# SUBQUERY

A subquery in SQL is a query nested inside another query. It is used to retrieve data that will be used in the main query as a condition to further restrict the data to be retrieved. Typically enclosed within parentheses, a subquery can appear in various parts of an SQL statement, such as the SELECT, FROM, or WHERE clause.

The subquery is executed first, and its result is passed to the main query for further processing.

Subqueries can be nested within other subqueries to achieve more complex queries.

Subqueries are employed for tasks like filtering, calculations, or to retrieve information dynamically based on the results of another query. While powerful, excessive use of subqueries can impact performance, and alternatives like JOINs may be more efficient in certain scenarios.

A subquery encapsulates a complete SQL query within another query, allowing for a modular and organized approach to constructing complex queries.

# TYPES OF SUBQUERIES

## Single-Row Subquery:

A single-row subquery returns only one row of results.

Typically used in scenarios where the result of the subquery is compared with a single value.

Often involves comparison operators such as =, >, <, >=, <= to compare the result with a specific value.

Since it returns only one row, it's essential to handle scenarios where the subquery might not return any results or when it returns multiple rows (which could lead to errors).

Database engines often optimize single-row subqueries, but it's still crucial to ensure efficient query design for performance.

## Multi-Row Subquery:

A multi-row subquery returns multiple rows of results.

Typically used when the result of the subquery needs to be compared with or used in conjunction with multiple values.

Often involves multiple-row operators such as IN, ANY, ALL for comparisons with sets of values.

Since it returns multiple rows, careful consideration is needed to handle these results in the main query.

Database engines optimize multi-row subqueries, but it's crucial to ensure that the query is designed efficiently for performance.

## Correlated Subquery:

A correlated subquery is a subquery that refers to columns of the outer query, creating a relationship (correlation) between the main and subquery.

Correlated subqueries are employed when the subquery needs to be evaluated for each row processed by the outer query.

The subquery references columns from the tables in the outer query, creating a dynamic connection between the two.

For each row processed by the outer query, the subquery is executed, and its result is used in the context of that specific row.

While powerful, correlated subqueries can be less efficient than non-correlated ones, and careful optimization is necessary to maintain performance.

Correlated subqueries can be applied to various types of queries, including SELECT, UPDATE, and DELETE statements.

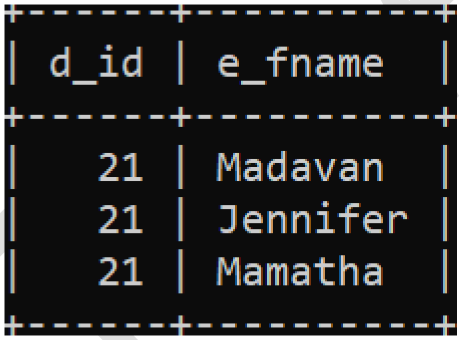
## Queries On Subquery

### 1. Write a query to display the department and the first name for all employees who works in the same department in which 'Jennifer' works.

**Solution:**

```
select d_id, e_fname
from employee
where d_id =
(
select d_id
from employee
where e_fname = "Jennifer");
```

**Output:**



```
+------+----------+
| d_id | e_fname  |
+------+----------+
|   21 | Madavan  |
|   21 | Jennifer |
|   21 | Mamatha  |
+------+----------+
```

## Explanation:

**SELECT d_id, e_fname:** The outer query selects the columns d_id and e_fname from the table employee.

**WHERE d_id = (...):** This condition filters the rows from the employee table based on the value of d_id.

**(SELECT d_id FROM employee WHERE e_fname = 'Jennifer'):** The subquery retrieves the d_id from the employee table where the e_fname is equal to 'Jennifer'. Correlation:

**d_id = (...):** The correlation occurs here. The result of the subquery is used to filter the outer query based on the condition d_id = (result of subquery).

The query aims to select the d_id and e_fname columns from the employee table for employees who share the same department (d_id) as the employee with the first name 'Jennifer'.

The result will be a list of employees who belong to the same department as Jennifer, including their department IDs (d_id) and first names (e_fname).

**Note:**

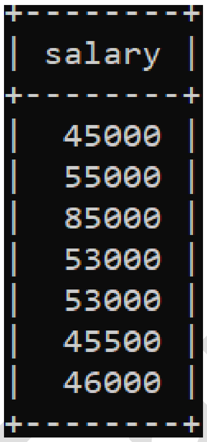
It's important to ensure that the subquery returns a single value, as the outer query expects a single value for the equality comparison (d_id = ...). If the subquery returns multiple values, it would result in an error.

## 2. Write a query to display the salary of all employees whose salary is greater than the average salary of employee.

### Solution:

```
select salary
from employee
where salary >
(
select avg(salary)
from employee);
```

### Output:



```
+--------+
| salary |
+--------+
|  45000 |
|  55000 |
|  85000 |
|  53000 |
|  53000 |
|  45500 |
|  46000 |
+--------+
```

## Explanation:

**SELECT salary:** The outer query selects the salary column from the employee table.

**WHERE salary > (...):** This condition filters the rows from the employee table based on the value of the salary being greater than a certain threshold.
Subquery:

**(SELECT avg(salary) FROM employee):** The subquery

calculates the average (avg) salary from the employee table.

**salary** > **(...):** The outer query filters employees based on the condition that their salary is greater than the average salary calculated by the subquery.

The query aims to retrieve the salaries of employees who earn more than the average salary of all employees in the employee table.
Result:

The result will be a list of salaries that are higher than the average salary in the employee table.

