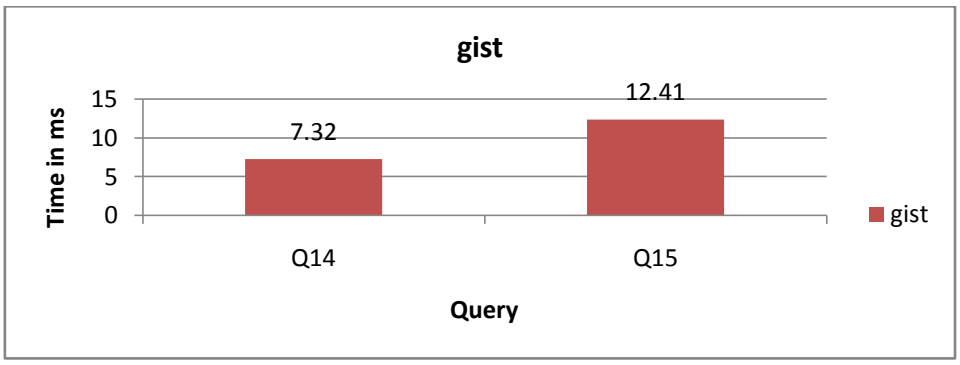

Fig 13 Time taken GiST index(Spatial Relationship)

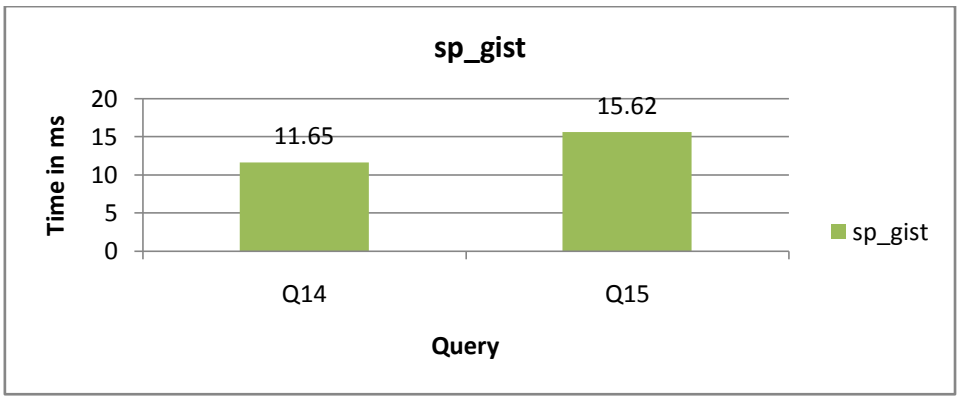

Fig 14 Time taken SP_GiST index(Spatial Relationship)

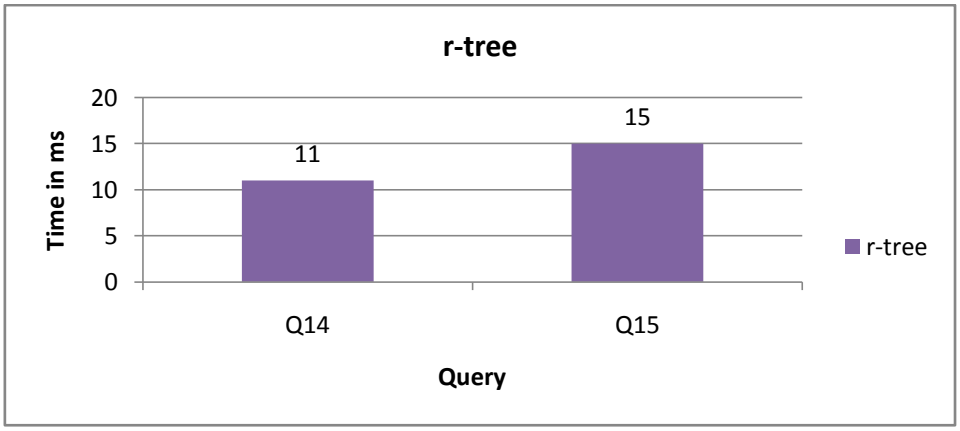

Fig 15 Time taken R-tree index(Spatial Relationship)

