Πανεπιστήμιο Ιωαννίνων

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ και Πληροφορικής

Προπτυχιακό Μάθημα: «Βάσεις Δεδομένων»

Δεύτερη Σειρά Ασκήσεων

<u>Όνομα Φοιτητή – Α.Μ.:</u>

Γεώργιος Κρομμύδας – 3260

ΙΩΑΝΝΙΝΑ,

2020

Άσκηση-1η: (Σχεσιακή Άλγεβρα):

Χρησιμοποιώντας τους έτοιμους πίνακες **Trip**, **Station** και **Weather**, παρακάτω φαίνονται τα αποτελέσματα σε σχεσιακή άλγεβρα των παρακάτω ερωτήσεων.

(α) Για να βρούμε τις κυκλικές διαδρομές, θα πρέπει πρώτα να επιλέξουμε από τον πίνακα **Trip** τις διαδρομές όπου το **start_station_name** = **end_station_name** και **start_station_id** = **end_station_id**. Στη συνέχεια, κάνουμε συνένωση με τον πίνακα **Station** με την προϋπόθεση ότι το όνομα και του σταθμού και το id του είναι ίδια. Στο τέλος προβάλουμε τα δύο γνωρίσματα id και station_name και να εξάγουμε την πληροφορία που χρειαζόμαστε.

$$\pi_{id,station\_name}\left(Station \bowtie_A \left(\sigma_B(Trip)\right)\right)$$

όπου το

$$A = (station\_name = start\_station\_name\ AND\ station\_id = start\_station\_id)$$

και

$$B = (start\_station\_name = end\_station\_name\ AND\ start\_station\_id = end\_station\_id)$$

(β) Για να βρούμε τις πόλεις που εμφανίζουν είτε ομίχλη είτε βροχή αλλά όχι και τα δύο, θα πρέπει πρώτα να επιλέξουμε τον πίνακα **Weather** με τα εκάστοτε **events** και να ενώσουμε τις προβολές του γνωρίσματος **city**. Ωστόσο, θα πρέπει να αφαιρέσουμε τα γεγονότα που εμφανίζουν και τα δύο.

$$(\pi_{city}(Station \bowtie \sigma_{events = 'fog'}(Weather)) \cup \pi_{city}(Station \bowtie \sigma_{events = 'rain}(Weather))$$

$$-\pi_{city}(Station \bowtie \sigma_{events = 'fog-rain}(Weather))$$

(γ) Για να βρούμε την μέγιστη και ελάχιστη χρονικά διαδρομή θα πρέπει να δημιουργήσουμε δύο νέους πίνακες με τα ίδια γνωρίσματα:

$$\text{T1(id1, duration1, start\_time1, end\_time1, ...)} \leftarrow Trip(id, duration, \dots)$$

$$\text{T2(id2, duration2, start\_time2, end\_time2, ...)} \leftarrow Trip(id, duration, \dots)$$

όπου αντίστοιχα το duration1 και το duration2 είναι ο ελάχιστος και μέγιστος χρόνος διαδρομής. Τώρα, παρακάτω φαίνεται σε σχεσιακή άλγεβρα το παραπάνω ερώτημα:

$$\pi_{duration\ ,duration}\ (T1\ \bowtie_{start\_time1\ =\ start\_time2\ AND\ end\_time1\ <\ end\_time2}\ T2)$$

(δ) Για αυτό το ερώτημα πρέπει αρχικά να επιλέξουμε τις πλειάδες του πίνακα **Weather**, όπου η **min_temp** είναι λιγότερη από 50F και στη συνέχεια να βρούμε τις πόλεις τι οποίες παρατηρείται αυτό το φαινόμενο. Στη συνέχεια, συνενώνουμε τον πίνακα που παράγεται μαζί με τον πίνακα **Trip** με την συνθήκη αφετηρίας ή τερματισμού για το εκάστοτε ποδήλατο. Τέλος, προβάλουμε τα γνωρίσματα **bike_id** και **city** για να βρούμε ποια ποδήλατα είτε ξεκίνησαν είτε έφτασαν στην συγκεκριμένη πόλη.

$$\pi_{bike\_id,city}(Trip\ \bowtie_A\ (Station\ \bowtie_{zip\_code\ =\ zip\_code}\ (\sigma_{min\_temp\ <\ 50}(Weather))))$$