**Note:**

This type of query is useful for identifying employees with above-average salaries and can be adapted for various analytical purposes.
Consideration:

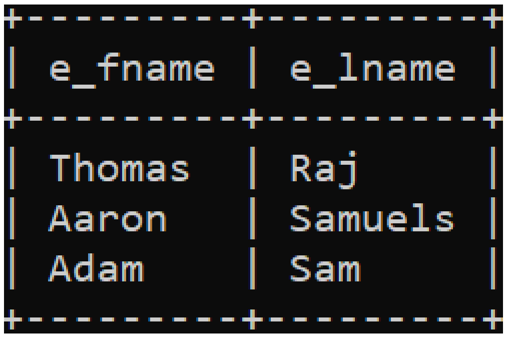
It assumes that the salary column is numeric and that there are records in the employee table for the comparison to be meaningful.

## 3. Write a query to display the names of employees who work in the same department as the employee with the highest salary.

**Solution:**

```
select e_fname, e_lname
from employee
where d_id =
(select d_id
from employee
where salary =
(select max(salary)
from employee));
```

## Output:



```
+---------+---------+
| e_fname | e_lname |
+---------+---------+
| Thomas  | Raj     |
| Aaron   | Samuels |
| Adam    | Sam     |
+---------+---------+
```

## Explanation:

**SELECT e_fname, e_lname:** The outer query selects the columns e_fname and e_lname from the employee table.

**WHERE d_id = (...):** This condition filters the rows

from the employee table based on the value of d_id.

**(SELECT max(salary) FROM employee):** The innermost subquery calculates the maximum (max) salary from the employee table.

**(SELECT d_id FROM employee WHERE salary = (...)):** The middle subquery retrieves the d_id from the employee table where the salary is equal to the maximum salary calculated in the innermost subquery.

**d_id = (...):** The outer query filters employees based on the condition that their d_id is equal to the department ID of employees with the maximum salary.

The query aims to select the first and last names (e_fname and e_lname) of employees who belong to the department of the employee with the highest salary in the company.

The result will be a list of employees in the same department as the employee with the maximum salary, including their first and last names.

**Note:**

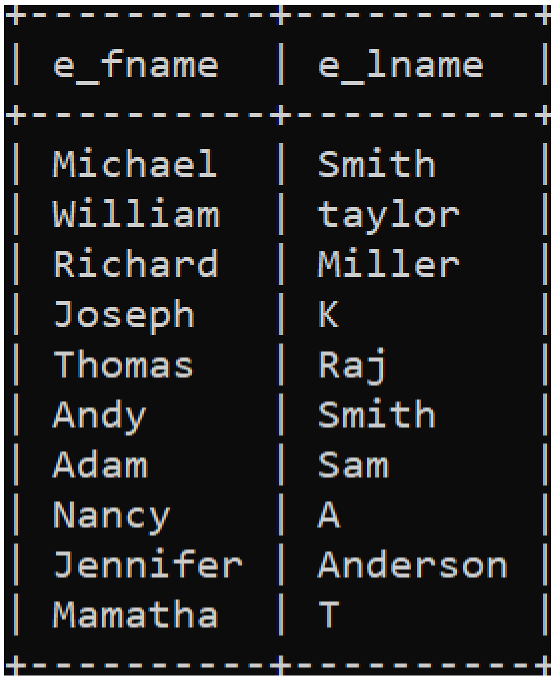
It assumes that there is at least one employee with a non-null salary in the employee table for the comparison to be meaningful.

4. **Write a query to display the employee who have a salary greater than any employee in department 21.**

**Solution:**

```
select e_fname, e_lname
from employee
where salary > any
(select salary
from employee
where d_id = 21);
```

## Output:



```
+-----------+----------+
| e_fname   | e_lname  |
+-----------+----------+
| Michael   | Smith    |
| William   | taylor   |
| Richard   | Miller   |
| Joseph    | K        |
| Thomas    | Raj      |
| Andy      | Smith    |
| Adam      | Sam      |
| Nancy     | A        |
| Jennifer  | Anderson |
| Mamatha   | T        |
+-----------+----------+
```

## Explanation:

**SELECT e_fname, e_lname:** The outer query selects the columns e_fname and e_lname from the employee table.

**WHERE salary > ANY (...):** This condition filters the rows from the employee table based on the value of salary being greater than any of the salaries returned by the subquery.

**(SELECT salary FROM employee WHERE d_id = 21):** The subquery retrieves salaries from the employee table for employees belonging to the department with d_id equal to 21.

**salary > ANY (...):** The outer query filters employees based on the condition that their salary is greater than any of the salaries in the subquery result.

The query aims to select the first and last names (e_fname and e_lname) of employees whose salary is higher than at least one salary in the department with d_id equal to 21.

The result will be a list of employees with salaries higher than any salary in the specified department, including their first and last names.

**Note:**

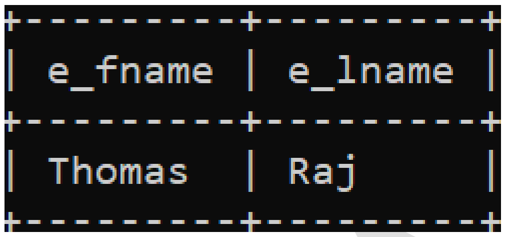
The use of ANY compares the left-hand side (salary) with any value on the right-hand side of the subquery result, allowing for a flexible condition based on the comparison with multiple values.