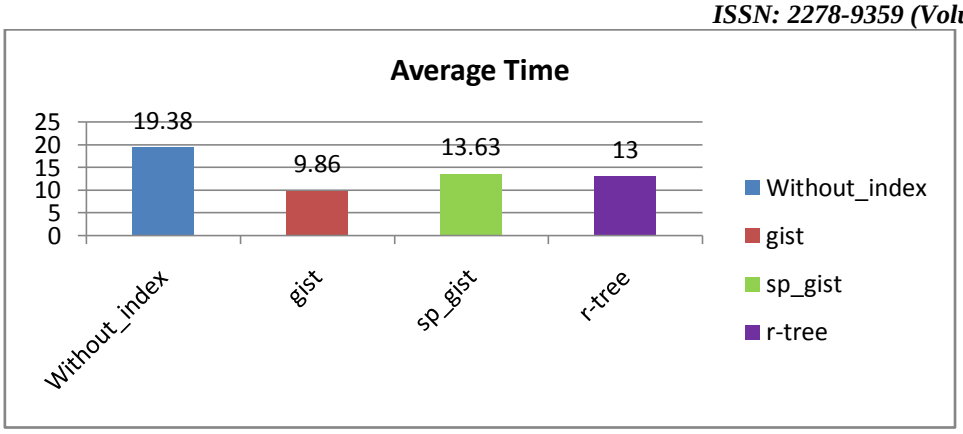

Fig16 Average time taken by indexes(Spatial Relationship)

From the average graph this shows that GiST performs best for Spatial Relationship queries.

D. Spatial Join

The time taken without index is shown in Fig 17 and Fig 18 to Fig 20 demonstrates the total time taken to execute the queries of Spatial join by three different indexing structures.

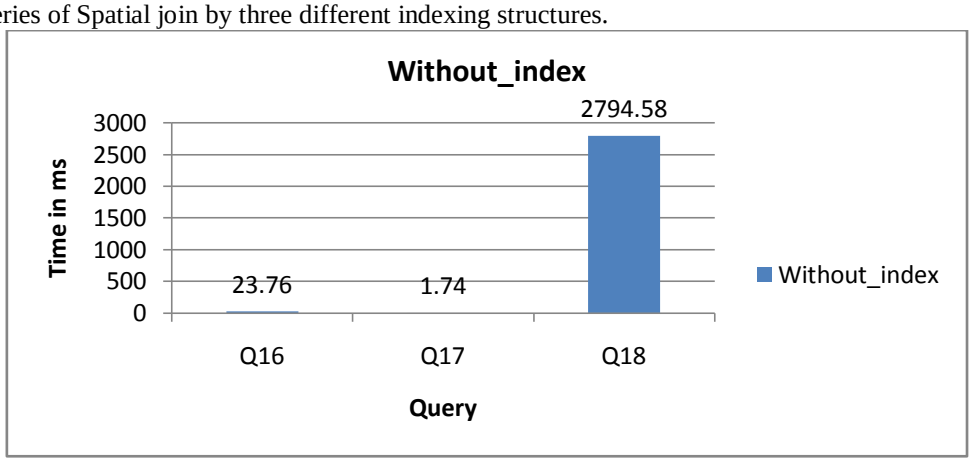

Fig 17 Time taken without index(Spatial Join)

