SQL INTERVIEW QUESTIONS

SQL interview questions!

1. Find the second-highest salary in a table without using LIMIT or TOP.

2. Write a SQL query to find all employees who earn more than their managers.

3. Find the duplicate rows in a table without using GROUP BY.

4. Write a SQL query to find the top 10% of earners in a table.

5. Find the cumulative sum of a column in a table.

6. Write a SQL query to find all employees who have never taken a leave.

7. Find the difference between the current row and the next row in a table.

8. Write a SQL query to find all departments with more than one employee.

9. Find the maximum value of a column for each group without using GROUP BY.

10. Write a SQL query to find all employees who have taken more than 3 leaves in a month.

11.Write a query to find the second highest salary in an employee table.

12. How do you identify duplicate rows in a table and delete them?

13.Write a query to calculate the percentage of sales for each product.

14. How do you retrieve the top N records for each category in a dataset?

15. Write a query to join two tables and fetch records that exist in one table but not the other.

16. Write a SQL query to find the third highest salary from an employee table with the following columns: EID, ESalary.

17. Create a SQL procedure using ESalary as a parameter that selects all EIDs from the Employee table where ESalary is less than 50,000.

18. For the Employee table (with columns EID and ESalary), retrieve all EIDs with odd salaries and join this with another table, empdetails (with columns EID and EDOB), to obtain EDOB.

19. How would you use the LEAD or LAG function in SQL to compare week-over-week data?

20.Types of Joins in SQL, explain each one of them with examples.

21.State difference between UNION and UNION ALL functions in SQL.

22.What is CTEs and state the use-case of it.

23.What is TEMPORARY TABLE and how it is different from CTEs?

24.What's the difference between DELETE, DROP and TRUNCATE.

25. What are window functions, and how do they differ from aggregate functions? Can you give a use case?

26. Explain indexing. When would an index potentially reduce performance, and how would you approach indexing strategy for a large dataset?

27. Write a query to retrieve customers who have made purchases in the last 30 days but did not purchase anything in the previous 30 days.

28. Given a table of transactions, find the top 3 most purchased products for each category.

29. How would you identify duplicate records in a large dataset, and how would you remove only the duplicates, retaining the first occurrence?

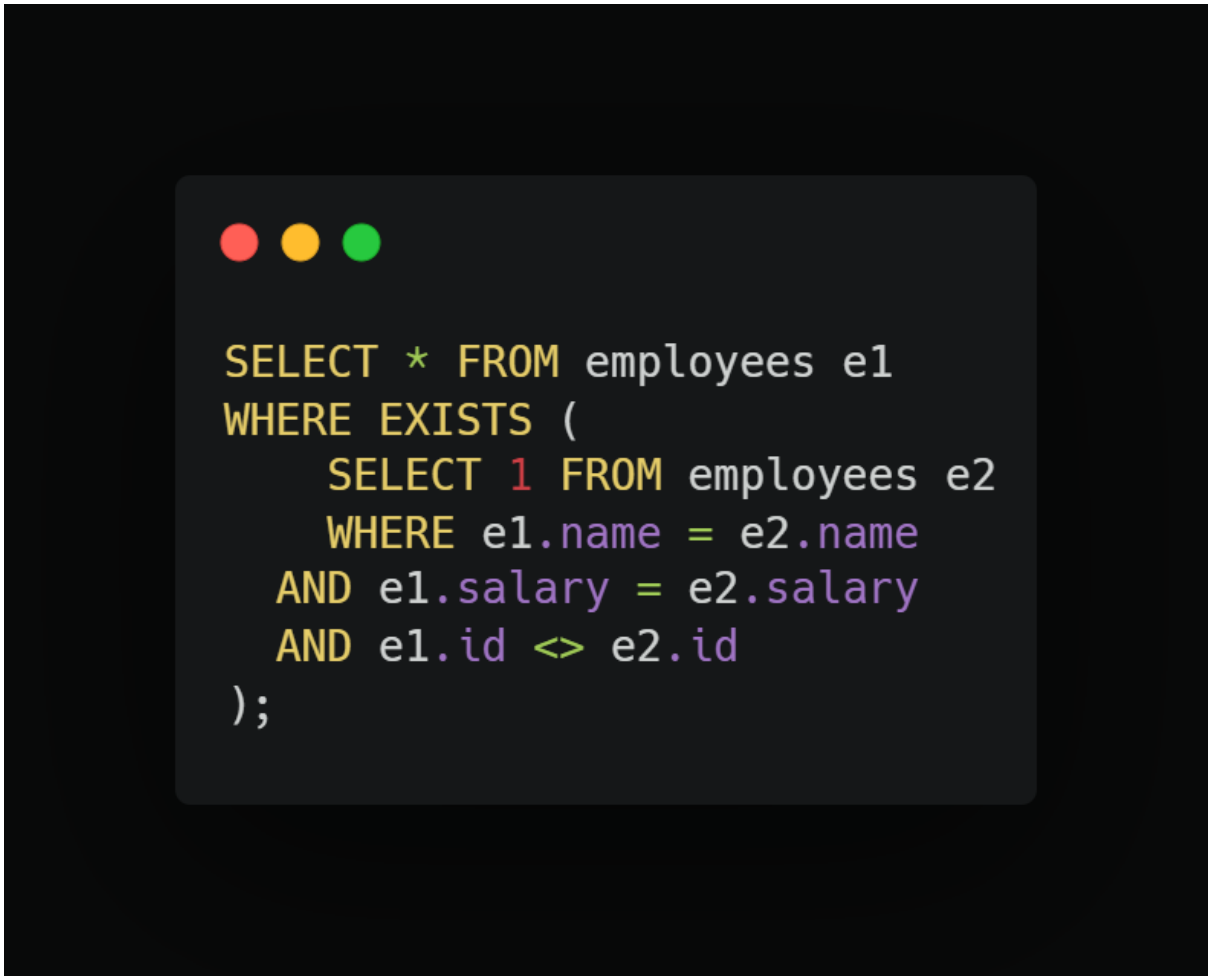
1. Find the second-highest salary in a table without using LIMIT or TOP.

```sql
SELECT MAX(salary) FROM employees
WHERE salary < (SELECT MAX(salary) FROM
employees);
```

