CSE 511 Project Phase 1 Report

Team Members (Group 33):

Janam Vaidya - jvaidya4@asu.edu
Vishal Krishna Bettadapur - vbettada@asu.edu
Khushank Goyal - kgoyal7@asu.edu
Shreeshiv Patel - spate137@asu.edu

Abstract:

This progress report has been written to define the problem mentioned in phase 1 of the course project and our attempt at developing a solution for it.

Problem Statement:

The problem mentioned in the Project Milestone 2 states that a peer-to-peer taxicab firm wants to develop and run multiple spatial queries on their database. Their database contains geographic data and the real-time location of their customers.
Spatial Queries: Queries in spatial databases that take into consideration geometrical information.
The goal is to write two user-defined functions ‘ST_Contains’ and ‘ST_Within’ in Spark SQL to run the following spatial queries:

- Range Query: Given a query rectangle and a set of points, find all points which are in the rectangle.
- Range join Query: Given a set of rectangles and a set of points, find all pairs such that the point is in a rectangle.
- Distance Query: Given a point and distance, find all points that lie within the distance
- Distance join query: Given two sets of points and distance, find all point pairs that are within the distance.

Function Implementations:

We write two user-defined functions (ST_Within and ST_Contains) and implement them in SpatialQuery.scala.

ST_Within:

From the 2 given points, we collect the x and y coordinates. Let's represent the coordinates of the first point with x1 as the x-coordinate and y1 as the y-coordinate. Similarly, let's say x2 and y2 are the coordinates of the second point. The distance between these points can be calculated using the formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

ST_Within returns a boolean value depending on the value of d. If the value of d calculated using the above formula is less than or equal to the given value of distance, which means that the first point is within a given distance from the second point, then the boolean value true is returned.

If the value of d calculated using the above formula is greater than the given value of distance, which means that the first point is not within a given distance from the second point, then the boolean value of false is returned.

ST_Contains:

We first collect the x and y coordinates of the rectangle and the point given. The rectangle string is split into two points since the rectangle string has two points that are situated at the end of one of the diagonals of the given rectangle. Let's say that these coordinates are rx1, ry1 for one end of the diagonal and rx2, ry2 for the other end. Let's also represent the x and y coordinates of the given point using px and py respectively. The next step in the algorithm requires us to find the minimum and maximum coordinates among the rectangle coordinates. Then we find if the point lies within the rectangle.

if px is > rectangles minimum x coordinate and px is < rectangles maximum x coordinate
Similarly, for py, if py is > rectangles minimum y coordinate and py is < rectangles maximum y coordinate.

If the above conditions are satisfied, a boolean value of true is returned indicating that the given point lies inside the rectangle. Otherwise, a boolean value of false will be returned thereafter indicating that the rectangle does not contain the point.

Spatial Queries:

Below are the four spatial queries that are implemented with the help of the two user-defined functions mentioned above.

1). Range Query:

We use the ST_Contains function for this query. This query takes the input of a set of points P, and a rectangle R and returns all those points that lie within the rectangle.

2). Range join query:

This query takes input from a set of rectangles R and a set of points P. The query returns all such rectangles, and point pairs satisfying the condition that a point p1 from P lies within a rectangle r1 from R.

3). Distance query:

We use the ST_Within function for this query. This query takes input a point P and a distance D and returns all those points that lie within a distance D from point P.

4). Distance join query:

This query takes two sets of points P1 and P2, and a distance D as input. The query will return all the pairs of points p1, p2 that satisfy the condition that p1 belongs to P1, p2 belongs to P2 and p1 is within distance D from point p2.

Testing:

After installing the required tools and integrating the above method definitions in the template. We used IntelliJ IDE for debugging and testing. To test the project locally:

1) Append.master("local[*]") after .config("spark.some.config.option", "somevalue") in SparkSQLExample.scala

2) Change “provided” to “compile” in build.sbt

3) Add configuration to run with the following settings:
• Provide module name
• Select the Java version to use
• Add the main class. (packagename.classname)
• Provide the arguments as given in the example input file

4) Run with the above configuration This creates the result folder and 4 subfolders namely output0, output1, output2, output3. Each of these folders will contain the required csv files along with other.crc and .csv.crc files. Compare the values in the csv file with the ones in the exampleanswer file.

