Absolutely! Below are the **detailed, interview-ready notes** (without emojis) for the **last four SQL queries** you've asked. These are formatted in a clear and structured way for easy revision and deep understanding.

**✅ Query 1: Student Exam Attendance Report**

SELECT s.student\_id, s.student\_name, sub.subject\_name, COUNT(e.subject\_name) AS attended\_exams

FROM Students s

CROSS JOIN Subjects sub

LEFT JOIN Examinations e

ON s.student\_id = e.student\_id

AND e.subject\_name = sub.subject\_name

GROUP BY s.student\_id, s.student\_name, sub.subject\_name

ORDER BY s.student\_id, s.student\_name;

**Purpose**

Get a report of how many times each student has attended an exam for every subject, even if they didn’t take that subject.

**Concepts Covered**

* CROSS JOIN: Forms all combinations between students and subjects.
* LEFT JOIN: Ensures all combinations are preserved even when no matching exam is found.
* COUNT(column): Skips NULL values, so it's ideal to count only existing matches.
* GROUP BY: Groups by student and subject for accurate counting.
* ORDER BY: Sorts the final output for readability.

**Key Interview Concepts**

* **CROSS JOIN**: Cartesian product of two tables.
* **NULL handling** in joins and counting.
* **Join chaining and filtering logic.**

**✅ Query 2: Managers With At Least 5 Subordinates**

SELECT e1.name

FROM employee e1

JOIN employee e2

ON e1.id = e2.managerId

GROUP BY e2.managerId

HAVING COUNT(e2.managerId) >= 5;

**Purpose**

Find names of managers who supervise 5 or more employees.

**Concepts Covered**

* **Self-Join**: The employee table is joined with itself.
* GROUP BY and HAVING: Used to aggregate and filter groups by manager.
* COUNT(): Counts how many employees each manager supervises.

**Key Interview Concepts**

* **Self-Join**: Table joins to itself using foreign key relationships.
* **GROUP BY vs HAVING**: HAVING is used after aggregation, not WHERE.
* Understanding organizational hierarchy with SQL.

**✅ Query 3: Average Processing Time per Machine**

SELECT a1.machine\_id, ROUND(AVG(a2.timestamp - a1.timestamp), 3) AS processing\_time

FROM Activity a1

JOIN Activity a2

ON a1.machine\_id = a2.machine\_id

AND a1.process\_id = a2.process\_id

AND a1.timestamp < a2.timestamp

GROUP BY a1.machine\_id;

**Purpose**

Calculate average processing time for each machine by subtracting start time from end time.

**Concepts Covered**

* Table self-join with conditions.
* Timestamp subtraction.
* Aggregation with AVG() and formatting with ROUND().

**Key Interview Concepts**

* Working with **timestamps/duration**.
* **RANGE-based joins** (matching rows before/after a point).
* **Data aggregation** per machine or process.

**✅ Query 4: Employees With Bonus < 1000 or No Bonus**

SELECT e.name, b.bonus

FROM Employee e

LEFT JOIN Bonus b

ON e.empId = b.empId

WHERE b.bonus < 1000

OR b.bonus IS NULL;

**Purpose**

List all employees who received a bonus less than 1000 or no bonus at all.

**Concepts Covered**

* LEFT JOIN: Includes all employees even if no bonus entry exists.
* Handling NULL values using IS NULL.
* Conditional filtering using OR.

**Key Interview Concepts**

* **LEFT JOIN** ensures inclusion of all from the left table.
* Handling **missing data** (null bonuses).
* Importance of OR logic in WHERE clause.

Let me know if you'd like these saved as a downloadable PDF or want to continue with your next query.

Here are the **detailed and interview-ready SQL notes** for the query you just provided, structured clearly to help you revise effectively and understand every concept.

## ✅ ****Query****

SELECT s.user\_id,

ROUND(AVG(IF(c.action = 'confirmed', 1, 0)), 2) AS confirmation\_rate

FROM Signups s

LEFT JOIN Confirmations c

ON s.user\_id = c.user\_id

GROUP BY s.user\_id;

## 📌 ****Objective****

Calculate the **confirmation rate** for each user based on the number of confirmation actions they completed out of the total confirmation attempts (including those not confirmed).

## 🔍 ****Step-by-Step Explanation****

### 1. FROM Signups s

* Starts with the Signups table which contains one record per user who signed up.

### 2. LEFT JOIN Confirmations c ON s.user\_id = c.user\_id

* Joins the Confirmations table to track each confirmation action.
* LEFT JOIN is used to ensure **users who have not confirmed at all** are still included in the result with a 0 rate.

### 3. IF(c.action = 'confirmed', 1, 0)

* For each confirmation record:
  + If the action is 'confirmed', assign 1.
  + Otherwise (e.g., 'failed', 'pending'), assign 0.
* This transforms the action into a **numeric value** for aggregation.

### 🔷 🔹 ****New Concept: Conditional Aggregation with**** IF() ****Inside**** AVG()

* This is an **elegant way to calculate ratios** or rates directly in SQL.

#### **How It Works:**

* AVG(IF(...)) treats 1 as confirmed and 0 as not.
* So the average becomes:

confirmation rate=Number of confirmed actionsTotal number of confirmation records\text{confirmation rate} = \frac{\text{Number of confirmed actions}}{\text{Total number of confirmation records}}

### 4. ROUND(..., 2)

* Rounds the average to **2 decimal places**, making it cleaner to display (e.g., 0.67 instead of 0.6666666...).

### 5. GROUP BY s.user\_id

* Groups the data per user so that the confirmation rate is calculated **per individual user**.

## 📘 ****Key Concepts Covered****

| **Concept** | **Description** |
| --- | --- |
| **LEFT JOIN** | Ensures all users from Signups appear, even if they have no confirmation records. |
| **IF(condition, 1, 0)** | Converts a condition to a numeric binary output for calculation. |
| **AVG()** | Aggregates values into a mean – used here to compute proportion of successful confirmations. |
| **ROUND(x, n)** | Rounds the result to n decimal places – essential for displaying percentages or rates cleanly. |
| **GROUP BY** | Ensures rate is calculated for each user individually. |

## ✅ ****Summary (Revision Notes)****

* **Goal**: Compute how likely each user is to confirm after signup.
* **Pattern Learned**: Use AVG(IF(condition, 1, 0)) to calculate rates/percentages.
* **Important Join**: LEFT JOIN used to include users with no confirmations (producing a rate of 0).
* **Why ROUND is used**: For clean and precise output (especially for UIs or reports).
* **This Query Shows**:
  + How to work with **event-based logs** (signups and follow-ups).
  + How to use **SQL for analytics** (calculating rates/ratios).

### 🧠 Sample Output Example

| **user\_id** | **confirmation\_rate** |
| --- | --- |
| 1 | 1.00 |
| 2 | 0.50 |
| 3 | 0.00 |

Would you like this saved in PDF format as well? Or continue with the next query?