2. Write a SQL query to find all employees who earn more than their managers.

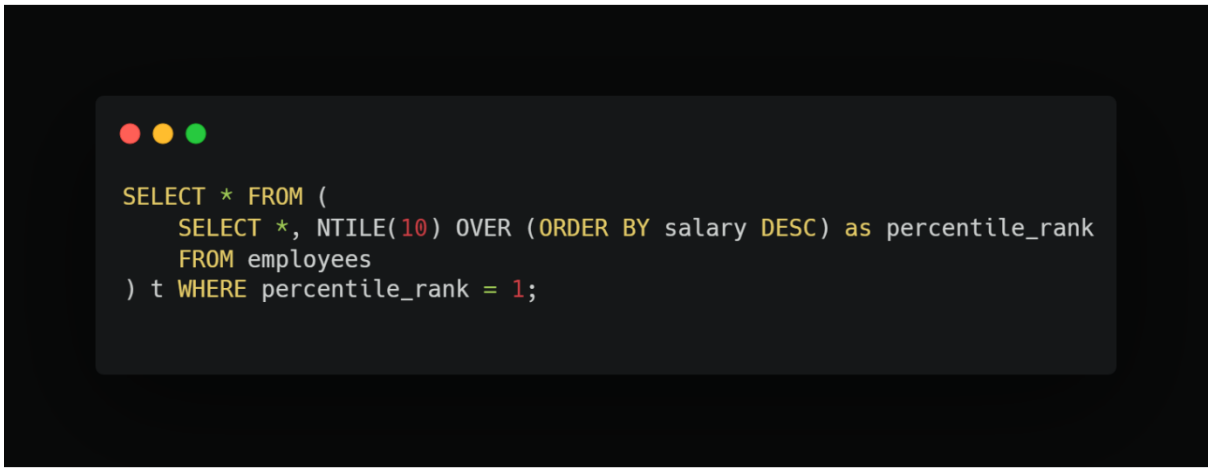
```sql
SELECT e1.* FROM employees e1
JOIN employees e2 ON e1.manager_id = e2.id
WHERE e1.salary > e2.salary;
```

3. Find the duplicate rows in a table without using GROUP BY.



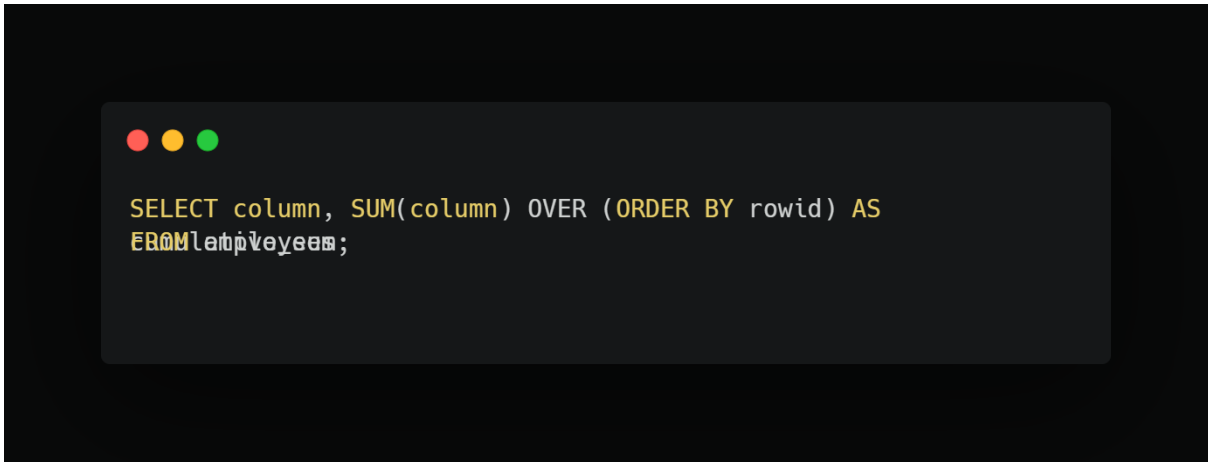
```sql
SELECT * FROM employees e1
WHERE EXISTS (
    SELECT 1 FROM employees e2
    WHERE e1.name = e2.name
  AND e1.salary = e2.salary
  AND e1.id <> e2.id
);
```