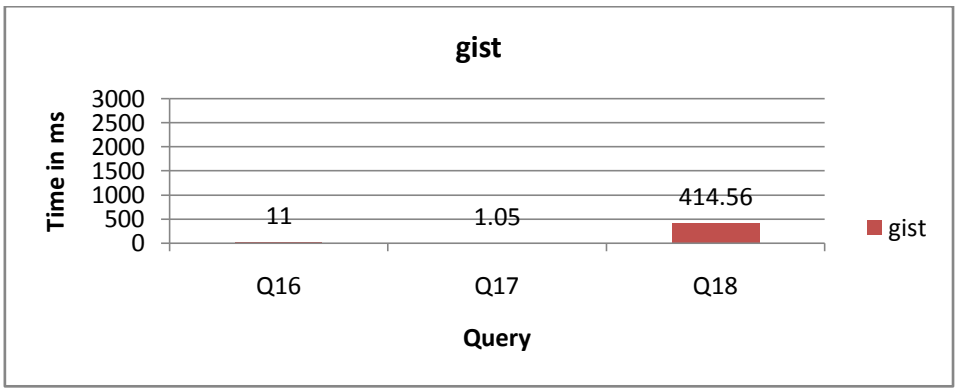

Fig 18 Time taken GiST index(Spatial Join)

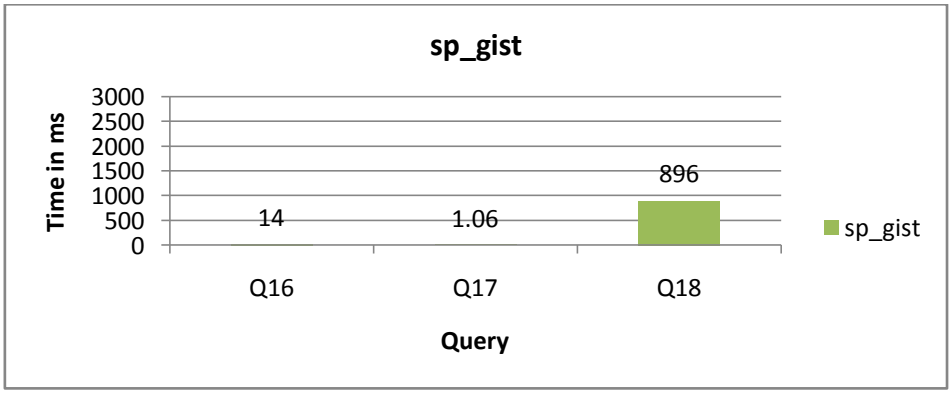

Fig 19 Time taken SP_GiST index(Spatial Join)

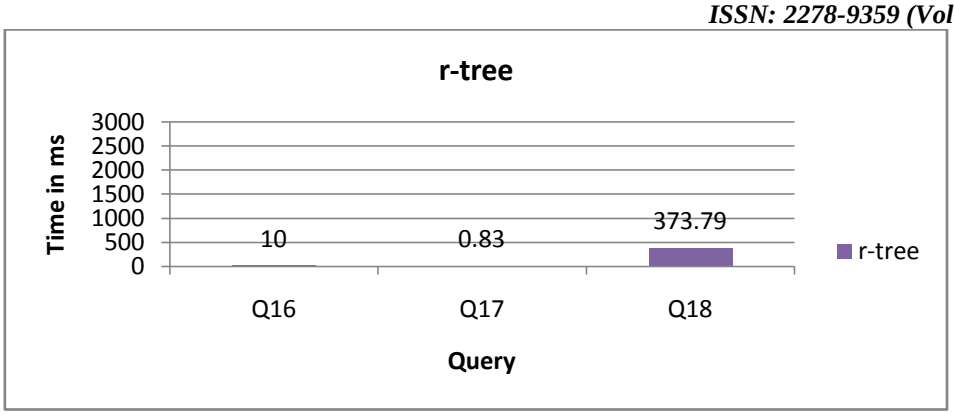

Fig 20 Time taken R-tree index(Spatial Join)

