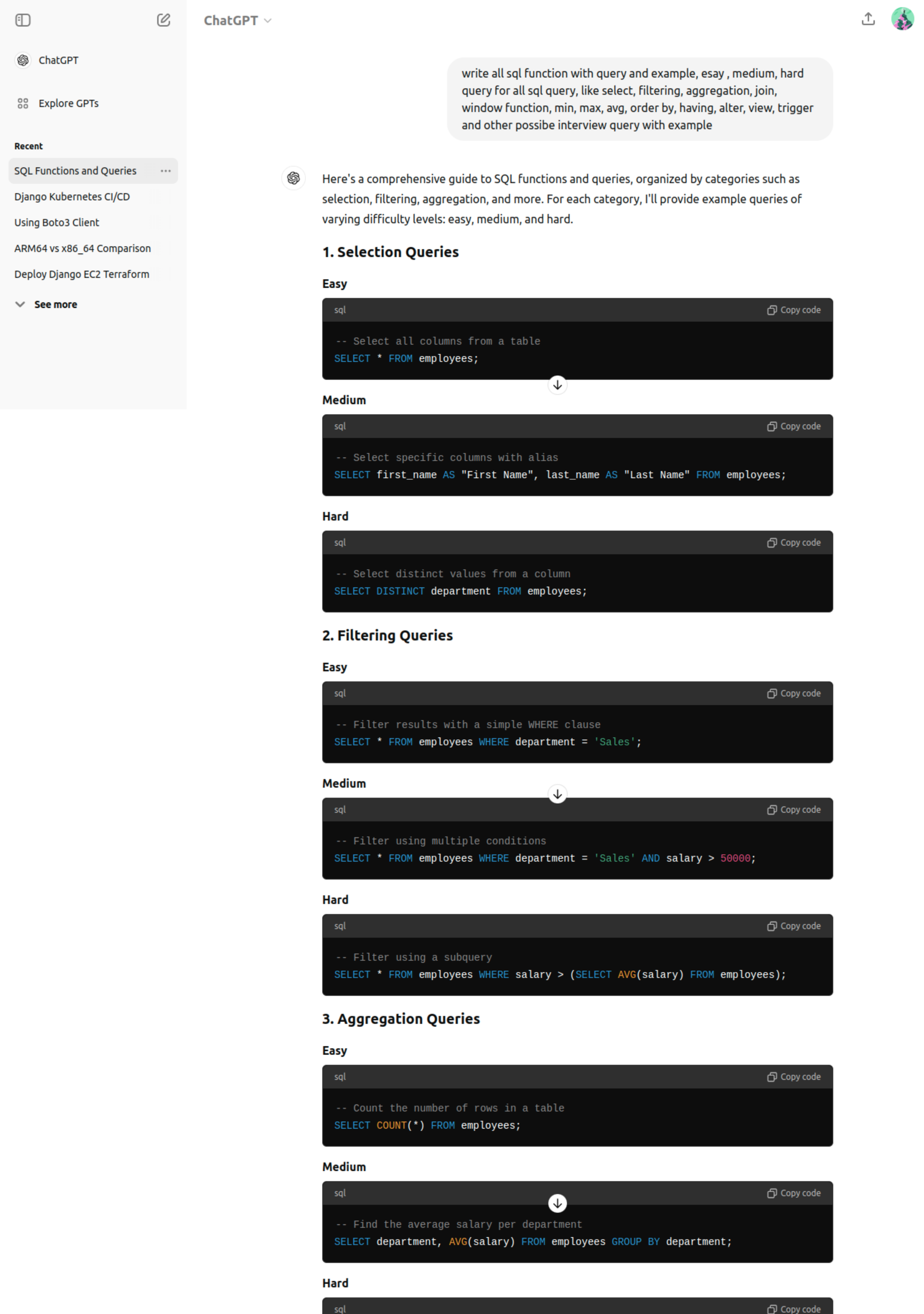

write all sql function with query and example, esay , medium, hard query for all sql query, like select, filtering, aggregation, join, window function, min, max, avg, order by, having, alter, view, trigger and other possibe interview query with example

Here's a comprehensive guide to SQL functions and queries, organized by categories such as selection, filtering, aggregation, and more. For each category, I'll provide example queries of varying difficulty levels: easy, medium, and hard.