4. Write a SQL query to find the top 10% of earners in a table.

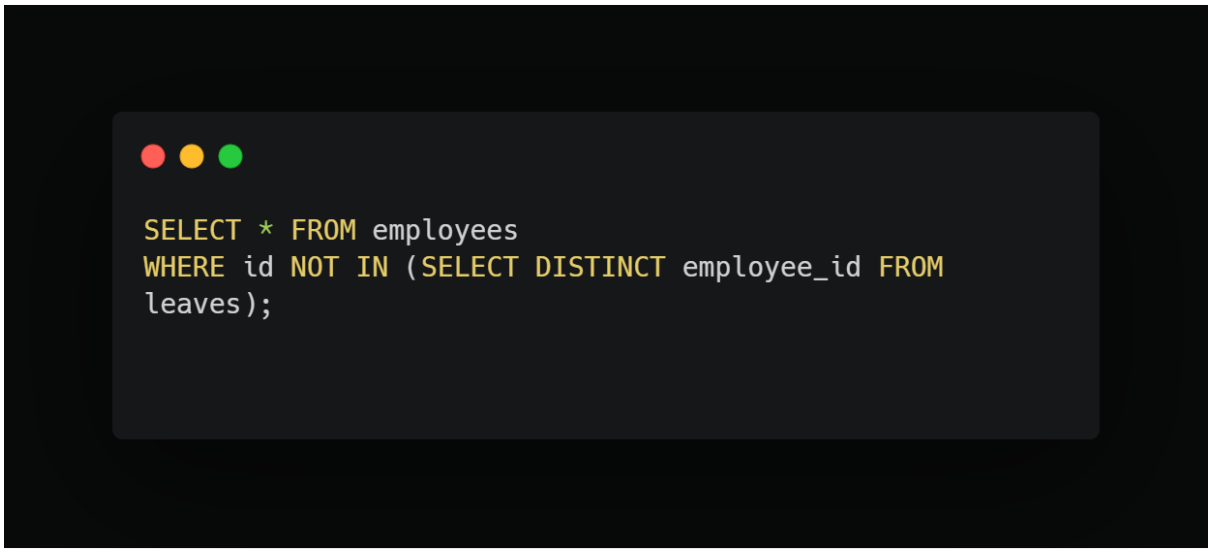


```sql
SELECT * FROM (
    SELECT *, NTILE(10) OVER (ORDER BY salary DESC) as percentile_rank
    FROM employees
) t WHERE percentile_rank = 1;
```

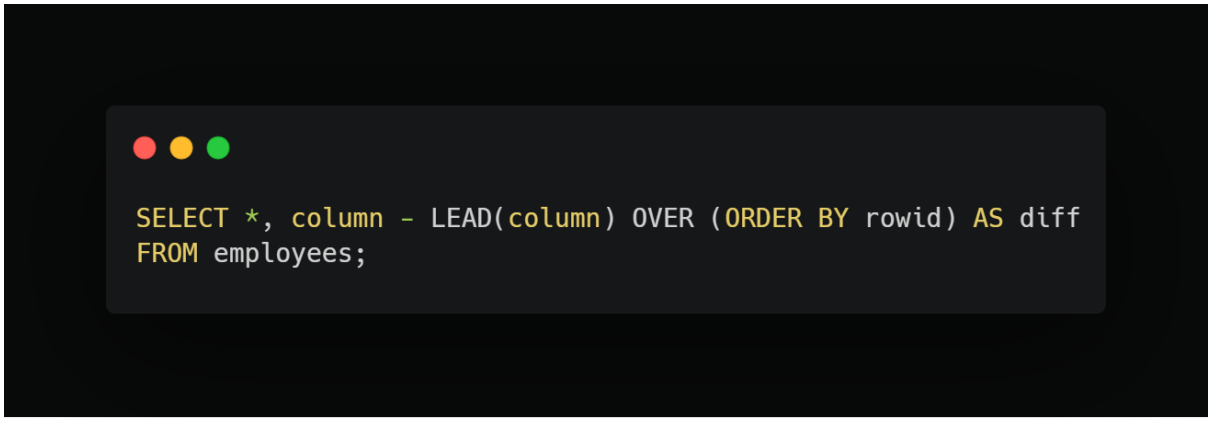
5. Find the cumulative sum of a column in a table.



6. Write a SQL query to find all employees who have never taken a leave.



7. Find the difference between the current row and the next row in a table.



8. Write a SQL query to find all departments with more than one employee.

```sql
SELECT department
FROM employees
GROUP BY department HAVING COUNT(*) >
1;
```

9. Find the maximum value of a column for each group without using GROUP BY.

```sql
SELECT * FROM (
    SELECT *, RANK() OVER (PARTITION BY group_column ORDER BY column DESC) AS rnk
    FROM employees
) t WHERE rnk = 1;
```