**5. Write a query to display all employees who have a more salary than all employees in department 24.**

## Solution:

```
select e_fname, e_lname
from employee
where salary > all
(select salary
from employee
where d_id = 24);
```

## Output:



```
+---------+---------+
| e_fname | e_lname |
+---------+---------+
| Thomas  | Raj     |
+---------+---------+
```

## Explanation:

**SELECT e_fname, e_lname:** The outer query selects the columns e_fname and e_lname from the employee table.

**WHERE salary > ALL (...):** This condition filters the rows from the employee table based on the value of salary being greater than all of the salaries returned by the subquery.

**(SELECT salary FROM employee WHERE d_id = 24):** The subquery retrieves salaries from the employee table for employees belonging to the department with d_id equal to 24.

**salary > ALL (...):** The outer query filters employees based on the condition that their salary is greater than all of the salaries in the subquery result.

The query aims to select the first and last names (e_fname and e_lname) of employees whose salary is higher than every salary in the department with d_id equal to 24.

The result will be a list of employees with salaries higher than every salary in the specified department, including their first and last names.

**Note:**

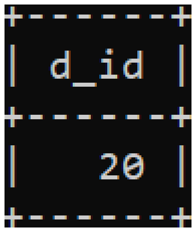
The use of ALL ensures that the condition is satisfied only if the specified condition (salary > ALL) holds true for every value in the subquery result.

## 6. Write a query to display the departments who don't have any employees working in it.

### Solution:

```
 select d_id
 from department
 where d_id not in
(select d_id
from employee
where d_id is not null);
```

### Output:



```
+------+
| d_id |
+------+
|   20 |
+------+
```

### Explanation:

**SELECT d_id:** The outer query selects the d_id column from the department table.

**WHERE d_id NOT IN (...):** This condition filters the rows from the department table based on the value of d_id not being present in the result of the subquery.

**(SELECT d_id FROM employee WHERE d_id IS NOT NULL):** The subquery retrieves distinct d_id values from the employee table where d_id is not null.

**d_id NOT IN (...):** The outer query filters departments based on the condition that their d_id is not present in the set of d_id values from the employee table.

The query aims to select the d_id values from the department table for departments where no employees have a null d_id (meaning every department has at least one employee).

The result will be a list of department IDs (d_id) where every department has at least one employee with a non-null department ID.

**Note:**

This type of query can be useful for finding departments with complete employee information or ensuring data integrity in scenarios where all departments should have associated employees.

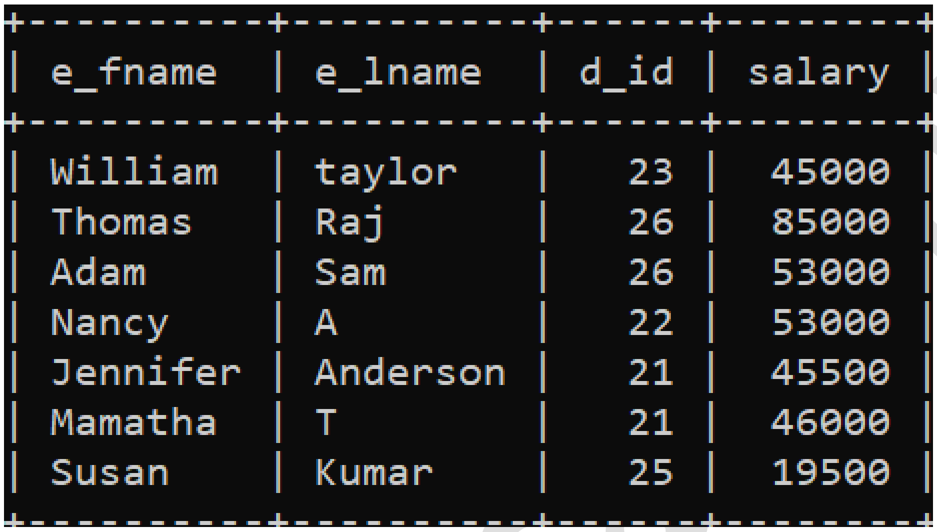
## 7. Write a query to display the employees who earn more than the average salary of their department.

**Solution:**

```
select e1.e_fname, e1.e_lname, e1.d_id, e1.salary
from employee as e1
where e1.salary >
```

```
(select avg(e2.salary)
from employee as e2
where e2.d_id = e1.d_id);
```

## Output:



```
+-----------+-----------+------+--------+
| e_fname   | e_lname   | d_id | salary |
+-----------+-----------+------+--------+
| William   | taylor    |   23 |  45000 |
| Thomas    | Raj       |   26 |  85000 |
| Adam      | Sam       |   26 |  53000 |
| Nancy     | A         |   22 |  53000 |
| Jennifer  | Anderson  |   21 |  45500 |
| Mamatha   | T         |   21 |  46000 |
| Susan     | Kumar     |   25 |  19500 |
+-----------+-----------+------+--------+
```

## Explanation:

**SELECT e1.e_fname, e1.e_lname, e1.d_id, e1.salary:** The outer query selects the first name (e_fname), last name (e_lname), department ID (d_id), and salary (salary) columns from the employee table, aliasing it as e1.

**WHERE e1.salary > (...):** This condition filters the rows from the aliased employee table (e1) based on the value of salary being greater than the average salary for the same department.

**(SELECT avg(e2.salary) FROM employee AS e2**

**WHERE e2.d_id = e1.d_id):** The subquery calculates the average salary (avg) for employees in the same department (d_id) as the employee from the outer query (e1).

**e2.d_id = e1.d_id**: The correlation occurs here, linking the outer query's employee (e1) with the employees in the subquery (e2) based on their shared department ID.

