Admissions & Support: support@mockattempt.com
Phone / WhatsApp: +91 95400 07917
Course Code: SQLD • **Updated:** 18 Nov 2026
https://mockattempt.com/courses/sql-postgresql-mysql-database-training

SQLD • RELATIONAL DATABASES & SQL ENGINEERING **BEGINNER TO JOB-READY**

100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER

Complete SQL, PostgreSQL & MySQL Database Engineering

Master production SQL across PostgreSQL and MySQL from foundational tables and joins to 3NF schema design, EXPLAIN ANALYZE index tuning, ACID isolation levels, Window Functions, Recursive CTEs, and Top 50 SQL interview queries.

Duration: 5 Weeks • Weekend Live (Sat–Sun • 2h/Day) (80 Hrs) • **Modules:** 5 • **Live Topics:** 43 • **Subtopics:** 344+ • **Labs & Tasks:** 86 • **Mocks / AI Vivas:** 43 / 43 • **Projects:** 5 Module + 1 Live Client Prod

All-Inclusive Direct Fee (Incl. 18% GST • Zero Hidden Charges): Total Payable: ₹1,499 (Net Tuition ₹1,270 + Included 18% GST ₹229) • **Split Option:** 100% Credit to Job Track • **Effective ROI:** ~₹19/Guided Hr (~₹35/Live Topic)

Faculty: 100% Active Working IT Professionals • **Weekend-Only Live Batch (7:00 AM – 8:00 PM IST Flexible Slots):** Weekend-Only Live Online Batch (Sat – Sun • 2 Hrs/Day • 7:00 AM – 8:00 PM) (Saturday & Sunday Only (2 Hours Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 2-Hour Weekend Slot Based on Working Professional Trainer Availability) • 10–12 Weeks Weekend Track)

✓ **Trainer LeetCode & HackerRank Mentorship (SQL, PostgreSQL & MySQL (LeetCode SQL 50 & HackerRank SQL)): Live trainer support to crack LeetCode & HackerRank problems, optimize complexity, and handle edge cases.**

Core Technologies & Engineering Stack: SQL (DDL/DML/DQL) • PostgreSQL 16 • MySQL 8 • Joins & Subqueries • ER Modeling & 3NF • Composite Indexing • EXPLAIN ANALYZE • ACID Transactions • Window Functions & CTEs

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Write clean, accurate DDL, DML, and complex multi-table INNER/LEFT/SELF join & subquery reports in PostgreSQL and MySQL.
2. Model normalized 1NF/2NF/3NF/BCNF enterprise schemas with PK, FK, UNIQUE, and CHECK data-integrity constraints.
3. Diagnose sequential table scans and tune query latency using composite B-Tree indexes and EXPLAIN ANALYZE.
4. Solve Top-N per group, running totals, YoY growth, and hierarchical org-chart queries using Window Functions and Recursive CTEs.
5. Crack LeetCode and HackerRank problems in SQL, PostgreSQL & MySQL (LeetCode SQL 50 & HackerRank SQL) with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
6. Automated Testing, Debugging & Quality Gates (SQL (DDL/DML/DQL), PostgreSQL 16, MySQL 8, Joins & Subqueries, ER Modeling & 3NF) — Automated SQL (DDL/DML/DQL), PostgreSQL 16, MySQL 8, Joins & Subqueries, ER Modeling & 3NF test suites • Linting & static checks • Mentor Git pull-request reviews — ensuring your code never breaks silently when real users hit edge cases.
7. 100% Topic-Wise Timed Mock Tests, Spoken MockAttempt AI Viva & Live Client Production — Verify every single Complete SQL, PostgreSQL & MySQL Database Engineering topic through timed MCQ exams and spoken technical interviews before deploying your Live Client Project.
8. Translated live client functional and non-functional requirements into modular SQL (DDL/DML/DQL), PostgreSQL 16, MySQL 8, Joins & Subqueries, ER Modeling & 3NF production architecture.
9. Implemented core application modules, input validation, security hardening, and optimized data/state workflows.
10. Completed peer Git pull-request code reviews, automated verification suites, and edge-case debugging sprints.
11. Deployed the verified build to live client production with monitoring, documentation, and mentor sign-off.
12. Clear 100% topic-wise Timed MCQ Mock Tests and Spoken MockAttempt AI Technical Interviews across every module of Complete SQL, PostgreSQL & MySQL Database Engineering.

Eligibility & Prerequisites: No prior SQL or database experience required — Week 1 begins from database fundamentals and PostgreSQL/MySQL setup. • Basic computer & internet familiarity on a Windows, macOS, or Linux laptop (8 GB+ RAM recommended for smooth live coding labs). • Ideal for B.Tech / BCA / MCA / B.Sc IT students, non-CS beginners, career switchers, and working developers upgrading to 2026 production standards. • Commitment to attend Weekend-Only live classes (Saturday & Sunday, 2 Hours/Day between 7:00 AM – 8:00 PM IST taught by Working IT Professionals), complete weekday real-repository checklists, and practice topic-wise Mock Tests & AI Interviews. • Dedication to write original code in your GitHub portfolio and complete the mentor-guided Live Client Production Sprint.