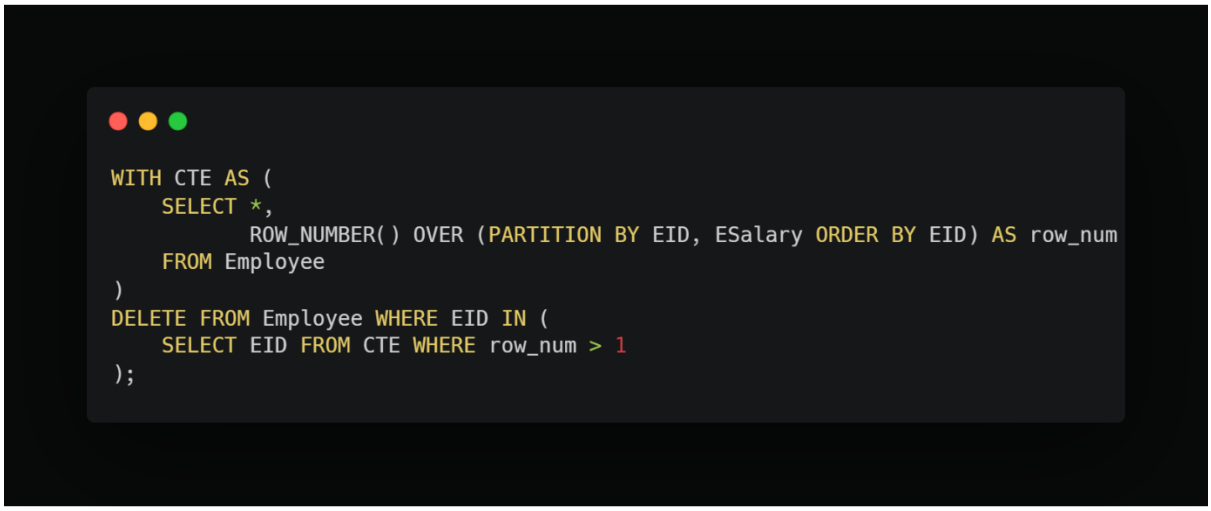
10. Write a SQL query to find all employees who have taken more than 3 leaves in a month.

```sql
SELECT employee_id FROM leaves
WHERE MONTH(leave_date) =
MONTH(CURRENT_DATE)
GROUP BY employee_id HAVING COUNT(*) > 3;
```

11.Write a query to find the second highest salary in an employee table.

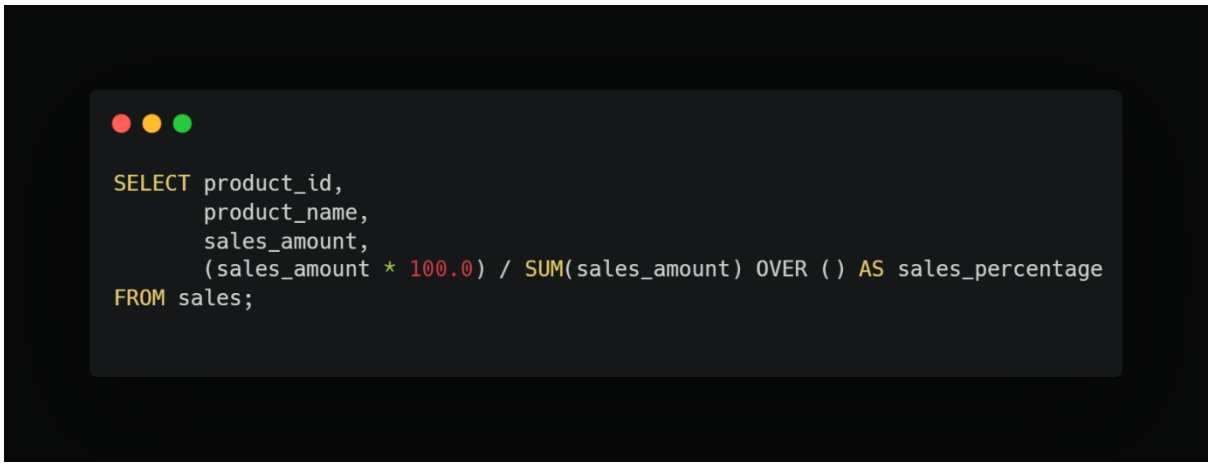
```sql
SELECT MAX(salary)
FROM employees
WHERE salary < (SELECT MAX(salary) FROM
employees);
```

12. How do you identify duplicate rows in a table and delete them?



```sql
WITH CTE AS (
    SELECT *,
           ROW_NUMBER() OVER (PARTITION BY EID, ESalary ORDER BY EID) AS row_num
    FROM Employee
)
DELETE FROM Employee WHERE EID IN (
    SELECT EID FROM CTE WHERE row_num > 1
);
```

13.Write a query to calculate the percentage of sales for each product.



```sql
SELECT product_id,
       product_name,
       sales_amount,
       (sales_amount * 100.0) / SUM(sales_amount) OVER () AS sales_percentage
FROM sales;
```

14. How do you retrieve the top N records for each category in a dataset?

```sql
SELECT * FROM (
    SELECT *,
           RANK() OVER (PARTITION BY category ORDER BY sales DESC) AS rnk
    FROM products
) t WHERE rnk <= 5;
```