όπου το

$$A\ =\ ((start\_station\_name\ =\ station\_name\ AND\ start\_station\_id\ =\ station\_id\ )$$
$$OR\ (end\_station\_name\ =\ station\_name\ AND\ end\_station\_id\ =\ station\_id))$$

(ε) Αρχικά, επιλέγουμε τις πλειάδες του πίνακα **Trip** που περιέχουν το ποδήλατο με **id** 318, δηλαδή τις διαδρομές που έχει κάνει το συγκεκριμένο ποδήλατο. Στη συνέχεια, συνενώνουμε με τον πίνακα **Station** με την συνθήκη πως είτε θα ξεκινάει από τον συγκεκριμένο σταθμό είτε θα φτάνει. Τέλος, προβάλουμε τις πόλεις που έχει περάσει το συγκεκριμένο ποδήλατο.

$$\pi_{city}(Station\ \bowtie_A\ (\sigma_{bike\_id\ =\ 318}(Trip)))$$

όπου το

$$A\ =\ ((start\_station\_name\ =\ station\_name\ AND\ start\_station\_id\ =\ station\_id\ )$$
$$OR\ (end\_station\_name\ =\ station\_name\ AND\ end\_station\_id\ =\ station\_id))$$

Άσκηση-2η: (Σχεσιακός Λογισμός):

(α) Χρειαζόμαστε τις πόλεις που εμφανίζουν είτε βροχή είτε ομίχλη, αλλά όχι και τα δύο. Σε σχεσιακό λογισμό είναι:

$$\{t.city \mid Station(t)\ and\ [\ ((\exists\ d1)\ Weather(d1)\ and\ d1.events\ =\ 'fog')$$

$$or\big((\exists d2)Weather(d2)andd2.event = 'rain'\big)$$

$$and\ d1.zip\_code\ =\ d2.zip\_code)\ and\ (d1.zip\_code\ =\ t.zip\_code)]$$

$$and\ [(not(\exists\ d3)\ Weather(d3)\ and\ d3.events\ =\ 'fog - rain')$$

$$and\ d3.zip\_code\ =\ t.zip\_code]\}$$

(β) Χρειαζόμαστε να βρούμε την ελάχιστη και μέγιστη χρονικά διαδρομή που μπορεί να υπάρξει. Πρώτα ας δούμε πως γίνεται με χρήση υπαρξιακού ποσοδείκτη:

$$\{t1.duration, t2.duration \mid Trip(t1)\ and\ Trip(t2)$$

$$and\ t1.start\_time = t2.start\_time$$

$$and\ [((\exists\ d1)\ Trip(d1)\ and\ d1.end\_time\ =\ t1.end\_time)$$

$$and\ ((\exists\ d2)\ Trip(d2)\ and\ d2.end\_time\ =\ t2.end\_time)$$

$$and\ (d1.end\_time\ <\ d2.end\_time)]\}$$

Τώρα ας δούμε πως μετατρέπεται η σχέση με χρήση του καθολικού ποσοδείκτη.

(∀ t) P(t) ≡ not (∃ t) (not P(t))

(∃ t) (P(t)) ≡ not (∀ t) (not P(t))

Χρησιμοποιώντας τις παραπάνω σχέσεις, προκύπτει ότι:

$$\{t1.duration, t2.duration \mid Trip(t1)\ and\ Trip(t2)$$

$$and\ t1.start\_time = t2.start\_time$$

$$and\ [(not(\forall\ d1)\ Trip(d1)\ or\ d1.end\_time\ \neq\ t1.end\_time)$$

$$or\ (not(\forall\ d2)\ Trip(d2)\ or\ d2.end\_time\ \neq\ t2.end\_time)$$

$$or\ (d1.end\_time\ >\ d2.end\_time)]\}$$

Άσκηση-3η: (SQL):

(α) Τα παρακάτω ερωτήματα είναι σε SQL και βγάζουν τα αντίστοιχα αποτελέσματα:

(i) Αυτή η ερώτηση επιστρέφει την μέγιστη και ελάχιστη χρονικά διαδρομή.

```
SELECT  MAX(duration),  MIN(duration)
FROM    Trip;
```