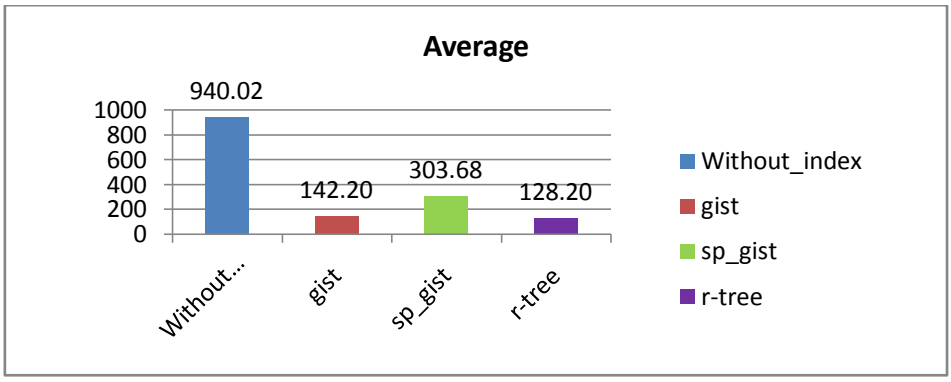

Fig 21 Average time taken by all indexes(Spatial Join).

In spatial join this can notice that Q18 on an average took a lot of time without or with index.The query was about finding the toatal population and racial makeup of all the 28 neighborhoods of Manhattan borough so it was natural for the database to take some time to compute the query. Here R-tree gives best performance among all.

E. Nearst Neighbor

The time taken in Fig 22 shows without index Nearst Neighbor queries and Fig 22 to Fig 25 shows the execution time when indexd using GiST, SP_GiST and R-tree respectively.

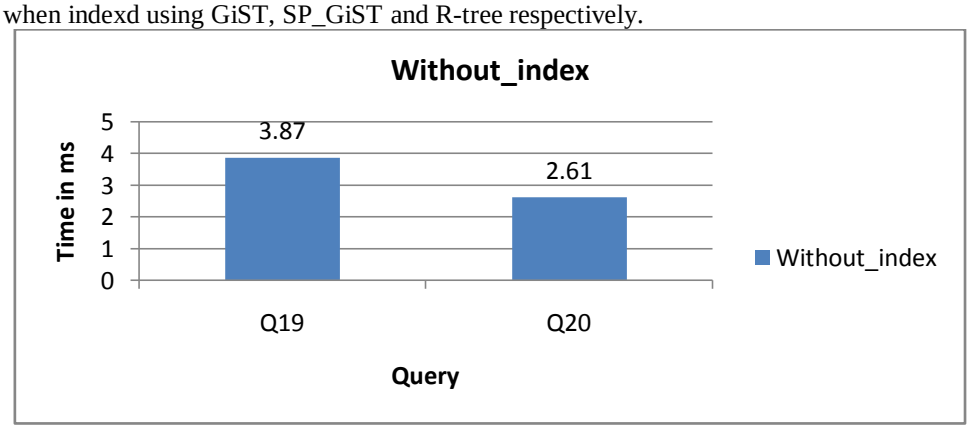

Fig 22 Time taken without index(Nearest Neighbor)

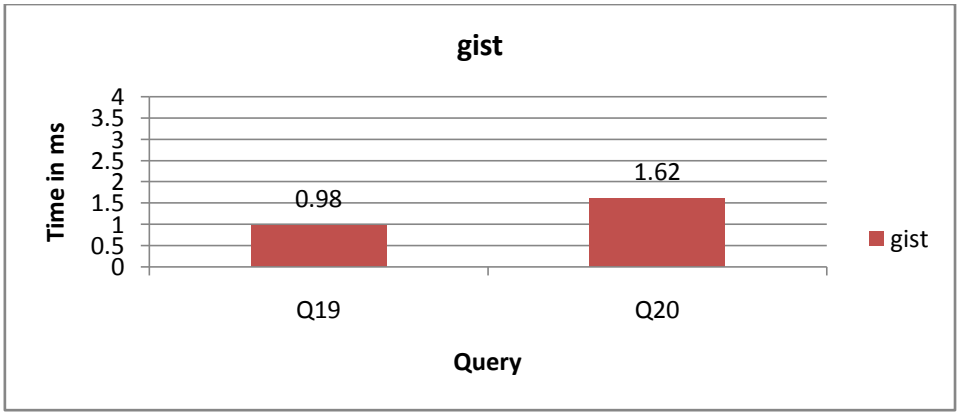

Fig 23 Time taken GiST index(Nearest Neighbor)