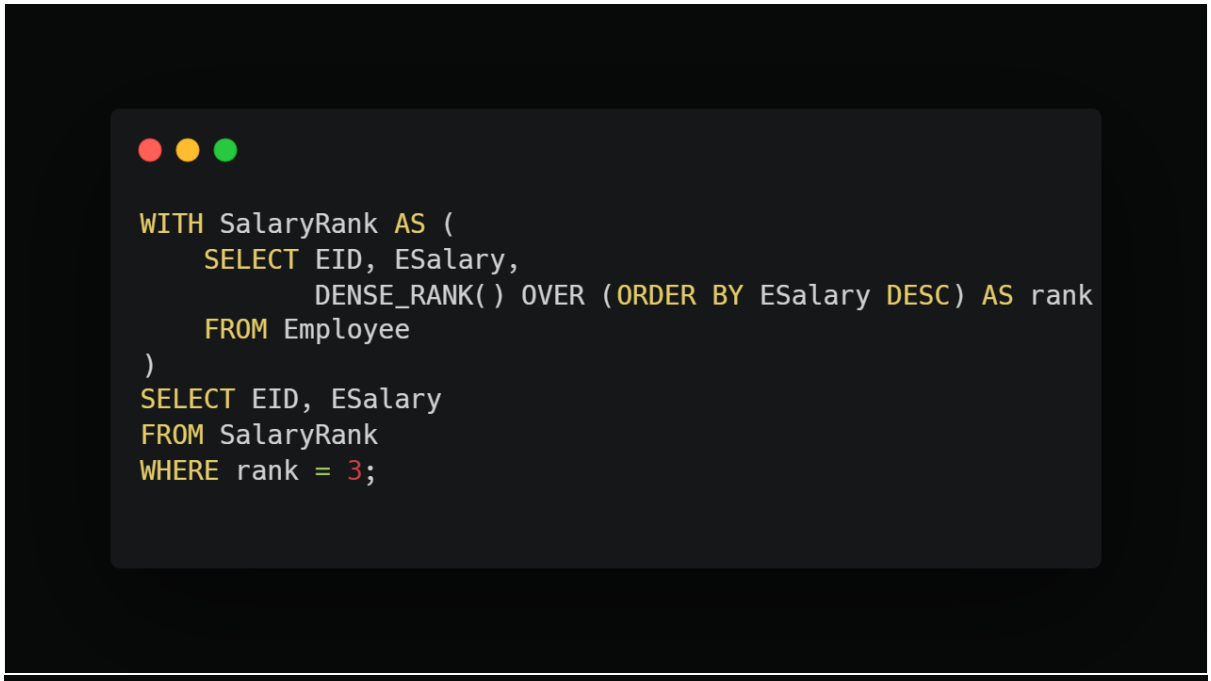
15. Write a query to join two tables and fetch records that exist in one table but not the other.

```sql
SELECT e.*
FROM employees e
LEFT JOIN department d ON e.department_id = d.id
WHERE d.id IS NULL;
```

16. Write a SQL query to find the third highest salary from an employee table with the following columns: EID, ESalary.

🚀 **When to Use?**

- Use **DENSE_RANK()** if you want to consider duplicate salaries.
- Use **ROW_NUMBER()** if you want strict ranking without duplicates affecting the position.



```sql
WITH SalaryRank AS (
    SELECT EID, ESalary,
           DENSE_RANK() OVER (ORDER BY ESalary DESC) AS rank
    FROM Employee
)
SELECT EID, ESalary
FROM SalaryRank
WHERE rank = 3;
```

```sql
WITH SalaryRank AS (
    SELECT EID, ESalary,
           ROW_NUMBER() OVER (ORDER BY ESalary DESC) AS row_num
    FROM Employee
)
SELECT EID, ESalary
FROM SalaryRank
WHERE row_num = 3;
```

17. Create a SQL procedure using ESalary as a parameter that selects all EIDs from the Employee table where ESalary is less than 50,000.

```sql
CREATE PROCEDURE GetLowSalaryEmployees(IN salary_limit INT)
BEGIN
    SELECT EID, ESalary
    FROM Employee
    WHERE ESalary < salary_limit;
END

-- EXECUTE

CALL GetLowSalaryEmployees(50000);
```

18. For the Employee table (with columns EID and ESalary), retrieve all EIDs with odd salaries and join this with another table, empdetails (with columns EID and EDOB), to obtain EDOB.

```sql
SELECT e.EID, d.EDOB
FROM Employee e
JOIN empdetails d ON e.EID = d.EID
WHERE e.ESalary % 2 <> 0;
```

19. How would you use the LEAD or LAG function in SQL to compare week-over-week data?

```sql
SELECT EID, week, ESalary,
       LAG(ESalary) OVER (PARTITION BY EID ORDER BY week) AS prev_week_salary,
       ESalary - LAG(ESalary) OVER (PARTITION BY EID ORDER BY week) AS
FROM Employee;
```

20.Types of Joins in SQL, explain each one of them with examples.

➡️ **Types of Joins in SQL**

Joins in SQL combine records from two or more tables based on a related column.

1 NNER JOIN

✅ Returns only matching records from both tables.

2 LEFT JOIN (LEFT OUTER JOIN)

✅ Returns all records from the left table and matching records from the right table. If no match, NULL is returned.

3 RIGHT JOIN (RIGHT OUTER JOIN)

✅ Returns **all records** from the **right table** and matching records from the **left table**.