2. DETAILED ZERO-TO-EXPERT MODULE, TOPIC & SUBTOPIC SYLLABUS (5 MODULES • 43 LIVE TOPICS • 344+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–3: Month 1 – Core SQL, Joins, Normalization & Indexing (Weeks 1–3) (28 Topics / 224 Subtopics) • Modules 4–5: Month 2 – Transactions, Window Analytics & Enterprise SQL Capstone (Weeks 4–5) (15 Topics / 120 Subtopics)

MODULE 1 OF 5 • STAGE 1 • 10 LIVE + LAB HOURS • 10 TOPICS (80 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: Relational Database Fundamentals & Core SQL Queries

Build complete fluency in relational database fundamentals and SQL using PostgreSQL (with MySQL syntax comparisons) — DDL, DML, DQL, constraints, 1:1 / 1:N / M:N relationships, INNER/OUTER/SELF joins, scalar & correlated subqueries, aggregations, and set operations.

Objectives: ✓ Design relational tables in PostgreSQL with accurate data types and PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, and NOT NULL constraints. • ✓ Write safe DML (`INSERT`, `UPDATE`, `DELETE`) and expressive `SELECT` queries with filtering, pattern matching, and sorting. • ✓ Aggregate business metrics using `GROUP BY` and `HAVING` while respecting SQL logical execution order (`FROM` -> `WHERE` -> `GROUP BY` -> `HAVING` -> `SELECT` -> `ORDER BY`). • ✓ Model 1:1, 1:N, and M:N join tables and write INNER, LEFT, RIGHT, FULL OUTER, and SELF joins accurately. • ✓ Solve multi-step reporting questions using single-row, multi-row, correlated subqueries, and `UNION` / `INTERSECT` / `EXCEPT`.

1.1 Database, DBMS and RDBMS Introduction [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: File systems vs DBMS vs RDBMS • Codd’s relational model • PostgreSQL vs MySQL overview • Client-server connection lifecycle • Production Pitfall Guard (Database, DBMS and RDBMS Introduction) • Hands-On Verification (Database, DBMS and RDBMS Introduction) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Database

Purpose: Choosing PostgreSQL as the system-of-record store for financial, e-commerce, and HR backend systems.

What Topic Is Next → Next Topic 1.2: Relational Database Architecture

1.2 Relational Database Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Schemas • Tables (relations) • Rows (tuples) • Columns (attributes) • Production Pitfall Guard (Relational Database Architecture) • Hands-On Verification (Relational Database Architecture) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Relational Database Architecture

Purpose: Isolating multi-module tables cleanly inside PostgreSQL schemas (`hr`, `payroll`, `audit`).

What Topic Is Next → Next Topic 1.3: SQL Data Types

1.3 SQL Data Types [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `BIGINT` • `INTEGER` • `NUMERIC(p • S)` • Production Pitfall Guard (SQL Data Types) • Hands-On Verification (SQL Data Types) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for SQL Data Types

Purpose: Using `NUMERIC(12,2)` for exact money amounts and `TIMESTAMPTZ` for timezone-safe UTC timestamps.

What Topic Is Next → Next Topic 1.4: SQL Command Categories (DDL, DML, DQL, DCL and TCL)

1.4 SQL Command Categories (DDL, DML, DQL, DCL and TCL) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: DDL (`CREATE/ALTER/DROP/TRUNCATE`) • DML (`INSERT/UPDATE/DELETE`) • DQL (`SELECT`) • DCL (`GRANT/REVOKE`) • Production Pitfall Guard (SQL Command Categories (DDL, DML, DQL, DCL and TCL)) • Hands-On Verification (SQL Command Categories (DDL, DML, DQL, DCL and TCL)) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for SQL Command Categories (DDL, DML, DQL, DCL and TCL)

Purpose: Separating Flyway/Liquibase schema migrations (DDL) from runtime repository operations (DML/DQL/TCL) and least-privilege DB roles (DCL).

What Topic Is Next → Next Topic 1.5: Database and Table Creation (CREATE, ALTER, DROP)

1.5 Database and Table Creation (CREATE, ALTER, DROP) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `CREATE TABLE` • `GENERATED ALWAYS AS IDENTITY` • `ALTER TABLE ADD/ALTER/DROP COLUMN` • `DROP TABLE IF EXISTS` • Production Pitfall Guard (Database and Table Creation (CREATE, ALTER, DROP)) • Hands-On Verification (Database and Table Creation (CREATE, ALTER, DROP)) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Database

Purpose: Writing repeatable, forward-compatible SQL migration scripts for production schema evolution.

What Topic Is Next → Next Topic 1.6: Data Manipulation (INSERT, UPDATE, DELETE)

1.6 Data Manipulation (INSERT, UPDATE, DELETE) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-row `INSERT ... RETURNING` • Targeted `UPDATE ... WHERE` • Safe `DELETE ... WHERE` • `ON CONFLICT` upserts • Production Pitfall Guard (Data Manipulation (INSERT, UPDATE, DELETE)) • Hands-On Verification (Data Manipulation (INSERT, UPDATE, DELETE)) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Data Manipulation (INSERT, UPDATE, DELETE)

Purpose: Fetching generated IDs in one round-trip with `INSERT ... RETURNING id` and preventing full-table accidental updates.

What Topic Is Next → Next Topic 1.7: Data Retrieval (SELECT and DISTINCT)

1.7 Data Retrieval (SELECT and DISTINCT) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Explicit column projection (avoiding `SELECT *`) • Column aliases • Computed expressions • `COALESCE` • Production Pitfall Guard (Data Retrieval (SELECT and DISTINCT)) • Hands-On Verification (Data Retrieval (SELECT and DISTINCT)) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Data Retrieval

(SELECT and DISTINCT)

Purpose: Fetching only required DTO columns over the network to reduce database I/O and memory consumption.

