SQL Query Answers

Below is a cleaned up list of the SQL interview-style questions (based on your screenshots) along with matching SQL answers.

Query 1: Daily Net Balance Change Per Customer

Question:
Write a SQL query to display **customer_id**, **transaction_date** (date only), and **net_amount** for each customer by day, where **`net_amount = sum(deposits – withdrawals)`** on that date. Include only days when the customer had at least one transaction.

```sql

SELECT
  customer_id,
  DATE(transaction_date) AS transaction_date,
  SUM(
    CASE
      WHEN transaction_type = 'deposit' THEN amount
      WHEN transaction_type = 'withdraw' THEN -amount
      ELSE 0
    END
  ) AS net_amount
FROM transactions
GROUP BY customer_id, DATE(transaction_date)
ORDER BY customer_id, transaction_date;
```

Query 2: Customers with ≥ 3 Deposits Over ₹5,000 in the Same Month

Question:
Find customers who made at least **3 high-value deposits** (amount > 5000) within the same calendar month. Output each **customer_id**, the **year-month** (e.g. **`'2023-10'`**), and the **count** of such deposits.

```sql

SELECT
  customer_id,
  DATE_FORMAT(transaction_date, '%Y-%m') AS month_label,
  COUNT(*) AS high_value_deposit_count
FROM transactions
WHERE transaction_type = 'deposit'
  AND amount > 5000
GROUP BY customer_id, month_label
HAVING COUNT(*) >= 3
ORDER BY customer_id, month_label;
```

Query 3: Running Total of Deposits per Customer

Question:
Compute the running total of deposit amounts for each customer in chronological order of transactions. Include **customer_id**, full **transaction_date** (timestamp), **amount**, and **running_total**.

```sql
SELECT
  customer_id,
  transaction_date,
  amount,
  SUM(amount) OVER (
    PARTITION BY customer_id
    ORDER BY transaction_date
    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
  ) AS running_total
FROM transactions
WHERE transaction_type = 'deposit'
ORDER BY customer_id, transaction_date;
```

Query 4: Students Who Scored Above Class Subject Average in *Every* Subject They Appeared

Question:
From **`student_marks(student_id, student_name, subject, marks)`**, return students who scored higher than the **average mark** of each subject they took. Output **student_id** and **student_name** only if they outperformed the average in *all* their subjects.

```sql
SELECT DISTINCT s.student_id, s.student_name
FROM student_marks s
WHERE NOT EXISTS (
  SELECT 1
  FROM student_marks sub
  WHERE sub.student_id = s.student_id
    AND sub.marks <= (
      SELECT AVG(marks)
      FROM student_marks
      WHERE subject = sub.subject
    )
);
```

Query 5: Rank of Each Student per Subject

Question:
Assign a rank to each student **within each subject** based on their marks (1 = highest). Include **student_id**, **student_name**, **subject**, **marks**, and **subject_rank**.

```sql
SELECT
  student_id,
  student_name,
  subject,
  marks,
  RANK() OVER (
    PARTITION BY subject
    ORDER BY marks DESC
  ) AS subject_rank
FROM student_marks
ORDER BY subject, subject_rank;
```

(You can use DENSE_RANK() instead of RANK() if you prefer no gaps between ranks.)

Query 6: Consistent Top Performers (≥85 in Every Subject)

Question:
Identify students who scored **at least 85 marks in every subject** they've taken. Output **student_id**, **student_name**.

```sql
SELECT
  student_id,
  student_name
FROM student_marks
GROUP BY student_id, student_name
HAVING MIN(marks) >= 85;
```

Query 7: Top Student per Class-Section by Total Marks

Question:
Given a table **`students(roll_no, name, class, section, subject, marks)`**, find **the top-performing student (by total marks)** in each **`(class, section)`** pair. Output **class**, **section**, **name**, **total_marks**.

```sql
WITH totals AS (
  SELECT
    class,
    section,
    name,
    SUM(marks) AS total_marks
  FROM students
  GROUP BY class, section, name
)
SELECT
  class,
  section,
  name,
  total_marks
FROM (
  SELECT
    *,
    RANK() OVER (
      PARTITION BY class, section
      ORDER BY total_marks DESC
    ) AS rnk
  FROM totals
) t
WHERE rnk = 1
ORDER BY class, section;
```

Query 8: Average Marks per Class & Subject

Question:
Compute the **average marks** (rounded to 2 decimal places) for each **`(class, subject)`**. Output **class**, **subject**, **average_marks**, sorted ascending by **`average_marks`**.

```sql
SELECT
  class,
  subject,
  ROUND(AVG(marks), 2) AS average_marks
FROM students
GROUP BY class, subject
ORDER BY average_marks ASC;
```

✅ Summary

These queries leverage a mix of:

- **Aggregate functions** (`SUM()`, `AVG()`, `COUNT()`)
- **Conditional logic** with `CASE`
- **Window functions** (`SUM() OVER`, `RANK()`)
- **Filtering** with `HAVING` and `NOT EXISTS`

Let me know if you'd like sample data or result previews!