General [IMPORTANT]

According to HW2 specs, any of the following situations causes a deduction of 0.5 pts.

- Don't name files properly (naming convention refers to HW2 specs.), However,
    - Error in file extension (e.g. Q1.sql.txt, see Piazza @111) is OK.
    - Because of case insensitiveness, both **q1.sql** or **Q1.SQL** should be OK.
- Submit a .zip (required to submit individual files).

The following depend on cases.

- Submit/resubmit after the deadline. (be sure to do it ahead of time, and make sure you did)
  Refer to the deadline policy in this course, 10% penalty per day after the deadline.
- A solution is partially wrong/correct or doesn't explain properly. (See details in upcoming sections).
  Give half pts (0.5pts for Q1-4, 1pt for Q5, 0.5pts for each Q5.bonus)
- A solution is totally wrong, doesn't make any sense (**rarely happens**).
  Deduct full pts (1pt for Q1-4, 2pts for Q5).

Other potential cases

- **Alternative solutions are VERY possible**, better to look into/think of/run their scripts thoroughly.
- A fully correct answer for Q2-5 should contain only **A Query statement**, not multiple query statements.
- A .sql may not be successfully executable (by "*source Q#.sql*"), including but not limited to the following:
    - .sql files are not executable due to missing of ";" at the end of any statements.
    - Some students may comment out those DDL/DML in their answers.

  Then we can easily fix them. E.g. Kindly copy/paste into the terminal if you would like to run it. **If this happens almost everywhere in one's submissions (making you mad and you don't know how to fix it), then we propose to deduct 0.5 or 1 pt (depending on how many mistakes and how mad you are) due to the unsuccessful execution of .sql files**. Otherwise, be lenient and nice :)
- TBD

Don't forget to leave proper feedback, which makes regrading easy.

Setup and Cleanup

Make sure you are using Mysql Shell connecting to a Mysql instance.

To run a solution on Mysql Shell, if they don't Setup or Cleanup (only solution is provided), then we may need to Setup (e.g. create database, use database, etc.) and Cleanup (e.g. drop database, etc.) **for grading purposes**. Could refer to provided solution files.

Please read the following rubrics along with the solution code and HW2 specs.

```
DROP DATABASE IF EXISTS testDB;

CREATE DATABASE testDB;
USE testDB;

CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);

Insert into Persons
VALUES(23, "Chary", "Saty", "USC", "LA");

select * from Persons;

DROP DATABASE IF EXISTS testDB;

For Ubuntu 20.04

#Installation

sudo apt-get remove docker docker-engine docker.io containerd runc

sudo apt-get update

sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg
--dearmor -o /etc/apt/keyrings/docker.gpg

echo "deb [arch=$(dpkg --print-architecture)
signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo
tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io
docker-compose-plugin
```

```
#Install mysql
sudo docker pull mysql

#Verify
sudo docker images

#Install mysql shell
https://dev.mysql.com/doc/mysql-shell/8.0/en/mysql-shell-install-linux-qui
ck.html
1. Download "Adding the MySQL APT Repository" configuration deb and run
it.
sudo dpkg -i /PATH/version-specific-package-name.deb
2. sudo apt-get update
3. sudo apt-get install mysql-shell

####Installation done####

#Run mysql docker image
sudo docker run -d -e MYSQL_ROOT_PASSWORD=test --name hw2 -p
127.0.0.1:3307:3306 mysql
sudo docker ps

#Make mysql server always restart on boot
sudo docker update --restart always hw2

#Connect to mysql
mysqlsh --uri root@127.0.0.1:3307
```

Q1

- Equivalent or alternative solution, once it makes sense, either SQL statements **OR textual explanation** is acceptable.
- It's OK if SQL statements are provided but fail to be successfully executed. However, the explanation must make sense.

For the 1st issue, see if there is a CHECK when creating the table, it could also come with a trigger or a constraint.

startTime < endTime.

For the 2nd issue, see if a trigger (may combine with procedure or function) is created to avoid insertion of the following situation.

