CSCI585 | HW2 Rubrics | Fall 2022

These are sample solutions to the problems. <u>Please note</u> that there could be more than one way to solve the question which has not been described below. If the query written by the student produces the correct output and is logical, then full marks should be awarded.

Q1.

A possible solution for the two problems can be this -

In the table ProjectRoomBookings, we just maintain roomNum, startTime, and groupName. We DO NOT keep the field 'endTime' in the table. So now the table creation looks like this -

```
CREATE TABLE ProjectRoomBookings
(roomNum INTEGER NOT NULL,
startTime INTEGER NOT NULL,
groupName CHAR(10) NOT NULL,
PRIMARY KEY (roomNum, startTime));
```

Explanation -

This means that for each slot booking for a particular room by a particular group, we add an entry to the ProjectRoomBookings table for each hour in the interval for which the room is booked. For example, if a group say "DBChamps" wants to book room say 1 for the slot 8 AM - 10 AM, then there will be two entries in the table for this. One entry will be (1,8,DBChamps) for 8 AM - 9 AM. Second entry will be (1,9,DBChamps) for 9 AM - 10 AM. Since roomNum and startTime both form a composite PK, this will be unique.

Problem 1 - Start Time could be incorrectly entered to be later than End Time.
This problem gets solved with the above fix as there there is no concept of the end time in the table now. So we do not need to maintain the end time now since we are storing each hour a particular group has booked a specific room.

Problem 2 - A new entry (for a new group) could be accidentally put in to occupy a room, even before the existing group in that room is done using that room.
This problem gets solved with the above fix as the roomNum,startTime pair is unique as it is the composite PK. No two entries can have the same roomNum and startTime but different groupName as this would lead to a violation. For example - if a group say "DBChamps" wants to book room say 1 for the slot 8 AM - 10 AM, then there will be two entries in the table for this. One entry will be (1,8,DBChamps) for 8 AM - 9 AM. Second entry will be (1,9,DBChamps) for 9 AM - 10 AM. Since roomNum and startTime both form a composite PK, this will be unique. Now if a group say "AlgoChamps" wants to book room say 1 for the slot 9 AM - 11 AM, the entries that would have to be made to make this possible will be (1,9,AlgoChamps) and (1,10,AlgoChamps). But the table already has an entry for (1,9,"DBChamps") and hence the "AlgoChamps" group would not be able to book the room until 10 AM when the "DBChamps" leave the room.

Another possible solution for the two problems can be this -

Problem 1 - Start Time could be incorrectly entered to be later than End Time.

```
CREATE TABLE ProjectRoomBookings(
roomNum INTEGER NOT NULL,
startTime INTEGER NOT NULL,
endTime INTEGER NOT NULL,
groupName CHAR(10) NOT NULL,
PRIMARY KEY (roomNum, startTime),
CHECK (endTime > startTime),
CHECK (startTime BETWEEN 7 and 18),
CHECK (endTime BETWEEN 7 and 18)
);
```

We add the necessary checks to the table creation statement to make sure that problem 1 does not arise.

Problem 2 - A new entry (for a new group) could be accidentally put in to occupy a room, even before the existing group in that room is done using that room.

```
CREATE TRIGGER tr_book
BEFORE INSERT ON ProjectRoomBookings
for each row
begin
declare stime int;
declare etime int;
set stime=(SELECT startTime from ProjectRoomBookings);
set etime=(SELECT endTime from ProjectRoomBookings);
if new.startTime>stime and new.startTime<etime
then signal sqlstate '45000';
end if;
if new.startTime<stime and new.endTime>stime
then signal sqlstate '45000';
end if;
```

We add this trigger which runs before every insertion into our table and makes sure that problem 2 does not arise by raising the signal Userdefined Exception whenever there are overlapping entries.

PS: [Saty] - A **textual explanation** that **clearly** states what to do, is also acceptable [there can be a mild deduction for not being clear/specific - 'mild' because the HW description doesn't provide details of what makes for a good explanation].

Q2.

Database Used - livesql (or any other valid database)

Creating a Table -

```
CREATE TABLE Enrollment
(sid number not null,
className varchar(50) not null,
grade varchar(1) not null,
PRIMARY KEY(sid, class_name));
```

Inserting entries into the table -

```
INSERT ALL
  INTO Enrollment(sid, className, grade) values(123, 'Synthesis algorithms', 'A')
  INTO Enrollment(sid, className, grade) values(123, 'EDM synthesis', 'B')
  INTO Enrollment(sid, className, grade) values(123, 'MIDI controllers', 'B')
  INTO Enrollment(sid, className, grade) values(662, 'Sound mixing', 'B')
  INTO Enrollment(sid, className, grade) values(662, 'EDM synthesis', 'A')
  INTO Enrollment(sid, className, grade) values(662, 'Electronic Music Fundamentals', 'A')
  INTO Enrollment(sid, className, grade) values(662, 'MIDI controllers', 'B')
  INTO Enrollment(sid, className, grade) values(345, 'MIDI controllers', 'A')
  INTO Enrollment(sid, className, grade) values(345, 'Electronic Music Fundamentals', 'B')
  INTO Enrollment(sid, className, grade) values(345, 'EDM synthesis', 'A')
  INTO Enrollment(sid, className, grade) values(555, 'EDM synthesis', 'B')
  INTO Enrollment(sid, className, grade) values(555, 'Electronic Music Fundamentals', 'B')
  INTO Enrollment(sid, className, grade) values(213, 'Electronic Music Fundamentals', 'A')
SELECT * FROM dual;
```