What Topic Is Next → Next Topic 1.8: Filtering Rows (WHERE, AND, OR, NOT)

1.8 Filtering Rows (WHERE, AND, OR, NOT) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Predicate evaluation • Operator precedence (` • ` binds tighter than `OR`!) • Explicit parentheses • Production Pitfall Guard (Filtering Rows (WHERE, AND, OR, NOT)) • Hands-On Verification (Filtering Rows (WHERE, AND, OR, NOT)) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Filtering Rows (WHERE, AND, OR, NOT)

Purpose: Building accurate search filters in HR and e-commerce product catalogs.

What Topic Is Next → Next Topic 1.9: SQL Operators (IN, BETWEEN, LIKE, ILIKE, IS NULL)

1.9 SQL Operators (IN, BETWEEN, LIKE, ILIKE, IS NULL) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Set membership `IN` • Inclusive range `BETWEEN` • Wildcard `LIKE` (`%` • `_`) / PostgreSQL `ILIKE` • Production Pitfall Guard (SQL Operators (IN, BETWEEN, LIKE, ILIKE, IS NULL)) • Hands-On Verification (SQL Operators (IN, BETWEEN, LIKE, ILIKE, IS NULL)) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for SQL Operators (IN, BETWEEN, LIKE, ILIKE, IS NULL)

Purpose: Implementing prefix autocomplete (`LIKE 'rav%'`) and handling nullable `manager_id` / `terminated_at` columns.

What Topic Is Next → Next Topic 1.10: Sorting Results (ORDER BY)

1.10 Sorting Results (ORDER BY) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-column `ASC`/`DESC` sorting • `NULLS LAST` / `NULLS FIRST` • Deterministic tie-breaker columns (`ORDER BY salary DESC • Id ASC`) • Production Pitfall Guard (Sorting Results (ORDER BY)) • Hands-On Verification (Sorting Results (ORDER BY)) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Sorting Results (ORDER BY)

Purpose: Preventing duplicate or skipped records across paginated API pages by always including a unique tie-breaker column.

What Topic Is Next → Module 1 Capstone (Week 9 Lab Build: PostgreSQL Enterprise HR Schema & 40-Query Pack) → Next Topic 2.1: Aggregation Functions (COUNT, SUM, AVG, MIN, MAX)

Module 1 Hands-On Capstone Project (100 Marks): Week 9 Lab Build: PostgreSQL Enterprise HR Schema & 40-Query Pack — Author a complete PostgreSQL DDL schema (`departments`, `employees`, `projects`, `employee_projects`) with PK/FK/UNIQUE/CHECK constraints and 40 analytical queries covering INNER/LEFT/SELF j...

MODULE 2 OF 5 • STAGE 1 • 10 LIVE + LAB HOURS • 9 TOPICS (72 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: Multi-Table SQL Joins, Subqueries, Views & Aggregations

Build complete fluency in relational database fundamentals and SQL using PostgreSQL (with MySQL syntax comparisons) — DDL, DML, DQL, constraints, 1:1 / 1:N / M:N relationships, INNER/OUTER/SELF joins, scalar & correlated subqueries, aggregations, and set operations.

Objectives: ✓ Design relational tables in PostgreSQL with accurate data types and PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, and NOT NULL constraints. • ✓ Write safe DML (`INSERT`, `UPDATE`, `DELETE`) and expressive `SELECT` queries with filtering, pattern matching, and sorting. • ✓ Aggregate business metrics using `GROUP BY` and `HAVING` while respecting SQL logical execution order (`FROM` -> `WHERE` -> `GROUP BY` -> `HAVING` -> `SELECT` -> `ORDER BY`). • ✓ Model 1:1, 1:N, and M:N join tables and write INNER, LEFT, RIGHT, FULL OUTER, and SELF joins accurately. • ✓ Solve multi-step reporting questions using single-row, multi-row, correlated subqueries, and `UNION` / `INTERSECT` / `EXCEPT`.

2.1 Aggregation Functions (COUNT, SUM, AVG, MIN, MAX) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `COUNT(*)` vs `COUNT(column)` (ignores NULLs) vs `COUNT(DISTINCT col)` • `SUM` • `ROUND(AVG(...) • 2)` • Production Pitfall Guard (Aggregation Functions (COUNT, SUM, AVG, MIN, MAX)) • Hands-On Verification (Aggregation Functions (COUNT, SUM, AVG, MIN, MAX)) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Aggregation Functions (COUNT, SUM, AVG, MIN, MAX)

Purpose: Generating dashboard KPI cards (total payroll, headcount, unassigned employees, average salary) in a single SQL query.

What Topic Is Next → Next Topic 2.2: Grouping Data (GROUP BY and HAVING)

2.2 Grouping Data (GROUP BY and HAVING) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Grouping keys • Non-aggregated column rule • Pre-aggregation `WHERE` filtering vs post-aggregation `HAVING` filtering • Production Pitfall Guard (Grouping Data (GROUP BY and HAVING)) • Hands-On Verification (Grouping Data (GROUP BY and HAVING)) • Grouping Data (GROUP BY and HAVING) — Interview Readiness • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Grouping Data (GROUP BY and HAVING)

Purpose: Finding departments with more than 5 active employees or average salary above ₹80,000.

What Topic Is Next → Next Topic 2.3: Integrity Constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, NOT NULL)

2.3 Integrity Constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, NOT NULL) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Declarative data integrity • `ON DELETE RESTRICT / CASCADE / SET NULL` • `CHECK (salary > 0)` • Composite unique constraints • Production Pitfall Guard (Integrity Constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, NOT NULL)) • Hands-On Verification (Integrity Constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, NOT NULL)) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Integrity Constraints (PRIMARY KEY, FOREIGN KEY, UNI...

