MySQL: Queries and Trigegers

Vahid Mirjalili

Problem 1. Queries on an emplyee database

- First create a new DATABASE using the root user and grant all priveliges to the cbuser:

```
mysql> CREATE DATABASE cse880;
Query OK, 1 row affected (0.04 sec)

mysql> grant all on cse880.* to 'cbuser'@'localhost' identified by 'cbpas
Query OK, 0 rows affected (0.03 sec)
```

Then login as cbuser and use the created database:

```
mysql> sHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| cookbook           |
| cse880             |
+--------------------+
3 rows in set (0.00 sec)

mysql> USE cse880;
Database changed
```

- Create the follwoing tables and insert rows:

```
mysql> CREATE TABLE S (
    -> C INT,
    -> D VARCHAR(20),
    -> PRIMARY KEY (C)
    -> );
Query OK, 0 rows affected (0.10 sec)

mysql> CREATE TABLE R (
    -> A VARCHAR(20),
    -> B INT,
    -> PRIMARY KEY (A),
    -> FOREIGN KEY (B) REFERENCES S(C)
    -> );
Query OK, 0 rows affected (0.10 sec)
```

- Insert rows as follows:

```
mysql> INSERT INTO S (C,D) VALUES (2,'d1'), (1,'d2'), (3,'d2');
Query OK, 3 rows affected (0.03 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

```
mysql> INSERT INTO R (A,B) VALUES ('a1',1), ('a2', 2);
Query OK, 2 rows affected (0.04 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

- Query results:
- Query 1:

```
mysql> SELECT * FROM R,S;
+----+------+---+------+
| A  | B    | C | D    |
+----+------+---+------+
| a1 |    1 | 1 | d2   |
| a2 |    2 | 1 | d2   |
| a1 |    1 | 2 | d1   |
| a2 |    2 | 2 | d1   |
| a1 |    1 | 3 | d2   |
| a2 |    2 | 3 | d2   |
+----+------+---+------+
6 rows in set (0.00 sec)
```

- Query 2:

```
mysql> SELECT * FROM R,S WHERE B=C;
+----+------+---+------+
| A  | B    | C | D    |
+----+------+---+------+
| a1 |    1 | 1 | d2   |
| a2 |    2 | 2 | d1   |
+----+------+---+------+
2 rows in set (0.02 sec)
```

- Query 3:

```
mysql> SELECT B FROM R  WHERE B NOT IN (SELECT C FROM R,S WHERE B!=C);
Empty set (0.00 sec)
```

Problem 2: complex queries

- Create the entity and relation tables:

employee's table:

```
Database changed
mysql> CREATE TABLE employee(
    -> empSSno INT,
    -> mgrSSno INT,
    -> sal REAL,
    -> PRIMARY KEY (empSSno));
```

```
Query OK, 0 rows affected (0.33 sec)
```

manager's table:

```
mysql> CREATE TABLE manager (
    -> mgrSSno INT,
    -> dno INT,
    -> PRIMARY KEY (mgrSSno));
Query OK, 0 rows affected (0.21 sec)
```

employee_department table:

```
mysql> CREATE TABLE empdept (
    -> empSSno INT,
    -> dno INT,
    -> PRIMARY KEY (empSSno));
Query OK, 0 rows affected (0.28 sec)
```

Then, we create some employees with random salary, store them in a file, which then can be inserted into the database:

```
echo 444505555 | awk '
  BEGIN{srand(); print "employee\nempSSno mgrSSno sal\nINT INT REAL\nPRIM
  {
    for (i=0; i<=10; i++) {
        print $1+i, $1 + (int(i/4)+1), rand()*70000
    }
  }' > data/employees.txt
```

Insert the employee information into the table:

```
python src/makeTable.py ~/csedb.pass data/employees.txt
```

The above python script needs the password information for the DB-user as its first argument, and the file containing raw employee information as its 2nd argument.

Now, we have the table ready in our database:

```
mysql> select * from employee;
+-----------+-----------+---------+
| empSSno   | mgrSSno   | sal     |
+-----------+-----------+---------+
| 444505555 | 444505558 | 14190.8 |
| 444505556 | 444505558 |   14925 |
| 444505557 | 444505558 | 34848.7 |
| 444505558 | 444505558 | 12060.4 |
| 444505559 | 444505561 | 48564.6 |
| 444505560 | 444505561 | 25517.3 |
| 444505561 | 444505561 | 49099.8 |
| 444505562 | 444505561 | 59578.2 |
```

```
| 444505563 | 444505564 |   50511 |
| 444505564 | 444505564 | 48938.9 |
| 444505565 | 444505564 | 16176.4 |
+-----------+-----------+---------+
11 rows in set (0.00 sec)
```

- Query 1: Get all those empSSno who are earning more than their managers.

```
mysql> SELECT e1.empSSno
    -> FROM employee AS e1, employee AS e2
    -> WHERE e1.mgrSSno=e2.empSSno AND e1.sal > e2.sal;
+-----------+
| empSSno   |
+-----------+
| 444505555 |
| 444505556 |
| 444505557 |
| 444505562 |
| 444505563 |
+-----------+
5 rows in set (0.00 sec)
```

- Query 2: Find employee SSno for those employees who work for at least all those departments that the employee with SSno 444505555 works for.