1. Selection Queries

Easy

```sql
-- Select all columns from a table
SELECT * FROM employees;
```

Medium

```sql
-- Select specific columns with alias
SELECT first_name AS "First Name", last_name AS "Last Name" FROM employees;
```

Hard

```sql
-- Select distinct values from a column
SELECT DISTINCT department FROM employees;
```

2. Filtering Queries

Easy

```sql
-- Filter results with a simple WHERE clause
SELECT * FROM employees WHERE department = 'Sales';
```

Medium

```sql
-- Filter using multiple conditions
SELECT * FROM employees WHERE department = 'Sales' AND salary > 50000;
```

Hard

```sql
-- Filter using a subquery
SELECT * FROM employees WHERE salary > (SELECT AVG(salary) FROM employees);
```

3. Aggregation Queries

Easy

```sql
-- Count the number of rows in a table
SELECT COUNT(*) FROM employees;
```

Medium

```sql
-- Find the average salary per department
SELECT department, AVG(salary) FROM employees GROUP BY department;
```

Hard

```sql
-- Find departments with total salary greater than a certain amount
SELECT department FROM employees GROUP BY department HAVING SUM(salary) > 100000;
```

4. Join Queries

Easy

```sql
-- Inner join two tables
SELECT employees.first_name, departments.name FROM employees
JOIN departments ON employees.department_id = departments.id;
```

Medium

```sql
-- Left join with a condition
SELECT e.first_name, d.name FROM employees e
LEFT JOIN departments d ON e.department_id = d.id
WHERE d.name IS NOT NULL;
```

Hard

```sql
-- Join three tables
SELECT e.first_name, d.name, p.name FROM employees e
JOIN departments d ON e.department_id = d.id
JOIN projects p ON e.project_id = p.id;
```

5. Window Function Queries

Easy

```sql
-- Calculate the running total of salaries
SELECT first_name, salary, SUM(salary) OVER (ORDER BY first_name) AS running_total
FROM employees;
```

Medium

```sql
-- Rank employees by salary within each department
SELECT first_name, department, salary, RANK() OVER (PARTITION BY department ORDER
FROM employees;
```

Hard

```sql
-- Calculate the moving average of salaries
SELECT first_name, salary, AVG(salary) OVER (ORDER BY first_name ROWS BETWEEN 1 PRE
FROM employees;
```

6. Min, Max, Avg Functions

Easy

```sql
-- Find the minimum salary
SELECT MIN(salary) FROM employees;
```

Medium

```sql
-- Find the maximum salary in each department
SELECT department, MAX(salary) FROM employees GROUP BY department;
```

Hard

```sql
-- Find employees with a salary higher than the average salary
SELECT first_name, salary FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

7. Order By Queries

Easy

```sql
-- Order employees by salary
SELECT * FROM employees ORDER BY salary DESC;
```

Medium

```sql
-- Order by multiple columns
SELECT * FROM employees ORDER BY department, salary DESC;
```

Hard

```sql
-- Order by a calculated field
SELECT first_name, (salary + bonus) AS total_compensation
FROM employees ORDER BY total_compensation DESC;
```

8. Having Clause

Easy

```sql
-- Use HAVING with aggregation
SELECT department, COUNT(*) FROM employees GROUP BY department HAVING COUNT(*) > 5;
```

Medium

```sql
-- HAVING with multiple conditions
SELECT department, AVG(salary) FROM employees GROUP BY department HAVING AVG(salary
```

Hard

```sql
-- HAVING with a subquery
SELECT department FROM employees GROUP BY department HAVING AVG(salary) > (SELECT A
```

9. Alter Table Queries

Easy

```sql
-- Add a new column to a table
ALTER TABLE employees ADD COLUMN middle_name VARCHAR(50);
```

Medium

```sql
-- Modify the data type of a column
ALTER TABLE employees ALTER COLUMN salary TYPE DECIMAL(10, 2);
```

Hard

```sql
-- Drop a column from a table
ALTER TABLE employees DROP COLUMN middle_name;
```

10. View Queries

Easy

```sql
-- Create a simple view
CREATE VIEW sales_employees AS SELECT * FROM employees WHERE department = 'Sales';
```

Medium

```sql
-- Create a view with a join
CREATE VIEW employee_details AS
SELECT e.first_name, e.last_name, d.name AS department_name
FROM employees e
JOIN departments d ON e.department_id = d.id;
```

Hard

```sql
-- Create a view with aggregation
CREATE VIEW department_salaries AS
SELECT department, SUM(salary) AS total_salary FROM employees GROUP BY department;
```

11. Trigger Queries

Easy

```sql
-- Create a simple trigger
CREATE TRIGGER update_salary BEFORE UPDATE ON employees
FOR EACH ROW EXECUTE FUNCTION log_salary_change();
```

Medium

