**Some Useful Links:**
SQL Practical question
https://www.edureka.co/blog/interview-questions/sql-query-interview-questions
SQL Theory question
https://www.edureka.co/blog/interview-questions/sql-interview-questions

1. **Write a SQL query to retrieve the count of orders for each customer from tables named "Customers" and "Orders," joining them based on the customer ID**

```
SELECT c.CustomerID, c.Name, COUNT(o.OrderID) AS OrderCount
FROM Customers c LEFT JOIN Orders o
ON c.CustomerID = o.CustomerID
GROUP BY c.CustomerID, c.Name;
```

2. **Write a SQL query to update the email address of a customer with a specific ID in a table named "Customers."**

```
UPDATE Customers
SET Email = 'newemail@example.com'
WHERE CustomerID = 123;
```

3. **Write a SQL query to delete all records from a table named "Orders" that have a status of "Cancelled."**

```
DELETE FROM Orders
WHERE Status = 'Cancelled';
```

4. **Given a table called "Employees" with columns "EmployeeID" and "Salary," write an SQL query to retrieve the top 5 employees with the highest salaries.**

```
SELECT EmployeeID, Salary FROM Employees
ORDER BY Salary DESC
LIMIT 5;
```

5. **Consider two tables, "Customers" and "Orders," with columns "CustomerID" in both tables. Write an**

```
SELECT DISTINCT a.CustomerID, a.CustomerName
FROM Customers a INNER JOIN Orders b
ON a.CustomerID = b.CustomerID;
```

6. **Given a table called "Products" with columns "ProductID" and "Quantity," write an SQL query to calculate the total quantity of all products.**

```
SELECT SUM(Quantity) AS TotalQuantity FROM Products;
```

7. **Given a table called "Orders" with columns "OrderID," "OrderDate," and "TotalAmount," write an SQL query to calculate the average total amount of orders placed per month.**

```
SELECT EXTRACT(MONTH FROM OrderDate) AS Month, AVG(TotalAmount) AS AverageAmount
FROM Orders
GROUP BY EXTRACT(MONTH FROM OrderDate);
```

8. **Write a SQL query to calculate the average salary of employees in each department from the "Employees" table.**

```
SELECT Department, AVG(Salary) AS AverageSalary FROM Employees GROUP BY Department;
```

9. **Write a SQL query to retrieve the names of customers who have placed orders in the "Orders" table**

```
SELECT DISTINCT CustomerName FROM Orders;
```

## EmployeeInfo Table:

| EmpID | EmpFname | EmpLname | Department | Project | Address | DOB | Gender |
|---|---|---|---|---|---|---|---|
| 1 | Sanjay | Mehra | HR | P1 | Hyderabad(HYD) | 01/12/1976 | M |
| 2 | Ananya | Mishra | Admin | P2 | Delhi(DEL) | 02/05/1968 | F |
| 3 | Rohan | Diwan | Account | P3 | Mumbai(BOM) | 01/01/1980 | M |
| 4 | Sonia | Kulkarni | HR | P1 | Hyderabad(HYD) | 02/05/1992 | F |
| 5 | Ankit | Kapoor | Admin | P2 | Delhi(DEL) | 03/07/1994 | M |

**EmployeePosition Table:**

| EmpID | EmpPosition | DateOfJoining | Salary |
|---|---|---|---|
| 1 | Manager | 01/05/2022 | 500000 |
| 2 | Executive | 02/05/2022 | 75000 |
| 3 | Manager | 01/05/2022 | 90000 |
| 2 | Lead | 02/05/2022 | 85000 |
| 1 | Executive | 01/05/2022 | 300000 |

**10. Write a query to fetch the EmpFname from the EmployeeInfo table in the upper case and use the ALIAS name as EmpName.**

```
SELECT UPPER(EmpFname) AS EmpName FROM EmployeeInfo;
```

**11. Write a query to fetch the number of employees working in the department 'HR'.**

```
SELECT COUNT(*) FROM EmployeeInfo WHERE Department = 'HR'
```

