

# CS731 Software Testing Project: Robust DAO Persistence Layer

[https://github.com/RutulPatel007/School\\_Manager](https://github.com/RutulPatel007/School_Manager)

November 26, 2025

## Project Goal

To deeply explore advanced testing techniques, specifically **Mutation Testing**, to ensure the quality and robustness of a Data Access Object (DAO) architecture implemented using Java and JDBC.

## Team Details

Role	Name
Team Member 1	Rutul
Team Member 2	Siddeshwar

## 1 Project Overview and Architecture

This project is a back-end Java application designed to manage core entities of a university or school using a clean, layered architecture.

Figure 1: Layered Architecture Overview (Model, DAO, Persistence)

### 1.1 Architectural Layers

1. **Model Layer ('.models')**: Contains Plain Old Java Objects (POJOs) like ‘Student’, ‘Teacher’, ‘Book’, etc., holding data state.
2. **Data Access Object (DAO) Layer ('.dao')**: Contains the core logic for communicating with the database. This layer abstracts all SQL queries and JDBC operations (e.g., ‘BookDAO’, ‘StudentDAO’) and extends a generic ‘BaseDAO’.
3. **Persistence Layer**: A MySQL database used to store all entity data.

### 1.2 Core Functionality

The application provides full persistence functionality across five major domains:

- **Students**: CRUD operations, CGPA updates, retrieving the topper.
- **Teachers**: CRUD operations, address updates, salary increment, retrieving the highest-paid teacher.
- **Courses**: CRUD operations, managing course details.

- **Books & Library:** CRUD operations for books, linking books to courses, managing library metadata.
- 

## 2 Deep Dive into Testing Strategy

The primary focus of this project is verifying the quality of the test cases using **Mutation Testing** (PIT).

### 2.1 What is Mutation Testing?

Mutation testing is a fault-based testing technique used to evaluate the effectiveness of a test suite. It works by introducing small, deliberate changes (called **mutants**) into the source code to simulate potential programming errors.

Figure 2: Mutation Testing Workflow: Generation, Execution, and Analysis

1. **Mutant Generation:** The tool (PIT) inserts mutations (e.g., changing `if (a > b)` to `if (a >= b)`, changing `return true` to `return false`).
2. **Mutant Execution:** The test suite is run against each mutated version of the code.
3. **Result:**
  - **Killed Mutant:** Test fails due to mutant → Strong suite.
  - **Surviving Mutant:** All tests pass despite change → Weak suite (gap in assertions).

### 2.2 Covered Test Types

Our test suite is a combination of both **Unit** and **Integration** tests:

Test Type	Focus Area	Verification Detail
<b>Integration</b>	All DAO Methods (CRUD)	Verifies interaction with live MySQL database: SQL syntax, Foreign Keys, and actual persistence.
<b>Unit Testing</b>	Mapping & Utilities	Isolated verification of logic like ‘mapResultSetToEntity’ and parameter binding.
<b>Edge Case</b>	All DAO Methods	Tests designed to kill mutants: null inputs, zero/negative IDs, empty result sets, and forced exceptions.

---

### 3 Tools and Technology Stack

Category	Tool/Technology	Role in Project
Language	Java (JDK 17+)	Core application development.
Build	Apache Maven	Project management and dependencies.
Database	MySQL (JDBC)	Persistent storage for school entities.
Testing	JUnit 5 Jupiter	Framework for Unit/Integration tests.
Mutation	PIT	Tool for assessing test suite quality.

---

### 4 Setup and Execution Guide

#### Code Repository Link

[INSERT GITHUB/GITLAB REPOSITORY LINK HERE]

#### 4.1 Environment Preparation

- Ensure Java (JDK) and Maven are installed.
- Start your MySQL Server.

#### 4.2 Database Setup

```
1 # 1. Create Database
2 mysql -u root -padmin -e "CREATE DATABASE IF NOT EXISTS school_db;"
```

```
3
4 # 2. Initialize Schema
5 mysql -u root -padmin school_db < sql/schem.sql
```

Listing 1: Database Initialization

#### 4.3 Running Tests and PIT Analysis

```
1 # 1. Run Standard Tests (Coverage)
2 mvn clean test
3
4 # 2. Run Mutation Testing (Strength)
5 mvn org.pitest:pitest-maven:mutationCoverage
6
7 # 3. View Results (Open in Browser)
8 open target/pit-reports/index.html
```

Listing 2: Test Execution Commands

#### 4.4 Final Quality Metrics

Metric	Score	Comment
Line Coverage	99%	High line coverage across all DAOs.
Mutation Coverage	82%	Strong mutation score demonstrating effective integration tests.
Test Strength	95%	95% of covered lines are asserted by robust tests.

## 5 Team Contribution Breakdown

The project required collaborative effort, with each team member taking ownership of specific components.

### Rutul's Individual Contribution (50%)

- **Core Entity Development:** Implemented DAOs and Models for **Library** and **Book** entities.
- **Foundational Development:** Wrote essential **JUnit Test Cases** for ‘LibraryDAO’ and ‘BookDAO’ (CRUD/List retrieval).
- **Advanced Testing (Refinement I):** Performed initial **PIT Mutation Analysis**. Designed targeted mutation-killing tests for ‘BookDAO’ and ‘LibraryDAO’.
- **Tooling and Infrastructure:** Led Maven (‘pom.xml’) configuration, dependency setup, and report generation.

### Siddeshwar's Individual Contribution (50%)

- **Core Entity Development:** Implemented DAOs and Models for **Student**, **Teacher**, and **Course**. Developed abstract **BaseDAO**.
- **Foundational Development:** Wrote essential **JUnit Test Cases** for ‘StudentDAO’, ‘TeacherDAO’, and ‘CourseDAO’.
- **Advanced Testing (Refinement II):** Conducted iterative **PIT Mutation Analysis** and authored the **Mutation Testing Strategy**. Designed tests to kill surviving mutants in Base/Student/Teacher DAOs.
- **Quality Assurance & Documentation:** Authored the **README**, performed final verification, and compiled the submission package.

---

## 6 AI Tool Usage Acknowledgment

We affirm that all critical test case design, mutation analysis, and structural modifications were executed **manually** by the team. We acknowledge the use of **Google Gemini** for:

- **Source Code Scaffolding:** Generation of standard boilerplate code and initial JDBC query structures.
- **Code Documentation:** Insertion of Javadoc-style annotations.
- **Documentation Formatting:** Assistance in structuring this project report.