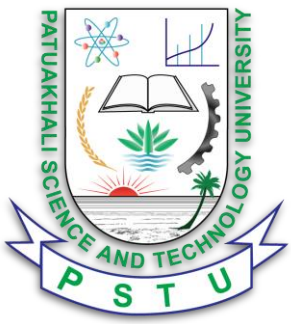

Course Title : Database System Sessional
Course Code : CCE - 224

Lab Problem : 03

Submitted To

Professor Dr. Md. Samsuzzaman
Department of Computer and Communication Engineering
Faculty Of Computer Science & Engineering.

Submitted By

Name	:	Md Mohidul Alam
Id	:	1902016
Reg. No	:	08722
Session	:	2019 - 20

Patuakhali Science & Technology University
Dumki, Patuakhali - 8602

1 (a)

Find the titles of courses in the Comp. Sci. department that have 3 credit

```
SELECT title FROM course WHERE dept_name = 'Computer
Science‘ AND credits = 3;
```

1 (b)

Find the IDs of all students who were taught by an instructor named Einstein; make sure there are no duplicates in the result.

```
SELECT DISTINCT student.ID FROM student
JOIN takes ON takes.ID = student.ID
JOIN teaches USING(course_id, sec_id, semester, year)
JOIN instructor WHERE instructor.name = 'Jane Smith';
```

1 (c)

Find the highest salary of any instructor

```
select max(salary) from instructor
```

1 (d)

Find all instructors earning the highest salary (there may be more than one with the same salary).

```
select ID, name
from instructor
where salary = (select max(salary) from instructor)
```

1 (e)

Find the enrollment of each section that was offered in Fall 2017.

```
select course id, sec id, count(ID)
from section natural join takes
where semester = 'Autumn'
and year = 2009
group by course_id, sec_id
```

1 (f)

Find the maximum enrollment, across all sections, in Fall 2017.

```
SELECT MAX(enrollment)
FROM (
  SELECT COUNT(ID) AS enrollment
  FROM section
  NATURAL JOIN takes
  WHERE semester = 'Autumn' AND year = 2009
  GROUP BY course_id, sec_id
) AS enrollment_table;
```

1 (g) Find the sections that had the maximum enrollment in Fall 2017

```
WITH sec_enrollment AS (
  SELECT course_id, sec_id, COUNT(ID) AS enrollment
  FROM section
  NATURAL JOIN takes
  WHERE semester = 'Autumn' AND year = 2009
  GROUP BY course_id, sec_id
)
SELECT course_id, sec_id
FROM sec_enrollment
WHERE enrollment = (
  SELECT MAX(enrollment)
  FROM sec_enrollment
);
```

2 (a) Find the total grade points earned by the student with ID '12345', across all courses taken by the student.

```
SELECT SUM(grade_points.points * takes.grade) AS
total_grade_points
FROM takes
JOIN grade_points ON takes.grade = grade_points.grade
WHERE takes.ID = '12345';
```

2 (b) Find the grade point average (GPA) for the above student, that is, the total grade points divided by the total credits for the associated courses.

```
SELECT SUM(credits * points)/SUM(credits) AS GPA
FROM (takes NATURAL JOIN course) NATURAL JOIN
grade_points
WHERE ID = '12345';
```

2 (c) Find the ID and the grade-point average of each student.

```
SELECT ID, SUM(credits * points)/SUM(credits) AS GPA
FROM (takes NATURAL JOIN course) NATURAL JOIN grade_points
GROUP BY ID;
```

3 (a) Increase the salary of each instructor in the Comp. Sci. department by 10%.

```
UPDATE instructor
SET salary = salary * 1.1
WHERE dept_name = 'Computer Science';
```

3 (b) Delete all courses that have never been offered (i.e., do not occur in the section relation).

```
DELETE FROM course
WHERE NOT EXISTS (
    SELECT *
    FROM section
    WHERE section.course_id = course.course_id
    AND section.course_id = course.course_id
);
```

