CSCI 3901 Project External Documentation Final Version

Shubham Mishra (B00917146)

Overview

This program implements PowerService class that is used to insert/read/update the information related to distribution hubs and postal codes in the database and report useful information generated by processing the data stored in the database.

Files and external data

The program consists of 8 files:

- PowerService.java
  - It is the main class that contains data entry, reporting and planning methods related to distribution hubs and postal codes.
- Point.java
  - It contains setters/getters of x and y coordinate of a point.
- HubImapct.java
  - It contains setters/getters of hubID and the impact(number of people who regain service per hour of repair) of a hub
- DamagedPostalCodes.java
  - It contains setters/getters of postalCode and the total estimated repair time of all hubs that service it.
- Constants.java
  - It contains all the constants that are needed by different classes.
- Db.java
  - It contains queries needed to insert/read/update the data from database.
- HubInfo.java
  - It contains information related to a distribution hub.
- RepairPlan.java
  - It helps in finding the path(intermediate hubs to repair) from start hub to end hub.

Data Structures and their relations to each other

- ArrayList has been used in some cases to store data fetched from database through query.
- repairPlan method uses graph concept to find the repair path.

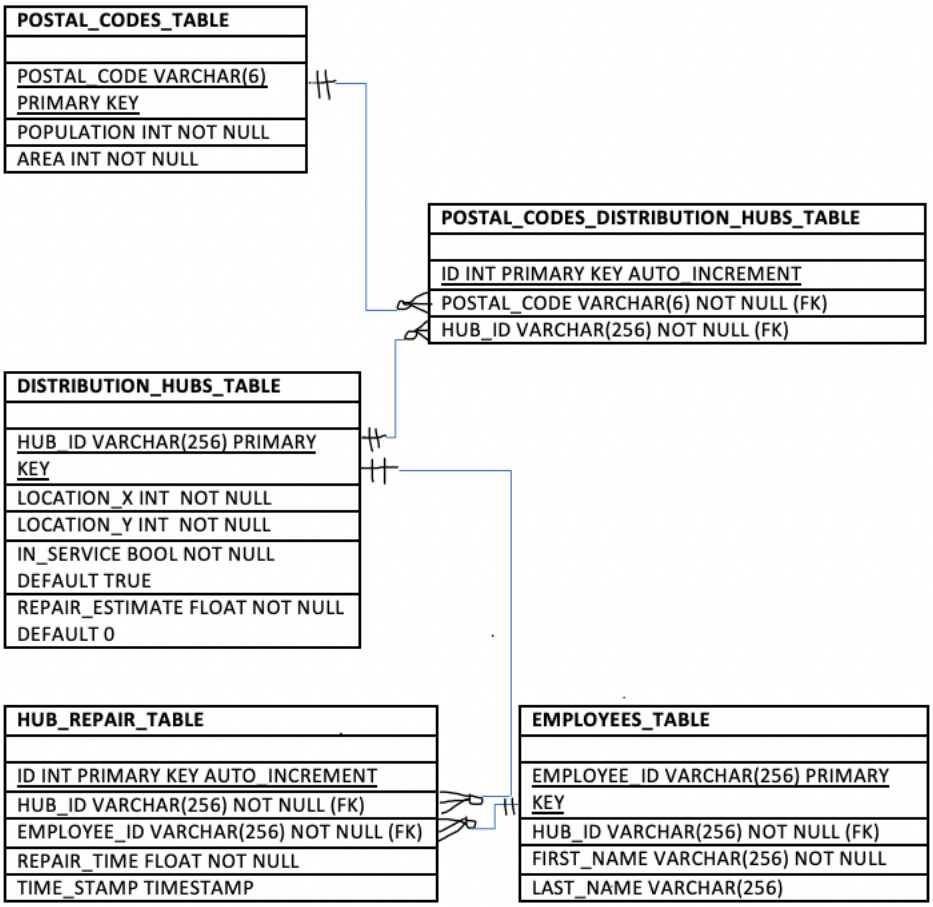
Assumptions

- The input postal code identifier will not overlap with existing postal codes identifiers.
- The postal codes are stable. Once created, we won't delete a postal code.
- Employee would report the repair via their cell phone before leaving power distribution hub.