**e1.salary > (...):** The outer query filters employees based on the condition that their salary is greater than the average salary of employees in the same department.

The query aims to select employees whose salary is higher than the average salary of employees in the same department.

The result will be a list of employees with their first name, last name, department ID, and salary, meeting the specified condition.

**Note:**

The use of aliases (e1 and e2) clarifies which instance of the employee table is being referred to in each part of the query. The correlation ensures that the average salary is calculated per department.

# JOINS

In SQL, A JOIN operation is used to combine rows from two or more tables based on a related column between them. Joins are fundamental for retrieving data from multiple tables in a relational database.

Joins are based on a specified condition using the ON clause. This condition defines how the tables are related.

It's common to use table aliases to make SQL statements more readable when working with multiple tables in a query.

SQL joins are commonly used to retrieve data from related tables, such as combining customer data with order data or employee data with department data.

Remember that the choice of the join type depends on the specific requirements of your query and the relationships between the tables you are working with.

There are several types of JOINs, including:

## INNER JOIN:

It returns only the rows that have matching values in both the tables. It is the most commonly used join in MySQL.

### Syntax:

```
select column_name
from table1 inner join table2
```

```
on table1.column_name = table2.column_name;
```

## LEFT OUTER JOIN:

It returns only the rows that have matching values in both the tables and also non-matching values from the left table.

Syntax:

```
select column_name
from table1 left join table2
on table1.column_name = table2.column_name;
```

## NATURAL JOIN:

It returns only the rows that have matching values in both the tables and it will eliminate one of the identical column and one identical column which is left it will be displayed in first.

Syntax:

```
select column_name
from table1 natural join table2;
```

## RIGHT OUTER JOIN:

It returns the rows that have matching values in both the tables and also non-matching values from the right table.

### Syntax:

```
select column_name
from table1 right join table2
on table1.column_name = table2.column_name;
```

## FULL OUTER JOIN:

It returns only the rows that have matching values in both the tables and non-matching values from the left table and also non matching values from the right table.

### Syntax:

```
(select column_name
from table1 left  join table2
on table1.column_name = table2.column_name)
union
(select column_name
from table1 right  join table2
on table1.column_name = table2.column_name);
```

## CROSS JOIN:

It is used to combine every row from one table with every row from another table, resulting in a cartesian product. In other words, it returns all possible combinations of rows from both tables.

Syntax:

select column_name
from table1 cross join table2

## SELF JOIN:

It is a way to join a table to itself. It can be useful in situations where you need to compare rows within the same table.

Syntax:

```
select column_name
from table1 t1, table1 t2
where condition;
```

## Queries On Joins

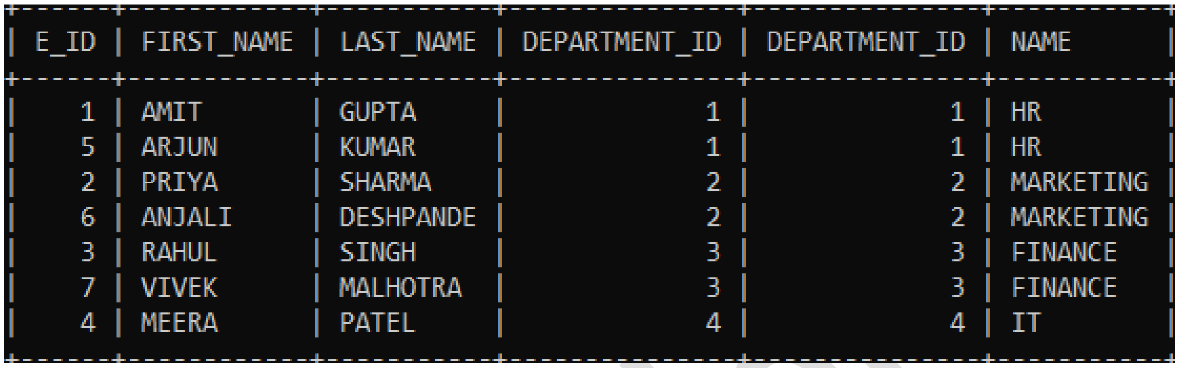
1. **Write a query to display all employees along with their department names, excluding employees without a department assigned.**

**Solution:**

select *
from employee e1 inner join department d1

on e1.department_id = d1.department_id;

**Output:**



```
+------+------------+-----------+---------------+---------------+-----------+
| E_ID | FIRST_NAME | LAST_NAME | DEPARTMENT_ID | DEPARTMENT_ID | NAME      |
+------+------------+-----------+---------------+---------------+-----------+
|    1 | AMIT       | GUPTA     |             1 |             1 | HR        |
|    5 | ARJUN      | KUMAR     |             1 |             1 | HR        |
|    2 | PRIYA      | SHARMA    |             2 |             2 | MARKETING |
|    6 | ANJALI     | DESHPANDE |             2 |             2 | MARKETING |
|    3 | RAHUL      | SINGH     |             3 |             3 | FINANCE   |
|    7 | VIVEK      | MALHOTRA  |             3 |             3 | FINANCE   |
|    4 | MEERA      | PATEL     |             4 |             4 | IT        |
+------+------------+-----------+---------------+---------------+-----------+
```

**Explanation:**

***SELECT :** This part of the query specifies that you want to select all columns from the result of the join operation.

**FROM employee e1:** This clause specifies that you are working with the "employee" table and aliasing it as "e1." The "e1" alias is used to refer to the "employee" table in the query.

**INNER JOIN department d1:** This clause is where the actual join operation occurs. It combines the "employee" table (aliased as "e1") with the "department" table, which is aliased as "d1." This means you are matching and combining rows from the "employee" and "department" tables based on a common column.

