MySQL Practical Evaluation Test

⏱ Total Duration: 180 Minutes

🧠 Level: Expert / Hard

📦 Test Structure (4 Blocks):  
1. Advanced Schema Design with Constraints & Indexing  
2. Multi-Layer Querying & Analytical SQL  
3. Optimization, View Materialization, and Cost Reporting  
4. Advanced Procedures, Triggers, and Auditing Functions

# ✅ Block 1: Advanced Schema Design with Constraints & Indexing (30 Minutes)

📌 Objective:  
Design a normalized schema for a large-scale corporate system, with referential integrity, composite keys, indexing strategy, and temporal fields.

🛠 Tasks:  
Create a database: enterprise\_ops

departments  
- dept\_id INT PRIMARY KEY AUTO\_INCREMENT  
- dept\_name VARCHAR(150) UNIQUE  
- division VARCHAR(100)  
- created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP  
  
employees  
- emp\_id INT PRIMARY KEY AUTO\_INCREMENT  
- emp\_name VARCHAR(120)  
- email VARCHAR(150) UNIQUE  
- salary DECIMAL(12, 2)  
- bonus\_percent DECIMAL(5,2) DEFAULT 0.00  
- joining\_date DATE  
- active\_flag BOOLEAN DEFAULT TRUE  
- dept\_id INT, FK → departments  
- manager\_id INT, FK → employees(emp\_id)  
- last\_updated TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP  
  
projects  
- project\_id INT PRIMARY KEY AUTO\_INCREMENT  
- project\_name VARCHAR(120) UNIQUE  
- start\_date DATE  
- end\_date DATE  
- budget DECIMAL(15, 2)  
- status ENUM('Planning','Active','Closed')  
  
employee\_projects  
- emp\_id INT FK → employees  
- project\_id INT FK → projects  
- hours\_allocated INT  
- assigned\_on DATE DEFAULT CURRENT\_DATE  
- Composite PK: (emp\_id, project\_id)  
  
Indexing Requirements:  
- Composite index on (dept\_id, joining\_date)  
- Index on projects(status, start\_date)  
- Index on employee\_projects(hours\_allocated)  
- Use FULLTEXT on projects.project\_name

# ✅ Block 2: Multi-Layer Querying & Analytical SQL (45 Minutes)

📌 Objective:  
Write advanced and layered queries using CTEs, subqueries, correlated subqueries, window functions, and dynamic filters.

1. 1. Get the top 3 most expensive projects (by total employee cost: hours\_allocated \* salary) using a window function and partition by status.
2. 2. List employees who joined in the last 2 years, are not assigned to any Active or Planning project, and have bonus > avg department bonus.
3. 3. For each department, show: total employees, number of inactive employees, highest salary, and variance in salary using VARIANCE().
4. 4. List managers who have more than 5 direct reports and at least 2 reports working on more than 3 projects.
5. 5. Using CTEs, build a timeline report: Project ID, Name, Duration (in days), Allocated Hours, Employee Count (Active projects started in last 365 days).

# ✅ Block 3: Optimization, View Materialization, and Reporting (35 Minutes)

📌 Objective:  
Use views, materialization, and query analysis to build enterprise-level reports and optimize heavy queries.

1. 1. Create a view department\_performance\_summary with dept\_id, total\_salary, avg\_bonus, employee\_count, projects\_count (for departments with > 5 employees).
2. 2. Create a materialized summary table project\_summary with project\_id, project\_name, total\_hours, cost\_estimate, remaining\_budget, last\_synced. Write SQL to refresh daily.
3. 3. Pick your most expensive query from Block 2, run EXPLAIN ANALYZE, identify index usage and recommend changes.
4. 4. Create a pivot-style report using conditional aggregation: show count of employees in each bonus range (0–5%, 5–10%, >10%) for each department.

# ✅ Block 4: Advanced Procedures, Triggers, and Auditing Functions (70 Minutes)

📌 Objective:  
Implement robust business logic and automation using MySQL's stored routines and triggers.

1. 1. Stored Procedure: restructure\_department - Moves employees based on salary and logs changes into dept\_transfer\_log.
2. 2. Trigger: prevent\_overbudget\_assignment - Checks total cost before INSERT into employee\_projects and blocks if it exceeds project budget.
3. 3. Function: get\_employee\_earnings(emp\_id INT) - Calculates total earnings from project assignments including bonus multiplier.
4. 4. Audit Trigger: AFTER UPDATE on employees logs salary and bonus changes into employee\_audit\_log.

# 📄 Submission Guidelines

- A single .sql file with all DDLs, queries, procedures, triggers, and functions  
- Screenshots of outputs, validations, and EXPLAIN results  
- A .txt or .docx file listing optimization suggestions, assumptions, or logic used