- **EMPLOYEES_TABLE already exists and has some rows already inserted in it.**

Key algorithms and design elements

Database design



- **Add postal code**
  - Postal code along with area and population are inserted into db using following query
    - ```
      INSERT INTO POSTAL_CODES_TABLE(POSTAL_CODE, POPULATION, AREA) VALUES
          ("p1", 20, 100)
          ON DUPLICATE KEY UPDATE
          POPULATION=20,
          AREA=100;
      ```
  -
- **Add distribution hub**
  - Distribution hub along with its location are inserted into the db using following query
    - ```
      INSERT INTO DISTRIBUTION_HUBS_TABLE(HUB_ID, LOCATION_X, LOCATION_Y) VALUES
          ("h1", 30, 30)
          ON DUPLICATE KEY UPDATE
          LOCATION_X=30,
          LOCATION_y=30,
          IN_SERVICE=TRUE,
          REPAIR_ESTIMATE=0;
      ```
  -
  - To link hub with the postal codes it serves, first it is checked that the serviced areas (postal codes) supplied in input should be present in the db. If true following query is used:
    - ```
      INSERT INTO POSTAL_CODES_DISTRIBUTION_HUBS_TABLE(POSTAL_CODE, HUB_ID) VALUES
          ("p1", "h1"), ("p1", "h2");
      ```
  -
- **Hub damage**
  - To report damage to hub, first it is checked that the hub id supplied in input should be present in the db. If true following query is used:
    - ```
      UPDATE DISTRIBUTION_HUBS_TABLE
          SET IN_SERVICE=FALSE, REPAIR_ESTIMATE=10
          WHERE HUB_ID="h1";
      ```
  -
- **Hub repair**
  - To report repair to a hub, first it is checked that the hub id supplied in input should be present in the db. If true following query is used to log repair to the hub:
    - ```
      INSERT INTO HUB_REPAIR_TABLE(HUB_ID, EMPLOYEE_ID, REPAIR_TIME, IN_SERVICE)
      VALUES ("h1", "5698", 10, TRUE);
      ```
  -
  - If the hub got in service after repair, distribution hub info is also updated for that hub using following query
    - ```
      UPDATE DISTRIBUTION_HUBS_TABLE
          SET IN_SERVICE=TRUE, REPAIR_ESTIMATE=0
          WHERE HUB_ID="h1";
      ```
  -
- **People out of service**

- Following query is used to find number of people out of service

```
WITH T1 AS
    (SELECT *
        FROM POSTAL_CODES_TABLE JOIN POSTAL_CODES_DISTRIBUTION_HUBS_TABLE USING(POSTAL_CODE)
        JOIN DISTRIBUTION_HUBS_TABLE USING (HUB_ID)),
T2 AS
    (SELECT POSTAL_CODE, COUNT(*) as TOTAL_HUBS, POPULATION
        FROM T1
        GROUP BY POSTAL_CODE),
T3 AS
    (SELECT POSTAL_CODE, COUNT(*) as HUBS_OUT_OF_SERVICE
        FROM T1 WHERE IN_SERVICE=FALSE
        GROUP BY POSTAL_CODE)
SELECT COALESCE(SUM((POPULATION*HUBS_OUT_OF_SERVICE)/TOTAL_HUBS), 0) AS PEOPLE_OUT_OF_SERVICE
   FROM T2 JOIN T3 USING(POSTAL_CODE);
```

- **Most damaged postal codes**
  - Following query is used to most damaged postal codes where limit is as received from input

```
SELECT POSTAL_CODE, SUM(REPAIR_ESTIMATE) AS TOTAL_REPAIRS
    FROM POSTAL_CODES_TABLE JOIN POSTAL_CODES_DISTRIBUTION_HUBS_TABLE USING(POSTAL_CODE)
    JOIN DISTRIBUTION_HUBS_TABLE USING (HUB_ID)
    GROUP BY POSTAL_CODE
    HAVING TOTAL_REPAIRS>0
    ORDER BY TOTAL_REPAIRS DESC
    LIMIT 5;
```