```sql
-- Trigger to prevent negative salary updates
CREATE OR REPLACE FUNCTION check_salary()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.salary < 0 THEN
    RAISE EXCEPTION 'Salary cannot be negative';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER salary_check BEFORE UPDATE ON employees
FOR EACH ROW EXECUTE FUNCTION check_salary();
```

Hard

```sql
-- Trigger to automatically update the last_modified column
CREATE OR REPLACE FUNCTION update_last_modified()
RETURNS TRIGGER AS $$
BEGIN
  NEW.last_modified := NOW();
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_timestamp BEFORE UPDATE ON employees
FOR EACH ROW EXECUTE FUNCTION update_last_modified();
```

12. Other Interview Queries

Easy

```sql
-- Basic subquery
SELECT first_name, salary FROM employees WHERE salary = (SELECT MAX(salary) FROM em
```

Medium

```sql
-- Correlated subquery
SELECT e.first_name, e.salary FROM employees e
WHERE e.salary > (SELECT AVG(salary) FROM employees WHERE department_id = e.departm
```

Hard

```sql
-- Complex query with CTE and window function
WITH EmployeeRank AS (
  SELECT first_name, salary, DENSE_RANK() OVER (ORDER BY salary DESC) AS rank
  FROM employees
)
SELECT first_name, salary FROM EmployeeRank WHERE rank <= 3;
```

These examples cover a wide range of SQL topics and should help in preparing for interviews by providing a solid foundation in writing and understanding SQL queries. If you have any specific areas you'd like to explore further, feel free ask!

2/2

rewrite above with explanation. exaplain each query, write schema for each query, and insert some dummy for it

Below, I'll explain each SQL query with the corresponding schema and some dummy data for practice. I'll cover each category from selection to advanced topics, ensuring you have a comprehensive understanding of each query.

Schema for Queries

Let's define a basic schema for the examples:

Employees Table

```sql
CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    department_id INT,
    salary DECIMAL(10, 2),
    bonus DECIMAL(10, 2),
    last_modified TIMESTAMP
);
```

Departments Table

```sql
CREATE TABLE departments (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50)
);
```

Projects Table

```sql
CREATE TABLE projects (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100)
);
```

Dummy Data

```sql
-- Insert data into departments
INSERT INTO departments (name) VALUES ('Sales'), ('HR'), ('Engineering');

-- Insert data into employees
INSERT INTO employees (first_name, last_name, department_id, salary, bonus, last_mo
VALUES
('John', 'Doe', 1, 60000, 5000, NOW()),
('Jane', 'Smith', 2, 75000, 7000, NOW()),
('Alice', 'Johnson', 3, 80000, 6000, NOW()),
('Bob', 'Brown', 1, 55000, 4000, NOW()),
('Charlie', 'Davis', 2, 72000, 3000, NOW());

-- Insert data into projects
INSERT INTO projects (name) VALUES ('Project Alpha'), ('Project Beta');
```

1. Selection Queries

Selection queries are used to retrieve data from a table.

Easy

```sql
-- Select all columns from a table
SELECT * FROM employees;
```

Explanation: This query retrieves all columns and rows from the `employees` table.

Medium

```sql
-- Select specific columns with alias
SELECT first_name AS "First Name", last_name AS "Last Name" FROM employees;
```

Explanation: This query selects only the `first_name` and `last_name` columns from the `employees` table and assigns them aliases for readability.

Hard

```sql
-- Select distinct values from a column
SELECT DISTINCT department_id FROM employees;
```

Explanation: This query retrieves distinct department IDs, showing unique departments without duplicates.

2. Filtering Queries

Filtering queries are used to retrieve specific data that matches given criteria.

Easy

```sql
-- Filter results with a simple WHERE clause
SELECT * FROM employees WHERE department_id = 1;
```

Explanation: This query selects all employees from the `Sales` department.

Medium

```sql
-- Filter using multiple conditions
SELECT * FROM employees WHERE department_id = 1 AND salary > 50000;
```

Explanation: This query selects employees from the `Sales` department who earn a salary greater than $50,000.

Hard

```sql
-- Filter using a subquery
```

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

Explanation: This query selects employees whose salary is above the average salary of all employees.

3. Aggregation Queries

Aggregation queries are used to perform calculations on multiple rows, returning a single value.

Easy

```sql
-- Count the number of rows in a table
SELECT COUNT(*) FROM employees;
```

Explanation: This query returns the total number of employees.

Medium

