Top 100 SQL Query Interview Questions for Practice

Let’s prepare sample data for SQL practice.

Sample Table – Worker

1. `CREATE DATABASE ORG;`

This command is used to create a new database named "ORG".

- **CREATE DATABASE**:
  - **CREATE DATABASE** is an SQL statement used to create a new database.
- **ORG**:
  - "ORG" is the name of the database being created.

2. SHOW DATABASES;

This command is used to display a list of all databases on the SQL server.

SHOW DATABASES:

SHOW DATABASES is an SQL statement that lists all the databases present on the SQL server.

3. USE ORG;

This command is used to switch to the "ORG" database, making it the current active database for subsequent SQL operations.

USE:

USE is an SQL statement that specifies which database you want to work with.

ORG:

"ORG" is the name of the database that you want to switch to.

Explanation in Detail:

Creating a Database:

```sql
CREATE DATABASE ORG;
```

Displaying All Databases:

```sql
SHOW DATABASES;
```

- After executing the above command, you will see a list of all databases, including the newly created "ORG" database.

Example output:

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| org                |
| performance_schema |
| sys                |
+--------------------+
```

Switching to the New Database:

```sql
USE ORG;
```

Example:

Here's how you can execute these commands:

sql Copy code

```sql
CREATE DATABASE ORG;
SHOW DATABASES;
USE ORG;
```

SQL Components:

```sql
CREATE TABLE Worker (
    WORKER_ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    FIRST_NAME CHAR(25),
    LAST_NAME CHAR(25),
    SALARY INT(15),
    JOINING_DATE DATETIME,
    DEPARTMENT CHAR(25)
);
```

1. **CREATE TABLE:**
    - The CREATE TABLE statement is used to create a new table in the database.
2. **Worker:**
    - "Worker" is the name of the table being created.
3. **Columns:**
    - **The table "Worker" has the following columns:**
        - **WORKER_ID:**
            - **Data Type: INT**
            - **Constraints:**
                - **NOT NULL:** Ensures that this column cannot have a NULL value.
                - **PRIMARY KEY**: Specifies that this column is the primary key of the table**.**

- **AUTO_INCREMENT (in MySQL) or IDENTITY(1,1) (in SQL Server):** Automatically generates a unique value for each new row.

- **FIRST_NAME:**
  - **Data Type: CHAR(25)**
  - This column can store up to 25 characters.
- **LAST_NAME:**
  - **Data Type:** CHAR(25)
  - This column can store up to 25 characters.
- **SALARY:**
  - **Data Type:** INT(15)
  - This column stores integer values up to 15 digits.
- **JOINING_DATE:**
  - **Data Type:** DATETIME
  - This column stores date and time values.
- **DEPARTMENT:**
  - **Data Type:** CHAR(25)
  - This column can store up to 25 characters.

Complete Query:

sql Copy code

```sql
CREATE TABLE Worker (
    WORKER_ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    FIRST_NAME CHAR(25),
    LAST_NAME CHAR(25),
    SALARY INT(15),
    JOINING_DATE DATETIME,
    DEPARTMENT CHAR(25)
);
```

How to Insert Data into Table:

SQL Query:

sql Copy code

```sql
INSERT INTO Worker
(WORKER_ID, FIRST_NAME, LAST_NAME, SALARY, JOINING_DATE, DEPARTMENT) VALUES
    (001, 'Monika', 'Arora', 100000, '21-02-20 09.00.00', 'HR'),
    (002, 'Niharika', 'Verma', 80000, '21-06-11 09.00.00', 'Admin'),
    (003, 'Vishal', 'Singhal', 300000, '21-02-20 09.00.00', 'HR'),
    (004, 'Amitabh', 'Singh', 500000, '21-02-20 09.00.00', 'Admin'),
    (005, 'Vivek', 'Bhati', 500000, '21-06-11 09.00.00', 'Admin'),
    (006, 'Vipul', 'Diwan', 200000, '21-06-11 09.00.00', 'Account'),
    (007, 'Satish', 'Kumar', 75000, '21-01-20 09.00.00', 'Account'),
    (008, 'Geetika', 'Chauhan', 90000, '21-04-11 09.00.00', 'Admin');
```

Explanation:

INSERT INTO Worker

- **INSERT INTO:**
  - The INSERT INTO statement is used to insert new records into a table.
- **Worker:**
  - "Worker" is the name of the table into which the records will be inserted.

(WORKER_ID, FIRST_NAME, LAST_NAME, SALARY, JOINING_DATE, DEPARTMENT) VALUES

- **(WORKER_ID, FIRST_NAME, LAST_NAME, SALARY, JOINING_DATE, DEPARTMENT):**
  - These are the column names of the "Worker" table where the data will be inserted.
- **VALUES:**
  - The VALUES keyword is used to specify the values to be inserted into the respective columns.

Records to Insert:

The query is inserting the following records into the "Worker" table:

1. **WORKER_ID = 001, FIRST_NAME = 'Monika', LAST_NAME = 'Arora', SALARY = 100000, JOINING_DATE = '21-02-20 09.00.00', DEPARTMENT = 'HR'**
2. **WORKER_ID = 002, FIRST_NAME = 'Niharika', LAST_NAME = 'Verma', SALARY = 80000, JOINING_DATE = '21-06-11 09.00.00', DEPARTMENT = 'Admin'**
3. **WORKER_ID = 003, FIRST_NAME = 'Vishal', LAST_NAME = 'Singhal', SALARY = 300000, JOINING_DATE = '21-02-20 09.00.00', DEPARTMENT = 'HR'**
4. **WORKER_ID = 004, FIRST_NAME = 'Amitabh', LAST_NAME = 'Singh', SALARY = 500000, JOINING_DATE = '21-02-20 09.00.00', DEPARTMENT = 'Admin'**
5. **WORKER_ID = 005, FIRST_NAME = 'Vivek', LAST_NAME = 'Bhati', SALARY = 500000, JOINING_DATE = '21-06-11 09.00.00', DEPARTMENT = 'Admin'**
6. **WORKER_ID = 006, FIRST_NAME = 'Vipul', LAST_NAME = 'Diwan', SALARY = 200000, JOINING_DATE = '21-06-11 09.00.00', DEPARTMENT = 'Account'**
7. **WORKER_ID = 007, FIRST_NAME = 'Satish', LAST_NAME = 'Kumar', SALARY = 75000, JOINING_DATE = '21-01-20 09.00.00', DEPARTMENT = 'Account'**
8. **WORKER_ID = 008, FIRST_NAME = 'Geetika', LAST_NAME = 'Chauhan', SALARY = 90000, JOINING_DATE = '21-04-11 09.00.00', DEPARTMENT = 'Admin'**

Example:

After executing the INSERT INTO query, the "Worker" table will have the following records:

WORKER_ID	FIRST_NAME	LAST_NAME	SALARY	JOINING_DATE	DEPARTMENT
001	Monika	Arora	100000	21-02-20 09.00.00	HR
002	Niharika	Verma	80000	21-06-11 09.00.00	Admin
003	Vishal	Singhal	300000	21-02-20 09.00.00	HR
004	Amitabh	Singh	500000	21-02-20 09.00.00	Admin
005	Vivek	Bhati	500000	21-06-11 09.00.00	Admin
006	Vipul	Diwan	200000	21-06-11 09.00.00	Account
007	Satish	Kumar	75000	21-01-20 09.00.00	Account
008	Geetika	Chauhan	90000	21-04-11 09.00.00	Admin

Create another table:Bonus

```
CREATE TABLE Bonus (
    WORKER_REF_ID INT,
    BONUS_AMOUNT INT(10),
    BONUS_DATE DATETIME,
    FOREIGN KEY (WORKER_REF_ID)
        REFERENCES Worker(WORKER_ID)
        ON DELETE CASCADE
);
```

SQL Components:

1. **CREATE TABLE:**
    - The CREATE TABLE statement is used to create a new table in the database.
2. **Bonus:**
    - "Bonus" is the name of the table being created.
3. **WORKER_REF_ID, BONUS_AMOUNT, BONUS_DATE:**
    - These are the columns of the "Bonus" table with their respective data types.
4. **FOREIGN KEY:**
    - The FOREIGN KEY constraint is used to ensure the referential integrity of the data in the table.
5. **REFERENCES Worker(WORKER_ID):**
    - This specifies that the WORKER_REF_ID column in the "Bonus" table is a foreign key that references the WORKER_ID column in the "Worker" table.
6. **ON DELETE CASCADE:**
    - This specifies the action to be taken when a referenced row in the "Worker" table is deleted. CASCADE means that if a row in the "Worker" **table is** deleted, then the corresponding rows in the "Bonus" table with the same WORKER_REF_ID will also be automatically deleted.

Explanation:

Here's the breakdown of the query:

```sql
CREATE TABLE Bonus (
    WORKER_REF_ID INT,
    BONUS_AMOUNT INT(10),
    BONUS_DATE DATETIME,
    FOREIGN KEY (WORKER_REF_ID)
        REFERENCES Worker(WORKER_ID)
        ON DELETE CASCADE
);
```

Table Creation:

```sql
CREATE TABLE Bonus (
```

This line begins the creation of the "Bonus" table.

Columns:

```sql
WORKER_REF_ID INT,
BONUS_AMOUNT INT(10),
BONUS_DATE DATETIME,
```

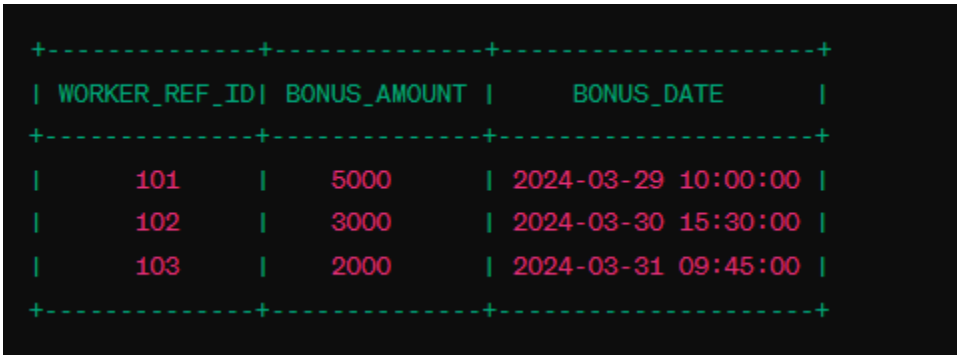
- WORKER_REF_ID: An integer column to store the reference ID of the worker.
- BONUS_AMOUNT: An integer column to store the bonus amount.
- BONUS_DATE: A datetime column to store the date of the bonus.

Foreign Key Constraint:

```sql
FOREIGN KEY (WORKER_REF_ID)
    REFERENCES Worker(WORKER_ID)
    ON DELETE CASCADE
```

- *FOREIGN KEY (WORKER_REF_ID):* This line sets up the WORKER_REF_ID column as a foreign key.
- *REFERENCES Worker(WORKER_ID):* This specifies that the foreign key references the WORKER_ID column in the "Worker" table.
- *ON DELETE CASCADE*: This means that if a row in the "Worker" table is deleted, then the corresponding rows in the "Bonus" table with the same WORKER_REF_ID will also be automatically deleted to maintain referential integrity.

Here's how the "Bonus" table might look after creating it:



```
+--------------+--------------+---------------------+
| WORKER_REF_ID| BONUS_AMOUNT |      BONUS_DATE     |
+--------------+--------------+---------------------+
|      101     |    5000      | 2024-03-29 10:00:00 |
|      102     |    3000      | 2024-03-30 15:30:00 |
|      103     |    2000      | 2024-03-31 09:45:00 |
+--------------+--------------+---------------------+
```

This table has three columns:

- WORKER_REF_ID: The reference ID of the worker.
- BONUS_AMOUNT: The bonus amount.
- BONUS_DATE: The date and time when the bonus was given.

The **WORKER_REF_ID** is a foreign key that references the **WORKER_ID** column in the "Worker" table. If a row with a specific **WORKER_ID** is deleted from the "Worker" table, the corresponding rows in the "Bonus" table with the same **WORKER_REF_ID** will be automatically deleted due to the ***ON DELETE CASCADE*** constraint.

Insert record into Bonus table:

```
INSERT INTO Bonus
(WORKER_REF_ID, BONUS_AMOUNT, BONUS_DATE) VALUES
    (001, 5000, '23-02-20'),
    (002, 3000, '23-06-11'),
    (003, 4000, '23-02-20'),
    (001, 4500, '23-02-20'),
    (002, 3500, '23-06-11');
```

SQL Components:

1. **INSERT INTO**:
    - The **INSERT INTO** statement is used to insert new records into a table.
2. **Bonus**:
    - **Bonus** is the name of the table where the data will be inserted.
3. **(WORKER_REF_ID, BONUS_AMOUNT, BONUS_DATE)**:
    - These are the columns of the **Bonus** table into which data will be inserted.
4. **VALUES**:
    - The **VALUES** keyword is used to specify the values to be inserted into the specified columns.

Explanation:

The query is inserting multiple rows of data into the **Bonus** table with the following columns: **WORKER_REF_ID**, **BONUS_AMOUNT**, and **BONUS_DATE**.

Data to be Inserted:

Here is the data that will be inserted into the **Bonus** table:

WORKER_REF_ID	BONUS_AMOUNT	BONUS_DATE
001	5000	23-02-20
002	3000	23-06-11
003	4000	23-02-20
001	4500	23-02-20
002	3500	23-06-11

Query Breakdown:

```sql
INSERT INTO Bonus
(WORKER_REF_ID, BONUS_AMOUNT, BONUS_DATE) VALUES
    (001, 5000, '23-02-20'),
    (002, 3000, '23-06-11'),
    (003, 4000, '23-02-20'),
    (001, 4500, '23-02-20'),
    (002, 3500, '23-06-11');
```

- **INSERT INTO Bonus**:
    - This specifies that the data should be inserted into the **Bonus** table.
- **(WORKER_REF_ID, BONUS_AMOUNT, BONUS_DATE)**:
    - This part specifies the columns of the **Bonus** table where the data will be inserted.
- **VALUES**:
    - This keyword indicates that the following values will be inserted into the specified columns.
- The rows inside the **VALUES** clause:
    - Each row represents a set of values to be inserted into the respective columns of the **Bonus** table.

Example:

If the **Bonus** table is initially empty, executing the above query will insert the provided data into the table. After executing the query, the **Bonus** table will look like:

WORKER_REF_ID	BONUS_AMOUNT	BONUS_DATE
001	5000	23-02-20
002	3000	23-06-11
003	4000	23-02-20
001	4500	23-02-20
002	3500	23-06-11

Create a new table Worker_Title:

```
CREATE TABLE Title (
    WORKER_REF_ID INT,
    WORKER_TITLE CHAR(25),
    AFFECTED_FROM DATETIME,
    FOREIGN KEY (WORKER_REF_ID)
        REFERENCES Worker(WORKER_ID)
        ON DELETE CASCADE
);
```

SQL Components:

1. **CREATE TABLE**:
    - The **CREATE TABLE** statement is used to create a new table in the database.
2. **Title**:
    - "Title" is the name of the table being created.
3. **Column Definitions**:
    - **WORKER_REF_ID INT**:
        - **WORKER_REF_ID** is a column of type **INT** (integer).
    - **WORKER_TITLE CHAR(25)**:
        - **WORKER_TITLE** is a column of type **CHAR(25)** which can store up to 25 characters.
    - **AFFECTED_FROM DATETIME**:
        - **AFFECTED_FROM** is a column of type **DATETIME** which can store date and time values.
4. **FOREIGN KEY Constraint**:
    - **FOREIGN KEY (WORKER_REF_ID)**:
        - This defines a foreign key constraint on the **WORKER_REF_ID** column.
    - **REFERENCES Worker(WORKER_ID)**:
        - This specifies that the **WORKER_REF_ID** column references the **WORKER_ID** column in the **Worker** table.
    - **ON DELETE CASCADE**:
        - This means that if a record in the **Worker** table is deleted, all related records in the **Title** table with the corresponding **WORKER_REF_ID** will also be deleted automatically.

Explanation in Detail:

The query is creating a table named "Title" with the following columns:

1. **WORKER_REF_ID**:
    - This column is of type **INT** and is likely used to reference a worker's ID from another table (presumably the **Worker** table).
2. **WORKER_TITLE**:
    - This column is of type **CHAR(25)** and is intended to store the title of the worker, such as "Manager", "Developer", etc.
3. **AFFECTED_FROM**:
    - This column is of type **DATETIME** and is likely used to store the date and time from which the title is affected or valid.
4. **FOREIGN KEY Constraint**:
    - The **FOREIGN KEY** constraint ensures the integrity of the data. It makes sure that the values in the **WORKER_REF_ID** column in the **Title** table correspond to values in the **WORKER_ID** column in the **Worker** table.
    - The **ON DELETE CASCADE** clause ensures referential integrity. If a record in the **Worker** table is deleted, all related records in the **Title** table will also be deleted automatically.

In this example:

- The **Worker** table has been created with a **WORKER_ID**, **FIRST_NAME**, and **LAST_NAME**.
- A worker named "John Doe" with **WORKER_ID** of 1 has been added to the **Worker** table.
- The **Title** table has been created with a **WORKER_REF_ID**, **WORKER_TITLE**, and **AFFECTED_FROM**.
- A title of "Manager" for the worker with **WORKER_REF_ID** of 1 has been added to the **Title** table.