(NEW.startTime >= startTime AND NEW.startTime < endTime) OR
(NEW.endTime > startTime AND NEW.endTime <= endTime))

Another easy way (without concerning the efficiency) to solve both issues could be Piazza @91, which assumes multiple INSERT statements when booking a period of time. It’s acceptable.

```
-- Setup

DROP DATABASE IF EXISTS HW2Q1DB;

CREATE DATABASE HW2Q1DB;
USE HW2Q1DB;

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

-- Solution

/*    To solve the 1st issue */
ALTER TABLE ProjectRoomBookings
ADD CONSTRAINT CHK_BOOKING CHECK (
            startTime>=7 AND /* OK if they don't have */
            endTime<=18 AND        /* OK if they don't have */
            startTime < endTime /* This check is IMPORTANT */
);


/*    To solve the 2nd issue */
DELIMITER $$
CREATE TRIGGER roomAvailability
BEFORE INSERT
ON ProjectRoomBookings FOR EACH ROW
BEGIN
      IF ( SELECT COUNT(*) FROM ProjectRoomBookings
            WHERE roomNum = NEW.roomNum
      AND ((NEW.startTime >= startTime AND NEW.startTime < endTime) OR
         (NEW.endTime > startTime AND NEW.endTime <= endTime)) ) <> 0 /*
This check is IMPORTANT */
    THEN
      SET @message_text = CONCAT('Room ', NEW.roomNum, ' is occupied
during ', NEW.startTime, ' to ', NEW.endTime, '.');
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = @message_text;
      END IF;
END$$
```

```
DELIMITER ;
/* Could define a trigger for UPDATE in the same way,
      but not necessary once an answer show the above
*/


-- Test cases (can be ignored) --

/*  Test1:
      Valid value
*/
INSERT INTO ProjectRoomBookings
VALUES(1, 7, 11, "Team1");

INSERT INTO ProjectRoomBookings
VALUES(1, 11, 12, "Team2");

SELECT * FROM ProjectRoomBookings;

DELETE FROM ProjectRoomBookings WHERE groupName = "Team1";
DELETE FROM ProjectRoomBookings WHERE groupName = "Team2";

/*  Test2:
      Invalid startTime (or endTime)
*/
INSERT INTO ProjectRoomBookings
VALUES(1, 6, 18, "Team1");

SELECT * FROM ProjectRoomBookings;

DELETE FROM ProjectRoomBookings WHERE groupName = "Team1";

/*  Test3:
      startTime > endTime
*/
INSERT INTO ProjectRoomBookings
VALUES(1, 9, 7, "Team1");

SELECT * FROM ProjectRoomBookings;

DELETE FROM ProjectRoomBookings WHERE groupName = "Team1";

/*  Test4:
      Book unavailable room
*/

INSERT INTO ProjectRoomBookings
```

```
VALUES(1, 7, 11, "Team1");

INSERT INTO ProjectRoomBookings
VALUES(1, 8, 10, "Team2");

SELECT * FROM ProjectRoomBookings;

DELETE FROM ProjectRoomBookings WHERE groupName = "Team1";
DELETE FROM ProjectRoomBookings WHERE groupName = "Team2";

-- Clean up options
DROP TRIGGER IF EXISTS roomAvailability;
DROP DATABASE IF EXISTS HW2Q1DB;
```

Q2

- To get full points, an answer must show the correct SQL query statement
- The testing data could be what is shown in HW2.Q2 specs.
- In the output, it’s OK if ***className***s with the same ***Total*** # switch places. (i.e. Javascript and Python, Processing and Java)

```
-- Setup

DROP DATABASE IF EXISTS HW2Q2DB;

CREATE DATABASE HW2Q2DB;
USE HW2Q2DB;

/*	A student won't take a class many times, since it's within a term. */
CREATE TABLE Enrollment (
	SID CHAR(3) NOT NULL,
	ClassName CHAR(11) NOT NULL,
	Grade CHAR(2),
	PRIMARY KEY (SID, ClassName)
);

INSERT INTO Enrollment VALUES ('123', 'Processing', 'A');
INSERT INTO Enrollment VALUES ('123', 'Python', 'B');
INSERT INTO Enrollment VALUES ('123', 'Scratch', 'B');
INSERT INTO Enrollment VALUES ('662', 'Java', 'B');
INSERT INTO Enrollment VALUES ('662', 'Python', 'A');
INSERT INTO Enrollment VALUES ('662', 'JavaScript', 'A');
INSERT INTO Enrollment VALUES ('662', 'Scratch', 'B');
INSERT INTO Enrollment VALUES ('345', 'Scratch', 'A');
INSERT INTO Enrollment VALUES ('345', 'JavaScript', 'B');
INSERT INTO Enrollment VALUES ('345', 'Python', 'A');
INSERT INTO Enrollment VALUES ('555', 'Python', 'B');
INSERT INTO Enrollment VALUES ('555', 'JavaScript', 'B');
INSERT INTO Enrollment VALUES ('213', 'JavaScript', 'A');

SELECT * FROM Enrollment;

-- Solution
SELECT ClassName, COUNT(SID) AS Total
FROM Enrollment
GROUP BY ClassName
ORDER BY Total DESC;


-- Clean up option
DROP DATABASE IF EXISTS HW2Q2DB;
```