4 FULL JOIN (FULL OUTER JOIN)

✅ Returns **all records** when there is a match in either table.

5 CROSS JOIN

✅ Returns **Cartesian Product** (each row from one table is combined with every row from another table).

21.State difference between UNION and UNION ALL functions in SQL.

➡️ Difference Between UNION and UNION ALL

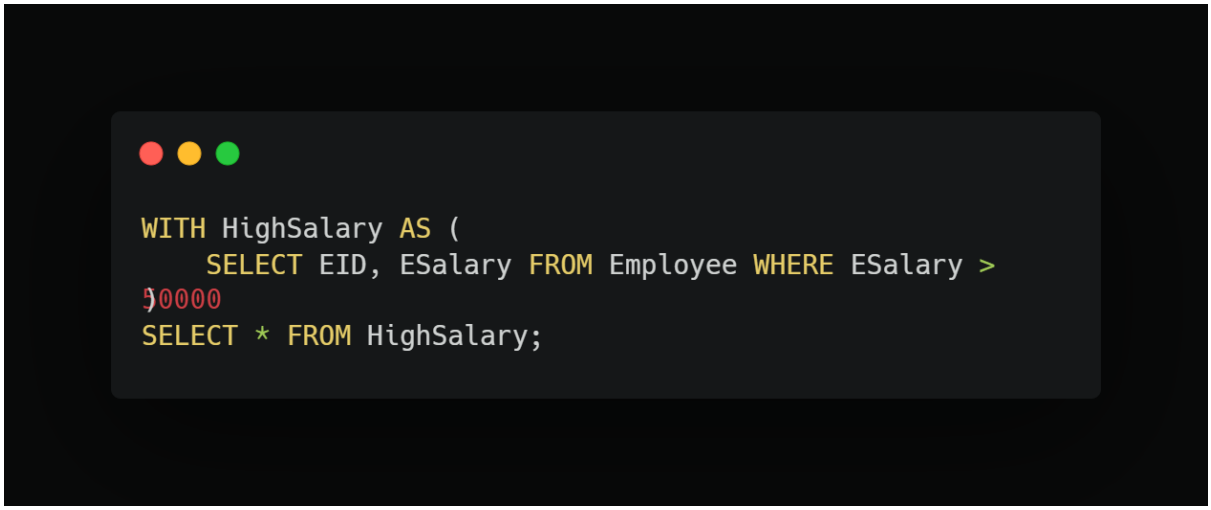
Feature	UNION	UNION ALL
Duplicates	Removes duplicates	Keeps all records
Performance	Slower (due to deduplication)	Faster
Usage	Used when unique results are needed	Used when duplicates are allowed

22.What is CTEs and state the use-case of it.

CTE (Common Table Expression) is a temporary result set used within a query.

➡️ Use-Case:

- **Improves query readability.**
- **Helps recursive queries (e.g., hierarchical data).**
- **Used for temporary filtering inside a complex query.**



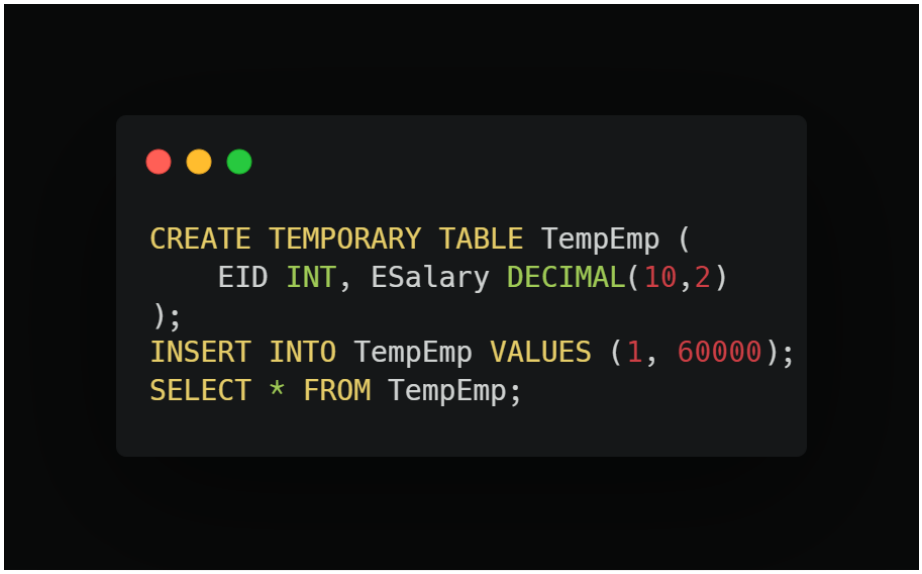
```
WITH HighSalary AS (
    SELECT EID, ESalary FROM Employee WHERE ESalary >
50000
SELECT * FROM HighSalary;
```

23.What is TEMPORARY TABLE and how it is different from CTEs?

Temporary Table: A table that exists only during a session.