```sql
-- Find the average salary per department
SELECT department_id, AVG(salary) FROM employees GROUP BY department_id;
```

Explanation: This query calculates the average salary for each department.

Hard

```sql
-- Find departments with total salary greater than a certain amount
SELECT department_id FROM employees GROUP BY department_id HAVING SUM(salary) > 10
```

Explanation: This query identifies departments where the total salary exceeds $100,000.

4. Join Queries

Join queries are used to combine rows from two or more tables based on a related column.

Easy

```sql
-- Inner join two tables
SELECT employees.first_name, departments.name FROM employees
JOIN departments ON employees.department_id = departments.id;
```

Explanation: This query joins `` `employees` `` and `` `departments` `` tables, displaying employee names and their department names.

Medium

```sql
-- Left join with a condition
SELECT e.first_name, d.name FROM employees e
LEFT JOIN departments d ON e.department_id = d.id
WHERE d.name IS NOT NULL;
```

Explanation: This query performs a left join to include all employees, ensuring only those with a valid department are shown.

Hard

```sql
-- Join three tables
SELECT e.first_name, d.name, p.name FROM employees e
JOIN departments d ON e.department_id = d.id
JOIN projects p ON e.id = p.id;
```

Explanation: This query joins `` `employees` ``, `` `departments` ``, and `` `projects` `` tables to display employee names, department names, and project names.

5. Window Function Queries

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

Easy

sql Copy code

```sql
-- Calculate the running total of salaries
SELECT first_name, salary, SUM(salary) OVER (ORDER BY first_name) AS running_total
FROM employees;
```

Explanation: This query calculates a runnin tal of salaries ordered by first name.

Medium

sql Copy code

```sql
-- Rank employees by salary within each department
SELECT first_name, department_id, salary, RANK() OVER (PARTITION BY department_id
FROM employees;
```

Explanation: This query ranks employees by salary within each department.

Hard

sql Copy code

```sql
laries
) OVER (ORDER BY first_name ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS moving_avg
```

Explanation: This query calculates the moving average of salaries, considering the previous, current, and next rows.

6. Min, Max, Avg Functions

These functions are used to find the minimum, maximum, and average values in a dataset.

Easy

sql Copy code

```sql
-- Find the minimum salary
SELECT MIN(salary) FROM employees;
```

Explanation: This query retrieves the minimum salary from the `` `employees` `` table.

Medium

sql Copy code

```sql
-- Find the maximum salary in each department
SELECT department_id, MAX(salary) FROM employees GROUP BY department_id;
```

Explanation: This query finds the highest salary in each department.

Hard

sql Copy code

```sql
-- Find employees with a salary high than the average salary
SELECT first_name, salary FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

Explanation: This query selects employees whose salary is above the average salary.

7. Order By Queries

Order By queries are used to sort the result set based on one or more columns.

Easy

sql Copy code

```sql
-- Order employees by salary
SELECT * FROM employees ORDER BY salary DESC;
```

Explanation: This query sorts employees by salary in descending order.

Medium

```sql
-- Order by multiple columns
SELECT * FROM employees ORDER BY department_id, salary DESC;
```

Explanation: This query sorts employees first by department, then by salary in descending order.

Hard

```sql
-- Order by a calculated field
SELECT first_name, (salary + bonus) AS total_compensation
FROM employees ORDER BY total_compensation DESC;
```

Explanation: This query calculates total compensation and orders employees by this value in descending order.

8. Having Clause

The HAVING clause is used to filter groups based on a condition.

Easy

```sql
HAVING with aggregation
department_id, COUNT(*) FROM employees GROUP BY department_id HAVING COUNT(*) > 1;
```

Explanation: This query selects departments with more than one employee.

Medium

```sql
-- HAVING with multiple conditions
SELECT department_id, AVG(salary) FROM employees GROUP BY department_id HAVING AVG(
```

Explanation: This query filters departments with an average salary greater than $50,000 and more than two employees.

Hard

```sql
-- HAVING with a subquery
SELECT department_id FROM employees GROUP BY department_id HAVING AVG(salary) > (SE
```

Explanation: This query identifies departments with an average salary above the overall average salary.

9. Alter Table Queries

Alter table queries are used to modify the structure of an existing table.

Easy

```sql
-- Add a new column to a table
ALTER TABLE employees ADD COLUMN middle_name VARCHAR(50);
```

Explanation: This query adds a new column `` `middle_name` `` to the `` `employees` `` table.

Medium

```sql
-- Modify the data type of a column
ALTER TABLE employees ALTER COLUMN salary TYPE DECIMAL(12, 2);
```

Explanation: This query changes the data type of the `` `salary` `` column to a more precise decimal format.