```
INSERT INTO Title
    (WORKER_REF_ID, WORKER_TITLE, AFFECTED_FROM)
VALUES
    (001, 'Manager', '2023-02-20 00:00:00'),
    (002, 'Executive', '2023-06-11 00:00:00'),
    (008, 'Executive', '2023-06-11 00:00:00'),
    (005, 'Manager', '2023-06-11 00:00:00'),
    (004, 'Asst. Manager', '2023-06-11 00:00:00'),
    (007, 'Executive', '2023-06-11 00:00:00'),
    (006, 'Lead', '2023-06-11 00:00:00'),
    (003, 'Lead', '2023-06-11 00:00:00');
```

Output:

WORKER_REF_ID	WORKER_TITLE	AFFECTED_FROM
001	Manager	2023-02-20 00:00:00
002	Executive	2023-06-11 00:00:00
008	Executive	2023-06-11 00:00:00
005	Manager	2023-06-11 00:00:00
004	Asst. Manager	2023-06-11 00:00:00
007	Executive	2023-06-11 00:00:00
006	Lead	2023-06-11 00:00:00
003	Lead	2023-06-11 00:00:00

Start Practice with 100 SQL Query Interview Questions.

Q-1. Write an SQL query to fetch "FIRST_NAME" from the Worker table using the alias name <WORKER_NAME>.

```sql
SELECT FIRST_NAME AS <WORKER_NAME> FROM Worker;
```

In this query:

- `SELECT FIRST_NAME`: specifies the column to be retrieved.
- `AS <WORKER_NAME>`: assigns an alias `<WORKER_NAME>` to the `FIRST_NAME` column in the result set.
- `FROM Worker`: specifies the table from which the data will be retrieved.

Q-2. Write an SQL query to fetch "FIRST_NAME" from the Worker table in upper case.

```sql
SELECT UPPER(FIRST_NAME) AS FIRST_NAME_UPPERCASE FROM Worker;
```

In this query:

- `SELECT UPPER(FIRST_NAME)`: specifies the `FIRST_NAME` column to be retrieved and converted to upper case using the `UPPER()` function.
- `AS FIRST_NAME_UPPERCASE`: assigns an alias `FIRST_NAME_UPPERCASE` to the uppercased `FIRST_NAME` column in the result set.
- `FROM Worker`: specifies the table from which the data will be retrieved.

Q-3. Write an SQL query to fetch unique values of DEPARTMENT from the Worker table.

```
SELECT DISTINCT DEPARTMENT FROM Worker;
```

In this query:

- SELECT DISTINCT DEPARTMENT: specifies the DEPARTMENT column to be retrieved with unique values using the DISTINCT keyword.
- FROM Worker: specifies the table from which the data will be retrieved.

Q-4. Write an SQL query to print the first three characters of FIRST_NAME from the Worker table.

```
Select substring(FIRST_NAME,1,3) from Worker;
```

Q-5. Write an SQL query to find the position of the alphabet (‘a’) in the first name column ‘Amitabh’ from the Worker table.

```
Select INSTR(FIRST_NAME, BINARY'a') from Worker where FIRST_NAME = 'Amitabh';
```

SELECT CHARINDEX('a', FIRST_NAME) AS PositionOfA FROM Worker WHERE FIRST_NAME = 'Amitabh';

In this query:

- SELECT CHARINDEX('a', FIRST_NAME) AS PositionOfA: uses the CHARINDEX() function to find the position of the character 'a' in the FIRST_NAME column. The result is aliased as PositionOfA.
- FROM Worker: specifies the table from which the data will be retrieved.
- WHERE FIRST_NAME = 'Amitabh': filters the records where the FIRST_NAME is 'Amitabh'.

Q-6. Write an SQL query to print the FIRST_NAME from the Worker table after removing white spaces from the right side.

Ans.

```
Select RTRIM(FIRST_NAME) from Worker;
```

In this query:

- SELECT RTRIM(FIRST_NAME): uses the RTRIM() function to remove trailing white spaces from the FIRST_NAME column.
- AS FIRST_NAME_CLEAN: assigns an alias FIRST_NAME_CLEAN to the cleaned FIRST_NAME column in the result set.
- FROM Worker: specifies the table from which the data will be retrieved.

Q-7. Write an SQL query to print the DEPARTMENT from the Worker table after removing white spaces from the left side.

Ans.

```
Select LTRIM(DEPARTMENT) from Worker;
```

Q-8. Write an SQL query that fetches the unique values of DEPARTMENT from the Worker table and prints its length.

Ans.

```
Select distinct length(DEPARTMENT) from Worker;
```

Q-9. Write an SQL query to print the FIRST_NAME from the Worker table after replacing 'a' with 'A'.

Ans.

```
Select REPLACE(FIRST_NAME,'a','A') from Worker;
```

Q-10. Write an SQL query to print the FIRST_NAME and LAST_NAME from the Worker table into a single column COMPLETE_NAME. A space char should separate them.

Ans.

```
Select CONCAT(FIRST_NAME, ' ', LAST_NAME) AS 'COMPLETE_NAME' from Worker;
```

Q-11. Write an SQL query to print all Worker details from the Worker table order by FIRST_NAME Ascending.

Ans.

```
Select * from Worker order by FIRST_NAME asc;
```

Q-12. Write an SQL query to print all Worker details from the Worker table order by FIRST_NAME Ascending and DEPARTMENT Descending.

Ans.

```
Select * from Worker order by FIRST_NAME asc,DEPARTMENT desc;
```

Q-13. Write an SQL query to print details for Workers with the first names "Vipul" and "Satish" from the Worker table.

Ans.

```
Select * from Worker where FIRST_NAME in ('Vipul','Satish');
```

Q-14. Write an SQL query to print details of workers excluding first names, "Vipul" and "Satish" from the Worker table.

Ans.

```
Select * from Worker where FIRST_NAME not in ('Vipul','Satish');
```

Q-15. Write an SQL query to print details of Workers with DEPARTMENT name as "Admin".

Ans.

```
Select * from Worker where DEPARTMENT like 'Admin%';
```

Q-16. Write an SQL query to print details of the Workers whose FIRST_NAME contains 'a'.

Ans.

```
Select * from Worker where FIRST_NAME like '%a%';
```

Q-17. Write an SQL query to print details of the Workers whose FIRST_NAME ends with 'a'.

Ans.

```
Select * from Worker where FIRST_NAME like '%a';
```

Q-18. Write an SQL query to print details of the Workers whose FIRST_NAME ends with 'h' and contains six alphabets.

Ans.

```
Select * from Worker where FIRST_NAME like '_____h';
```

Q-19. Write an SQL query to print details of the Workers whose SALARY lies between 100000 and 500000.

Ans.

```
Select * from Worker where SALARY between 100000 and 500000;
```

Q-20. Write an SQL query to print details of the Workers who joined in Feb 2021.

Ans.

```
Select * from Worker where year(JOINING_DATE) = 2021 and month(JOINING_DATE)
= 2;
```

Q-21. Write an SQL query to fetch the count of employees working in the department 'Admin'.

Ans.

```
SELECT COUNT(*) FROM worker WHERE DEPARTMENT = 'Admin';
```

Q-22. Write an SQL query to fetch worker names with salaries >= 50000 and <= 100000.

Ans.

```
SELECT CONCAT(FIRST_NAME, ' ', LAST_NAME) As Worker_Name, Salary
FROM worker
WHERE WORKER_ID IN
(SELECT WORKER_ID FROM worker
WHERE Salary BETWEEN 50000 AND 100000);
```

Q-23. Write an SQL query to fetch the number of workers for each department in descending order.

Ans.

```
SELECT DEPARTMENT, count(WORKER_ID) No_Of_Workers
FROM worker
GROUP BY DEPARTMENT
ORDER BY No_Of_Workers DESC;
```

Q-24. Write an SQL query to print details of the Workers who are also Managers.

Ans.

```
SELECT DISTINCT W.FIRST_NAME, T.WORKER_TITLE
FROM Worker W
INNER JOIN Title T
ON W.WORKER_ID = T.WORKER_REF_ID
AND T.WORKER_TITLE in ('Manager');
```

Q-25. Write an SQL query to fetch duplicate records having matching data in some fields of a table.

Ans.