- **Fix order**
  - Following query is used to find the fix order

```
WITH T1 AS
    (SELECT *
        FROM POSTAL_CODES_TABLE JOIN POSTAL_CODES_DISTRIBUTION_HUBS_TABLE USING(POSTAL_CODE)
        JOIN DISTRIBUTION_HUBS_TABLE USING (HUB_ID)),
T2 AS
    (SELECT POSTAL_CODE, POPULATION/COUNT(*) as POPULATION_PER_HUB
        FROM T1
        GROUP BY POSTAL_CODE),
T3 AS
    (SELECT HUB_ID, POSTAL_CODE, REPAIR_ESTIMATE FROM T1  WHERE IN_SERVICE=FALSE)
SELECT HUB_ID, SUM(POPULATION_PER_HUB)/REPAIR_ESTIMATE AS HUB_IMPACT
    FROM T2 JOIN T3 USING(POSTAL_CODE)
    GROUP BY HUB_ID
    ORDER BY HUB_IMPACT DESC;
```

  - The first limit hubs from the result of above query are then returned.
- **Rate of service restoration**
  - To compute it we need following info:
    - fixOrder: fetched using fix order query
    - repair estimates for all hubs: SELECT HUB_ID, REPAIR_ESTIMATE FROM DISTRIBUTION_HUBS_TABLE;
    - total population: SELECT SUM(POPULATION) AS TOTAL_POPULATION FROM POSTAL_CODES_TABLE;

        - People out of service to calculate people already having power: fetched using people out of service query
        - To compute rate of service restoration list:
            - We traverse hubs in fixorder and calculate percentage population restored by each one of them one by one
            - We maintain percentage value of people already restored with power and add previous calculated value to it.
            - We also keep track of sum of increments we have reached so far.
            - We compare people in power with sum of increments we have reached so far and update rate of service restoration list accordingly.
- **Repair Plan**
    - Find out hub with largest impact (using fix order function) within maxDistance from start hub. Mark it as end hub
    - Find out other hubs that lie inside the rectangle formed by start and end hub as diagonal.
    - Setup the constraints given in the problem statement: like finding the x and y direction to move in order to reach end hub, maxTime at intermediate hubs
    - Starting from start hub use depth first search algo to traverse all other hubs
    - Keep checking the problem constraints are met for every hub we visit.
    - Use the equation of line (ax+by=c) to determine the side of diagonal the hub lies in.
    - Return the path with max sum of impacts of all hubs.
- **Underserved postal codes by population**
    - Following query is used to find it:

```
WITH T1 AS
    (SELECT *
        FROM POSTAL_CODES_TABLE JOIN POSTAL_CODES_DISTRIBUTION_HUBS_TABLE USING(POSTAL_CODE)
        JOIN DISTRIBUTION_HUBS_TABLE USING (HUB_ID))
SELECT POSTAL_CODE, COUNT(*)/POPULATION as HUBS_PER_PERSON
        FROM T1
        GROUP BY POSTAL_CODE
        ORDER BY HUBS_PER_PERSON
        LIMIT 5;
```

    -
- **Underserved postal codes by area**
    - Following query is used to find it:

```
WITH T1 AS
    (SELECT *
        FROM POSTAL_CODES_TABLE JOIN POSTAL_CODES_DISTRIBUTION_HUBS_TABLE USING(POSTAL_CODE)
        JOIN DISTRIBUTION_HUBS_TABLE USING (HUB_ID))
SELECT POSTAL_CODE, COUNT(*)/AREA as HUBS_PER_AREA
        FROM T1
        GROUP BY POSTAL_CODE
        ORDER BY HUBS_PER_AREA
        LIMIT 5;
```

    -

Limitations

- No methods to delete the information added through data entry methods
- Dependant on correct information entered by the field employee. If employee enters wrong information then the computations will go wrong.