➡️ Temporary Table vs. CTE

Feature	CTE	Temporary Table
Scope	Only for **one query**	Available for the **entire session**
Storage	**Memory-based**	Stored in **TempDB**
Performance	Faster for **small data**	Better for **large datasets**
Recursion Support	✅ Yes	❌ No



```
CREATE TEMPORARY TABLE TempEmp (
    EID INT, ESalary DECIMAL(10,2)
);
INSERT INTO TempEmp VALUES (1, 60000);
SELECT * FROM TempEmp;
```

24.What's the difference between DELETE, DROP and TRUNCATE.

➡️ Difference Between DELETE, DROP, and TRUNCATE

Command	Removes	Rollback Possible?	Performance	Affects Structure?
DELETE	**Specific rows**	✅ Yes (with `WHERE`)	**Slower** (row-by-row)	❌ No
TRUNCATE	**All rows**	❌ No	**Faster** (resets storage)	❌ No
DROP	**Entire table**	❌ No	**Fastest**	✅ Yes (removes table)

DELETE FROM Employee WHERE EID = 5; -- Removes specific record

TRUNCATE TABLE Employee; -- Removes all data but keeps structure

DROP TABLE Employee; -- Deletes entire table including structure

25. What are window functions, and how do they differ from aggregate functions? Can you give a use case?

1️⃣ Window Functions vs. Aggregate Functions

✅ **Window Functions** perform calculations **across a specific window (subset) of rows** related to the current row. Unlike aggregate functions, **they do not collapse multiple rows into a single row.**

✅ **Aggregate Functions** summarize multiple rows **into a single output row.**

Feature	Window Functions	Aggregate Functions
Output Rows	Returns multiple rows	Returns a single row
Grouping Required?	No (`OVER` clause used)	Yes (`GROUP BY` required)
Example Function	`RANK()`, `ROW_NUMBER()`	`SUM()`, `COUNT()`
Example Query	`SUM(Salary) OVER (PARTITION BY DeptID)`	`SUM(Salary) GROUP BY DeptID`

```
SELECT EID, ESalary, DeptID,
       RANK() OVER (PARTITION BY DeptID ORDER BY ESalary DESC) AS rank
FROM Employee;
```

26. Explain indexing. When would an index potentially reduce performance, and how would you approach indexing strategy for a large dataset?

Indexing in SQL

✅ **Indexing** improves the speed of data retrieval by creating a **lookup table** for faster searches.

- Common types: **B-Tree, Hash, Full-Text, Composite Index**

📌 **When Index Reduces Performance**

1. **Frequent INSERT/UPDATE/DELETE operations** → Overhead due to index maintenance.
2. **Small tables** → Full table scans may be faster.
3. **Too many indexes** → High storage and query optimization costs.

📌 **Indexing Strategy for Large Datasets**

- **Use indexes on frequently searched columns** (e.g., WHERE, JOIN, ORDER BY).
- **Avoid indexing columns with many duplicate values** (low cardinality).
- **Use Composite Index for multiple filters** (WHERE col1 = X AND col2 = Y).
- **Monitor with EXPLAIN ANALYZE** to check index effectiveness.

✅ **Types of Indexes in SQL**

Indexes in SQL improve the **speed of data retrieval** operations. However, they also come with storage and performance trade-offs. Here are the main types of indexes used in SQL:

---

1 Primary Index

- **Automatically created** when defining a PRIMARY KEY in a table.
- Ensures **each row is uniquely identifiable**.
- Uses **clustered indexing** (depending on the database).

🔹 **Example:**

```
CREATE TABLE Employee (
  EID INT PRIMARY KEY,
  ESalary INT
);
```

- Here, EID is a **primary index**.

---

2 Unique Index

- Ensures **all values in a column are unique** (but allows NULL values).
- Similar to PRIMARY KEY, but a table can have **multiple** unique indexes.

🔹 **Example:**

```
CREATE UNIQUE INDEX idx_emp_email ON Employee (Email);
```

- This ensures no two employees have the same email.

---

3 Clustered Index

- **Reorders the actual table data** to match the index.
- **Only one clustered index** is allowed per table.
- Improves retrieval speed since related data is stored **physically together**.

🔹 **Example:**

```
CREATE CLUSTERED INDEX idx_emp_salary ON Employee (ESalary);
```

- Now, the table data is physically sorted by ESalary.

---

4 Non-Clustered Index

- **Stores index separately** from table data.
- A table can have **multiple** non-clustered indexes.
- Good for **search queries**, but slower than clustered indexes.

🔹 **Example:**

```
CREATE INDEX idx_emp_name ON Employee (EName);
```

- The database **stores the index separately** and points to actual data.

---

5 Composite Index (Multi-Column Index)

- An index on **multiple columns** (instead of just one).
- Useful for queries filtering by **multiple conditions**.

🔹 **Example:**

```
CREATE INDEX idx_emp_dept_salary ON Employee (Department, ESalary);
```

- This improves queries like:

```
SELECT * FROM Employee WHERE Department = 'IT' AND ESalary > 50000;
```

---

6 Full-Text Index

- Used for **fast text searching** (instead of simple LIKE queries).
- Available in databases like **MySQL, PostgreSQL, and SQL Server**.