(ii) Η συγκεκριμένη ερώτηση επιστρέφει τις πόλεις που έχει επισκευθεί το ποδήλατο με **bike_id** = 318.

```
SELECT  S.city
FROM    Station AS S, Trip AS T
WHERE   T.bike_id = 318 AND ((T.start_station_name = S.station_name AND T.start_station_id = S.station_id)
                             OR (T.end_station_name = S.station_name AND T.end_station_id = S.station_id));
```

(iii) Αυτή η ερώτηση επιστρέφει το ποσοστό των κυκλικών διαδρομών.

```
SELECT DISTINCT COUNT(T.id)*100.0/(SELECT COUNT(*) FROM TRIP)
FROM    Trip AS T
WHERE   T.start_station_name = T.end_station_name AND T.start_station_id = T.end_station_id
GROYP BT T.id;
```

(iv) Η συγκεκριμένη ερώτηση επιστρέφει τα ποδήλατα και το πλήθος τους που έχουν είτε ως αφετηρία είτε ως προορισμό πόλεις με καιρικά φαινόμενα.

```
SELECT  T.bike_id, COUNT(T.bike_id)
FROM    Trip AS T, Station AS S
WHERE ((T.start_station_name = S.station_name AND T.start_station_id = S.station_id)
        OR (T.end_station_name = S.station_name AND T.end_station_id = S.station_id))
        AND (S.city IN (SELECT  W.date
                        FROM   Weather AS W
                        WHERE  W.zip_code = S.zip_code AND W.events IS NOT NULL))
GROYP BY T.bike_id;
```

(v) Αυτή η ερώτηση επιστρέφει τις 5 πιο δημοφιλείς πόλεις που αποτελούν είτε ως αφετηρία είτε ως προορισμό.

```
WITH temp(tcity, tcitycount)
AS(   SELECT city, COUNT(city)
      FROM   Station AS S NATURAL JOIN Trip AS T
      WHERE  ((S.station_id = T.start_station_id AND S.station_name = T.start_station_name)
              OR (S.station_id = T.end_station_id AND S.station_name = T.end_station_name))
      GROYP BY city
      HAVING COUNT(city) = 5)

SELECT T.tcity, T.tcitycount
FROM   temp AS T
WHERE T.tcitycount IN (SELECT MAX(A.tcitycount)
                       FROM  temp AS A)
ORDER BY T.tcity ASC;
```

(β) Αυτή η ερώτηση εισάγει δεδομένα με μορφή SFW στον πίνακα **Trip**. Εισάγει τις διαδρομές τις οποίες έκανε το ποδήλατο **bike_id** = 318, που είχε είτε ως αφετηρία είτε ως προορισμό την πόλη San Jose.

```
INSERT INTO Trip
SELECT S.station_name, S.station_id
FROM   Trip AS T, Staion AS S
WHERE  (((S.station_id = T.start_station_id AND S.station_name = T.start_station_name)
        OR (S.station_id = T.end_station_id AND S.station_name = T.end_station_name))
       AND T.bike_id = 318 AND S.city = 'San Jose');
```