**12. Write a query to get the current date.**

You can write a query as follows in SQL Server:

```
SELECT GETDATE();
```

You can write a query as follows in MYSQL:

```
SELECT SYSTDATE();
```

**13. Write a query to retrieve the first four characters of EmpLname from the EmployeeInfo table.**

```
SELECT SUBSTRING(EmpLname, 1, 4) FROM EmployeeInfo;
SELECT LEFT(EmpLname,4) FROM EmployeeInfo;
```

**14. Write a query to fetch only the place name(string before brackets) from the Address column of EmployeeInfo table.**

**[Note:** The LOCATE() function returns the position of the first occurrence of a substring in a string SELECT LOCATE("3", "W3Schools.com") AS MatchPosition;**]**

**Using the MID function in MySQL :**

```
SELECT MID(Address, 0, LOCATE('(',Address)) FROM EmployeeInfo;
```

**Using SUBSTRING:**

```
SELECT SUBSTRING(Address, 1, CHARINDEX('(',Address)) FROM EmployeeInfo;
```

**15. Write a query to create a new table that consists of data and structure copied from the other table.**

Using the SELECT INTO command:

```
SELECT * INTO NewTable FROM EmployeeInfo WHERE 1 = 0;
```

Using the CREATE command in MySQL:

```
CREATE TABLE NewTable
AS
SELECT * FROM EmployeeInfo;
```

**16. Write q query to find all the employees whose salary is between 50000 to 100000.**

```
SELECT * FROM EmployeePosition WHERE Salary BETWEEN '50000' AND '100000';
```

**17. Write a query to find the names of employees that begin with 'S'**

```
SELECT * FROM EmployeeInfo WHERE EmpFname LIKE 'S%';
```

**18. Write a query to fetch top N records.**

By using the TOP command in SQL Server:

```
SELECT TOP N * FROM EmployeePosition ORDER BY Salary DESC;
```

By using the LIMIT command in MySQL:

```
SELECT * FROM EmpPosition ORDER BY Salary DESC LIMIT N;
```

**19. Write a query to retrieve the EmpFname and EmpLname in a single column as "FullName". The first name and the last name must be separated with space.**

```
SELECT CONCAT(EmpFname, ' ', EmpLname) AS 'FullName' FROM EmployeeInfo;
```

**20. Write a query find number of employees whose DOB is between 02/05/1970 to 31/12/1975 and are grouped according to gender**

```
SELECT COUNT(*), Gender FROM EmployeeInfo WHERE DOB BETWEEN '02/05/1970 '
AND '31/12/1975' GROUP BY Gender;
```

**21. Write a query to fetch all the records from the EmployeeInfo table ordered by EmpLname in descending order and Department in the ascending order.**

```
SELECT * FROM EmployeeInfo ORDER BY EmpFname desc, Department asc;
```

**22. Write a query to fetch details of employees whose EmpLname ends with an alphabet 'A' and contains five alphabets.**

```
SELECT * FROM EmployeeInfo WHERE EmpLname LIKE '____a';
```

**23. Write a query to fetch details of all employees excluding the employees with first names, "Sanjay" and "Sonia" from the EmployeeInfo table.**

```
SELECT * FROM EmployeeInfo WHERE EmpFname NOT IN ('Sanjay','Sonia');
```

**24. Write a query to fetch details of employees with the address as "DELHI(DEL)".**

```
SELECT * FROM EmployeeInfo WHERE Address LIKE 'DELHI(DEL)%';
```

**25. Write a query to fetch all employees who also hold the managerial position.**

```
SELECT E.EmpFname, E.EmpLname, P.EmpPosition
FROM EmployeeInfo E INNER JOIN EmployeePosition P
ON E.EmpID = P.EmpID AND P.EmpPosition IN ('Manager');

SELECT E.EmpFname, E.EmpLname, P.EmpPosition
FROM EmployeeInfo E INNER JOIN EmployeePosition P
ON E.EmpID = P.EmpID
where P.EmpPosition ='Manager';
```

**26. Write a query to fetch the department-wise count of employees sorted by department's count in ascending order.**

```
SELECT Department, count(EmpID) AS EmpDeptCount FROM EmployeeInfo
GROUP BY Department
ORDER BY EmpDeptCount ASC;
```

**30. Write a query to calculate the even and odd records from a table.**

To retrieve the even records from a table, you have to use the MOD() function as follows:

```
SELECT EmpID FROM (SELECT rowno, EmpID from EmployeeInfo)
WHERE MOD(rowno,2)=0;
```

