General SQL Topics

1. INTRODUCTION TO SQL

- **SQL**: Structured Query Language, used to interact with relational databases.
- **Relational Database**: Stores data in tables (rows & columns).
- **Common RDBMS**: MySQL, PostgreSQL, SQLite, Oracle, SQL Server.

---

2. SQL DATA TYPES

- **INT**: Integer numbers
- **VARCHAR(n)**: Variable-length string with a maximum of n characters
- **CHAR(n)**: Fixed-length string
- **DATE / DATETIME**: Date or timestamp
- **DECIMAL(p, s)**: Exact numeric value with precision and scale
- **BOOLEAN**: True or False
- **TEXT**: Long-form string data

---

3. BASIC SQL COMMANDS

➤ Create Table

```
CREATE TABLE Employees (
  id INT PRIMARY KEY,
  name VARCHAR(100),
  salary INT,
  department VARCHAR(50)
);
```

➤ Insert

```
INSERT INTO Employees (id, name, salary, department)
VALUES (1, 'Alice', 60000, 'HR');
```

➤ Select

```
SELECT * FROM Employees;
SELECT name, salary FROM Employees;
SELECT * FROM Employees WHERE salary > 50000;
```

➤ Update

```
UPDATE Employees SET salary = 70000 WHERE id = 1;
```

➤ Delete

```
DELETE FROM Employees WHERE id = 1;
```

---

4. FILTERING DATA

➤ WHERE, AND, OR, NOT

```
SELECT * FROM Employees
WHERE department = 'HR' AND salary > 50000;
```

➤ IN, BETWEEN, LIKE

```
SELECT * FROM Employees WHERE department IN ('HR', 'Sales');
SELECT * FROM Employees WHERE salary BETWEEN 40000 AND 70000;
SELECT * FROM Employees WHERE name LIKE 'A%'; -- starts with A
```

---

5. SORTING & LIMITING

```
SELECT * FROM Employees ORDER BY salary DESC;
SELECT * FROM Employees LIMIT 5;
```

---

6. AGGREGATE FUNCTIONS

```
SELECT COUNT(*) FROM Employees;
SELECT AVG(salary) FROM Employees;
SELECT SUM(salary), MAX(salary), MIN(salary) FROM Employees;
```

---

7. GROUPING & FILTERING AGGREGATES

```
SELECT department, AVG(salary) AS avg_salary
FROM Employees
GROUP BY department
HAVING AVG(salary) > 50000;
```

---

8. JOINS

➤ Syntax:

```
SELECT e.name, d.name
FROM Employees e
```

```
JOIN Departments d ON e.department_id = d.id;
```

8.1. INNER JOIN

```
SELECT e.name, d.dept_name
FROM Employees e
INNER JOIN Departments d ON e.department_id = d.id;
```

- ✅ Only matching department IDs shown.

8.2. LEFT JOIN

```
SELECT e.name, d.dept_name
FROM Employees e
LEFT JOIN Departments d ON e.department_id = d.id;
```

- ✅ All employees shown, even if no department.

8.3. RIGHT JOIN

```
SELECT e.name, d.dept_name
FROM Employees e
RIGHT JOIN Departments d ON e.department_id = d.id;
```

- ✅ All departments shown, even if no employee.

8.4. FULL OUTER JOIN

```
SELECT e.name, d.dept_name
FROM Employees e
LEFT JOIN Departments d ON e.department_id = d.id
UNION
SELECT e.name, d.dept_name
FROM Employees e
RIGHT JOIN Departments d ON e.department_id = d.id;
```

---

9. SUBQUERIES

Used to embed one query inside another.

```
SELECT name FROM Employees
WHERE salary > (
  SELECT AVG(salary) FROM Employees
);
```

---

10. CONSTRAINTS

- **PRIMARY KEY**: Uniquely identifies each record
- **FOREIGN KEY**: Links to primary key of another table
- **UNIQUE**: Ensures all values in a column are different
- **NOT NULL**: Disallows NULL values
- **CHECK**: Ensures a condition is met

11. NORMALIZATION

- Process of organizing data to reduce redundancy and improve data integrity.
- **1NF**: Atomic values
- **2NF**: No partial dependencies
- **3NF**: No transitive dependencies
- Ensures data integrity and minimizes duplication.

🎯 Why Normalize?

- Eliminate duplicate data
- Ensure data consistency
- Improve data structure for querying and updates

📚 Normal Forms (NF)

✓ 1NF – First Normal Form

Rule: All values must be atomic (indivisible).
Fix: Remove repeating groups and store one value per cell.

