CS303 : DataBases and Information Systems

Assignment 2

Sourabh Bhosale
200010004

September 28, 2022

1 Problem 1

Suppose that we have relation marks(ID, score) and we wish to assign grades to students based on the score as follows : grade F if score $<$ 40, grade C if 40 $\leq$ score $<$ 60, grade B if 60 $\leq$ score $<$ 80, and grade A if 80 $\leq$ score. Write SQL queries to do the following :

(a) Display the grade for each student, based on the marks relation.

If we want to update the original table, then first approach does that.

```
-- approach 1
ALTER TABLE marks
  ADD grade VARCHAR(5);
UPDATE marks
SET grade = (CASE
             WHEN score >= 80 THEN 'A'
             WHEN score >= 60 AND score < 80 THEN 'B'
             WHEN score >= 40 AND score < 60 THEN 'C'
             ELSE 'F'
             END);
SELECT *
FROM marks;
```

If we don't want to update the original table, then second approach does that.

```
-- approach 2
SELECT ID, marks, (CASE
             WHEN score >= 80 THEN 'A'
             WHEN score >= 60 AND score < 80 THEN 'B'
             WHEN score >= 40 AND score < 60 THEN 'C'
             ELSE 'F'
             END) AS grade
FROM marks;
```

(b) Find the number of students with each grade.

```
SELECT grade, COUNT(ID)
FROM marks
GROUP BY grade;
```

2 Problem 2

Write SQL queries for each of the following.

Given information about tables.

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

Assumptions: Here, we are assuming that the choice of Primary Key (PK) is ambiguous here. So we shall make the assumption that 'employee name' is unique, similar cases while considering the 'company name' for company relation etc.

(a) Give all employees of "First Bank Corporation" a 10 percent raise.

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

(b) Give all managers of "First Bank Corporation" a 10 percent raise.

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

(c) Delete all tuples in the works relation for employees of "Small Bank Corporation".

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

(d) 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
WHERE employee.city = company.city AND
  works.`company name` = company.`company name`;
```

(e) Find all employees in the database who live in the same cities and on the same streets as do their managers.

```
SELECT A.`employee name`
FROM employee AS A,manages AS M, employee AS B
WHERE A.`employee name` = M.`employee name` AND
  M.`manager name` = B.`employee name` AND
  A.street = B.street AND A.city = B.city;
```

(f) Find all employees who earn more than the average salary of all employees of their company.

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

(g) Find the company that has the smallest payroll.

In the first approach, we made the assumption that payroll is sum of salaries of all the employees in the company.

```
-- approach 1
SELECT `company name`
FROM works
GROUP BY `company name`
HAVING SUM(salary) <= ALL (SELECT SUM(salary)
                           FROM works
                           GROUP BY `company name`);
```

In the second approach, we made the assumption that payroll is total number of all the employees in the company.

```
-- approach 2
SELECT `company name`
FROM works
GROUP BY `company name`
HAVING COUNT(`employee name`) <= ALL (SELECT COUNT(`employee name`)
                                      FROM works
                                      WHERE salary > 0
                                      GROUP BY `company name`);
```

(h) Find the company that has the most employees.

```
SELECT `company name`
FROM works
GROUP BY `company name`
HAVING COUNT(DISTINCT `employee name`) >= ALL (
          SELECT COUNT(DISTINCT `employee name`)
          FROM works
          GROUP BY `company name`);
```

(i) Find those companies whose employees earn a higher salary, on average, than the average salary at "First Bank Corporation".

```
SELECT `company name`
FROM works
GROUP BY `company name`
HAVING AVG(salary) > (SELECT AVG(salary)
                      FROM works
                      WHERE `company name` = "First Bank Corporation");
```

(j) Modify the database so that "Jones" now lives in "Newtown".

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

(k) 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.

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

3 Problem 3

Consider the following 2 tables - "users" and "training_details". Write a query to get the list of users who took a training lesson more than once in the same day, grouped by user and training lesson, each ordered from the most recent lesson date to oldest date.

user_id	username
1	John Doe
2	Jane Don
3	Alice Jones
4	Lisa Romero

Table 1: users

user_training_id	user_id	training_id	training_date
1	1	1	"2015-08-02"
2	2	1	"2015-08-03"
3	3	2	"2015-08-02"
4	4	2	"2015-08-04"
5	2	2	"2015-08-03"
6	1	1	"2015-08-02"
7	3	2	"2015-08-04"
8	4	3	"2015-08-03"
9	1	4	"2015-08-03"
10	3	1	"2015-08-02"
11	4	2	"2015-08-04"
12	3	2	"2015-08-02"
13	1	1	"2015-08-02"

Table 2: training_details

3.1 Query

In the first approach, we are considering for each specific training lesson (i.e. for each specific training_id)

