CS303: Assignment 2

B Siddharth Prabhu
200010003@iitdh.ac.in

27 September 2022

1 Answers to Question 1: Grade Assignment

Given database table is as follows:

```
marks (ID, score)
```

Before starting querying, we must make note of the assumptions to be made about this database:

- `ID` is a Primary Key.
- `score` is of a Numeric Data Type.

(a) Write an SQL Query to display the grade for each student, based on the marks relation.

```sql
SELECT ID, score,
    CASE
    WHEN score<40 THEN "F"
    WHEN score<60 AND score>=40 THEN "C"
    WHEN score<80 AND score>=60 THEN "B"
    ELSE "A"
    END AS grade
FROM marks;
```

(b) Write an SQL Query to find the number of students with each grade.

```sql
WITH gradelist(ID, score, grade) AS (
    SELECT ID, score,
        CASE
        WHEN score<40 THEN "F"
        WHEN score<60 AND score>=40 THEN "C"
        WHEN score<80 AND score>=60 THEN "B"
        ELSE "A" END AS grade
    FROM marks
    )
SELECT grade, COUNT(ID)
FROM gradelist
GROUP BY grade
ORDER BY grade;
```

2 Answers to Question 2: Employee Schema

Given database tables are as follows:

```
employee (employee name, street, city)
works (employee name, company name, salary)
company (company name, city)
manages (employee name, manager name)
```

Before starting querying, we must make note of the assumptions to be made about this database:

- Just like Question 1 of the CS303 Assignment 1, the choice of Primary Key is ambiguous here. There should be an employee ID for unique identification, and also a company ID would be ideal. The next two points present the problems and the solution.
- Problem: Even if we take the PK of 'employee' table to be (`employee name`, `street`, `city`), the database will fail if two employees of the same name live on the same street in the same city. ***Not only that,*** but this has other implications: The 'manages' table would fail (e.g. if we want the manager of a particular employee) since two employees of the same name may have different managers. So, distinguishability would be a problem.
- Solution: To tackle the issue described in the point above, we shall make the assumption that `employee name` is unique, or simply contains an employee ID within itself, much like the default Moodle usernames! A similar assumption would be made for company name. ***(This is an elaborate way to say "Let the employee/company names be unique.")*** Note that back-ticks (`) are used for multi-word column names.

(a) Write an SQL Query to give all employees of "First Bank Corporation" a 10% raise.

This Query uses the SQL DML UPDATE statement.

```
UPDATE works
SET salary = 1.1*salary
WHERE `company name` = "First Bank Corporation";
```

(b) Write an SQL Query to give all managers of "First Bank Corporation" a 10% raise.

This Query uses the SQL DML UPDATE statement, and the IN operator.

```
UPDATE works
SET salary = 1.1*salary
WHERE `employee name` IN (
    SELECT `manager name`
    FROM manages
)
AND `company name` = "First Bank Corporation";
```

(c) Write an SQL Query to delete all tuples in the works relation for employees of "Small Bank Corporation".

```
DELETE FROM works
WHERE `company name` = "Small Bank Corporation";
```

(d) Write an SQL Query to find all employees in the database who live in the same cities as the companies for which they work.

```
SELECT `employee name`
FROM (employee NATURAL JOIN works), company AS C
WHERE works.`company name` = C.`company name`
AND employee.city = C.city;
```

(e) Write an SQL Query to find all employees in the database who live in the same cities and on the same streets as do their managers.

```
WITH managerdetails(`manager name`, `street`, `city`)
AS (
    SELECT *
    FROM employee
    WHERE `employee name` IN (
        SELECT `manager name`
        FROM manages
    )
)
SELECT `employee name`
FROM (employee NATURAL JOIN manages), managerdetails AS M
WHERE manages.`manager name` = M.`manager name`
AND employee.city = M.city
AND employee.street = M.street;
```

(f) Write an SQL Query to find all employees who earn more than the average salary of all employees of their company.

```
SELECT `employee name`
FROM works AS W1
WHERE salary > (
    SELECT AVG(salary)
    FROM works AS W2
    WHERE W1.`company name` = W2.`company name`
);
```

(g) Write an SQL Query to find the company that has the smallest payroll.

Payroll can mean one of two things. Let us consider both meanings:

```
-- If assuming that payroll is number of people who get paid by a company
WITH c_payrolls(`company name`, `payroll size`)
AS (
    SELECT `company name`, COUNT(`employee name`)
    FROM works
    WHERE salary > 0
    -- if salary is zero, we assume employee is NOT on payroll of a company.
    GROUP BY `company name`
)
SELECT *
FROM c_payrolls
WHERE `payroll size` = (
    SELECT MIN(`payroll size`)
    FROM c_payrolls
);
```

```
-- If Assuming that payroll is total amount paid out by a company
-- first, get total salary amt paid out by each company
WITH c_payrolls(`company name`, `payroll`)
AS (
    SELECT `company name`, SUM(salary)
    FROM works
    GROUP BY `company name`
)
SELECT *
FROM c_payrolls
WHERE `payroll` = (
    SELECT MIN(`payroll`)
    FROM c_payrolls
);
```

(h) Write an SQL Query to find the company that has the most employees.

```
WITH c_empl(`company name`, `num of empl`)
AS (
    SELECT `company name`, COUNT(`employee name`)
    FROM works
    -- if salary is zero, we assume employee is NOT on payroll of a company.
    -- *BUT they are still an employee of the company.*
    GROUP BY `company name`
)
SELECT *
FROM c_empl
WHERE `num of empl` = (
    SELECT MAX(`num of empl`)
    FROM c_empl
);
```

(i) Write an SQL Query to find those companies whose employees earn a higher salary, on average, than the average salary at "First Bank Corporation".