**ON e1.department_id** = d1.department_id: This part of the query specifies the join condition. It indicates that you want to join rows where the "department_id" column in the "employee" table (e1) matches the "department_id" column in the "department" table (d1). In other words, you're joining employees with their respective departments based on the department ID.

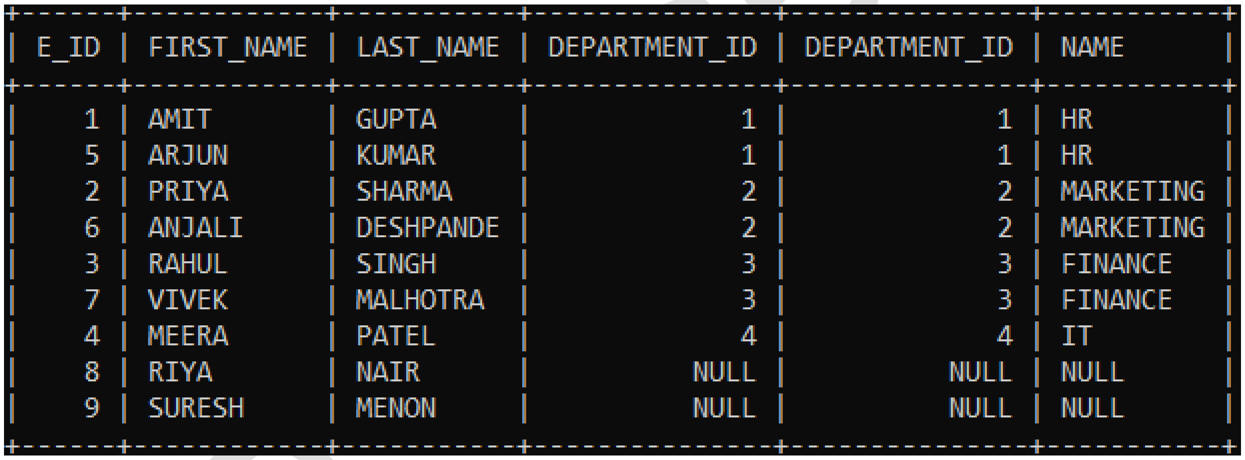
So, the result of this query will be a combined result set that includes all columns from both the "employee" and "department" tables where there is a match between the "department_id" in both tables. This allows you to retrieve information about employees and their associated departments in a single result set.

## 2. Write a query to display all employees along with their department names, including employees without a department assigned.

### Solution:

```
select*
from employee e1 left join department d1
on e1.department_id = d1.department_id;
```

### Output:



```
+------+------------+-----------+---------------+---------------+-----------+
| E_ID | FIRST_NAME | LAST_NAME | DEPARTMENT_ID | DEPARTMENT_ID | NAME      |
+------+------------+-----------+---------------+---------------+-----------+
|    1 | AMIT       | GUPTA     |             1 |             1 | HR        |
|    5 | ARJUN      | KUMAR     |             1 |             1 | HR        |
|    2 | PRIYA      | SHARMA    |             2 |             2 | MARKETING |
|    6 | ANJALI     | DESHPANDE |             2 |             2 | MARKETING |
|    3 | RAHUL      | SINGH     |             3 |             3 | FINANCE   |
|    7 | VIVEK      | MALHOTRA  |             3 |             3 | FINANCE   |
|    4 | MEERA      | PATEL     |             4 |             4 | IT        |
|    8 | RIYA       | NAIR      |          NULL |          NULL | NULL      |
|    9 | SURESH     | MENON     |          NULL |          NULL | NULL      |
+------+------------+-----------+---------------+---------------+-----------+
```

### Explanation:

***SELECT :** This part of the query specifies that you want to select all columns from the result of the join operation.

**FROM employee e1:** This clause specifies that you are working with the "employee" table and aliasing it as "e1." The "e1" alias is used to refer to the "employee" table in the query.

**LEFT JOIN department d1:** This clause is where the left join operation occurs. It combines the "employee" table (aliased as "e1") with the "department" table, which is aliased as "d1." This means you are matching and combining rows from the "employee" and "department" tables based on a common column, but in a left join, all rows from the left table are included in the result.

**ON e1.department_id = d1.department_id:** This part of the query specifies the join condition. It indicates that you want to join rows where the "department_id" column in the "employee" table (e1) matches the "department_id" column in the "department" table (d1). In other words, you're joining employees with their respective departments based on the department ID.

