

-60%

```

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

```

-60%

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



-60%

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](#)

2. How to delete DUPLICATE records from a table using a SQL Query?

Consider the same EMPLOYEE table as source discussed in previous question

METHOD-1: Using ROWID and ROW_NUMBER Analytic Function

STEP-1: Using ROW_NUMBER Analytic function, assign row numbers to each unique set of records. Select ROWID of the rows along with the source columns

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

ROWID	EMPLOYEE_ID	NAME	SALARY	ROW_NUMBER
AAASnBAAEAAACrWAAA	100	Jennifer	4400	1
AAASnBAAEAAACrWAAB	100	Jennifer	4400	2
AAASnBAAEAAACrWAAC	101	Michael	13000	1
AAASnBAAEAAACrWAAD	101	Michael	13000	2
AAASnBAAEAAACrWAAE	101	Michael	13000	3
AAASnBAAEAAACrWAAF	102	Pat	6000	1
AAASnBAAEAAACrWAAG	102	Pat	6000	2
AAASnBAAEAAACrWAAH	103	Den	11000	1

^



-60%

```

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:

ROWID
AAASnBAAEAAACrWAAB
AAASnBAAEAAACrWAAD
AAASnBAAEAAACrWAAE
AAASnBAAEAAACrWAAG

STEP-3: Delete the records from the source table using the ROWID values fetched in previous step

Query:

```

DELETE FROM EMP WHERE ROWID IN (
SELECT ROWID FROM(
SELECT ROWID,
ROW_NUMBER() OVER(PARTITION BY EMPLOYEE_ID,NAME,SALARY ORDER BY EMPLOYEE_ID) AS
ROW_NUMBER
FROM EMPLOYEE)
WHERE ROW_NUMBER > 1);

```

Result:

The table EMPLOYEE will have below records after deleting the duplicates

ROWID	EMPLOYEE_ID	NAME	SALARY
AAASnBAAEAAACrWAAA	100	Jennifer	4400
AAASnBAAEAAACrWAAC	101	Michael	13000
AAASnBAAEAAACrWAAF	102	Pat	6000
AAASnBAAEAAACrWAAH	103	Den	11000

-60%

METHOD-2: Using ROWID and Correlated subquery

Correlated subquery is used for row-by-row processing. With a normal nested subquery, the inner **SELECT** query runs once and executes first. The returning values will be used by the main query. A correlated subquery, however, executes once for every row of the outer query. In other words, the inner query is driven by the outer query.

In the below query, we are comparing the ROWIDs' of the unique set of records and keeping the record with MIN ROWID and deleting all other rows.

Query:

```
DELETE FROM EMPLOYEE A WHERE ROWID > (SELECT MIN(ROWID) FROM EMPLOYEE B WHERE  
B.EMPLOYEE_ID = A.EMPLOYEE_ID );
```

Result:

The table EMPLOYEE will have below records after deleting the duplicates

ROWID	EMPLOYEE_ID	NAME	SALARY
AAASnBAAEAAACrWAAA	100	Jennifer	4400
AAASnBAAEAAACrWAAC	101	Michael	13000
AAASnBAAEAAACrWAAF	102	Pat	6000
AAASnBAAEAAACrWAAH	103	Den	11000



ARCHANAM

Receive the blessings of the Lord
on your special occasions

SPONSOR NOW

-60%

```
DELETE FROM EMPLOYEE A WHERE ROWID < (SELECT MAX(ROWID) FROM EMPLOYEE B WHERE
B.EMPLOYEE_ID = A.EMPLOYEE_ID );
```

Result:

The table EMPLOYEE will have below records after deleting the duplicates

ROWID	EMPLOYEE_ID	NAME	SALARY
AAASnBAAEAAACrWAAA	100	Jennifer	4400
AAASnBAAEAAACrWAAC	101	Michael	13000
AAASnBAAEAAACrWAAF	102	Pat	6000
AAASnBAAEAAACrWAAH	103	Den	11000

3. How to read TOP 5 records from a table using a SQL query?

Consider below table DEPARTMENTS as the source data

```
CREATE TABLE Departments(
  Department_ID number,
  Department_Name varchar(50)
);