```
SELECT WORKER_TITLE, AFFECTED_FROM, COUNT(*)
FROM Title
```

```
GROUP BY WORKER_TITLE, AFFECTED_FROM
HAVING COUNT(*) > 1;
```

Q-26. Write an SQL query to show only odd rows from a table.

Ans.

```
SELECT * FROM Worker WHERE MOD (WORKER_ID, 2) <> 0;
```

Q-27. Write an SQL query to show only even rows from a table.

Ans.

```
SELECT * FROM Worker WHERE MOD (WORKER_ID, 2) = 0;
```

Q-28. Write an SQL query to clone a new table from another table.

Ans.

```
SELECT * INTO WorkerClone FROM Worker;
```

The general way to clone a table without information is:

```
SELECT * INTO WorkerClone FROM Worker WHERE 1 = 0;
```

An alternate way to clone a table (for MySQL) without data is:

```
CREATE TABLE WorkerClone LIKE Worker;
```

Q-29. Write an SQL query to fetch intersecting records of two tables.

Ans.

```
(SELECT * FROM Worker)
INTERSECT
```

```
(SELECT * FROM WorkerClone);
```

Q-30. Write an SQL query to show records from one table that another table does not have.

Ans.

```
SELECT * FROM Worker
MINUS
SELECT * FROM Title;
```

Q-31. Write an SQL query to show the current date and time.

Ans.

The following MySQL query returns the current date:

```
SELECT CURDATE();
```

Whereas the following MySQL query returns the current date and time:

```
SELECT NOW();
```

Here is a SQL Server query that returns the current date and time:

```
SELECT getdate();
```

Find this Oracle query that also returns the current date and time:

```
SELECT SYSDATE FROM DUAL;
```

Q-32. Write an SQL query to show the top n (say 10) records of a table.

Ans.

MySQL query to return the top n records using the LIMIT method:

```
SELECT * FROM Worker ORDER BY Salary DESC LIMIT 10;
```

SQL Server query to return the top n records using the TOP command:

```
SELECT TOP 10 * FROM Worker ORDER BY Salary DESC;
```

Oracle query to return the top n records with the help of ROWNUM:

```
SELECT * FROM (SELECT * FROM Worker ORDER BY Salary DESC)
WHERE ROWNUM <= 10;
```

Now, that you should have a solid foundation in intermediate SQL, let's take a look at some more advanced SQL query questions. These questions will require us to use more complex SQL syntax and concepts, such as nested queries, joins, unions, and intersects.

Q-33. Write an SQL query to determine the nth (say n=5) highest salary from a table.

Ans.

MySQL query to find the nth highest salary:

```
SELECT Salary FROM Worker ORDER BY Salary DESC LIMIT n-1,1;
```

SQL Server query to find the nth highest salary:

```
SELECT TOP 1 Salary
FROM (
 SELECT DISTINCT TOP n Salary
 FROM Worker
```

```
 ORDER BY Salary DESC
 )
ORDER BY Salary ASC;
```

Q-34. Write an SQL query to determine the 5th highest salary without using the TOP or limit method.

Ans.

The following query is using the correlated subquery to return the 5th highest salary:

```
SELECT Salary
FROM Worker W1
WHERE 4 = (
 SELECT COUNT( DISTINCT ( W2.Salary ) )
 FROM Worker W2
 WHERE W2.Salary >= W1.Salary
 );
```

Use the following generic method to find the nth highest salary without using TOP or limit.

```
SELECT Salary
FROM Worker W1
WHERE n-1 = (
 SELECT COUNT( DISTINCT ( W2.Salary ) )
 FROM Worker W2
 WHERE W2.Salary >= W1.Salary
 );
```

Q-35. Write an SQL query to fetch the list of employees with the same salary.

Ans.

```
Select distinct W.WORKER_ID, W.FIRST_NAME, W.Salary
from Worker W, Worker W1
```

```
where W.Salary = W1.Salary
and W.WORKER_ID != W1.WORKER_ID;
```

Q-36. Write an SQL query to show the second-highest salary from a table.

Ans.

```
Select max(Salary) from Worker
where Salary not in (Select max(Salary) from Worker);
```

Q-37. Write an SQL query to show one row twice in the results from a table.

Ans.

```
select FIRST_NAME, DEPARTMENT from worker W where W.DEPARTMENT='HR'
union all
select FIRST_NAME, DEPARTMENT from Worker W1 where W1.DEPARTMENT='HR';
```

Q-38. Write an SQL query to fetch intersecting records of two tables.

Ans.

```
(SELECT * FROM Worker)
INTERSECT
(SELECT * FROM WorkerClone);
```

Q-39. Write an SQL query to fetch the first 50% of records from a table.

Ans.

```
SELECT *
FROM WORKER
```

```
WHERE WORKER_ID <= (SELECT count(WORKER_ID)/2 from Worker);
```

Practicing SQL query interview questions is a great way to improve your understanding of the language and become more proficient in using it. However, in addition to improving your technical skills, practicing SQL query questions can also help you advance your career. Many employers are looking for candidates who have strong SQL skills, so being able to demonstrate your proficiency in the language can give you a competitive edge.

Q-40. Write an SQL query to fetch the departments that have less than five people in them.

Ans.

```
SELECT DEPARTMENT, COUNT(WORKER_ID) as 'Number of Workers' FROM Worker GROUP BY DEPARTMENT HAVING COUNT(WORKER_ID) < 5;
```

Q-41. Write an SQL query to show all departments along with the number of people in there.

Ans.

The following query returns the expected result:

```
SELECT DEPARTMENT, COUNT(DEPARTMENT) as 'Number of Workers' FROM Worker GROUP BY DEPARTMENT;
```

Q-42. Write an SQL query to show the last record from a table.

Ans.

```
Select * from Worker where WORKER_ID = (SELECT max(WORKER_ID) from Worker);
```

Q-43. Write an SQL query to fetch the first row of a table.

Ans.

```
Select * from Worker where WORKER_ID = (SELECT min(WORKER_ID) from Worker);
```

Q-44. Write an SQL query to fetch the last five records from a table.

Ans.

```
SELECT * FROM Worker WHERE WORKER_ID <=5
UNION
SELECT * FROM (SELECT * FROM Worker W order by W.WORKER_ID DESC) AS W1 WHERE
W1.WORKER_ID <=5;
```

Q-45. Write an SQL query to print the names of employees having the highest salary in each department.

Ans.

```
SELECT t.DEPARTMENT,t.FIRST_NAME,t.Salary from(SELECT max(Salary) as
TotalSalary,DEPARTMENT from Worker group by DEPARTMENT) as TempNew
Inner Join Worker t on TempNew.DEPARTMENT=t.DEPARTMENT
 and TempNew.TotalSalary=t.Salary;
```

Q-46. Write an SQL query to fetch three max salaries from a table.

Ans.

```
SELECT distinct Salary from worker a WHERE 3 >= (SELECT count(distinct Salary)
from worker b WHERE a.Salary <= b.Salary) order by a.Salary desc;
```

Q-47. Write an SQL query to fetch three min salaries from a table.

Ans.

```
SELECT distinct Salary from worker a WHERE 3 >= (SELECT count(distinct Salary)
from worker b WHERE a.Salary >= b.Salary) order by a.Salary desc;
```

Q-48. Write an SQL query to fetch nth max salaries from a table.

Ans.

```
SELECT distinct Salary from worker a WHERE n >= (SELECT count(distinct Salary)
from worker b WHERE a.Salary <= b.Salary) order by a.Salary desc;
```

Q-49. Write an SQL query to fetch departments along with the total salaries paid for each of them.

Ans.

```
SELECT DEPARTMENT, sum(Salary) from worker group by DEPARTMENT;
```

Q-50. Write an SQL query to fetch the names of workers who earn the highest salary.

Ans.

