CSE 460 Project2: SQL Query

Due 23:59 11/02/2017 EST

Date 10/17/2017

1 Submission

Failure to comply the submission specifications will incur penalties for EACH violations

1.1 What to submit

A zip file has to be submitted through the 'submit_cse460' submission script by 11/02/2017 11:59PM EST. **ONLY** zip extension will be accepted.

1.2 Zip file naming convention

Write your *ubit* (**NO SPACE!**) for the filename, e.g. *jsmith.zip*, where *jsmith* is the ubit. The project is an **INDIVIDUAL** project, copied solutions will be considered violations of academic integrity.

1.3 Sub-structure of zip file

- On unzipping the zip file, there should be a folder named with your ubit *ubit.*
- Under the folder *ubit*, there should be two .sql files, name the first one as prob1.sql, this file should contain your answers for Problem 1, and name the second one as prob2.sql, this file should contain your answers for Problem 2.

2 Problem 1(50 points)

Consider below relational DB schema: FLIGHT($\underline{\text{Origin, Destination}}$, Cost) for a flight information management system, keys are underlined, see flight.sql for more details of the table. Answer following questions and write your answers in your prob1.sql file, use comments to separate your answers for each question.

- **1.1 (10 points)** Run the CREATE TABLE statement in flight.sql use the DBMS on your own laptop, and insert following data into the table, write your INSERT INTO statement(s) in your .sql file.

Origin	Destination	Cost
ATL	ORD	1000
BUF	ORD	1500
BUF	JFK	300
JFK	ORD	200
JFK	LAX	600
BUF	LAX	2000
LAX	LAS	500
JFK	CLT	150
JFK	MIA	300
BUF	BOS	150
DTW	BUF	200
LAX	SEA	100
DTW	IAH	800
IAH	EWR	870

  sol: either a set of INSERT INTO statements or a single INSERT INTO statement.

- **1.2(30 points)** Given a relation instance I of FLIGHT, define *cycle* as: start from an origin A, if there's a path to fly back to A, then we say there's a cycle for the origin A, e.g. if we have tuples $< A, B, J >, < B, C, K >, < C, A, L >, < D, A, M >$, then there's a cycle for A, a cycle for B and also a cycle for C, but no cycle for D.

  Assuming that there's no cycle for any of the origin in the instance, write a SQL query to find all the destinations that are reachable from 'BUF' and the lowest costs to fly to those destinations.
  sol:

```
WITH RECURSIVE fl(fr,t,c) AS (
SELECT * FROM flight
UNION ALL
SELECT f.origin, fl.t, f.cost+fl.c as c
FROM flight f, fl
WHERE f.destination =fl.fr)

SELECT fl.t, MIN(c) FROM fl WHERE fl.fr='BUF' GROUP BY fl.t;
```

- **1.3(10 points)** Insert a new flight record:

Origin	Destination	Cost
SEA	LAX	150

into the flight table, what could happen if you run your query for 1.2 on the updated set of data? and why? Write your answer for this question in detail as comments in your .sql file.

sol: The query may not terminate due to there are “cycles” in the data.

3 Problem 2(50 points)

Read and understand the DB schema in enrollment.sql, Student.dept indicates the departments of the students, Enroll.grade represents the GPA, Course.dept contains just the departmental acronym, e.g., 'CSE', indicating the departments that offer the courses, write following SQL queries in your prob2.sql file, use comments to separate your answers for each question. You are encouraged to create your own testing data to verify your queries.

- **2.1(10 points)** Find the names of students who are **not** enrolled in any course offered by 'CSE' department.

```
select name
from student
where sno not in
    (select sno
    from enroll, course where enroll.cno=course.cno
    and course.dept='CSE');
```

- **2.2(15 points)** For every student, list the student name and the cumulative GPA (if GPA is not applicable to a student, still list the student name and return *null* for the cumulative GPA value).

```
select s.sno, s.name, avg(e.grade)
from student s
left join
enroll e
on s.sno=e.sno
group by s.sno;
```

- **2.3 (25 points)** For every department, list the department name and the total number of courses such that more than half of the students enrolled in such a course are from outside of the department that offers the course.

```
CREATE VIEW CS(Offer,Major,CNO,SNO) AS
SELECT c.Dept, s.Dept, c.CNO, s.SNO
FROM COURSE c, ENROLL e, STUDENT s
WHERE c.CNO=e.CNO
```

```
AND s.SNO=e.SNO;

CREATE VIEW EQC AS
SELECT cs1.CNO, COUNT(DISTINCT cs1.SNO) AS N
FROM CS cs1
WHERE cs1.OFFER = cs1.MAJOR
GROUP BY cs1.CNO;

CREATE VIEW NEQC AS
SELECT cs1.CNO, COUNT(DISTINCT cs1.SNO) AS N
FROM CS cs1
WHERE cs1.OFFER<> cs1.MAJOR
GROUP BY cs1.CNO;

CREATE VIEW C3 AS
SELECT EQC.CNO
FROM EQC, NEQC
WHERE EQC.CNO=NEQC.CNO
AND EQC.N < NEQC.N;

SELECT c.Dept, COUNT(DISTINCT ccc.CNO)
FROM COURSE c left join C3 ccc
ON c.cno=ccc.cno
GROUP BY c.Dept;
```