Now, we create a random data for **employee-department** table:

```
grep '^[0-9]' data/employees.txt | awk '
    BEGIN{
        srand();
        print "empldept\nempSSno dno\nINT INT\nPRIMARY KEY (empSSno,dno)"
    }
    {
        n=int(rand()*3)+1;
        for (i=1; i<=n; i++) {
            print $1, i
        }
    }' > data/empldept.txt
```

and insert this file into the database:

```
python src/makeTable.py ~/csedb.pass data/empldept.txt
```

Now, we can view the table as below:

```
mysql> SELECT * FROM empldept LIMIT 5;
+-----------+-----+
| empSSno   | dno |
+-----------+-----+
| 444505555 |   1 |
| 444505555 |   2 |
```

```
| 444505556 |   2 |
| 444505556 |   3 |
| 444505557 |   1 |
+-----------+-----+
5 rows in set (0.00 sec)
```

First, we find all the entries that work in the same department as the specified employee:

```
mysql> SELECT empSSno,dno
       FROM empldept
       WHERE dno IN (SELECT dno FROM empldept WHERE empSSno=444505555);
+-----------+-----+
| empSSno   | dno |
+-----------+-----+
| 444505555 |   1 |
| 444505555 |   2 |
| 444505556 |   2 |
| 444505557 |   1 |
| 444505558 |   2 |
| 444505560 |   1 |
| 444505561 |   2 |
| 444505562 |   1 |
| 444505562 |   2 |
| 444505563 |   1 |
| 444505563 |   2 |
| 444505564 |   2 |
+-----------+-----+
12 rows in set (0.00 sec)
```

Then, we apply a group by empSSno the result and count the number of departments that each employee work for:

```
   SELECT empSSno,COUNT(dno)
   FROM empldept
   WHERE dno IN (SELECT dno FROM empldept WHERE empSSno=444505555)
   GROUP BY empSSno;
+-----------+------------+
| empSSno   | COUNT(dno) |
+-----------+------------+
| 444505555 |          2 |
| 444505556 |          1 |
| 444505557 |          1 |
| 444505558 |          1 |
| 444505560 |          1 |
| 444505561 |          1 |
| 444505562 |          2 |
| 444505563 |          2 |
| 444505564 |          1 |
+-----------+------------+
9 rows in set (0.00 sec)
```

Finally, we only select those that work for the same number of departments as the specified employee:

```
mysql> SELECT empSSno
       FROM empldept
       WHERE dno IN (SELECT dno FROM empldept WHERE empSSno=444505555)
       GROUP BY empSSno
       HAVING COUNT(dno)=(SELECT COUNT(dno) FROM empldept WHERE empSSno=4

+-----------+
| empSSno   |
+-----------+
| 444505555 |
| 444505562 |
| 444505563 |
+-----------+
3 rows in set (0.00 sec)
```

- Query 3:

```
mysql> SELECT dno,AVG(sal)
    -> FROM employee
    -> JOIN empldept ON employee.empSSno=empldept.empSSno
    -> WHERE sal>20000
    -> GROUP BY dno HAVING COUNT(employee.empSSno)>3;
+-----+-----------+
| dno | AVG(sal)  |
+-----+-----------+
|   1 |   42613.8 |
|   2 | 52031.975 |
+-----+-----------+
2 rows in set (0.00 sec)
```

Problem 3. Create Triggers for Derived Attributes

Players table:

```
mysql> CREATE TABLE players (
       pid INT,
       name CHAR(20),
       city CHAR(20),
       phone CHAR(10),
       salary REAL,
       avg_score1 REAL,
       PRIMARY KEY (pid));
Query OK, 0 rows affected (0.10 sec)
```

teams table:

```
mysql> CREATE TABLE teams (
       tid INT,
       tname CHAR(20),
       city CHAR(20),
       avg_score3 REAL,
       PRIMARY KEY (tid));
Query OK, 0 rows affected (0.15 sec)
```

Games table:

```
mysql> CREATE TABLE games (
    -> gid INT,
    -> gname CHAR(20),
    -> PRIMARY KEY (gid));
Query OK, 0 rows affected (0.09 sec)
```

