## SQL Questions (with Aggregate Functions) and Answers

**Assumptions:**

- You have a database named "cohort" and are using the "Students" table as defined in the "Data Definition Language (DDL) Statements" section.
- The "Students" table has been populated with data.

**Questions and Answers:**

1. Write a SQL query to retrieve the first name, last name, and email address of all students in the "Students" table.
```
SELECT
    first_name,
    last_name,
    email
FROM
    Students;
```

2. Write a SQL query to find the total number of students in the "Students" table.
```
SELECT
    COUNT(*) AS total_students
FROM
    Students;
```

3. Write a SQL query to retrieve all students whose major is 'Computer Science'.
```
SELECT
    *
FROM
    Students
WHERE
    major = 'Computer Science';
```

4. Write a SQL query to update the email address of the student with student_id = 3 to 'new_email@example.com'.
```
UPDATE
    Students
SET
    email = 'new_email@example.com'
WHERE
    student_id = 3;
```

5. Write a SQL query to delete the student with student_id = 2 from the "Students" table.
```
DELETE FROM
    Students
```

```
WHERE
  student_id = 2;
```

6. Write a SQL query to find the average age of all students in the "Students" table.

```
SELECT
  AVG(age) AS average_age
FROM
  Students;
```

7. Write a SQL query to find the maximum age among all students.

```
SELECT
  MAX(age) AS max_age
FROM
  Students;
```

8. Write a SQL query to count the number of students in each major. Display the major and the count.

```
SELECT
  major,
  COUNT(*) AS student_count
FROM
  Students
GROUP BY
  major;
```

9. Write a SQL query to find the total number of unique majors in the "Students" table.

```
SELECT
  COUNT(DISTINCT major) AS unique_majors_count
FROM
  Students;
```

10. Write a SQL query to find the youngest student in each major

```
SELECT
  major,
  MIN(age) AS youngest_age
FROM
  Students
GROUP BY
  major;
```

# SQL Joins and Subqueries

This document covers SQL joins and subqueries, explaining their purpose, types, and usage with examples.

### SQL Joins

A JOIN combines rows from two or more tables based on a related column between them. The primary types of joins are:

1. **INNER JOIN:** Returns only rows with matching values in both tables.
    - **Inner Join Example:**

```
SELECT
  e.employee_name,
  d.department_name
FROM
  employees e
INNER JOIN
  departments d ON e.department_id = d.department_id;
```

    - **Explanation:** This query returns only employees who have a matching department. Rows without a corresponding department are excluded.
    - **Tables:**
        - Employees: Contains employee data, including employee_id, employee_name, and department_id.
        - Departments: Contains department data, including department_id and department_name.
    - **ON Clause:** The join is performed based on the department_id column, which is common to both tables.
    - **Result:** Only employees with department_id values that exist in both tables are included.
    - **Example Data:**
        - Employees Table:

| employee_id | employee_name | department_id |
|---|---|---|
| 1 | Alice | 101 |
| 2 | Bob | 102 |
| 3 | Charlie | 103 |

| | | |
|---|---|---|
| 4 | David | 104 |

- Departments Table:

| department_id | department_name |
|---|---|
| 101 | HR |
| 102 | IT |
| 104 | Marketing |

- Employees INNER JOIN Departments:

| employee_id | employee_name | department_id | department_name |
|---|---|---|---|
| 1 | Alice | 101 | HR |
| 2 | Bob | 102 | IT |
| 4 | David | 104 | Marketing |

- **Explanation of the Result:** Only the employees with department_id values 101, 102, and 104 appear in the result because these values exist in both tables. Charlie, with department_id 103, is not included, as there is no matching department with ID 103 in the Departments table.

2. **LEFT JOIN (or LEFT OUTER JOIN):** Returns all rows from the left table, along with the matched rows from the right table. If there is no match, NULL is returned for the right table's columns.
   - **Left Join Example:**

```
SELECT
   e.employee_name,
   d.department_name
FROM
   employees e
LEFT JOIN
   departments d ON e.department_id = d.department_id;
```

   - **Explanation:** All employees are returned, whether they have a department or not. For employees without departments, the department column will show

NULL.