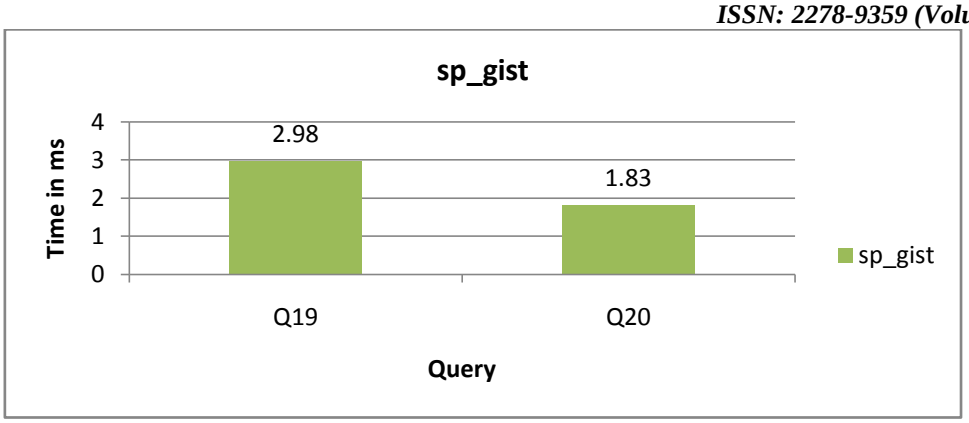

Fig 24 Time taken SP_GiST index(Nearest Neighbor)

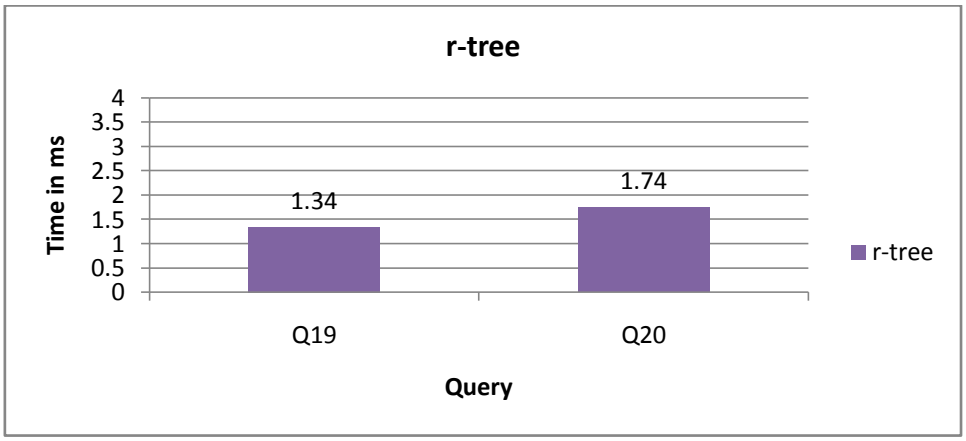

Fig 25 Time taken R-tree index(Nearest Neighbor)

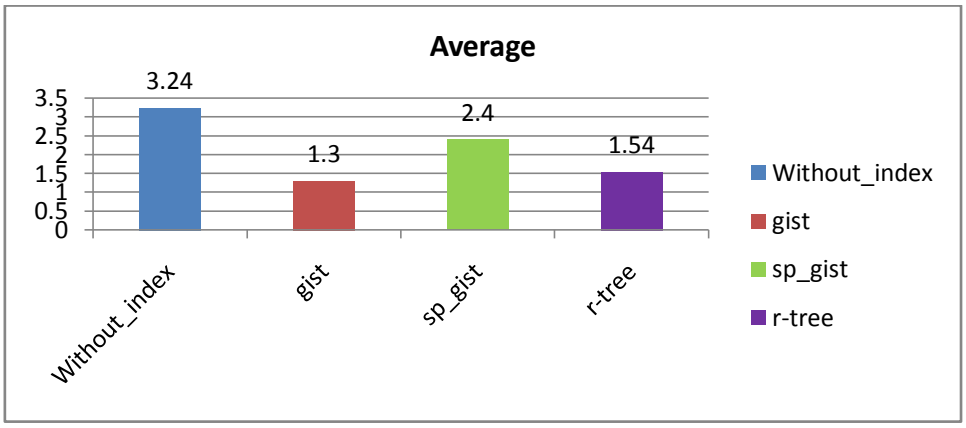

Fig 26 Average time taken by all indexex(Nearst Neighborhood)

As shown in Fig 26 it is shows that GiST gives best performance in order to execute the Nearest Neighbor.