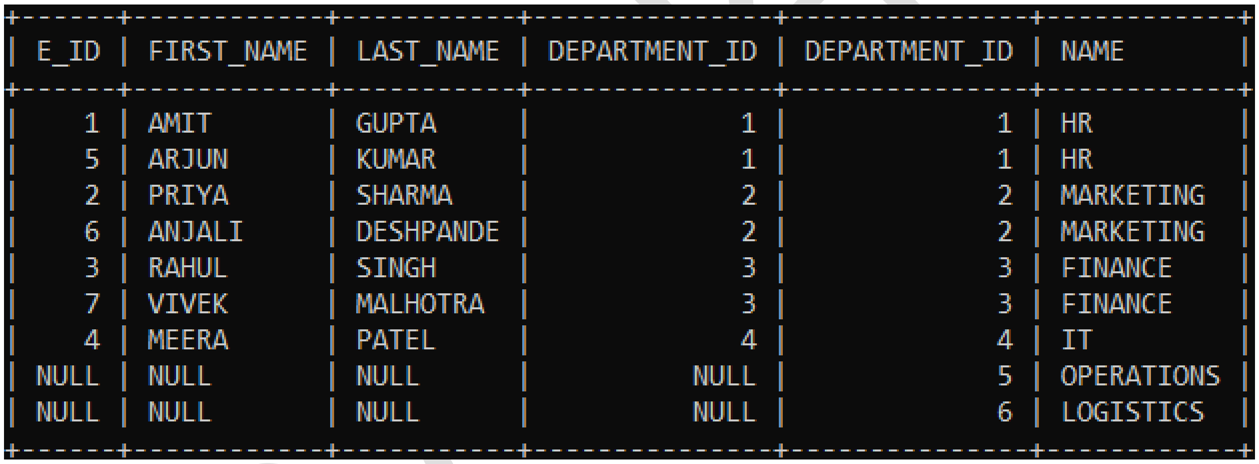
So, the result of this query will include all rows from the "employee" table (e1), and only the matching rows from the "department" table (d1) based on the "department_id" column. If there's no match for an employee's department in the "department" table, the result will still include the employee's information, with NULL values for the department columns from the "department" table. This allows you to retrieve a list of employees and their associated departments, and employees without departments will also be included in the result.

## 3. Write a query to display all employee along with their department names, including departments without employees

### Solution:

```
select *
from employee e1 right join department d1
on e1.department_id = d1.department_id;
```

### Output:



```
+------+------------+-----------+---------------+---------------+------------+
| E_ID | FIRST_NAME | LAST_NAME | DEPARTMENT_ID | DEPARTMENT_ID | NAME       |
+------+------------+-----------+---------------+---------------+------------+
|    1 | AMIT       | GUPTA     |             1 |             1 | HR         |
|    5 | ARJUN      | KUMAR     |             1 |             1 | HR         |
|    2 | PRIYA      | SHARMA    |             2 |             2 | MARKETING  |
|    6 | ANJALI     | DESHPANDE |             2 |             2 | MARKETING  |
|    3 | RAHUL      | SINGH     |             3 |             3 | FINANCE    |
|    7 | VIVEK      | MALHOTRA  |             3 |             3 | FINANCE    |
|    4 | MEERA      | PATEL     |             4 |             4 | IT         |
| NULL | NULL       | NULL      |          NULL |             5 | OPERATIONS |
| NULL | NULL       | NULL      |          NULL |             6 | LOGISTICS  |
+------+------------+-----------+---------------+---------------+------------+
```

### Explanation:

***SELECT :** This part of the query specifies that you want to select all columns from the result of the join operation.

**FROM employee e1:** This clause specifies that you are working with the "employee" table and aliasing it as "e1." The "e1" alias is used to refer to the "employee" table in the query.

**RIGHT JOIN department d1:** This clause is where the right join operation occurs. It combines the "employee" table (aliased as "e1") with the "department" table, which is aliased as "d1." In a right join, all rows from the right table are included in the result.

**ON e1.department_id = d1.department_id:** This part of the query specifies the join condition. It indicates that you want to join rows where the "department_id" column in the "employee" table (e1) matches the "department_id" column in the "department" table (d1). In other words, you're joining employees with their respective departments based on the department ID.

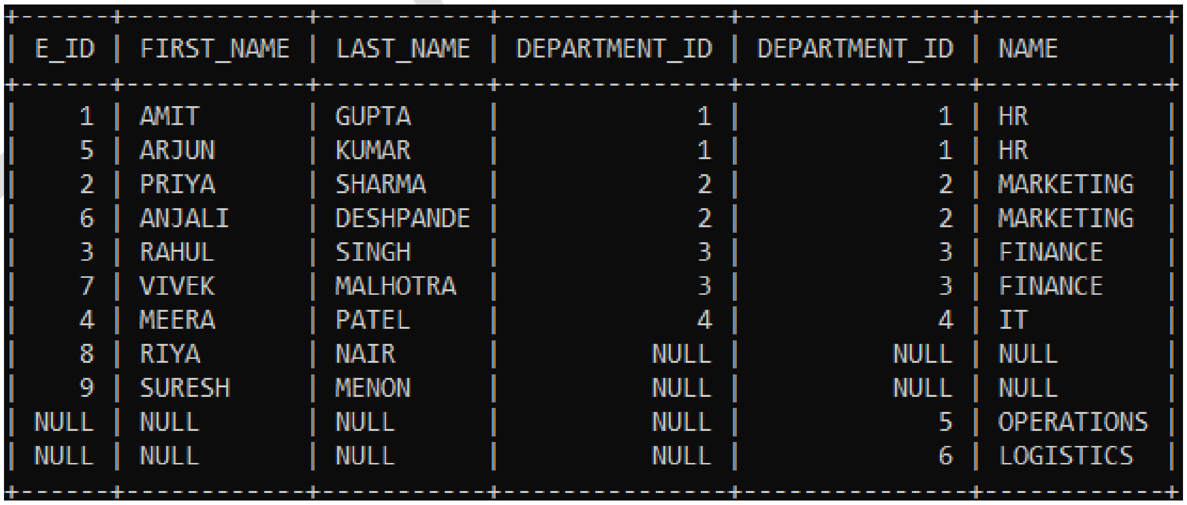
So, the result of this query will include all rows from the "department" table (d1), and only the matching rows from the "employee" table (e1) based on the "department_id" column. If there's no match for a department in the "employee" table, the result will still include the department's information, with NULL values for the employee columns from the "employee" table. This allows you to retrieve a list of departments and their associated employees, and departments without employees will also be included in the result.