```
SELECT FIRST_NAME, SALARY from Worker WHERE SALARY=(SELECT max(SALARY) from
Worker);
```

SQL Query for Microsoft SQL Server Databse:

---Create database

Create database ORG;

--how to use ORG database

```
Use ORG
----
--how to create table
Create table Worker
(
  WORKER_ID INT IDENTITY(1,1) PRIMARY KEY,
  FIRST_NAME CHAR(25),
  LAST_NAME CHAR(25),
  SALARY INT,
  JOINING_DATE DATETIME,
  DEPARMENT CHAR(25)
);

SELECT * FROM Worker

--DATA INSERT
INSERT INTO Worker(FIRST_NAME,LAST_NAME,SALARY,JOINING_DATE,DEPARMENT)
VALUES
( 'Monika', 'Arora', 100000, '2024-02-20', 'HR'),
( 'Niharika', 'Verma', 80000, '2024-01-11 ', 'Admin'),
( 'Vishal', 'Singhal', 300000, '2024-02-20', 'HR'),
( 'Amitabh', 'Singh', 500000, '2024-02-20 ', 'Admin'),
( 'Vivek', 'Bhati', 500000, '2024-02-11 ', 'Admin'),
( 'Vipul', 'Diwan', 200000, '2024-03-11', 'Account'),
( 'Satish', 'Kumar', 75000, '2024-01-20 ', 'Account'),
( 'Geetika', 'Chauhan', 90000, '2024-02-11', 'Admin');
```

```
INSERT INTO Worker(FIRST_NAME,LAST_NAME,SALARY,JOINING_DATE,DEPARTMENT)
VALUES

( 'Ravi', 'Yadav', 100000, '2021-02-20', 'HR')


-- BONUS

CREATE TABLE Bonus (

        WORKER_REF_ID INT,

        BONUS_AMOUNT INT,

        BONUS_DATE DATETIME,

        FOREIGN KEY (WORKER_REF_ID)

                REFERENCES Worker(WORKER_ID)

            ON DELETE CASCADE

);


SELECT * FROM Bonus

INSERT INTO Bonus

        (WORKER_REF_ID, BONUS_AMOUNT, BONUS_DATE) VALUES

                (005, 5000, '2024-02-20'),

                (002, 3000, '2024-03-11'),

                (003, 4000, '2024-02-20'),

                (005, 4500, '2024-02-20'),

                (002, 3500, '2024-03-11');



CREATE TABLE Title (
```

```
        WORKER_REF_ID INT,
        WORKER_TITLE CHAR(25),
        AFFECTED_FROM DATETIME,
        FOREIGN KEY (WORKER_REF_ID)  REFERENCES Worker(WORKER_ID)  ON
DELETE CASCADE
);



INSERT INTO Title
        (WORKER_REF_ID, WORKER_TITLE, AFFECTED_FROM) VALUES
  (009, 'Manager', '2023-02-20 00:00:00'),
  (002, 'Executive', '2023-06-11 00:00:00'),
  (008, 'Executive', '2023-06-11 00:00:00'),
  (005, 'Manager', '2023-06-11 00:00:00'),
  (004, 'Asst. Manager', '2023-06-11 00:00:00'),
  (007, 'Executive', '2023-06-11 00:00:00'),
  (006, 'Lead', '2023-06-11 00:00:00'),
  (003, 'Lead', '2023-06-11 00:00:00');


  ----
  sELECT * FROM Worker
  SELECT * FROM Bonus
  SELECT * FROM Title
```

----------50 SQL QUERY----------

--Q-1. Write an SQL query to fetch “FIRST_NAME” from the Worker table using the alias name <WORKER_NAME>.

```
SELECT FIRST_NAME AS WORKER_NAME FROM Worker
```

--Q-2. Write an SQL query to fetch “FIRST_NAME” from the Worker table in upper case.

```
SELECT UPPER(FIRST_NAME)FIRST_NAME       FROM Worker
```

--Q-3. Write an SQL query to fetch unique values of DEPARTMENT from the Worker table.

```
SELECT DISTINCT DEPARTMENT FROM Worker
```

---hOW TO CHANGE COLOUMN NAME

```
EXEC SP_RENAME 'Worker.DEPARMENT','DEPARTMENT','COLUMN'
```

--Q-4. Write an SQL query to print the first three characters of FIRST_NAME from the Worker table.

```
SELECT FIRST_NAME     FROM Worker
```

```
SELECT SUBSTRING(FIRST_NAME,1,3) AS FIRST_NAME FROM Worker
```

--Q-5. Write an SQL query to find the position of the alphabet (‘a’) in the first name column ‘Amitabh’ from the Worker table.

```
--SELECT FIRST_NAME     FROM Worker WHERE FIRST_NAME='Amitabh'

SELECT CHARINDEX('n',FIRST_NAME) AS NAME FROM Worker WHERE FIRST_NAME='Monika'
```

--Q-6. Write an SQL query to print the FIRST_NAME from the Worker table after removing

--white spaces from the right side.

```
SELECT RTRIM(FIRST_NAME)FIRST_NAME     FROM Worker
```

--Q-7. Write an SQL query to print the FIRST_NAME from the Worker table after removing

--white spaces from the LEFT side.

```
SELECT LTRIM(FIRST_NAME)FIRST_NAME     FROM Worker
```

--Q-8. Write an SQL query that fetches the unique values of DEPARTMENT from

--the Worker table and prints its length.

```
SELECT DISTINCT DEPARTMENT,LEN(DEPARTMENT) AS LENGTH FROM Worker
```

--Q-9. Write an SQL query to print the FIRST_NAME from the Worker table after replacing ‘a’ with ‘A’.

```
SELECT REPLACE(FIRST_NAME,'a','A')FIRST_NAME FROM Worker
```

--Q-10. Write an SQL query to print the FIRST_NAME and LAST_NAME from the Worker table into a single column

--COMPLETE_NAME. A space char should separate them.

```
SELECT (FIRST_NAME+LAST_NAME) AS COMPLETE_NAME FROM Worker
```

--Q-11. Write an SQL query to print all Worker details from the Worker table order by FIRST_NAME Ascending.

```
SELECT * FROM Worker ORDER BY FIRST_NAME ASC
```

--Q-12. Write an SQL query to print all Worker details from the Worker table order by FIRST_NAME Ascending and DEPARTMENT Descending.

```
SELECT * FROM Worker ORDER BY FIRST_NAME ASC,DEPARTMENT DESC
```

---Q-13. Write an SQL query to print details for Workers with the first names “Vipul” and “Satish” from the Worker table.

```
SELECT * FROM Worker WHERE FIRST_NAME IN('VIPUL','SATISH','CODER')
```

----Q-14. Write an SQL query to print details of workers excluding first names, “Vipul” and “Satish” from the Worker table.

```
SELECT * FROM Worker WHERE FIRST_NAME NOT IN('VIPUL','SATISH')
```

--Q-15. Write an SQL query to print details of Workers with DEPARTMENT name as “Admin”.

```
SELECT * FROM Worker WHERE DEPARTMENT='ADMIN'
```

```
SELECT * FROM Worker WHERE DEPARTMENT LIKE 'ADMIN%'
```

--Q-16. Write an SQL query to print details of the Workers whose FIRST_NAME contains ‘a’.

```
SELECT * FROM Worker WHERE FIRST_NAME LIKE '%a%'
```

--Q-17. Write an SQL query to print details of the Workers whose FIRST_NAME ends with ‘a’.

```
SELECT * FROM Worker WHERE FIRST_NAME LIKE '%A'
```

--Q-18. Write an SQL query to print details of the Workers whose

--FIRST_NAME ends with ‘h’ and contains six alphabets.

```
SELECT * FROM Worker WHERE FIRST_NAME LIKE '_____h'
```

--Q-19. Write an SQL query to print details of the Workers whose SALARY lies between 100000 and 500000.

```
SELECT * FROM Worker WHERE SALARY BETWEEN 100000 and 500000
```

--Q-20. Write an SQL query to print details of the Workers who joined in Feb 2021.

```
SELECT * FROM Worker WHERE YEAR(JOINING_DATE)=2021 AND
MONTH(JOINING_DATE)=2
```

--Q-21. Write an SQL query to fetch the count of employees working in the department ‘Admin’.

```
SELECT COUNT(*)AS 'count of employees' FROM Worker WHERE
DEPARTMENT='ADMIN'
```

--Q-22. Write an SQL query to fetch worker names with salaries >= 50000 and <= 100000.