Query to output result -

```
SELECT className AS “Class Name”, COUNT(className) AS “Num Of Student Enrolled”
FROM Enrollment
GROUP BY className
ORDER BY COUNT(class_name) DESC;
```

Q3.

Database Used - livesql (or any other valid database)

Creating a Table -

```
CREATE TABLE ProjectStatus
(pid varchar(10) not null,
step number not null,
status varchar(1) not null,
PRIMARY KEY(pid, step));
```

Inserting entries into the table -

```
INSERT ALL
  INTO ProjectStatus(pid, step, status) values('P100', 0, 'C')
  INTO ProjectStatus(pid, step, status) values('P100', 1, 'W')
  INTO ProjectStatus(pid, step, status) values('P100', 2, 'W')
  INTO ProjectStatus(pid, step, status) values('P201', 0, 'C')
  INTO ProjectStatus(pid, step, status) values('P201', 1, 'C')
  INTO ProjectStatus(pid, step, status) values('P333', 0, 'W')
  INTO ProjectStatus(pid, step, status) values('P333', 1, 'W')
  INTO ProjectStatus(pid, step, status) values('P333', 2, 'W')
  INTO ProjectStatus(pid, step, status) values('P333', 3, 'W')
SELECT * FROM dual;
```

Query to output result -

```
SELECT pid AS “Project ID”
FROM ProjectStatus
WHERE step = 0 AND status = 'C' AND pid IN (SELECT pid FROM ProjectStatus WHERE status = 'W' AND step = 1);
```

Since Step 0 means just starting the project, every project will have 2 or more steps (including 0). And we can assume that all Cs will occur before all Ws. Hence we can only check step 1 for W and conclude.

NOTE: If a student makes an assumption that a project can have just 1 step as well and writes some query that gives the correct output taking into consideration this assumption, then that is fine as well since the question does not explicitly state that each project has 2 or more steps.

Q4.

Since the student can create whatever table(s) he/she needs, and populate it/them with their own data, there is no fixed answer for this question. If the query written by the student produces the correct output and is logical, then full marks should be awarded.

One possible solution to this problem can be:

Classes

prof	class	hr_Rate	num_students
A	1	50	20
A	2	50	30
B	3	60	10
B	4	60	40
B	5	60	50
C	6	70	10

Prof_table

prof	hr_rate
A	50
B	60
C	70

Since bonus = hourly_rate * sum_of_class_counts * 0.1, the query for the above shall be -

```
SELECT MAX(bonus)
        FROM (
        SELECT prof, hr_rate * sum_of_class_counts * 0.1 as bonus
        FROM (
                SELECT a.prof, a.sum_of_class_counts, b.hr_rate
                FROM
                        (SELECT Prof, SUM(num_students) as sum_of_class_counts
                        FROM classes
                        GROUP BY prof) as a
                INNER JOIN Prof_table as b
                ON a.prof = b.prof) as temp) as temp2
```

Q5. **One possible solution to this problem can be:**

```
SELECT instructors
FROM table
WHERE subject in (‘Electronic Music Fundamentals’,’MIDI controllers’,’EDM synthesis’) AND count(subject) = 3
```

Another possible solution:

```
SELECT instructor
FROM table inst1
WHERE inst1.subject = 'Electronic Music Fundamentals'
AND EXISTS (SELECT inst1.instructor FROM table inst2 WHERE inst2.subject = ‘MIDI controllers’ AND inst1.instructor =
inst2.instructor
AND EXISTS (SELECT inst2.instructor FROM table inst3 WHERE inst3.subject = ‘EDM synthesis’ AND inst2.instructor =
inst3.instructor));
```

Another possible solution:

```
SELECT instructor
FROM table AS inst RIGHT JOIN subjects AS subj
ON inst.subject = subj.subject
GROUP BY inst.instructor
HAVING COUNT(inst.instructor) = (SELECT COUNT(*) FROM subjects);
```

Another possible solution:

```
SELECT instructor
FROM table
WHERE Subject = 'Electronic Music Fundamentals'
INTERSECT
SELECT Instructor
FROM table
WHERE Subject = 'MIDI controllers'
INTERSECT
SELECT Instructor
FROM table
WHERE Subject = 'EDM synthesis';
```