Navigating SQL Interviews (Part II)

Listen Carefully
Pay close attention to the interviewer's question and make sure you understand the requirements clearly. If there's any ambiguity, don't hesitate to ask for clarification.

Plan Your Approach
Before writing any code, take a moment to plan your approach. Think about the steps you need to take to solve the problem and consider the most efficient way to achieve the desired outcome.

Break Down the Problem
Break down the problem into smaller, manageable parts. Identify the key components of the query and think about how you can tackle each part individually.

Start Simple
Begin by writing the basic structure of the query without worrying too much about optimization or complexity. This will help you get started and build momentum.

Optimize and Refine
Once you have a working solution, look for opportunities to optimize and refine your query. Consider factors like performance, readability, and maintainability.

Ask Questions
Don't be afraid to ask questions or seek clarification if you're unsure about something. Interviewers appreciate candidates who demonstrate curiosity and a willingness to learn.

Practice, Practice, Practice
Practice writing SQL queries regularly to improve your skills and build confidence. Work on a variety of problems and familiarize yourself with different SQL functions, clauses, and techniques.

Stay Calm and Confident
Lastly, stay calm and confident during the interview. Remember that it's okay to take your time and think through the problem carefully. Approach each question with a positive attitude and do your best to showcase your abilities.

The subsequent questions are commonly posed during interviews for roles in data analysis and data science. The complexity level fluctuates based on the interviewer's preferences and the candidate's responses to the questions. This repository serves as a valuable resource for promptly reviewing pertinent topics and ensuring readiness for the interview process.

DDL

Write a SQL command to create a database named `CompanyDB`.

`CREATE DATABASE CompanyDB;`.

How do you create a table named `Departments` with columns `DeptID` (integer) and `DeptName` (string)?

```
CREATE TABLE Departments (
    DeptID INT PRIMARY KEY,
    DeptName VARCHAR(100) NOT NULL
);
```

Write a command to create a table `Projects` with a foreign key to `Departments`.

```
CREATE TABLE Projects (
    ProjectID INT PRIMARY KEY,
    ProjectName VARCHAR(100) NOT NULL,
    DeptID INT,
    FOREIGN KEY (DeptID) REFERENCES Departments(DeptID)
);
```

What is the SQL command to add a column `StartDate` (date) to the `Projects` table?

`ALTER TABLE Projects ADD StartDate DATE;`.

Write a command to create an index on the `DeptName` column in the `Departments` table.

`CREATE INDEX idx_deptname ON Departments(DeptName);`.

How do you rename a table from `OldTable` to `NewTable`?

`ALTER TABLE OldTable RENAME TO NewTable;`.

Write a command to drop the `Projects` table.

`DROP TABLE Projects;`.

How do you truncate the `Employees` table?

`TRUNCATE TABLE Employees;`.

Write a command to remove a column `DeptID` from the `Projects` table.

`ALTER TABLE Projects DROP COLUMN DeptID;`.

How do you specify a primary key when creating a table?

`CREATE TABLE table_name (column1 datatype PRIMARY KEY, column2 datatype, ...);`

How do you add a foreign key constraint when creating a table?

`CREATE TABLE table_name (column1 datatype, column2 datatype, FOREIGN KEY (column2) REFERENCES other_table(column));`

How do you create a table with a unique constraint?

`CREATE TABLE table_name (column1 datatype, column2 datatype UNIQUE, ...);`

How do you create a table with a composite primary key?

CREATE TABLE table_name (column1 datatype, column2 datatype, PRIMARY KEY (column1, column2));

How do you add a check constraint to ensure a column 'age' is greater than 18?

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

How to add or remove primary constraint to a column?

- ALTER TABLE Persons ADD PRIMARY KEY (ID);
- ALTER TABLE table_name DROP PRIMARY KEY_

How do you add a NOT NULL constraint to a column?

`ALTER TABLE table_name MODIFY column_name datatype NOT NULL;`.