```
SELECT FIRST_NAME,SALARY FROM Worker WHERE SALARY >=50000 AND
SALARY<=100000
```

--Q-23. Write an SQL query to fetch the number of workers for each department in descending order.

SELECT DEPARTMENT,COUNT(WORKER_ID)AS No_Of_Workers FROM Worker GROUP BY DEPARTMENT

ORDER BY No_Of_Workers DESC

Explanation:

1. **SELECT DEPARTMENT, COUNT(WORKER_ID) AS No_Of_Workers**
    - **SELECT DEPARTMENT**: Specifies the column from which data will be retrieved.
    - **COUNT(WORKER_ID) AS No_Of_Workers**: Counts the number of **WORKER_ID** entries for each **DEPARTMENT** and aliases the count as **No_Of_Workers**.
2. **FROM Worker**
    - Specifies the table (**Worker**) from which the data will be retrieved.
3. **GROUP BY DEPARTMENT**
    - Groups the result set by the **DEPARTMENT** column. This means the query will count the number of workers for each unique department.
4. **ORDER BY No_Of_Workers DESC**
    - Orders the result set by the **No_Of_Workers** in descending order. This will list the departments with the highest number of workers first.

Example:

Let's say you have the following **Worker** table:

DEPARTMENT	No_Of_Workers
IT	3
HR	3
Sales	1

--Q-24. Write an SQL query to print details of the Workers who are also Managers.

SELECT W.FIRST_NAME,T.WORKER_TITLE FROM Worker AS W

INNER JOIN Title AS T

ON W.WORKER_ID=T.WORKER_REF_ID AND T.WORKER_TITLE IN('Manager')

Explanation:

1. Tables and Aliases:

- **Worker AS W**:
  - This aliases the **Worker** table as **W**. The alias **W** is used to reference the **Worker** table in the query.
- **Title AS T**:
  - This aliases the **Title** table as **T**. The alias **T** is used to reference the **Title** table in the query.

2. JOIN Condition:

```
INNER JOIN Title AS T
ON W.WORKER_ID = T.WORKER_REF_ID
```

- **INNER JOIN**:
  - This joins the **Worker** table (**W**) with the **Title** table (**T**) based on the condition specified.
- **ON W.WORKER_ID = T.WORKER_REF_ID**:
  - This specifies the join condition. It joins rows from **Worker** and **Title** where the **WORKER_ID** in **Worker** matches the **WORKER_REF_ID** in **Title**.

3. Filtering Condition:

```
AND T.WORKER_TITLE IN ('Manager')
```

- **AND T.WORKER_TITLE IN ('Manager')**:
  - This is a filtering condition that restricts the result set to only include rows where the **WORKER_TITLE** in the **Title** table is 'Manager'.

4. SELECT Statement:

```
SELECT W.FIRST_NAME, T.WORKER_TITLE
```

- **SELECT W.FIRST_NAME, T.WORKER_TITLE**:

- This selects the **FIRST_NAME** from the **Worker** table and **WORKER_TITLE** from the **Title** table.

Overall Query:

The query is selecting the **FIRST_NAME** from the **Worker** table and the **WORKER_TITLE** from the **Title** table, but only for workers who have a title of 'Manager'.

When you execute the query, it will return:

FIRST_NAME	WORKER_TITLE
John	Manager
Emily	Manager

--Q-25. Write an SQL query to fetch duplicate records having matching data in some fields of a table.

```
SELECT WORKER_TITLE,AFFECTED_FROM,COUNT(*) AS 'COUNT' FROM Title
GROUP BY WORKER_TITLE,AFFECTED_FROM

HAVING COUNT(*)>1
```

Explanation:

1. **SELECT WORKER_TITLE, AFFECTED_FROM, COUNT(*) AS 'COUNT'**

- **SELECT**: This keyword is used to select specific columns from a table.
- **WORKER_TITLE, AFFECTED_FROM**: These are the columns selected.
- **COUNT(*) AS 'COUNT'**: This counts the number of occurrences of each combination of **WORKER_TITLE** and **AFFECTED_FROM** and aliases the result as **COUNT**.

2. **FROM Title**

- This specifies that the data is being selected from the **Title** table.

3. **GROUP BY WORKER_TITLE, AFFECTED_FROM**

- The **GROUP BY** clause is used to group rows that have the same values in specified columns (in this case, **WORKER_TITLE** and **AFFECTED_FROM**).
- This means that the query will count the number of occurrences of each unique combination of **WORKER_TITLE** and **AFFECTED_FROM**.

4. **HAVING COUNT(*) > 1**

- The **HAVING** clause is used to filter the results after the **GROUP BY** operation has been performed.
- **COUNT(*) > 1**: This condition filters the groups to only include those where the count of occurrences (**COUNT(*)**) is greater than 1.

Summary:

The SQL query is retrieving records from the **Title** table where:

- **WORKER_TITLE** and **AFFECTED_FROM** are selected columns.
- The query counts the number of occurrences of each unique combination of **WORKER_TITLE** and **AFFECTED_FROM**.
- It then filters the results to only include those combinations that occur more than once.

The result will be:

WORKER_TITLE	AFFECTED_FROM	COUNT
Executive	2023-06-11 00:00:00	2
Manager	2023-06-11 00:00:00	2

---Q-26. Write an SQL query to show only odd rows from a table.

```
SELECT * FROM Worker     WHERE WORKER_ID%2=1
```

--Q-27. Write an SQL query to show only even rows from a table.

```
SELECT * FROM Worker     WHERE WORKER_ID%2=0
```

--Q-28. Write an SQL query to clone a new table from another table.

```
SELECT * INTO WorkerClone FROM Worker
```

--Q-29. Write an SQL query to fetch intersecting records of two tables.

```
SELECT * FROM Worker INTERSECT (SELECT * FROM WorkerClone)
```

--Q-30. Write an SQL query to show records from one table that another table does not have.

SELECT * FROM Worker MINUS SELECT * FROM Title

MS SQL Server does not support the **MINUS** operator. Instead, you can achieve the same result using the **EXCEPT** operator.

```
SELECT * FROM Worker EXCEPT SELECT * FROM Title;
```

--Q-31. Write an SQL query to show the current date and time.

SELECT GETDATE()

---Q-32. Write an SQL query to show the top n (say 10) records of a table.

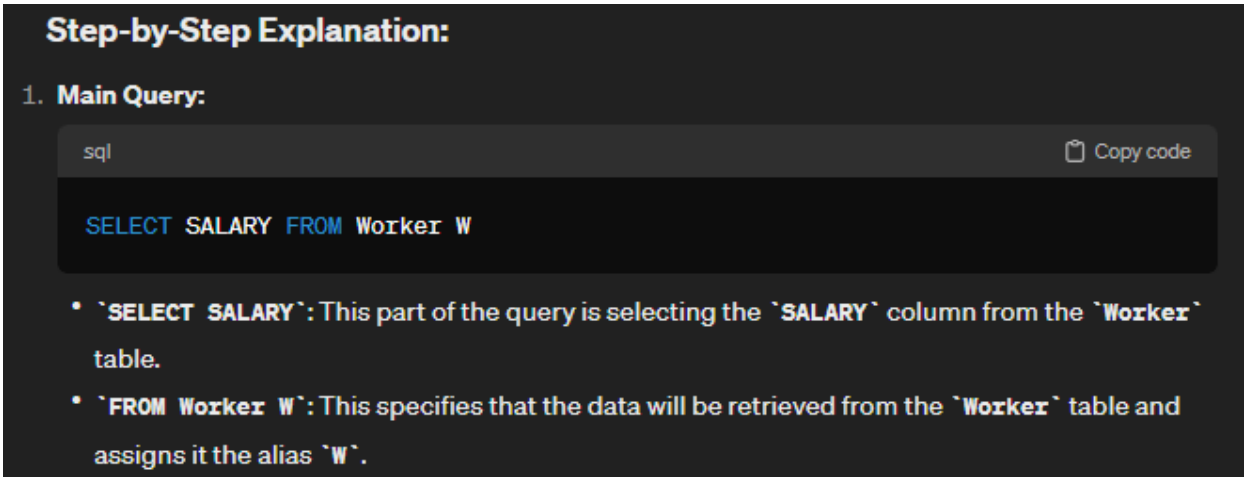
SELECT TOP 2 * FROM Worker ORDER BY SALARY DESC

--Q-33. Write an SQL query to determine the nth (say n=5) highest salary from a table.

```
SELECT TOP 13 SALARY
FROM
(
SELECT DISTINCT TOP 5 SALARY FROM Worker ORDER BY SALARY DESC
) AS SALARY
ORDER BY SALARY ASC
```

--Q-34. Write an SQL query to determine the 5th highest salary without using the TOP or limit method.