## 4. Write a query to display all employees along with their department names, including employees without department assigned and departments without employees.

### Solution:

```
(select *
from employee e1 left join department d1
on e1.department_id = d1.department_id)
union
(select *
from employee e1 right join department d1
on e1.department_id = d1.department_id);
```

### Output:



```
+------+------------+-----------+---------------+---------------+------------+
| E_ID | FIRST_NAME | LAST_NAME | DEPARTMENT_ID | DEPARTMENT_ID | NAME       |
+------+------------+-----------+---------------+---------------+------------+
|    1 | AMIT       | GUPTA     |             1 |             1 | HR         |
|    5 | ARJUN      | KUMAR     |             1 |             1 | HR         |
|    2 | PRIYA      | SHARMA    |             2 |             2 | MARKETING  |
|    6 | ANJALI     | DESHPANDE |             2 |             2 | MARKETING  |
|    3 | RAHUL      | SINGH     |             3 |             3 | FINANCE    |
|    7 | VIVEK      | MALHOTRA  |             3 |             3 | FINANCE    |
|    4 | MEERA      | PATEL     |             4 |             4 | IT         |
|    8 | RIYA       | NAIR      |          NULL |          NULL | NULL       |
|    9 | SURESH     | MENON     |          NULL |          NULL | NULL       |
| NULL | NULL       | NULL      |          NULL |             5 | OPERATIONS |
| NULL | NULL       | NULL      |          NULL |             6 | LOGISTICS  |
+------+------------+-----------+---------------+---------------+------------+
```

## Explanation:

**(select * ...):** This part of the query starts with a subquery using parentheses. It's essentially performing a LEFT JOIN between the "employee" table (aliased as "e1") and the "department" table (aliased as "d1") based on the "department_id" column, and selects all columns with the alias "e1."

**UNION:** The UNION operator is used to combine the result sets of two separate queries.

**(select * ...):** This part is another subquery, but this time it's performing a RIGHT JOIN between the "employee" table (aliased as "e1") and the "department" table (aliased as "d1") based on the "department_id" column, and selects all columns with the alias "e1."

**Overall Logic:** The query first performs a LEFT JOIN and selects all matching rows from the "employee" table and associated department data (or NULL for employees with no matching department). Then, it performs a RIGHT JOIN, selecting all matching rows from the "department" table and associated employee data (or NULL for departments with no matching employees). The result sets of these two joins are combined using UNION.

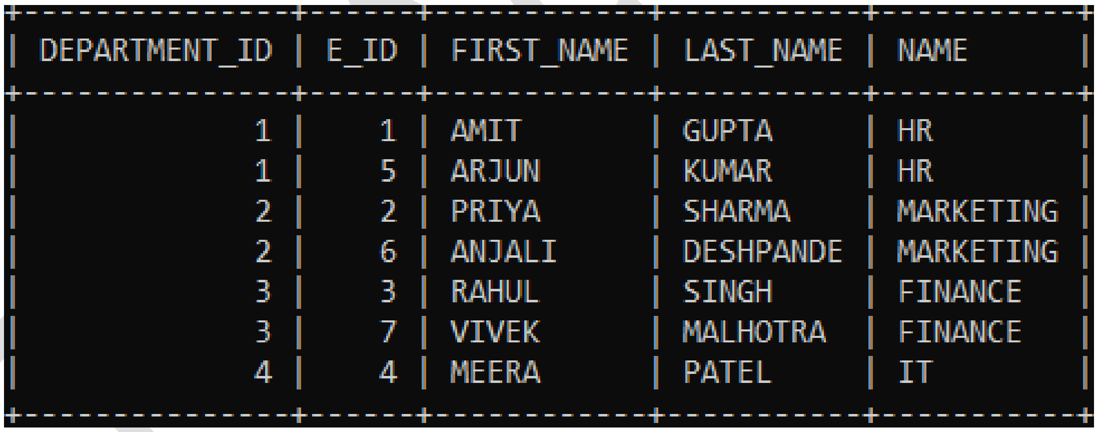
**Result:** The result of this query will include a combined result set that includes all rows from both the left and right joins. This effectively provides a list of employees and their associated departments, as well as a list of departments and their associated employees. If there are departments without employees or employees without departments, those records will still be included in the result with NULL values in the respective columns.

**Note:** It's important to ensure that the columns selected in both subqueries have the same structure (i.e., the same number and order of columns) for the UNION operation to work correctly

## 5. NATURAL JOIN QUERY

```
select *
from employee e1 natural join department d1;
```

**Output:**



```
+---------------+------+------------+-----------+-----------+
| DEPARTMENT_ID | E_ID | FIRST_NAME | LAST_NAME | NAME      |
+---------------+------+------------+-----------+-----------+
|             1 |    1 | AMIT       | GUPTA     | HR        |
|             1 |    5 | ARJUN      | KUMAR     | HR        |
|             2 |    2 | PRIYA      | SHARMA    | MARKETING |
|             2 |    6 | ANJALI     | DESHPANDE | MARKETING |
|             3 |    3 | RAHUL      | SINGH     | FINANCE   |
|             3 |    7 | VIVEK      | MALHOTRA  | FINANCE   |
|             4 |    4 | MEERA      | PATEL     | IT        |
+---------------+------+------------+-----------+-----------+
```

**Explanation:**

***SELECT :** This part of the query specifies that you want to select all columns from the result of the join operation.

**FROM employee e1:** This clause specifies that you are working with the "employee" table and aliasing it as "e1." The "e1" alias is used to refer to the "employee" table in the query.