Q3

- To get full points, an answer must show the correct SQL query statement.
- **NOTE**, there may be **a project with step 0 only**. So in addition to what is shown in HW2.Q3 specs, the testing data should add one more row (refer to the solution code). OR, See if they have an assumption that this kind of project doesn't exist (i.e. In a project, the # of steps always greater than 0). Otherwise, the query should be able to handle this situation.

```
-- Setup

DROP DATABASE IF EXISTS HW2Q3DB;

CREATE DATABASE HW2Q3DB;
USE HW2Q3DB;

CREATE TABLE ProjectStatus (
        PID CHAR(4) NOT NULL,
        Step INTEGER NOT NULL,
        Status CHAR(1) NOT NULL,
        PRIMARY KEY (PID, Step)
);
```

```
INSERT INTO ProjectStatus VALUES ('P100', 0, 'C');
INSERT INTO ProjectStatus VALUES ('P100', 1, 'W');
INSERT INTO ProjectStatus VALUES ('P100', 2, 'W');
INSERT INTO ProjectStatus VALUES ('P201', 0, 'C');
INSERT INTO ProjectStatus VALUES ('P201', 1, 'C');
INSERT INTO ProjectStatus VALUES ('P333', 0, 'W');
INSERT INTO ProjectStatus VALUES ('P333', 1, 'W');
INSERT INTO ProjectStatus VALUES ('P333', 2, 'W');
INSERT INTO ProjectStatus VALUES ('P333', 3, 'W');

/*	Exceptional case:
	A project has step 0 only.
*/
INSERT INTO ProjectStatus VALUES ('P099', 0, 'C');

-- Solution

SELECT PID
FROM ProjectStatus
WHERE Step = 0
	AND Status = 'C'
	AND PID NOT IN (SELECT DISTINCT PID FROM ProjectStatus WHERE
Status = 'C' AND Step > 0);

/*	The key word DISTINCT is not really necessary */
/*	Note: The query would output 'P100' and 'P099' */



-- Clean up option
DROP DATABASE IF EXISTS HW2Q3DB;
```

Q4

Grading this one needs more effort than Q1-Q3.

- To get full points, an answer must show the correct SQL query statement.
- The answer depends on the tables created, data inserted, the hourly_rate set. They must **show or explain** how they achieve those (e.g. data definition/manipulation, assumptions, etc). Otherwise, even though the solution looks neat, it's partially wrong.
- To grade this question, if the query statement looks confusing, may try to generate several test dataset based on their data schema and assumptions.
- For better understanding, the solution code is also presented by steps. Due to different designs of data schema, students may skip some steps. In other words, they assume

some intermediate VIEWs have been created as Tables. It’s OK once the design data schema makes sense.