```
SELECT SALARY FROM Worker W
WHERE 5=(
SELECT COUNT(DISTINCT(W2.SALARY)) FROM Worker W2 WHERE
W2.SALARY>=W.SALARY
)
```

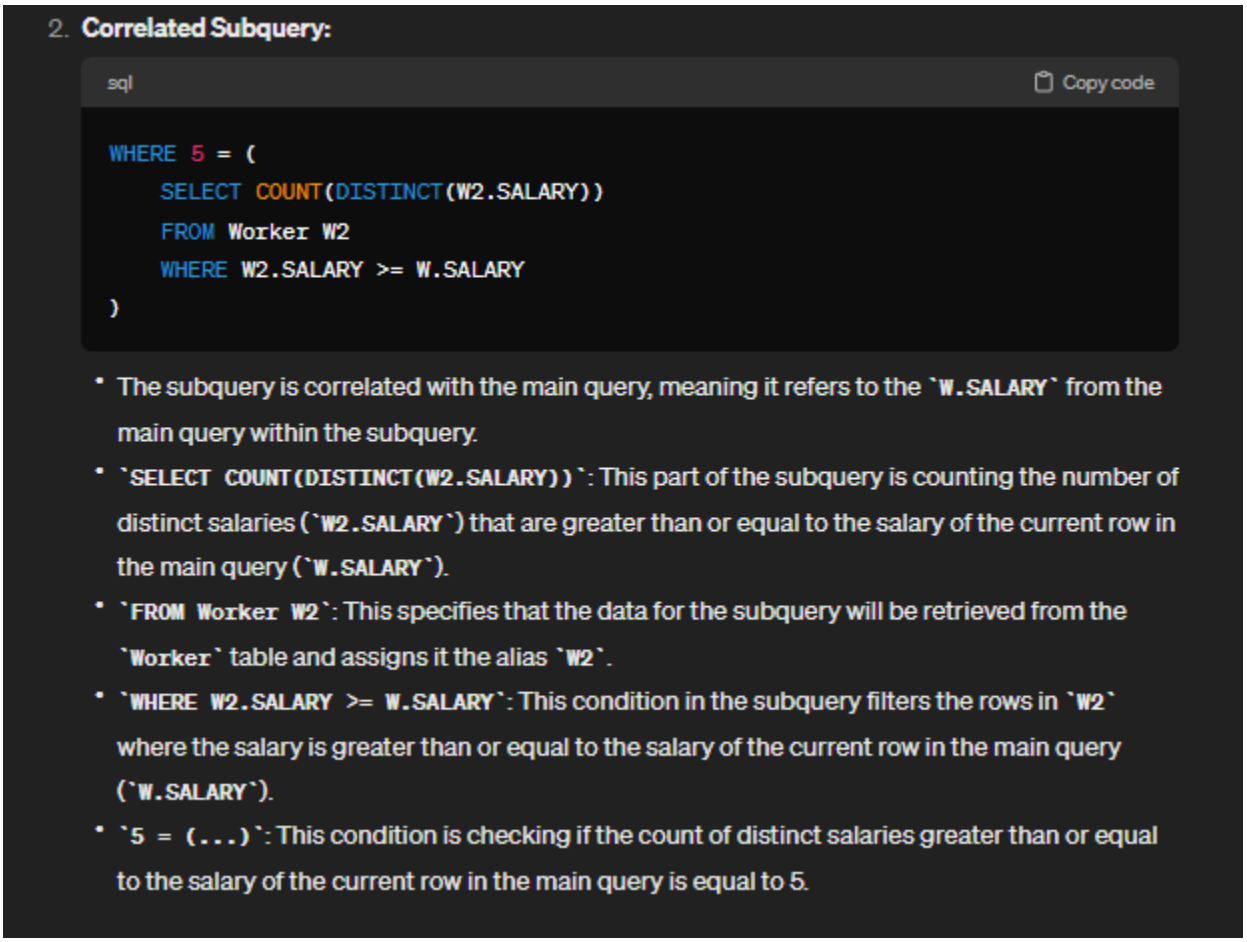


Step-by-Step Explanation:

1. **Main Query:**

```sql
SELECT SALARY FROM Worker W
```

- `SELECT SALARY`: This part of the query is selecting the `SALARY` column from the `Worker` table.
- `FROM Worker W`: This specifies that the data will be retrieved from the `Worker` table and assigns it the alias `W`.



2. **Correlated Subquery:**

```sql
WHERE 5 = (
    SELECT COUNT(DISTINCT(W2.SALARY))
    FROM Worker W2
    WHERE W2.SALARY >= W.SALARY
)
```

- The subquery is correlated with the main query, meaning it refers to the `W.SALARY` from the main query within the subquery.
- `SELECT COUNT(DISTINCT(W2.SALARY))`: This part of the subquery is counting the number of distinct salaries (`W2.SALARY`) that are greater than or equal to the salary of the current row in the main query (`W.SALARY`).
- `FROM Worker W2`: This specifies that the data for the subquery will be retrieved from the `Worker` table and assigns it the alias `W2`.
- `WHERE W2.SALARY >= W.SALARY`: This condition in the subquery filters the rows in `W2` where the salary is greater than or equal to the salary of the current row in the main query (`W.SALARY`).
- `5 = (...)`: This condition is checking if the count of distinct salaries greater than or equal to the salary of the current row in the main query is equal to 5.

----Q-35. Write an SQL query to fetch the list of employees with the same salary.

SELECT DISTINCT W.WORKER_ID,W.FIRST_NAME,W.SALARY FROM Worker W,Worker W1

WHERE W.SALARY=W1.SALARY AND W.WORKER_ID!=W1.WORKER_ID

Explanation:

1. Tables and Alias:

FROM Worker W, Worker W1

- The query uses two instances of the **Worker** table, named **W** and **W1**, to compare the salaries of different workers.

2. SELECT Clause:

SELECT DISTINCT W.WORKER_ID, W.FIRST_NAME, W.SALARY

- The query selects the **WORKER_ID**, **FIRST_NAME**, and **SALARY** columns from the **Worker** table.

3. WHERE Clause:

WHERE W.SALARY = W1.SALARY AND W.WORKER_ID != W1.WORKER_ID

- **W.SALARY = W1.SALARY**: This condition ensures that the two workers being compared have the same salary.
- **W.WORKER_ID != W1.WORKER_ID**: This condition ensures that the workers being compared are not the same worker.

4. DISTINCT Keyword:

SELECT DISTINCT

- The **DISTINCT** keyword is used to ensure that the resulting pairs of workers are unique and not duplicated.

Summary:

The SQL query retrieves distinct pairs of workers (**W** and **W1**) who have the same salary but are different workers. It then selects and displays their **WORKER_ID**, **FIRST_NAME**, and **SALARY**.

```
--Q-36. Write an SQL query to show the second-highest salary from a table.

SELECT MAX(SALARY)'second-highest salary' FROM Worker

WHERE SALARY NOT IN(SELECT MAX(SALARY) FROM Worker)
```

Step-by-Step Explanation:

1. **Subquery - Finding the Maximum Salary**:

SELECT MAX(SALARY) FROM Worker

This subquery finds the maximum salary from the **Worker** table.

2. **Main Query - Finding the Second-Highest Salary**:

SELECT MAX(SALARY) AS 'second-highest salary' FROM Worker WHERE SALARY NOT IN (SELECT MAX(SALARY) FROM Worker);

- **WHERE SALARY NOT IN (SELECT MAX(SALARY) FROM Worker)**:
  - This part of the query filters out the maximum salary from the main table.
  - The **NOT IN** clause excludes the maximum salary (i.e., the highest salary) obtained from the subquery.
- **SELECT MAX(SALARY) AS 'second-highest salary'**:
  - This part of the query then selects the maximum salary from the filtered results.
  - The **AS 'second-highest salary'** alias the result column to 'second-highest salary' for clarity.

--Q-37. Write an SQL query to show one row twice in the results from a table.

SELECT FIRST_NAME,DEPARTMENT FROM Worker W WHERE W.DEPARTMENT='HR'

UNION ALL

SELECT FIRST_NAME,DEPARTMENT FROM Worker W WHERE W.DEPARTMENT='HR'

SELECT * FROM Worker UNION ALL SELECT * FROM Worker

--Q-38. Write an SQL query to fetch intersecting records of two tables.

SELECT * FROM Worker INTERSECT (SELECT * FROM WorkerClone)

--Q-39. Write an SQL query to fetch the first 50% of records from a table.

```
SELECT * FROM Worker
WHERE WORKER_ID<=(SELECT COUNT(WORKER_ID)/2 FROM Worker)
```

---Q-40. Write an SQL query to fetch the departments that have less than five people in them.

```
SELECT DEPARTMENT ,COUNT(WORKER_ID)AS 'No_OF_Worker' FROM Worker
GROUP BY DEPARTMENT HAVING COUNT(WORKER_ID)<5;
```

--Q-41. Write an SQL query to show all departments along with the number of people in there.

```
SELECT DEPARTMENT, COUNT(DEPARTMENT) AS 'No_OF_Worker' FROM Worker GROUP BY DEPARTMENT
```

--Q-42. Write an SQL query to show the last record from a table.

```
SELECT * FROM Worker WHERE WORKER_ID=(SELECT MAX(WORKER_ID)FROM Worker)
```

--Q-43. Write an SQL query to fetch the first row of a table.

```
SELECT * FROM Worker WHERE WORKER_ID=(SELECT MIN(WORKER_ID)FROM Worker)
```

--Q-44. Write an SQL query to fetch the last five records from a table.