Similarly, to retrieve the odd records from a table, you can write a query as follows:

```
SELECT EmpID FROM (SELECT rowno, EmpID from EmployeeInfo)
WHERE MOD(rowno,2)=1;
```

**31. Write a SQL query to retrieve employee details from EmployeeInfo table who have a date of joining in the EmployeePosition table.**

```
SELECT EmployeeInfo.*, EmployeePosition.DateOfJoining
FROM EmployeeInfo INNER JOIN EmployeePosition
```

```
ON EmployeeInfo.EmployeeID = EmployeePosition.EmployeeID

SELECT * FROM EmployeeInfo E
WHERE EXISTS
(SELECT * FROM EmployeePosition P WHERE E.EmpId = P.EmpId);
```

**32. Write a query to retrieve two minimum and maximum salaries from the EmployeePosition table.**

```
Select salary from employeeposition
Order by salary desc limit 2;
Select salary from employeeposition
Order by salary asc limit 2;
```

To retrieve two minimum salaries, you can write a query as below:

```
SELECT DISTINCT Salary FROM EmployeePosition E1
 WHERE 2 >= (SELECT COUNT(DISTINCT Salary)FROM EmployeePosition E2
  WHERE E1.Salary >= E2.Salary) ORDER BY E1.Salary DESC;
```

(Certainly! Let's break down the query step by step:

1. The outer query selects the distinct salaries from the "EmployeePosition" table: `SELECT DISTINCT Salary FROM EmployeePosition E1`.
2. The inner query `(SELECT COUNT(DISTINCT Salary) FROM EmployeePosition E2 WHERE E1.Salary >= E2.Salary)` calculates the count of distinct salaries that are less than or equal to each salary in the outer query.
3. The condition `2 >=` is used to filter the results. It ensures that only salaries with a count of distinct salaries less than or equal to 2 are included in the result set.
4. Finally, `ORDER BY E1.Salary DESC` sorts the result set in descending order based on the salary.

In essence, the query is retrieving distinct salaries from the "EmployeePosition" table that have a count of distinct salaries less than or equal to 2. It then orders them in descending order based on the salary. This can be useful to find the two highest salaries in the table or to retrieve salaries that occur less frequently.

)

To retrieve two maximum salaries, you can write a query as below:

```
SELECT DISTINCT Salary FROM EmployeePosition E1
 WHERE 2 >= (SELECTCOUNT(DISTINCT Salary) FROM EmployeePosition E2
  WHERE E1.Salary <= E2.Salary) ORDER BY E1.Salary DESC;
```

**33. Write a query to find the Nth highest salary from the table without using TOP/limit keyword.**

```
SELECT Salary FROM (SELECT Salary, RANK() OVER (ORDER BY Salary DESC) AS SalaryRank
 FROM Employeeposition) AS RankedSalaries
WHERE SalaryRank = N

SELECT Salary
FROM EmployeePosition E1
WHERE N-1 = (
      SELECT COUNT( DISTINCT ( E2.Salary ) )
      FROM EmployeePosition E2
      WHERE E2.Salary >  E1.Salary );
```

**34. Write a query to retrieve duplicate records from a table.**

```
SELECT EmpID, EmpFname, Department, COUNT(*)
FROM EmployeeInfo GROUP BY EmpID, EmpFname, Department
HAVING COUNT(*) > 1;
```

**35. Write a query to retrieve the list of employees working in the same department.**

```
Select DISTINCT E.EmpID, E.EmpFname, E.Department
FROM EmployeeInfo E, Employee E1
WHERE E.Department = E1.Department AND E.EmpID != E1.EmpID;

SELECT e1.EmployeeID, e1.EmployeeName, e1.Department
FROM Employees e1 JOIN Employees e2
ON e1.Department = e2.Department
WHERE e1.EmployeeID <> e2.EmployeeID
```

**36. Write a query to retrieve the last 3 records from the EmployeeInfo table.**

Using LIMIT (for databases like MySQL, PostgreSQL, SQLite):

```
SELECT * FROM EmployeeInfo
ORDER BY EmployeeID DESC
LIMIT 3;
```