V. CONCLUSION

This paper compare the performance of three different spatial indexing structure for five different categories of queries.This comparison is performed on database PostgreSQL/PostGRE.PostGIS is spatial extension of PostgreSQL.By these experiments it is shown that R-tree gives best performance in Simlpe SQL and Geometry because arithmetic computation performed fast in R-tree whereas GiST gives best performance in Spatial Relation Ship and Nearest Neighbor queries because GiST supports neighbor search better than R-tree.In future one can build a spatio-temporal indexing structure which index dynamic data with an additional dimension of time.

REFERENCES

[1] Henry F.Korth,S.Sudarshan A. Silberschatz, Database System Concepts, 5th ed.: McGraw-Hill, 2006.
[2] S.R Lavanya S.Srividhya, "Comparitive Analysis of R-Tree and R+-Tree in Spatial Database," in International Conference on Intelligent Computing Applications, 2014, pp. 449-453.
[3] M.Fouladgar, R.Elmasri, K.Jitkajornwanich N.Pant, "Performance Comparison of SpatialL Indexing Structure for Different Query Types," in Proceedings of 57th IRF International Conference, Pune, india, 2016, pp. 43-50.

[4] A. Guttman, "R-Trees:A Dynamic Index Structure For Spatial Searching," ACM SIGMOD, pp. 47-57, June 1984.
[5] K. Unterauer R. Bayer, "Prefix B-Trees," ACM Transactions on Database Systems, vol. 2, no. 1, pp. 11-26, March 1977.
[6] D.Garg P.Patel, "Comparison of Advance Tree Data Structure," International Journal of Computer Applications, vol. 41, no. 2, pp. 11-20, March 2012.
[7] Jeffrey F. Naughton, Avi Pfeffer Joseph M. Hellerstein, "Generalized Search Trees for Database Systems," in Proceedings of the 21st VLDB Conference Zurich, Switzerland, 1995.
[8] Ihab F.ilyas Walid G.Aref, "SP-GiST: An Extensible Database Index for Supporting Space Partitioning Trees," Journal of Intelligent Information Systems, December 2001.
[9] J. L. Bentley R. A. Finkel, "Quad Trees : A Data structure for Retrieval on Composite Keys," Springer-Verlag, vol. 4, pp. 1-9, March 1974.
[10] J. L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," ACM, vol. 18, no. 9, pp. 509-517, September 1975.

Spatial and non-Spatial Queries-

Appendix A

Q1: Select name from nyc_neighborhoods

```
SELECT name
FROM nyc_neighborhoods;
```

Q2: Select all the neighborhood names which are under ‘Manhattan’ borough.

```
SELECT name
FROM nyc_neighborhoods
WHERE boroname = ‘Manhattan’;
```

Q3: Find number of letters in all the neighborhood names in Brooklyn.

```
SELECT char_length (name)
FROM nyc_neighborhoods
WHERE boroname = 'Brooklyn';
```

Q4: What is the population of the city of New York?

```
SELECT Sum (popn_total) AS population
FROM nyc_census_blocks;
```

Q5: Find the total population of the borough The Bronx.

```
SELECT Sum (popn_total) AS population
FROM nyc_census_blocks
WHERE boroname = 'The Bronx';
```

Q6: Find the percentage of white people for each borough.

```
SELECT boroname
100*Sum (popn_white) / Sum (popn_total) AS white_pct
FROM nyc_census_blocks
GROUP BY boroname;
```

Appendix B

Q7: Compute the area of the ‘West Village’ neighborhood.

```
SELECT ST_Area (the_geom)
FROM nyc_neighborhoods
WHERE name = 'West Village';
```