```
SELECT TOP 5 * FROM Worker ORDER BY WORKER_ID DESC
```

--Q-45. Write an SQL query to print the names of employees having the highest salary in each department.

```
SELECT T.DEPARTMENT,T.FIRST_NAME,T.SALARY FROM(SELECT MAX(SALARY)AS
TOTALSALARY, DEPARTMENT FROM Worker

GROUP BY DEPARTMENT)AS TEMPNEW

INNER JOIN Worker T ON TEMPNEW.DEPARTMENT=T.DEPARTMENT AND
TEMPNEW.TOTALSALARY=T.SALARY
```

```
--Q-46. Write an SQL query to fetch three max salaries from a table.

SELECT DISTINCT SALARY FROM Worker W WHERE 3>=(SELECT COUNT(DISTINCT
SALARY)FROM Worker W1 WHERE

W.SALARY<=W1.SALARY) ORDER BY W.SALARY DESC
```

Explanation:

1. **Main Query:**

SELECT DISTINCT SALARY FROM Worker W

This part of the query retrieves the distinct salaries from the **Worker** table and aliases the table as **W**.

2. **Subquery:**

SELECT COUNT(DISTINCT SALARY) FROM Worker W1 WHERE W.SALARY <= W1.SALARY

- The subquery counts the number of distinct salaries that are less than or equal to the salary of each worker from the main query (**W.SALARY**).
- This subquery is correlated with the main query, meaning it is executed for each row in the main query.

3. **WHERE Clause:**

WHERE 3 >= (SELECT COUNT(DISTINCT SALARY) FROM Worker W1 WHERE W.SALARY <= W1.SALARY)

- This condition filters the rows where the count of distinct salaries that are less than or equal to the salary of the current worker is less than or equal to 3.

- Essentially, this will retrieve the salaries of the top 3 highest earners in the **Worker** table.

4. **ORDER BY Clause:**

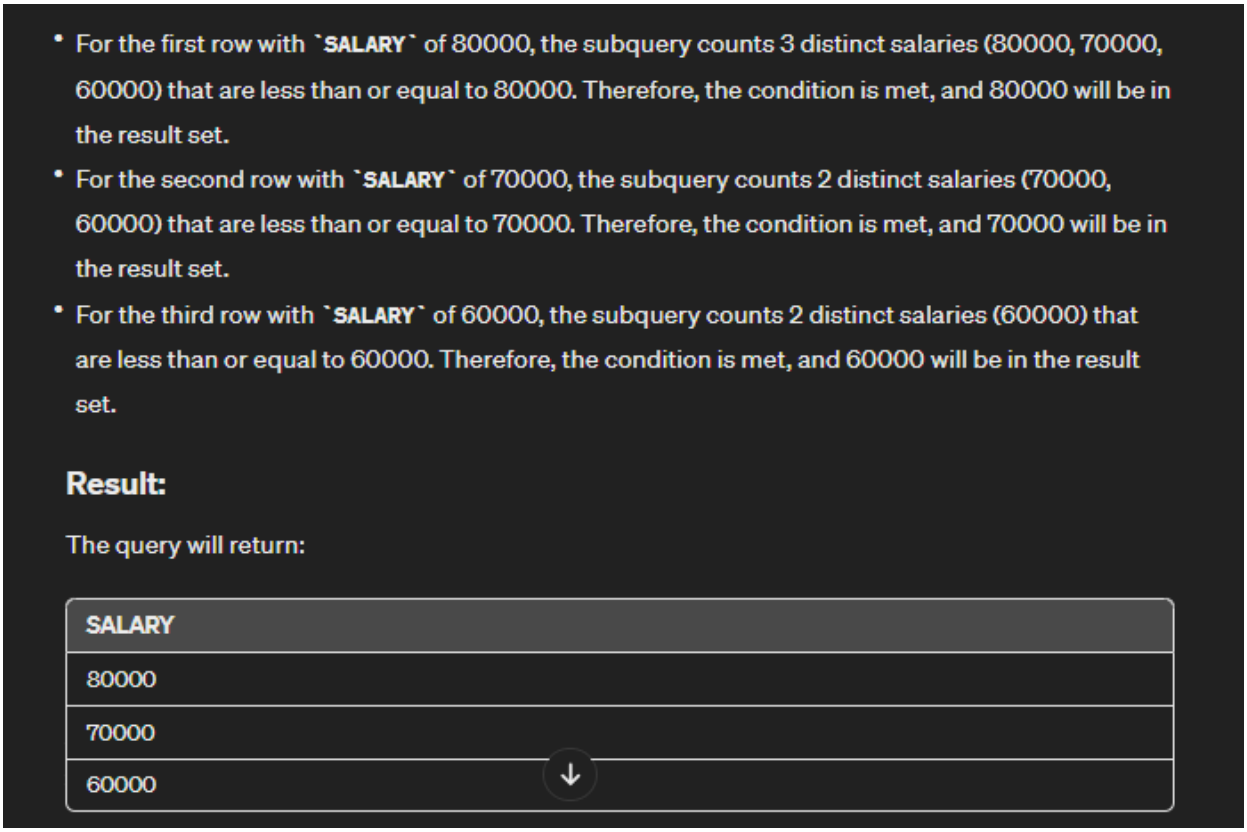
ORDER BY W.SALARY DESC;

- This orders the result set by salary in descending order.

Example:

Let's consider a simple `Worker` table:

ID	SALARY
1	50000
2	60000
3	50000
4	70000
5	60000
6	80000



--Q-47. Write an SQL query to fetch three min salaries from a table.

SELECT DISTINCT SALARY FROM Worker W WHERE 3>=(SELECT COUNT(DISTINCT SALARY)FROM Worker W1 WHERE

W.SALARY>=W1.SALARY) ORDER BY W.SALARY DESC

Explanation:

Main Query:

SELECT DISTINCT SALARY FROM Worker W

- This part of the query retrieves all distinct salary values from the **Worker** table and assigns the table an alias **W**.

Subquery:

SELECT COUNT(DISTINCT SALARY) FROM Worker W1 WHERE W.SALARY >= W1.SALARY

- This subquery counts the number of distinct salary values that are less than or equal to the salary of each record in the main query. It uses the same **Worker** table but with a different alias **W1**.

Explanation of the WHERE clause:

WHERE 3 >= (SELECT COUNT(DISTINCT SALARY) FROM Worker W1 WHERE W.SALARY >= W1.SALARY)

- The **WHERE** clause filters the records based on the count of distinct salary values.
- The condition **3 >=** indicates that we are interested in the top 3 highest salaries.

ORDER BY clause:

ORDER BY W.SALARY DESC;

- This sorts the result set in descending order based on the salary.

Explanation with an Example:

Let's assume we have the following data in the `Worker` table:

ID	SALARY
1	5000
2	3000
3	7000
4	7000
5	6000
6	3000

Step 1: For each record in the `Worker` table, the subquery counts how many distinct salaries are less than or equal to its own salary.

- For `ID = 1` (Salary = 5000), there are 4 distinct salaries less than or equal to 5000.
- For `ID = 2` (Salary = 3000), there are 2 distinct salaries less than or equal to 3000.
- For `ID = 3` (Salary = 7000), there are 5 distinct salaries less than or equal to 7000.
- For `ID = 4` (Salary = 7000), there are 5 distinct salaries less than or equal to 7000.
- For `ID = 5` (Salary = 6000), there are 4 distinct salaries less than or equal to 6000.
- For `ID = 6` (Salary = 3000), there are 2 distinct salaries less than or equal to 3000.

Step 2: The main query filters the records to include only those where the count from the subquery is 3 or less (indicating the top 3 highest salaries).

- The records with `ID = 3, 4, and 5` will be included in the result because they have one of the top 3 distinct salaries.

Final Result:

SALARY
7000
6000
5000

--Q-48. Write an SQL query to fetch nth max salaries from a table.

SELECT DISTINCT SALARY FROM Worker W1 WHERE 8>=(SELECT COUNT(DISTINCT SALARY) FROM

Worker W2 WHERE W1.SALARY<=W2.SALARY)ORDER BY W1.SALARY DESC

--Q-49. Write an SQL query to fetch departments along with the total salaries paid for each of them

SELECT DEPARTMENT, SUM(SALARY)AS TOT_SALARY FROM Worker GROUP BY DEPARTMENT

--Q-50. Write an SQL query to fetch the names of workers who earn the highest salary.

SELECT DISTINCT FIRST_NAME,SALARY FROM Worker WHERE SALARY=(SELECT MAX(SALARY) FROM Worker)