Using TOP (for databases like SQL Server, Microsoft Access):

```
SELECT TOP 3 * FROM EmployeeInfo
ORDER BY EmployeeID DESC;
```

```
SELECT * FROM EmployeeInfo WHERE
EmpID <=3 UNION SELECT * FROM
(SELECT * FROM EmployeeInfo E ORDER BY E.EmpID DESC)
AS E1 WHERE E1.EmpID <=3;
```

1. The first part of the query, `SELECT * FROM EmployeeInfo WHERE EmpID <= 3`, retrieves the records from the "EmployeeInfo" table where the EmpID is less than or equal to 3. This part retrieves the first 3 records based on the EmpID column.
2. The UNION operator is used to combine the result set from the first query with the result set from the second query.
3. The second part of the query, `SELECT * FROM (SELECT * FROM EmployeeInfo E ORDER BY E.EmpID DESC) AS E1 WHERE E1.EmpID <= 3`, is a subquery that retrieves all the records from the "EmployeeInfo" table and sorts them in descending order based on the EmpID column using the ORDER BY clause. The subquery is aliased as E1.
4. The outer query then selects the records from the subquery (E1) where the EmpID is less than or equal to 3. This part retrieves the last 3 records based on the EmpID column.

By using the UNION operator, the query combines the results of both parts to retrieve a total of 6 records: the first 3 records (EmpID <= 3) and the last 3 records (sorted by EmpID in descending order and EmpID <= 3).

**37. Write a query to find the third-highest salary from the EmpPosition table.**

```
SELECT TOP 1 salary
FROM(
SELECT TOP 3 salary
FROM employee_table
ORDER BY salary DESC) AS emp
ORDER BY salary ASC;
```

```
SELECT Salary FROM (SELECT Salary, RANK() OVER (ORDER BY Salary DESC) AS SalaryRank
 FROM Employeeposition) AS RankedSalaries
WHERE SalaryRank = N
```

**44. Write a query to display the first and the last record from the EmployeeInfo table**

To display the first record from the EmployeeInfo table, you can write a query as follows:

```
SELECT * FROM EmployeeInfo WHERE EmpID = (SELECT MIN(EmpID) FROM
EmployeeInfo);
```

To display the last record from the EmployeeInfo table, you can write a query as follows:

```
SELECT * FROM EmployeeInfo WHERE EmpID = (SELECT MAX(EmpID) FROM
EmployeeInfo);
```

**45. Write a query to add email validation to your database**

```
SELECT Email FROM EmployeeInfo WHERE NOT REGEXP_LIKE(Email, '[A-Z0-9._%+-
]+@[A-Z0-9.-]+.[A-Z]{2,4}', 'i');
```

**46. Write a query to retrieve Departments who have less than 2 employees working in it.**

```
SELECT DEPARTMENT, COUNT(EmpID) as 'EmpNo' FROM EmployeeInfo
GROUP BY DEPARTMENT
HAVING COUNT(EmpD) < 2;
```

**47. Write a query to retrieve EmpPostion along with total salaries paid for each of them**

```
SELECT EmpPosition, SUM(Salary) from EmployeePosition
GROUP BY EmpPosition;
```

**48. Write a query to fetch 50% records from the EmployeeInfo table.**

```
SELECT * FROM EmployeeInfo
WHERE EmpID <= (SELECT COUNT(EmpID)/2 from EmployeeInfo);
```

**49. Write the SQL query to get the third maximum salary of an employee from a table named employees.**

Employee table

| employee_name | salary |
|---|---|
| A | 24000 |
| C | 34000 |
| D | 55000 |
| E | 75000 |
| F | 21000 |
| G | 40000 |
| H | 50000 |

```
        SELECT * FROM(
        SELECT employee_name, salary, DENSE_RANK()
        OVER(ORDER BY salary DESC) as rank FROM Employee)
50. WHERE rank=&n;
        To find 3rd highest salary set n = 3
```

**51. How to remove duplicate rows in SQL?**

If the SQL table has duplicate rows, the duplicate rows must be removed. Let's assume the following table as our dataset:

| ID | Name | Age |
|---|---|---|
| 1 | A | 21 |
| 2 | B | 23 |
| 2 | B | 23 |
| 4 | D | 22 |
| 5 | E | 25 |
| 6 | G | 26 |
| 5 | E | 25 |