- Again, a fully correct answer should contain only **A Query statement**, not multiple query statements.

```
-- Setup

DROP DATABASE IF EXISTS HW2Q4DB;

CREATE DATABASE HW2Q4DB;
USE HW2Q4DB;


-- Setup: Using the tables given in this Q2 and Q5

/*    Assumptions:
1. A student won't take a class many times, since it's within a term.
2. Professors who teach a class always share the same # of students. i.e.
no seperated sessions, etc.
3. hourly_rate = 50
*/
CREATE TABLE Enrollment (
      SID CHAR(3) NOT NULL,
      ClassName CHAR(11) NOT NULL,
      Grade CHAR(2),
      PRIMARY KEY (SID, ClassName)
);

INSERT INTO Enrollment VALUES ('123', 'Processing', 'A');
INSERT INTO Enrollment VALUES ('123', 'Python', 'B');
INSERT INTO Enrollment VALUES ('123', 'Scratch', 'B');
INSERT INTO Enrollment VALUES ('662', 'Java', 'B');
INSERT INTO Enrollment VALUES ('662', 'Python', 'A');
INSERT INTO Enrollment VALUES ('662', 'JavaScript', 'A');
INSERT INTO Enrollment VALUES ('662', 'Scratch', 'B');
INSERT INTO Enrollment VALUES ('345', 'Scratch', 'A');
INSERT INTO Enrollment VALUES ('345', 'JavaScript', 'B');
INSERT INTO Enrollment VALUES ('345', 'Python', 'A');
INSERT INTO Enrollment VALUES ('555', 'Python', 'B');
INSERT INTO Enrollment VALUES ('555', 'JavaScript', 'B');
INSERT INTO Enrollment VALUES ('213', 'JavaScript', 'A');

CREATE TABLE Instructor (
      Instructor CHAR(6) NOT NULL,
      Subject CHAR(11) NOT NULL,
      PRIMARY KEY (Instructor, Subject)
);
```

```
INSERT INTO Instructor VALUES ('Aleph', 'Scratch');
INSERT INTO Instructor VALUES ('Aleph', 'Java');
INSERT INTO Instructor VALUES ('Aleph', 'Processing');
INSERT INTO Instructor VALUES ('Bit', 'Python');
INSERT INTO Instructor VALUES ('Bit', 'JavaScript');
INSERT INTO Instructor VALUES ('Bit', 'Java');
INSERT INTO Instructor VALUES ('CRC', 'Python');
INSERT INTO Instructor VALUES ('CRC', 'JavaScript');
INSERT INTO Instructor VALUES ('Dat', 'Scratch');
INSERT INTO Instructor VALUES ('Dat', 'Python');
INSERT INTO Instructor VALUES ('Dat', 'JavaScript');
INSERT INTO Instructor VALUES ('Emscr', 'Scratch');
INSERT INTO Instructor VALUES ('Emscr', 'Processing');
INSERT INTO Instructor VALUES ('Emscr', 'JavaScript');
INSERT INTO Instructor VALUES ('Emscr', 'Python');

-- Soulution

/* Step 1: Calculate number of students in each class */
CREATE VIEW ClassCount AS
SELECT ClassName, COUNT(SID) AS Total
FROM Enrollment
GROUP BY ClassName;

SELECT * FROM ClassCount;

/* Step 2: Map professor to # of students in each class */

CREATE VIEW Instructor_ClassCount AS
SELECT I.Instructor AS Instructor,
         I.Subject AS Subject,
         C.Total AS Total
FROM Instructor I
JOIN ClassCount C
ON I.Subject = C.ClassName;

SELECT * FROM Instructor_ClassCount;

/* Step 3: Calculate bounus for each professor */

CREATE VIEW InstructorBonus AS
SELECT Instructor, SUM(Total) AS sum_of_class_counts
FROM Instructor_ClassCount
GROUP BY Instructor;

SELECT * FROM InstructorBonus;
```

```
/* Step 4: Calculate MAX bounus */
SELECT MAX(50 * sum_of_class_counts * 0.1) AS "Highest Bonus"
FROM InstructorBonus;


/* To wrap up everything, this is the final solution */
/* Again, assume hourly_rate = 50 */
SELECT MAX(50 * sum_of_class_counts * 0.1) AS "Highest Bonus"
FROM (
      SELECT Instructor, SUM(Total) AS sum_of_class_counts
      FROM (
            SELECT I.Instructor AS Instructor,
            I.Subject AS Subject,
            C.Total AS Total
            FROM Instructor I
            JOIN (
                  SELECT ClassName, COUNT(SID) AS Total
                  FROM Enrollment
                  GROUP BY ClassName
            ) C
            ON I.Subject = C.ClassName
      ) AS Instructor_ClassCount
      GROUP BY Instructor
) AS InstructorBonus;

-- Clean up option
DROP DATABASE IF EXISTS HW2Q4DB;
```

Q5

Grading this one needs more effort than Q1-Q3.

- To get full points, an answer must show the correct SQL query statement **and explain what the query does**. Otherwise, partially wrong.
- "Can hardcode the subjects just for submission purposes, but your query should work for ANY such table!" **If they hardcode the subject names in the query, it's partially wrong**, because if the # of required classes != 3, the query doesn't work. (See Piazza @80).
- Bonus points.
  - "+1 extra point if an additional solution is in a very different way!" +2 points at most, if they come up with more than 2 different ways. **The total score of this HW2 never exceeds 6 pts.**
  - **If Q5.sql is partially correct, but Q5_v2.sql is fully correct, one should get 2 (for Q5_v2) + 0.5(for Q5) = 2.5 pts.** You know what this means :)