Q8: Compute the area of ‘Manhattan’ in acres. (The unit given to us in the data is in meters)

```
SELECT Sum (ST_Area (the_geom) ) / 4047
FROM nyc_neighborhoods
WHERE boroname = 'Manhattan';
```

Q9: Compute the number of the census blocks with hole in New York City

```
SELECT Count (*)
FROM nyc_census_blocks
WHERE ST_NumInteriorRings (ST_GeometryN(geom,1) ) > 0;
```

Q10 Find the total length of all the streets in New York City in Kilometers.


```
SELECT Sum (ST_Length (the_geom) ) / 1000
FROM nyc_streets;
```

Q11: Find the length of the street 'Columbus Cir'.

```
SELECT ST_Length (the_geom)
FROM nyc_streets
WHERE name = 'Columbus Cir';
```

Q12: What is the JSON representation of the boundary of 'West Village'?

```
SELECT ST_AsGeoJSON (the_geom)
FROM nyc_neighborhoods
WHERE name = 'West Village';
```

Q13: Summarized by the type, calculate the length of the streets in New York.

```
SELECT type, Sum(ST_Length(the_geom)) AS length
FROM nyc_streets
GROUP BY type
ORDER BY length DESC;
```

Appendix C

Q14: For the street named 'W Lake Dr' find the geometry value.

```
SELECT ST_AsText (the_geom)
FROM nyc_streets
WHERE name = 'W Lake Dr';
```

Q15 For the street named 'Walsh Ct' find the geometry value.

```
SELECT ST_AsText (the_geom)
FROM nyc_streets
WHERE name = 'Walsh Ct';
```

Appendix D

Q16: Find the distance between 'Columbus Cir' and 'Fulton Ave'.

```
SELECT ST_Distance (ST_GeomFromText (( SELECT ST_AsText (the_geom)
FROM nyc_streets
WHERE name = 'Columbus Cir'), 26918),
ST_GeomFromText (( SELECT ST_AsText (the_geom )
FROM nyc_streets
WHERE name = 'Fulton Ave'), 26918)
) / 1000 as Distance_in_Kms;
```

If we look carefully, in this query the user first tries to find WKT representation of Columbus Cir and Fulton Ave, then, by using the function ST_Distance calculates the distance between them.

Q17: Find the neighborhood of 'South Ferry' subway station.

```
SELECT
nyc_subway_stations.name,
nyc_neighborhoods.name,
nyc_neighborhoods.boroname
FROM nyc_neighborhoods
JOIN nyc_subway_stations
ON ST_Contains (nyc_neighborhoods.the_geom,nyc_subway_stations.the_geom )
WHERE nyc_subway_stations.name = 'South Ferry';
```

Q18: What is the population and racial make-up of the neighborhoods of Manhattan?

```
SELECT
nyc_neighborhoods.name,
Sum (nyc_census_blocks.popn_total),
100.0 * Sum(nyc_census_blocks.popn_white) /Sum(nyc_census_blocks.popn_total),
100.0 * Sum(nyc_census_blocks.popn_black) /Sum(nyc_census_blocks.popn_total)
FROM nyc_neighborhoods
JOIN nyc_census_blocks
ON
ST_Intersects(nyc_neighborhoods.the_geom,nyc_census_blocks.geom)
WHERE nyc_neighborhoods.boroname ='Manhattan'
```



```
GROUP BY nyc_neighborhoods.name
ORDER BY white_pct DESC;
```

Appendix E

Q19: What subway station is in ‘Bensonhurst’?

```
SELECT s.name, s.routes
FROM nyc_subway_stations AS s
JOIN nyc_neighborhoods AS n
ON ST_Contains(n.the_geom, s.the_geom)
WHERE n.name = 'Bensonhurst';
```

Q20: What is the closest street to ‘Cortlandt’ subway station?

```
SELECT streets.gid, streets.name
FROM nyc_streets streets, nyc_subway_stations subways
WHERE subways.name = 'Cortlandt'
ORDER BY ST_Distance(streets.the_geom, subways.geom)
ASC
LIMIT 1;
```

