
⚡ Let's Begin: Part 1 – Core SQL and Database Concepts

---

1. What is a database index, and why would you use one? Can you give an example where an index might actually slow things down?

✅ **Answer:**
An **index** is a data structure (like B-trees or hash) used to **improve query performance** by allowing faster data retrieval. It's like an index in a book.

✅ **Why use one:**

Speeds up SELECT, WHERE, JOIN, and ORDER BY operations.

✅ **Can slow down:**

INSERT, **UPDATE**, or **DELETE** operations because the index must be updated.

Too many indexes = slower write performance and increased storage.

✅ **Example:**

```
CREATE INDEX idx_customer_name ON customers(name);
```

This improves:

```
SELECT * FROM customers WHERE name = 'Alice';
```

But slows down:

```
INSERT INTO customers VALUES (...);  -- index needs to update
```

---

2. Explain the difference between a primary key and a foreign key. Can a table have more than one foreign key?

✅ **Primary Key:**

Uniquely identifies a row.

Cannot contain NULLs.

Only **one per table**.

✅ **Foreign Key:**

Refers to a primary key in another table.

Enforces referential integrity.

A table **can have multiple foreign keys**.

✅ **Example:**

```
CREATE TABLE orders (
   order_id INT PRIMARY KEY,
   customer_id INT,
   FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

---

3. What is a database transaction? Why would you use one when updating multiple tables at the same time?

✅ A **transaction** is a sequence of operations executed as a single logical unit of work, with **ACID** properties.

✅ Useful to:

Ensure **data integrity**

Prevent partial updates

Support **rollback** on failure

✅ **Example:**

```
BEGIN;

UPDATE accounts SET balance = balance - 100 WHERE id = 1;
UPDATE accounts SET balance = balance + 100 WHERE id = 2;

COMMIT;
```

---

4. What are the different types of relationships between tables? Give an example of a many-to-many relationship.

✅ **Types:**

One-to-One: A person has one passport.

One-to-Many: A customer can place many orders.

Many-to-Many: Students can enroll in many courses and courses can have many students.

✅ **Many-to-Many Example:**

```
CREATE TABLE student_course (
```

```
    student_id INT,
    course_id INT,
    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES students(id),
    FOREIGN KEY (course_id) REFERENCES courses(id)
);
```

---

5. What is normalization? Why is it important, and why might you denormalize?

✅ **Normalization**: Organizing data to reduce redundancy and improve integrity.

1NF, 2NF, 3NF, BCNF...

✅ **Importance:**

Avoids anomalies (insert/update/delete)

Saves storage

Ensures consistency

✅ **Denormalization:**

Used for performance (fewer joins in OLAP/reporting)

Adds controlled redundancy

---

6. What is a view in a database? How is it different from a table? Can you update data through a view?

✅ **View**: A virtual table based on a SELECT query.

✅ **Difference:**

Table stores data, view does not.

Views present filtered or joined data.

✅ **Updatable Views**: Yes, **if** it references only one table and no aggregate/grouping.

✅ **Example:**

```
CREATE VIEW active_customers AS
SELECT id, name FROM customers WHERE status = 'active';
```

✅ **Update through view (if simple):**

```
UPDATE active_customers SET name = 'John' WHERE id = 101;
```

---

7. Common types of NoSQL databases & differences from relational:

✅ **Types:**

Document: MongoDB

Key-Value: Redis

Columnar: Cassandra

Graph: Neo4j

✅ **Differences:**

No fixed schema

Horizontal scaling

Optimized for specific use cases (e.g., large-scale real-time apps)

---

8. What is ACID in the context of databases?

✅ **ACID Properties:**

Atomicity: All or nothing

Consistency: Valid state maintained

Isolation: No interference between concurrent transactions

Durability: Once committed, it's permanent

✅ Ensures **data integrity** in multi-user environments.

---

9. What is the purpose of the UNION operator?

✅ Combines **results from two SELECT** queries.

Removes duplicates (use UNION ALL to keep duplicates)

```
SELECT city FROM customers
UNION
SELECT city FROM suppliers;
```

---

10. What is a subquery in SQL?

✅ A query nested inside another query.

✅ **Example:**

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

---

11. How do you delete data from a table?

```
DELETE FROM customers WHERE country = 'India';
```

---

12. What is the purpose of TRUNCATE TABLE?

✅ Deletes **all rows** from a table quickly.

- Cannot be rolled back (in some DBs)
- No WHERE clause

```
TRUNCATE TABLE customers;
```

---

13. What is a view in SQL?

✅ Already covered above in Q6.

---

14. What is the purpose of CREATE INDEX?

✅ Improves **query performance** on large tables.

```
CREATE INDEX idx_order_date ON orders(order_date);
```

---

15. What is a stored procedure in SQL?

✅ A precompiled set of SQL statements.

✅ Benefits:

- Reusability
- Better security and maintainability

```
CREATE PROCEDURE GetCustomerOrders @CustomerID INT
AS
BEGIN
   SELECT * FROM orders WHERE customer_id = @CustomerID;
END;
```

---

16. DELETE vs TRUNCATE vs DROP

Command	Deletes Rows	Removes Structure	Rollback Possible	WHERE Clause
DELETE	Yes	No	Yes	Yes
TRUNCATE	Yes (all)	No	Depends (No in MySQL)	No
DROP	Yes (all)	Yes	No	No

---

17. What is Data Warehousing? How is it different from a transactional DB?

✅ **Data Warehouse**:

Optimized for **analysis**, reporting, OLAP

Stores historical data

Denormalized schema

✅ **Transactional DB**:

For day-to-day operations

Highly normalized

Supports OLTP

---

18. Find customers who placed more than 5 orders:

```
SELECT customer_id, COUNT(*) AS order_count
FROM orders
GROUP BY customer_id
HAVING COUNT(*) > 5;
```

---

19. You accidentally deleted data – can you get it back?

✅ Only possible if:

Backups exist

Using **transactions** and haven't committed yet

```
ROLLBACK; -- if inside a transaction
```

---

20. Increase all product prices by 10%:

```
UPDATE products
SET price = price * 1.10;
```

✅ Part 2 – Joins, Locks, Constraints, and SQL Operations (Q21–Q45)

✅ SQL Interview Questions – Part 2 (Q21–Q45)

With **Questions + Detailed Answers + SQL Examples**

---

21. What are the different types of database locks, and how do they prevent concurrency issues?

✅ **Answer:**

Database locks ensure **data consistency and integrity** during concurrent access.

Shared Lock: Allows read access; multiple readers allowed.

Exclusive Lock: For writes/updates; only one writer allowed.

Row-level Lock: Locks a single row (fine-grained).

Table-level Lock: Locks the entire table (coarse-grained).

✅ **Purpose**:

Prevents **dirty reads**, **lost updates**, and **inconsistent reads** in multi-user environments.

---

22. Explain optimistic locking and pessimistic locking.

✅ **Answer:**

Optimistic Locking assumes conflicts are rare. Data is validated before commit using a version/timestamp.

Pessimistic Locking assumes conflicts are likely and locks the record early.

✅ Optimistic Locking Example:

```
UPDATE employees
SET salary = 60000, version = version + 1
WHERE id = 1 AND version = 2;
```

✅ Pessimistic Locking Example:

```
SELECT * FROM employees WHERE id = 1 FOR UPDATE;
```

---

23. Discuss the uses of inner join, left join, and full outer join. Provide real-world examples.

✅ Answer:

INNER JOIN: Returns records with matches in both tables.

LEFT JOIN: Returns all records from left table and matched records from right.

FULL OUTER JOIN: Returns all records from both sides, matched or not.

✅ Example:

```
-- INNER JOIN: Only customers with orders
SELECT c.name, o.order_id
FROM customers c
INNER JOIN orders o ON c.id = o.customer_id;

-- LEFT JOIN: All customers, even those with no orders
SELECT c.name, o.order_id
FROM customers c
LEFT JOIN orders o ON c.id = o.customer_id;

-- FULL OUTER JOIN: All customers and all orders (if supported)
SELECT c.name, o.order_id
FROM customers c
FULL OUTER JOIN orders o ON c.id = o.customer_id;
```

---

24. You have a column named 'customer_address'. How would you rename it to 'customer_location'?

✅ Answer:

```
ALTER TABLE customers
RENAME COLUMN customer_address TO customer_location;
```

MySQL < 8.0:

```
ALTER TABLE customers CHANGE customer_address customer_location VARCHAR(255);
```

---

25. You want to ensure that every customer in your table has a unique customer ID. How would you enforce this in SQL?

✅ **Answer:**

```
ALTER TABLE customers
ADD CONSTRAINT unique_customer_id UNIQUE (customer_id);
```

If not already the **PRIMARY KEY**, you can make it one:

```
ALTER TABLE customers
ADD PRIMARY KEY (customer_id);
```

---

26. What is a deadlock in a database? How do deadlocks occur, and what are strategies to handle them?

✅ **Answer:**

A **deadlock** occurs when two or more transactions are **waiting for each other** to release locks.

✅ **Example Scenario**:

T1 locks row A and waits for row B.

T2 locks row B and waits for row A.

✅ **Prevention**:

Lock resources in consistent order.

Use **timeouts** or **deadlock detection** to roll back one of the transactions.

---

27. What are the advantages and disadvantages of using stored procedures in SQL?

✅ **Answer:**

Advantages:

Better performance (precompiled)

Reusability

Encapsulation

Security (control access)

Disadvantages:

Harder to debug/version

Vendor-dependent syntax

✅ **Use Case**:

Complex business logic reused often.

---

28. You have a table of orders and a table of customers. How would you ensure every order is associated with a valid customer?

✅ **Answer:**

Use a **foreign key constraint**:

```
ALTER TABLE orders
ADD CONSTRAINT fk_customer
FOREIGN KEY (customer_id)
REFERENCES customers(customer_id);
```

---

29. You have two tables: customers and orders. How do you combine them to show customer names for each order?

✅ **Answer:**

```
SELECT o.order_id, c.name
FROM orders o
JOIN customers c ON o.customer_id = c.id;
```

---

30. How do you find the highest salary in your employee table?

✅ **Answer:**

```
SELECT MAX(salary) AS highest_salary
FROM employees;
```

---

31. You have a table with customer reviews. How do you find the average rating for each product?

✅ **Answer:**

```
SELECT product_id, AVG(rating) AS average_rating
FROM reviews
GROUP BY product_id;
```

---

32. What is a database view, and how is it different from a table? What are the pros and cons of using views?

✅ **Answer:**

A **view** is a **virtual table** created from a query.

Differences:

Feature	Table	View
Storage	Stores data	Doesn't store data
Speed	Fast	Depends on underlying query
Updatable	Yes	Sometimes

Advantages:

Simplifies complex queries

Increases security by restricting data access

Disadvantages:

May slow performance

Limited update capabilities

---

33. You have a table with order dates. How do you extract the month from the date?

✅ **Answer:**

```
SELECT order_id, MONTH(order_date) AS order_month
FROM orders;
```

PostgreSQL: EXTRACT(MONTH FROM order_date)

---

34. You have a table with duplicate customer records. How would you remove duplicates?

✅ **Answer (using CTE):**

```
WITH cte AS (
  SELECT *, ROW_NUMBER() OVER (PARTITION BY name, email ORDER BY id) AS rn
  FROM customers
)
DELETE FROM customers
```

```
WHERE id IN (SELECT id FROM cte WHERE rn > 1);
```

---

35. You want to find all customers who live in a specific city. How would you do this?

✅ Answer:

```
SELECT * FROM customers
WHERE city = 'Delhi';
```

---

36. You want to sort a customer table by their last name. How would you do this?

✅ Answer:

```
SELECT * FROM customers
ORDER BY last_name ASC;
```

---

37. Write a SQL query to find the difference between the highest and lowest salaries in each department.

✅ Answer:

```
SELECT department_id,
     MAX(salary) - MIN(salary) AS salary_diff
FROM employees
GROUP BY department_id;
```

---

38. Write a SQL query to find the names of employees who are also customers.

✅ Answer (if name is the common field):

```
SELECT e.name
FROM employees e
JOIN customers c ON e.name = c.name;
```

Or using EXISTS:

```
SELECT name FROM employees e
WHERE EXISTS (
  SELECT 1 FROM customers c WHERE c.name = e.name
);
```

---

39. Write a SQL query to calculate the total sales for each product, including only sales made during weekdays.

✅ Answer:

```
SELECT product_id, SUM(amount) AS total_sales
FROM orders
```

```
WHERE DAYOFWEEK(order_date) BETWEEN 2 AND 6
GROUP BY product_id;
```

PostgreSQL: EXTRACT(DOW FROM order_date) (1–5 = Mon–Fri)

---

40. Write a SQL query to identify the customers who have placed the most orders.

✅ **Answer:**

```
SELECT customer_id, COUNT(*) AS order_count
FROM orders
GROUP BY customer_id
ORDER BY order_count DESC
LIMIT 1;
```

---

41. Write a SQL query to update the shipping address for a customer's most recent order.

✅ **Answer:**

```
UPDATE orders
SET shipping_address = 'New Address'
WHERE customer_id = 101
  AND order_date = (
    SELECT MAX(order_date)
    FROM orders
    WHERE customer_id = 101
  );
```

---

42. Write a SQL query to calculate the average rating for each product, considering only ratings submitted in the last year.

✅ **Answer:**

```
SELECT product_id, AVG(rating) AS avg_rating
FROM reviews
WHERE review_date >= CURRENT_DATE - INTERVAL 1 YEAR
GROUP BY product_id;
```

---

43. Write a SQL query to find the products that have been ordered by all customers.

✅ **Answer:**

```
SELECT product_id
FROM orders
GROUP BY product_id
HAVING COUNT(DISTINCT customer_id) = (SELECT COUNT(*) FROM customers);
```

---

44. Write a SQL query to generate a report that shows the monthly sales trend for each product category.

✅ **Answer:**

```
SELECT p.category, DATE_FORMAT(o.order_date, '%Y-%m') AS month,
    SUM(o.amount) AS monthly_sales
FROM orders o
JOIN products p ON o.product_id = p.id
GROUP BY p.category, month
ORDER BY month, p.category;
```

PostgreSQL: Use TO_CHAR(order_date, 'YYYY-MM')

---

45. You want to make sure that a customer's age is always greater than 18 when entered into the database. How would you enforce this rule?

✅ **Answer (with CHECK constraint):**

```
ALTER TABLE customers
ADD CONSTRAINT chk_age CHECK (age > 18);
```

✅ SQL Interview Questions – Part 3 (Q46–Q70)

Focus: **Customer behavior, business logic, aggregation, filtering, updates, and analysis**

---

46. You have a table with sales data, and you want to see the total sales for each month. How would you do this?

✅ **Answer:**

```
SELECT DATE_FORMAT(order_date, '%Y-%m') AS month,
    SUM(amount) AS total_sales
FROM sales
GROUP BY month
ORDER BY month;
```

PostgreSQL: TO_CHAR(order_date, 'YYYY-MM')

---

47. You have a table with customer names, but some names are in uppercase, lowercase, and mixed. How would you make them all consistently uppercase?

✅ **Answer:**

```
SELECT UPPER(name) AS normalized_name
FROM customers;
```

To update permanently:

```
UPDATE customers
SET name = UPPER(name);
```

---

48. You need to find the customers who haven't placed an order in the last year. How would you do this?

✅ Answer:

```
SELECT * FROM customers
WHERE id NOT IN (
  SELECT DISTINCT customer_id
  FROM orders
  WHERE order_date >= CURRENT_DATE - INTERVAL 1 YEAR
);
```

---

49. Write a SQL query to identify the customers who have placed orders in all our store locations.

✅ Answer:

```
SELECT customer_id
FROM orders
GROUP BY customer_id
HAVING COUNT(DISTINCT store_location_id) = (SELECT COUNT(DISTINCT store_location_id)
FROM stores);
```

---

50. Write a SQL query to calculate the total discount amount given to each customer.

✅ Answer:

```
SELECT customer_id, SUM(discount_amount) AS total_discount
FROM orders
GROUP BY customer_id;
```

---

51. Write a SQL query to find the employees who were hired in the same year.

✅ Answer:

```
SELECT EXTRACT(YEAR FROM hire_date) AS hire_year, COUNT(*) AS employees_count
FROM employees
GROUP BY hire_year
HAVING COUNT(*) > 1;
```

---

52. Write a SQL query to find the products with the highest and lowest prices within each category.

✅ Answer (using CTE and window functions):

```
WITH ranked AS (
  SELECT *,
       RANK() OVER (PARTITION BY category ORDER BY price ASC) AS min_rank,
       RANK() OVER (PARTITION BY category ORDER BY price DESC) AS max_rank
  FROM products
)
SELECT * FROM ranked
WHERE min_rank = 1 OR max_rank = 1;
```

---

53. Write a SQL query to calculate the average salary for each job title, excluding the top 10% of salaries.

✅ Answer (using PERCENT_RANK):

```
WITH ranked AS (
  SELECT *, PERCENT_RANK() OVER (PARTITION BY job_title ORDER BY salary DESC) AS
rank_pct
  FROM employees
)
SELECT job_title, AVG(salary) AS avg_salary
FROM ranked
WHERE rank_pct > 0.1
GROUP BY job_title;
```

---

54. Write a SQL query to find the customers who have placed orders on their birthdays.

✅ Answer:

```
SELECT c.id, c.name
FROM customers c
JOIN orders o ON c.id = o.customer_id
WHERE DAY(c.birth_date) = DAY(o.order_date)
  AND MONTH(c.birth_date) = MONTH(o.order_date);
```

---

55. Write a SQL query to generate a report that shows the daily sales for each product, including only days where sales exceeded a certain threshold.

✅ Answer:

```
SELECT product_id, order_date, SUM(amount) AS daily_sales
FROM orders
GROUP BY product_id, order_date
HAVING SUM(amount) > 1000; -- threshold
```

---

56. Write a SQL query to find the departments with the highest average employee salary.

✅ **Answer:**

```
SELECT department_id, AVG(salary) AS avg_salary
FROM employees
GROUP BY department_id
ORDER BY avg_salary DESC
LIMIT 1;
```

---

57. We need to find the products that have received the most positive reviews. How would you write a query to do this?

✅ **Answer:**

```
SELECT product_id, COUNT(*) AS positive_reviews
FROM reviews
WHERE rating >= 4
GROUP BY product_id
ORDER BY positive_reviews DESC
LIMIT 5;
```

---

58. We need to calculate the total number of working days between two dates, excluding weekends and holidays. How would you write a query to do this?

✅ **Answer (assuming a holidays table):**

```
SELECT COUNT(*) AS working_days
FROM calendar
WHERE date BETWEEN '2024-01-01' AND '2024-12-31'
  AND WEEKDAY(date) < 5
  AND date NOT IN (SELECT holiday_date FROM holidays);
```

---

59. Write a SQL query to find customers who haven't placed an order in the last six months.

✅ **Answer:**

```
SELECT * FROM customers
WHERE id NOT IN (
  SELECT DISTINCT customer_id
  FROM orders
  WHERE order_date >= CURRENT_DATE - INTERVAL 6 MONTH
);
```

---

60. Write a SQL query to update the email addresses for customers who live in a specific city.

✅ **Answer:**

```
UPDATE customers
SET email = CONCAT('updated_', email)
WHERE city = 'Mumbai';
```

---

61. Write a SQL query that calculates the profit margin for each product and displays them in descending order to find out which products have the highest profit margins.

✅ Answer:

```
SELECT id, name,
    ((price - cost_price) / price) * 100 AS profit_margin
FROM products
ORDER BY profit_margin DESC;
```

---

62. Write a SQL query to find customers who have placed orders in both the 'Electronics' and 'Clothing' categories.

✅ Answer:

```
SELECT customer_id
FROM orders o
JOIN products p ON o.product_id = p.id
WHERE p.category IN ('Electronics', 'Clothing')
GROUP BY customer_id
HAVING COUNT(DISTINCT p.category) = 2;
```

---

63. Write a SQL query that categorizes customers as 'High Value', 'Medium Value', or 'Low Value' based on their total purchase value.

✅ Answer:

```
SELECT customer_id,
    SUM(amount) AS total_spent,
    CASE
      WHEN SUM(amount) >= 10000 THEN 'High Value'
      WHEN SUM(amount) >= 5000 THEN 'Medium Value'
      ELSE 'Low Value'
    END AS customer_category
FROM orders
GROUP BY customer_id;
```

---

64. Write a SQL query to calculate the average order value for each customer.

✅ Answer:

```
SELECT customer_id,
    AVG(amount) AS avg_order_value
FROM orders
GROUP BY customer_id;
```

---

65. Write a SQL query that shows the total number of orders placed on each day of the week to find out the busiest day of the week for an online store.

✅ **Answer:**

```
SELECT DAYNAME(order_date) AS day_name,
      COUNT(*) AS total_orders
FROM orders
GROUP BY day_name
ORDER BY total_orders DESC;
```

---

66. Write a SQL query that updates the 'discount' field for customers who have placed more than 5 orders, giving them a 10% discount.

✅ **Answer:**

```
UPDATE customers
SET discount = 0.10
WHERE id IN (
  SELECT customer_id
  FROM orders
  GROUP BY customer_id
  HAVING COUNT(*) > 5
);
```

---

67. Write a SQL query to find the second highest salary among employees without using a subquery.

✅ **Answer (using** DENSE_RANK()**):**

```
SELECT salary
FROM (
  SELECT salary, DENSE_RANK() OVER (ORDER BY salary DESC) AS rnk
  FROM employees
) AS ranked
WHERE rnk = 2;
```

---

68. Write a SQL query to delete orders that were placed more than two years ago and have a status of 'Cancelled'.

✅ **Answer:**

```
DELETE FROM orders
WHERE status = 'Cancelled'
  AND order_date < CURRENT_DATE - INTERVAL 2 YEAR;
```

---

69. Write a SQL query to identify products that have never been ordered.

✅ **Answer:**

```
SELECT * FROM products
WHERE id NOT IN (
  SELECT DISTINCT product_id FROM orders
);
```

---

70. Write a SQL query to update the product prices by 5% for products in a specific category.

✅ Answer:

```
UPDATE products
SET price = price * 1.05
WHERE category = 'Electronics';
```

✅ SQL Interview Questions – Part 4 (Q71–Q85)

Focus: **Time series, login activity, revenue, retention, segmentation, rolling metrics**

---

71. Write a SQL query to calculate the total revenue generated each quarter.

✅ Answer:

```
SELECT CONCAT(YEAR(order_date), '-Q', QUARTER(order_date)) AS quarter,
      SUM(amount) AS total_revenue
FROM orders
GROUP BY YEAR(order_date), QUARTER(order_date)
ORDER BY quarter;
```

---

72. Write a SQL query to find customers who made purchases in at least three consecutive months.

✅ Answer (using window functions):

```
WITH customer_months AS (
  SELECT customer_id,
       DATE_FORMAT(order_date, '%Y-%m') AS order_month
  FROM orders
  GROUP BY customer_id, order_month
),
numbered AS (
  SELECT customer_id, order_month,
       ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY order_month) AS rn
  FROM customer_months
),
grouped AS (
  SELECT customer_id,
       DATE_ADD(order_month, INTERVAL -rn MONTH) AS grp
  FROM numbered
```

```
)
SELECT customer_id
FROM grouped
GROUP BY customer_id, grp
HAVING COUNT(*) >= 3;
```

---

73. Write a SQL query to calculate the average revenue per active user (ARPU) for each month.

✅ Answer:

```
SELECT DATE_FORMAT(order_date, '%Y-%m') AS month,
      SUM(amount) / COUNT(DISTINCT customer_id) AS ARPU
FROM orders
GROUP BY month
ORDER BY month;
```

---

74. Write a SQL query to find the first purchase date for each customer.

✅ Answer:

```
SELECT customer_id, MIN(order_date) AS first_purchase
FROM orders
GROUP BY customer_id;
```

---

75. Write a SQL query to calculate the number of returning customers each month (those who have made at least one purchase before the current month).

✅ Answer:

```
WITH purchases AS (
  SELECT customer_id, order_date,
        DATE_FORMAT(order_date, '%Y-%m') AS month
  FROM orders
),
firsts AS (
  SELECT customer_id, MIN(order_date) AS first_order
  FROM purchases
  GROUP BY customer_id
)
SELECT p.month, COUNT(DISTINCT p.customer_id) AS returning_customers
FROM purchases p
JOIN firsts f ON p.customer_id = f.customer_id
WHERE p.order_date > f.first_order
GROUP BY p.month
ORDER BY p.month;
```

---

76. Write a SQL query to find the revenue contribution of the top 5% of customers by total spend.

✅ Answer:

```
WITH customer_total AS (
  SELECT customer_id, SUM(amount) AS total_spent
  FROM orders
  GROUP BY customer_id
),
ranked AS (
  SELECT *, NTILE(20) OVER (ORDER BY total_spent DESC) AS percentile
  FROM customer_total
)
SELECT SUM(total_spent) AS top_5_revenue
FROM ranked
WHERE percentile = 1;
```

---

77. Write a SQL query to calculate the churn rate (customers who didn't return after their first month).

✅ Answer:

```
WITH first_last AS (
  SELECT customer_id,
       MIN(DATE_FORMAT(order_date, '%Y-%m')) AS first_month,
       MAX(DATE_FORMAT(order_date, '%Y-%m')) AS last_month
  FROM orders
  GROUP BY customer_id
)
SELECT COUNT(*) AS churned_customers
FROM first_last
WHERE first_month = last_month;
```

---

78. Write a SQL query to find the monthly active users (MAU) and daily active users (DAU).

✅ Answer:

```
-- Daily Active Users
SELECT order_date, COUNT(DISTINCT customer_id) AS DAU
FROM orders
GROUP BY order_date;

-- Monthly Active Users
SELECT DATE_FORMAT(order_date, '%Y-%m') AS month,
     COUNT(DISTINCT customer_id) AS MAU
FROM orders
GROUP BY month;
```

---

79. Write a SQL query to calculate the lifetime value (LTV) of each customer.

✅ Answer:

```
SELECT customer_id,
     SUM(amount) AS lifetime_value
FROM orders
GROUP BY customer_id
ORDER BY lifetime_value DESC;
```

---

80. Write a SQL query to calculate the average order frequency for each customer.

✅ Answer:

```
WITH order_dates AS (
  SELECT customer_id,
        MIN(order_date) AS first_order,
        MAX(order_date) AS last_order,
        COUNT(*) AS total_orders
  FROM orders
  GROUP BY customer_id
)
SELECT customer_id,
      total_orders,
      DATEDIFF(last_order, first_order) / NULLIF(total_orders - 1, 0) AS avg_order_frequency_days
FROM order_dates;
```

---

81. Write a SQL query to find the top 3 product categories by revenue in each quarter.

✅ Answer:

```
WITH revenue_data AS (
  SELECT category,
        QUARTER(order_date) AS qtr,
        YEAR(order_date) AS yr,
        SUM(amount) AS revenue
  FROM orders o
  JOIN products p ON o.product_id = p.id
  GROUP BY category, qtr, yr
),
ranked AS (
  SELECT *, RANK() OVER (PARTITION BY yr, qtr ORDER BY revenue DESC) AS rnk
  FROM revenue_data
)
SELECT * FROM ranked
WHERE rnk <= 3;
```

---

82. Write a SQL query to count the number of customers who made their first purchase in each month.

✅ Answer:

```
WITH first_purchase AS (
  SELECT customer_id, MIN(order_date) AS first_date
  FROM orders
  GROUP BY customer_id
)
SELECT DATE_FORMAT(first_date, '%Y-%m') AS month,
      COUNT(*) AS new_customers
FROM first_purchase
```

```
GROUP BY month;
```

---

83. Write a SQL query to get the customer retention rate for the second month after signup.

✅ Answer:

```
WITH signup_months AS (
  SELECT customer_id, MIN(DATE_FORMAT(order_date, '%Y-%m')) AS signup_month
  FROM orders
  GROUP BY customer_id
),
retention AS (
  SELECT o.customer_id,
        s.signup_month,
        DATE_FORMAT(o.order_date, '%Y-%m') AS active_month
  FROM orders o
  JOIN signup_months s ON o.customer_id = s.customer_id
)
SELECT signup_month,
      COUNT(DISTINCT customer_id) AS total_signups,
      SUM(CASE WHEN PERIOD_DIFF(EXTRACT(YEAR_MONTH FROM active_month),
                        EXTRACT(YEAR_MONTH FROM signup_month)) = 1 THEN 1 ELSE 0 END)
AS retained_next_month,
      ROUND(100.0 * SUM(CASE WHEN PERIOD_DIFF(EXTRACT(YEAR_MONTH FROM
active_month),
                        EXTRACT(YEAR_MONTH FROM signup_month)) = 1 THEN 1 ELSE 0 END)
/ COUNT(*), 2) AS retention_rate
FROM retention
GROUP BY signup_month;
```

---

84. Write a SQL query to identify customers who always purchase the same product.

✅ Answer:

```
SELECT customer_id
FROM orders
GROUP BY customer_id
HAVING COUNT(DISTINCT product_id) = 1;
```

---

85. Write a SQL query to calculate the average time between purchases for each customer.

✅ Answer:

```
WITH ordered_dates AS (
  SELECT customer_id, order_date,
        LAG(order_date) OVER (PARTITION BY customer_id ORDER BY order_date) AS prev_order
  FROM orders
),
diffs AS (
  SELECT customer_id,
```

```
       DATEDIFF(order_date, prev_order) AS days_between
  FROM ordered_dates
  WHERE prev_order IS NOT NULL
)
SELECT customer_id, AVG(days_between) AS avg_days_between_orders
FROM diffs
GROUP BY customer_id;
```

✅ SQL Interview Questions – Part 5 (Q86–Q100)

Focus: **Growth metrics, user journey, funnel analysis, revenue trends, complex aggregations**

---

86. Write a SQL query to calculate quarterly revenue growth.

✅ **Answer:**

```
WITH quarterly_revenue AS (
  SELECT CONCAT(YEAR(order_date), '-Q', QUARTER(order_date)) AS quarter,
       SUM(amount) AS revenue
  FROM orders
  GROUP BY YEAR(order_date), QUARTER(order_date)
),
growth_calc AS (
  SELECT quarter, revenue,
       LAG(revenue) OVER (ORDER BY quarter) AS prev_revenue
  FROM quarterly_revenue
)
SELECT quarter, revenue,
      ROUND(100.0 * (revenue - prev_revenue) / prev_revenue, 2) AS growth_percent
FROM growth_calc;
```

---

87. Write a SQL query to identify users who viewed a product but didn't purchase it.

✅ **Assumption**: You have a user_events table with event_type = 'view' and 'purchase'.

```
SELECT DISTINCT user_id
FROM user_events
WHERE event_type = 'view'
  AND user_id NOT IN (
    SELECT DISTINCT user_id
    FROM user_events
    WHERE event_type = 'purchase'
  );
```

---

88. Write a SQL query to calculate the conversion rate from signup to first purchase.

✅ Answer:

```
WITH signup_users AS (
  SELECT DISTINCT user_id
  FROM user_events
  WHERE event_type = 'signup'
),
first_purchase AS (
  SELECT DISTINCT user_id
  FROM user_events
  WHERE event_type = 'purchase'
)
SELECT
  COUNT(DISTINCT f.user_id) AS converted_users,
  COUNT(DISTINCT s.user_id) AS total_signups,
  ROUND(100.0 * COUNT(DISTINCT f.user_id) / COUNT(DISTINCT s.user_id), 2) AS
conversion_rate
FROM signup_users s
LEFT JOIN first_purchase f ON s.user_id = f.user_id;
```

---

89. Write a SQL query to find the most common funnel path (e.g., view → cart → purchase).

✅ Answer:

```
SELECT user_id,
      GROUP_CONCAT(event_type ORDER BY event_time SEPARATOR ' → ') AS funnel_path
FROM user_events
GROUP BY user_id;
```

---

90. Write a SQL query to calculate the dropout rate at each funnel stage.

✅ Answer:

```
SELECT
  'View' AS stage, COUNT(DISTINCT user_id) AS users
FROM user_events
WHERE event_type = 'view'

UNION ALL

SELECT
  'Add to Cart' AS stage, COUNT(DISTINCT user_id)
FROM user_events
WHERE event_type = 'add_to_cart'

UNION ALL

SELECT
  'Purchase' AS stage, COUNT(DISTINCT user_id)
FROM user_events
WHERE event_type = 'purchase';
```

You can calculate dropout manually from this result.

---

91. Write a SQL query to identify customers who upgraded their subscription plan.

✅ Answer (assuming you have subscription_changes table):

```
SELECT user_id, MIN(change_date) AS upgrade_date
FROM subscription_changes
WHERE old_plan < new_plan
GROUP BY user_id;
```

---

92. Write a SQL query to find products with declining sales over the last 3 months.

✅ Answer:

```
WITH monthly_sales AS (
  SELECT product_id,
       DATE_FORMAT(order_date, '%Y-%m') AS month,
       SUM(quantity) AS total_qty
  FROM orders
  GROUP BY product_id, month
),
ranked AS (
  SELECT product_id, month, total_qty,
       ROW_NUMBER() OVER (PARTITION BY product_id ORDER BY month DESC) AS rn
  FROM monthly_sales
)
SELECT s1.product_id
FROM ranked s1
JOIN ranked s2 ON s1.product_id = s2.product_id AND s1.rn = 1 AND s2.rn = 2
JOIN ranked s3 ON s1.product_id = s3.product_id AND s3.rn = 3
WHERE s1.total_qty < s2.total_qty AND s2.total_qty < s3.total_qty;
```

---

93. Write a SQL query to calculate the average number of products per purchase.

✅ Answer:

```
SELECT AVG(product_count) AS avg_products_per_order
FROM (
  SELECT order_id, COUNT(product_id) AS product_count
  FROM order_items
  GROUP BY order_id
) AS sub;
```

---

94. Write a SQL query to get the top 3 revenue-generating cities per month.

✅ Answer:

```
WITH city_month_revenue AS (
  SELECT city,
       DATE_FORMAT(order_date, '%Y-%m') AS month,
```

```
        SUM(amount) AS revenue
  FROM orders
  GROUP BY city, month
),
ranked AS (
  SELECT *, RANK() OVER (PARTITION BY month ORDER BY revenue DESC) AS rnk
  FROM city_month_revenue
)
SELECT *
FROM ranked
WHERE rnk <= 3;
```

---

95. Write a SQL query to identify the first touchpoint that led to a conversion.

✅ **Assume table** user_events(event_type, event_time, user_id)

```
WITH first_touch AS (
  SELECT user_id, MIN(event_time) AS first_event_time
  FROM user_events
  WHERE event_type IN ('ad_click', 'organic', 'referral')
  GROUP BY user_id
),
joined AS (
  SELECT e.user_id, e.event_type
  FROM user_events e
  JOIN first_touch f ON e.user_id = f.user_id AND e.event_time = f.first_event_time
)
SELECT user_id, event_type AS first_touchpoint
FROM joined;
```

---

96. Write a SQL query to calculate the bonus payout if each sale above ₹10,000 gives ₹500 bonus.

✅ **Answer:**

```
SELECT COUNT(*) * 500 AS total_bonus
FROM orders
WHERE amount > 10000;
```

---

97. Write a SQL query to identify top 3 products with the highest increase in monthly revenue.

✅ **Answer:**

