Assignment

```
mysql> use assignment;
Database changed
mysql> CREATE TABLE employee (
    ->     employee_id INT PRIMARY KEY,
    ->     name VARCHAR(50),
    ->     department_id INT,
    ->     salary DECIMAL(10, 2)
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql> CREATE TABLE department (
    ->     department_id INT PRIMARY KEY,
    ->     department_name VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql> INSERT INTO employee (employee_id, name, department_id, salary)
    -> VALUES
    ->     (1, 'Alice', 101, 6000.00),
    ->     (2, 'Bob', 102, 5000.00),
    ->     (3, 'Charlie', 101, 5500.00);
Query OK, 3 rows affected (0.05 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> INSERT INTO department (department_id, department_name)
    -> VALUES
    ->     (101, 'HR'),
    ->     (102, 'Finance'),
    ->     (103, 'IT');
Query OK, 3 rows affected (0.05 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> CREATE TABLE project (
    ->     project_id INT PRIMARY KEY,
    ->     project_name VARCHAR(50),
    ->     department_id INT,
    ->     start_date DATE
    -> );
```

```
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO project (project_id, project_name, department_id,
start_date)
    -> VALUES
    ->     (1, 'Project A', 101, '2023-01-15'),
    ->     (2, 'Project B', 102, '2023-02-10'),
    ->     (3, 'Project C', 103, '2023-03-05');
Query OK, 3 rows affected (0.04 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

```
mysql> select * from employee;
+-------------+---------+---------------+---------+
| employee_id | name    | department_id | salary  |
+-------------+---------+---------------+---------+
|           1 | Alice   |           101 | 6000.00 |
|           2 | Bob     |           102 | 5000.00 |
|           3 | Charlie |           101 | 5500.00 |
+-------------+---------+---------------+---------+
3 rows in set (0.00 sec)

mysql> select * from department;
+---------------+-----------------+
| department_id | department_name |
+---------------+-----------------+
|           101 | HR              |
|           102 | Finance         |
|           103 | IT              |
+---------------+-----------------+
3 rows in set (0.00 sec)

mysql> select * from project;
+------------+--------------+---------------+------------+
| project_id | project_name | department_id | start_date |
+------------+--------------+---------------+------------+
|          1 | Project A    |           101 | 2023-01-15 |
|          2 | Project B    |           102 | 2023-02-10 |
|          3 | Project C    |           103 | 2023-03-05 |
+------------+--------------+---------------+------------+
3 rows in set (0.00 sec)
```

PART 1

1.Write a SQL query to count the number of employees in each department. Display the department name and the count of employees.

```
mysql> select department.department_name,count(employee.employee_id) from department join employee on department.department_id = empl
oyee.department_id group by department.department_id;
+-----------------+-----------------------------+
| department_name | count(employee.employee_id) |
+-----------------+-----------------------------+
| HR              |                           2 |
| Finance         |                           1 |
+-----------------+-----------------------------+
2 rows in set (0.00 sec)
```

.

2.Write a SQL query to find all projects that are managed by the HR department. Display the project names and their start dates.

```
mysql> select project.project_name , project.start_date from project join department on department.department_id = project.department
_id where department.department_id = 101;
+--------------+------------+
| project_name | start_date |
+--------------+------------+
| Project A    | 2023-01-15 |
+--------------+------------+
1 row in set (0.00 sec)
```

3. Write a SQL statement to increase the salary of all employees in the Finance department by 10%.

```
mysql> update employee set salary = salary*1.1 where department_id = 102;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from employee;
+-------------+---------+---------------+---------+
| employee_id | name    | department_id | salary  |
+-------------+---------+---------------+---------+
|           1 | Alice   |           101 | 6000.00 |
|           2 | Bob     |           102 | 5500.00 |
|           3 | Charlie |           101 | 5500.00 |
+-------------+---------+---------------+---------+
3 rows in set (0.00 sec)
```

4. Write a SQL query to retrieve a combined list of all employee names and all project names. Ensure there are no duplicates in the result.

```
mysql> select name AS label from employee UNION select project_name AS label from project;
+-----------+
| label     |
+-----------+
| Alice     |
| Bob       |
| Charlie   |
| Project A |
| Project B |
| Project C |
+-----------+
6 rows in set (0.00 sec)
```

5. Write a SQL query to list all employee salaries and project IDs from the employees and projects tables. Include duplicates if they exist.

```
mysql> select employee.name,employee.salary, project.project_id from employee join project where project.project_id = employee.employee_id group
 by project.project_id;
+---------+---------+------------+
| name    | salary  | project_id |
+---------+---------+------------+
| Alice   | 6000.00 |          1 |
| Bob     | 5500.00 |          2 |
| Charlie | 5500.00 |          3 |
+---------+---------+------------+
3 rows in set (0.00 sec)
```

6. Write a SQL query to find the names of employees who are in both the HR and IT departments. Return only those employee names that exist in both departments.

```
mysql> SELECT name FROM employee WHERE department_id IN (SELECT department_id FROM department WHERE department_name = 'HR') INTERSECT SELECT nam
e FROM employee WHERE department_id IN (SELECT department_id FROM department WHERE department_name = 'IT');
Empty set (0.04 sec)
```

7. Write a SQL query to find names that are common between the employees table and the departments table.

```
mysql> select employee.name as label from employee INTERSECT select department.department_name as label from department;
Empty set (0.00 sec)
```

8. Write a SQL query to retrieve the names of employees who do not manage anyone (i.e., employees who do not have any direct reports). Use the MINUS operator to exclude managers from the list of all employees.

–

9. Write a SQL query to find department names that do not have any employees assigned to them. Use the MINUS operator to exclude departments with employees.

10. Write a SQL query to list all unique project names and their start dates from the projects table, ensuring there are no duplicates.

```
mysql> select distinct project_name, start_date from project;
+--------------+------------+
| project_name | start_date |
+--------------+------------+
| Project A    | 2023-01-15 |
| Project B    | 2023-02-10 |
| Project C    | 2023-03-05 |
+--------------+------------+
3 rows in set (0.01 sec)
```

11. Write a SQL query to count how many employees there are in the employees table and how many projects are listed in the projects table. Display both counts.

```
mysql> SELECT (SELECT COUNT(*) FROM employee) AS employee_count, (SELECT COUNT(*) FROM project) AS project_count;
+----------------+---------------+
| employee_count | project_count |
+----------------+---------------+
|              3 |             3 |
+----------------+---------------+
1 row in set (0.01 sec)
```

12. Write a SQL query that finds employee names who have a salary greater than 6000 and also project names. Use an intersection of two queries.

```
mysql> select employees.name from employees join projects on employees.depar
tment_id=projects.department_id where employees.salary>=6000;
+-------+
| name  |
+-------+
| Alice |
+-------+
1 row in set (0.00 sec)
```

13. Write a SQL query to combine all project names and employee names into a single list. Ensure there are no duplicates in the result.

```
mysql> select project_name name from projects union select name from employe
es;
+------------+
| name       |
+------------+
| Project A  |
| Project B  |
| Project C  |
| Alice      |
| Bob        |
| Charlie    |
+------------+
6 rows in set (0.00 sec)
```

14. Write a SQL query to find all employees working on projects starting with 'Project A' or 'Project B'. Use the UNION operation to combine results.

```
mysql> select employees.name from employees join projects on employees.depar
tment_id = projects.department_id where projects.project_name like 'Project A%' union select employees.name from employees join projects on employees.de
partment_id=projects.department_id where projects.project_name like 'Project B%';
+-------+
| name  |
+-------+
| Alice |
| Bob   |
+-------+
2 rows in set (0.00 sec)
```

15. Write a SQL query to find unique salary values from the employees table and project IDs from the projects table using the UNION operation.