We got the expected answer as per the exampleanswer file indicating the code is working as expected. Later on, we provided custom values in the arguments and manually calculated to check if it's working as expected with these other values as well and it did work as expected.

Appendix (Phase 1 Results):

Range Query

```
+--------------------+
|                 _c0|
+--------------------+
|-93.579565,33.205413|
|-93.417285,33.171084|
|-93.493952,33.194597|
|-93.436889,33.214568|
+--------------------+
```

Range Join Query

```
+--------------------+--------------------+
|                 _c0|                 _c0|
+--------------------+--------------------+
|-93.63173,33.0183...|-93.579565,33.205413|
|-93.63173,33.0183...|-93.417285,33.171084|
|-93.63173,33.0183...|-93.493952,33.194597|
|-93.63173,33.0183...|-93.436889,33.214568|
|-93.595831,33.150...|-93.491216,33.347274|
|-93.595831,33.150...|-93.477292,33.273752|
|-93.595831,33.150...|-93.420703,33.466034|
|-93.595831,33.150...|-93.571107,33.247214|
|-93.595831,33.150...|-93.579235,33.387148|
|-93.595831,33.150...|-93.442892,33.370218|
|-93.595831,33.150...|-93.579565,33.205413|
|-93.595831,33.150...|-93.573212,33.375124|
|-93.595831,33.150...|-93.417285,33.171084|
|-93.595831,33.150...|-93.577585,33.357227|
|-93.595831,33.150...|-93.441874,33.352392|
|-93.595831,33.150...|-93.493952,33.194597|
|-93.595831,33.150...|-93.436889,33.214568|
|-93.595831,33.150...|-93.437081,33.360932|
|-93.442326,33.248...|-93.242238,33.288578|
|-93.442326,33.248...|-93.224276,33.320149|
+--------------------+--------------------+
only showing top 20 rows
```

Distance Query

```
+--------------------+
|                 _c0|
+--------------------+
|-88.331492,32.324142|
|-88.175933,32.360763|
|-88.388954,32.357073|
| -88.221102,32.35078|
|-88.323995,32.950671|
|-88.231077,32.700812|
|-88.349276,32.548266|
|-88.304259,32.488903|
| -88.182481,32.59966|
|-87.534883,31.934442|
| -87.49702,31.894541|
|-88.153618,33.261297|
|-87.586341,31.959751|
| -87.43091,31.901283|
|-87.989825,33.138512|
|-88.279714,33.056158|
|-87.849593,32.514133|
|-87.727727,32.072313|
|-87.997666,32.067377|
|-87.754018,31.933427|
+--------------------+
only showing top 20 rows
```

Distance Join Query

```
+--------------------+--------------------+
|                 _c0|                 _c0|
+--------------------+--------------------+
|-88.331492,32.324142|-88.331492,32.324142|
|-88.331492,32.324142|-88.388954,32.357073|
|-88.331492,32.324142|-88.383822,32.349204|
|-88.331492,32.324142| -88.384664,32.34299|
|-88.331492,32.324142|-88.401397,32.341222|
|-88.331492,32.324142|-88.414987,32.338364|
|-88.331492,32.324142|-88.277689,32.310778|
|-88.331492,32.324142|-88.382818,32.319915|
|-88.331492,32.324142|-88.366119,32.402014|
|-88.331492,32.324142|-88.265642,32.359191|
|-88.175933,32.360763|-88.175933,32.360763|
|-88.175933,32.360763| -88.221102,32.35078|
|-88.175933,32.360763|-88.158469,32.372466|
|-88.175933,32.360763|-88.133374,32.367435|
|-88.175933,32.360763|-88.265642,32.359191|
|-88.388954,32.357073|-88.331492,32.324142|
|-88.388954,32.357073|-88.388954,32.357073|
|-88.388954,32.357073|-88.383822,32.349204|
|-88.388954,32.357073| -88.384664,32.34299|
|-88.388954,32.357073|-88.401397,32.341222|
+--------------------+--------------------+
only showing top 20 rows
```