PlaysFor relation table:

```
mysql> CREATE TABLE playsfor (
       pid INT,
       tid INT,
       avg_score2 REAL,
       PRIMARY KEY (pid,tid),
       FOREIGN KEY (pid) REFERENCES players(pid),
       FOREIGN KEY (tid) REFERENCES teams(tid));
Query OK, 0 rows affected (0.15 sec)
```

TeamGames relation table:

```
mysql> CREATE TABLE teamgames (
       gid INT,
       tid INT,
       score INT,
       PRIMARY KEY (gid, tid),
       FOREIGN KEY (gid) REFERENCES games(gid),
       FOREIGN KEY (tid) REFERENCES teams(tid));
Query OK, 0 rows affected (0.21 sec)
```

PTG relation table:

```
mysql> CREATE TABLE ptg (
       pid INT,
       tid INT,
       gid INT,
       score INT,
       PRIMARY KEY (pid,tid,gid),
       FOREIGN KEY (pid) REFERENCES players(pid),
       FOREIGN KEY (tid) REFERENCES teams(tid),
       FOREIGN KEY (gid) REFERENCES games(gid));
Query OK, 0 rows affected (0.20 sec)
```

```
mysql> delimiter //
mysql> CREATE TRIGGER calavgscores
       AFTER INSERT ON ptg
       FOR EACH ROW
       BEGIN
          UPDATE players
          SET avg_score1=(
             SELECT AVG(score)
             FROM ptg
             WHERE ptg.pid=NEW.pid)
          WHERE players.pid=NEW.pid;

          UPDATE playsfor
          SET avg_score2=(
             SELECT AVG(score)
             FROM ptg
             WHERE ptg.pid=NEW.pid AND ptg.tid=NEW.tid)
          WHERE playsfor.pid=NEW.pid AND playsfor.tid=NEW.tid;

          UPDATE teams
          SET avg_score3=(
             SELECT AVG(score) FROM ptg
             WHERE ptg.tid=NEW.tid)
          WHERE teams.tid=NEW.tid;
       END;//
Query OK, 0 rows affected (0.08 sec)
mysql> delimiter ;
```

Initial State of Tables:

```
mysql> select * from players;
+-----+-------+--------------+------------+--------+------------+
| pid | name  | city         | phone      | salary | avg_score1 |
+-----+-------+--------------+------------+--------+------------+
|   0 | name1 | east lansing | 5177776666 |  70000 |          0 |
|   2 | name2 | ann arbor    | 2177776666 |  65000 |          0 |
+-----+-------+--------------+------------+--------+------------+
2 rows in set (0.00 sec)

mysql> select * from teams;
+-----+------------+--------------+------------+
| tid | tname      | city         | avg_score3 |
+-----+------------+--------------+------------+
| 111 | spartan    | east lansing |          0 |
| 112 | wolverines | ann arbor    |          0 |
+-----+------------+--------------+------------+
2 rows in set (0.00 sec)

mysql> select * from playsfor;
+-----+-----+------------+
```

```
| pid | tid | avg_score2 |
+-----+-----+------------+
|   0 | 111 |          0 |
|   2 | 111 |          0 |
|   2 | 112 |          0 |
+-----+-----+------------+
3 rows in set (0.00 sec)

mysql> select * from games;
+-----+-------+
| gid | gname |
+-----+-------+
| 300 | sp-wv |
+-----+-------+
1 row in set (0.00 sec)
```

Inserting 4 rows into PTG table:

```
mysql> insert into ptg VALUES(0,111,290,4),(2,111,290,2),(0,111,300,7),(2
Query OK, 4 rows affected (0.17 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

Final State of Tables:

```
mysql> select * from players;
+-----+-------+--------------+------------+--------+------------+
| pid | name  | city         | phone      | salary | avg_score1 |
+-----+-------+--------------+------------+--------+------------+
|   0 | name1 | east lansing | 5177776666 |  70000 |        5.5 |
|   2 | name2 | ann arbor    | 2177776666 |  65000 |        1.5 |
+-----+-------+--------------+------------+--------+------------+
2 rows in set (0.00 sec)

mysql> select * from teams;
+-----+------------+--------------+-------------+
| tid | tname      | city         | avg_score3  |
+-----+------------+--------------+-------------+
| 111 | spartan    | east lansing | 4.333333333 |
| 112 | wolverines | ann arbor    |           1 |
+-----+------------+--------------+-------------+
2 rows in set (0.00 sec)

mysql> select * from playsfor;
+-----+-----+------------+
| pid | tid | avg_score2 |
+-----+-----+------------+
|   0 | 111 |        5.5 |
|   2 | 111 |          2 |
|   2 | 112 |          1 |
+-----+-----+------------+
3 rows in set (0.00 sec)
```