(γ) Σε αυτό το ερώτημα υλοποιήθηκε η παραπάνω βάση με χρήση της python.

```python
#!/usr/bin/python

import mysql.connector
import sys

'''
    This code creates the database with the tables Trip, Station and Weather.
    The next step is to load the data into the table
    and then finally  to execute some question regarding the information of the tables.
'''


USER = "root"
database_name = "BikeRide"
PSW = '*****'

def show_databases(cursor):
    cursor.execute("SHOW DATABASES")
    databases = cursor.fetchall()
    for database in databases:
        print(database)

def create_database(cursor):
    cursor.execute("CREATE DATABASE %s" %database_name)

def drop_database(cursor):
    cursor.execute("DROP DATABASE %s" %database_name)


def print_results(query, cursor):
    print(query)
    results = cursor.fetchall()
    for item in results:
        print(item)
    print("Number of results: " + str(len(results)))
```

```python
if __name__ == '__main__':


    # Creation of the database
    my_db = mysql.connector.connect(host = 'localhost', user = USER, passwd = PSW)
    cursor = my_db.cursor()
    show_databases(cursor)

    drop_database(cursor)
    create_database(cursor)
    my_db.close()

    # Connection in the database and creation of the tables
    my_db = mysql.connector.connect(host='localhost', user = USER, passwd = PSW, database = database_name)
    cursor = my_db.cursor()

    cursor.execute("CREATE TABLE Weather(date_w DATE NOT NULL,\
                                max_temp REAL,\
                                mean_temp REAL,\
                                min_temp REAL,\
                                max_visibility_miles REAL,\
                                mean_visibility_miles REAL,\
                                min_visibility_miles REAL,\
                                max_wind_speed_mph REAL,\
                                mean_wind_speed_mph REAL,\
                                max_gust_speed_mph REAL,\
                                cloud_cover REAL,\
                                events TEXT,\
                                wind_dir_degrees REAL,\
                                zip_code VARCHAR(256) NOT NULL,\
                                PRIMARY KEY(date_w,zip_code))ENGINE=innodb;"
    )
```

```
cursor.execute("CREATE TABLE Station(station_id SMALLINT NOT NULL,\
                                     station_name TEXT,\
                                     latitude REAL,\
                                     longtitude REAL,\
                                     dock_count SMALLINT,\
                                     city TEXT,\
                                     installation_date DATE,\
                                     zip_code VARCHAR(256),\
                                     PRIMARY KEY(station_id),\
                                     INDEX(zip_code),\
                                     FOREIGN KEY(zip_code) REFERENCES Weather(zip_code) ON DELETE CASCADE)ENGINE=innodb;"
)
cursor.execute("CREATE TABLE Trip(id INTEGER NOT NULL,\
                                  duration INTEGER,\
                                  start_time TIMESTAMP,\
                                  start_station_name TEXT,\
                                  start_station_id SMALLINT,\
                                  end_time TIMESTAMP,\
                                  end_station_name TEXT,\
                                  end_station_id SMALLINT,\
                                  bike_id SMALLINT,\
                                  PRIMARY KEY(id),\
                                  INDEX(start_station_id),\
                                  INDEX(end_station_id),\
                                  FOREIGN KEY(start_station_id) REFERENCES Station(station_id) ON DELETE CASCADE,\
                                  FOREIGN KEY(end_station_id) REFERENCES Station(station_id) ON DELETE CASCADE)ENGINE=innodb;"
)


# Load csv files into the tables
cursor.execute("SET GLOBAL local_infile=1")
query='LOAD DATA LOCAL INFILE "weather.csv" INTO TABLE Weather FIELDS TERMINATED BY ";" LINES TERMINATED BY "\n"'
cursor.execute(query)
query='LOAD DATA LOCAL INFILE "station.csv" INTO TABLE Station FIELDS TERMINATED BY ";" LINES TERMINATED BY "\n"'
cursor.execute(query)
query='LOAD DATA LOCAL INFILE "trip.csv" INTO TABLE Trip FIELDS TERMINATED BY ";" LINES TERMINATED BY "\n"'
cursor.execute(query)
```

```
cursor.execute('COMMIT')

#Check if the data successfully loaded into the tables
print("-------  PRINT DATA -------")
query  = "SELECT * FROM Trip"
cursor.execute(query)
print_results(query, cursor)

query  = "SELECT * FROM Station"
cursor.execute(query)
print_results(query, cursor)

query  = "SELECT * FROM Weather"
cursor.execute(query)
print_results(query, cursor)
```

(i)Σε αυτό το ερώτημα εισάγει τις παραμέτρους ο χρήστης, έτσι ώστε να εκτελεστεί η παρακάτω ερώτηση:

```
if(len(sys.argv) < 2):
    print("Please insert the correct parameters!!")
    sys.exit(1)

# First Question
Tbl = sys.argv[1]
Fld = sys.argv[2]
val = sys.argv[3]
query = "SELECT * FROM (%s) WHERE (%s).(%s) = (%s);"
vals = (Tbl, Tbl, Fld, val)
cursor.execute(query, vals)
print_results(query, cursor)
```

(ii) Σε αυτό το ερώτημα εκτελείτε η παρακάτω ερώτηση, η οποία εμφανίζει τις πόλεις μς τους εκάστοτε σταθμούς τους.

```
# Second Question
query = "SELECT Station.city, Station.station_name FROM Station;"
cursor.execute(query)
print_results(query, cursor)

my_db.close()
```