- **Tables:**
  - Employees: Contains data about employees, including their employee_id, employee_name, and department_id.
  - Departments: Contains data about departments, including department_id and department_name.
- **ON Clause:** The join is based on the department_id column, which is common to both tables.
- **Result:** All employees are included. If an employee's department_id does not match any entry in the Departments table, NULL values appear for the department_name column.
- **Example Data:**
  - Employees Table:

| employee_id | employee_name | department_id |
|---|---|---|
| 1 | Alice | 101 |
| 2 | Bob | 102 |
| 3 | Charlie | 103 |
| 4 | David | 104 |
| 5 | Eva | 105 |

  - Departments Table:

| department_id | department_name |
|---|---|
| 101 | HR |
| 102 | IT |
| 104 | Marketing |

  - Employees LEFT JOIN Departments:

| employee_id | employee_name | department_id | department_name |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 1 | Alice | 101 | HR |
| 2 | Bob | 102 | IT |
| 3 | Charlie | 103 | NULL |
| 4 | David | 104 | Marketing |
| 5 | Eva | 105 | NULL |

- **Explanation of the Result:** All employees are returned from the Employees table. Employees Charlie and Eva have department_id values (103 and 105) that do not match any rows in the Departments table. Therefore, the department_name for these employees is NULL.

3. **RIGHT JOIN (or RIGHT OUTER JOIN):** Returns all rows from the right table, along with the matched rows from the left table. If there is no match, NULL is returned for the left table's columns.
   - **Right Join Example:**

```
SELECT
   e.employee_id,
   e.employee_name,
   e.department_id,
   d.department_name
FROM
   Employees e
RIGHT JOIN
   Departments d ON e.department_id = d.department_id;
```

   - **Explanation:** All departments are returned, regardless of whether they have any employees. If a department has no employees, the employee_name column will be NULL.
   - **Tables:**
     - Employees: Contains data about employees, including their employee_id, employee_name, and department_id.
     - Departments: Contains data about departments, including department_id and department_name.
   - **ON Clause:** The join is based on the department_id column, which is common to both tables.
   - **Result:** All departments are included. If a department's department_id does

not match any entry in the Employees table, NULL values appear for columns from the Employees table.

- **Example Data:**
  - Employees Table:

| employee_id | employee_name | department_id |
|---|---|---|
| 1 | Alice | 101 |
| 2 | Bob | 102 |
| 3 | Charlie | 103 |
| 4 | David | 104 |
| 5 | Eva | 105 |

  - Departments Table:

| department_id | department_name |
|---|---|
| 101 | HR |
| 102 | IT |
| 104 | Marketing |
| 106 | Finance |

  - Employees RIGHT JOIN Departments:

| employee_id | employee_name | department_id | department_name |
|---|---|---|---|
| 1 | Alice | 101 | HR |
| 2 | Bob | 102 | IT |
| 4 | David | 104 | Marketing |

| | | | |
|---|---|---|---|
| NULL | NULL | 106 | Finance |

- **Explanation of the Result:** All departments are returned from the Departments table. The department Finance (106) does not have any matching department_id in the Employees table. Therefore, the employee_id and employee_name for this department are NULL.

4. **FULL OUTER JOIN:** Returns all rows from both tables. If there is no match in one table, NULL values are returned for the columns of the table without a match. (Note: MySQL doesn't support FULL OUTER JOIN directly.)
   - **Full Join Example:**

```
SELECT
    e.employee_id,
    e.employee_name,
    e.department_id,
    d.department_id AS dept_id,
    d.department_name
FROM
    Employees e
LEFT JOIN
    Departments d ON e.department_id = d.department_id
UNION
SELECT
    e.employee_id,
    e.employee_name,
    e.department_id,
    d.department_id AS dept_id,
    d.department_name
FROM
    Employees e
RIGHT JOIN
    Departments d ON e.department_id = d.department_id;
```

   - **Explanation:**
     - LEFT JOIN: The first part returns all rows from Employees and matching rows from Departments. Any department with no matching employee results in NULL for columns from Departments.
     - RIGHT JOIN: The second part returns all rows from Departments and matching rows from Employees. Any employee without a matching department has NULL in the department columns.

- UNION: Combines the results from both joins, ensuring that all unique rows are included, effectively mimicking a FULL OUTER JOIN.

