SQL JOB READY JOURNEY

Tanvir Hassan Ruhan

Shahjalal University of Science and Technology, Sylhet

Department of Statistics (2019-2025)

Aspiring Data Analyst, Data Science enthusiast ,Statistics Graduate

Day 1-6 Fully Basics

Day -7

Day 7 SQL Notes – Aggregations & GROUP BY Advanced

1.GROUP BY Basics Used to group rows based on column(s).

Often combined with aggregate functions: COUNT(), SUM(), AVG(), MIN(), MAX().

👉 **Syntax:**

```
SELECT

column,

aggregate_function(column)

FROM table

 GROUP BY column;
```

Example: Find the number of students in each department:

```
SELECT

department,

COUNT(*) AS total_students

FROM students

GROUP BY department;
```

2. GROUP BY Multiple Columns You can group by more than one column.

 Example: Find number of students per department and gender:

```
SELECT

department, gender,
COUNT(*) AS total_students

FROM students

GROUP BY department, gender;
```

3. HAVING Clause WHERE filters before grouping.

HAVING filters after grouping.

👉 **Example: Find departments with more than 5 students:**

```
SELECT department,

 COUNT(*) AS total_students

FROM students

 GROUP BY department

HAVING COUNT() > 5;
```

4. Conditional Aggregation (CASE WHEN) Count or sum based on conditions inside aggregate.

👉 **Example: Find number of male vs female students per course:**

```
SELECT

c.course_name,

SUM(

CASE WHEN s.gender = 'Male' THEN 1 ELSE 0 END) AS male_count,

SUM(

CASE WHEN s.gender = 'Female' THEN 1 ELSE 0 END) AS female_count

FROM students s

JOIN enrollments e ON s.student_id = e.student_id

JOIN courses c ON e.course_id = c.course_id

GROUP BY c.course_name;
```

5. DISTINCT in Aggregations To avoid duplicate counts.

👉 **Example: Count distinct students enrolled in each course:**

```
SELECT
 c.course_name,
COUNT(DISTINCT e.student_id) AS unique_students
 FROM courses c
LEFT JOIN enrollments e ON c.course_id = e.course_id
GROUP BY c.course_name;
```

6. JOIN + GROUP BY + HAVING Often interview-style queries require combining them.

👉 **Example: Find average GPA per course, but only show courses with more than 3 students:**

```
SELECT
c.course_name,
ROUND(AVG(s.gpa), 2) AS avg_gpa
FROM students s
JOIN enrollments e ON s.student_id = e.student_id
 JOIN courses c ON e.course_id = c.course_id
GROUP BY c.course_name
 HAVING COUNT(s.student_id) > 3;
```

🔑 Quick Tips Use WHERE → before grouping

Use HAVING → after grouping

GROUP BY always comes after WHERE but before ORDER BY

When mixing JOINs, always GROUP BY the grouping column from the main output

Use CASE WHEN for category-wise counts inside the same query

🎯 Day 7 Preparation Checklist ✅ Understand GROUP BY (single & multiple columns) ✅ Master HAVING vs WHERE ✅ Write queries with CASE WHEN inside aggregations ✅ Try DISTINCT with aggregates ✅ Practice JOIN + GROUP BY + HAVING

Day 8

Day 8 SQL Notes – Subqueries & Advanced Filtering

1. What is a Subquery? A query inside another query.

Used in SELECT, FROM, or WHERE.Always written inside ().

👉 Example (in WHERE):

1.Finds students whose GPA is higher than the average GPA.

SELECT name, gpa

FROM students

WHERE gpa > (SELECT AVG(gpa) FROM students);

2. Types of Subqueries (A) Simple Subquery Does not depend on outer query.Runs once, result used by outer query.

👉 **Example:**

SELECT name

FROM students

WHERE department = (SELECT department FROM students WHERE name = 'Alice');

(B) Correlated Subquery Inner query depends on outer query.

👉 Example: Finds students whose GPA is higher than avg GPA of their own department.

SELECT

s1.name,

s1.department,

s1.gpa

FROM students s1

WHERE s1.gpa > (SELECT AVG(s2.gpa)

FROM students s2

WHERE s1.department = s2.department);

3. Subquery in SELECT You can use a subquery to calculate extra values.

👉 **Example:**

SELECT name, gpa,

(SELECT AVG(gpa) FROM students) AS overall_avg

FROM students;

4. Subquery in FROM (Derived Table) Treats subquery as a temporary table.

👉 **Example:**

SELECT department, avg_gpa

FROM (SELECT department, AVG(gpa) AS avg_gpa

FROM students

GROUP BY department) dept_avg WHERE avg_gpa > 3.5;

5. Subquery with IN / NOT IN 👉 Example:

Find students enrolled in Computer Science courses.

SELECT name FROM students

WHERE student_id IN

(SELECT student_id FROM enrollments e

JOIN courses c ON e.course_id = c.course_id

WHERE c.department = 'Computer Science');

6. Subquery with EXISTS / NOT EXISTS

Example: Find students who are enrolled in at least one course.

SELECT s.name

FROM students s

WHERE EXISTS

(SELECT 1 FROM enrollments e WHERE e.student_id = s.student_id);

Example (NOT EXISTS):

Find students who are not enrolled in any course.

SELECT s.name

FROM students s

WHERE NOT EXISTS (SELECT 1 FROM enrollments e WHERE e.student_id = s.student_id);

7. Subquery with ANY / ALL

ANY → greater than at least one value.

ALL → greater than all values.

Example: Find students whose GPA is higher than all students in Physics dept:

SELECT name, gpa

FROM students

WHERE gpa > ALL (SELECT gpa FROM students WHERE department = 'Physics');

8. Combining Conditions

Example: Find students not in CS courses and GPA > 3.5:

```
 SELECT name, gpa
FROM students WHERE gpa > 3.5 AND
student_id NOT IN ( SELECT student_id FROM enrollments e
 JOIN courses c ON e.course_id = c.course_id
WHERE c.department = 'Computer Science' );
```

📝 Day 8 Summary Subquery types:

In SELECT, FROM, WHERE

Simple vs Correlated

Operators with subqueries:

IN, NOT IN, EXISTS, NOT EXISTS

ANY, ALL

Use Cases:

Filter students by GPA compared to avg

Check enrollments (EXISTS/NOT EXISTS)

Compare values within groups (correlated subquery)

✅ Day 8 Checklist Understand subquery placement (SELECT, FROM, WHERE)

Practice simple + correlated subqueries

Use IN/NOT IN, EXISTS/NOT EXISTS

Try ANY/ALL examples

Solve at least 5–6 practice problems

Day 9 SQL Notes

Window Functions & Ranking

1. What are Window Functions?

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

Unlike GROUP BY, they do not collapse rows — every row is preserved.

Useful for ranking, running totals, moving averages, and percentiles.

Syntax Template:

s <window_function>() OVER ([PARTITION BY column1, column2, ...] [ORDER BY column3 [ASC|DESC]])

2. ROW_NUMBER() Assigns a unique sequential number to each row within a partition.

Ties do not share the same number.

Example: Rank students by GPA in each department:

SELECTname, department, gpa,

ROW_NUMBER() OVER (PARTITION BY department ORDER BY gpa DESC) AS rn

FROM students;

Output:

name department gpa rn Alice CS 4.0 1 Bob CS 3.8 2 Carol Math 3.9 1

3. RANK() vs DENSE_RANK() Both assign ranks based on ordering, but handle ties differently.

Function Behavior RANK() Ties get the same rank, but next rank is skipped DENSE_RANK() Ties get the same rank, next rank is not skipped

Example: Rank students by GPA in CS department:

SELECT name, department, gpa,

RANK() OVER (PARTITION BY department ORDER BY gpa DESC) AS rank,

DENSE_RANK() OVER (PARTITION BY department ORDER BY gpa DESC) AS dense_rank

FROM students

WHERE department = 'CS';

If two students tie for 1st, RANK() next rank = 3, DENSE_RANK() next rank = 2

4. NTILE(n) Divides rows into n roughly equal groups.

Useful for quartiles, percentiles, performance groups

Example: Divide students into 4 performance quartiles:

SELECT

name, department, gpa,

NTILE(4) OVER (ORDER BY gpa DESC) AS quartile FROM students;

NTILE(4) → 1 = top 25%, 4 = bottom 25%

5. PARTITION BY Groups rows before applying window function

Each partition resets the numbering/ranking

Example: Top student per department:

SELECT name, department, gpa,

ROW_NUMBER() OVER (PARTITION BY department ORDER BY gpa DESC) AS rn

FROM students WHERE rn = 1;

6. ORDER BY in Window Functions Determines the ranking or sequential order.

Use ASC or DESC depending on requirement

Example: Bottom 3 GPA students per department

SELECT name, department, gpa,

ROW_NUMBER() OVER (PARTITION BY department ORDER BY gpa ASC) AS rn

FROM students WHERE rn <= 3;

7. Common Use Cases Top-N per group → Top 3 students per department

Percentiles & quartiles → NTILE()

Cumulative totals / running sums → SUM() OVER(...)

Rankings with ties → RANK() vs DENSE_RANK()

Identify previous/next rows → LEAD(), LAG()

8. Job-Ready Tips Always use aliases for readability

PARTITION BY → for grouping

ORDER BY → for ranking or sequencing

Combine window functions with WHERE, ORDER BY, or JOIN for interview-level queries

Very common in HackerRank Medium-Hard & DataLemur problems

🔷 9. Cheat Sheet – Window Functions Function Purpose Notes

ROW_NUMBER() Sequential numbering Ties get different numbers

RANK() Rank with gaps Ties share rank, next rank skipped

DENSE_RANK() Rank without gaps Ties share rank, next rank not skipped

NTILE(n) Divides rows into n groups Useful for quartiles/percentiles

SUM() OVER() Running total Can use PARTITION BY + ORDER BY

LEAD(), LAG() Access next/previous row Useful for comparisons or differences

✅ Day 9 Preparation Checklist Understand ROW_NUMBER(), RANK(), DENSE_RANK(), NTILE()

Practice PARTITION BY + ORDER BY combinations

Solve Top-N per group queries

Solve quartile/percentile queries using NTILE()

Apply window functions with joins or filtering

Day 10 SQL Notes

Advanced Window Functions (LEAD, LAG, Running Totals, Moving Averages)

1. What are LEAD() and LAG()?

LEAD() → Accesses the next row relative to the current row

LAG() → Accesses the previous row relative to the current row

Both are window functions; require OVER() clause.

Syntax:

LEAD(column, offset, default) OVER (PARTITION BY column1 ORDER BY column2)

LAG(column, offset, default) OVER (PARTITION BY column1 ORDER BY column2)

offset → how many rows forward/backward (default = 1)

default → value if no row exists (optional)

Example – Compare each student's GPA with previous student in the same department:

```
SELECT name, department, gpa,
 LAG(gpa, 1) OVER (PARTITION BY department ORDER BY gpa DESC) AS prev_gpa,
 LEAD(gpa, 1) OVER (PARTITION BY department ORDER BY gpa DESC) AS next_gpa
FROM students;
```

2. Cumulative Aggregates (Running Totals) Use window functions with SUM() or COUNT() to get running totals

Rows are not collapsed, so every row shows cumulative result

Example – Running total of GPA per department:

```
SELECT name, department, gpa,
SUM(gpa) OVER (PARTITION BY department ORDER BY gpa DESC) AS cumulative_gpa
FROM students;
```

Can also do running count:

```
SELECT name, department,
COUNT(*) OVER (PARTITION BY department ORDER BY gpa DESC) AS cumulative_count
FROM students;
```

3. Moving Averages Calculate average over a sliding window of rows

Use ROWS BETWEEN in window function

Example – 3-student moving average GPA per department:

SELECT name, department, gpa,

AVG(gpa) OVER (PARTITION BY department ORDER BY gpa DESC ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS moving_avg

FROM students;

2 PRECEDING AND CURRENT ROW → considers current + 2 previous rows

4. Combining Ranking + Analytics Combine ROW_NUMBER(), RANK(), NTILE() with cumulative aggregates or LEAD/LAG

Example – Top 3 students per department with cumulative GPA:

SELECT name, department, gpa,

ROW_NUMBER() OVER (PARTITION BY department ORDER BY gpa DESC) AS rn,

SUM(gpa) OVER (PARTITION BY department ORDER BY gpa DESC) AS cumulative_gpa

FROM students WHERE rn <= 3;

5. PARTITION BY & ORDER BY Nuances

PARTITION BY → defines groups (like GROUP BY, but keeps rows separate)

ORDER BY → defines sequence within each partition

ROWS BETWEEN / RANGE → define window frame for cumulative/moving aggregates

🔷 6. Practical Use Cases Function Use Case

LEAD() Compare next period sales / next student score

LAG() Compare previous month revenue / previous GPA

SUM() OVER() Running total of sales, GPA, revenue

AVG() OVER() Moving average of sales or GPA

ROW_NUMBER() Top-N per group

NTILE() Quartiles / percentile analysis

🔷 7. Job-Ready Tips Always alias your columns for readability

Window functions are often combined with filters and joins

Running totals / moving averages are common in analytics and business SQL tasks

Use LEAD/LAG to detect trends or differences

HackerRank/DataLemur loves these patterns for medium-hard questions

🔷 8. Cheat Sheet – Day 10 Function Purpose Notes LEAD(col, n, default) Next row value Optional offset & default LAG(col, n, default) Previous row value Optional offset & default SUM(col) OVER() Running total Use PARTITION + ORDER BY AVG(col) OVER() Moving average Use ROWS BETWEEN for sliding window ROW_NUMBER() Sequential numbering Unique per partition RANK() / DENSE_RANK() Ranking Handle ties NTILE(n) Divide into n groups Quartiles / Percentiles

🔷 9. Day 10 Checklist Understand LEAD() / LAG()

Practice running totals with SUM()

Practice moving averages with AVG()

Combine ranking + cumulative analytics

Solve Top-N per group with LEAD/LAG problems

Day 11 Notes

Complex Joins & Multi-table Queries

1. Self Join 👉 Self-Join মানে একই টেবিলকে দুইবার ব্যবহার করা।

Example: একই department এ থাকা student pair বের করো।

```
SELECT a.name AS student1, b.name AS student2, a.department
FROM students a
 JOIN students b ON a.department = b.department AND a.student_id < b.student_id;
```

2. Multi-table Join 👉 এক query তে 3 বা তার বেশি টেবিল join করা।

Example: student + enrollments + courses

```
 SELECT s.name, c.course_name
FROM students s
 JOIN enrollments e ON s.student_id = e.student_id
JOIN courses c ON e.course_id = c.course_id;
```

3. Different Joins INNER JOIN 👉 দুপাশে match থাকলে row দেখাবে।

```
SELECT s.name, c.course_name
FROM students s I
NNER JOIN enrollments e ON s.student_id = e.student_id
INNER JOIN courses c ON e.course_id = c.course_id;
```

4.LEFT JOIN 👉 Left table এর সব row + যদি match থাকে Right এর।

```
SELECT s.name, c.course_name
 FROM students s
LEFT JOIN enrollments e ON s.student_id = e.student_id
LEFT JOIN courses c ON e.course_id = c.course_id;
```

5.RIGHT JOIN 👉 Right table এর সব row + যদি match থাকে Left এর।

```
SELECT s.name, c.course_name
 FROM students s
RIGHT JOIN enrollments e ON s.student_id = e.student_id;
```

FULL OUTER JOIN (MySQL-এ নেই, PostgreSQL এ আছে) 👉 দুপাশের সব row, match থাকুক বা না থাকুক।