```
mysql> select distinct salary from employees union select distinct project_i
d from projects;
+--------+
| salary |
+--------+
|   6000 |
|   5500 |
|      1 |
|      2 |
|      3 |
+--------+
5 rows in set (0.00 sec)
```

16. Write a SQL query to find names of employees who are also project managers (i.e., listed in both the employees and projects tables). Use the INTERSECT operation.

```
mysql> select employees.name from employees intersect select projects.projec
t_name from projects;
Empty set (0.00 sec)
```

17. Write a SQL query to retrieve a combined list of employees earning more than 7000 and all project names. Ensure that there are no duplicates in the result.

```
mysql> select distinct name from employees where salary>7000 union select di
stinct project_name from projects;
+-----------+
| name      |
+-----------+
| Project A |
| Project B |
| Project C |
+-----------+
3 rows in set (0.00 sec)
```

18. Write a SQL query to find all departments that do not have any projects assigned to them using the MINUS operator.

Part 2

1. Implement transaction control in a database using TCL commands (COMMIT, ROLLBACK,

SAVEPOINT) to manage data consistency during multiple operations (TABLE 2)

```
mysql> select * from accounts;
+------------+----------------+---------+
| account_id | account_holder | balance |
+------------+----------------+---------+
|        101 | Alice          | 5000.00 |
|        102 | Bob            | 3000.00 |
|        103 | Charlie        | 7000.00 |
+------------+----------------+---------+
3 rows in set (0.00 sec)
```

2. Write a transaction that simulates the following banking operations:
    a. Begin a transaction.

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
```

    b. Deduct $1000 from Alice's account.

```
mysql> update accounts set balance=balance-1000 where account_holder='Alice';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from accounts;
+------------+----------------+---------+
| account_id | account_holder | balance |
+------------+----------------+---------+
|        101 | Alice          | 4000.00 |
|        102 | Bob            | 3000.00 |
|        103 | Charlie        | 7000.00 |
+------------+----------------+---------+
3 rows in set (0.00 sec)
```

    c. Deduct $500 from Bob's account.

```
mysql> update accounts set balance=balance-500 where account_holder='Bob';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from accounts;
+------------+----------------+---------+
| account_id | account_holder | balance |
+------------+----------------+---------+
|        101 | Alice          | 4000.00 |
|        102 | Bob            | 2500.00 |
|        103 | Charlie        | 7000.00 |
+------------+----------------+---------+
3 rows in set (0.00 sec)
```

    d. Use a SAVEPOINT after deducting from Bob's account.

```
mysql> savepoint bob_deduction;
Query OK, 0 rows affected (0.00 sec)
```

e. Attempt to deduct $8000 from Charlie's account (this should fail due to insufficient balance)

```
mysql> update accounts set balance=balance-8000 where account_holder='Charlie';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from accounts;
+------------+----------------+----------+
| account_id | account_holder | balance  |
+------------+----------------+----------+
|        101 | Alice          |  4000.00 |
|        102 | Bob            |  2500.00 |
|        103 | Charlie        | -1000.00 |
+------------+----------------+----------+
3 rows in set (0.00 sec)
```

f. Roll back to the savepoint, ensuring only Charlie's transaction is undone.

```
mysql> rollback to savepoint bob_deduction;
Query OK, 0 rows affected (0.00 sec)
```

g. • Commit the transaction.

```
mysql> commit;
Query OK, 0 rows affected (0.01 sec)
```

3. After executing the above steps, query the bank_accounts table to verify the final balances

```
mysql> select * from accounts;
+------------+----------------+---------+
| account_id | account_holder | balance |
+------------+----------------+---------+
|        101 | Alice          | 4000.00 |
|        102 | Bob            | 2500.00 |
|        103 | Charlie        | 7000.00 |
+------------+----------------+---------+
3 rows in set (0.00 sec)
```