- **Example Data:**
  - Employees Table:

| employee_id | employee_name | department_id |
|---|---|---|
| 1 | Alice | 101 |
| 2 | Bob | 102 |
| 3 | Charlie | 103 |
| 4 | David | 104 |
| 5 | Eva | 105 |

  - Departments Table:

| department_id | department_name |
|---|---|
| 101 | HR |
| 102 | IT |
| 104 | Marketing |
| 106 | Finance |

  - Employees OUTER JOIN Departments on Department ID:

| employee_id | employee_name | department_id | dept_id | department_name |
|---|---|---|---|---|
| 1 | Alice | 101 | 101 | HR |
| 2 | Bob | 102 | 102 | IT |
| 3 | Charlie | 103 | NULL | NULL |

| | | | | |
|---|---|---|---|---|
| 4 | David | 104 | 104 | Marketing |
| 5 | Eva | 105 | NULL | NULL |
| NULL | NULL | NULL | 106 | Finance |

- **Explanation of the Result:**
  - All rows from both tables are included.
  - Any row from Employees that does not have a matching department_id in Departments returns NULL in the dept_id and department_name columns (e.g., Charlie and Eva).
  - Any row from Departments that does not have a matching employee_id in Employees returns NULL in the employee_id and employee_name columns (e.g., Finance department).
  - This approach ensures a complete FULL OUTER JOIN, where all rows from both tables are included, with NULL values filling in for missing matches.

5. **CROSS JOIN:** Returns the Cartesian product of the rows from both tables. Every row from the left table is combined with every row from the right table. It does not require a common column.
   - **CrossJoin Example:**

```
SELECT
   p.product_id,
   p.product_name,
   s.supplier_id,
   s.supplier_name
FROM
   Products p
CROSS JOIN
   Suppliers s;
```

   - **Explanation:** Combines each product with every supplier, resulting in all possible pairs of Products and Suppliers. Since there are no join conditions, every row in Products matches with every row in Suppliers.
   - **Example Data:**
     - Products Table:

| **product_id** | **product_name** |
|---|---|
| 1 | Laptop |

| | |
|---|---|
| 2 | Phone |
| 3 | Tablet |

- Suppliers Table:

| supplier_id | supplier_name |
|---|---|
| 1 | Supplier A |
| 2 | Supplier B |
| 3 | Supplier C |

- Product and Suppliers Cross Join:

| product_id | product_name | supplier_id | supplier_name |
|---|---|---|---|
| 1 | Laptop | 1 | Supplier A |
| 1 | Laptop | 2 | Supplier B |
| 1 | Laptop | 3 | Supplier C |
| 2 | Phone | 1 | Supplier A |
| 2 | Phone | 2 | Supplier B |
| 2 | Phone | 3 | Supplier C |
| 3 | Tablet | 1 | Supplier A |
| 3 | Tablet | 2 | Supplier B |
| 3 | Tablet | 3 | Supplier C |

- **Explanation of the Result:** Each product is paired with each supplier, resulting in 3 x 3 = 9 rows. This combination represents every possible pairing, with no filter applied on relationships between Products and Suppliers.

**Table Creation Scripts**

```
CREATE TABLE departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(50) NOT NULL
);

CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    employee_name VARCHAR(50) NOT NULL,
    department_id INT,
    manager_id INT,
    salary INT
);

CREATE TABLE categories (
    category_id INT PRIMARY KEY,
    category_name VARCHAR(50)
);

CREATE TABLE products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(50),
    category_id INT,
    FOREIGN KEY (category_id) REFERENCES categories(category_id)
);

INSERT INTO departments (department_id, department_name) VALUES
    (1, 'Human Resources'),
    (2, 'Engineering'),
    (3, 'Finance'),
    (4, 'Sales');

INSERT INTO employees (employee_id, employee_name, department_id, manager_id,
salary) VALUES
    (1, 'Alice', 1, NULL, 100),
    (2, 'Bob', 2, 5, 200),
    (3, 'Charlie', 3, NULL, 150),
    (4, 'David', 4, 6, 300),
```

```
    (5, 'Emma', 2, NULL, 170),
    (6, 'Frank', 4, NULL, 220),
    (7, 'Grace', 2, 5, 310),
    (8, 'Hannah', 1, 1, 330),
    (9, 'Ivan', 4, 6, 350),
    (10, 'Jack', 3, 3, 50),
    (11, 'Jacky', 5, 3, 50);

INSERT INTO categories (category_id, category_name) VALUES
    (1, 'Electronics'),
    (2, 'Furniture'),
    (3, 'Clothing');

INSERT INTO products (product_id, product_name, category_id) VALUES
    (1, 'Laptop', 1),
    (2, 'Table', 2),
    (3, 'Shirt', 3),
    (4, 'Headphones', 1),
    (5, 'Sofa', NULL); -- Product without a category
```