Purpose: Guaranteeing zero orphan foreign keys and zero negative prices/salaries even if a bug slips into application code.

What Topic Is Next → Next Topic 2.4: Entity Relationships (1:1, 1:N and M:N)

2.4 Entity Relationships (1:1, 1:N and M:N) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: One-to-One (`employee_profiles` unique FK) • One-to-Many (`departments` -> `employees`) • Many-to-Many (`employees` <-> `projects` via composite-PK join table) • Production Pitfall Guard (Entity Relationships (1:1, 1:N and M:N)) • Hands-On Verification (Entity Relationships (1:1, 1:N and M:N)) • Entity Relationships (1:1, 1:N and M:N) — Interview Readiness • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Entity Relationships (1:1, 1:N and M:N)

Purpose: Designing clean join tables (`employee_projects` with `assigned_at`, `role`) for enterprise domain models.

What Topic Is Next → Next Topic 2.5: SQL Joins (INNER, LEFT, RIGHT and FULL OUTER JOIN)

2.5 SQL Joins (INNER, LEFT, RIGHT and FULL OUTER JOIN) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: ANSI `JOIN ... ON` syntax • Preserving unmatched left rows with `LEFT JOIN` • Finding anti-joins (`LEFT JOIN ... WHERE right.id IS NULL`) • `FULL OUTER JOIN` • Production Pitfall Guard (SQL Joins (INNER, LEFT, RIGHT and FULL OUTER JOIN)) • Hands-On Verification (SQL Joins (INNER, LEFT, RIGHT and FULL OUTER JOIN)) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for SQL Joins (INNER, LEFT, RIGHT and FULL OUTER JOIN)

Purpose: Listing all departments including newly created departments with 0 employees (`LEFT JOIN` + `COUNT(e.id)`).

What Topic Is Next → Next Topic 2.6: Self Joins

2.6 Self Joins [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Joining a table to itself using mandatory table aliases (`emp` vs `mgr`) to query hierarchical parent-child rows • Production Pitfall Guard (Self Joins) • Hands-On Verification (Self Joins) • Self Joins — Interview Readiness • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Self Joins • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Listing every employee alongside their reporting manager's name (including CEO via `LEFT JOIN`).

What Topic Is Next → Next Topic 2.7: Subqueries (Scalar and Multi-Row)

2.7 Subqueries (Scalar and Multi-Row) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Scalar subqueries (`salary > (SELECT AVG(salary)...)`) • `IN` subqueries • Derived inline tables in `FROM` • `NOT IN` NULL trap! • Production Pitfall Guard (Subqueries (Scalar and Multi-Row)) • Hands-On Verification (Subqueries (Scalar and Multi-Row)) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Subqueries (Scalar and Multi-Row)

Purpose: Comparing individual performance or pricing against company-wide or category benchmarks.

What Topic Is Next → Next Topic 2.8: Correlated Subqueries and EXISTS

2.8 Correlated Subqueries and EXISTS [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Outer-row-dependent subqueries • `EXISTS` / `NOT EXISTS` short-circuit semi-joins • Comparing against per-department averages • Production Pitfall Guard (Correlated Subqueries and EXISTS) • Hands-On Verification (Correlated Subqueries and EXISTS) • Correlated Subqueries and EXISTS — Interview Readiness • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Correlated Subqueries

Purpose: Finding employees earning above *their own department's* average salary or customers who placed at least one order this month.

What Topic Is Next → Next Topic 2.9: Set Operations (UNION, UNION ALL, INTERSECT, EXCEPT)

2.9 Set Operations (UNION, UNION ALL, INTERSECT, EXCEPT) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Column-count & type compatibility • Deduplicating `UNION` vs fast `UNION ALL` • Common rows `INTERSECT` • Set difference `EXCEPT` • Production Pitfall Guard (Set Operations (UNION, UNION ALL, INTERSECT, EXCEPT)) • Hands-On Verification (Set Operations (UNION, UNION ALL, INTERSECT, EXCEPT)) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Set Operations (UNION, UNION ALL, INTERSECT, EXCEPT)

Purpose: Combining active and archived ledger feeds (`UNION ALL`) or auditing discrepancies between two tables (`EXCEPT`).

What Topic Is Next → Module 2 Capstone (Week 2 Practical Build: Multi-Table SQL Joins, Subqueries, Views & Aggregations) → Next Topic 3.1: Database Normalization (1NF, 2NF, 3NF and BCNF)

Module 2 Hands-On Capstone Project (100 Marks): Week 2 Practical Build: Multi-Table SQL Joins, Subqueries, Views & Aggregations — Implement and verify a clean, runnable SQLD repository covering Build complete fluency in relational database fundamentals and SQL using PostgreSQL (with MySQL syntax comparisons) — DDL, DML...

MODULE 3 OF 5 • STAGE 1 • 10 LIVE + LAB HOURS • 9 TOPICS (72 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: ER Modeling, 3NF Normalization & B-Tree Index Tuning

Design normalized, high-performance relational schemas in PostgreSQL (with MySQL comparisons) — 1NF–BCNF normalization, composite B-Tree indexing, EXPLAIN ANALYZE query plans, views & materialized views, functions/procedures/triggers, ACID isolation levels, locking/deadlocks, keyset pagination, window functions, and recursive CTEs.