  - 'Very different' means - the approaches do have to be totally distinct, eg. you can't use NOT to invert an existing solution, or use IN() instead of OR, etc. In other words, it means something semantically different instead of syntactically different only. **If a query is not 'different', it deserves no points**.
- Again, a fully correct answer should contain only **A Query statement**, not multiple query statements.

```
-- Setup

DROP DATABASE IF EXISTS HW2Q5DB;

CREATE DATABASE HW2Q5DB;
USE HW2Q5DB;

CREATE TABLE Instructor (
      Instructor CHAR(6) NOT NULL,
      Subject CHAR(11) NOT NULL,
      PRIMARY KEY (Instructor, Subject)
);

INSERT INTO Instructor VALUES ('Aleph', 'Scratch');
INSERT INTO Instructor VALUES ('Aleph', 'Java');
INSERT INTO Instructor VALUES ('Aleph', 'Processing');
INSERT INTO Instructor VALUES ('Bit', 'Python');
INSERT INTO Instructor VALUES ('Bit', 'JavaScript');
INSERT INTO Instructor VALUES ('Bit', 'Java');
INSERT INTO Instructor VALUES ('CRC', 'Python');
INSERT INTO Instructor VALUES ('CRC', 'JavaScript');
INSERT INTO Instructor VALUES ('Dat', 'Scratch');
INSERT INTO Instructor VALUES ('Dat', 'Python');
INSERT INTO Instructor VALUES ('Dat', 'JavaScript');
INSERT INTO Instructor VALUES ('Emscr', 'Scratch');
INSERT INTO Instructor VALUES ('Emscr', 'Processing');
INSERT INTO Instructor VALUES ('Emscr', 'JavaScript');
INSERT INTO Instructor VALUES ('Emscr', 'Python');

/* As mentioned in the HW2 specs
      You can hardcode the subjects just for submission purposes,
      but your query should work for ANY such table!
*/
CREATE TABLE RequiredSubject (
      Subject CHAR(11) UNIQUE
);

INSERT INTO RequiredSubject VALUES ('JavaScript');
INSERT INTO RequiredSubject VALUES ('Scratch');
INSERT INTO RequiredSubject VALUES ('Python');
```

```
-- Solution 1

SELECT Instructor
FROM Instructor I, RequiredSubject R
WHERE I.Subject = R.Subject
/* WHAT THE QUERY DOES:
      Find out Instructors who can teach RequiredSubjects
      Since it's a many-to-many relation,
      in the result up to here, one row shows a Instructor-to-Subject
mapping.
*/
GROUP BY Instructor
HAVING COUNT(*) = (SELECT COUNT(*) FROM RequiredSubject);
/* WHAT THE QUERY DOES:
      Aggregate Instructors with restricting condition that
      the # of Subject a Instructor can teach should equal to the total #
of RequiredSubjects.
*/

/* Many alternative answers will be simliar to this one.
      e.g. to use JOIN is an acceptable different answer.

      SELECT Instructor
      FROM Instructor I
      JOIN RequiredSubject R
      ON I.Subject = R.Subject
      GROUP BY Instructor
      HAVING COUNT(*) = (SELECT COUNT(*) FROM RequiredSubject);
*/


-- Solution 2

/* To use a Procedure, then the hard-code RequiredSubject Table is not
necessary */
DELIMITER $$

CREATE PROCEDURE searchQualifiedInstructor    (
      IN requiredSubjects CHAR(255)
      /* WHAT THE QUERY DOES:
            Pass requiredSubjects as a string variable in stead of
hard-code Table now
      */
)
BEGIN
      SELECT Instructor
      FROM Instructor
      WHERE FIND_IN_SET(Subject, requiredSubjects)
      /* WHAT THE QUERY DOES:
            if Subject appears in requiredSubjects list, then output who
teaches this Subject
      */
      GROUP BY Instructor
      HAVING COUNT(*) = LENGTH(requiredSubjects)
             - LENGTH( REPLACE ( requiredSubjects, ",", "") ) + 1;
            /* WHAT THE QUERY DOES:
                        A trick to get the total # of RequiredSubjects =
                        # of "," occurrences + 1
            */
END $$

DELIMITER ;

call searchQualifiedInstructor('JavaScript,Scratch,Python');
/* WHAT THE QUERY DOES:
      Just call the Procedure to get the result
*/


-- Clean up option
DROP DATABASE IF EXISTS HW2Q5DB;
```