3 (c) Insert every student whose tot cred attribute is greater than 100 as an instructor in the same department, with a salary of $10,000

```
INSERT INTO instructor (ID, name, dept_name, salary)
SELECT ID, name, dept_name, 10000
FROM student
WHERE tot_cred > 100;
```

4 (a) Find the total number of people who owned cars that were involved in accidents in 2017

```
SELECT COUNT(DISTINCT owner_id)
FROM cars
WHERE  car_id  IN  (
    SELECT  car_id
    FROM accidents
    WHERE YEAR(accident_date) = 2017
);
```

4 (b) Delete all year-2010 cars belonging to the person whose ID is '12345'.

```
DELETE FROM cars
WHERE owner_id = '12345'
AND YEAR(year) = 2010;
```

5 (a)

Suppose that we have a 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 ≤ score < 60, grade B if 60 ≤ score < 80, and grade A if 80 ≤ score. Write SQL queries to do the following:

- **a. Display the grade for each student, based on the marks relation.**
- **b. Find the number of students with each grade.**

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

```sql
SELECT grade, COUNT(*) as num_students
FROM (
  SELECT ID,
    CASE
      WHEN score < 40 THEN 'F'
      WHEN score >= 40 AND score < 60 THEN 'C'
      WHEN score >= 60 AND score < 80 THEN 'B'
      ELSE 'A'
    END AS grade
  FROM marks
) AS grade_table
GROUP BY grade;
```

6

The SQL like operator is case sensitive (in most systems), but the lower() function on strings can be used to perform case-insensitive matching. To show how,
write a query that finds departments whose names contain the string “sci” as a
substring, regardless of the case.

```sql
SELECT *
FROM department
WHERE lower(dept_name) LIKE '%sci%';
```

7

The SQL query selects values of p.a1 that are either in r1 or in r2 when there is a match between the values in p.a1 and either r1.a1 or r2.a1.
If either r1 or r2 is empty, then the query will still return the values of p.a1 that match the non-empty relation. For example, if r1 is empty, the query will still return the values of p.a1 that match the values in r2.a1. Similarly, if r2 is empty, the query will still return the values of p.a1 that match the values in r1.a1. If both r1 and r2 are empty, the query will return an empty result set.

8 (a) Find the ID of each customer of the bank who has an account but not a loan

```
SELECT DISTINCT a.ID
FROM account a
WHERE a.ID NOT IN (
  SELECT l.ID
  FROM loan l
)
```

8 (b) Find the ID of each customer who lives on the same street and in the same city as customer '12345'

```
SELECT DISTINCT c.ID
FROM customer c
WHERE c.street = (
  SELECT street
  FROM customer
  WHERE ID = '12345'
) AND c.city = (
  SELECT city
  FROM customer
  WHERE ID = '12345'
) AND c.ID <> '12345'
```

8 (c) Find the name of each branch that has at least one customer who has an account in the bank and who lives in “Harrison”.

```
SELECT DISTINCT b.name
FROM branch b
JOIN account a ON b.branch_id = a.branch_id
JOIN customer c ON a.ID = c.ID
WHERE c.city = 'Harrison'
```

9 (a) Find the ID, name, and city of residence of each employee who works for “First Bank Corporation”

```
SELECT ID, name, city
FROM Employee
WHERE company = 'First Bank Corporation';
```

9 (b) Find the ID, name, and city of residence of each employee who works for “First Bank Corporation” and earns more than $10000.

```
SELECT ID, name, city
FROM Employee
WHERE company = 'First Bank Corporation'
AND salary > 10000;
```

9 (c) Find the ID of each employee who does not work for “First Bank Corporation”.

```
SELECT ID
FROM Employee
WHERE company != 'First Bank Corporation';
```

9(d)

Find the ID of each employee who earns more than every employee of "Small Bank Corporation".

```
SELECT e1.ID
FROM Employee e1
WHERE salary > ALL (SELECT salary FROM
Employee WHERE
company = 'Small Bank Corporation');
```

9 (e)

Assume that companies may be located in several cities. Find the name of each company that is located in every city in which "Small Bank Corporation" is located.

```
SELECT c1.name  FROM
Company c1  WHERE NOT
EXISTS (
    SELECT city FROM Company c2 WHERE c2.name = 'Small Bank
Corporation'
    AND NOT EXISTS (
        SELECT * FROM Company c3 WHERE c3.name = c1.name
AND c3.city = c2.city
    )
);
```

9 (f)

Find the name of the company that has the most employees (or companies, in the case where there is a tie for the most).

```
SELECT name
FROM Company
GROUP BY name
HAVING COUNT(*) = (
    SELECT COUNT(*)
    FROM Employee
   GROUP BY company
    ORDER BY COUNT(*) DESC
    LIMIT 1
);
```

9 (g)

Find the name of each company whose employees earn a higher salary, on average, than the average salary at "First Bank Corporation".

```
SELECT c1.name
FROM Company c1, Employee e1
WHERE c1.name = e1.company
GROUP BY c1.name
HAVING AVG(e1.salary) > (SELECT AVG(salary) FROM
Employee WHERE company = 'First Bank Corporation');
```

10 (a)

Modify the database so that the employee whose ID is '12345' now lives in “Newtown”.

```
UPDATE employee
SET city = 'Newton'
WHERE name = 'Jones';
```

10 (b)

Give each manager of “First Bank Corporation” a 10 percent raise unless the salary becomes greater than $100000; in such cases, give only a 3 percent raise.

```
UPDATE employee
SET salary = CASE
   WHEN company = 'First Bank Corporation' AND title =
'Manager' AND salary * 1.1 <= 100000 THEN salary * 1.1
  WHEN company = 'First Bank Corporation' AND title =
'Manager' THEN salary * 1.03
  ELSE salary
END
WHERE company = 'First Bank Corporation' AND title =
'Manager';
```

11 (a)

Find the names of all students who have taken at least one Comp. Sci. course; make sure there are no duplicate names in the result.

```
SELECT name
FROM student
NATURAL JOIN takes
NATURAL JOIN course
WHERE course.dept = 'Comp. Sci.';
```

11 (b)

Find the IDs and names of all students who have not taken any course offering before Spring 2009.

```
select id, name
from student
except
select id, name
from student natural join takes
where year < 2009
```

11 (c)

For each department, find the maximum salary of instructors in that department. You may assume that every department has at least one instructor.

```
SELECT dept, MAX(salary)
FROM instructor
GROUP BY dept;
```

11 (d)

Find the lowest, across all departments, of the per-department maximum salary computed by the preceding query.

```
SELECT MIN(maxsalary)
FROM (SELECT dept, MAX(salary) AS maxsalary
      FROM instructor
     GROUP BY dept) AS deptmax
```

12 (a)

Create a new course "CS-001", titled "Weekly Seminar", with 0 credits.

```
INSERT INTO course
(course_id, title,
dept_name, credits)
VALUES ('CS-001', 'Weekly
Seminar', 'Comp. Sci.', 0);
```

12 (b)

Create a section of this course in Autumn 2009, with section id of 1.

```
INSERT INTO section
VALUES ('CS-001', 1,
'Autumn', 2009, NULL, NULL,
NULL);
```

12 (c)

Enroll every student in the Comp. Sci. department in the above secton.

```
INSERT INTO takes
SELECT id, 'CS-001', 1, 'Autumn', 2009, NULL
FROM student
WHERE dept_name = 'Comp. Sci.';
```

12 (d)

Delete enrollments in the above section where the student's name is Chavez.

```
DELETE FROM takes
WHERE course_id = 'CS-001' AND section_id =
1 AND year = 2009
AND semester = 'Autumn'
AND id IN (SELECT id FROM student WHERE
name = 'Chavez')
```

12 (e)

Delete the course CS-001. What will happen if you run this delete statement without first deleting offerings (sections) of this course.

```
delete from takes
where course id = 'CS-001'

delete from section
where course id = 'CS-001'

delete from course
where course id = 'CS-001'
```

12 (f)

Delete all takes tuples corresponding to any section of any course with the word "database" as a part of the title; ignore case when matching the word with the title.

```
DELETE FROM takes
WHERE course_id IN (
  SELECT course_id
  FROM course
  WHERE LOWER(title) LIKE '%database%'
)
```

13

Write SQL DDL corresponding to the schema in Figure 3.18. Make any reasonable assumptions about data types, and be sure to declare primary and foreign keys.

- person (driver id, name, address)
- car (license, model, year)
- accident (report number, date, location)
- owns (driver id, license)
- participated (report number, license, driver id, damage amount)

```
CREATE TABLE person (
    driver_id VARCHAR(50) PRIMARY KEY,
    name VARCHAR(50),
   address VARCHAR(50)
);
```

```
CREATE TABLE car (
  license varchar(50) PRIMARY KEY,
  model varchar(50),
  year integer
);
```

```
CREATE TABLE accident (
    report_number INTEGER,
    date DATE,
    location VARCHAR(50),
    PRIMARY KEY (report_number)
);
```

```
CREATE TABLE owns (
  driver_id varchar(50),
  license varchar(50),
  PRIMARY KEY (driver_id, license),
  FOREIGN KEY (driver_id) REFERENCES person(driver_id),
  FOREIGN KEY (license) REFERENCES car(license)
);
```

```
CREATE TABLE participated (
    report_number integer,
    license varchar(50),
    driver_id varchar(50),
    damage_amount integer,
    primary key (report_number, license),
    foreign key (report_number) references accident(report_number),
    foreign key (license) references car(license)
);
```

13 (a)

Find the number of accidents in which the cars belonging to "John Smith" were involved.

```
SELECT COUNT(*)
FROM accident
WHERE EXISTS (
  SELECT *
  FROM participated, owns, person
  WHERE owns.driver_id = person.driver_id
  AND person.name = 'John Smith'
  AND owns.license = participated.license
  AND accident.report_number = participated.report_number
)
```

13 (b)

Update the damage amount for the car with license number "AABB2000" in the accident with report number "AR2197" to $3000.

```
UPDATE participated
SET damage_amount = 3000
WHERE report_number = 'AR2197' AND license = 'AABB2000';
```

14 (a)

Find all customers who have an account at all the branches located in "Brooklyn"

```
with branchcount as (
    select count(*) as branch_count
    from branch
    where branch_city = 'Brooklyn'
)
select customer_name
from customer c
where branch_count = (
    select count(distinct branch_name)
    from customer join depositor using
                  (customer_name) join account using
                  (account_number) join branch using
                  (branch_name)
    where customer_name = c.customer_name
)
```

14 (b) Find out the total sum of all loan amounts in the bank

```
select sum(amount) from loan
```

14 (c) Find the names of all branches that have assets greater than those of at least one branch located in “Brooklyn”.

```
SELECT branch_name
FROM branch
WHERE assets >
(SELECT assets
FROM branch
WHERE branch_city = 'Brooklyn');
```

15 (a) Find the names of all employees who work for First Bank Corporation

```
SELECT employee_name
FROM works
WHERE company_name = 'First Bank Corporation';
```

15 (b) Find all employees in the database who live in the same cities as the companies for which they work.

```
SELECT e.employee_name
FROM employee e, works w, company c
WHERE e.employee_name = w.employee_name
  AND e.city = c.city
  AND w.company_name = c.company_name;
```

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

```
SELECT P.employee_name
 FROM employee P, employee R, manages M
 WHERE P.employee_name = M.employee_name
   AND M.manager_name = R.employee_name
   AND P.street = R.street
   AND P.city = R.city;
```

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

```
select employee name from works T where salary > (select avg
(salary) from works S where T.company name = S.company name)
```

15 (e) Find the company that has the smallest payroll.

```
SELECT company_name
FROM works
GROUP BY company_name
HAVING SUM(salary) <= ALL (
    SELECT SUM(salary)
    FROM works
   GROUP BY company_name
)
```

16 (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'
```

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

```
UPDATE works
SET salary = salary * 1.1
WHERE employee_name IN (SELECT manager_name FROM manages)
AND company_name = 'First Bank Corporation';
```

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

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

17 List two reasons why null values might be introduced into the database.

- ❖ “null” signifies an unknown value.
- ❖ “null” is also used when a value does not exist.

18 Show that, in SQL, <> all is identical to not in

Let the set S denote the result of an SQL subquery. We compare (x <> all S) with (x not in S). If a particular value x1 satisfies (x1 <> all S) then for all elements y of S x1 6= y. Thus x1 is not a member of S and must satisfy (x1 not in S). Similarly, suppose there is a particular value x2 which satisfies (x2 not in S). It cannot be equal to any element w belonging to S, and hence (x2 <> all S) will be satisfied. Therefore the two expressions are equivalent.

20 (a) Print the names of members who have borrowed any book published by "McGraw-Hill".

```
SELECT name
FROM member m, book b, borrowed l
WHERE m.memb_no = l.memb_no
AND l.isbn = b.isbn
AND b.publisher = 'McGrawHill';
```

20 (b) Print the names of members who have borrowed all books published by "McGraw-Hill".

```
SELECT DISTINCT m.name
FROM member m
WHERE NOT EXISTS (
    (SELECT isbn
    FROM book
    WHERE publisher = 'McGrawHill')
    EXCEPT
    (SELECT isbn
    FROM borrowed l
    WHERE l.memb_no = m.memb_no)
);
```

20 (c) For each publisher, print the names of members who have borrowed more than five books of that publisher.

```
SELECT publisher, name
FROM (
    SELECT publisher, name, COUNT(isbn)
    FROM member m, book b, borrowed l
    WHERE m.memb_no = l.memb_no
    AND l.isbn = b.isbn
    GROUP BY publisher, name
) AS membpub (publisher, name, count_books)
WHERE count_books > 5;
```

20 (d) Print the average number of books borrowed per member. Take into account that if an member does not borrow any books, then that member does not appear in the borrowed relation at all.

```
with memcount as
(select count(*) as total_members
from member)
select count(*)/memcount.total_members as
average_borrowed_per_member
from borrowed, memcount;
```

21

Rewrite the where clause where unique (select title from course) without using the unique construct.

```
SELECT COUNT(title)
FROM course
=
SELECT COUNT(DISTINCT title)
FROM course;
```

22

Consider the query:

```
select course id, semester, year, section id, avg (credits
earned) from takes natural join student where year = 2009
group by course id, semester, year, section id having count
(ID) >= 2;
```

Explain why joining section as well in the from clause would not change the result.

The common attributes of takes and section form a foreign key of takes, referencing section. As a result, each takes tuple would match at most one one section tuple, and there would not be any extra tuples in any group. Further, these attributes cannot take on the null value, since they are part of the primary key of takes. Thus, joining section in the from clause would not cause any loss of tuples in any group. As a result, there would be no change in the result.

22

Consider the query:

```
with dept total (dept name, value) as (select dept name,
sum(salary) from instructor group by dept name), dept total
avg(value) as (select avg(value) from dept total) select dept
name from dept total, dept total avg where dept total.value
>= dept total avg.value;
```

Rewrite this query without using the with construct

There are several ways to write this query. One way is to use subqueries in the where clause, with one of the subqueries having a second level subquery in the from clause as below.

```
SELECT DISTINCT department AS d
FROM instructor i
WHERE (SELECT SUM(salary)
       FROM instructor
       WHERE department = i.department)
       >= (SELECT AVG(s)
           FROM (SELECT SUM(salary) AS s
                 FROM instructor
                 GROUP BY department) AS temp);
```