The following SQL query removes the duplicate ids from the table:

```
DELETE FROM table WHERE ID IN (SELECT ID, COUNT(ID) FROM table
GROUP BY ID
HAVING COUNT (ID) > 1);
```

**51. How to select UNIQUE records from a table using a SQL Query?**

**Consider below EMPLOYEE table as the source data**

```
CREATE TABLE EMPLOYEE (
   EMPLOYEE_ID NUMBER(6,0),
   NAME VARCHAR2(20),
   SALARY NUMBER(8,2)
);

INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(100,'Jennifer',4400);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(100,'Jennifer',4400);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(101,'Michael',13000);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(101,'Michael',13000);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(101,'Michael',13000);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(102,'Pat',6000);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(102,'Pat',6000);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(103,'Den',11000);

SELECT * FROM EMPLOYEE;
```

| EMPLOYEE_ID | NAME | SALARY |
|---|---|---|
| 100 | Jennifer | 4400 |
| 100 | Jennifer | 4400 |
| 101 | Michael | 13000 |
| 101 | Michael | 13000 |
| 101 | Michael | 13000 |
| 102 | Pat | 6000 |
| 102 | Pat | 6000 |
| 103 | Den | 11000 |

# METHOD-1: Using GROUP BY Function

**GROUP BY clause is used with SELECT statement to collect data from multiple records and group the results by one or more columns. The GROUP BY clause returns one row per group. By applying GROUP BY function on all the source columns, unique records can be queried from the table.**

Below is the query to fetch the unique records using GROUP BY function.

**Query:**

```
SELECT
  EMPLOYEE_ID,
  NAME,
  SALARY
FROM EMPLOYEE
GROUP BY EMPLOYEE_ID, NAME, SALARY;
```

**Result:**

| EMPLOYEE_ID | NAME | SALARY |
|---|---|---|

| | | |
|---|---|---|
| 100 | Jennifer | 4400 |
| 101 | Michael | 13000 |
| 102 | Pat | 6000 |
| 103 | Den | 11000 |

# METHOD-2: Using ROW_NUMBER Analytic Function

**The ROW_NUMBER Analytic function is used to provide consecutive numbering of the rows in the result by the ORDER selected for each PARTITION specified in the OVER clause. It will assign the value 1 for the first row and increase the number of the subsequent rows.**

Using ROW_NUMBER Analytic function, assign row numbers to each unique set of records.

**Query:**

```
SELECT
  EMPLOYEE_ID,
  NAME,
  SALARY,
  ROW_NUMBER() OVER(PARTITION BY EMPLOYEE_ID,NAME,SALARY ORDER BY EMPLOYEE_ID) AS
ROW_NUMBER
FROM EMPLOYEE;
```

**Result:**

| EMPLOYEE_ID | NAME | SALARY | ROW_NUMBER |
|---|---|---|---|
| 100 | Jennifer | 4400 | 1 |
| 100 | Jennifer | 4400 | 2 |
| 101 | Michael | 13000 | 1 |
| 101 | Michael | 13000 | 2 |
| 101 | Michael | 13000 | 3 |

| | | | |
|---|---|---|---|
| 102 | Pat | 6000 | 1 |
| 102 | Pat | 6000 | 2 |
| 103 | Den | 11000 | 1 |

Once row numbers are assigned, by querying the rows with row number 1 will give the unique records from the table.

**Query:**

```
SELECT EMPLOYEE_ID, NAME, SALARY FROM(
  SELECT
    EMPLOYEE_ID,
    NAME,
    SALARY,
    ROW_NUMBER() OVER(PARTITION BY EMPLOYEE_ID,NAME,SALARY ORDER BY EMPLOYEE_ID) AS
ROW_NUMBER
  FROM EMPLOYEE)
WHERE ROW_NUMBER = 1;
```

**Result:**

| EMPLOYEE_ID | NAME | SALARY |
|---|---|---|
| 101 | Michael | 13000 |
| 100 | Jennifer | 4400 |
| 102 | Pat | 6000 |
| 103 | Den | 11000 |