```
WITH monthly_revenue AS (
  SELECT product_id,
        DATE_FORMAT(order_date, '%Y-%m') AS month,
        SUM(amount) AS revenue
  FROM orders
  GROUP BY product_id, month
),
growth_calc AS (
  SELECT product_id, month, revenue,
```

```
        revenue - LAG(revenue) OVER (PARTITION BY product_id ORDER BY month) AS
revenue_growth
  FROM monthly_revenue
)
SELECT product_id, month, revenue_growth
FROM growth_calc
ORDER BY revenue_growth DESC
LIMIT 3;
```

---

98. Write a SQL query to detect duplicate user registrations by email.

✅ Answer:

```
SELECT email, COUNT(*) AS registrations
FROM users
GROUP BY email
HAVING COUNT(*) > 1;
```

---

99. Write a SQL query to calculate the revenue contribution of new vs. returning customers.

✅ Answer:

```
WITH first_orders AS (
  SELECT customer_id, MIN(order_date) AS first_order
  FROM orders
  GROUP BY customer_id
),
labeled AS (
  SELECT o.*,
        CASE
          WHEN o.order_date = f.first_order THEN 'New'
          ELSE 'Returning'
        END AS customer_type
  FROM orders o
  JOIN first_orders f ON o.customer_id = f.customer_id
)
SELECT customer_type, SUM(amount) AS revenue
FROM labeled
GROUP BY customer_type;
```

---

100. Write a SQL query to calculate rolling 7-day average revenue.

✅ Answer:

```
SELECT order_date,
      ROUND(AVG(daily_revenue) OVER (ORDER BY order_date ROWS BETWEEN 6
PRECEDING AND CURRENT ROW), 2) AS rolling_7_day_avg
FROM (
  SELECT order_date, SUM(amount) AS daily_revenue
  FROM orders
  GROUP BY order_date
) AS sub;
```

✅ SQL Interview Questions – Part 6 (Q101–Q120)

Focus: **Database Design, Normalization, Stored Procedures, Triggers, Indexing, Partitioning, Optimization, Analytics**

---

101. What is normalization in SQL? Explain 1NF, 2NF, and 3NF.

✅ **Answer:**

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity.

1NF (First Normal Form):
Each column should contain atomic values, and each row should be unique.
Example: No repeating groups or arrays.

2NF (Second Normal Form):
It must be in 1NF + no partial dependency (non-key attribute depends on full primary key).
Fix by breaking into separate tables.

3NF (Third Normal Form):
It must be in 2NF + no transitive dependency (non-key attribute depends on another non-key).
Move dependent columns to another table.

---

102. Write a SQL query to check which table violates 1NF.

✅ **Example** (detect columns with comma-separated values):

```
SELECT *
FROM students
WHERE FIND_IN_SET(',', hobbies) > 0;
```

---

103. What is denormalization? Why and when is it used?

✅ **Answer:**

Denormalization is the process of combining normalized tables into fewer tables to optimize read performance by reducing joins.

✅ **Use cases**:

Data warehousing

OLAP reporting

Frequent joins slowing performance

---

104. Write a stored procedure to get total sales for a given month and year.

✅ Answer (MySQL syntax):

```
DELIMITER //

CREATE PROCEDURE GetMonthlySales(IN input_month INT, IN input_year INT)
BEGIN
  SELECT SUM(amount) AS total_sales
  FROM orders
  WHERE MONTH(order_date) = input_month AND YEAR(order_date) = input_year;
END //

DELIMITER ;
```

---

105. How do you call the stored procedure created above?

```
CALL GetMonthlySales(6, 2024);
```

---

106. Write a trigger that logs every time a new customer is added.

✅ Answer (MySQL example):

```
CREATE TRIGGER log_new_customer
AFTER INSERT ON customers
FOR EACH ROW
INSERT INTO customer_log(customer_id, log_time)
VALUES (NEW.customer_id, NOW());
```

---

107. What are indexes in SQL? How do they work?

✅ Answer:

Indexes are special lookup tables that speed up data retrieval. SQL databases use **B-trees** or **hash indexes** under the hood.

Indexes are created on columns used in WHERE, JOIN, ORDER BY.

Too many indexes → slower INSERT/UPDATEs.

---

108. Create an index on the email column in a users table.

```
CREATE INDEX idx_email ON users(email);
```

---

109. What is a composite index? Give an example.

✅ **Answer:**

A **composite index** is an index on multiple columns. It's useful for queries filtering on multiple fields.

```
CREATE INDEX idx_name_dob ON users(last_name, date_of_birth);
```

---

110. Explain clustered vs. non-clustered index.

✅ **Answer:**

Clustered Index: Determines the physical order of data in the table. Only one per table.

Non-Clustered Index: Separate from data, stores pointers to the actual rows.

---

111. What is query optimization?

✅ **Answer:**

Query optimization is the process of making a query run as efficiently as possible.

Techniques include:

Using indexes

Avoiding SELECT *

Using LIMIT for pagination

Optimizing joins and subqueries

Using EXPLAIN to analyze performance

---

112. How do you analyze a slow query in SQL?

✅ **Answer:**

Use:

```
EXPLAIN SELECT * FROM orders WHERE customer_id = 123;
```

Check:

- Index usage
- Row estimates
- Join algorithms (Nested Loop, Hash Join, etc.)

---

113. What are SQL partitions?

✅ **Answer:**

Partitioning is dividing a table into parts (horizontal slices) to improve query performance.

Types:

- Range
- List
- Hash
- Composite

---

114. Create a range-partitioned table based on order date.

✅ **Example (MySQL):**

```
CREATE TABLE orders_partitioned (
  order_id INT,
  order_date DATE,
  amount DECIMAL(10, 2)
)
PARTITION BY RANGE (YEAR(order_date)) (
  PARTITION p2023 VALUES LESS THAN (2024),
  PARTITION p2024 VALUES LESS THAN (2025),
  PARTITION pmax VALUES LESS THAN MAXVALUE
);
```

---

115. What is a materialized view? How is it different from a regular view?

✅ **Answer:**

Materialized View stores the result of a query physically and can be refreshed manually or periodically.

View is a virtual table, runs query every time.

✅ **Use case**: When performance matters for expensive aggregations.

---

116. Write a materialized view to store monthly sales.

✅ (PostgreSQL syntax):

```
CREATE MATERIALIZED VIEW monthly_sales AS
SELECT DATE_TRUNC('month', order_date) AS month,
      SUM(amount) AS total_sales
FROM orders
GROUP BY DATE_TRUNC('month', order_date);
```

---

117. Refresh the materialized view created above.

```
REFRESH MATERIALIZED VIEW monthly_sales;
```

---

118. Write a recursive query to display an employee hierarchy.

✅ Table: employees(id, name, manager_id)

```
WITH RECURSIVE emp_hierarchy AS (
  SELECT id, name, manager_id, 1 AS level
  FROM employees
  WHERE manager_id IS NULL

  UNION ALL

  SELECT e.id, e.name, e.manager_id, h.level + 1
  FROM employees e
  JOIN emp_hierarchy h ON e.manager_id = h.id
)
SELECT * FROM emp_hierarchy
ORDER BY level;
```

---

119. What are window functions in SQL? Give examples.

✅ Answer:

Window functions perform calculations across a set of rows related to the current row.

Examples:

ROW_NUMBER()

RANK()

LEAD()/LAG()

SUM() OVER (PARTITION BY ...)

✅ Example:

```
SELECT user_id, order_date,
    RANK() OVER (PARTITION BY user_id ORDER BY order_date) AS order_rank
FROM orders;
```

---

120. Write a query to find the highest-grossing product per category using window function.

```
SELECT *
FROM (
  SELECT product_id, category, revenue,
      RANK() OVER (PARTITION BY category ORDER BY revenue DESC) AS rnk
  FROM products
) ranked
WHERE rnk = 1;
```

✅ SQL Interview Questions – Part 7 (Q121–Q140)

Focus: **Data Engineering, Real-World Use Cases, Analytics, Python Integration**

---

121. What is the difference between OLTP and OLAP systems?

✅ **Answer:**

Feature	OLTP (Online Transaction Processing)	OLAP (Online Analytical Processing)
Purpose	Handle real-time transactions	Support complex analytical queries
Operations	INSERT, UPDATE, DELETE	SELECT, GROUP BY, JOIN, AGGREGATIONS
Normalization	Highly normalized	Often denormalized
Data volume	Low to medium	Large (historical data)
Example	Banking system	Business Intelligence dashboards

---

122. What is ETL? Describe each step.

✅ **Answer:**

ETL = Extract, Transform, Load

Extract: Pull raw data from source systems (e.g., SQL, APIs).

Transform: Clean, join, filter, and aggregate data.

Load: Insert data into target databases (e.g., data warehouse).

Tools: Apache Airflow, Talend, dbt, SSIS

---

123. What are common use cases for SQL in data engineering?

✅ **Answer:**

Data cleaning and transformation

Loading data into warehouses

Partitioning large datasets

Schema validation

Scheduling queries for pipelines

---

124. How do you track changes in source tables for incremental load?

✅ **Answer:**

Use **change data capture (CDC)** or audit fields like:

last_updated

created_at

row_version

✅ **Example**:

```
SELECT *
FROM orders
WHERE last_updated > '2024-01-01';
```

---

125. What are surrogate keys vs natural keys?

✅ **Answer:**

Natural Key: A key from real-world data (e.g., email, SSN)

Surrogate Key: Artificial key (e.g., auto-increment ID)

✅ Surrogate keys are preferred in data warehouses to maintain consistency.

---

126. Write SQL to pivot order data to show monthly sales per product.

✅ **Answer**:

```
SELECT product_id,
    SUM(CASE WHEN MONTH(order_date) = 1 THEN amount ELSE 0 END) AS Jan,
    SUM(CASE WHEN MONTH(order_date) = 2 THEN amount ELSE 0 END) AS Feb,
    ...
    SUM(CASE WHEN MONTH(order_date) = 12 THEN amount ELSE 0 END) AS Dec
FROM orders
GROUP BY product_id;
```

---

127. What is a slowly changing dimension (SCD)?

✅ **Answer:**

SCD handles changes in dimensional data over time.

Type 1: Overwrite old data (no history)

Type 2: Create new row (with start/end dates)

Type 3: Track limited history using columns

---

128. Create an SCD Type 2 structure for customers.

```
-- Dim_Customer with start_date, end_date, is_current

INSERT INTO dim_customer (cust_id, name, start_date, end_date, is_current)
VALUES (101, 'John Smith', '2024-01-01', '9999-12-31', 1);
```

Update old record:

```
UPDATE dim_customer
SET end_date = '2025-01-01', is_current = 0
WHERE cust_id = 101 AND is_current = 1;

-- Then insert new row for new name, same cust_id
```

---

129. Write a SQL query to calculate Year-over-Year (YoY) growth.

✅ **Answer:**

```
SELECT year,
    revenue,
    LAG(revenue) OVER (ORDER BY year) AS prev_year_revenue,
```

```
    ROUND(((revenue - LAG(revenue) OVER (ORDER BY year)) /
        LAG(revenue) OVER (ORDER BY year)) * 100, 2) AS YoY_growth
FROM yearly_sales;
```

---

130. What are common window function pitfalls?

✅ **Answer:**

Using RANK() instead of ROW_NUMBER() in deduplication

Forgetting to PARTITION BY

Misinterpreting cumulative SUM() without ordering

Performance issues on large unindexed tables

---

131. What is the difference between RANK(), DENSE_RANK(), and ROW_NUMBER()?

Function	Behavior
RANK()	Skips ranks on tie (e.g., 1, 1, 3)
DENSE_RANK()	No gaps (e.g., 1, 1, 2)
ROW_NUMBER()	Unique row ID (e.g., 1, 2, 3)

---

132. Write a query to find second highest salary per department.

```
SELECT *
FROM (
  SELECT dept_id, emp_name, salary,
      DENSE_RANK() OVER (PARTITION BY dept_id ORDER BY salary DESC) AS rnk
  FROM employees
) ranked
WHERE rnk = 2;
```

---

133. Write a CTE to find running total of sales.

```
WITH sales_cte AS (
  SELECT order_date, amount,
      SUM(amount) OVER (ORDER BY order_date) AS running_total
  FROM orders
)
SELECT * FROM sales_cte;
```

---

134. What is the difference between WHERE and HAVING clause?

✅ **Answer:**

WHERE: Filters rows before grouping

HAVING: Filters after aggregation

```
SELECT customer_id, SUM(amount) AS total
FROM orders
GROUP BY customer_id
HAVING total > 1000;
```

---

135. What is the difference between UNION and UNION ALL?

✅ **Answer:**

UNION: Removes duplicates

UNION ALL: Includes all rows (faster)

```
SELECT name FROM A
UNION
SELECT name FROM B;
```

---

136. Write a query to find customers who ordered in both Jan and Feb.

```
SELECT customer_id
FROM orders
WHERE MONTH(order_date) IN (1, 2)
GROUP BY customer_id
HAVING COUNT(DISTINCT MONTH(order_date)) = 2;
```

---

137. What is the use of COALESCE in SQL?

✅ **Answer:**

Returns the first non-null value from a list.

```
SELECT name, COALESCE(email, phone, 'N/A') AS contact_info
FROM users;
```

---

138. What are common real-world SQL use cases?

Dashboard reports

Product analytics

Fraud detection

Inventory tracking

User retention analysis

Financial forecasting

---

139. Write a query to find monthly retention of users.

```
WITH monthly_users AS (
  SELECT user_id, DATE_TRUNC('month', signup_date) AS signup_month,
       DATE_TRUNC('month', last_active_date) AS active_month
  FROM users
)
SELECT signup_month, active_month, COUNT(*) AS retained_users
FROM monthly_users
GROUP BY signup_month, active_month
ORDER BY signup_month, active_month;
```

---

140. How can Python be used with SQL for analytics?

✅ **Answer:**

Use libraries like pandas, sqlalchemy, sqlite3, psycopg2

Run SQL queries from Python

Load results into DataFrames

Combine SQL + pandas for full workflows

```
import pandas as pd
import sqlite3

conn = sqlite3.connect('sales.db')
df = pd.read_sql("SELECT * FROM orders", conn)
print(df.head())
```

✅ SQL Interview Questions – Part 8 (Q141–Q160)

Focus: **Real-World Scenarios, Projects, Complex Queries, Optimization**

---

141. Write a query to get the top 3 products by revenue for each category.

```
SELECT *
FROM (
  SELECT category_id, product_id, SUM(amount) AS revenue,
       RANK() OVER (PARTITION BY category_id ORDER BY SUM(amount) DESC) AS rnk
  FROM sales
  GROUP BY category_id, product_id
```

```
) ranked
WHERE rnk <= 3;
```

---

142. How would you detect duplicate records in a customer table?

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

---

143. How do you delete duplicate rows but keep one?

```
DELETE FROM customers
WHERE id NOT IN (
  SELECT MIN(id)
  FROM customers
  GROUP BY name, email
);
```

---

144. Write a query to find orders that have all products from a given category.

```
SELECT order_id
FROM order_items oi
JOIN products p ON oi.product_id = p.id
WHERE p.category = 'Electronics'
GROUP BY order_id
HAVING COUNT(DISTINCT p.id) = (
    SELECT COUNT(DISTINCT id)
    FROM products
    WHERE category = 'Electronics'
);
```

---

145. What is query optimization? How do you improve performance?

✅ **Answer:**

Query optimization improves speed and resource usage by:

- Adding proper indexes
- Avoiding SELECT *
- Reducing subqueries and using joins
- Filtering early (WHERE clause)
- Using EXPLAIN PLAN to analyze execution

---

146. How do indexes affect performance?

Improve: Search, JOIN, ORDER BY, WHERE speed

Slow down: INSERT, UPDATE, DELETE due to index maintenance

Use indexes **only on frequently queried columns**.

---

147. How do you debug a slow SQL query?

✅ **Steps:**

Use EXPLAIN or EXPLAIN ANALYZE

Check for full table scans

Add appropriate indexes

Reduce large joins or use temp tables

Break into smaller subqueries

---

148. Write a query to find the percentage contribution of each department's salary to total.

```
SELECT department_id,
    SUM(salary) AS dept_total,
    ROUND(SUM(salary) * 100.0 / (SELECT SUM(salary) FROM employees), 2) AS
percent_contribution
FROM employees
GROUP BY department_id;
```

---

149. How do you find gaps in a sequence of order IDs?

```
SELECT o1.order_id + 1 AS missing_id
FROM orders o1
LEFT JOIN orders o2 ON o1.order_id + 1 = o2.order_id
WHERE o2.order_id IS NULL;
```

---

150. What are materialized views?

✅ **Answer:**

A **materialized view** stores query results physically.

Faster than regular views for frequent reporting.

Can be refreshed manually or on a schedule.

```
CREATE MATERIALIZED VIEW sales_summary AS
SELECT product_id, SUM(amount) AS total
FROM sales
GROUP BY product_id;
```

---

151. What is denormalization? When is it used?

✅ **Answer:**

Denormalization is adding redundant data to avoid joins.

Used in OLAP/data warehousing for read-heavy workloads.

Example: storing total sales in product table for faster lookup.

---

152. Write a query to rank users by total purchase and assign levels: Platinum (Top 10%), Gold (Next 20%), etc.

```
WITH ranked AS (
  SELECT user_id, SUM(amount) AS total_spent,
       NTILE(10) OVER (ORDER BY SUM(amount) DESC) AS decile
  FROM purchases
  GROUP BY user_id
)
SELECT user_id, total_spent,
      CASE
        WHEN decile = 1 THEN 'Platinum'
        WHEN decile <= 3 THEN 'Gold'
        WHEN decile <= 6 THEN 'Silver'
        ELSE 'Bronze'
      END AS level
FROM ranked;
```

---

153. How do you handle dirty data in SQL pipelines?

✅ **Approaches:**

Use NULLIF, COALESCE, IS NULL to handle missing values

Filter out bad rows: WHERE LENGTH(name) > 0

Use CAST() or TRY_CAST() for data type issues

Validate ranges: WHERE age BETWEEN 0 AND 120

---

154. Write SQL to compare sales of the same month this year vs last year.

```
SELECT
   EXTRACT(MONTH FROM order_date) AS month,
   SUM(CASE WHEN EXTRACT(YEAR FROM order_date) = 2025 THEN amount END) AS
this_year,
   SUM(CASE WHEN EXTRACT(YEAR FROM order_date) = 2024 THEN amount END) AS
last_year
FROM orders
WHERE EXTRACT(YEAR FROM order_date) IN (2024, 2025)
GROUP BY month
ORDER BY month;
```

---

155. What is the benefit of using WITH (CTE) over subqueries?

✅ **Answer:**

Increases **readability** and **reusability**

Supports recursion

Avoids repeating complex subqueries

Can be broken down and debugged step-by-step

---

156. Write a recursive CTE to calculate factorial.

```
WITH RECURSIVE factorial(n, fact) AS (
  SELECT 1, 1
  UNION ALL
  SELECT n + 1, (n + 1) * fact
  FROM factorial
  WHERE n < 5
)
SELECT * FROM factorial;
```

---

157. What are some common SQL anti-patterns to avoid?

✅ **List:**

Using `SELECT *`

Not using `WHERE` for DELETE/UPDATE

Using subqueries instead of JOINs

Ignoring NULLs in comparisons

Over-indexing or indexing low-cardinality columns

---

158. Write SQL to get consecutive login streaks per user.

```
WITH login_data AS (
  SELECT user_id, login_date,
        ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY login_date) -
        DATEDIFF(day, '2000-01-01', login_date) AS grp
  FROM logins
),
streaks AS (
  SELECT user_id, MIN(login_date) AS start_date, MAX(login_date) AS end_date, COUNT(*) AS
streak_days
  FROM login_data
  GROUP BY user_id, grp
)
SELECT * FROM streaks
WHERE streak_days >= 3;
```

---

159. What is sharding in SQL databases?

✅ **Answer:**

Sharding splits large tables across **multiple servers** by:

- Region
- Customer ID
- Hash

Improves scalability and parallel processing.

---

160. How do you load large data (1M+ rows) efficiently into a SQL table?

✅ **Answer:**

- Use BULK INSERT or COPY command
- Disable indexes during load
- Load in batches (e.g., 10k at a time)
- Avoid row-by-row inserts in loops

```
COPY sales FROM '/path/to/file.csv' DELIMITER ',' CSV HEADER;
```

✅ SQL Interview Questions – Part 9 (Q161–Q180)

Focus: **Security, Access Control, Python + SQL, Normalization, Reporting, Use Cases**

---

161. What is SQL injection? How can you prevent it?

✅ **Answer:**

SQL injection is a security vulnerability where malicious input modifies SQL queries.

Prevention:

- Use **parameterized queries** or **prepared statements**
- Avoid concatenating user input into queries
- Use ORMs like SQLAlchemy or Django ORM

Example in Python (safe):

```
cursor.execute("SELECT * FROM users WHERE username = %s", (username,))
```

---

162. How do you manage user access in SQL databases?

✅ **Answer:**

- Use `CREATE USER` and `GRANT` statements
- Follow **principle of least privilege**

```
CREATE USER analyst WITH PASSWORD 'securepass';
GRANT SELECT ON sales TO analyst;
```

---

163. What are roles in SQL databases?

✅ **Answer:**
Roles are **collections of privileges** that can be assigned to users.

```
CREATE ROLE readonly;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO readonly;
GRANT readonly TO user_1;
```

---

164. What are the different levels of normalization?

✅ **Answer:**

- **1NF**: Atomic values

2NF: No partial dependency

3NF: No transitive dependency

BCNF: Stronger form of 3NF

Normalization reduces redundancy and improves integrity.

---

165. Give an example of 1NF to 3NF transformation.

Unnormalized Table:

StudentID Name Course1 Course2

To 1NF:

StudentID Name Course

To 2NF (Split course):

StudentID Name
CourseID Course

To 3NF (Split names):

| StudentID | FirstName | LastName |

---

166. How do you connect SQL to Python?

✅ **Answer:**
Use libraries like psycopg2, sqlite3, mysql-connector, SQLAlchemy.

Example using sqlite3**:**

```
import sqlite3
conn = sqlite3.connect('db.sqlite')
cursor = conn.cursor()
cursor.execute("SELECT * FROM customers")
```

---

167. How do you export SQL query output to CSV using Python?

```
import csv
import psycopg2

conn = psycopg2.connect(...)
cursor = conn.cursor()
```

```
cursor.execute("SELECT * FROM sales")

with open("sales.csv", "w", newline='') as f:
    writer = csv.writer(f)
    writer.writerow([desc[0] for desc in cursor.description])  # column names
    writer.writerows(cursor.fetchall())
```

---

168. What is data warehousing?

✅ **Answer:**

A **data warehouse** stores large volumes of structured data for analysis. It integrates data from multiple sources for OLAP.

Examples:

- Snowflake
- Amazon Redshift
- Google BigQuery

---

169. What is ETL?

✅ **Answer:**

- **Extract** data from sources
- **Transform** (clean, join, normalize)
- **Load** into data warehouse

Can be automated with tools like **Apache Airflow**, **Talend**, **dbt**.

---

170. Write SQL to pivot monthly revenue from rows to columns.

```
SELECT product_id,
    SUM(CASE WHEN MONTH(order_date) = 1 THEN revenue END) AS Jan,
    SUM(CASE WHEN MONTH(order_date) = 2 THEN revenue END) AS Feb,
    ...
FROM sales
GROUP BY product_id;
```

---

171. What is a surrogate key vs a natural key?

✅ **Answer:**

Natural Key: Comes from real-world data (e.g., email)

Surrogate Key: Artificial ID (e.g., auto-increment id)

Surrogate keys are preferred in warehouses for simplicity.

---

172. Write SQL to unpivot monthly columns into rows.

```
SELECT product_id, 'Jan' AS month, jan AS revenue FROM sales_summary
UNION ALL
SELECT product_id, 'Feb', feb FROM sales_summary
...
```

---

173. What is a star schema vs snowflake schema?

✅ **Star Schema:**

One fact table + denormalized dimension tables

Faster for querying

✅ **Snowflake Schema:**

Fact table + normalized dimensions (more joins)

---

174. Write SQL to return customers who made purchases every month in 2024.

```
SELECT customer_id
FROM purchases
WHERE YEAR(date) = 2024
GROUP BY customer_id
HAVING COUNT(DISTINCT MONTH(date)) = 12;
```

---

175. What are views and how are they used in reporting?

✅ **Answer:**

A **view** is a saved SQL query

Abstracts complex joins

Used for security and simplified reporting

```
CREATE VIEW monthly_sales AS
SELECT MONTH(date) AS month, SUM(amount) FROM orders GROUP BY MONTH(date);
```

---

176. How do you detect outliers in SQL?

✅ **Example using Z-score method:**

```
WITH stats AS (
  SELECT AVG(amount) AS avg_amt, STDDEV(amount) AS std_amt
  FROM sales
)
SELECT * FROM sales, stats
WHERE ABS(amount - avg_amt) > 3 * std_amt;
```

---

177. How do you automate SQL reports daily?

✅ **Options:**

Use **cron job** + Python script

Use **Airflow** DAGs

Use **BI Tools** (Power BI, Tableau, Looker)

Email results using script + SMTP

---

178. What is an execution plan?

✅ **Answer:**

An **execution plan** shows how a query will be run (e.g., index scan, nested loop). Helps in debugging performance.

```
EXPLAIN SELECT * FROM orders WHERE customer_id = 101;
```

---

179. What are OLTP vs OLAP systems?

Feature	OLTP	OLAP
Purpose	Real-time transactions	Analytical processing
Data	Highly normalized	Denormalized
Examples	Banking, eCommerce	BI dashboards, reporting
Queries	Short and fast	Complex, aggregated

---

180. What is the difference between UNION and UNION ALL?

Clause	Description
UNION	Removes duplicates
UNION ALL	Includes all rows (faster, no dedup)

```
SELECT name FROM employees
UNION
SELECT name FROM managers;
```

✅ Q181. You need to find the employees who have taken the most sick leaves in the last year.

Answer:

Use aggregation with filtering on leave_type and leave_date.

```
SELECT employee_id, COUNT(*) AS sick_leaves
FROM employee_leaves
WHERE leave_type = 'Sick'
  AND leave_date >= DATE_SUB(CURDATE(), INTERVAL 1 YEAR)
GROUP BY employee_id
ORDER BY sick_leaves DESC
LIMIT 1;
```

✅ Q182. Write a SQL query to calculate the customer retention rate for each month.

Answer:

Assuming logins or orders are tracked monthly:

```
WITH monthly_customers AS (
  SELECT customer_id, DATE_FORMAT(order_date, '%Y-%m') AS month
  FROM orders
  GROUP BY customer_id, month
),
retention AS (
  SELECT curr.month,
         COUNT(DISTINCT curr.customer_id) AS total_customers,
         COUNT(DISTINCT prev.customer_id) AS retained_customers
  FROM monthly_customers curr
  LEFT JOIN monthly_customers prev
    ON curr.customer_id = prev.customer_id
   AND curr.month = DATE_FORMAT(DATE_ADD(STR_TO_DATE(prev.month, '%Y-%m'),
INTERVAL 1 MONTH), '%Y-%m')
  GROUP BY curr.month
)
SELECT month,
       retained_customers,
       total_customers,
       ROUND(retained_customers * 100.0 / total_customers, 2) AS retention_rate
FROM retention;
```

✅ Q183. You have a table with time logs. Write a query to calculate total hours worked per employee per week.

Answer:

```
SELECT employee_id,
    YEAR(log_date) AS year,
    WEEK(log_date) AS week,
    SUM(hours_worked) AS total_hours
FROM time_logs
GROUP BY employee_id, year, week;
```

✅ Q184. Write a query to list top 5 departments with highest total salary expense.

Answer:

```
SELECT department_id, SUM(salary) AS total_salary
FROM employees
GROUP BY department_id
ORDER BY total_salary DESC
LIMIT 5;
```

✅ Q185. Write a SQL query to show employees whose salaries are above the department average.

Answer:

```
SELECT e.employee_id, e.name, e.salary, e.department_id
FROM employees e
JOIN (
  SELECT department_id, AVG(salary) AS avg_salary
  FROM employees
  GROUP BY department_id
) d ON e.department_id = d.department_id
WHERE e.salary > d.avg_salary;
```

✅ Q186. Write a SQL query to show the cumulative sum of sales by date.

Answer:

```
SELECT order_date,
    SUM(sales_amount) AS daily_sales,
    SUM(SUM(sales_amount)) OVER (ORDER BY order_date) AS cumulative_sales
FROM orders
GROUP BY order_date;
```

✅ Q187. Write a query to get the running total of orders for each customer.

Answer:

```
SELECT customer_id,
      order_date,
      order_id,
      SUM(order_total) OVER (PARTITION BY customer_id ORDER BY order_date) AS
running_total
FROM orders;
```

✅ Q188. You want to flag customers who made their first purchase in the last month.

Answer:

```
SELECT customer_id,
      MIN(order_date) AS first_purchase_date,
      CASE
        WHEN MIN(order_date) >= DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
        THEN 'New'
        ELSE 'Existing'
      END AS customer_status
FROM orders
GROUP BY customer_id;
```

✅ Q189. Write a query to return products with highest price per category.

Answer:

```
SELECT p.*
FROM products p
JOIN (
  SELECT category_id, MAX(price) AS max_price
  FROM products
  GROUP BY category_id
) maxed
ON p.category_id = maxed.category_id AND p.price = maxed.max_price;
```

✅ Q190. Write a SQL query to calculate churn rate.

Answer:

Assume churned customers = didn't return in current month.

```
WITH current_month AS (
  SELECT DISTINCT customer_id
  FROM orders
  WHERE MONTH(order_date) = MONTH(CURDATE())
    AND YEAR(order_date) = YEAR(CURDATE())
),
previous_month AS (
  SELECT DISTINCT customer_id
  FROM orders
  WHERE MONTH(order_date) = MONTH(DATE_SUB(CURDATE(), INTERVAL 1 MONTH))
```

```
   AND YEAR(order_date) = YEAR(DATE_SUB(CURDATE(), INTERVAL 1 MONTH))
)
SELECT COUNT(*) AS churned_customers,
     ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM previous_month), 2) AS churn_rate
FROM previous_month
WHERE customer_id NOT IN (SELECT customer_id FROM current_month);
```

✅ Q191. Write a query to get the second most sold product.

Answer:

```
SELECT product_id, SUM(quantity) AS total_quantity
FROM order_items
GROUP BY product_id
ORDER BY total_quantity DESC
LIMIT 1 OFFSET 1;
```

✅ Q192. Write a SQL query to find the average order value for each product category.

Answer:

```
SELECT c.category_name,
     ROUND(AVG(o.total_amount), 2) AS avg_order_value
FROM orders o
JOIN order_items oi ON o.order_id = oi.order_id
JOIN products p ON oi.product_id = p.product_id
JOIN categories c ON p.category_id = c.category_id
GROUP BY c.category_name;
```

✅ Q193. Write a SQL query to compare this year's monthly sales to last year.

Answer:

```
SELECT MONTH(order_date) AS month,
     SUM(CASE WHEN YEAR(order_date) = YEAR(CURDATE()) THEN total_amount ELSE 0
END) AS this_year_sales,
     SUM(CASE WHEN YEAR(order_date) = YEAR(CURDATE()) - 1 THEN total_amount ELSE 0
END) AS last_year_sales
FROM orders
GROUP BY MONTH(order_date);
```

✅ Q194. Write a query to return products with increasing month-over-month sales.

Answer:

```
WITH monthly_sales AS (
  SELECT product_id,
```

```
        DATE_FORMAT(order_date, '%Y-%m') AS month,
        SUM(quantity) AS total_quantity
  FROM order_items oi
  JOIN orders o ON oi.order_id = o.order_id
  GROUP BY product_id, month
),
ranked AS (
  SELECT *,
        ROW_NUMBER() OVER (PARTITION BY product_id ORDER BY month) AS rn
  FROM monthly_sales
),
diffs AS (
  SELECT m1.product_id,
        m1.month,
        m1.total_quantity,
        m1.total_quantity - COALESCE(m2.total_quantity, 0) AS diff
  FROM ranked m1
  LEFT JOIN ranked m2
    ON m1.product_id = m2.product_id AND m1.rn = m2.rn + 1
)
SELECT product_id
FROM diffs
GROUP BY product_id
HAVING MIN(diff) > 0;
```

✅ Q195. Write a SQL query to find customers who place repeat orders every month.

Answer:

```
WITH monthly_orders AS (
  SELECT customer_id, DATE_FORMAT(order_date, '%Y-%m') AS month
  FROM orders
  GROUP BY customer_id, month
),
repeat_customers AS (
  SELECT customer_id, COUNT(DISTINCT month) AS active_months
  FROM monthly_orders
  GROUP BY customer_id
)
SELECT customer_id
FROM repeat_customers
WHERE active_months = (SELECT COUNT(DISTINCT DATE_FORMAT(order_date, '%Y-%m'))
FROM orders);
```

✅ Q196. Write a SQL query to get the longest-serving employees.

Answer:

```
SELECT employee_id, name, hire_date
FROM employees
ORDER BY hire_date ASC
LIMIT 5;
```

✅ Q197. Write a SQL query to get the total revenue from new customers (first-time buyers).

Answer:

```
WITH first_orders AS (
  SELECT customer_id, MIN(order_date) AS first_order
  FROM orders
  GROUP BY customer_id
)
SELECT SUM(o.total_amount) AS total_revenue
FROM orders o
JOIN first_orders f ON o.customer_id = f.customer_id AND o.order_date = f.first_order;
```

✅ Q198. Write a SQL query to identify customers who upgraded to premium membership.

Answer:

```
SELECT customer_id, MIN(change_date) AS upgrade_date
FROM membership_changes
WHERE old_level != 'Premium' AND new_level = 'Premium'
GROUP BY customer_id;
```

✅ Q199. Write a SQL query to detect sudden spikes in sales for a product.

Answer:

```
WITH sales_by_day AS (
  SELECT product_id, order_date, SUM(quantity) AS total
  FROM order_items
  JOIN orders USING(order_id)
  GROUP BY product_id, order_date
),
rolling_avg AS (
  SELECT *,
       AVG(total) OVER (PARTITION BY product_id ORDER BY order_date ROWS BETWEEN 7
PRECEDING AND 1 PRECEDING) AS avg_prev_week
  FROM sales_by_day
)
SELECT *
FROM rolling_avg
WHERE total > 2 * avg_prev_week;
```

✅ Q200. Write a SQL query to detect anomalies in daily sales compared to a moving average.

Answer:

```
WITH daily_sales AS (
```

```
  SELECT order_date, SUM(total_amount) AS sales
  FROM orders
  GROUP BY order_date
),
moving_avg AS (
  SELECT order_date, sales,
        AVG(sales) OVER (ORDER BY order_date ROWS BETWEEN 7 PRECEDING AND
CURRENT ROW) AS avg_sales
  FROM daily_sales
)
SELECT order_date, sales, avg_sales
FROM moving_avg
WHERE sales > 1.5 * avg_sales OR sales < 0.5 * avg_sales;
```