Example (Bad):

ID	Name	Phones
1	Alice	1234, 5678

Fix (1NF):

ID	Name	Phone
1	Alice	1234
1	Alice	5678

✅ 2NF – Second Normal Form

Rule: Be in 1NF + No Partial Dependency on a composite key.
Fix: Move partially dependent data to a new table.

Example:
Composite key: (StudentID, CourseID)

StudentID	CourseID	StudentName

Fix (2NF):

- Table 1: StudentCourses(StudentID, CourseID)
- Table 2: Students(StudentID, StudentName)

✅ 3NF – Third Normal Form

Rule: Be in 2NF + No transitive dependency.
Fix: Remove data that's indirectly dependent on the primary key.

Example (Bad):

EmpID	Name	DeptID	DeptName

Fix (3NF):

- Table 1: Employees(EmpID, Name, DeptID)
- Table 2: Departments(DeptID, DeptName)

□ When Not to Normalize?

- For read-heavy applications (analytics, reporting)
- When query performance is more important than data integrity

---

✅ Real-Life Analogy

- **1NF:** Each house (record) has its own mailbox (cell), not a shared one
- **2NF:** One key opens only one mailbox (no shared access)

- **3NF:** The key opens the right mailbox and not someone else’s cabinet (no indirect dependencies)

---

12. INDEXING

Indexing is a powerful technique to speed up `SELECT` queries on large tables.

- Speeds up `WHERE`, `JOIN`, `GROUP BY`, and `ORDER BY` queries
- Boosts search performance on large datasets.
- Avoid on frequently updated or low-cardinality columns
- Use `EXPLAIN` to test effectiveness
- Avoid over-indexing (inserts & updates become slower).

✓ Types

- **Single-column Index**

  ```
  CREATE INDEX idx_salary ON Employees(salary);
  ```

- **Composite Index**

  ```
  CREATE INDEX idx_name_dept ON Employees(name, department);
  ```

---

13. VIEWS

- A **view** is a virtual table based on a query
- They do not store data. Use **Materialized Views** (if supported) for cached result.
- Simplify complex queries.
- Enhance security (restrict column access)
- Enable reusable logic

✓ Example

```
CREATE VIEW HR_Employees AS
SELECT name, salary
FROM Employees
WHERE department = 'HR';
```

---

14. TRANSACTIONS & ACID

A **Transaction** is a logical unit of work consisting of one or more SQL statements.

Use `ROLLBACK;` to undo changes before `COMMIT;`.

- **ACID**:
    - **Atomicity**: All or none
    - **Consistency**: Valid data state
    - **Isolation**: No interference
    - **Durability**: Permanent changes

✓ Example

```
BEGIN;
UPDATE Accounts SET balance = balance - 100 WHERE id = 1;
UPDATE Accounts SET balance = balance + 100 WHERE id = 2;
COMMIT;
```

---

15. WINDOW FUNCTIONS

Window functions perform calculations across rows related to the current row — without collapsing them like `GROUP BY`.

Used for row-level calculations without grouping.

Use cases : Leaderboards, Change Tracking, Tired analysis

RANK(), **DENSE_RANK()**, **ROW_NUMBER()**

✓ Example

```
SELECT name, salary,
  RANK() OVER (ORDER BY salary DESC) AS salary_rank
FROM Employees;
```

🔑 Popular Window Functions

- `ROW_NUMBER()`, `RANK()`, `DENSE_RANK()`
- `LAG()`, `LEAD()`
- `NTILE(n)` (for percentiles)

---

16. COMMON TABLE EXPRESSIONS (CTEs)

CTEs are temporary result sets used within queries — easier to read & maintain.

✓ Syntax

```
WITH HighEarners AS (
  SELECT * FROM Employees WHERE salary > 50000
)
SELECT name FROM HighEarners;
```

□ Benefits

- Break complex queries into readable blocks
- Use recursive CTEs for hierarchical data

---

17. UNION vs UNION ALL

```
SELECT name FROM Managers
UNION
SELECT name FROM Employees; -- Removes duplicates

SELECT name FROM Managers
UNION ALL
SELECT name FROM Employees; -- Keeps duplicates
```

---

18. SET OPERATIONS

```
SELECT name FROM A
INTERSECT
SELECT name FROM B;

SELECT name FROM A
EXCEPT
SELECT name FROM B;
```

- ⚠ Not supported in MySQL; supported in PostgreSQL/SQL Server.