**NATURAL JOIN department d1:** This clause is where the NATURAL JOIN operation occurs. A NATURAL JOIN is a type of JOIN that automatically joins tables based on columns with the same name and data type.

**Overall Logic:** The NATURAL JOIN automatically identifies and joins rows from the "employee" and "department" tables where columns with the same name and data type match.

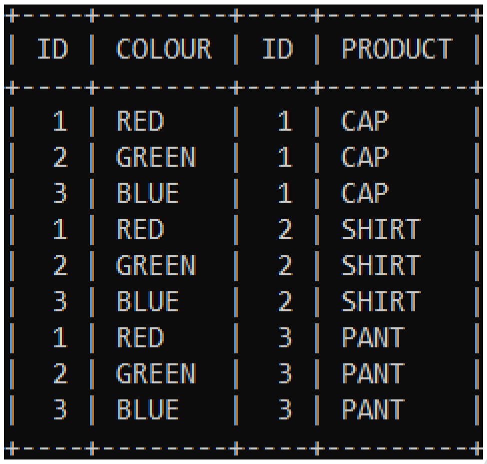
**Result:** The result of this query will include all columns from both the "employee" and "department" tables, where there are matching column names and data types. It effectively joins rows based on the common columns without the need for explicitly specifying the join condition.

**Note:** NATURAL JOINs can be convenient when you have tables with columns that have the same name and data type and you want to join them based on that commonality. However, they can be less explicit than other join types, so it's important to ensure that the common columns have the same meaning in both tables.

## 6. CROSS JOIN QUERY

```
select *
from colour cross join product;
```

### Output:



```
+----+--------+----+---------+
| ID | COLOUR | ID | PRODUCT |
+----+--------+----+---------+
|  1 | RED    |  1 | CAP     |
|  2 | GREEN  |  1 | CAP     |
|  3 | BLUE   |  1 | CAP     |
|  1 | RED    |  2 | SHIRT   |
|  2 | GREEN  |  2 | SHIRT   |
|  3 | BLUE   |  2 | SHIRT   |
|  1 | RED    |  3 | PANT    |
|  2 | GREEN  |  3 | PANT    |
|  3 | BLUE   |  3 | PANT    |
+----+--------+----+---------+
```

### Explanation:

***SELECT :** This part of the query specifies that you want to select all columns from the result of the join operation.

**FROM colour:** This clause specifies that you are working with the "colour" table.

**CROSS JOIN product:** The CROSS JOIN is used to combine every row from the "colour" table with every row from the "product" table, creating a Cartesian product of the two tables. There is no specific join condition, so it pairs each row from the "colour" table with every row from the "product" table.

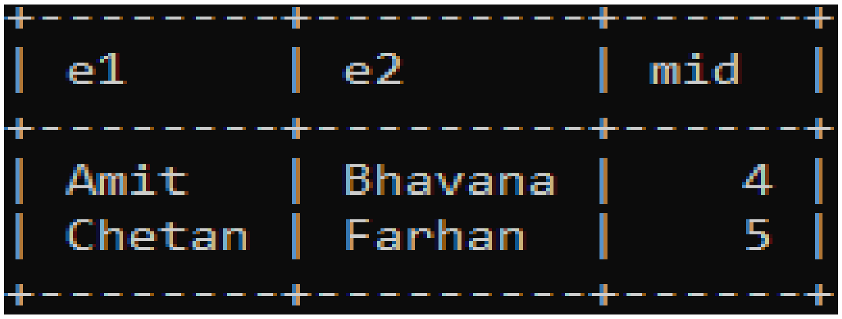
So, the result of this query will include every possible combination of rows from the "colour" and "product" tables. It essentially results in a new table where each row represents a combination of a color and a product, and the number of rows in the result will be the product of the number of rows in the "colour" table and the number of rows in the "product" table. This can lead to a large result set if both tables have many rows.

## 7. Write a query to find such employees who has the same manager.

### Solution:

```
select T1.name e1, T2.name e2, T1.mid
from employee_manager T1, employee_manager T2
where T1.eid < T2.eid and T1.mid = T2.mid;
```

### Output:



```
+----------+-----------+-------+
| e1       | e2        | mid   |
+----------+-----------+-------+
| Amit     | Bhavana   |     4 |
| Chetan   | Farhan    |     5 |
+----------+-----------+-------+
```

## Explanation:

SELECT T1.name e1, T2.name e2, T1.mid: This part of the query specifies the columns you want to select in the result. It aliases the "name" column from the first instance of the "employee_manager" table as "e1," aliases the "name" column from the second instance of the "employee_manager" table as "e2," and selects the "mid" column from the first instance, which is also aliased as "mid."

FROM employee_manager T1, employee_manager T2: This clause indicates that you are working with two instances of the "employee_manager" table, aliased as "T1" and "T2." This is effectively creating a self-join, allowing you to compare different rows from the same table.

WHERE T1.eid < T2.eid and T1.mid = T2.mid: The WHERE clause specifies the conditions for the join. It filters the rows to only include pairs of rows where the "eid" (employee ID) from the first instance is less than the "eid" from the second instance, and where the "mid" (manager ID) in both instances is the same. In other words, it selects pairs of employees (e1 and e2) who have the same manager (mid), but where e1 has a lower employee ID (eid) than e2.

So, the result of this query will include pairs of employees (e1 and e2) who share the same manager (mid), but it only selects those pairs where e1 has a lower employee ID (eid) than e2. This is useful for identifying employees who report to the same manager but have different seniority levels within the organization.