**Use Case: Indexed CRUD & JOIN Queries in MySQL (<50ms)**

**Business Scenario: Employee-Project Tracker**

You’re building a lightweight employee-project tracking module in **MySQL**, and your primary goals are:

* Sub-50ms latency on **CRUD and JOIN operations**.
* Fast lookups, joins, and filters using **indexes**.
* Scalable schema using **normalized relations** and **foreign keys**.

**Database Schema in MySQL**

**1. Employees Table**

sql

CREATE TABLE Employees (

emp\_id INT PRIMARY KEY,

name VARCHAR(100),

department VARCHAR(50),

salary DECIMAL(10,2),

manager\_id INT

);

**2. Projects Table**

sql

CREATE TABLE Projects (

proj\_id INT PRIMARY KEY,

proj\_name VARCHAR(100),

emp\_id INT,

FOREIGN KEY (emp\_id) REFERENCES Employees(emp\_id)

);

**🚀 Indexing for Performance**

sql

-- Filter and JOIN optimization

CREATE INDEX idx\_department ON Employees(department);

CREATE INDEX idx\_manager ON Employees(manager\_id);

CREATE INDEX idx\_emp\_id\_fk ON Projects(emp\_id);

MySQL will use **BTREE indexes** (default for InnoDB) to optimize query access paths.

**CRUD + JOIN Queries under 50ms**

**1. INSERT (Create)**

sql

INSERT INTO Employees (emp\_id, name, department, salary, manager\_id)

VALUES (201, 'John Wick', 'Engineering', 95000.00, 101);

**Fast because**: Uses primary key (emp\_id), no lookups needed.

**2. SELECT with JOIN**

sql

SELECT e.name, p.proj\_name

FROM Employees e

JOIN Projects p ON e.emp\_id = p.emp\_id

WHERE e.department = 'Engineering';

**Fast because**:

* Employees.department is indexed.
* Projects.emp\_id is indexed.
* EXPLAIN will confirm use of index join.

**3. UPDATE by Primary Key**

sql

UPDATE Employees

SET department = 'Research'

WHERE emp\_id = 201;

**Fast because**:

* MySQL uses clustered index on PRIMARY KEY.

**4. DELETE by Primary Key**

sql

DELETE FROM Employees

WHERE emp\_id = 201;

**Fast because**:

* Uses direct row location from primary index.

**5. LEFT JOIN with NULL Filtering**

sql

SELECT e.name

FROM Employees e

LEFT JOIN Projects p ON e.emp\_id = p.emp\_id

WHERE p.proj\_id IS NULL;

**Use case**: Find unassigned employees.

**Validate with EXPLAIN in MySQL**

**Example:**

sql

EXPLAIN SELECT e.name, p.proj\_name

FROM Employees e

JOIN Projects p ON e.emp\_id = p.emp\_id

WHERE e.department = 'Engineering';

**Key Columns to Check:**

* type: Should be ref or const (not ALL).
* key: Index used.
* rows: Should be low (index access).
* Extra: Should say Using where; Using index if it’s a covering index.

**Performance Monitoring (MySQL-Specific)**

**1. Enable Profiling (MySQL ≤ 5.7):**

sql

SET profiling = 1;

-- Your query

SELECT ...;

SHOW PROFILES;

**2. Use performance\_schema (MySQL 8+):**

Enable event timing:

sql

UPDATE performance\_schema.setup\_consumers

SET ENABLED = 'YES' WHERE NAME LIKE '%statement%';

Or measure query response via:

sql

SHOW SESSION STATUS LIKE 'Last\_query\_cost';

**Best Practices for <50ms Queries in MySQL**

| **Area** | **Best Practice** |
| --- | --- |
| Indexing | Index foreign keys, filters (WHERE), and join keys |
| Joins | Use indexed columns; avoid joining large unfiltered datasets |
| Read | Avoid SELECT \*, use only required columns |
| Write | Use primary keys for UPDATE/DELETE to avoid full table scans |
| NULLs | Use IS NULL rather than = NULL |
| Explain | Always use EXPLAIN to verify index usage |
| Covering Index | Add composite indexes when filtering + selecting same columns |