Objectives: ✓ Normalize raw business spreadsheets from unnormalized form through 1NF, 2NF, 3NF, and BCNF while knowing when controlled denormalization makes sense. • ✓ Create single-column, composite, partial, and unique indexes and verify index usage via `EXPLAIN ANALYZE`. • ✓ Manage Views, Materialized Views, SQL Functions, Stored Procedures, and Audit Triggers responsibly. • ✓ Reason accurately about ACID properties, Transaction Isolation Levels, row locks (`SELECT ... FOR UPDATE`), and database deadlocks. • ✓ Write enterprise reporting queries using Keyset Pagination, Window Functions (`ROW_NUMBER`, `RANK`, `DENSE_RANK`, `LAG`, `LEAD`), and Recursive CTEs.

3.1 Database Normalization (1NF, 2NF, 3NF and BCNF) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Atomic attributes (1NF) • Eliminating partial dependencies on composite keys (2NF) • Eliminating transitive dependencies (3NF) • Boyce-Codd Normal Form (BCNF) • Production Pitfall Guard (Database Normalization (1NF, 2NF, 3NF and BCNF)) • Hands-On Verification (Database Normalization (1NF, 2NF, 3NF

and BCNF)) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Database Normalization (1NF, 2NF, 3NF and BCNF)

Purpose: Eliminating update, insertion, and deletion anomalies when designing order, inventory, and HR schemas.

What Topic Is Next → Next Topic 3.2: Strategic Denormalization

3.2 Strategic Denormalization [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Read-heavy vs write-heavy trade-offs • Snapshotting historical facts (`order_items.unit_price_at_purchase`) • Summary caches • Production Pitfall Guard (Strategic Denormalization) • Hands-On Verification (Strategic Denormalization) • Strategic Denormalization — Interview Readiness • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Strategic Denormalization

Purpose: Storing `unit_price` snapshot on `order_items` so future catalog price edits never alter historical invoices!

What Topic Is Next → Next Topic 3.3: Database Indexes (Single-Column, Composite and Unique)

3.3 Database Indexes (Single-Column, Composite and Unique) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: B-Tree structure (`O(log n)`) • Composite index leftmost-prefix rule `(department_id • Salary)` • Unique & partial indexes • Production Pitfall Guard (Database Indexes (Single-Column, Composite and Unique)) • Hands-On Verification (Database Indexes (Single-Column, Composite and Unique)) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Database Indexes (Single-Column, Composite and Uniqu...

Purpose: Accelerating foreign-key joins (PostgreSQL does NOT auto-index FK columns!) and multi-column search filters.

What Topic Is Next → Next Topic 3.4: Query Execution Plans (EXPLAIN and EXPLAIN ANALYZE)

3.4 Query Execution Plans (EXPLAIN and EXPLAIN ANALYZE) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Seq Scan` vs `Index Scan` vs `Index Only Scan` vs `Bitmap Heap Scan` • `Nested Loop` / `Hash Join` / `Merge Join` • Estimated vs actual rows • Production Pitfall Guard (Query Execution Plans (EXPLAIN and EXPLAIN ANALYZE)) • Hands-On Verification (Query Execution Plans (EXPLAIN and EXPLAIN ANALYZE)) • Query Execution Plans (EXPLAIN and EXPLAIN ANALYZE) — Interview Readiness • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Query Execution Plans (EXPLAIN and EXPLAIN ANALYZE)

Purpose: Proving with empirical millisecond evidence why an API query is slow before deploying an index migration.

What Topic Is Next → Next Topic 3.5: SQL Query Optimization

3.5 SQL Query Optimization [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: SARGable predicates (avoiding functions on indexed columns like `LOWER(email)` without functional index) • Avoiding `SELECT *` • Covering indexes • Production Pitfall Guard (SQL Query Optimization) • Hands-On Verification (SQL Query Optimization) • SQL Query Optimization — Interview Readiness • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for SQL Query Optimization

Purpose: Rewriting `WHERE EXTRACT(YEAR FROM created_at) = 2026` into a SARGable range `created_at >= '2026-01-01' AND created_at < '2027-01-01'` so the B-Tree index is used!

What Topic Is Next → Next Topic 3.6: Database Views

3.6 Database Views [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `CREATE OR REPLACE VIEW` • Security column masking (hiding salary/PII) • Simplifying complex reusable joins • Production Pitfall Guard (Database Views) • Hands-On Verification (Database Views) • Database Views — Interview Readiness • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Database Views

Purpose: Exposing a clean `v_public_employee_directory` view without sensitive payroll columns to reporting consumers.

What Topic Is Next → Next Topic 3.7: Materialized Views

3.7 Materialized Views [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `CREATE MATERIALIZED VIEW` • Persisted query results on disk • Indexing materialized views • `REFRESH MATERIALIZED VIEW CONCURRENTLY` • Production Pitfall Guard (Materialized Views) • Hands-On Verification (Materialized Views) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Materialized Views

Purpose: Precomputing heavy analytical dashboards refreshed every 15 minutes instead of scanning millions of rows on every page load.

What Topic Is Next → Next Topic 3.8: Stored Procedures

3.8 Stored Procedures [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: PostgreSQL `CREATE PROCEDURE` (`plpgsql`) • `CALL` • Transactional control inside procedures • When logic belongs in Java Service vs DB procedure • Production Pitfall Guard (Stored Procedures) • Hands-On Verification (Stored Procedures) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Stored Procedures

Purpose: Executing bulk archival or data-cleanup DBA routines close to the data engine.

What Topic Is Next → Next Topic 3.9: User-Defined SQL Functions

3.9 User-Defined SQL Functions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Scalar • Table-returning functions (`CREATE FUNCTION ... RETURNS`) • SQL vs PL/pgSQL functions • `IMMUTABLE` / `STABLE` / `VOLATILE` markers • Production Pitfall Guard (User-Defined SQL Functions) • Hands-On Verification (User-Defined SQL Functions) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for User-Defined SQL Functions

Purpose: Encapsulating reusable fiscal-year or net-compensation calculations inside reporting SQL queries.