Here, we use scalar subquerying, and the WITH clause.

```
WITH c_averages(`company name`, `avg_salary`)
AS (
    SELECT `company name`, AVG(salary)
    FROM works
    GROUP BY `company name`
)
SELECT `company name`
FROM c_averages
WHERE `avg_salary` > (
    SELECT `avg_salary`
    FROM c_averages
    WHERE `company name` = "First Bank Corporation"
);
```

(j) Write an SQL Query to modify the database so that "Jones" now lives in "Newtown".

```
UPDATE employee
SET city = "Newtown"
WHERE `employee name` = "Jones";
```

(k) Write an SQL Query to give all managers of "First Bank Corporation" a 10 percent raise unless the salary becomes greater than $100,000; in such cases, give only a 3 percent raise.

The assumption we make about the question here is that: In case the salary would be more than $100,000 ***after*** a 10% increase, then give only a 3% salary hike. Else, just give a 10% salary hike.

```
UPDATE works
SET salary = (
    CASE
    WHEN salary*1.1 > 100000 THEN salary*1.03
    ELSE salary*1.1
    END
)
WHERE `employee name` IN (
    SELECT `manager name`
    FROM manages
)
AND `company name` = "First Bank Corporation";
```

3 Answer to Question 3: Training Schema

Given database tables are as follows:

```
users (user_id, username)
training_details (user_training_id, user_id, training_id, training_date)
```

Before starting querying, we must make note of the assumptions to be made about this database:

- We are given that 'user_id' is PK of table 'users', and 'user_training_id' is PK of table 'training_details'. We are asked to find "users who have taken a training lesson more than once in a day".
- We may have a case where user 1 may have done training lesson 1 multiple times in a day. To get the count of such trainings, we must group by (user_id, training_id, training_date).
- We may have a case where user 1 does training lesson 1 'x' times on some day, and also training lesson 2 'y' times on the same day. We may want (x + y). Here, we group by (user_id, training_date).
- Let us consider each case, since the question contains the wording: "grouped by user and training lesson", but also says "taken **a** training lesson" which is still vague.

The required SQL Queries are as follows:

```sql
-- Approach 1: user 1 does training lesson 1 multiple times, and we want count of that.
-- (Also assuming we just want *user details* of such users)
SELECT user_id, username
FROM users
WHERE user_id IN (
    SELECT user_id
    FROM training_details
    GROUP BY user_id, training_id, training_date
    HAVING COUNT(user_training_id) > 1
);

-- Approach 2: user 1 does multiple lessons per day, which may or may not be distinct.
-- Here, we want number of lessons done by a user in a day.
-- (This time let's assume we want user details and training date as well)
SELECT users.user_id, username, training_date, COUNT(*) AS num_of_times
FROM users, training_details
WHERE users.user_id = training_details.user_id
GROUP BY user_id, training_date
HAVING COUNT(*) > 1
ORDER BY user_id ASC, training_date DESC;
```

Note the assumptions made in each approach, as the meanings are very different.

4 Answer to Question 4: Running Race Schema

Given database tables are as follows:

```
Runners (id, name)
Races (id, event, winner_id)
```

We assume that 'id' is the PK within each table, and that 'winner_id' column of table 'Races' references Runners(id). We are given a query, which we have to interpret in simple words:

```
SELECT * FROM runners WHERE id NOT IN (
    SELECT winner_id FROM races
);
```

Answer: In simple words, we are getting all the info (id, name) of those runners who have NOT won any of the racing events (only if all winner_id entries are NOT NULL).
For this database, it will return EMPTY SET because for every single record comparison with respect to the outer query, the "NOT IN" returns unknown/null.

5 Answers to Question 5: Book Publisher Schema

Given database tables are as follows:

```
books (book_id, title, total_pages, rating, isbn, published_date, publisher_id)
publishers (publisher_id, name)
```

Before starting querying, we must make note of the assumptions to be made about this database:

- We are given that 'book_id' is PK of table 'books', and 'publisher_id' is PK of table 'publishers'. We are also given ER diagram of the database, and we can infer that books(publisher_id) references publishers(publisher_id).
- A book may have 0 or 1 publisher associated with it, while a publisher may have 0, 1, or any non-negative number of books associated with them.
- Based on this, we can easily get the desired output using (natural) left and right outer joins.

(a) Write an SQL query which will return information about books with publishers, irrespective of whether a book has associated publishers or not.

```
SELECT *
FROM books NATURAL LEFT OUTER JOIN publisher;
```

Books with no corresponding publishers in the database will still appear in the output, with NULL entry under publishers 'name' column.

(b) Write an SQL query which will return information about books with publishers, irrespective of whether the publisher has any published books or not.

```
SELECT *
FROM books NATURAL RIGHT OUTER JOIN publisher;
```

Publishers with no corresponding books in the database will still appear in the output, with NULL entry under all books columns.