```
-- approach 1
SELECT username, training_date, training_id, COUNT(user_training_id)
FROM users
INNER JOIN training_details ON(users.user_id = training_details.user_id)
GROUP BY username, training_date, training_id
HAVING COUNT(*) > 1
ORDER BY username, training_date DESC;
-- for specific training id.
```

In the second approach, we are counting all training lessons taken in same day.

```
-- approach 2
SELECT username, training_date, COUNT(user_training_id)
FROM users
INNER JOIN training_details ON(users.user_id = training_details.user_id)
GROUP BY username, training_date
HAVING COUNT(*) > 1
ORDER BY username, training_date DESC;
```

4 Problem 4

4.1 Consider the following tables - 'runners' and 'races'. What is the meaning of the query given below?

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

id	name
1	John Doe
2	Jane Doe
3	Alice Jones
4	Bobby Louis
5	Lisa Romero

Table 3: runners

id	event	winner_id
1	100 meter dash	2
2	500 meter dash	3
3	cross-country	2
4	triathalon	NULL

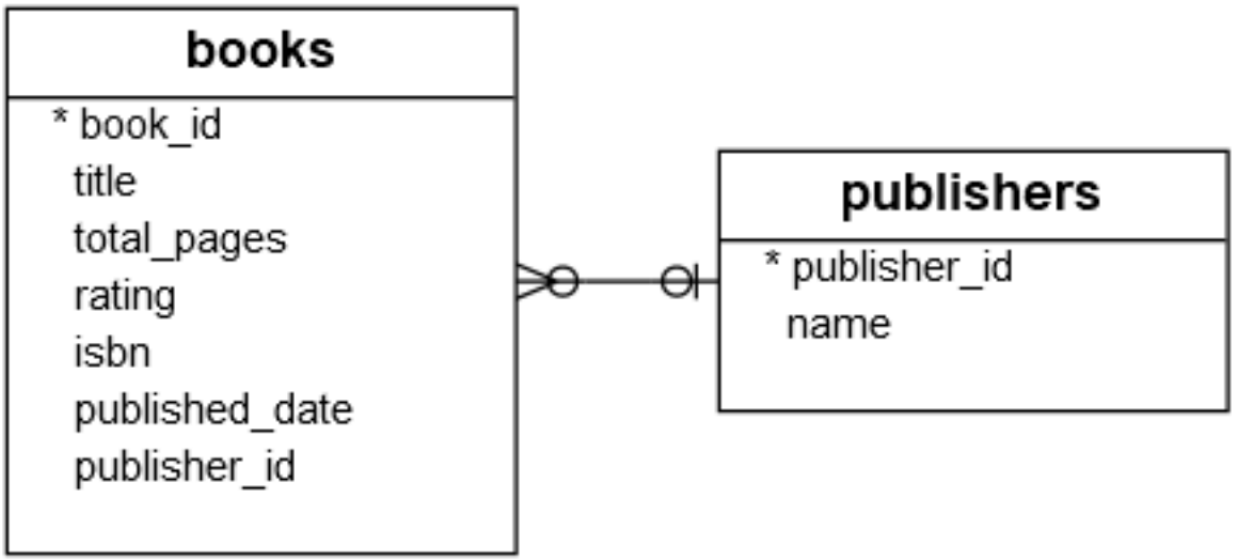
Table 4: races

4.2 Results

```
Answer: The result of the query will be an empty set. The reason for
that is mentioned below:
If the set being evaluated by the SQL NOT IN condition contains any
values that are null, then the outer query here will return an empty
set, even if there are many runner ids that match winner_ids in the
races table.
The correct query that will avoid this issue would be as follows:
    SELECT *
    FROM runners
    WHERE id NOT IN (SELECT winner_id
                     FROM races
                     WHERE winner_id IS NOT null);
```

5 Problem 5

Consider the following 2 tables: books and publishers. Book_id and publisher_id are primary keys in the corresponding tables.



A publisher may have zero or many books while a book may belong to zero or one publisher. The relationship between the books table and the publishers table is zero-to-many. Based on the information given above, write queries for the following:

(a) A 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 publishers;
```

(b) A 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 publishers;
```