**Related Article:** [SQL Analytic Functions Interview Questions]

# 5. What is the result of Normal Join, Left Outer Join, Right Outer Join and Full Outer Join between the tables A & B?

## Table_A

| COL |
| --- |
| 1 |
| 1 |
| 0 |
| null |

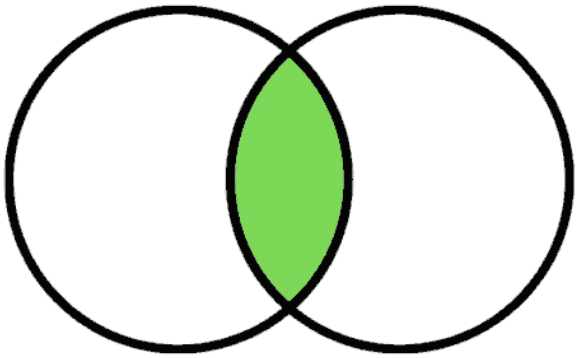
## Table_B

| COL |
| --- |
| 1 |
| 0 |
| null |
| null |

## Normal Join:

**Normal Join or Inner Join is the most common type of join. It returns the rows that are exact match between both the tables.**

The following Venn diagram illustrates a Normal join when combining two result sets:



**Query:**

```
SELECT
   a.COL AS A,
   b.COL AS B
FROM TABLE_A a JOIN TABLE_B b
ON a.COL = b.COL;
```

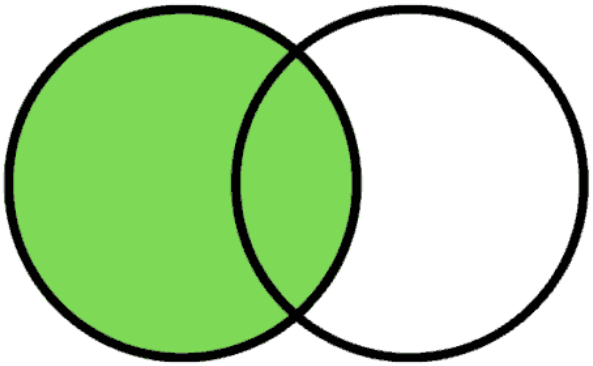
**Result:**

| A | B |
|---|---|
| 1 | 1 |
| 1 | 1 |
| 0 | 0 |

# Left Outer Join:

**The Left Outer Join returns all the rows from the left table and only the matching rows from the right table. If there is no matching row found from the right table, the left outer join will have NULL values for the columns from right table.**

The following Venn diagram illustrates a Left join when combining two result sets:



**Query:**

```
SELECT
   a.COL AS A,
   b.COL AS B
FROM TABLE_A a LEFT OUTER JOIN TABLE_B b
ON a.COL = b.COL;
```

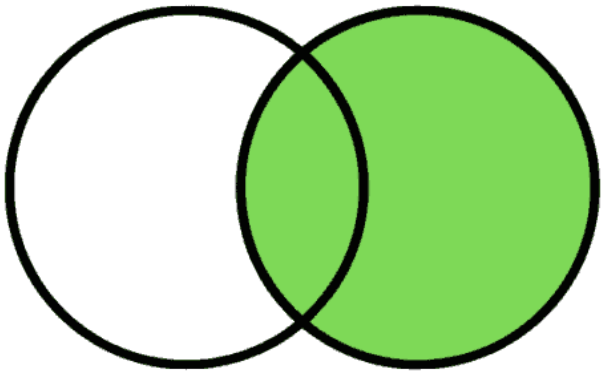
**Result:**

| A | B |
|---|---|
| 1 | 1 |
| 1 | 1 |
| 0 | 0 |
| NULL | NULL |

# Right Outer Join:

**The Right Outer Join returns all the rows from the right table and only the matching rows from the left table. If there is no matching row found from the left table, the right outer join will have NULL values for the columns from left table.**

The following Venn diagram illustrates a Right join when combining two result sets:



**Query:**

```
SELECT
   a.COL AS A,
   b.COL AS B
FROM TABLE_A a RIGHT OUTER JOIN TABLE_B b
ON a.COL = b.COL;
```

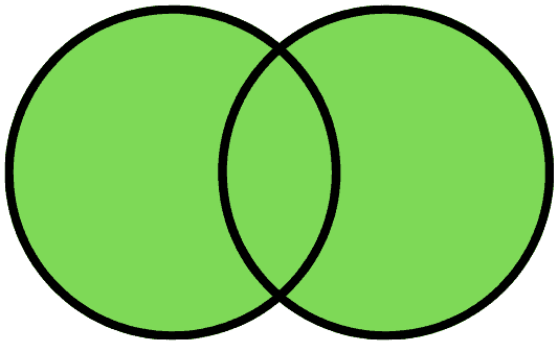
**Result:**

| A | B |
|---|---|
| 1 | 1 |
| 1 | 1 |
| 0 | 0 |
| NULL | NULL |
| NULL | NULL |

# Full Outer Join:

**The Full Outer Join returns all the rows from both the right table and the left table. If there is no matching row found, the missing side columns will have NULL values.**

The following Venn diagram illustrates a Full join when combining two result sets:



**Query:**

```
SELECT
   a.COL AS A,
   b.COL AS B
FROM TABLE_A a FULL OUTER JOIN TABLE_B b
ON a.COL = b.COL;
```

**Result:**

| A | B |
|---|---|
| 1 | 1 |
| 1 | 1 |
| 0 | 0 |
| NULL | NULL |
| NULL | NULL |
| NULL | NULL |

***NOTE:*** *NULL do not match with NULL*

# 6. How to find the employee with second MAX Salary using a SQL query?

**Consider below EMPLOYEES table as the source data**

```
CREATE TABLE Employees(
   EMPLOYEE_ID NUMBER(6,0),
   NAME VARCHAR2(20 BYTE),
   SALARY NUMBER(8,2)
);

INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(100,'Jennifer',4400);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(101,'Michael',13000);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(102,'Pat',6000);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(103,'Den', 11000);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(104,'Alexander',3100);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(105,'Shelli',2900);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(106,'Sigal',2800);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(107,'Guy',2600);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(108,'Karen',2500);

SELECT * FROM Employees;
```

| EMPLOYEE_ID | NAME | SALARY |
|---|---|---|
| 100 | Jennifer | 4400 |
| 101 | Michael | 13000 |
| 102 | Pat | 6000 |
| 103 | Den | 11000 |
| 104 | Alexander | 3100 |
| 105 | Shelli | 2900 |
| 106 | Sigel | 2800 |
| 107 | Guy | 2600 |
| 108 | Karen | 2500 |

# METHOD-1: Without using SQL Analytic Functions

In order to find the second MAX salary, employee record with MAX salary needs to be eliminated. It can be achieved by using below SQL query.

**Query:**

```
SELECT MAX(salary) AS salary FROM Employees WHERE salary NOT IN (
  SELECT MAX(salary) AS salary FROM Employees);
```

**Result:**

| SALARY |
| --- |
| 11000 |

The above query only gives the second MAX salary value. In order to fetch the entire employee record with second MAX salary we need to do a self-join on Employee table based on Salary value.

**Query:**

```
WITH TEMP AS(
  SELECT MAX(salary) AS salary FROM Employees WHERE salary NOT IN (
    SELECT MAX(salary) AS salary FROM Employees)
)
SELECT a.* FROM Employees a JOIN TEMP b on a.salary = b.salary;
```

**Result:**

| EMPLOYEE_ID | NAME | SALARY |
| --- | --- | --- |
| 103 | Den | 11000 |

# METHOD-2: Using SQL Analytic Functions

**Query:**

**The DENSE_RANK is an analytic function that calculates the rank of a row in an ordered set of rows starting from 1. Unlike the RANK function, the DENSE_RANK function returns rank values as consecutive integers.**

```
SELECT
  Employee_Id,
  Name,
  Salary
FROM(
  SELECT
    Employees.*,
    DENSE_RANK() OVER(ORDER BY Salary DESC) as SALARY_RANK
```

```
  FROM Employees
)
WHERE SALARY_RANK =2;
```

**Result:**

| EMPLOYEE_ID | NAME | SALARY |
|---|---|---|
| 103 | Den | 11000 |

***By replacing the value of SALARY_RANK, any highest salary rank can be found easily.***

**Related Article:** [SQL Analytic Functions Interview Questions]