DML

How do you insert a new record into the `Departments` table?

```
INSERT INTO Departments (DeptID, DeptName)
VALUES (1, 'HR');
```

Write a command to update the `DeptName` of `DeptID` 1 to 'Human Resources'.

```
UPDATE Departments
SET DeptName = 'Human Resources'
WHERE DeptID = 1;
```

How do you delete a record from the `Departments` table where `DeptID` is 1?

`DELETE FROM Departments WHERE DeptID = 1;`.

Write a command to insert multiple records into the `Departments` table.

```
INSERT INTO Departments (DeptID, DeptName)
VALUES (2, 'IT'), (3, 'Finance');
```

Write a query to update the salary of an employee with ID 5 to 70000.

`UPDATE employees SET salary = 70000 WHERE id = 5;`

Write a query to delete an employee with the name 'Jane Smith'.

`DELETE FROM employees WHERE name = 'Jane Smith';`

Write a query to insert multiple new products.

`INSERT INTO products (name, price) VALUES ('Product1', 100), ('Product2', 200), ('Product3', 300);`

Write a query to update the price of all products to 10% more than their current price.

`UPDATE products SET price = price * 1.10;`

Write a query to delete all orders placed before 2022.

`DELETE FROM orders WHERE order_date < '2022-01-01';`

DQL

What is the syntax to select distinct records?

SELECT DISTINCT column_name FROM table_name;

Write a query to find departments with average salaries greater than 50000.

SELECT department_id, AVG(salary) FROM employees GROUP BY department_id
HAVING AVG(salary) > 50000;

Write a query to find the department with the highest average salary.

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

Write a query to find the top 3 highest salaries from the `employees` table.

`SELECT salary FROM employees ORDER BY salary DESC LIMIT 3;`.

Write a query using the HAVING clause to filter groups of customers (customer_id) with a count of orders (order_id) greater than 2.

SELECT customer_id, COUNT(order_id) AS order_count
FROM orders
GROUP BY customer_id
HAVING COUNT(order_id) > 2;

How do you find the total number of employees in each department?

`SELECT department_id, COUNT(*) FROM employees GROUP BY department_id;`.

Write a query to find the average salary of employees in each department.

`SELECT department_id, AVG(salary) FROM employees GROUP BY department_id;`.

How do you find departments with more than 5 employees?

`SELECT department_id, COUNT(*) FROM employees GROUP BY department_id HAVING COUNT(*) > 5;`.

Functions

How can you find the maximum value in a column?

SELECT MAX(column_name) FROM table_name;

How can you extract a substring from a string?

SUBSTRING(string, start, length);

Extract the first 5 characters from `code` column in `items` table.

SELECT SUBSTRING(code, 1, 5) FROM items;

Write a query to concatenate first and last name from a table `employees`.

SELECT CONCAT(first_name, ' ', last_name) AS full_name FROM employees;

How do you replace all occurrences of 'a' with 'e' in a column?

SELECT REPLACE(column_name, 'a', 'e') FROM table_name;

Write a query to convert the `description` column to uppercase in `products` table.

SELECT UPPER(description) FROM products;

Filtering using the Like Operator

What is the purpose of the LIKE operator?

LIKE operator is used for pattern matching in SQL.

How do you filter records where a column starts with 'A'?

SELECT * FROM table_name WHERE column_name LIKE 'A%';

Write a query to find records where the column contains 'abc' anywhere.

SELECT * FROM table_name WHERE column_name LIKE '%abc%';

How do you find records where a column ends with 'z'?

SELECT * FROM table_name WHERE column_name LIKE '%z';

Write a query to find records where the column has 'a' as the second character.

```
SELECT * FROM table_name WHERE column_name LIKE '_a%';
```

Joins

Write a query to perform an INNER JOIN between two tables.

```
SELECT a.column1, b.column2
FROM table1 a INNER JOIN table2 b
ON a.common_column = b.common_column;
```

Write a query to perform a LEFT JOIN between two tables.

```
SELECT a.column1, b.column2
FROM table1 a LEFT JOIN table2 b
ON a.common_column = b.common_column;
```

Write a query to perform a RIGHT JOIN between two tables.

```
SELECT a.column1, b.column2
FROM table1 a RIGHT JOIN table2 b
ON a.common_column = b.common_column;
```