What Topic Is Next → Module 3 Capstone (Week 10 Lab Build: 3NF Schema Normalization, Index Tuning & Window Analytics) → Next Topic 4.1: Database Triggers

Module 3 Hands-On Capstone Project (100 Marks): Week 10 Lab Build: 3NF Schema Normalization, Index Tuning & Window Analytics — Normalize an unnormalized enterprise dataset into 3NF/BCNF, benchmark composite B-Tree indexes on 100k rows using `EXPLAIN ANALYZE`, and build Top-N per department `DENSE_RANK` + `WITH RECUR...

MODULE 4 OF 5 • STAGE 2 • 10 LIVE + LAB HOURS • 9 TOPICS (72 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: ACID Transactions, Isolation Levels, Window Functions & CTEs

Design normalized, high-performance relational schemas in PostgreSQL (with MySQL comparisons) — 1NF–BCNF normalization, composite B-Tree indexing, EXPLAIN ANALYZE query plans, views & materialized views, functions/procedures/triggers, ACID isolation levels, locking/deadlocks, keyset pagination, window functions, and recursive CTEs.

Objectives: ✓ Normalize raw business spreadsheets from unnormalized form through 1NF, 2NF, 3NF, and BCNF while knowing when controlled denormalization makes sense. • ✓ Create single-column, composite, partial, and unique indexes and verify index usage via `EXPLAIN ANALYZE`. • ✓ Manage Views, Materialized Views, SQL Functions, Stored Procedures, and Audit Triggers responsibly. • ✓ Reason accurately about ACID properties, Transaction Isolation Levels, row locks (`SELECT ... FOR UPDATE`), and database deadlocks. • ✓ Write enterprise reporting queries using Keyset Pagination, Window Functions (`ROW_NUMBER`, `RANK`, `DENSE_RANK`, `LAG`, `LEAD`), and Recursive CTEs.

4.1 Database Triggers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `BEFORE` vs `AFTER` row/statement triggers • `NEW` • `OLD` record variables • Automated `updated_at` stamping • Production Pitfall Guard (Database Triggers) • Hands-On Verification (Database Triggers) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Database Triggers

Purpose: Recording tamper-evident salary-change audit rows automatically whenever `hr.employees.salary` changes.

What Topic Is Next → Next Topic 4.2: Transactions (BEGIN, COMMIT, ROLLBACK and SAVEPOINT)

4.2 Transactions (BEGIN, COMMIT, ROLLBACK and SAVEPOINT) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: All-or-nothing atomic work units • `BEGIN` • `COMMIT` • `ROLLBACK` • Production Pitfall Guard (Transactions (BEGIN, COMMIT, ROLLBACK and SAVEPOINT)) • Hands-On Verification (Transactions (BEGIN, COMMIT, ROLLBACK and SAVEPOINT)) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Transactions (BEGIN, COMMIT, ROLLBACK and SAVEPOINT)

Purpose: Ensuring wallet debit + ledger credit + order status update either all commit together or all roll back cleanly on failure.

What Topic Is Next → Next Topic 4.3: ACID Properties

4.3 ACID Properties [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Atomicity (WAL undo/abort) • Consistency (constraints/invariants) • Isolation (MVCC concurrency) • Durability (WAL fsync crash recovery) • Production Pitfall Guard (ACID Properties) • Hands-On Verification (ACID Properties) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for ACID Properties

Purpose: Explaining in backend interviews *how* PostgreSQL guarantees zero corrupted balances across power loss and concurrent traffic.

What Topic Is Next → Next Topic 4.4: Transaction Isolation Levels

4.4 Transaction Isolation Levels [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Dirty Read • Non-Repeatable Read • Phantom Read • Write Skew • Production Pitfall Guard (Transaction Isolation Levels) • Hands-On Verification (Transaction Isolation Levels) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Transaction Isolation Levels

Purpose: Preventing concurrent double-spend or lost updates by choosing Repeatable Read / Serializable or explicit row locking.

What Topic Is Next → Next Topic 4.5: Database Locking (Shared and Exclusive Locks)

4.5 Database Locking (Shared and Exclusive Locks) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Row-level `FOR SHARE` vs `FOR UPDATE` • `FOR UPDATE SKIP LOCKED` (job queues) • Pessimistic vs optimistic concurrency control • Production Pitfall Guard (Database Locking (Shared and Exclusive Locks)) • Hands-On Verification (Database Locking (Shared and Exclusive Locks)) • Database Locking (Shared and Exclusive Locks) — Interview Readiness • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Database Locking (Shared and Exclusive Locks)

Purpose: Locking an inventory or wallet row with `SELECT ... FOR UPDATE` during checkout, or polling outbox events via `FOR UPDATE SKIP LOCKED`.

What Topic Is Next → Next Topic 4.6: Database Deadlocks

4.6 Database Deadlocks [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Cross-row lock cycles between two transactions • PostgreSQL deadlock detector (`deadlock_timeout`) • Automatic victim abort (`SQLSTATE 40P01`) • Consistent primary-key update ordering • Production Pitfall Guard (Database Deadlocks) • Hands-On Verification (Database Deadlocks) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Database Deadlocks

Purpose: Sorting item IDs ascending before batch `UPDATE` / `FOR UPDATE` statements so concurrent checkouts never deadlock!