INSERT INTO DEPARTMENTS VALUES('10','Administration');
INSERT INTO DEPARTMENTS VALUES('20','Marketing');
INSERT INTO DEPARTMENTS VALUES('30','Purchasing');
INSERT INTO DEPARTMENTS VALUES('40','Human Resources');
INSERT INTO DEPARTMENTS VALUES('50','Shipping');
INSERT INTO DEPARTMENTS VALUES('60','IT');
INSERT INTO DEPARTMENTS VALUES('70','Public Relations');
INSERT INTO DEPARTMENTS VALUES('80','Sales');

SELECT * FROM Departments;
```

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
30	Purchasing

-60%

70	Public Relations
80	Sales

ROWNUM is a “Pseudocolumn” that assigns a number to each row returned by a query indicating the order in which Oracle selects the row from a table. The first row selected has a ROWNUM of 1, the second has 2, and so on.

Query:

```
SELECT * FROM Departments WHERE ROWNUM <= 5;
```

Result:

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
30	Purchasing
40	Human Resources
50	Shipping

4. How to read LAST 5 records from a table using a SQL query?

Consider the same DEPARTMENTS table as source discussed in previous question.

In order to select the last 5 records we need to find (**count of total number of records – 5**) which gives the count of records from first to last but 5 records.

-60%



Using the MINUS function we can compare **all records from DEPARTMENTS table** with **records from first to last but 5 from DEPARTMENTS table** which give the last 5 records of the table as result.

MINUS operator is used to return all rows in the first SELECT statement that are not present in the second SELECT statement.

Query:

```
SELECT * FROM Departments  
  
MINUS  
  
SELECT * FROM Departments WHERE ROWNUM <= (SELECT COUNT(*)-5 FROM Departments);
```

Result:

DEPARTMENT_ID	DEPARTMENT_NAME
40	Human Resources
50	Shipping
60	IT
70	Public Relations
80	Sales

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



	-60%
0	
null	

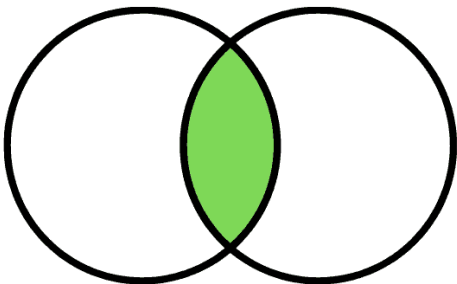
Table_B

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:



```
FROM TABLE_A JOIN TABLE_B
ON a.COL = b.COL;
```

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.

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

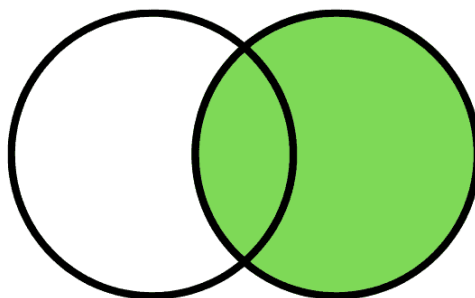
-60%

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;
```

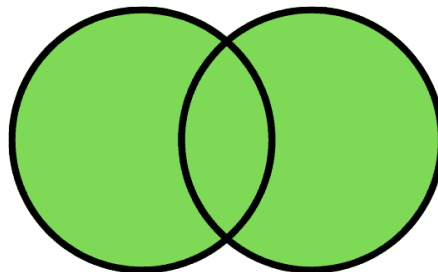
Result:

-60%

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:

 \wedge

	-60%	
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	Sigal	2800

-60%

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





-60%

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](#)

7. How to find the employee with third MAX Salary using a SQL query without using Analytic Functions?

Consider the same **EMPLOYEES** table as source discussed in previous question

In order to find the third MAX salary, we need to eliminate the top 2 salary records. But we cannot use the same method we used for finding second MAX salary (not a best practice). Imagine if we have to find the fifth MAX salary. We should not be writing a query with four nested sub queries.

STEP-1:

The approach here is to first list all the records based on Salary in the descending order with MAX salary or  and MIN salary at bottom. Next, using ROWNUM select the top 2 records. 

-60%

WHERE ROWNUM < 3;

Result:

Salary
13000
11000

STEP-2:

Next find the MAX salary from EMPLOYEE table which is not one of top two salary values fetched in the earlier step.

Query:

```
SELECT MAX(salary) as salary FROM Employees WHERE salary NOT IN (  
    SELECT salary FROM(  
        SELECT salary FROM Employees ORDER BY salary DESC)  
    WHERE ROWNUM < 3  
);
```

Result:

SALARY
6000

STEP-3:

In order to fetch the entire employee record with third MAX salary we need to do a self-join on Employee table based on Salary value.

Query:

