DALHOUSIE UNIVERSITY

CSCI 3901: Software Development Concepts
Fall 2022

"Project"

Professor
Dr. Michael McAllister

Submitted by: Mudra Verma
Banner ID: B00932103
Email ID: mudraverma@dal.ca

December 15, 2022

Problem:

Design a system that will manage power service among postal codes. It will record hubs that have damage and people who are experiencing power outages due to them. One postal code can have multiple hubs and one hub can service multiple postal codes. A repair plan needs to be generated for an employee to follow to fix the hubs. The path will be respect to the maximum impact that can be achieved in given time frame.

Database Design

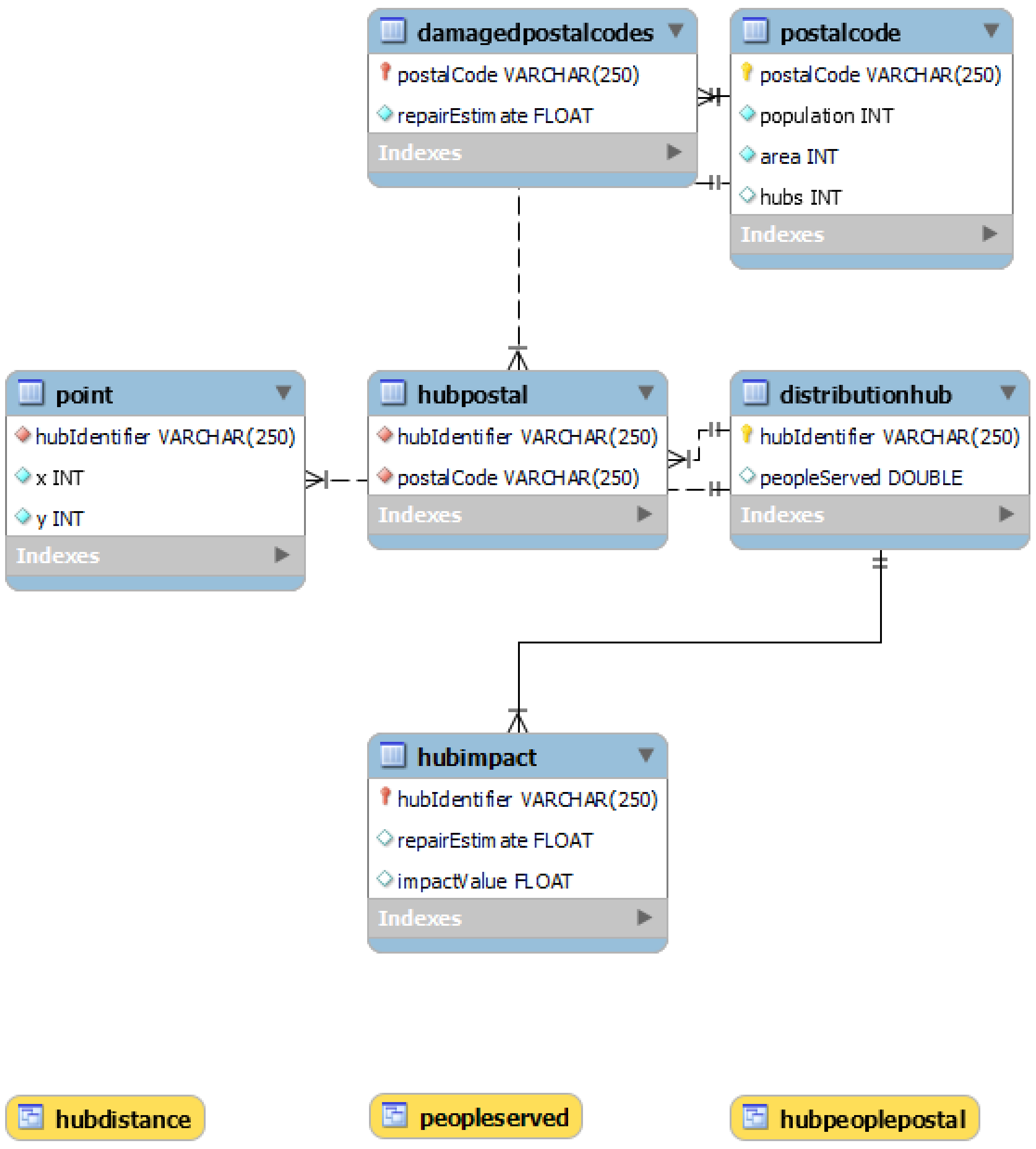

Figure 1: ER diagram

The database has 6 tables and 3 views. Each table stores following data:

1. postalcodes:

postalCode (PK): String identifier of postalCode
population: Population of province
Area: Area serviced in square meters
Hubs: No. of hubs serviced by the postal code

2. distributionhub:
   hubIdentifier (PK): ID for each hub in the system
   peopleServed: peopleServed by the hub found by taking the sum of population in province per hub it services. Used a view called peopleServed to make operation easier.

3. hubpostal:
   hubIdentifier (FK): ID for each hub
   postalCode (FK): postalCode it services
   Implemented to introduce multi value attributes in tables using normalization.

4. hubimpact:
   hubIdentifier (FK): ID for each hub
   repairEstimate: hours of repair time needed to be functional
   impactValue: Impact of hub i.e no. of people who will regain service per hour of repair. Calculated by dividing peopleserved by each hub with repairEstimate. peopleServed calculated in table distributionhub.

5. damagedpostalcodes:
   postalCode (FK): postal codes that have damage
   repairEstimate: sum of repair time needed for the postal code. Calculated with respect to each hub identifier serviced by the postal code (calculated in hubpostal) and doing a left join on the hubs.

6. Table point:
   hubIdentifier (FK): ID for hub.
   X: x UTM coordinate of the hub.
   Y: y UTm coordinate of the hub
   Implemented to introduce normalization.

7. View hubpeoplepostal:
   postalCode: Identifier for the postalcodes
   Hubs: No. of hubs servicing the postalCode

8. View peopleServed:
   hubIdentifier: Id for each hub
   peopleServed: No. of people served by the hub. If multiple hubs are servicing one postal code then peopleServed is a fraction of population by total hubs in the province.

9. Hubdistance:
   hubIdentifier: ID for each hub
   X: x UTM coordinate of the hub.
   Y: y UTm coordinate of the hub
   Distance: distance of the hub from given startHub

impactValue: Impact of hub i.e no. of people who will regain service per hour of repair.
repairEstimate: hours of repair time needed to be functional.

Methods:

addPostalCode():

An object is created of class postal code which contains variables postalCode identifier, population and area. The object is created using a parameterized constructor. The class contains two methods:

1. addPost(): To add new postal codes
2. updatePost(): Update values associated with an existing postal code

The values are stored in the postalcode table. If an exception of a duplicate entry is thrown the values are updated.

addDistributionHub():

An object is created of a class distribution hub which contains hubIdentifier, location and set of serviced postal codes. Location is an object of the point container class which creates UTM coordinates. The insert query is broken into 3 parts where the hubIdentifier is first added in the distributionhub table. Next the points are stored in the Point table. And lastly using an iterator postalcodes are added in the hubpostal table. A separate table for multiple hubs servicing one postal code was created to add multi valued attributes. The class contains two methods:

1. addHub(): Add new hubs
2. updateHub(): Update values associated with existing hubs.

hubDamage():

The hubIdentifier and repairEstimate are added in table hubimpact. The class contain two methods:

1. addImpact(): To add new hubs which have damage.
2. updateImpact(): To update damage to existing hubs.

hubRepair():

The method records repairs done on specified hubs. It extracts the repairEstimate of the hub from hubimpact for each hub and subtracts it with the amount of repair done. It then updates the new repairEstimate with a call to hubDamage. If the amount of repair done is equivalent to the repairEstimate it indicates that the hub is in service by setting boolean inService to true and deleting the hub from hubImpact table.

peopleOutOfService():

An SQL query created a table of people served by each damaged hub from hubimpact. A resultset is built using this and the sum of people is calculated.

mostDamagedPostalCodes():

Using an insert select statement postalCodes and their repairEstimates are inserted into table damagedpostalcodes. Repair estimate of postal code is calculated by taking sum of repairEstimate of each hub servicing the postal code from table hubimpact which only contains damaged hubs. Data from damagedpostalcodes is extracted in a resultset after arranging in descending order. Then the limit top rows are stored in the DamagedPostalCode class objects. If a tie break situation occurs then the elements with the same values are returned.

fixOrder():

This impactValue of all the hubs is calculated and stored in table hubimpact. Impact value is the number of people who will regain service per hour of repair. It is calculated by dividing people served by the hub with the repair estimate of that hub. People per hub is stored in the table distributionhub. Data from hubimpact is extracted in a resultset after arranging it in descending order. Then the limit top rows are stored in HubImpact class objects. In a tie break situation hubs with similar impactvalue are also stored.

rateOfServiceRestoration():

This method returns the rate at which percent of the population will be returned to service. A variable 'totalPopulation' is calculated that contains people who are currently out of service. This is calculated by taking sum of people at damaged hubs stored in distributionhub table. Next 'totalHours' variable that contains the total number of repairs needed across the system is calculated. 'timePerson' i.e time needed per person is stored. Increment value is multiplied by 100 to be used in calculations with respect to percentage. This is how it will be considered throughout the code. Now 'peoplePercent' which is the number of people for 'increment' percent is calculated. 'peoplePercent' is multiplied by time needed per person. This entry is rounded off to give value in integer. Lastly, increment is incremented to commute to the next value.

repairPlan():

1. Values of startHub are extracted i.e x coordinate, y coordinate, repairEstimate and impactValue.
2. An object of the HubImpact class is created for startHub which is stored in our final path.
3. Variable totalTime and distanceCover are used to increment values and keep track.
4. A view 'hubdistance is created which contains values of hubs and their distance from start value. It contains only the hubs which are within range of maxDistance. This view is ordered in descending order of impactValue.
5. The first row in this view is the endHub i.e the hub with the greatest impact.
6. An object of the HubImpact class is created for endHub which is stored in our final path.
7. Variables totalTime and distanceCover are incremented.
8. A resultset query will be calculated which will contain all the hubs in range of startHub to endHub.
9. Now the xMono, yMono, distance and impact paths will be calculated.
10. For xMono this query will be sorted in ascending 'x' order i.e An x monotone path.
11. For yMono this query will be sorted in ascending 'y' order i.e A y monotone path.
12. For distance this query will be sorted in ascending distance order i.e The closed variables will be considered first.
13. For impact this query will be sorted in descending impactValue order i.e The highest impacted variable first.
14. For each hub in the result set values conditions will be checked.
15. Two flags for xInc and YInc will be maintained which will check whether the coordinates of nextHub are incrementing or decrementing.
16. Two variables 'midpointX' and 'midpointY' will be created to make sure the diagonal is only crossed once, midpointX = startX and midpointY = startY.

Consider this diagram:

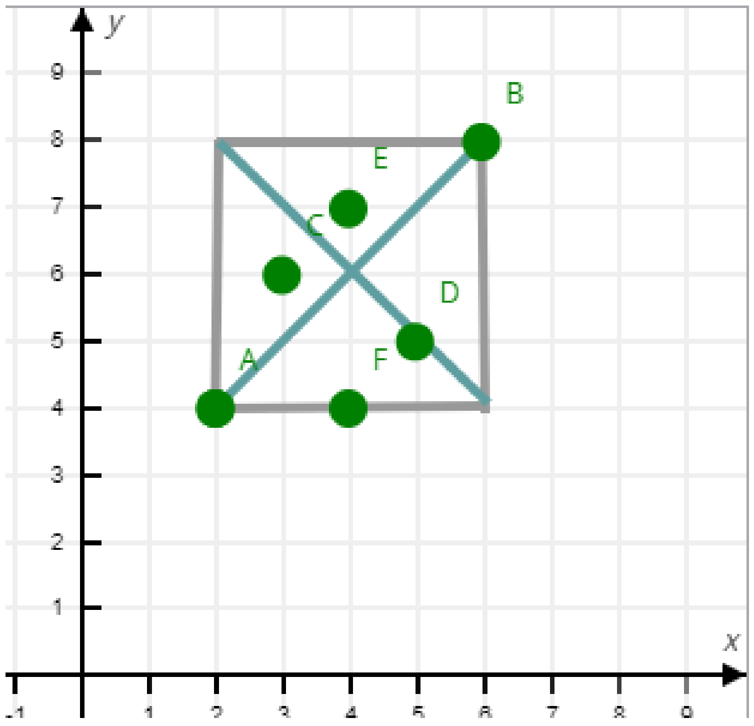

Figure 2: Plotting of hubs

Here A(2,4), B(6,8), C(3,6), D(5,5), E(4,7), F(4,4)
Consider we want to go from A to D, here the diagonal is crossed once and the calculations are:
currentX = 5, startX = 2, endX = 6 and midpointX = 2
(5-2) > (6-2)/2 = true
So the diagonal will be crossed. Verifying for y coordinate:
currentY = 5, startY = 4, endY = 8 and midpointY = 4
(5-4) > (8-4)4 = true
This way will be computing whether a point passes a diagonal or not. A diagonal variable will be maintained which will be incremented once the diagonal is crossed. This variable should only be incremented once.

17. In one condition we will check whether after adding this variable time will be less than maxTime, distance will be less than maxDistance and diagonal is incremented only once, and for an x monotone path the flag xInc should always be true.
18. If the hub satisfies all the conditions it will be added to the path.
19. The final path is returned which is sorted in increasing order of distance and stored as value in a Map. The key of the Map will be the impact of the entire path.
20. Now the yMono, distance, and impact paths will be calculated and stored in the map.
21. The path with the highest totalImpact will be returned as the repairPlan.

underservedPostalByPopulation():

The count of postal codes is divided by population to find the postal codes which have the least number of hubs per person. This query is arranged in ascending order of postal by population to get the postal code that is the least served first.

underservedPostalByArea():

The count of postal codes is divided by area to find the postal codes which have the least number of hubs per square meter. The resultset is arranged in ascending order to get the least served postal code first.

Assumptions:

1. While computing repairPlan the best optimal solution is one of 4 paths created.