---

19. NULL Handling

```
SELECT name, COALESCE(nickname, 'No Nick') FROM Users;
SELECT NULLIF(salary, 0);  -- Returns NULL if salary = 0
```

- Use **IS NULL / IS NOT NULL**
- **COALESCE()**, **IFNULL()**, **NULLIF()**

---

20. CASE Statements

```
SELECT name,
CASE
  WHEN salary > 70000 THEN 'High'
  WHEN salary > 40000 THEN 'Medium'
  ELSE 'Low'
END AS salary_level
FROM Employees;
```

---

21. Stored Procedures & Functions (Advanced)

Stored Procedures are saved SQL blocks that perform actions.
Functions return a single value. Use procedures for reusable logic. Use functions in `SELECT`, `WHERE`, or `JOIN` clauses

✅ Procedure Example

```
CREATE PROCEDURE GetHighEarners()
BEGIN
  SELECT * FROM Employees WHERE salary > 70000;
END;
```

✅ Function Example

```
CREATE FUNCTION TaxAmount(salary DECIMAL)
RETURNS DECIMAL
BEGIN
  RETURN salary * 0.10;
END;
```

---

22. Temporary Tables

```
CREATE TEMPORARY TABLE Temp_Employees AS
SELECT * FROM Employees WHERE department = 'HR';
```

---

23. Triggers (Advanced - Rare)

```
CREATE TRIGGER before_insert_trigger
BEFORE INSERT ON Employees
FOR EACH ROW
SET NEW.salary = IF(NEW.salary < 0, 0, NEW.salary);
```

- Used for validation, logging, auto-calculation.

---

24. Data Definition vs Data Manipulation vs Data Control

- **DDL (Definition)**: CREATE, ALTER, DROP
- **DML (Manipulation)**: SELECT, INSERT, UPDATE, DELETE
- **DCL (Control)**: GRANT, REVOKE
- **TCL (Transaction)**: COMMIT, ROLLBACK

INTERVIEW QUERY PATTERNS

➤ Nth Highest Salary

```
SELECT DISTINCT salary
FROM Employees
ORDER BY salary DESC
LIMIT 1 OFFSET 1; -- 2nd highest
```

➤ Duplicate Rows

```
SELECT name, COUNT(*)
FROM Employees
GROUP BY name
HAVING COUNT(*) > 1;
```

➤ Same Salary Employees

```
SELECT * FROM Employees
WHERE salary IN (
  SELECT salary FROM Employees
  GROUP BY salary
  HAVING COUNT(*) > 1
);
```

➤ Employees with Max Salary Per Department

```
SELECT name, department_id, salary
FROM (
  SELECT *,
         RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) AS rnk
  FROM Employees
) ranked
WHERE rnk = 1;
```

➤ Self Join Example

```
SELECT A.name AS Employee, B.name AS Manager
FROM Employees A
JOIN Employees B ON A.manager_id = B.id;
```

➤ EXISTS vs IN vs JOIN

```
-- EXISTS
SELECT name FROM Employees e
WHERE EXISTS (
  SELECT 1 FROM Departments d WHERE d.id = e.department_id
);
-- IN
SELECT name FROM Employees
WHERE department_id IN (SELECT id FROM Departments);
-- JOIN
SELECT e.name, d.name FROM Employees e
JOIN Departments d ON e.department_id = d.id;
```

Advanced SQL Topics

1. Recursive CTEs (Made Simple)

A **Recursive CTE** lets a query call itself, useful when working with **hierarchies** like employees & managers or categories & subcategories.

Easy Analogy:

Think of it like a loop in SQL. Start with one person (manager), then find their subordinates, then subordinates of subordinates, and so on.

Syntax:

```
WITH RECURSIVE cte_name AS (
  SELECT ...  -- Starting point (anchor)
  UNION ALL
  SELECT ... FROM cte_name ... -- Repeat until done
)
SELECT * FROM cte_name;
```

Example:

```
WITH RECURSIVE Subordinates AS (
  SELECT id, name, manager_id FROM Employees WHERE id = 1 -- Top manager
  UNION ALL
  SELECT e.id, e.name, e.manager_id
  FROM Employees e
  JOIN Subordinates s ON e.manager_id = s.id
)
SELECT * FROM Subordinates;
```

✓ This will return all employees working under manager ID 1, even indirectly.

---

2. Window Functions

These functions **look at other rows** in the result **without grouping** them. Useful when you want to compare each row with others.

Everyday Use Case:

"Show each employee's salary along with the previous and next employee's salary."

Key Functions:

- LAG(column) – Looks **before** current row
- LEAD(column) – Looks **after** current row
- NTILE(n) – Divides into n equal groups (for percentiles/quartiles)
- RANK(), DENSE_RANK(), ROW_NUMBER() – Ranking logic

Example:

```
SELECT name, salary,
    LAG(salary) OVER (ORDER BY salary) AS prev_salary,
    LEAD(salary) OVER (ORDER BY salary) AS next_salary
FROM Employees;
```

---

3. Working with JSON in SQL

Modern databases allow storing and querying **JSON data** inside SQL columns (especially useful in dynamic or semi-structured data).

PostgreSQL:

```
SELECT data->>'name' AS name FROM orders WHERE data->>'status' = 'shipped';
```

MySQL:

```
SELECT JSON_EXTRACT(data, '$.name') AS name FROM orders;
```

🔍 Here, data is a column containing JSON like: { "name": "Laptop", "status": "shipped" }

---

4. Performance Optimization Techniques

Optimizing your SQL queries can **make your app faster**, especially for large data.

a. Indexing

- Like a book index: helps locate rows faster
- Useful for WHERE, JOIN, and ORDER BY

```
CREATE INDEX idx_salary ON Employees(salary);
```

b. Query Refactoring

- Avoid SELECT * → only select needed columns
- Use EXISTS instead of IN when subquery is large
- Apply filters early (use WHERE before GROUP BY)

c. Execution Plan

Use EXPLAIN to understand how the database runs your query:

```
EXPLAIN SELECT * FROM Employees WHERE department_id = 3;
```

---

5. Stored Procedures & Functions (Simplified)

Stored Procedures = SQL scripts stored in the database that can be reused. Like a saved function.

Example with Loop:

```
DELIMITER //
CREATE PROCEDURE GiveBonus()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE emp_id INT;
  DECLARE emp_cursor CURSOR FOR SELECT id FROM Employees;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  OPEN emp_cursor;
  read_loop: LOOP
    FETCH emp_cursor INTO emp_id;
    IF done THEN
      LEAVE read_loop;
    END IF;
    UPDATE Employees SET salary = salary + 5000 WHERE id = emp_id;
  END LOOP;
  CLOSE emp_cursor;
END //
DELIMITER ;
```

✅ This procedure gives a ₹5000 bonus to all employees.

---

6. Importing/Exporting Data

Moving data between CSV files and SQL is common in real projects.

MySQL:

```
LOAD DATA INFILE '/path/file.csv'
INTO TABLE Employees
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

PostgreSQL:

```
COPY Employees(name, salary)
FROM '/path/file.csv'
DELIMITER ','
CSV HEADER;
```

---

7. Role-Based Access Control (Security Basics)

SQL lets you give different permissions to different users.

Example:

```
GRANT SELECT, INSERT ON Employees TO 'analyst';
REVOKE DELETE ON Employees FROM 'analyst';
```

✅ 'analyst' can view and add data, but can't delete anything.

---

8. Star vs Snowflake Schema

Used in **Data Warehouses** & **Reporting Tools**.

Star Schema:

- One central fact table (e.g., Sales)
- Linked to dimension tables (Customer, Product)
- Easy to query, faster for reporting

Snowflake Schema:

- Dimensions are normalized
- Less redundancy but more joins

---

9. OLTP vs OLAP (Database Types Simplified)

Feature	OLTP	OLAP
Purpose	Daily transactions	Data analysis
Design	Highly normalized tables	Denormalized schema
Examples	Banking, eCommerce	Reporting,Data Warehouse

🎯 INTERVIEW & PLACEMENT

◆ Basic Level (Entry / Fresher)

1. **What is SQL?**
   Structured Query Language used to interact with relational databases.
2. **What is the difference between WHERE and HAVING?**
   WHERE filters rows before aggregation, HAVING filters after aggregation.
3. **What is a Primary Key?**
   A unique identifier for each row in a table. Cannot contain NULL values.
4. **Difference between INNER JOIN and LEFT JOIN?**
   INNER JOIN: Returns matched rows only.
   LEFT JOIN: Returns all rows from the left table, even if no match.
5. **How do you handle NULLs in SQL?**
   Use IS NULL, IS NOT NULL, COALESCE(), IFNULL(), NULLIF().
6. **What is a Foreign Key?**
   It links two tables by referring to the primary key of another table.
7. **What is normalization?**
   Process of organizing data to reduce redundancy and improve integrity.
8. **What is denormalization?**
   Combining tables to improve read performance by reducing joins.
9. **What does the DISTINCT keyword do?**
   Removes duplicate rows from the result set.
10. **What is the use of LIMIT or TOP?**
    Used to limit the number of rows returned by a query.

◆ Intermediate Level

11. **What are aggregate functions?**
    Functions like SUM(), AVG(), COUNT(), MAX(), MIN().
12. **What is a subquery?**
    A query nested inside another query.
13. **What is the difference between UNION and UNION ALL?**
    UNION removes duplicates, UNION ALL includes all rows.
14. **Explain CASE statement.**
    Used to apply conditional logic within queries.
15. **What are indexes in SQL?**
    They improve the speed of data retrieval.
16. **What is a view?**
    A virtual table based on the result-set of a query.
17. **What are window functions?**
    Functions like RANK(), ROW_NUMBER() that work across rows.
18. **Difference between RANK() and DENSE_RANK()?**
    RANK() skips ranks on ties, DENSE_RANK() does not.
19. **What is a CTE (Common Table Expression)?**
    A temporary result set defined within the execution scope of a query.
20. **Explain ACID properties.**
    Atomicity, Consistency, Isolation, Durability - ensures reliable transactions.

21. **What is a composite key?**
    A primary key made of multiple columns.
22. **Difference between DELETE and TRUNCATE?**
    DELETE can be conditional and logs row-by-row deletion; TRUNCATE removes all rows faster without logging each deletion.
23. **What is a surrogate key?**
    A unique identifier for an entity that is not derived from application data.
24. **What is referential integrity?**
    Ensures foreign key values match primary key values in the referenced table.
25. **How does EXISTS differ from IN?**
    EXISTS stops on first match; IN evaluates all results.
26. **How do you use GROUP BY with multiple columns?**
    You can group by multiple columns by separating them with commas.
27. **Can we use ORDER BY with GROUP BY?**
    Yes. GROUP BY groups the data; ORDER BY sorts the grouped results.
28. **What is a scalar subquery?**
    A subquery that returns exactly one value.
29. **What are correlated subqueries?**
    Subqueries that refer to columns from the outer query.
30. **What is the use of ISNULL() or IFNULL()?**
    To replace NULL values with custom values.

◆ Advanced Level

31. **What are triggers in SQL?**
    Procedures that automatically execute on certain events.
32. **What are stored procedures?**
    Reusable blocks of SQL statements stored in the database.
33. **What is the use of EXPLAIN or EXPLAIN PLAN?**
    To understand how the database executes a query.
34. **What are transactions?**
    A unit of work that is performed against a database.
35. **How do you optimize a slow query?**
    Use indexing, limit joins, avoid SELECT *, and use EXPLAIN to analyze.
36. **Difference between clustered and non-clustered index?**
    Clustered index determines row order in the table; non-clustered does not.
37. **Difference between OLTP and OLAP systems?**
    OLTP: Online Transaction Processing (day-to-day operations).
    OLAP: Online Analytical Processing (data analysis and reporting).
38. **What are materialized views?**
    Stored query results that can be refreshed periodically.
39. **How do you implement pagination in SQL?**
    Using LIMIT and OFFSET or ROW_NUMBER() for custom logic.
40. **How do you handle duplicate rows?**
    Using ROW_NUMBER() or DISTINCT or CTEs with filtering.
41. **What are the different types of joins?**
    INNER, LEFT, RIGHT, FULL OUTER, CROSS JOIN, SELF JOIN.
42. **What is a CROSS JOIN?**
    Returns Cartesian product of two tables.

43. **Can you sort by an alias in SQL?**
    Yes, you can use the alias name in the ORDER BY clause.
44. **What is the difference between SQL and NoSQL?**
    SQL is relational and uses tables. NoSQL is non-relational and uses documents, key-value pairs, etc.
45. **What are the common data types in SQL?**
    INT, VARCHAR, DATE, BOOLEAN, DECIMAL, TEXT.
46. **Can a table have multiple foreign keys?**
    Yes, a table can reference multiple other tables using foreign keys.
47. **What happens if you violate a foreign key constraint?**
    The query fails with an integrity constraint violation.
48. **How can you change a column datatype in SQL?**
    Using ALTER TABLE table_name MODIFY column_name new_datatype;
49. **What are NULL-safe operators?**
    Operators like <=> in MySQL allow safe comparison with NULL.
50. **How would you detect and remove duplicate records?**
    Using CTE and ROW_NUMBER() to filter duplicates.

Real Company SQL Questions

Google

- Second Highest Salary

```
SELECT MAX(salary) AS SecondHighest
FROM Employees
WHERE salary < (SELECT MAX(salary) FROM Employees);
```

- Remove Duplicate Emails Using ROW_NUMBER()

```
WITH RankedEmails AS (
  SELECT *, ROW_NUMBER() OVER (PARTITION BY email ORDER BY id) AS rn
  FROM Users
)
DELETE FROM RankedEmails WHERE rn > 1;
```

- Employees Earning More Than Average Salary

```
SELECT name, salary
FROM Employees
WHERE salary > (SELECT AVG(salary) FROM Employees);
```

Amazon

- Customers Who Ordered in Every Month

```
SELECT customer_id
FROM Orders
GROUP BY customer_id
HAVING COUNT(DISTINCT MONTH(order_date)) = 12;
```

- Top 3 Earners in Each Department

```
SELECT *
FROM (
  SELECT *, DENSE_RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) AS rank
  FROM Employees
) ranked
WHERE rank <= 3;
```

- Find Returning Users Using LAG

```
SELECT user_id, order_date,
    LAG(order_date) OVER (PARTITION BY user_id ORDER BY
order_date) AS previous_order
FROM Orders;
```

TCS

- Employees Joined in Last 6 Months

```
SELECT * FROM Employees
WHERE join_date >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH);
```

- Manager-Reportee Mapping

```
SELECT e.name AS Employee, m.name AS Manager
FROM Employees e
JOIN Employees m ON e.manager_id = m.id;
```

- Department with Highest Average Salary

```
SELECT department_id
FROM Employees
GROUP BY department_id
ORDER BY AVG(salary) DESC
LIMIT 1;
```

Flipkart

- Top 5 Selling Products by Revenue

```
SELECT product_id, SUM(price * quantity) AS revenue
FROM Orders
GROUP BY product_id
ORDER BY revenue DESC
LIMIT 5;
```

- Users with More Than 3 Failed Transactions

```
SELECT user_id
FROM Transactions
WHERE status = 'FAILED'
```

```
GROUP BY user_id
HAVING COUNT(*) > 3;
```

- Orders Not Yet Shipped

```
SELECT *
FROM Orders
WHERE status = 'PLACED' AND shipped_date IS NULL;
```

SQL Problem-Solving Round

- Nth Highest Salary

```
SELECT DISTINCT salary
FROM Employees
ORDER BY salary DESC
LIMIT 1 OFFSET N-1;
```

- Duplicate Emails

```
SELECT email, COUNT(*)
FROM Users
GROUP BY email
HAVING COUNT(*) > 1;
```

- Top Earner per Department

```
SELECT name, department_id, salary
FROM (
  SELECT *, RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) AS rnk
  FROM Employees
) ranked
WHERE rnk = 1;
```

- Customers Who Never Ordered

```
SELECT c.customer_id, c.name
FROM Customers c
LEFT JOIN Orders o ON c.customer_id = o.customer_id
WHERE o.order_id IS NULL;
```

- Employees Hired in the Last Month

```
SELECT * FROM Employees
WHERE hire_date BETWEEN DATE_SUB(CURDATE(), INTERVAL 1
MONTH) AND CURDATE();
```

- User Order Summary

```
SELECT user_id, COUNT(*) AS order_count, SUM(total_amount) AS
total_spent
FROM Orders
GROUP BY user_id;
```

- 7-Day Rolling Login Count

```
SELECT user_id, login_date,
       COUNT(*) OVER (
         PARTITION BY user_id ORDER BY login_date
         ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
       ) AS seven_day_logins
FROM Logins;
```

- Funnel Drop-off Analysis

```
SELECT stage, COUNT(DISTINCT user_id) AS user_count
FROM Funnel
GROUP BY stage
ORDER BY stage;
```

- Average Time Between Orders

```
SELECT user_id, AVG(DATEDIFF(order_date, LAG(order_date) OVER
(PARTITION BY user_id ORDER BY order_date))) AS
avg_days_between_orders
FROM Orders;
```

- Self Join for Employee Manager Mapping

```
SELECT e.name AS Employee, m.name AS Manager
FROM Employees e
JOIN Employees m ON e.manager_id = m.id;
```