🔹 **Example (MySQL Full-Text Search):**

```
CREATE FULLTEXT INDEX idx_emp_desc ON Employee (Description);
```

- Enables **fast** searching like:

```
SELECT * FROM Employee WHERE MATCH(Description) AGAINST ('developer');
```

---

7 Bitmap Index (Used in Data Warehousing)

- Stores indexes as **bitmaps** (good for low-cardinality data, like Gender: Male/Female).
- Common in **OLAP databases** (Oracle, PostgreSQL).

🔹 **Example (Oracle):**

```
CREATE BITMAP INDEX idx_emp_gender ON Employee (Gender);
```

🔹 **Summary Table**

Index Type	Description
Primary Index	Auto-created on PRIMARY KEY, unique, and mostly clustered.
Unique Index	Ensures unique values in a column but allows NULL.
Clustered Index	Rearranges table data to match the index. One per table.

Index Type	Description
Non-Clustered Index	Index stored separately, multiple allowed per table.
Composite Index	Index on multiple columns, speeds up multi-condition queries.
Full-Text Index	Optimized for text-based searches.
Bitmap Index	Best for low-cardinality values, mainly for OLAP workloads.

📌 **Key Takeaways**

✓ **Clustered indexes** are best for sorting and searching **large datasets**.
✓ **Non-clustered indexes** improve searches without affecting table storage order.
✓ **Composite indexes** optimize queries using multiple columns.
✓ **Full-text indexes** are best for searching long text fields efficiently.

Let me know if you need more details! 🚀

27. Write a query to retrieve customers who have made purchases in the last 30 days but did not purchase anything in the previous 30 days.

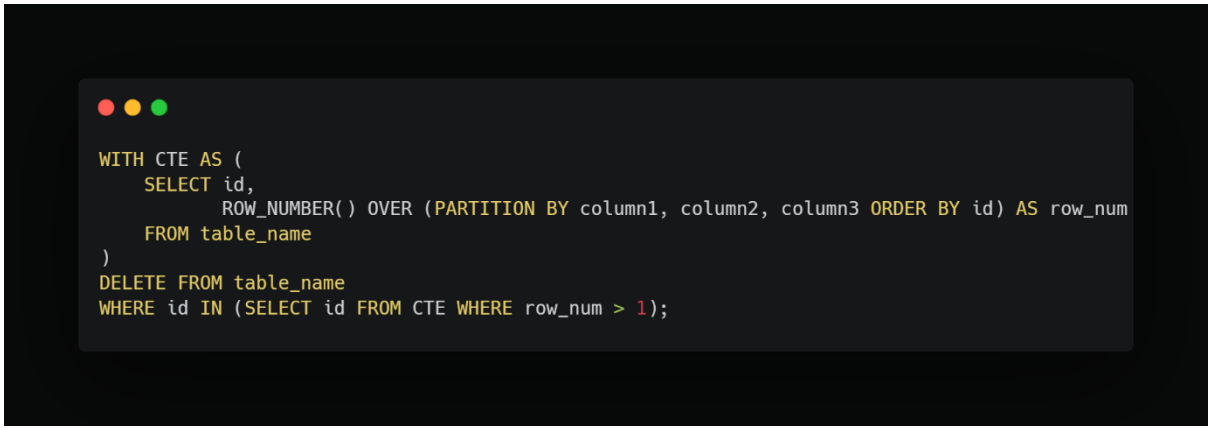
```sql
WITH CustomerPurchases AS (
    SELECT
        customer_id,
        purchase_date,
        LAG(purchase_date) OVER (PARTITION BY customer_id ORDER BY purchase_date) AS prev_purchase_date
    FROM Purchases
    WHERE purchase_date >= CURDATE() - INTERVAL 60 DAY  -- Consider only the last 60 days
)
SELECT customer_id
FROM CustomerPurchases
WHERE
    purchase_date >= CURDATE() - INTERVAL 30 DAY  -- Purchased in last 30 days
    AND (prev_purchase_date IS NULL OR prev_purchase_date < CURDATE() - INTERVAL 30 DAY); -- Did not
purchase in previous 30 days
```

```sql
CREATE INDEX idx_employee_salary ON Employee (ESaSELECT DISTINCT
FROM Purchases c
WHERE c.PurchaseDate BETWEEN CURDATE() - INTERVAL 30 DAY AND CURDATE()
AND NOT EXISTS (
    SELECT 1 FROM Purchases p
    WHERE p.CustomerID = c.CustomerID
    AND p.PurchaseDate BETWEEN CURDATE() - INTERVAL 60 DAY
  AND CURDATE() - INTERVAL 30 DAY
);
```

28. Given a table of transactions, find the top 3 most purchased products for each category.

```sql
SELECT CategoryID, ProductID, COUNT(*) AS TotalPurchases,
       DENSE_RANK() OVER (PARTITION BY CategoryID ORDER BY COUNT(*) DESC) AS rank
FROM Transactions
GROUP BY CategoryID, ProductID
HAVING rank <= 3;
```

29. How would you identify duplicate records in a large dataset, and how would you remove only the duplicates, retaining the first occurrence?



```sql
WITH CTE AS (
    SELECT id,
           ROW_NUMBER() OVER (PARTITION BY column1, column2, column3 ORDER BY id) AS row_num
    FROM table_name
)
DELETE FROM table_name
WHERE id IN (SELECT id FROM CTE WHERE row_num > 1);
```