6. Cross Join 👉 দুই টেবিলের সব row এর cartesian product।

Example: সব department × সব course combination

```
SELECT d.department_name, c.course_name
FROM departments d  CROSS JOIN courses c;
```

7. Joins + Aggregation 👉 Join এর পরে GROUP BY।

Example: প্রতি course এ কতজন student enrolled

```
SELECT c.course_name,
COUNT(e.student_id) AS total_students
 FROM courses c
LEFT JOIN enrollments e ON c.course_id = e.course_id
GROUP BY c.course_name;
```

8. Joins + Filtering 👉 WHERE / HAVING এর সাথে join করা।

Example: এমন course দেখাও যেখানে 2 জনের বেশি student আছে।

```
SELECT c.course_name,
COUNT(e.student_id) AS total_students
FROM courses c
JOIN enrollments e ON c.course_id = e.course_id
GROUP BY c.course_name
HAVING COUNT(e.student_id) > 2;
```

9. Aliasing 👉 Alias দিয়ে query readable করা।

```
SELECT s.name, c.course_name
FROM students AS s
JOIN enrollments AS e ON s.student_id = e.student_id
 JOIN courses AS c ON e.course_id = c.course_id;
```

10.Real-world Use Cases (Job Ready) Find employees who report to the same manager → Self Join

Show all customers and their orders → INNER/LEFT Join

Find products never ordered → LEFT Join + WHERE NULL

Get top-selling product categories → Joins + GROUP BY

Generate all possible staff × shift combinations → CROSS Join

10. Quick Cheat Sheet Join Type What it Does

INNER JOIN Only matched rows

LEFT JOIN All from Left + matched Right

RIGHT JOIN All from Right + matched Left

FULL JOIN All rows from both sides

SELF JOIN Table joins with itself

CROSS JOIN Cartesian product JOIN + GROUP BY Aggregation after join

Day 12 Notes

Advanced Subqueries & CTEs

1. Advanced Subqueries Subquery = একটি query যা অন্য query এর ভিতরে লেখা থাকে।

Types of Subqueries:

a) Scalar Subquery শুধুমাত্র একটি value return করে।সাধারণত SELECT এর সাথে ব্যবহার হয়।

Example:

SELECT name,

(SELECT AVG(gpa) FROM students) AS avg_gpa

FROM students;

→ প্রতিটি student এর সাথে overall average GPA দেখাবে।

b) Correlated Subquery Outer query এর row অনুযায়ী execute হয়। Inner query outer query এর value ব্যবহার করে।

Example:

SELECT s.name, s.gpa

FROM students s

WHERE s.gpa > (SELECT AVG(gpa) FROM students

WHERE department = s.department);

→ শুধুমাত্র সেই student দেখাবে যাদের GPA department average GPA এর চেয়ে বেশি।

c) EXISTS / NOT EXISTS Boolean result দেয়।

Check করে inner query result আছে কি না।

Example:

SELECT name

FROM students s WHERE EXISTS

(SELECT 1 FROM enrollments e WHERE e.student_id = s.student_id);

→ শুধু সেই students দেখাবে যারা কোনো course এ enrolled।

d) IN / NOT IN with Subquery একটি set এর মধ্যে value check করে।

Example:

```
SELECT name FROM students
 WHERE department IN
 ( SELECT department FROM students GROUP BY department HAVING COUNT(*) > 2 );
```

→ সেই department এর students দেখাবে যেখানে ২ এর বেশি students আছে।

2. Common Table Expressions (CTEs) CTE = Temporary result set যা query এর execution এ use হয়।

Basic Syntax:

```
WITH cte_name AS
 ( SELECT ... )
 SELECT * FROM cte_name;
```

Example:

```
WITH avg_gpa_cte AS
( SELECT department, AVG(gpa) AS avg_gpa
 FROM students
GROUP BY department )
SELECT s.name, s.department, s.gpa, c.avg_gpa
 FROM students s
 JOIN avg_gpa_cte c ON s.department = c.department;
```

→ প্রতিটি student এর department average GPA দেখাবে।

3. Recursive CTE নিজের উপর re-run হয় যতক্ষণ condition satisfied থাকে।

Hierarchical / tree data query করতে কাজে লাগে।

Example (PostgreSQL syntax):

WITH RECURSIVE subordinates AS

(SELECT employee_id, manager_id, name

FROM employees

WHERE manager_id IS NULL

UNION ALL

SELECT e.employee_id, e.manager_id, e.name FROM employees e

INNER JOIN subordinates s ON e.manager_id = s.employee_id)

*SELECT * FROM subordinates;*

→ organization hierarchy বের করবে।

🔷 4. Advantages of CTEs Complex query সহজ করে তোলে।

Reusable logical blocks তৈরি করে।

Readability ও maintenance improve করে।

🔷 5. Practical Use Cases (Job Ready) Function Use Case Scalar Subquery Calculate average / min / max values inline Correlated Subquery Conditional filtering based on group data EXISTS / NOT EXISTS Check existence before returning results IN / NOT IN Filter based on dynamic sets CTE Break large queries into manageable steps Recursive CTE Hierarchical data processing

🔷 6. Quick Cheat Sheet – Day 12 Concept Syntax Example Scalar Subquery (SELECT AVG(gpa) FROM students) Correlated Subquery WHERE gpa > (SELECT AVG(gpa) FROM students WHERE department = s.department) EXISTS WHERE EXISTS (SELECT 1 FROM enrollments e WHERE e.student_id = s.student_id) IN WHERE department IN (SELECT department FROM students GROUP BY department HAVING COUNT(*) > 2) CTE WITH cte_name AS (...) SELECT * FROM cte_name; Recursive CTE WITH RECURSIVE ... SELECT * FROM cte_name;

💡 Pro Tip:

Subqueries ভালোভাবে ব্যবহার করতে পারলে HackerRank / DataLemur এর intermediate questions সহজে crack করা যায়।

CTE এর মাধ্যমে complex join + aggregation কে modular query তে ভাগ করে পড়া যায়।

Recursive CTE PostgreSQL এর সবচেয়ে শক্তিশালী tool — job interviews এ খুব valuable।

Day 13

Advanced Joins & Set Operations

1. **Self Join Self** Join মানে হলো একই টেবিলকে নিজের সাথেই join করা।
2. 👉 যখন row-to-row comparison করতে হবে বা hierarchy বের করতে হবে তখন কাজে লাগে।

Syntax:

SELECT

a.col, b.col

FROM table a

JOIN table b ON a.something = b.something;

Example: Students from the same department

SELECT

s1.name AS student1,

s2.name AS student2,

s1.department FROM students s1

JOIN students s2 ON s1.department = s2.department

AND s1.student_id < s2.student_id;

এখানে < condition দিলে duplicate pair avoid হবে।

Use Cases:

একই department এর student pair খুঁজে বের করা।

Employee–Manager relation বের করা।

Row comparison (যেমন salary gap between employees)।

🔷 **2. Multi-Table Joins (3+ Tables)** 👉 যখন multiple table এর data combine করতে হবে।

Example: Students with their courses

SELECT s.name, c.course_name, c.credits

FROM students s

JOIN enrollments e ON s.student_id = e.student_id

JOIN courses c ON e.course_id = c.course_id;

Use Cases:

1.Reporting queries (student + enrollment + course)।

2.Analytics queries (customer + orders + products)।

3.Denormalized dataset তৈরির জন্য।

🔷 **3. FULL OUTER JOIN (PostgreSQL only)** 👉 Combines LEFT + RIGHT join result 👉 MySQL doesn't support directly, workaround: UNION of left + right join.

PostgreSQL Example:

SELECT

s.name, c.course_name

FROM students s

FULL OUTER JOIN enrollments e ON s.student_id = e.student_id

FULL OUTER JOIN courses c ON e.course_id = c.course_id;

→ সব students আর সব courses আসবে, কেউ enrolled না হলেও।

MySQL Workaround:

SELECT s.name, c.course_name

FROM students s

LEFT JOIN enrollments e ON s.student_id = e.student_id

LEFT JOIN courses c ON e.course_id = c.course_id

UNION

SELECT s.name, c.course_name

FROM students s

RIGHT JOIN enrollments e ON s.student_id = e.student_id

RIGHT JOIN courses c ON e.course_id = c.course_id;

🔷 **4. CROSS JOIN** 👉 Cartesian Product (প্রতিটি row × প্রতিটি row)। 👉 Use Case: All combinations generate করা।

Example:

SELECT s.department, c.course_name

FROM students s

CROSS JOIN courses c;

→ প্রতিটি department × course combination আসবে।

Use Cases:

1.Test data generate

2.Possible pair তৈরি

3.Market basket analysis type কাজ

5. Set Operations যখন দুইটা SELECT এর result একসাথে compare/merge করতে চাই তখন ব্যবহার হয়।

a) UNION Duplicate remove করে।

SELECT name

FROM students

WHERE department = 'Mathematics'

UNION

SELECT name

FROM students WHERE

department = 'Computer Science';

b) UNION ALL Duplicate রাখে।

SELECT department

FROM students

UNION ALL

SELECT department

FROM students;

c) INTERSECT (PostgreSQL only) দুই query এর common rows।

SELECT name

FROM students

WHERE department = 'Mathematics'

INTERSECT

SELECT name

FROM students WHERE gpa > 3.5;

→ Math + GPA > 3.5 students।

d) EXCEPT (PostgreSQL only) First query থেকে second query বাদ দেবে।

SELECT name

FROM students

WHERE department = 'Mathematics'

EXCEPT

SELECT name

FROM students

WHERE gpa < 3.5;

→ শুধুমাত্র সেই Math students আসবে যাদের GPA ≥ 3.5

🔷 **6. Practical Job-ready Use Cases Topic Real Job Use Case**

Self Join Employee–Manager mapping, peer comparison Multi-Join Analytics reports (e.g., sales + customer + products)

FULL OUTER JOIN Complete list (even missing links)

CROSS JOIN Generate test scenarios, combinations

UNION Merge results from different conditions

INTERSECT Common users between two campaigns

EXCEPT Users in group A but not in group B

🔷 7. Quick Cheat Sheet – Day 13 Concept

Syntax

Self Join FROM table t1 JOIN table t2 ON t1.col = t2.col

Multi Join JOIN table3 ... FULL OUTER JOIN (Postgres) FROM A FULL OUTER JOIN B ON ...

FULL OUTER JOIN (MySQL) LEFT JOIN ... UNION RIGHT JOIN ... CROSS JOIN FROM A CROSS JOIN B UNION SELECT ... UNION SELECT ... INTERSECT (PG) SELECT ... INTERSECT SELECT ... EXCEPT (PG) SELECT ... EXCEPT SELECT ...

Day 13 Goal:

Self join fluently লিখতে পারবে।

Multi-table joins handle করতে পারবে।

PostgreSQL set operations confidently করতে পারবে।

HackerRank / DataLemur এ আসা advanced join প্রশ্ন গুলো solve করার confidence আসবে।

Day 14

Indexes, Query Optimization & Execution Plans (Very Very Important for Viva)

1. **Indexes Index কি?**

Index হলো database এর একটা special structure যা search speed বাড়ায়। এটা ঠিক বইয়ের “index” এর মতো — যেখানে page number দিয়ে fast access possible হয়।

2.কেন লাগে?

a)Large dataset এ query speed dramatically বাড়ায়

b)Searching, filtering, joining দ্রুত হয়

c)Sorting এবং grouping improve করে

Types of Indexes

Type Description

i)Primary Index Primary key উপর automatically তৈরি হয়

Unique Index Unique values enforce করে

Composite Index Multiple columns উপর index তৈরি করে

Full-text Index Text search জন্য

Index creation syntax

CREATE INDEX idx_column_name ON table_name(column_name);

Example:

CREATE INDEX idx_department ON students(department);

Index drop syntax

DROP INDEX idx_column_name;

Example – Before & After Index Without Index:

SELECT * FROM students WHERE department = 'Mathematics';

→ Full table scan, slower

With Index:

CREATE INDEX idx_department ON students(department);

*SELECT * FROM students WHERE department = 'Mathematics';*

→ Index scan, much faster

2. **Query Optimization Basics**

Why optimization matters Large dataset এ slow queries resource drain করে

Poor performance affects user experience

Best practices

i)Avoid SELECT * → শুধু প্রয়োজনীয় column নাও

ii)Use WHERE to filter early

iii)Avoid unnecessary joins

iv)Use LIMIT for testing

v)Avoid functions in WHERE clause (makes indexes useless)

Example:

SELECT name

FROM students

WHERE LOWER(department) = 'mathematics';

→ এখানে LOWER makes index useless। Better:

SELECT name

FROM students

WHERE department = 'Mathematics';

3. **Execution Plan Execution Plan কী?**

Database query execution এর roadmap, showing how query processed হয়।

Why it matters বুঝতে সাহায্য করে query slow কেন

Identify করে bottlenecks

How to check

*EXPLAIN SELECT * FROM students WHERE department = 'Mathematics';*

Key Terms Term Meaning Seq Scan Sequential scan of table (slow for large tables) Index Scan Index use করে faster scan Nested Loop Join এর একটি strategy Hash Join Large dataset join এর জন্য efficient

4. Performance Tips Index columns frequently used in WHERE, JOIN, ORDER BY

Avoid SELECT *

Filter early (WHERE clause before JOIN if possible)

Check EXPLAIN plan

Partition large tables if needed (PostgreSQL)

5. Real-World Examples Optimizing SELECT:

CREATE INDEX idx_gpa ON students(gpa);

SELECT name FROM students WHERE gpa > 3.5;

Optimizing JOIN:

CREATE INDEX idx_student_id ON enrollments(student_id);

CREATE INDEX idx_course_id ON enrollments(course_id);

SELECT s.name, c.course_name

FROM students s

JOIN enrollments e ON s.student_id = e.student_id

JOIN courses c ON e.course_id = c.course_id;

6. Quick Cheat Sheet – Day 14 Concept Syntax Create Index CREATE INDEX idx_name ON table(column); Drop Index DROP INDEX idx_name; Explain Plan EXPLAIN SELECT ... Filter early Use WHERE before JOIN Avoid SELECT * Use specific columns

Day 15

Views, Materialized Views & Query Refactoring

1.Views Definition: View = virtual table (query stored as a table)

Purpose: Simplify complex queries, reuse code, add security layer.

Syntax:

CREATE VIEW view_name AS SELECT column1, column2 FROM table WHERE condition;

Drop view:

DROP VIEW view_name;

Example:

CREATE VIEW cs_students AS

SELECT name, department

FROM students

WHERE department = 'Computer Science';

2. Materialized Views (PostgreSQL specific) Stored view with actual saved data .Needs refresh when underlying data changes

Syntax:

CREATE MATERIALIZED VIEW view_name AS

SELECT column1, column2

FROM table WHERE condition;

REFRESH MATERIALIZED VIEW view_name;

Example:

CREATE MATERIALIZED VIEW avg_gpa_per_department AS

SELECT department, AVG(gpa) AS avg_gpa

FROM students

GROUP BY department;

3. Query Refactoring Definition: Query কে efficient ও clean করে পুনরায় লেখা।

Purpose: Improve performance & maintainability

Methods:

i)Break large queries into smaller CTEs

ii)Avoid redundant joins

iii)Use indexes

Replace subqueries with joins when possible

Job-Ready Skills – Day 15 Create and manage views

Use materialized views effectively

Refactor queries for readability and performance

Know when to use views vs materialized views

Day 16 Notes

Advanced Aggregations & Analytical Thinking

Goal: Understand complex aggregations, conditional logic, and analytical summaries that help you think like a data analyst / SQL developer.