What Topic Is Next → Next Topic 4.7: Efficient SQL Pagination (LIMIT/OFFSET vs Keyset Seek)

4.7 Efficient SQL Pagination (LIMIT/OFFSET vs Keyset Seek) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why `OFFSET 100000 LIMIT 20` still scans & discards 100 • 000 rows • Vs `O(log n)` Keyset/Cursor pagination (`WHERE (created_at • Id) < (? • Production Pitfall Guard (Efficient SQL Pagination (LIMIT/OFFSET vs Keyset Seek)) • Hands-On Verification (Efficient SQL Pagination (LIMIT/OFFSET vs Keyset Seek)) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Efficient SQL Pagination (LIMIT/OFFSET vs Keyset See...

Purpose: Keeping infinite-scroll feeds and large order-export APIs sub-millisecond at page 5,000.

What Topic Is Next → Next Topic 4.8: Window Functions (ROW_NUMBER, RANK, DENSE_RANK, LEAD, LAG)

4.8 Window Functions (ROW_NUMBER, RANK, DENSE_RANK, LEAD, LAG) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `OVER (PARTITION BY ... ORDER BY ...)` • Top-N per group (`DENSE_RANK`) • Running totals (`SUM(...) OVER`) • Month-over-month deltas (`LAG`/`LEAD`) • Production Pitfall Guard (Window Functions (ROW_NUMBER, RANK, DENSE_RANK, LEAD, LAG)) • Hands-On Verification (Window Functions (ROW_NUMBER, RANK, DENSE_RANK, LEAD, LAG)) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Window Functions (ROW_NUMBER, RANK, DENSE_RANK, LEAD...

Purpose: Finding the Top 3 highest-paid employees *per department* (the #1 SQL interview problem!) and computing running account balances.

What Topic Is Next → Next Topic 4.9: Common Table Expressions (CTEs and Recursive CTEs)

4.9 Common Table Expressions (CTEs and Recursive CTEs) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `WITH` readable modular queries • Multi-CTE pipelines • `WITH RECURSIVE` anchor + recursive union for org-chart & category hierarchies • Production Pitfall Guard (Common Table Expressions (CTEs and Recursive CTEs)) • Hands-On Verification (Common Table Expressions (CTEs and Recursive CTEs)) • Common Table Expressions (CTEs and Recursive CTEs) — Interview Readiness • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Common Table Expressions (CTEs and Recursive CTEs)

Purpose: Traversing an arbitrary-depth reporting hierarchy (`manager_id`) or multi-level e-commerce category tree in a single SQL query.

What Topic Is Next → Module 4 Capstone (Week 4 Practical Build: ACID Transactions, Isolation Levels, Window Functions & CTEs) → Next Topic 5.1: PostgreSQL 16 vs MySQL 8 Engine Architecture

Module 4 Hands-On Capstone Project (100 Marks): Week 4 Practical Build: ACID Transactions, Isolation Levels, Window Functions & CTEs — Implement and verify a clean, runnable SQLD repository covering Design normalized, high-performance relational schemas in PostgreSQL (with MySQL comparisons) — 1NF–BCNF normalization, compos...

MODULE 5 OF 5 • STAGE 2 • 10 LIVE + LAB HOURS • 6 TOPICS (48 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: PostgreSQL vs MySQL Dialect Parity, Stored Routines & Top 50 SQL Interview Queries

Compare PostgreSQL 16 and MySQL 8 syntax/engine nuances (InnoDB vs Heap/MVCC, JSONB, UPSERT), write functions/triggers safely, and master Top 50 campus & product-company SQL interview query patterns.

Objectives: ✓ Navigate practical syntax differences between PostgreSQL (`SERIAL`/`IDENTITY`, `ILIKE`, `ON CONFLICT`) and MySQL 8 (`AUTO_INCREMENT`, `ON DUPLICATE KEY UPDATE`). • ✓ Query semi-structured JSON/JSONB columns with GIN indexes when appropriate without abusing EAV anti-patterns. • ✓ Solve classic SQL interview problems: Nth highest salary per department, consecutive login streaks, retention cohorts, and YoY revenue deltas.

5.1 PostgreSQL 16 vs MySQL 8 Engine Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: PostgreSQL MVCC heap + WAL vs MySQL InnoDB clustered primary-key B+Tree • `IDENTITY` vs `AUTO_INCREMENT` • Case sensitivity • Production Pitfall Guard (PostgreSQL 16 vs MySQL 8 Engine Architecture) • Hands-On Verification (PostgreSQL 16 vs MySQL 8 Engine Architecture) • PostgreSQL 16 vs MySQL 8 Engine Architecture — Interview Readiness • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for PostgreSQL 16 vs MySQL 8 Engine Architecture

Purpose: Writing portable SQL and understanding why UUIDv4 random primary keys fragment MySQL InnoDB clustered pages.

What Topic Is Next → Next Topic 5.2: JSON and JSONB Semi-Structured Columns & GIN Indexes

5.2 JSON and JSONB Semi-Structured Columns & GIN Indexes [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: PostgreSQL `JSONB` vs `JSON` • `->` • `->>` operators • `@>` containment with GIN index • Production Pitfall Guard (JSON and JSONB Semi-Structured Columns & GIN Indexes) • Hands-On Verification (JSON and JSONB Semi-Structured Columns & GIN Indexes) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for JSON

Purpose: Storing dynamic product specification attributes in `JSONB` while keeping core relational columns normalized.