Write a query to perform a FULL OUTER JOIN between two tables.

```
SELECT a.column1, b.column2
FROM table1 a FULL OUTER JOIN table2 b
ON a.common_column = b.common_column;
```

How do you perform a self join?

```
SELECT a.column1, b.column2
FROM table_name a JOIN table_name b
ON a.common_column = b.common_column;
```

How do you perform a compound join?

```
 SELECT * FROM table1 a JOIN table2 b
ON a.column1 = b.column1 AND a.column2 = b.column2;
```

Write a query to join four tables.

```
SELECT * FROM table1 a
JOIN table2 b ON a.common_column = b.common_column
```

JOIN table3 c ON b.common_column = c.common_column
JOIN table4 d ON c.common_column = d.common_column;`

Write a query to perform a cross join.

A cross join returns the Cartesian product of two tables, meaning it combines all rows from the first table with all rows from the second table.

SELECT a.column1, b.column2 FROM table1 a CROSS JOIN table2 b;

Write a query to find all employees who earn more than the average salary using a join.

SELECT e.* FROM employees e
JOIN (SELECT AVG(salary) AS avg_salary FROM employees) avg
ON e.salary > avg.avg_salary;

Sub Queries

Write a query using a subquery to find employees with salaries higher than the average salary.

SELECT * FROM employees WHERE salary > (SELECT AVG(salary) FROM employees);

Write a query using a subquery in the FROM clause.

SELECT sub.column1, sub.column2 FROM (SELECT column1, column2 FROM table_name) sub;

Write a query to find the second highest salary in the employees table.

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

Write a query to find all employees who earn more than the average salary using a join.

SELECT e.* FROM employees e
JOIN (SELECT AVG(salary) AS avg_salary FROM employees) avg
ON e.salary > avg.avg_salary;

WINDOW FUNCTIONS

Write a query to assign a unique rank to each employee within a department in descending order based on salary.

```
SELECT
    employee_id,
    department_id,
    salary,
    RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) AS rank
FROM employees;
```

Write a query to calculate a running total of sales amounts using window functions.

```
SELECT
    sales_date,
    sales_amount,
    SUM(sales_amount) OVER (ORDER BY sales_date) AS running_total
FROM sales;
```

Write a query to calculate the average salary of employees within each department using window functions.

```
SELECT
    department_id,
    employee_id,
    salary,
    AVG(salary) OVER (PARTITION BY department_id) AS avg_salary
FROM employees;
```

Write a query to get the previous row's salary value in the result set.

```
SELECT
    employee_id,
    salary,
    LAG(salary, 1) OVER (ORDER BY employee_id) AS previous_salary
FROM employees;
```

Write a query to get the next row's value in the result set.

```
SELECT
    employee_id,
    salary,
    LEAD(salary, 1) OVER (ORDER BY employee_id) AS next_salary
FROM employees;
```

Write a query to distribute the rows in an ordered partition based on the salary column into four equal groups.

```
SELECT
    employee_id,
    salary,
    NTILE(4) OVER (ORDER BY salary DESC) AS quartile
FROM employees;
```

Write a query to find the second highest salary in each department using window functions.

```
SELECT
    department_id,
    employee_id,
    salary
FROM (
    SELECT
        department_id,
        employee_id,
        salary,
        DENSE_RANK() OVER (PARTITION BY department_id ORDER BY salary DESC)
AS rank
    FROM employees
) AS ranked
WHERE rank = 2;
```

Write a query to get the first salary in each department ordered by salary.

```
SELECT
    employee_id,
    department_id,
    salary,
    FIRST_VALUE(salary) OVER (PARTITION BY department_id ORDER BY salary
DESC) AS first_salary
FROM employees;
```

Write a query to calculate a moving average over the last 3 months of sales.

```
SELECT
    sales_date,
    sales_amount,
    AVG(sales_amount) OVER (ORDER BY sales_date ROWS BETWEEN 2 PRECEDING
AND CURRENT ROW) AS moving_avg
```

```
FROM sales;
```

Write a query to list all employees, showing their current salary and the previous month's salary. (Use column hire_date to get previous month’s data.)

```
SELECT
    employee_id,
    salary,
    LAG(salary, 1) OVER (ORDER BY hire_date) AS previous_month_salary
FROM employees;
```

Write a query to find gaps in hire dates.

```
SELECT
    employee_id,
    hire_date,
    LAG(hire_date, 1) OVER (ORDER BY hire_date) AS previous_hire_date,
    DATEDIFF(day, LAG(hire_date, 1) OVER (ORDER BY hire_date), hire_date) AS
gap_days
FROM employees;
```

Write a query to calculate the cumulative distribution of salaries within a company.

```
SELECT
    employee_id,
    salary,
    CUME_DIST() OVER (ORDER BY salary) AS cumulative_distribution
FROM employees;
```

Write a query to calculate the difference between the current row's value and the value from the previous row.

```
SELECT
    employee_id,
    salary,
    salary - LAG(salary, 1) OVER (ORDER BY employee_id) AS salary_difference
FROM employees;
```

Write a query to perform data deduplication based on employee_id.

```
SELECT
employee_id,
department_id,
```

```
   salary
FROM (
   SELECT
      employee_id,
      department_id,
      salary,
      ROW_NUMBER() OVER (PARTITION BY employee_id ORDER BY department_id,
salary) AS row_num
   FROM employees
) AS numbered
WHERE row_num = 1;
```

Write a query to generate a sequential number for each row, resetting the number for each department.

```
   SELECT
      employee_id,
      department_id,
      ROW_NUMBER() OVER (PARTITION BY department_id ORDER BY employee_id) AS
row_num
   FROM employees;
```

Write a query to calculate a percentile value (50th percentile) of salaries within each department.

```
   SELECT
      employee_id,
      salary,
      PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY salary) OVER (PARTITION
BY department_id) AS median_salary
   FROM employees;
```

Write a query to return the second highest salary in each department using `NTH_VALUE`.

```
   SELECT
      employee_id,
      salary,
      department_id,
      NTH_VALUE(salary, 2) OVER (PARTITION BY department_id ORDER BY salary
DESC) AS second_highest_salary
   FROM employees;
```

Write a query to list the top 3 earners in each department.

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

Write a query to calculate the cumulative sum of salaries for each department.

```
SELECT
    department_id,
    employee_id,
    salary,
    SUM(salary) OVER (PARTITION BY department_id ORDER BY employee_id) AS
cumulative_sum
FROM employees;
```

Write a query to calculate the rank of each employee's salary within their department.

```
SELECT
    employee_id,
    department_id,
    salary,
    RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) AS
salary_rank
FROM employees;
```

Write a query to list all employees, showing their current salary and the next month's salary.

```
SELECT
    employee_id,
    salary,
    LEAD(salary, 1) OVER (ORDER BY hire_date) AS next_month_salary
FROM employees;
```

Write a query to normalize salaries within each department.

```
SELECT
    employee_id,
    department_id,
    salary,
    (salary - AVG(salary) OVER (PARTITION BY department_id)) / STDDEV(salary) OVER
(PARTITION BY department_id) AS normalized_salary
FROM employees;
```

Write a query to calculate the percentage rank of each employee's salary within their department.

```
SELECT
    employee_id,
    department_id,
    salary,
    PERCENT_RANK() OVER (PARTITION BY department_id ORDER BY salary) AS
percent_rank
FROM employees;
```

Write a query to find the cumulative sum of sales amounts, partitioned by region.

```
SELECT
    region,
    sales_date,
    sales_amount,
    SUM(sales_amount) OVER (PARTITION BY region ORDER BY sales_date) AS
cumulative_sales
FROM sales;
```

Write a query to find the difference between each employee's salary and the average salary of their department.

```
SELECT
    employee_id,
    department_id,
    salary,
    salary - AVG(salary) OVER (PARTITION BY department_id) AS salary_difference
FROM employees;
```

Write a query to get the highest and lowest salary in each department.

```
SELECT
    employee_id,
    department_id,
    salary,
    MAX(salary) OVER (PARTITION BY department_id) AS highest_salary,
    MIN(salary) OVER (PARTITION BY department_id) AS lowest_salary
FROM employees;
```

Write a query to find the 75th percentile of salaries in each department.

```
SELECT
    employee_id,
    department_id,
    salary,
    PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY salary) OVER (PARTITION
BY department_id) AS percentile_75
FROM employees;
```

Write a query to list employees and their department's cumulative average salary.

```
SELECT
    employee_id,
    department_id,
    salary,
    AVG(salary) OVER (PARTITION BY department_id ORDER BY employee_id) AS
cumulative_avg_salary
FROM employees;
```

Write a query to find the cumulative percentage of total sales by region.

```
SELECT
    region,
    sales_date,
    sales_amount,
    SUM(sales_amount) OVER (PARTITION BY region ORDER BY sales_date) * 100.0 /
SUM(sales_amount) OVER (PARTITION BY region) AS cumulative_percentage
FROM sales;
```

MISCELLANEOUS

Write a query to find all employees who work in departments that have more than 10 employees.

```
SELECT e.* FROM employees e
JOIN (SELECT department_id, COUNT(*) AS emp_count FROM employees GROUP BY
department_id HAVING emp_count > 10) d
ON e.department_id = d.department_id;`
```

Write a query to find the employees who have the same salary as someone in another department.

```
SELECT e1.* FROM employees e1 JOIN employees e2
ON e1.salary = e2.salary AND e1.department_id <> e2.department_id;
```

Write a query using a correlated subquery to find all employees who earn more than the average salary of their department.

```
SELECT * FROM employees e1
WHERE salary > (SELECT AVG(salary) FROM employees e2 WHERE e1.department_id =
e2.department_id);
```

Write a query to find the cumulative sum of salaries in the employees table.

```
SELECT salary, SUM(salary)
OVER (ORDER BY salary) AS cumulative_sum FROM employees;
```

Write a query to find the moving average of salaries in the employees table.

```
SELECT salary, AVG(salary) OVER (ORDER BY salary ROWS BETWEEN 2 PRECEDING
AND CURRENT ROW) AS moving_average FROM employees;
```

Write a query to find the top N employees by salary in each department.

```
SELECT * FROM (SELECT e.*, ROW_NUMBER() OVER (PARTITION BY department_id
ORDER BY salary DESC) AS rank FROM employees e) WHERE rank <= N;
```

Write a query to find the year-over-year growth in employee count.

```
SELECT EXTRACT(YEAR FROM hire_date) AS year, COUNT(*) AS employee_count,
LAG(COUNT(*)) OVER (ORDER BY EXTRACT(YEAR FROM hire_date)) AS
previous_year_count, (COUNT(*) - LAG(COUNT(*)) OVER (ORDER BY EXTRACT(YEAR
```

FROM hire_date))) / LAG(COUNT(*)) OVER (ORDER BY EXTRACT(YEAR FROM hire_date)) AS growth FROM employees GROUP BY EXTRACT(YEAR FROM hire_date);

Write a query to rank employees based on their salaries within their departments.
SELECT employee_id, department_id, salary, RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) AS salary_rank FROM employees;

Write a query using a subquery to find departments with the highest average salary.

SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
HAVING AVG(salary) = (SELECT MAX(avg_salary) FROM (SELECT AVG(salary) AS avg_salary FROM employees GROUP BY department_id) sub);`

Write a query using a window function to calculate the difference between each employee's salary and the average salary of their department.

SELECT employee_id, department_id, salary,
salary - AVG(salary) OVER (PARTITION BY department_id) AS salary_difference
FROM employees;

Write a query to find the nth highest salary in the employees table.

SELECT * FROM employees e1 WHERE N-1 = (SELECT COUNT(DISTINCT salary) FROM employees e2 WHERE e2.salary > e1.salary);

Write a query to find employees who have been with the company for more than 10 years.

SELECT * FROM employees
WHERE DATEDIFF(CURRENT_DATE, hire_date) > 3650;

Write a query to find employees whose salaries are above the 90th percentile.

SELECT * FROM employees
WHERE salary > (SELECT PERCENTILE_CONT(0.90) WITHIN GROUP (ORDER BY salary) FROM employees);`