Conditional Aggregation (CASE + Aggregate Functions) Concept:

We can use CASE WHEN inside SUM, COUNT, or AVG to apply conditional logic in aggregations.

Example: Count how many students have GPA above or below 3.5 in each department:

```
SELECT department_id,

SUM(CASE WHEN gpa >= 3.5 THEN 1 ELSE 0 END) AS high_achievers,

SUM(CASE WHEN gpa < 3.5 THEN 1 ELSE 0 END) AS low_achievers

FROM students

 GROUP BY department_id;
```

How it works:

i)CASE acts like an IF statement inside aggregate functions.

ii)Real-world Use: Counting active vs inactive users

iii)Sales above/below target

iv)Categorizing customers (e.g. loyal vs new)

Multiple Aggregations in the Same Query Combine several aggregate functions like COUNT, SUM, AVG, MAX, MIN.

Example:

```
SELECT department_id,
COUNT(*) AS total_students,
AVG(gpa) AS avg_gpa,
 MAX(gpa) AS highest_gpa,
MIN(gpa) AS lowest_gpa
FROM students
 GROUP BY department_id;
```

Pro Tip: Combine metrics in one query instead of running many separate ones — it saves time and improves performance.

HAVING Clause with Aggregations WHERE filters individual rows. HAVING filters aggregated results after GROUP BY.

Example: Show only departments with average GPA greater than 3.5:

```
SELECT department_id,
AVG(gpa) AS avg_gpa
FROM students
GROUP BY department_id
HAVING AVG(gpa) > 3.5;
```

Remember:

WHERE → filters before grouping

HAVING → filters after grouping

Advanced Grouping:

GROUPING SETS / ROLLUP / CUBE (PostgreSQL or BigQuery) These help to get multi-level summaries in one query.

Example (PostgreSQL):

SELECT department_id, course_id,

SUM(credits)

FROM courses

GROUP BY ROLLUP(department_id, course_id);

Output includes: Sum per department,Sum per course,Grand total

Real Use Case: Reports where you need both department totals + overall total in one query.

Analytical Problem Solving

Example 1: Highest Average GPA by Department

SELECT department_id,

AVG(gpa) AS avg_gpa

FROM students

GROUP BY department_id

ORDER BY avg_gpa DESC LIMIT 1;

Example 2: Percentage of Students per Department

SELECT department_id,

*ROUND((COUNT() * 100.0 / (SELECT COUNT() FROM students)), 2) AS percentage*

FROM students

GROUP BY department_id;

Example 3:

Department-wise GPA Category Breakdown

SELECT department_id,

SUM(CASE WHEN gpa >= 3.5 THEN 1 ELSE 0 END) AS high_gpa,

SUM(CASE WHEN gpa BETWEEN 3.0 AND 3.49 THEN 1 ELSE 0 END) AS mid_gpa,

SUM(CASE WHEN gpa < 3.0 THEN 1 ELSE 0 END) AS low_gpa

FROM students

GROUP BY department_id;

Day 17

Ranking & Advanced Window Functions

Goal of the Day Master how to: Rank, compare, and analyze records within groups .Use window functions (RANK(), DENSE_RANK(), ROW_NUMBER(), NTILE()).Build analytical queries like "top performers", "second highest", and “percentile” calculations

Introduction to Window Functions: Window functions operate over a set of rows (window) , but don’t collapse rows like GROUP BY does.

Syntax:

function_name(column) OVER (PARTITION BY column_to_group ORDER BY column_to_sort)

Example: Find GPA rank of each student within their department:

SELECT name, department_id, gpa,

RANK() OVER (PARTITION BY department_id ORDER BY gpa DESC) AS rank_in_dept

FROM students;

Output: Each department will have its own rank list.

Ranking Functions : Function Description Example Use

ROW_NUMBER() Assigns a unique number per row Deduplication, ordering

RANK() Ranks with gaps after ties “1, 2, 2, 4”

DENSE_RANK() Ranks without gaps “1, 2, 2, 3”

NTILE(n) Divides rows into n groups Percentiles, quartiles

Example:

SELECT name, department_id, gpa,

ROW_NUMBER() OVER (PARTITION BY department_id ORDER BY gpa DESC) AS row_num,

RANK() OVER (PARTITION BY department_id ORDER BY gpa DESC) AS rank_gap,

DENSE_RANK() OVER (PARTITION BY department_id ORDER BY gpa DESC) AS rank_dense

FROM students;

Practical Analytics with Window Functions :

Top N per Category Find top 1 GPA student per department:

```
SELECT * FROM
( SELECT name, department_id, gpa,
RANK() OVER (PARTITION BY department_id ORDER BY gpa DESC) AS rnk
FROM students )
 ranked WHERE rnk = 1;
```

Second Highest GPA in Department

```
SELECT * FROM
 ( SELECT name, department_id, gpa,
 DENSE_RANK() OVER (PARTITION BY department_id ORDER BY gpa DESC) AS rnk
FROM students )
ranked
WHERE rnk = 2;
```

Split Students into 4 GPA Quartiles (NTILE)

```
SELECT name, department_id, gpa,
NTILE(4) OVER (ORDER BY gpa DESC) AS quartile
FROM students;
```

Combining Window + Aggregate Logic We can mix aggregate and window logic for analytics.

Example: Show each student’s GPA and also their department average:

```
SELECT name, department_id, gpa,
 ROUND(AVG(gpa) OVER (PARTITION BY department_id), 2) AS dept_avg
 FROM students;
```

Example: Show GPA difference from department average:

```
SELECT name, department_id, gpa,
ROUND(gpa - AVG(gpa) OVER (PARTITION BY department_id), 2) AS diff_from_avg
FROM students;
```

Job-Level Applications You'll now be able to solve:

i)"Top 3 products by sales per region"

ii)"Find second highest salary per department"

iii)"Rank customers by spend percentile"

iv)"Compare employee salary to department average"

Day 18 Full Note

Advanced Joins & Complex Subqueries

Goal আজ তুমি শিখবে: i)Advanced join types**.**ii)Self joins,iii)Complex subqueries (including correlated),

iv)Combining joins and subqueries for real problems,v)SQL design patterns for analytics

Advanced Joins

1.INNER JOIN : Returns only matching rows from both tables.

SELECT a.name, b.course_name

FROM students a

INNER JOIN enrollments b ON a.student_id = b.student_id;

2. LEFT JOIN: Returns all rows from the left table, and matching rows from the right table. If no match, NULL is returned.

SELECT a.name, b.course_name

FROM students a

LEFT JOIN enrollments b ON a.student_id = b.student_id;

***3.*RIGHT JOIN:** Returns all rows from the right table, and matching rows from the left table.

SELECT a.name, b.course_name

FROM students a

RIGHT JOIN enrollments b ON a.student_id = b.student_id;

4. FULL OUTER JOIN (PostgreSQL) Returns all rows when there is a match in either table.

SELECT a.name, b.course_name

FROM students a

FULL OUTER JOIN enrollments b ON a.student_id = b.student_id;

5. CROSS JOIN Returns Cartesian product of rows.

SELECT a.name, b.course_name

FROM students a

CROSS JOIN courses b;

Self Join Self join হল একটি টেবিলকে নিজের সাথে join করা। Use cases: comparative analysis, hierarchical relationships, duplicate detection।

Example: Find employees whose salary is higher than their manager's salary:

SELECT

e1.name AS employee,

e2.name AS manager,

e1.salary

FROM employees e1

JOIN employees e2 ON e1.manager_id = e2.employee_id

WHERE e1.salary > e2.salary;

Complex Subqueries

Types of Subqueries

Single-row subquery → returns one value

SELECT name

FROM students

WHERE gpa = (SELECT MAX(gpa) FROM students);

Multiple-row subquery → returns multiple values

SELECT name

FROM students

WHERE department_id IN

(SELECT department_id FROM departments WHERE location = 'Dhaka');

Correlated subquery → depends on outer query

```
SELECT s.name, s.gpa
FROM students s
WHERE s.gpa > (SELECT AVG(gpa) FROM students WHERE department_id = s.department_id);
```

Nested subquery → subquery inside another subquery

```
SELECT name
 FROM students
 WHERE gpa = ( SELECT MAX(gpa) FROM students
WHERE department_id = ( SELECT department_id FROM departments WHERE name = 'Computer Science' ) );
```

Combining Joins + Subqueries Real-life queries often combine both for efficiency and clarity.

Example: Find names of students who are enrolled in the course with the highest credits.

```
SELECT s.name
FROM students s
JOIN enrollments e ON s.student_id = e.student_id
JOIN courses c ON e.course_id = c.course_id
WHERE c.credits = ( SELECT MAX(credits) FROM courses );
```

Real-World SQL Patterns Pattern Example Top N per group RANK() with PARTITION BY Compare rows in same table Self Join Filter based on aggregated data HAVING with Subquery Find gaps/missing data LEFT JOIN + NULL filter Hierarchical data Recursive CTE or Self Join

SQL Day 19 Full Notes

Topic: Advanced Aggregation & Analytics in SQL (Business-level Query Writing)

Learning Objectives আজ তুমি শিখবে—

i)Multiple column grouping ii)Conditional aggregation (CASE WHEN) , iii)ROLLUP, CUBE, GROUPING SETS

iv)Aggregate subquery in SELECT & HAVING Real business analytical patterns (Sales, GPA, etc.)

GROUP BY Basics (Revision) আগের দিনের মতো aggregate functions recap করি:

```
SELECT department,
 AVG(gpa) AS avg_gpa
 FROM students
GROUP BY department;
```

ব্যাখ্যা: প্রতিটি department অনুযায়ী GPA এর গড় বের করা।

GROUP BY Multiple Columns একাধিক column দিয়ে grouping করা যায়।

```
SELECT department, gender,
AVG(gpa) AS avg_gpa
FROM students
GROUP BY department, gender;
```

এতে department + gender কম্বিনেশন অনুযায়ী summary পাওয়া যাবে।

GROUP BY with Expressions Column expression বা calculation দিয়েও grouping করা যায়:

```
SELECT department,
ROUND(AVG(gpa), 2) AS rounded_avg
FROM students GROUP BY department;
```

ROUND() দিয়ে গড় GPA ২ দশমিক পর্যন্ত দেখানো হয়েছে।

Conditional Aggregation (CASE WHEN)

Conditional logic ব্যবহার করে আলাদা আলাদা segment summary তৈরি হয়:

SELECT department,

SUM(CASE WHEN gpa >= 3.5 THEN 1 ELSE 0 END) AS high_achievers,

SUM(CASE WHEN gpa < 3.5 THEN 1 ELSE 0 END) AS low_achievers

FROM students

GROUP BY department;

Use Case: Performance Report কোন department-এ কয়জন high achiever বা low achiever আছে।

Advanced GROUPING Techniques :ROLLUP (Subtotal + Grand Total)

SELECT department, gender,

AVG(gpa) AS avg_gpa

FROM students

GROUP BY ROLLUP(department, gender);

Generates subtotal per department + overall total row.

department gender avg_gpa Science Male 3.65 Science Female 3.72 Science NULL 3.68 ← subtotal NULL NULL 3.66 ← grand total

CUBE (All Combinations of Grouping Columns)

SELECT department, gender,

AVG(gpa) AS avg_gpa

FROM students

GROUP BY CUBE(department, gender);

Generates summary for:(department, gender),(department),(gender),(overall total)

GROUPING SETS (Custom Groupings)

SELECT department, gender,

AVG(gpa) AS avg_gpa

FROM students

GROUP BY GROUPING SETS ((department), (gender), ());

তুমি যেভাবে grouping করতে চাও, সেভাবে define করা যায়।

Aggregate Subquery Aggregate function-এর ভিতরেও subquery use করা যায়।

Example: যেসব department-এর average GPA overall average-এর থেকে বেশি:

SELECT department,

AVG(gpa) AS avg_gpa

FROM students

GROUP BY department

HAVING AVG(gpa) > (SELECT AVG(gpa) FROM students);

HAVING clause এ condition দেওয়া হয়েছে aggregate comparison এর জন্য।

Real-life Business Examples Scenario SQL Technique Department GPA comparison Aggregate subquery Top performing departments ORDER BY + LIMIT Category wise sales summary GROUP BY + SUM Sales Report by Region & Month ROLLUP Student segmentation CASE WHEN + GROUP BY

MySQL Specific Tip MySQL-এ ROLLUP ব্যবহার হয় নিচের মতো:

SELECT department, gender,

AVG(gpa) FROM students

GROUP BY department, gender WITH ROLLUP;

Practice Challenge (Try Yourself) Question: Write a query that shows each department's:

Number of male & female students, Total students, Average GPA,Compare with overall average GPA

(Hint: Use GROUP BY, CASE WHEN, and HAVING)

Quick Summary Topic Function Use Multiple Grouping GROUP BY Department + Gender summary Conditional Count CASE WHEN High vs Low achievers Subtotal ROLLUP Auto subtotal & total Full summary CUBE All combination summary Custom group GROUPING SETS Flexible grouping Comparison Aggregate Subquery Compare with overall average

Day 20

Advanced Window Functions (Part 2)

(LEAD, LAG, PERCENT_RANK, CUME_DIST, Moving Average, Cumulative Sum)

1. Ranking Functions Recap

ROW_NUMBER():Gives a unique sequence number for each row within a partition.

SELECT department,name,gpa,

ROW_NUMBER() OVER(PARTITION BY department ORDER BY gpa DESC) AS row_num

FROM students;

Use Case: When you need to pick *only one record per group* (like top 1 student).

2.**RANK() vs DENSE_RANK**():Both assign ranking, but handle ties differently.

SELECT department,name,gpa,

RANK() OVER(PARTITION BY department ORDER BY gpa DESC) AS rank_no,

DENSE_RANK() OVER(PARTITION BY department ORDER BY gpa DESC) AS dense_rank_no

FROM students;

Use Case:

RANK() → For competition ranking (skip ranks after ties).

DENSE_RANK() → For compact ranking (no gaps).

2. Aggregate Window Functions

You can use SUM(), AVG(), COUNT() inside an OVER() clause to create cumulative or partition-based aggregates.

SELECT department,name,gpa,

SUM(gpa) OVER(PARTITION BY department ORDER BY gpa DESC) AS cumulative_gpa,

AVG(gpa) OVER(PARTITION BY department) AS dept_avg

FROM students;

Use Case: Calculate running totals, departmental averages, or moving performance.

3. LEAD() & LAG()

i) LAG()

Gives value from the *previous* row in the same partition.

```
SELECT name, gpa,

LAG(gpa) OVER(ORDER BY gpa DESC) AS previous_gpa

FROM students;
```

ii) LEAD()

Gives value from the *next* row in the same partition.

```
SELECT  name,gpa,

LEAD(gpa) OVER(ORDER BY gpa DESC) AS next_gpa

FROM students;
```

Use Case:
Compare each student's GPA with previous or next student.
Perfect for trend analysis or performance difference.

iii)Difference Between Current and Previous GPA

```
SELECT name,gpa,

gpa - LAG(gpa) OVER(ORDER BY gpa DESC) AS gpa_diff

FROM students;
```

This helps you find who improved or dropped compared to the previous one.

iv). Moving Average & Cumulative Sum

Cumulative Sum:Use a window frame to accumulate values.

```
SELECT name,gpa,

SUM(gpa) OVER(ORDER BY gpa ROWS UNBOUNDED PRECEDING) AS cumulative_sum

FROM students;
```

Adds up all GPA values from the beginning up to the current row.

v)Moving Average: Average of a rolling window (like last 3 rows).

SELECT name,gpa,

ROUND(AVG(gpa) OVER(ORDER BY gpa ROWS BETWEEN 2 PRECEDING AND CURRENT ROW),2) AS moving_avg

FROM students;

Use Case: Trend smoothing average performance over last 3 observations.

vi) PERCENT_RANK() & CUME_DIST()

PERCENT_RANK(): Shows how far a row ranks relative to others (0 to 1 scale).

SELECT name,gpa,

ROUND(PERCENT_RANK() OVER(ORDER BY gpa DESC),2) AS percent_rank

FROM students;

Formula:
(Rank - 1) / (Total_Rows - 1)

vii) CUME_DIST()

Cumulative distribution: proportion of rows less than or equal to the current row.

SELECT name, gpa,

ROUND(CUME_DIST() OVER(ORDER BY gpa DESC),2) AS cume_dist

FROM students;

GPA	CUME_DIST()	Meaning
4.0	0.2	Top 20%
3.8	0.6	Top 60%

Perfect for percentile or rank-based grading systems.

6. Practical Use Cases

Use Case	Function
Find top 3 students per department	ROW_NUMBER() / RANK()
Compare GPA with previous student	LAG()
Calculate performance growth	LEAD()
Find cumulative GPA trend	SUM() OVER()
Calculate moving average	AVG() OVER(ROWS BETWEEN ...)
Determine percentile rank	PERCENT_RANK() / CUME_DIST()

7. Quick Summary Table

Function	Purpose	Example Use
ROW_NUMBER()	Unique numbering	Identify top 1
RANK()	Ranking with gaps	Competition ranking
DENSE_RANK()	Ranking without gaps	Ordered rank
LAG()	Previous row value	Compare with past
LEAD()	Next row value	Compare with future
SUM() OVER()	Running total	Cumulative GPA
AVG() OVER()	Moving average	Rolling mean
PERCENT_RANK()	Relative position	Percentile
CUME_DIST()	Cumulative distribution	Grading threshold

Day 21

SQL Query Optimization & Performance Tuning (Complete Note)

Learning Goal: To make SQL queries faster, cleaner, and scalable by using indexes, query plans, and optimization techniques — just like a real Data Analyst or Data Engineer.

What is Query Optimization?

Query Optimization হল এমন একটি প্রক্রিয়া যেখানে SQL Engine দেখে কোন উপায়ে query চালালে সবচেয়ে কম সময় ও resource লাগে।Database optimizer query plan তৈরি করে এবং সেই অনুযায়ী query execute করে।

Example:

*EXPLAIN SELECT * FROM students WHERE department = 'CSE';*

Output দেখাবে কোন index use হয়েছে, কত rows scan হয়েছে, estimated cost কত ইত্যাদি।

Execution Plan & Cost : EXPLAIN / EXPLAIN ANALYZE PostgreSQL Example:

*EXPLAIN ANALYZE SELECT * FROM students WHERE gpa > 3.5;*

EXPLAIN → Query plan দেখায় (but doesn't run)

EXPLAIN ANALYZE → Query চালিয়ে actual performance report দেয়

Cost → যত কম, তত ভালো performance

Indexing in SQL

What is an Index?

Index হল একটি special data structure (যেমন B-tree) যা database কে দ্রুত ডাটা খুঁজে পেতে সাহায্য করে।

Example:

CREATE INDEX idx_students_name ON students(name);

Index থাকলে WHERE, JOIN, এবং ORDER BY query অনেক দ্রুত চলে।

Types of Index: Type Description

Clustered Index -Actual table data সাজানো থাকে index অনুযায়ী (e.g., PRIMARY KEY)

Non-Clustered Index- আলাদা index structure তৈরি হয়

Composite Index- একাধিক কলাম নিয়ে তৈরি (e.g., ON students(department, gpa))

When Index Helps When you frequently use column in WHERE, JOIN, ORDER BY

When table is large and data is repeatedly searchedWhen Index Hurts Frequent INSERT/UPDATE/DELETE হলে index update করতে হয়, ফলে performance কমে Too many indexes = unnecessary overhead

Optimize WHERE and JOIN

Best Practices:

i)Use SELECT specific columns,

ii)avoid SELECT *

iii)Always filter early (in WHERE clause)

iv)Avoid functions in WHERE (e.g., LOWER(name) blocks index)

v)Use JOINs smartly on indexed columns

Example (Bad):

*SELECT * FROM students WHERE LOWER(name) = 'tanvir';*

Example (Better):

*SELECT * FROM students WHERE name = 'Tanvir';*

Subquery vs CTE vs Temp Table Type Description Performance Subquery Nested query inside main Slow if repeated CTE (WITH) Readable and reusable Better readability, same cost Temp Table Physical temp storage Best for reuse with large data

Example:

WITH high_gpa AS

*(SELECT * FROM students WHERE gpa > 3.5)*

SELECT name FROM high_gpa

WHERE department = 'CSE';

Avoid Expensive Operations Operation Use Instead DISTINCT Use GROUP BY if possible UNION Use UNION ALL if duplicates are okay IN() Replace with EXISTS() for better performance Function in WHERE Avoid if indexed column

Example:

Slow

*SELECT * FROM students WHERE id IN (SELECT student_id FROM enrollments);*

Faster

SELECT s. FROM students s WHERE EXISTS*

(SELECT 1 FROM enrollments e WHERE s.id = e.student_id);

Materialized Views (Advanced)

Used for caching query results that are expensive to compute every time.

Example:

CREATE MATERIALIZED VIEW top_students AS

SELECT department, name, gpa FROM students WHERE gpa >= 3.5;

To refresh updated data:

REFRESH MATERIALIZED VIEW top_students;

Query Execution Time Check PostgreSQL:

*\timing SELECT * FROM students WHERE gpa > 3.8;*

SQL Server / MySQL:

SET STATISTICS TIME ON;

Rewrite for Optimization (Example)

Before:

*SELECT * FROM students WHERE department = 'CSE' AND gpa > 3.5;*

After (with index + specific columns):

CREATE INDEX idx_dept_gpa ON students(department, gpa);

SELECT name, gpa FROM students WHERE department = 'CSE' AND gpa > 3.5;

Summary Table Concept Goal Example EXPLAIN See query plan EXPLAIN SELECT ... INDEX Speed up lookup CREATE INDEX ... JOIN Optimization Use indexed keys JOIN ON id Avoid DISTINCT Use GROUP BY EXISTS Faster than IN Materialized View Cache heavy queries CREATE MATERIALIZED VIEW ...

Expected Outcome (End of Day 21) By the end of Day 21, you'll be able to: Understand how the database engine executes your query .Use indexes to speed up performance .Compare CTE vs Subquery vs Temp Table .Analyze query cost using EXPLAIN . Apply real optimization in your own datasets

Day 22

SQL for Real-World Data Analysis (Full Note)

Learning Goal Analyze real business datasets like E-commerce sales, customer behavior, product performance.

Learn to translate business questions into SQL queries.

Build job-ready analytical queries for dashboards, reports, and interview challenges.

Business-Oriented Query Thinking Understand problem statement:

What are you trying to analyze?

e.g., "Top 5 products by sales" → aggregate SUM(quantity * price) and rank.

Identify table relationships:Customers → Orders → Order_Items → Products

Choose appropriate SQL technique:

JOINs, Subquery, CTE, Window Functions, Aggregations, CASE WHEN

Table Structure (Example)

Customers

```
CREATE TABLE customers (
 customer_id SERIAL PRIMARY KEY,
 name VARCHAR(50),
 country VARCHAR(50),
 join_date DATE );
```

Orders

```
CREATE TABLE orders (
order_id SERIAL PRIMARY KEY,
customer_id INT,
order_date DATE,
 total_amount NUMERIC(10,2) );
```

Products

CREATE TABLE products (

product_id SERIAL PRIMARY KEY,

product_name VARCHAR(50),

category VARCHAR(50),

price NUMERIC(10,2));

Order Items

CREATE TABLE order_items (

order_item_id SERIAL PRIMARY KEY,

order_id INT,

product_id INT,

quantity INT);

Common Business Queries Customer Analytics

1. Count new customers per month:

SELECT DATE_FORMAT(join_date,'%Y-%m') AS month,

COUNT(customer_id) AS new_customers

FROM customers

GROUP BY month;

2.Top countries by customers:

SELECT country,

COUNT(*) AS num_customers

FROM customers

GROUP BY country

ORDER BY num_customers DESC;

3.Sales Analytics Total revenue by month:

```
SELECT

DATE_FORMAT(order_date,'%Y-%m') AS month,

SUM(total_amount) AS revenue

 FROM orders

 GROUP BY month ORDER BY month;
```

4.Top 5 best-selling products:

```
SELECT p.product_name,

SUM(oi.quantity) AS total_sold

FROM order_items oi

JOIN products p ON oi.product_id = p.product_id

GROUP BY p.product_name

ORDER BY total_sold DESC

LIMIT 5;
```

5.Product Performance Products with declining sales:

```
SELECT

 p.product_name,

SUM(CASE WHEN MONTH(o.order_date)=1 THEN oi.quantity ELSE 0 END) AS Jan,

SUM(CASE WHEN MONTH(o.order_date)=2 THEN oi.quantity ELSE 0 END) AS Feb

FROM order_items oi

JOIN orders o ON oi.order_id = o.order_id

JOIN products p ON oi.product_id = p.product_id

GROUP BY p.product_name;
```

6.Average order value per customer:

```
SELECT o.customer_id,

ROUND(AVG(total_amount),2) AS avg_order_value

FROM orders o

GROUP BY o.customer_id;
```

6.Retention Metrics Repeat customers:

```
SELECT
 customer_id
FROM orders
GROUP BY customer_id
HAVING COUNT(order_id) >= 2;
```

7.Month with highest repeat order rate:

```
SELECT
DATE_FORMAT(order_date,'%Y-%m') AS month,
COUNT(DISTINCT customer_id) AS repeat_customers
 FROM orders
GROUP BY month
ORDER BY repeat_customers DESC;
```

Advanced SQL Concepts in

Practice Concept Use Case JOINs Connect customers, orders, products, order_items CTEs / Subqueries Stepwise analysis & readability Aggregations SUM, AVG, COUNT for KPIs Window Functions RANK, ROW_NUMBER for top products/customers CASE WHEN Category grouping or conditional metrics DATE Functions MONTH(), YEAR(), DATE_FORMAT() for trends

Key Analytical Metrics Metric Formula / SQL

1.Total Revenue = SUM(total_amount)

2.Average Order Value (AOV)= SUM(total_amount)/COUNT(orders)

Repeat Rate % of customers with ≥2 orders Monthly Active Customers COUNT(DISTINCT customer_id) Top Products SUM(quantity * price)

Customer Lifetime Value (LTV) SUM(total_amount) per customer

Optimization Tips Filter early using WHERE

Use EXPLAIN for query cost insights

Index columns often used in JOIN, WHERE, ORDER BY

i)Avoid SELECT *

ii)Use UNION ALL instead of UNION if duplicates are okay

iii)Use CTEs or temp tables for complex stepwise calculation

By the end of Day 22, you will be able to: Translate business questions into SQL queries

i)Analyze real E-commerce datasets

ii)Calculate sales, customer, product, and retention metrics

iii)Optimize queries for better performance

iv)Solve real-life interview & HackerRank SQL problems

Day 23

Detailed Notes: Advanced SQL Analytics (E-Commerce Dataset)

1.Advanced Joins & Multi-Table Analysis

আমরা Day 22 থেকে অনেক join শিখেছি। এখন complex multi-level joins নিয়ে কাজ করব।

Scenario:Customers → Orders → Order_Items → Products

Example: Total revenue by customer country and product category

```
SELECT c.country, p.category,
SUM(oi.quantity * p.price) AS total_revenue
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_items oi ON o.order_id = oi.order_id
 JOIN products p ON oi.product_id = p.product_id
GROUP BY c.country, p.category
 ORDER BY c.country, total_revenue DESC;
```

Tip:Multi-level joins crucial for business insights.Always alias tables (c, o, oi, p) for readability.

2.Complex Aggregations Conditional Aggregation

Use CASE WHEN inside aggregation to filter on-the-fly.

```
SELECT c.name,

SUM(CASE WHEN p.category='Electronics' THEN oi.quantity * p.price ELSE 0 END) AS electronics_spent, SUM(CASE
WHEN p.category='Clothing' THEN oi.quantity * p.price ELSE 0 END) AS clothing_spent

FROM customers c

JOIN orders o ON c.customer_id = o.customer_id

 JOIN order_items oi ON o.order_id = oi.order_id

JOIN products p ON oi.product_id = p.product_id

 GROUP BY c.name;
```

3.Multi-Column Grouping

```
GROUP BY c.country, p.category
```

Analyze country-category revenue at once.

HAVING Filter aggregated results

```
HAVING SUM(oi.quantity * p.price) > 500
```

Advanced Window Functions

Window functions allow row-level calculations with aggregation context.

Function	**Use Case**
ROW_NUMBER()	Unique row numbers per partition
RANK()	Rank rows with gaps for ties
DENSE_RANK()	Rank rows without gaps for ties
SUM() OVER()	Running total
AVG() OVER()	Rolling average

Example: Top-selling product per category

```
SELECT * FROM
( SELECT p.category, p.product_name, SUM(oi.quantity) AS total_sold,
RANK() OVER (PARTITION BY p.category ORDER BY SUM(oi.quantity) DESC) AS rank
FROM order_items oi
 JOIN products p ON oi.product_id = p.product_id
GROUP BY p.category, p.product_name ) ranked
WHERE rank = 1;
```

Tip: Use PARTITION BY to rank within each group.

Subqueries & CTEs

Inline Subqueries

```
SELECT name
FROM customers WHERE customer_id IN
( SELECT customer_id FROM orders WHERE total_amount > 500 );
```

CTE (Common Table Expression)

```
WITH customer_spending AS (
SELECT c.customer_id,
SUM(oi.quantity * p.price) AS total_spent
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_items oi ON o.order_id = oi.order_id
JOIN products p ON oi.product_id = p.product_id
GROUP BY c.customer_id )
 SELECT * FROM customer_spending WHERE total_spent > 1000;
```

Recursive CTE (Optional, Hierarchy/Path) Useful for managers → subordinates, not e-commerce but good to know.

Date & Time Analytics Extract Month/Year

SELECT

EXTRACT(MONTH FROM order_date) AS month,

SUM(total_amount) AS monthly_revenue

FROM orders

GROUP BY month

ORDER BY month;

Calculate Duration Between Dates

SELECT customer_id,

MIN(order_date) AS first_order,

MAX(order_date) AS last_order,

DATEDIFF(MAX(order_date), MIN(order_date)) AS days_between

FROM orders

GROUP BY customer_id;

Revenue Trends Monthly, quarterly, yearly aggregation for business insights.

Real-World Business Analysis Patterns Top customers by total revenue

Top products per category,Country-wise revenue & order count,Monthly/quarterly revenue trend,High-value orders & repeat customers,Products never ordered → LEFT JOIN + IS NULL,Average order value per customer,

Category-wise quantity analysis,customer first & last order date,Top N orders per month / category → Window functions

Bonus Techniques Handling NULLs:

COALESCE(column, default_value)

Conditional aggregation:

SUM(CASE WHEN category='Electronics' THEN quantity ELSE 0 END)

Combining multiple techniques:

Joins + Group By + Window Functions + CTE = job-ready SQL

Best Practices for Job-Ready

Queries Always alias tables (c, o, oi, p)

Follow logical order: JOIN → WHERE → GROUP BY → HAVING → ORDER BY → LIMIT

Use CTEs for readability in complex queries

Test queries on subset of data first

Comment queries for clarity in scripts/reports

Think in business logic terms, not just SQL syntax

Day 24

Advanced SQL Analytics (Updated with Percentage)

Multi-Level Aggregation + Conditional Aggregation : Aggregate by multiple dimensions: country × category × month

Conditional aggregation: CASE WHEN ... THEN ... END

Filter aggregated results: HAVING

Example: Revenue per country & category (with percentage)

```
WITH country_category_revenue AS (
 SELECT c.country, p.category,
SUM(oi.quantity * p.price) AS revenue
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_items oi ON o.order_id = oi.order_id
JOIN products p ON oi.product_id = p.product_id
GROUP BY c.country, p.category )
SELECT country, category, revenue,
ROUND(revenue * 100 / SUM(revenue) OVER (PARTITION BY country), 2) AS percentage_of_country_revenue FROM country_category_revenue
ORDER BY country, percentage_of_country_revenue DESC;
```

Skill: Multi-level aggregation + percentage calculation + window function

Advanced Window Functions Running total:

i)cumulative revenue

ii)Rolling average: revenue over last N months

iii)Top-N per category / customer

Example: Top 3 products per category by quantity sold

```
SELECT * FROM

( SELECT p.category, p.product_name,

SUM(oi.quantity) AS total_sold, RANK() OVER(PARTITION BY p.category ORDER BY SUM(oi.quantity) DESC) AS rank,

ROUND(SUM(oi.quantity)*100.0 / SUM(SUM(oi.quantity)) OVER(PARTITION BY p.category), 2) AS
percentage_of_category

 FROM order_items oi

JOIN products p ON oi.product_id = p.product_id

GROUP BY p.category, p.product_name ) ranked

 WHERE rank <= 3;
```

Skill: Window functions + percentage share per group + ranking

Date & Time Analytics Extract year, month, day

Customer first & last purchase date

Month-over-month revenue trend

Customer lifetime / days between orders

Example: Percentage contribution of each month to total yearly revenue

```
SELECT EXTRACT(YEAR FROM order_date) AS year,

EXTRACT(MONTH FROM order_date) AS month,

SUM(total_amount) AS month_revenue,

ROUND(SUM(total_amount)*100.0 / SUM(SUM(total_amount)) OVER(PARTITION BY EXTRACT(YEAR FROM
order_date)), 2) AS percentage_of_year

FROM orders

GROUP BY year, month

ORDER BY year, month;
```

Average Order Value (AOV) & Customer Metrics

AOV = total revenue ÷ total number of orders

Repeat customers & high-value orders

Customer revenue contribution percentage

WITH customer_total AS (

SELECT c.customer_id, c.name,

SUM(o.total_amount) AS total_spent

FROM customers c

JOIN orders o ON c.customer_id = o.customer_id

GROUP BY c.customer_id, c.name)

*SELECT *, ROUND(total_spent * 100.0 / SUM(total_spent) OVER(), 2) AS percentage_of_total_revenue*

FROM customer_total

ORDER BY total_spent DESC;

Subqueries & CTEs (Advanced) Multi-level CTEs for stepwise aggregation

Inline subqueries in WHERE / SELECT

Combine with percentage calculation for business insights

Real-World Business Insights Patterns Top customers by revenue & order count

Top products per category / country

Monthly / quarterly revenue trend + cumulative revenue + percentage share

Product contribution to revenue (%)

High-value orders & repeat customer analysis

Bonus Techniques Handle NULLs:

COALESCE(column, default_value)

Conditional aggregation + filtering

Combine joins + aggregation + window functions + CTEs + percentage for job-ready queries

Month-over-Month (MoM) Trend Analysis Purpose:

দেখার জন্য revenue বা sales কিভাবে মাসে মাসে পরিবর্তিত হচ্ছে।

সাধারণত growth rate বের করা হয়।

SQL Example (MySQL/PostgreSQL compatible):

WITH monthly_revenue AS(

SELECT DATE_FORMAT(order_date, '%Y-%m') AS month,

SUM(total_amount) AS revenue

FROM orders

GROUP BY month)

SELECT month, revenue,

LAG(revenue) OVER (ORDER BY month) AS previous_month_revenue,

*ROUND((revenue - LAG(revenue) OVER (ORDER BY month)) * 100.0 / LAG(revenue) OVER (ORDER BY month), 2) AS mom_growth_percentage*

FROM monthly_revenue

ORDER BY month;

Explanation:

LAG(revenue) OVER (ORDER BY month) → আগের মাসের revenue নেয়

*(current - previous) / previous * 100* → growth percentage

Output shows month, revenue, previous month revenue, MoM growth %

Year-over-Year (YoY) Trend Analysis Purpose:

দেখার জন্য sales বা revenue কিভাবে প্রতি বছর একই মাসে পরিবর্তিত হচ্ছে।সাধারণত seasonal trends বুঝতে ব্যবহার হয়।

SQL Example:

```
WITH monthly_revenue AS (
SELECT EXTRACT(YEAR FROM order_date) AS year,
EXTRACT(MONTH FROM order_date) AS month,
SUM(total_amount) AS revenue
FROM orders
 GROUP BY year, month )
 SELECT
year, month, revenue,
LAG(revenue) OVER (PARTITION BY month ORDER BY year) AS last_year_revenue,
ROUND((revenue - LAG(revenue) OVER (PARTITION BY month ORDER BY year)) * 100.0 / LAG(revenue) OVER
(PARTITION BY month ORDER BY year), 2) AS yoy_growth_percentage
 FROM monthly_revenue ORDER BY month, year;
```

Explanation:

PARTITION BY month → একই মাসের data year-wise partition করে

LAG(revenue) → আগের বছরের revenue নেয়

(current - last_year) / last_year * 100 → YoY growth percentage

Key Points MoM → month-to-month change

YoY → same month comparison year over year

Use window functions (LAG, LEAD) for previous period values

Combine with ROUND() for readable percentages

Useful for dashboards, reports, and business insights

Day 25

Advanced SQL (MySQL Version)

Aggregation | Window Functions | Percentages | Trends | Business Insights

Advanced Aggregation in MySQL

Basic GROUP BY Recap

```
SELECT category,
SUM(price * quantity) AS total_sales
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id
 GROUP BY category;
```

Conditional Aggregation with CASE: Counts how many orders exceed $500 per country.

```
SELECT country,
SUM(CASE WHEN total_amount > 500 THEN 1 ELSE 0 END) AS high_value_orders,
SUM(total_amount) AS total_revenue
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
 GROUP BY country;
```

GROUP BY with ROLLUP (MySQL)

```
SELECT category, country,
SUM(total_amount) AS total_sales
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
JOIN order_items oi ON o.order_id = oi.order_id
JOIN products p ON oi.product_id = p.product_id
GROUP BY category, country WITH ROLLUP;
```

shows subtotals per category and grand total.

Advanced Window Functions (MySQL 8+)

Ranking: RANK(), DENSE_RANK(), ROW_NUMBER()

SELECT category, product_name,

SUM(quantity) AS total_sold, RANK() OVER (PARTITION BY category ORDER BY SUM(quantity) DESC) AS category_rank

FROM order_items oi

JOIN products p ON oi.product_id = p.product_id

GROUP BY category, product_name;

Ranks products within each category.

Cumulative Total (Running Total) : Tracks revenue growth over time.

SELECT order_date,

SUM(total_amount) OVER (ORDER BY order_date) AS running_total

FROM orders;

Moving Average (Rolling Average): Shows 3-day moving average of total amount.

SELECT order_date,

AVG(total_amount) OVER (ORDER BY order_date ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS rolling_avg

FROM orders;

Percent of Total using Window Function: *Each product's contribution to total sales.*

SELECT product_name,

SUM(quantity) AS total_sold,

ROUND(100 * SUM(quantity) / SUM(SUM(quantity)) OVER (), 2) AS percent_of_total

FROM order_items oi

JOIN products p ON oi.product_id = p.product_id

GROUP BY product_name;

Date & Time Analytics (MySQL)

Extract Year, Month, Day

```
SELECT
 YEAR(order_date) AS order_year,
MONTH(order_date) AS order_month,
SUM(total_amount) AS monthly_sales
 FROM orders
GROUP BY order_year, order_month
ORDER BY order_year, order_month;
```

Month-over-Month (MoM) Growth

```
WITH monthly AS (
SELECT DATE_FORMAT(order_date, '%Y-%m') AS month,
SUM(total_amount) AS revenue
FROM orders
GROUP BY DATE_FORMAT(order_date, '%Y-%m') )
SELECT month, revenue,
LAG(revenue) OVER (ORDER BY month) AS prev_month,
ROUND((revenue - LAG(revenue) OVER (ORDER BY month)) * 100 / LAG(revenue) OVER (ORDER BY month), 2) AS
mom_growth
 FROM monthly;
```

Year-over-Year (YoY) Growth

```
WITH yearly AS (
SELECT YEAR(order_date) AS year,
SUM(total_amount) AS revenue
FROM orders
GROUP BY YEAR(order_date) )
 SELECT year, revenue,
LAG(revenue) OVER (ORDER BY year) AS prev_year,
 ROUND((revenue - LAG(revenue) OVER (ORDER BY year)) * 100 / LAG(revenue) OVER (ORDER BY year), 2) AS
yoy_growth FROM yearly;
```

Percentage & Contribution Analysis

Customer Contribution to Total Revenue

```
SELECT c.name,
SUM(o.total_amount) AS total_spent,
ROUND(100 * SUM(o.total_amount) / SUM(SUM(o.total_amount)) OVER (), 2) AS percent_of_total
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
GROUP BY c.name
 ORDER BY total_spent DESC;
```

Top 3 Products per Category (% Contribution)

```
WITH cat_sales AS (
SELECT p.category, p.product_name,
SUM(oi.quantity) AS total_qty,
 SUM(SUM(oi.quantity)) OVER (PARTITION BY p.category) AS category_total,
RANK() OVER (PARTITION BY p.category ORDER BY SUM(oi.quantity) DESC) AS rnk
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id
GROUP BY p.category, p.product_name )
SELECT category, product_name, total_qty,
ROUND(100 * total_qty / category_total, 2) AS pct_contribution
FROM cat_sales
WHERE rnk <= 3;
```

Real Business Insight Queries Use Case Key Concept MySQL Feature Top Customers Aggregation + Ranking SUM(), RANK() Product Share Percentage of total SUM() / SUM() OVER() Month-over-Month Growth comparison LAG()

Day 26

Advanced MySQL Analytics (Part 2)

Goal: Master advanced ranking, comparison, and nested query analysis to reach job-ready level (HackerRank/Datalemur standard).

1. Quick Recap from Day 25

Before we move ahead — remember:

CASE WHEN used for segmentation

Month-over-Month (MoM) growth

Rolling totals using SUM() OVER()

1.Percentage calculation using SUM() OVER() or total aggregation

Example:

```
SELECT country,
COUNT(order_id) AS total_orders,
SUM(CASE WHEN total_amount > 500 THEN 1 ELSE 0 END) AS high_value_orders,
ROUND(SUM(CASE WHEN total_amount > 500 THEN 1 ELSE 0 END)*100.0 / COUNT(*), 2) AS
high_value_percentage
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
GROUP BY country;
```

Ranking Functions (Real Business Use Case)

Ranking is essential in analytics it helps identify top performers, best-selling products, most valuable customers, etc.

2.Rank products by revenue in each category

```
SELECT p.category,p.product_name,

SUM(oi.quantity * p.price) AS total_sales,

RANK() OVER (PARTITION BY p.category ORDER BY SUM(oi.quantity * p.price) DESC) AS category_rank

FROM products p

JOIN order_items oi ON p.product_id = oi.product_id

GROUP BY p.category, p.product_name;
```

Explanation:

PARTITION BY = divide data by category

ORDER BY = rank within category

RANK() = allows same rank for ties

3.Top 3 products per category

```
WITH ranked_products AS (

SELECT p.category,p.product_name,

SUM(oi.quantity) AS total_qty,

RANK() OVER (PARTITION BY p.category ORDER BY SUM(oi.quantity) DESC) AS rnk

FROM products p

JOIN order_items oi ON p.product_id = oi.product_id

GROUP BY p.category, p.product_name

)

SELECT * FROM ranked_products

WHERE rnk <= 3;
```

This helps find the top 3 products sold in each category.

Comparative Analysis

Compare performance of one group against average or other groups.

4.Find customers spending above average

```
SELECT c.name,

SUM(o.total_amount) AS total_spent

FROM customers c

JOIN orders o ON c.customer_id = o.customer_id

GROUP BY c.name

HAVING total_spent > (SELECT AVG(total_amount) FROM orders);
```

Used in retention, loyalty, or customer value analysis.

Segmentation with CASE WHEN

Classify orders or customers based on spending level.

5.Order segmentation

```
SELECT order_id,total_amount,

CASE

WHEN total_amount > 1000 THEN 'High Value'

WHEN total_amount BETWEEN 500 AND 1000 THEN 'Medium Value'

ELSE 'Low Value'

END AS order_segment

FROM orders;
```

***6.*Count and percentage of each segment**

```
SELECT

CASE WHEN total_amount > 1000 THEN 'High Value'

      WHEN total_amount BETWEEN 500 AND 1000 THEN 'Medium Value'

ELSE 'Low Value' END AS segment,

COUNT(*) AS total_orders,

ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM orders), 2) AS percent_share

FROM orders

GROUP BY segment;
```

Nested & Correlated Subqueries

Subqueries help filter data based on another query’s result.

6.Products above category average price

```
SELECT product_name,category,price
FROM products p1
WHERE price > (SELECT AVG(price) FROM products p2
WHERE p2.category = p1.category
);
```

Finds premium-priced products.

7.Customers who ordered more than average total amount

```
SELECT name, total_amount
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
WHERE total_amount > ( SELECT AVG(total_amount) FROM orders);
```

Advanced Window Functions

Window functions let you calculate running totals, moving averages, or cumulative percentages.

8.Running total revenue (month-wise)

```
SELECT
DATE_FORMAT(order_date, '%Y-%m') AS month,
SUM(total_amount) AS monthly_revenue,
SUM(SUM(total_amount)) OVER (ORDER BY DATE_FORMAT(order_date, '%Y-%m')) AS running_total
FROM orders
GROUP BY DATE_FORMAT(order_date, '%Y-%m')
ORDER BY month;
```

9.Percentage contribution by month

```
SELECT

DATE_FORMAT(order_date, '%Y-%m') AS month,

SUM(total_amount) AS monthly_revenue,

ROUND(SUM(total_amount) * 100.0 / SUM(SUM(total_amount)) OVER (), 2) AS pct_of_total

FROM orders

GROUP BY DATE_FORMAT(order_date, '%Y-%m')

ORDER BY month;
```

Used in sales trend and financial performance dashboards.

Date-Based Business Analysis

Extract Year, Month, and analyze growth

```
SELECT

YEAR(order_date) AS year,

MONTH(order_date) AS month,

SUM(total_amount) AS revenue

FROM orders

GROUP BY YEAR(order_date), MONTH(order_date)

ORDER BY year, month;
```

Month-over-Month Growth

```
WITH monthly_revenue AS (
SELECT
DATE_FORMAT(order_date, '%Y-%m') AS month,
SUM(total_amount) AS revenue
FROM orders
GROUP BY DATE_FORMAT(order_date, '%Y-%m')
)
SELECT month,revenue,
LAG(revenue) OVER (ORDER BY month) AS prev_month,
ROUND((revenue - LAG(revenue) OVER (ORDER BY month)) * 100.0 / LAG(revenue) OVER (ORDER BY month), 2) AS mom_growth
FROM monthly_revenue;
```

Helps measure month-over-month growth trends.

Real Job-Ready Insights

After Day 26, you'll be able to:

1. Find top performers using RANK()
2. Classify data into segments using CASE WHEN
3. Use subqueries to compare with averages
4. Perform running totals and growth analysis
5. Generate business KPIs like growth %, contribution %, segmentation

DAY 27

Business Metrics, Retention & Behavioral Analytics

Concept Overview Today’s focus is on how businesses measure growth, customer value, and revenue patterns using SQL. These are real-world KPI metrics that analysts use in dashboards (like Power BI or Tableau) and interview problems (like at Google, Meta, or Shopee).

We’ll master:

i)Customer Lifetime Value (CLV)

ii)Average Order Value (AOV)

iii) Retention & Cohort basics

iv) Revenue Contribution

v) Ranking & Segmentation

vi) Month-over-Month growth

Customer Lifetime Value (CLV) Definition: The total amount a customer has spent with the company over their lifetime.

Formula:

CLV=∑(total_amount per customer)

SQL Query:

```
SELECT c.customer_id, c.name,

COUNT(DISTINCT o.order_id) AS total_orders,

SUM(o.total_amount) AS lifetime_value,

ROUND(SUM(o.total_amount) / COUNT(DISTINCT o.order_id), 2) AS avg_order_value

FROM customers c

JOIN orders o ON c.customer_id = o.customer_id

GROUP BY c.customer_id, c.name

ORDER BY lifetime_value DESC;
```

Insights:

i)Helps identify high-value customers

ii)Used for marketing segmentation & loyalty programs

Average Order Value (AOV) Definition: The average revenue per order placed.

Formula:

$$AOV = Total\ Revenue / \ Number\ \ of\ \ Orders$$

SQL Example:

```
SELECT

ROUND(SUM(total_amount) / COUNT(order_id), 2) AS avg_order_value

FROM orders;
```

Business Insight:

Low AOV → customers buy cheaper items

High AOV → premium customers or bundles

Revenue by Product Category Goal: Find how much each category contributes to total revenue.

```
SELECT p.category,

SUM(oi.quantity * oi.price) AS total_revenue,

ROUND(SUM(oi.quantity * oi.price) * 100.0 / SUM(SUM(oi.quantity * oi.price)) OVER (), 2) AS percent_share FROM
order_items oi

JOIN products p ON oi.product_id = p.product_id

GROUP BY p.category

ORDER BY total_revenue DESC;
```

Concept Used:

SUM() OVER() → calculates share of total revenue

Contribution Analysis Goal: Determine what % of total revenue comes from each country or product.

SQL:

```
SELECT c.country,

SUM(o.total_amount) AS total_revenue,

ROUND(SUM(o.total_amount) * 100.0 / SUM(SUM(o.total_amount)) OVER (), 2) AS contribution_pct

FROM customers c

JOIN orders o ON c.customer_id = o.customer_id

GROUP BY c.country

ORDER BY contribution_pct DESC;
```

Insight:

i)Used for revenue dashboards

ii)Helps decide where to focus marketing

Month-over-Month (MoM) Growth Formula:

*MoM Growth % = (Current – Previous) *100/(Previous)*

SQL:

```
WITH monthly_revenue AS (
SELECT DATE_FORMAT(order_date, '%Y-%m') AS month,
SUM(total_amount) AS revenue
FROM orders
GROUP BY DATE_FORMAT(order_date, '%Y-%m') )
SELECT month, revenue,
LAG(revenue) OVER (ORDER BY month) AS prev_revenue,
ROUND((revenue - LAG(revenue) OVER (ORDER BY month)) * 100.0 / LAG(revenue) OVER (ORDER BY month), 2) AS growth_pct
FROM monthly_revenue;
```

Insight:

i)Shows revenue trends

ii)Used in monthly performance reports

Ranking Customers within Each Country Goal: Identify top customers per country by spending.

```
SELECT c.country, c.name,
SUM(o.total_amount) AS total_spent,
RANK() OVER(PARTITION BY c.country ORDER BY SUM(o.total_amount) DESC) AS rank_in_country
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.country, c.name
ORDER BY c.country, rank_in_country;
```

Concept Used:RANK() with PARTITION BY = ranking within groups

Customer Segmentation Based on CLV Goal: Group customers by lifetime spending (business segmentation).

Segment Condition High CLV > 3000 Medium 1500 ≤ CLV ≤ 3000 Low CLV < 1500

SQL:

```
WITH customer_value AS (
 SELECT customer_id,
SUM(total_amount) AS clv
FROM orders
GROUP BY customer_id )
SELECT customer_id, clv,
CASE
WHEN clv > 3000 THEN 'High'
WHEN clv BETWEEN 1500 AND 3000 THEN 'Medium'
ELSE 'Low'
END AS segment
FROM customer_value;
```

Churn Identification (Inactive Customers) Goal: Detect customers who haven’t purchased recently.

```
SELECT customer_id,
MAX(order_date) AS last_purchase,
DATEDIFF(CURDATE(), MAX(order_date)) AS days_inactive,
CASE WHEN DATEDIFF(CURDATE(), MAX(order_date)) > 90 THEN 'Churned'
ELSE 'Active' END AS status
FROM orders
GROUP BY customer_id;
```

Insight:

i)Used for retention campaigns

ii)“Churned” customers may need reactivation offers

Cohort Analysis (Intro Level) Goal: Group users by first purchase month and track retention.

```
WITH first_purchase AS (
SELECT customer_id,
MIN(DATE_FORMAT(order_date, '%Y-%m')) AS cohort_month
FROM orders
GROUP BY customer_id )
SELECT f.cohort_month,
DATE_FORMAT(o.order_date, '%Y-%m') AS order_month,
COUNT(DISTINCT o.customer_id) AS active_customers
FROM first_purchase f
JOIN orders o ON f.customer_id = o.customer_id
GROUP BY f.cohort_month, DATE_FORMAT(o.order_date, '%Y-%m')
 ORDER BY f.cohort_month, order_month;
```

Use: Retention dashboards and lifecycle studies.

Key Interview Takeaways Metric Formula

SQL Concept Business Meaning

1.CLV SUM(total_amount) Customer total value

2.AOV SUM / COUNT Average per order

3.MoM Growth LAG() Trend over time Ranking

4.RANK() OVER() Top customers Segmentation

5.CASE WHEN Categorize customers Churn

6.DATEDIFF() Inactive detection Contribution

7.SUM() OVER() % of total revenue

Day 27 Practice Targets i)Build a query to get top 3 customers by CLV per country. Ii) Compute month-over-month growth for the last 6 months. Iii) Classify customers into high/medium/low value segments. Iii) Find each category's revenue contribution %.

Day 28

Advanced SQL Analytics: Ranking, Percentiles & Top-N Insights

Learning Objective After Day 28, you'll be able to:

i)Rank and segment records dynamically. ii)Identify top performers (Top-N queries).

iii)Calculate percentiles and quartiles. iv)Analyze distribution of sales, revenue, or performance.

Write job-ready SQL for analytics challenges (MySQL & PostgreSQL compatible).

Ranking Functions Overview Window ranking functions assign an order-based position to each row.

Functions:

Function Behavior Example

ROW_NUMBER() Assigns a unique number to each row. Useful for pagination.

RANK() Ranks rows; ties get the same rank but next rank skips numbers. Competition ranking (1,2,2,4...).
DENSE_RANK() Ranks rows without skipping ranks for ties. 1,2,2,3 pattern.

Example (MySQL/Postgres)

SELECT customer_id, total_amount,

RANK() OVER (ORDER BY total_amount DESC) AS rank_position,

DENSE_RANK() OVER (ORDER BY total_amount DESC) AS dense_rank,

ROW_NUMBER() OVER (ORDER BY total_amount DESC) AS row_num

FROM orders;

Explanation:

OVER(ORDER BY total_amount DESC) → defines order of ranking.

RANK() → gives same rank for equal total_amount but leaves a gap.

DENSE_RANK() → same rank for equal total_amount without gaps.

ROW_NUMBER() → unique index regardless of duplicates.

Partitioned Ranking (RANK() with PARTITION BY) We can calculate ranks within each category, not the whole dataset.

```
SELECT c.country, o.customer_id,

SUM(o.total_amount) AS total_spent,

RANK() OVER (PARTITION BY c.country ORDER BY SUM(o.total_amount) DESC) AS rank_in_country

FROM orders o

JOIN customers c ON o.customer_id = c.customer_id

 GROUP BY c.country, o.customer_id;
```

Use Case: "Top 3 customers by spending in each country."

Top-N Analysis (Per Group): To get only the Top 3 per category or region:

```
SELECT * FROM (

SELECT p.category, p.product_name,

SUM(oi.quantity * p.price) AS revenue,

RANK() OVER (PARTITION BY p.category ORDER BY SUM(oi.quantity * p.price) DESC) AS rnk

FROM products p

JOIN order_items oi ON p.product_id = oi.product_id

GROUP BY p.category, p.product_name )

ranked WHERE rnk <= 3;
```

Explained:Rank products within category by revenue.

Filter rnk <= 3 → top 3 per category.

Perfect for sales leaderboard or KPI dashboards.

NTILE() – Data Segmentation NTILE(n) divides ordered data into n equal groups (quantiles or quartiles).

Example:

```
SELECT customer_id, total_amount,

NTILE(4) OVER (ORDER BY total_amount DESC) AS quartile

FROM orders;
```

Result Meaning:

Quartile = 1 → Top 25% high spenders

Quartile = 4 → Bottom 25% customers

Use Case: *Segmentation in marketing or customer analytics (RFM, churn models).*

Percentile Calculations:(PostgreSQL Only) For deeper statistical insights, **use:i)PERCENT_RANK(),ii)CUME_DIST()**

```
SELECT product_id,

SUM(oi.quantity * p.price) AS total_revenue,

PERCENT_RANK() OVER (ORDER BY SUM(oi.quantity * p.price)) AS percentile_rank,

CUME_DIST() OVER (ORDER BY SUM(oi.quantity * p.price)) AS cumulative_dist

FROM products p

JOIN order_items oi ON p.product_id = oi.product_id

GROUP BY product_id;
```

Difference:Function Description

PERCENT_RANK() Rank position as a percentile between 0 and 1

CUME_DIST() Fraction of rows ≤ current value

Combining Ranking + Aggregation

Example: Top 2 categories by revenue contribution

```
WITH category_revenue AS (

SELECT category,

SUM(oi.quantity * p.price) AS total_revenue

FROM products p

JOIN order_items oi ON p.product_id = oi.product_id

GROUP BY category )

SELECT category, total_revenue,

RANK() OVER (ORDER BY total_revenue DESC) AS rnk

FROM category_revenue

WHERE rnk <= 2;
```

Handling Ties: Function Tie Behavior Example

RANK() Ties share same rank, skips next 1, 2, 2, 4

DENSE_RANK() Ties share same rank, no skip 1, 2, 2, 3

ROW_NUMBER() Ignores ties 1, 2, 3, 4

Real-World Business Use Cases Scenario

SQL Concept

Find top 3 performing stores per region

RANK() + PARTITION BY region Identify top 10% spenders

PERCENT_RANK() or CUME_DIST() Segment customers into quartiles

NTILE(4) Reward top 3 sales reps

DENSE_RANK() Track sales rank changes monthly LAG() + RANK()

Tips for Interview-Ready Queries Always use CTE (WITH) for readability in Top-N problems.

Use RANK() OVER (PARTITION BY …) instead of subqueries for better performance.

Prefer DENSE_RANK() when you want consecutive numbering without gaps**.**

Remember that MySQL 8+ supports all window functions; older versions don't.

Practice on HackerRank SQL (Advanced) and DataLemur Ranking Challenges.

Day 29

Advanced Ranking & Percentile Analytics (MySQL)

Objective Learn advanced ranking techniques using **PERCENT_RANK(), CUME_DIST(),** and **NTILE()** functions to perform percentile-based segmentation, ranking, and performance analysis ,essential for business intelligence.

1. PERCENT_RANK() Function

Definition: Calculates the relative rank of a row as a percentage within a result set.

Formula: Percent Rank=Rank – 1/ Total Rows − 1

Purpose:

i)Identify top-performing customers/products.

ii)Compare relative performance (e.g., top 10%, bottom 5%).

Example:

SELECT customer_id, total_amount,

ROUND(PERCENT_RANK() OVER (ORDER BY total_amount DESC), 2) AS pct_rank

FROM orders;

Result: Each customer gets a rank between 0 and 1 ,showing where they stand compared to others.

CUME_DIST() Function

Definition: Returns the cumulative distribution of a value . i.e., how far through the dataset a particular value lies.

Difference from PERCENT_RANK():CUME_DIST() includes the current row in calculation.

The highest value always = 1.0.

Example:

SELECT customer_id, total_amount,

ROUND(CUME_DIST() OVER (ORDER BY total_amount DESC), 2) AS cum_dist

FROM orders;

Use Case: Find what percentage of customers spent less than or equal to a given customer.

NTILE() Function (Percentile or Decile Groups)

Definition: Divides the ordered dataset into equal-sized buckets (e.g., quartiles, deciles, or percentiles).

Example (Percentile segmentation):

SELECT customer_id, total_amount,

NTILE(100) OVER (ORDER BY total_amount DESC) AS percentile_group

FROM orders;

Meaning: Customers with percentile_group = 1 are top 1% spenders, while 100 are bottom 1%.

Combining RANK + Percentile Segmentation

Example: Identify top 10% of customers by total spending.

WITH ranked AS (

SELECT customer_id,

SUM(total_amount) AS total_spent,

ROUND(PERCENT_RANK() OVER (ORDER BY SUM(total_amount) DESC), 2) AS pct_rank

FROM orders GROUP BY customer_id)

*SELECT * FROM ranked WHERE pct_rank >= 0.9;*

Output: Shows only customers in the top 10% by spending.

Business Use Cases

Objective SQL Function 💬 Description Identify top 10% spenders

PERCENT_RANK() Rank-based segmentation Find bottom 20% performing products

NTILE(5) Divides into 5 equal performance tiers Loyalty program tiers

NTILE(100) Creates percentile-based tiers Revenue distribution analysis

CUME_DIST() Shows cumulative pattern of total spend

Key Differences Summary Function Purpose Range Includes Current Row?

RANK() Position in order Integer Yes PERCENT_RANK() Relative rank (0–1) 0–1 No CUME_DIST() Cumulative distribution 0–1 . Yes NTILE(n) Divide into n buckets 1–n Yes

Practical Insights

i)Analyze top and bottom performers in business.

ii)Build customer percentile dashboards (Power BI / Tableau).

iii)Segment customers by loyalty or revenue contribution.

iv) Prepare data for statistical or predictive modeling.

Day 30

Advanced SQL Analytics for Real-World Business Problems

Objective By the end of Day 30, you'll be able to: Combine window functions, joins, CTEs, and conditional aggregation for complex queries.Analyze revenue trends, customer segments, and top-performing products**.**

Solve job-ready SQL problems like HackerRank, LeetCode, and DataLemur challenges. Perform business metrics analysis with SQL (AOV, CLV, segment percentages).

Complex Aggregation + Ranking Combine GROUP BY, window functions, and ranking for top-N insights.

Example: Top products by revenue per category with rank.

```
SELECT category, product_name,
SUM(quantity * price) AS total_revenue,
RANK() OVER (PARTITION BY category ORDER BY SUM(quantity * price) DESC) AS rank_in_category
FROM products p
JOIN order_items oi ON p.product_id = oi.product_id
GROUP BY category, product_name;
```

Use Cases:

1.Top 3 products per category

2.Top 5 customers per country

Advanced Percentile Analysis

Use PERCENT_RANK(), CUME_DIST(), and NTILE() to segment data.

Example: Top 10% customers by total spending

```
WITH customer_rank AS (
SELECT customer_id,
SUM(total_amount) AS total_spent,
PERCENT_RANK() OVER (ORDER BY SUM(total_amount) DESC) AS pct_rank
FROM orders
GROUP BY customer_id )
SELECT * FROM customer_rank WHERE pct_rank >= 0.9;
```

NTILE Example: Customer deciles (top 20% → decile 1)

```
SELECT customer_id,
SUM(total_amount) AS total_spent,
NTILE(5) OVER (ORDER BY SUM(total_amount) DESC) AS decile
FROM orders
GROUP BY customer_id;
```

Month-over-Month & Year-over-Year Trends Calculate growth %, running total, and rolling average for revenue.

Example: Month-over-month revenue growth

```
WITH monthly_revenue AS (
SELECT
DATE_FORMAT(order_date,'%Y-%m') AS month,
SUM(total_amount) AS revenue
FROM orders
GROUP BY month )
SELECT month, revenue,
LAG(revenue) OVER (ORDER BY month) AS previous_month,
ROUND((revenue - LAG(revenue) OVER (ORDER BY month)) / LAG(revenue) OVER (ORDER BY month) * 100,2) AS mom_growth_pct
FROM monthly_revenue;
```

Use Cases:

i)Track business growth

ii)Compare performance YoY or MoM

Customer Segmentation & Cohort Analysis: Segment customers using spending, orders, or percentile-based tiers.Combine NTILE(), RANK(), conditional aggregation for cohorts.

Example: Top 3 customers per country

```
WITH customer_country AS (
SELECT c.country, c.customer_id,
SUM(o.total_amount) AS total_spent,
RANK() OVER (PARTITION BY c.country ORDER BY SUM(o.total_amount) DESC) AS rank_in_country
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.country, c.customer_id )
SELECT * FROM customer_country
WHERE rank_in_country <= 3;
```

Conditional Aggregation & CASE Statements Use CASE WHEN to segment data dynamically.

Example: Order Value Segmentation

```
SELECT

COUNT(*) AS total_orders,

SUM(CASE WHEN total_amount > 1000 THEN 1 ELSE 0 END) AS high_value_orders,

SUM(CASE WHEN total_amount BETWEEN 500 AND 1000 THEN 1 ELSE 0 END) AS medium_value_orders,
SUM(CASE WHEN total_amount < 500 THEN 1 ELSE 0 END) AS low_value_orders

FROM orders;
```

Combining CTE + Subquery + Window Functions Solve multi-step business questions efficiently.

Example: Top 3 products per category + % contribution

```
WITH product_revenue AS (

SELECT category, product_name,

SUM(quantity * price) AS revenue

FROM products p

JOIN order_items oi ON p.product_id = oi.product_id

GROUP BY category, product_name )

SELECT category, product_name, revenue,

RANK() OVER (PARTITION BY category ORDER BY revenue DESC) AS rank_in_category,

ROUND(revenue / SUM(revenue) OVER (PARTITION BY category) * 100, 2) AS pct_of_category

FROM product_revenue

WHERE rank_in_category <= 3;
```

Real-World Business Metrics Metric SQL Approach

Average Order Value (AOV)= SUM(total_amount)/COUNT(*)

Customer Lifetime Value (CLV)= SUM(total_amount) OVER (PARTITION BY customer_id)

Top products by revenue SUM(quantity * price) + RANK() Revenue % contribution per segment SUM(...) / SUM(...) OVER()

Interview Questions and Answers

Top 100 SQL Interview Questions & Answers (Bangla + English Edition)

Section 1: SQL Basics & Filtering

1.Q: What is SQL?

A: SQL stands for Structured Query Language, used to manage and analyze data in relational databases.

বাংলা: SQL অর্থ Structured Query Language, যা relational database-এর ডেটা ব্যবস্থাপনা ও বিশ্লেষণে ব্যবহৃত হয়।

2.Q: What is the difference between a database and a table?

A: A database stores multiple tables, while a table stores data in rows and columns.

বাংলা: Database হলো একাধিক টেবিলের সমষ্টি, আর Table হলো সারি (row) ও কলাম (column) আকারে ডেটা রাখার জায়গা।

3.Q: What is a primary key?

A: A primary key uniquely identifies each row in a table.

বাংলা: Primary Key প্রতিটি রেকর্ডকে আলাদা করে চিহ্নিত করে।

4.Q: What is a foreign key?

A: A foreign key establishes a relationship between two tables.

বাংলা: Foreign Key দুইটি টেবিলের মধ্যে সম্পর্ক তৈরি করে।

5.Q: Difference between WHERE and HAVING clause?

A: WHERE filters before aggregation, HAVING filters after aggregation.

বাংলা: WHERE aggregation-এর আগে ফিল্টার করে, HAVING পরে।

6.Q: What is the difference between DISTINCT and GROUP BY?

A: DISTINCT removes duplicates, GROUP BY groups and aggregates data.

বাংলা: DISTINCT ডুপ্লিকেট সরায়, GROUP BY ডেটা গ্রুপ করে।

7.Q: What does ORDER BY do?

A: It sorts the result set in ascending or descending order.

বাংলা: ORDER BY ফলাফলকে ক্রমানুসারে সাজায়।

8.Q: What is the LIMIT clause used for?

A: To restrict the number of rows returned.

বাংলা: LIMIT দিয়ে নির্দিষ্ট সংখ্যক রেকর্ড ফেরানো হয়।

9.Q: What is the use of BETWEEN operator?

A: To filter values within a range.

বাংলা: BETWEEN একটি নির্দিষ্ট সীমার মধ্যে মান নির্বাচন করে।

10. Q: What is the use of LIKE operator?

A: It’s used for pattern matching with wildcards (% or _).

বাংলা: LIKE ব্যবহার হয় pattern match করার জন্য, যেমন % বা _ দিয়ে।

Section 2: Aggregate Functions & Grouping

11.Q: What are aggregate functions in SQL?

A: Functions that return a single value after performing a calculation on a set of values (SUM, AVG, COUNT, etc.).
বাংলা: Aggregate Function একাধিক মানের উপর গণনা চালিয়ে একটি মান দেয় (যেমন SUM, AVG, COUNT)।

12. Q: What is COUNT(*)?*

A: It counts all rows in a table, including NULLs.

বাংলা: COUNT() সব রেকর্ড গুনে, এমনকি NULL-ও।

13.Q: What is COUNT(column_name)?

A: It counts non-NULL values in a specific column.

বাংলা: নির্দিষ্ট কলামে যেসব মান NULL নয় সেগুলো গুনে।

14. Q: What is the use of GROUP BY with aggregates?

A: It groups rows and applies aggregate functions per group.

বাংলা: GROUP BY প্রতিটি গ্রুপে aggregate function চালায়।

15. Q: How can you filter groups in SQL?

A: By using the HAVING clause.

বাংলা: HAVING clause ব্যবহার করে গ্রুপ ফিল্টার করা যায়।

16.Q: How to find total revenue per category?

SELECT category, SUM(price)AS total_revenue

FROM products

GROUP BY category;

বাংলা: প্রতিটি category অনুযায়ী মোট রাজস্ব বের করতে SUM() এবং GROUP BY ব্যবহার হয়।

17.Q: How to calculate average order value (AOV)?

SELECT customer_id, AVG(total_amount) AS avg_order_value

FROM orders

GROUP BY customer_id;

বাংলা: AVG() দিয়ে প্রতিটি কাস্টমারের গড় অর্ডার মূল্য বের করা হয়।

18. Q: What is the difference between SUM() and COUNT()?

A: SUM() adds numeric values; COUNT() counts records.

বাংলা: SUM() যোগফল বের করে, COUNT() রেকর্ড সংখ্যা গণনা করে।

19. Q: What is the use of ROUND() function?

A: To round numeric results to a specific decimal place.

বাংলা: ROUND() কোনো সংখ্যা নির্দিষ্ট দশমিক পর্যন্ত রাউন্ড করে।

20.Q: How can you calculate percentage contribution?

SELECT category,

SUM(sales) AS total_sales,

*100 * SUM(sales)/SUM(SUM(sales)) OVER() AS percentage*

FROM sales

GROUP BY category;

বাংলা: Window function দিয়ে শতকরা অবদান হিসাব করা যায়।

Section 3: SQL Joins (10 Interview Questions + Answers) (Bangla + English)

21.Q: What is a JOIN in SQL?

A: A JOIN is used to combine rows from two or more tables based on a related column.

বাংলা: JOIN ব্যবহার হয় একাধিক টেবিলের ডেটা একসাথে আনতে, কোনো সম্পর্কিত কলামের ভিত্তিতে।

22. Q: What are the types of SQL JOINs?

A: INNER, LEFT, RIGHT, FULL, CROSS।

23.Q: What does INNER JOIN do?

A: Returns only matching records from both tables.

বাংলা: INNER JOIN দুই টেবিলের মিল থাকা রেকর্ডগুলো ফেরায়।

24.Q: What is LEFT JOIN?

A: Returns all rows from the left table and matching rows from the right.

বাংলা: LEFT JOIN বাম টেবিলের সব ডেটা এবং ডান টেবিলের মিল থাকা ডেটা দেয়।

25.Q: What is RIGHT JOIN?

A: Returns all rows from the right table and matching rows from the left.

বাংলা: RIGHT JOIN ডান টেবিলের সব ডেটা এবং বাম টেবিলের মিল থাকা ডেটা দেয়।

26. Q: What is FULL OUTER JOIN?

A: Returns all records from both tables, matching where possible.

বাংলা: FULL OUTER JOIN দুই টেবিলের সব ডেটা ফেরায়, যেখানে মিল আছে সেটাও দেখায়।

27. Q: What is CROSS JOIN?

A: Produces all possible combinations of rows between two tables (Cartesian product).

বাংলা: CROSS JOIN দুই টেবিলের সব সম্ভাব্য কম্বিনেশন তৈরি করে।

28.Q: What is a SELF JOIN?

A: A table joined with itself to compare rows within the same table.

বাংলা: SELF JOIN হলো একই টেবিলের মধ্যে তুলনা করার জন্য নিজের সাথেই JOIN করা।

29.Q: How to join three tables?

sql Copy code SELECT s.name, c.course_name, e.grade FROM students s JOIN enrollments e ON s.student_id = e.student_id JOIN courses c ON e.course_id = c.course_id; বাংলা: একাধিক টেবিল JOIN করার সময় একটার পর একটা JOIN ব্যবহার হয়।

30.Q: What is the difference between INNER JOIN and LEFT JOIN?

A: INNER JOIN returns only matching rows; LEFT JOIN includes unmatched rows from left.

বাংলা: INNER JOIN কেবল মিল থাকা রেকর্ড ফেরায়, LEFT JOIN বাম টেবিলের সব রেকর্ড রাখে।

Section 4: Subqueries & CTEs (10 Interview Questions + Answers)

31.Q: What is a subquery?

A: A query inside another query.

বাংলা: Subquery মানে এক কুয়েরির ভেতরে আরেকটা কুয়েরি।

32.Q: What are the types of subqueries?

A:i)Single-row subquery, ii)Multi-row subquery,iii)Correlated subquery

বাংলা: Subquery তিন রকম — single-row, multi-row, correlated।

33.Q: What is a correlated subquery?

A: It refers to columns from the outer query.

বাংলা: Correlated subquery বাইরের কুয়েরির কলামের ওপর নির্ভর করে।

34.Q: Example of subquery in WHERE clause?

SELECT name FROM students

WHERE department = (SELECT department FROM students WHERE name='Alice');

বাংলা: Subquery সাধারণত WHERE clause-এ ব্যবহৃত হয়।

35.Q: What is a CTE (Common Table Expression)?

A: A temporary result set defined using WITH clause.

বাংলা: CTE হলো WITH দিয়ে তৈরি অস্থায়ী টেবিল, যা পরে কুয়েরিতে ব্যবহার হয়।

35.Q: Example of a simple CTE?

A. *WITH high_sales AS (*

SELECT customer_id,

SUM(total_amount) AS total

FROM orders

GROUP BY customer_id)

*SELECT * FROM high_sales WHERE total > 1000;*

বাংলা: WITH ব্যবহার করে সাময়িক ফলাফল তৈরি করে পরে তা থেকে ডেটা নেওয়া হয়।

37.Q: Difference between CTE and Subquery?

A: CTE is more readable and reusable; subquery is nested and less readable.

বাংলা: CTE পড়তে ও ব্যবহার করতে সহজ, Subquery nested ও জটিল হয়।

38.Q: What is a Recursive CTE?

A: A CTE that refers to itself to process hierarchical data.

বাংলা: Recursive CTE এমন CTE যা নিজেকেই রেফার করে, যেমন hierarchy data তে ব্যবহৃত হয়।

39.Q: Example of recursive CTE (employee hierarchy)?

WITH RECURSIVE emp_hierarchy AS (

SELECT employee_id, name, manager_id

FROM employees

WHERE manager_id IS NULL

UNION ALL

SELECT e.employee_id, e.name, e.manager_id

FROM employees e

JOIN emp_hierarchy h ON e.manager_id = h.employee_id)

*SELECT * FROM emp_hierarchy;*

বাংলা: Recursive CTE দিয়ে manager-employee সম্পর্ক খুঁজে বের করা যায়।

40.Q: When should you use CTE instead of subquery?

A: When the logic is complex or reused multiple times.

বাংলা: যখন কুয়েরি জটিল বা বারবার ব্যবহার করতে হবে, তখন CTE ভালো।

Section 5: Window Functions (10 SQL Interview Questions + Answers) (Questions 41–50)

41.Q: What is a window function?

A: A function that performs calculations across a set of rows related to the current row, without grouping them.
বাংলা: Window function বর্তমান রেকর্ডের সাথে সম্পর্কিত অন্য রেকর্ডের উপর হিসাব চালায়, কিন্তু GROUP BY এর মতো ডেটা একত্র করে না।

42.Q: What clause is used with window functions?

A: The OVER() clause.

বাংলা: Window function ব্যবহারের সময় OVER() clause ব্যবহার হয়।

43.Q: Example of ROW_NUMBER()?

SELECT name,

ROW_NUMBER() OVER (ORDER BY total_amount DESC) AS rank_position

FROM customers;

বাংলা: ROW_NUMBER() প্রতিটি রেকর্ডকে ক্রমিকভাবে নম্বর দেয়।

44.Q: Difference between RANK() and DENSE_RANK()?

A:RANK() skips ranks for ties,DENSE_RANK() does not skip ranks

বাংলা: RANK() সমান মানে rank বাদ দেয়, DENSE_RANK() বাদ দেয় না।

45.Q: What does NTILE() do?

A: Divides rows into equal groups or buckets (e.g., quartiles, deciles).

বাংলা: NTILE() ডেটাকে সমান ভাগে ভাগ করে (যেমন ৫ ভাগ মানে ২০% করে গ্রুপ)।

46.Q: What does PARTITION BY do in window functions?

A: Divides data into groups before applying the window calculation.

বাংলা: PARTITION BY ব্যবহার করে ডেটাকে গ্রুপে ভাগ করা হয়, তারপরে window function প্রয়োগ হয়।

47.Q: How does ORDER BY work inside OVER()?

A: Determines the order of rows for window function calculations.

বাংলা: ORDER BY ঠিক করে দেয় কোন ক্রমে হিসাব চলবে window function-এ।

48. Q: Example of a running total using window function?

SELECT order_date,

SUM(total_amount) OVER (ORDER BY order_date) AS running_total

FROM orders;

বাংলা: SUM() OVER() ব্যবহার করে ধারাবাহিক যোগফল (running total) বের করা যায়।

49.Q: How to calculate rolling average of last 3 rows?

```
SELECT order_date,

AVG(total_amount) OVER (ORDER BY order_date ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS
rolling_avg

FROM orders;
```

বাংলা: এই কুয়েরি আগের দুই মাস ও বর্তমান মাসের গড় বের করে।

50.Q: Difference between GROUP BY and Window Function?

A: GROUP BY aggregates and reduces rows; window functions calculate without reducing rows.

বাংলা: GROUP BY রেকর্ড কমিয়ে দেয়, window function সব রেকর্ড রেখে হিসাব চালায়।

Section 6: Aggregate Functions & Conditional Logic (10 SQL Interview Questions + Answers)

51.Q: What are aggregate functions in SQL?

A: Functions that perform a calculation on multiple values and return a single result (SUM, AVG, COUNT, MAX, MIN).

বাংলা: Aggregate function একাধিক রেকর্ড থেকে একটি ফলাফল দেয়, যেমন SUM(), AVG()।

52.Q: Difference between COUNT(*) and COUNT(column)?*

*A: COUNT(*) counts all rows; COUNT(column) ignores NULL values.

বাংলা: COUNT(*) সব রেকর্ড গোনে, কিন্তু COUNT(column) NULL বাদ দেয়।

53.Q: What does AVG() do?

A: Calculates the average of numeric values.

বাংলা: AVG() সংখ্যাগুলোর গড় বের করে।

54.Q: What is the use of SUM()?

A: Adds up numeric column values.

বাংলা: SUM() সংখ্যার যোগফল দেয়।

55.Q: What is MIN() and MAX()?

A: MIN() returns smallest value, MAX() returns largest value.

বাংলা: MIN() সর্বনিম্ন মান আর MAX() সর্বোচ্চ মান দেয়।

56.Q: What is the purpose of the CASE WHEN statement?

A: Used for conditional logic inside SQL queries.

বাংলা: CASE WHEN শর্ত নির্ভর মান নির্ধারণে ব্যবহৃত হয়।

57.Q: Example of CASE WHEN for segmentation?

```
SELECT order_id,

CASE

WHEN total_amount > 1000 THEN 'High'

WHEN total_amount BETWEEN 500 AND 1000 THEN 'Medium'

ELSE 'Low' END AS order_segment

FROM orders;
```

বাংলা: CASE WHEN দিয়ে অর্ডারগুলো High, Medium, Low হিসেবে ভাগ করা হয়।

58.Q: How to count conditional data using CASE WHEN?

SELECT country,

SUM(CASE WHEN total_amount > 500 THEN 1 ELSE 0 END) AS high_value_orders

FROM orders

GROUP BY country;

বাংলা: SUM + CASE WHEN ব্যবহার করে শর্ত অনুযায়ী রেকর্ড গোনা যায়।

59.Q: What is COALESCE()?

A: Returns the first non-NULL value in a list.

বাংলা: COALESCE() প্রথম non-NULL মানটি ফেরায়।

60.Q: What is NULLIF()?

A: Returns NULL if two expressions are equal; otherwise returns the first expression.

বাংলা: NULLIF() দুটি মান সমান হলে NULL দেয়, না হলে প্রথম মান ফেরায়।

Section 7: Date, String & Mathematical Functions (Questions 61–70)

61.Q: How to extract the year from a date in SQL?

SELECT

EXTRACT(YEAR FROM order_date) AS year

FROM orders;

বাংলা: EXTRACT(YEAR FROM order_date) দিয়ে তারিখ থেকে বছর আলাদা করা যায়।

62.Q: How to get the month name from a date?

SELECT

MONTHNAME(order_date) AS month_name

FROM orders;

বাংলা: MONTHNAME() ফাংশন তারিখ থেকে মাসের নাম দেয় (যেমন January, February)।

63.Q: How to find the difference between two dates?

SELECT

DATEDIFF(delivery_date, order_date) AS days_difference

FROM orders;

বাংলা: DATEDIFF() দুটি তারিখের মধ্যে দিনের ব্যবধান বের করে।

64.Q: How to add or subtract days from a date?

SELECT

DATE_ADD(order_date, INTERVAL 7 DAY) AS next_week

FROM orders;

বাংলা: DATE_ADD() দিয়ে তারিখে দিন যোগ বা বাদ দেওয়া যায়।

65.Q: How to get the current date and time in SQL?

SELECT

CURRENT_DATE(),

CURRENT_TIME(),

NOW();

বাংলা: CURRENT_DATE(), CURRENT_TIME(), এবং NOW() ব্যবহার করে বর্তমান তারিখ ও সময় দেখা যায়।

66.Q: How to convert string to uppercase or lowercase?

SELECT

UPPER(customer_name),

LOWER(city) FROM customers;

বাংলা: UPPER() বড় হাতের অক্ষরে এবং LOWER() ছোট হাতের অক্ষরে রূপান্তর করে।

67.Q: How to combine two strings in SQL?

SELECT

CONCAT(first_name, ' ', last_name) AS full_name

FROM customers;

বাংলা: CONCAT() দিয়ে দুটি স্ট্রিং একত্রে যোগ করা যায়।

68.Q: How to find length of a string?

SELECT LENGTH(customer_name) FROM customers;

বাংলা: LENGTH() দিয়ে কোনো টেক্সটের অক্ষরের সংখ্যা জানা যায়।

69.Q: What is the purpose of SUBSTRING()?

SELECT SUBSTRING(customer_name, 1, 4) FROM customers;

বাংলা: SUBSTRING() স্ট্রিংয়ের নির্দিষ্ট অংশ আলাদা করে। যেমন “Tanvir” → “Tanv”।

70.Q: Example of mathematical functions in SQL?

SELECT ROUND(price, 2), CEIL(price), FLOOR(price) FROM products;

বাংলা:

ROUND() মানকে নির্দিষ্ট দশমিক পর্যন্ত রাউন্ড করে

CEIL() উপরের পূর্ণসংখ্যায় নেয়

FLOOR() নিচের পূর্ণসংখ্যায় নেয়

Section 8: Grouping, Filtering & HAVING Clause (Questions 71–80)

71.Q: What does GROUP BY do?

A: Groups rows that have the same values into summary rows.

বাংলা: GROUP BY এক বা একাধিক কলামের মান অনুযায়ী ডেটা গ্রুপ করে সারাংশ দেয়।

72.Q: Can we use WHERE and GROUP BY together?

A: Yes. WHERE filters rows before grouping.

বাংলা: WHERE clause দিয়ে গ্রুপ করার আগে রেকর্ড ফিল্টার করা যায়।

73.Q: What is HAVING clause used for?

A: Used to filter groups after aggregation.

বাংলা: HAVING clause গ্রুপ করা ফলাফলের পর শর্ত প্রয়োগে ব্যবহৃত হয়।

74.Q: Difference between WHERE and HAVING?

A. Clause Used On Filters WHERE Before GROUP BY Individual rows HAVING After GROUP BY Aggregated data

বাংলা: WHERE রেকর্ড ফিল্টার করে, HAVING ফিল্টার করে group করা ডেটা।

75.Q: Example of HAVING with SUM()?

SELECT customer_id,

SUM(total_amount) AS total_spent

FROM orders

GROUP BY customer_id

HAVING total_spent > 1000;

বাংলা: গ্রাহক অনুযায়ী মোট কেনাকাটা হিসাব করে, যার ১০০০ টাকার বেশি তারা ফিল্টার হয়।

76.Q: Can we use ORDER BY with GROUP BY?

A: Yes, ORDER BY sorts the grouped results.

বাংলা: GROUP BY এর পরে ORDER BY দিয়ে সাজানো যায়।

77.Q: How to count customers per country?

SELECT country,

COUNT(customer_id) AS total_customers

FROM customers

GROUP BY country;

বাংলা: COUNTRY অনুযায়ী মোট গ্রাহক সংখ্যা গণনা করা হয়।

78.Q: How to find average revenue per country with condition?

SELECT country,

AVG(total_amount) AS avg_revenue

FROM orders

GROUP BY country

HAVING avg_revenue > 500;

বাংলা: দেশের গড় রেভিনিউ ৫০০ টাকার বেশি এমন দেশগুলো ফিল্টার করা হয়।

79.Q: How to get number of orders per month?

SELECT

DATE_FORMAT(order_date, '%Y-%m') AS month,

COUNT(order_id) AS total_orders

FROM orders

GROUP BY month

ORDER BY month;

বাংলা: প্রতি মাসে কতগুলো অর্ডার হয়েছে তা দেখায়।

80.Q: Can we use multiple columns in GROUP BY?

A: Yes. Example:

SELECT country, city,

COUNT(*) AS total_customers

FROM customers

GROUP BY country, city;

বাংলা: একাধিক কলাম দিয়ে গ্রুপ করা যায় — যেমন দেশ ও শহর অনুসারে।

Section 9: Subqueries & CTEs (Questions 81–90)

81.Q: What is a subquery in SQL?

A subquery is a query inside another query.

*SELECT * FROM customers*

WHERE customer_id IN

(SELECT customer_id FROM orders WHERE total_amount > 500);

বাংলা: Subquery মানে মূল query-র ভিতরে আরেকটা query ব্যবহার করা হয়।

82.Q: Types of subqueries?

i)Single-row subquery,ii)Multi-row subquery,iii)Correlated subquery

83.Q: What is a correlated subquery?

SELECT c.name FROM customers c WHERE EXISTS

(SELECT 1 FROM orders o WHERE o.customer_id = c.customer_id AND o.total_amount > 1000);

বাংলা: এখানে subquery প্রতিটি customer-এর সাথে মিলিয়ে বারবার চলে — একে correlated বলে।

84.Q: Find customers who have never placed an order.

SELECT name FROM customers WHERE customer_id NOT IN (SELECT customer_id FROM orders);

বাংলা: যাদের order table-এ কোনো রেকর্ড নেই, তারা "never ordered" গ্রাহক।

85.Q: Find the highest-priced product using subquery.

SELECT * FROM products

WHERE price = (SELECT MAX(price) FROM products);

বাংলা: MAX() subquery-র মাধ্যমে সর্বোচ্চ দামের প্রোডাক্ট বের করা হয়।

86.Q: Difference between subquery and join?

A:Feature Subquery Join Performance Usually slower Faster for large data Readability Easier Sometimes complex Use case Filtering or nested logic Combining tables

বাংলা: Subquery nested filtering-এ ভালো, আর join বড় টেবিল merge-এ দ্রুত।

87.Q: What is a CTE (Common Table Expression)?

```
WITH high_spenders AS (
SELECT customer_id,
SUM(total_amount) AS total_spent
FROM orders
GROUP BY customer_id
HAVING total_spent > 1000 )
SELECT * FROM high_spenders;
```

বাংলা: CTE মানে অস্থায়ী নাম দেওয়া সাবকোয়েরি যা পরের query-তে ব্যবহার করা যায়।

88.Q: Difference between CTE and subquery?

A.CTE Subquery Reusable Not reusable Readable Nested, harder Can be recursive Cannot be recursive

বাংলা: CTE বেশি readable এবং একাধিকবার ব্যবহার করা যায়।

89.Q: What is a recursive CTE?

```
WITH RECURSIVE numbers AS (
SELECT 1 AS n
UNION ALL
SELECT n + 1
FROM numbers
WHERE n < 5 )
SELECT * FROM numbers;
```

বাংলা: Recursive CTE নিজের মধ্যেই পুনরাবৃত্তি করে (১–৫ পর্যন্ত সংখ্যা তৈরি করবে)।

90.Q: When should you prefer CTEs over subqueries?

Answer:

i)When readability matters

ii)When the same result is used multiple times

iii)When performing hierarchical or recursive operations

বাংলা: Query যদি জটিল হয় বা একই সাবকোয়েরি বারবার দরকার হয়, তখন CTE ব্যবহার করা শ্রেয়।

Section 10: Joins & Advanced Analytics (Questions 91–100)

91.Q: What is a JOIN?

A.A JOIN combines rows from two or more tables based on a related column.

বাংলা: দুটি টেবিলের সম্পর্কিত ডেটা একত্রে দেখাতে JOIN ব্যবহৃত হয়।

92Q: Types of JOINs in SQL?

A.INNER JOIN,LEFT JOIN,RIGHT JOIN,FULL JOIN (or FULL OUTER JOIN),CROSS JOIN

বাংলা: পাঁচ রকম join আছে—সবচেয়ে সাধারণ INNER JOIN।

93.Q: Show all customers and their orders (even if no order).

SELECT c.name, o.order_id, o.total_amount

FROM customers c

LEFT JOIN orders o ON c.customer_id = o.customer_id;

বাংলা: LEFT JOIN সব গ্রাহক দেখায়, এমনকি যারা কোনো অর্ডার করেনি।

94.Q: Find revenue per product category using JOIN.

SELECT p.category,

*SUM(oi.quantity * p.price) AS total_revenue*

FROM order_items oi

JOIN products p ON oi.product_id = p.product_id

GROUP BY p.category;

বাংলা: product table এর দাম × quantity দিয়ে category অনুযায়ী মোট রাজস্ব হিসাব করা হয়।

95.Q: Find customers who bought a product in the 'Electronics' category.

SELECT DISTINCT c.name

FROM customers c

JOIN orders o ON c.customer_id = o.customer_id

JOIN order_items oi ON o.order_id = oi.order_id

JOIN products p ON oi.product_id = p.product_id

WHERE p.category = 'Electronics';

বাংলা: JOIN দিয়ে customer-দের বের করা হয় যারা Electronics কিনেছে।

96.Q: What is a self-join?

SELECT e1.name AS employee,

e2.name AS manager

FROM employees e1

JOIN employees e2 ON e1.manager_id = e2.employee_id;

বাংলা: একই টেবিলকে নিজের সাথে join করলেই self-join।

97.Q: What is a cross join?

SELECT * FROM products CROSS JOIN customers;

বাংলা: দুটি টেবিলের প্রতিটি রেকর্ডের সবসম্ভাব্য কম্বিনেশন দেখায়।

98.Q: Use of window functions in analytics.

SELECT customer_id,

SUM(total_amount) OVER (PARTITION BY customer_id) AS lifetime_value,

RANK() OVER (ORDER BY SUM(total_amount) DESC) AS rank_position

FROM orders

GROUP BY customer_id;

বাংলা: Window function গ্রাহক অনুযায়ী lifetime value ও rank বের করে।

99.Q: What is the difference between RANK(), DENSE_RANK(), and ROW_NUMBER()?

A.Function Output Example RANK() 1, 2, 2, 4 DENSE_RANK() 1, 2, 2, 3 ROW_NUMBER() 1, 2, 3, 4

বাংলা: RANK() স্কিপ করে, DENSE_RANK() স্কিপ করে না, ROW_NUMBER() সবসময় ক্রমানুসারে চলে।

100.Q: Explain the concept of running total and moving average.

SELECT order_date,

SUM(total_amount) OVER (ORDER BY order_date) AS running_total,

AVG(total_amount) OVER (ORDER BY order_date ROWS 2 PRECEDING) AS moving_avg

FROM orders;