What Topic Is Next → Next Topic 5.3: Stored Procedures, Functions and Triggers (When to Use & Avoid)

5.3 Stored Procedures, Functions and Triggers (When to Use & Avoid) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: PL/pgSQL functions • Audit-trail triggers • Why heavy business orchestration belongs in versioned application services rather than hidden triggers • Production Pitfall Guard (Stored Procedures, Functions and Triggers (When to Use & Avoid)) • Hands-On Verification (Stored Procedures, Functions and Triggers (When to Use & Avoid)) • Stored Procedures, Functions and Triggers (When to Use & Avoid) — Interview Readiness • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for Stored Procedures

Purpose: Maintaining `updated_at` / immutable audit tables cleanly with lightweight database triggers.

What Topic Is Next → Next Topic 5.4: SQL Interview Pattern: Top-N & Nth Highest per Group (`DENSE_RANK`)

5.4 SQL Interview Pattern: Top-N & Nth Highest per Group (`DENSE_RANK`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Finding 2nd/3rd highest salary per department handling ties accurately via `DENSE_RANK() OVER (PARTITION BY dept_id ORDER BY salary DESC)` • Production Pitfall Guard (SQL Interview Pattern: Top-N & Nth Highest per Group (`DENSE_RANK`)) • Hands-On Verification (SQL Interview Pattern: Top-N & Nth Highest per Group (`DENSE_RANK`)) • SQL Interview Pattern: Top-N & Nth Highest per Group (`DENSE_RANK`) — Interview Readiness • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for SQL Interview Pattern • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics,

Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Cracking the #1 most frequently asked SQL interview & HackerRank OA query cleanly.

What Topic Is Next → Next Topic 5.5: SQL Interview Pattern: Consecutive Streaks (Gaps & Islands)

5.5 SQL Interview Pattern: Consecutive Streaks (Gaps & Islands) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Subtracting `ROW_NUMBER()` from date/sequence to group consecutive active days (`login_date - INTERVAL '1 day' * rn`) • Production Pitfall Guard (SQL Interview Pattern: Consecutive Streaks (Gaps & Islands)) • Hands-On Verification (SQL Interview Pattern: Consecutive Streaks (Gaps & Islands)) • SQL Interview Pattern: Consecutive Streaks (Gaps & Islands) — Interview Readiness • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for SQL Interview Pattern • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Computing 7-day active user streaks and attendance rewards in product analytics.

What Topic Is Next → Next Topic 5.6: SQL Interview Pattern: Month-over-Month Growth & Retention Cohorts

5.6 SQL Interview Pattern: Month-over-Month Growth & Retention Cohorts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `DATE_TRUNC('month' • Created_at)` • `LAG(revenue) OVER (ORDER BY month)` MoM % growth • `NULLIF` zero-division guard • Production Pitfall Guard (SQL Interview Pattern: Month-over-Month Growth & Retention Cohorts) • Hands-On Verification (SQL Interview Pattern: Month-over-Month Growth & Retention Cohorts) • Idiomatic SQLD Implementation: clean modular architecture, strict contracts & non-blocking patterns • Complete SQL, PostgreSQL & MySQL Database Engineering Production Runtime Execution & Memory Lifecycle for SQL Interview Pattern

Purpose: Generating executive MoM revenue growth and cohort retention tables directly in SQL.

What Topic Is Next → Module 5 Capstone → End-of-Course Live Client Project: Complete SQL, PostgreSQL & MySQL Database Engineering — Enterprise Client Production Platform

Module 5 Hands-On Capstone Project (100 Marks): Capstone: 3NF Enterprise E-Commerce & HR Analytics PostgreSQL Database — Design a normalized 12-table 3NF PostgreSQL database covering Users, Products, Inventory, Orders, Payments, and Departments, tuned with composite B-Tree indexes and accompanied by 30 analyti...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

End-of-Course Live Client Production Project + Official Internship Confirmation Letter

Project: Complete SQL, PostgreSQL & MySQL Database Engineering — Enterprise Client Production Platform • **Confirmed Role:** Complete SQL, PostgreSQL & MySQL Database Engineering Engineering Project Intern

Domain & Production Infra: Live Enterprise Relational Databases & SQL Engineering Client Engagement (SQL (DDL/DML/DQL), PostgreSQL 16, MySQL 8, Joins & Subqueries, ER Modeling & 3NF • Git CI/CD + Automated Testing + Live HTTPS Cloud Production)

- Translated live client functional and non-functional requirements into modular SQL (DDL/DML/DQL), PostgreSQL 16, MySQL 8, Joins & Subqueries, ER Modeling & 3NF production architecture.
- Implemented core application modules, input validation, security hardening, and optimized data/state workflows.
- Completed peer Git pull-request code reviews, automated verification suites, and edge-case debugging sprints.
- Deployed the verified build to live client production with monitoring, documentation, and mentor sign-off.

Target Job Roles: Complete SQL, PostgreSQL & MySQL Database Engineering Developer, Associate Relational Databases & SQL Engineering Engineer, Software Development Engineer (SDE-1)

Realistic Annual CTC Scope: **Fresher / Entry-Level (0–1 Yr):** ₹4.5 – ₹8.5 LPA • $55K–$85K Global **Mid-Level Product Engineer (1–3 Yrs):** ₹9.5 – ₹19.0 LPA • $85K–$130K Global **Senior / Multi-Stack Specialist (3–6+ Yrs):** ₹20.0 – ₹38.0+ LPA • $130K–$185K+ Global

MockAttempt Academy Pvt. Ltd. • Email: support@mockattempt.com • Phone/WhatsApp: +91 95400 07917 • Web: https://mockattempt.com/courses/sql-postgresql-mysql-database-training

Try 2 Free Topic-Wise Timed Topic Mock Tests & Spoken MockAttempt AI Interviews daily on every topic above before enrolling.