Hard

```sql
-- Drop a column from a table
ALTER TABLE employees DROP COLUMN middle_name;
```

Explanation: This query removes the `middle_name` column from the `employees` table.

10. View Queries

View queries create a virtual table based on a SELECT query.

Easy

```sql
-- Create a simple view
CREATE VIEW sales_employees AS SELECT * FROM employees WHERE department_id = 1;
```

Explanation: This query creates a view named `sales_employees` for employees in the `Sales` department.

Medium

```sql
-- Create a view with a join
CREATE VIEW employee_details AS
SELECT e.first_name, e.last_name, d.name AS department_name
FROM employees e
JOIN departments d ON e.department_id = d.id;
```

Explanation: This query creates a view `employee_details` that joins `employees` and `departments` tables.

Hard

```sql
-- Create a view with aggregation
CREATE VIEW department_salaries AS
SELECT department_id, SUM(salary) AS total_salary FROM employees GROUP BY departmen
```

Explanation: This query creates a view `department_salaries` that shows total salary by department.

11. Trigger Queries

Triggers are procedures that automatically execute in response to certain events on a table.

Easy

```sql
-- Create a simple trigger
CREATE OR REPLACE FUNCTION log_salary_change()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO salary_changes (employee_id, old_salary, new_salary, change_date)
    VALUES (OLD.id, OLD.salary, NEW.salary, NOW());
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_salary BEFORE UPDATE ON employees
FOR EACH ROW EXECUTE FUNCTION log_salary_change();
```

Explanation: This query creates a trigger `update_salary` to log salary changes in the `salary_changes` table.

Medium

```sql
-- Trigger to prevent negative salary updates
CREATE OR REPLACE FUNCTION check_salary()
CREATE OR REPLACE FUNCTION check_salary()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.salary < 0 THEN
    RAISE EXCEPTION 'Salary cannot be negative';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER salary_check BEFORE UPDATE ON employees
FOR EACH ROW EXECUTE FUNCTION check_salary();
```

Explanation: This query creates a trigger `salary_check` to prevent updating the salary with negative values.

Hard

```sql
-- Trigger to automatically update the last_modified column
CREATE OR REPLACE FUNCTION update_last_modified()
RETURNS TRIGGER AS $$
BEGIN
  NEW.last_modified := NOW();
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_timestamp BEFORE UPDATE ON employees
FOR EACH ROW EXECUTE FUNCTION update_last_modified();
```

Explanation: This query creates a trigger `update_timestamp` that updates the `last_modified` column with the current timestamp whenever a row is updated.

12. Other Interview Queries

These queries are designed to test deeper SQL knowledge and problem-solving skills.

Easy

```sql
-- Basic subquery
SELECT first_name, salary FROM employees WHERE salary = (SELECT MAX(salary) FROM em
```

Explanation: This query selects employees with the highest salary.

Medium

```sql
-- Correlated subquery
SELECT e.first_name, e.salary FROM employees e
WHERE e.salary > (SELECT AVG(salary) FROM employees WHERE department_id = e.departm
```

Explanation: This query selects employees whose salary is above the department's average salary.

Hard

```sql
-- Complex query with CTE and window function
WITH EmployeeRank AS (
  SELECT first_name, salary, DENSE_RANK() OVER (ORDER BY salary DESC) AS rank
  FROM employees
)
SELECT first_name, salary FROM EmployeeRank WHERE rank <= 3;
```

Explanation: This query uses a Common Table Expression (CTE) to rank employees by salary and selects the top three earners.

These examples cover a wide range of SQL topics and are designed to provide a solid foundation for writing and understanding SQL queries in various scenarios.

how to find second higest salary

To find the second highest salary in a table of employees, there are several methods you can use. Each method has its advantages and can be chosen based on the SQL dialect you're using or your specific requirements.

Here are a few common approaches:

1. Using a Subquery with `LIMIT` and `OFFSET`

This method is straightforward and works in SQL databases that support `LIMIT` and `OFFSET` (like MySQL, PostgreSQL).

```sql
SELECT DISTINCT salary
FROM employees
ORDER BY salary DESC
LIMIT 1 OFFSET 1;
```

Explanation:

- This query orders the distinct salaries in descending order.
- The `LIMIT 1 OFFSET 1` clause skips the first highest salary and selects the second one.

2. Using a Subquery with `MAX()`

This method is useful in most SQL databases, including those that do not support `LIMIT` and `OFFSET`.

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

Explanation: